United States Patent [19]

Watson et al.

[11] **3,859,635**

[45] **Jan. 7, 1975**

[54] PROGRAMMABLE CALCULATOR

[75] Inventors: **Robert E. Watson; Jack M. Walden; Charles W. Near,** all of Loveland, Colo.

[73] Assignee: **Hewlett-Packard Company,** Palo Alto, Calif.

[22] Filed: **June 15, 1971**

[21] Appl. No.: **153,437**

[52] **U.S. Cl.** **340/172.5**

[51] **Int. Cl.** **G06f 15/02**

[58] **Field of Search** 340/172.5

[56] **References Cited**

UNITED STATES PATENTS

3,364,395	1/1968	Donofrio	340/172.5 X
3,416,139	12/1968	Marx	340/172.5
3,436,734	4/1969	Pomerene	340/172.5
3,533,076	10/1970	Perkins et al.	340/172.5
3,533,076	10/1970	Perkins et al.	340/172.5
3,533,078	10/1970	Perkins et al.	340/172.5
3,602,894	8/1971	Igel et al.	340/172.5
3,623,012	11/1971	Lowry et al.	340/172.5
3,623,156	11/1971	Osborne	340/172.5
3,629,850	12/1971	Clark et al.	340/172.5
3,631,403	12/1971	Asbo et al.	340/172.5
3,631,536	12/1971	Mosman	340/172.5

OTHER PUBLICATIONS

Burroughs Series L2000 Electronic Billing Computer - Jan. 1969.

Burroughs TC500 Operation and Programming Manual - Feb. 1969, pages 5, 56, 57, 6, 61.

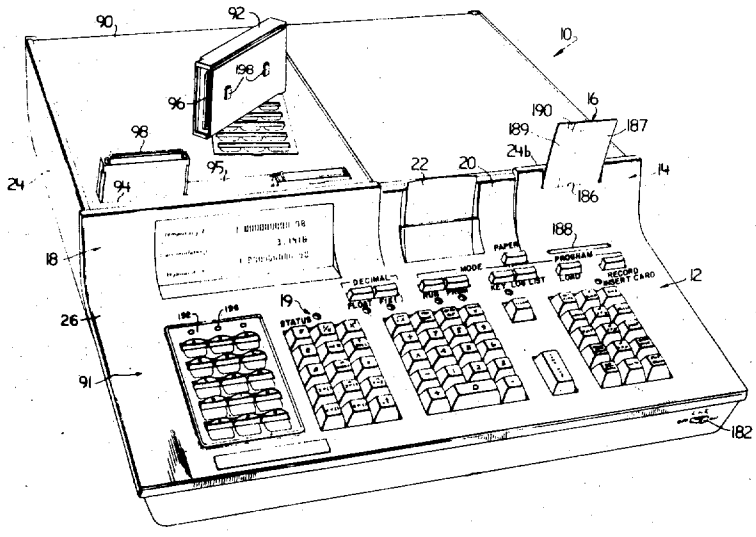
Primary Examiner—Gareth D. Shaw
Assistant Examiner—Mark Edward Nusbaum
Attorney, Agent, or Firm—Roland I. Griffin

[57] **ABSTRACT**

A modular read-write and read-only memory unit capable of employing both direct and indirect decimal and symbolic addressing, a central processing unit capable of performing both serial binary and parallel binary-coded-decimal direct and indirect memory register arithmetic, and an input-output control unit capable of bidirectionally transferring information between the central processing unit and a number of input and output units are controlled by a microprocessor included in the central processing unit. The input and output units include a keyboard input unit with a section capable of being defined by plug-in read-only memory modules and stored programs added by the user, a magnetic card reading and recording unit capable of bidirectionally transferring information between an external magnetic card and the calculator, and a solid state output display unit capable of displaying three lines of numeric information. An output printer unit capable of printing out every alphabetic and numeric character and many other symbols individually and in messages may also be included with the other input and output units.

The memory, central processing, input-output control, input, and output units are employed to provide an adaptable programmable calculator that may be operated manually by the user from the keyboard input unit or automatically by a program stored in the memory unit. This calculator may also be employed to load programs into the memory unit from the keyboard input unit, to separately transfer either data or programs bidirectionally between the memory unit and an external magnetic card, to code programs stored in the memory unit as being secure when they are transferred to an external magnetic card and thereby prevent users of the calculator from re-transferring them to an external magnetic card or obtaining any indication of the individual program steps once they are reloaded into the calculator, to edit programs stored in the memory unit and to print out keystroke logs, program lists, labels, and messages. The read-write memory available to the user may be expanded by the addition of program storage memory modules or by the alteration of the data storage memory control.

17 Claims, 361 Drawing Figures



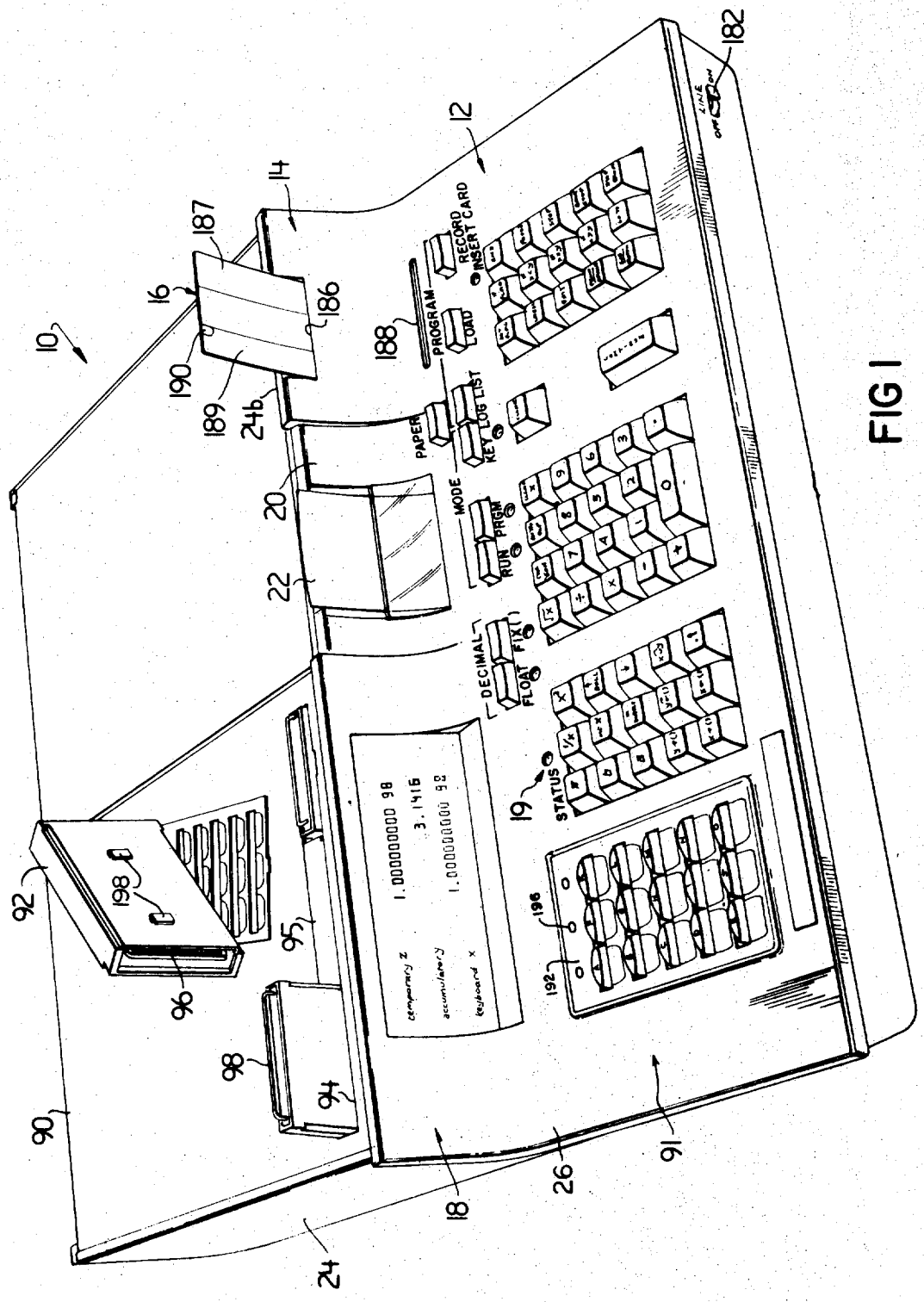

10
12
14
16
18
19
20
22
24
24b
26
90
91
92
94
95
96
98
182
186
187
188
189
190
192
196
198
RECORD
INSERT CARD
PROGRAM
LOAD
PAPER
KEY LOG LIST
MODE
RUN PRGM
DECIMAL
FLOAT FIX
STATUS
LINE
OFF ON
1.000000000 98
3.1416
1.000000000 98
temporary z
accumulatory
keyboard x

FIG 1

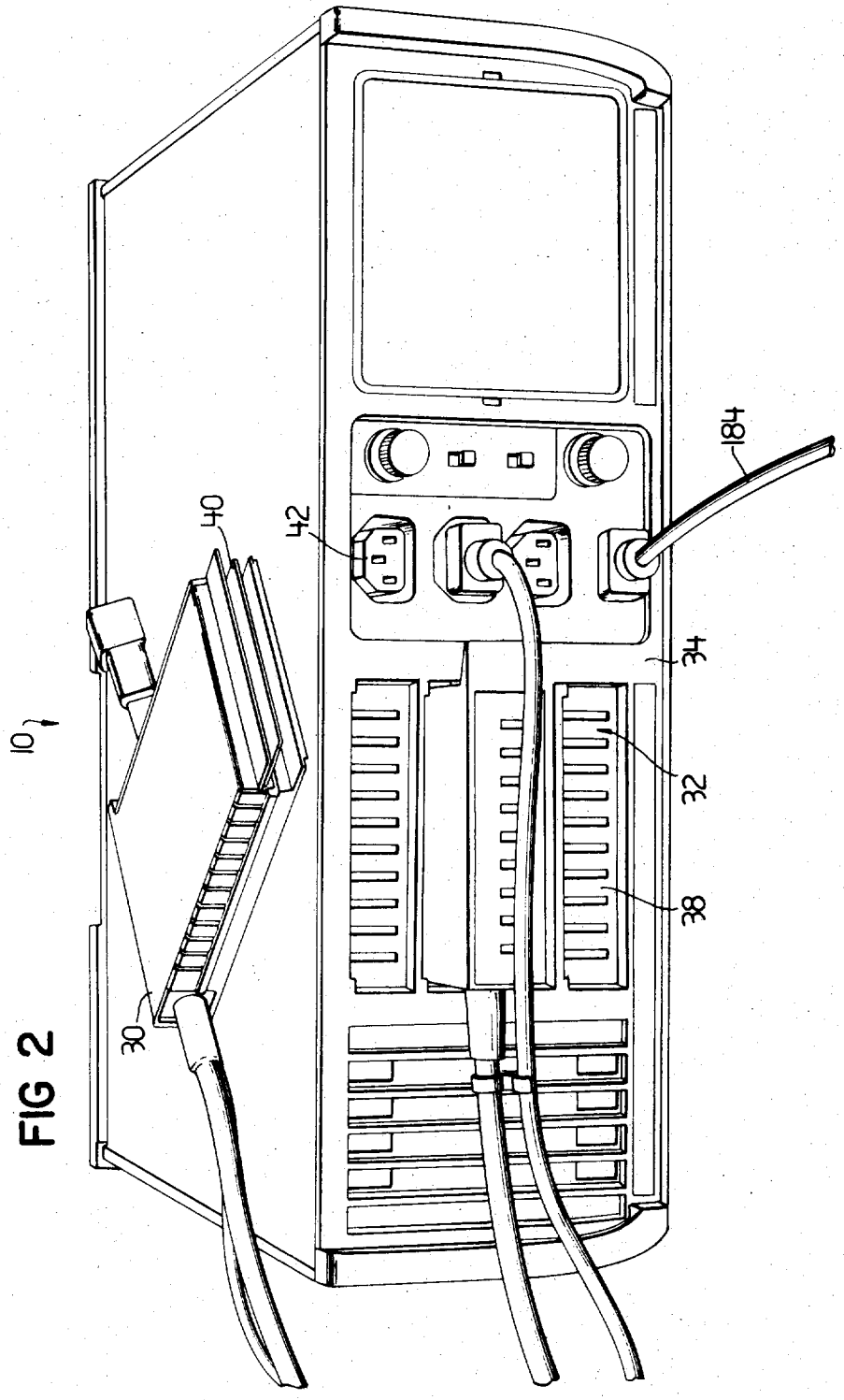


FIG 2

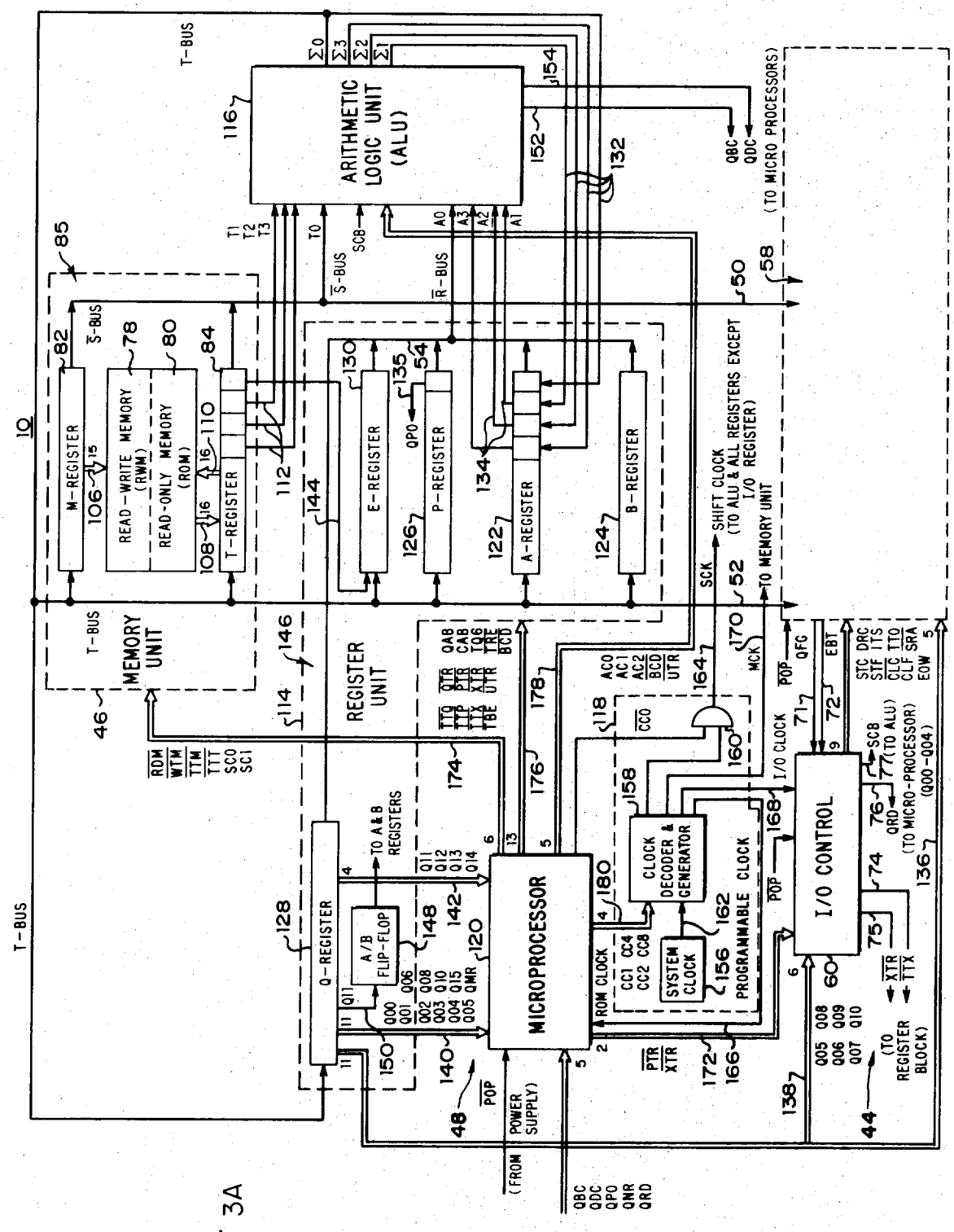

ARITHMETIC LOGIC UNIT (ALU)
MEMORY UNIT
REGISTER UNIT
M-REGISTER
READ-WRITE MEMORY (RWM)
READ-ONLY MEMORY (ROM)
T-REGISTER
E-REGISTER
P-REGISTER
A-REGISTER
B-REGISTER
Q-REGISTER
A/B FLIP-FLOP
MICROPROCESSOR
CLOCK DECODER & GENERATOR
SYSTEM CLOCK
PROGRAMMABLE CLOCK
I/O CONTROL
T-BUS
(TO MICRO PROCESSORS)
SHIFT CLOCK (TO ALU & ALL REGISTERS EXCEPT I/O REGISTER)
TO MEMORY UNIT
TO A&B REGISTERS
(FROM POWER SUPPLY)
(TO MICRO-PROCESSOR) (Q00-Q04)
(TO REGISTER BLOCK)

Fig 3A

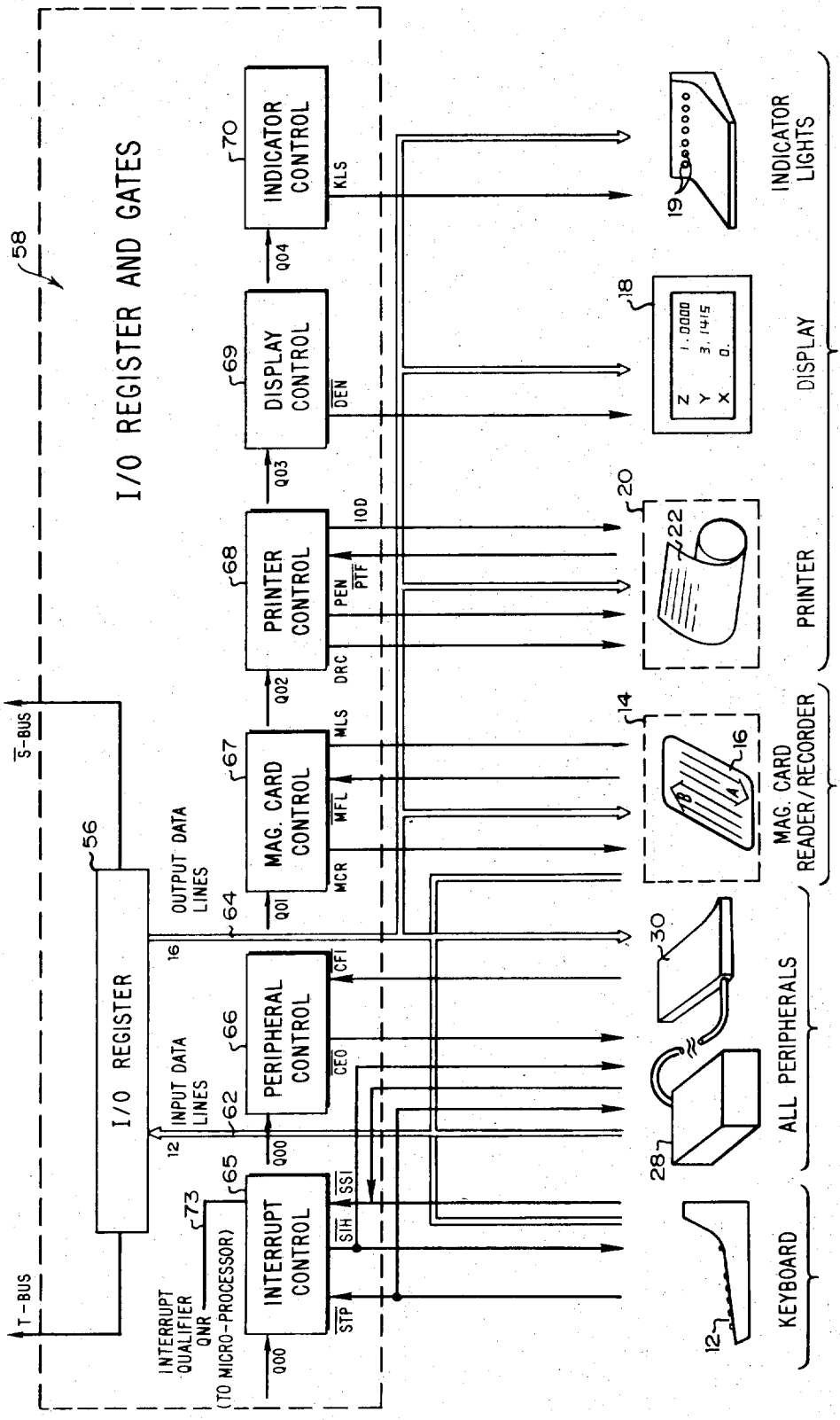

I/O REGISTER AND GATES
58
S-BUS
T-BUS
56
I/O REGISTER
16
OUTPUT DATA LINES
64
INPUT DATA LINES
12
62
INTERRUPT QUALIFIER QNR
(TO MICRO-PROCESSOR)
73
INTERRUPT CONTROL
65
PERIPHERAL CONTROL
66
MAG. CARD CONTROL
67
PRINTER CONTROL
68
DISPLAY CONTROL
69
INDICATOR CONTROL
70
Q00
Q01
Q02
Q03
Q04
SIH
SSI
STP
CEO
CFI
MCR
MFL
MLS
PEN
PTF
DRC
IOD
DEN
KLS
INTERRUPT INPUT
KEYBOARD
ALL PERIPHERALS
INPUT, OUTPUT
INPUT/OUTPUT OR
INTERRUPT INPUT
MAG. CARD READER/RECORDER
PRINTER
DISPLAY
INDICATOR LIGHTS
INPUT/OUTPUT
OUTPUT
12
28
30
14
16
20
22
18
19
Z
Y
X
1.0000
3.1415
0.

Fig 3B

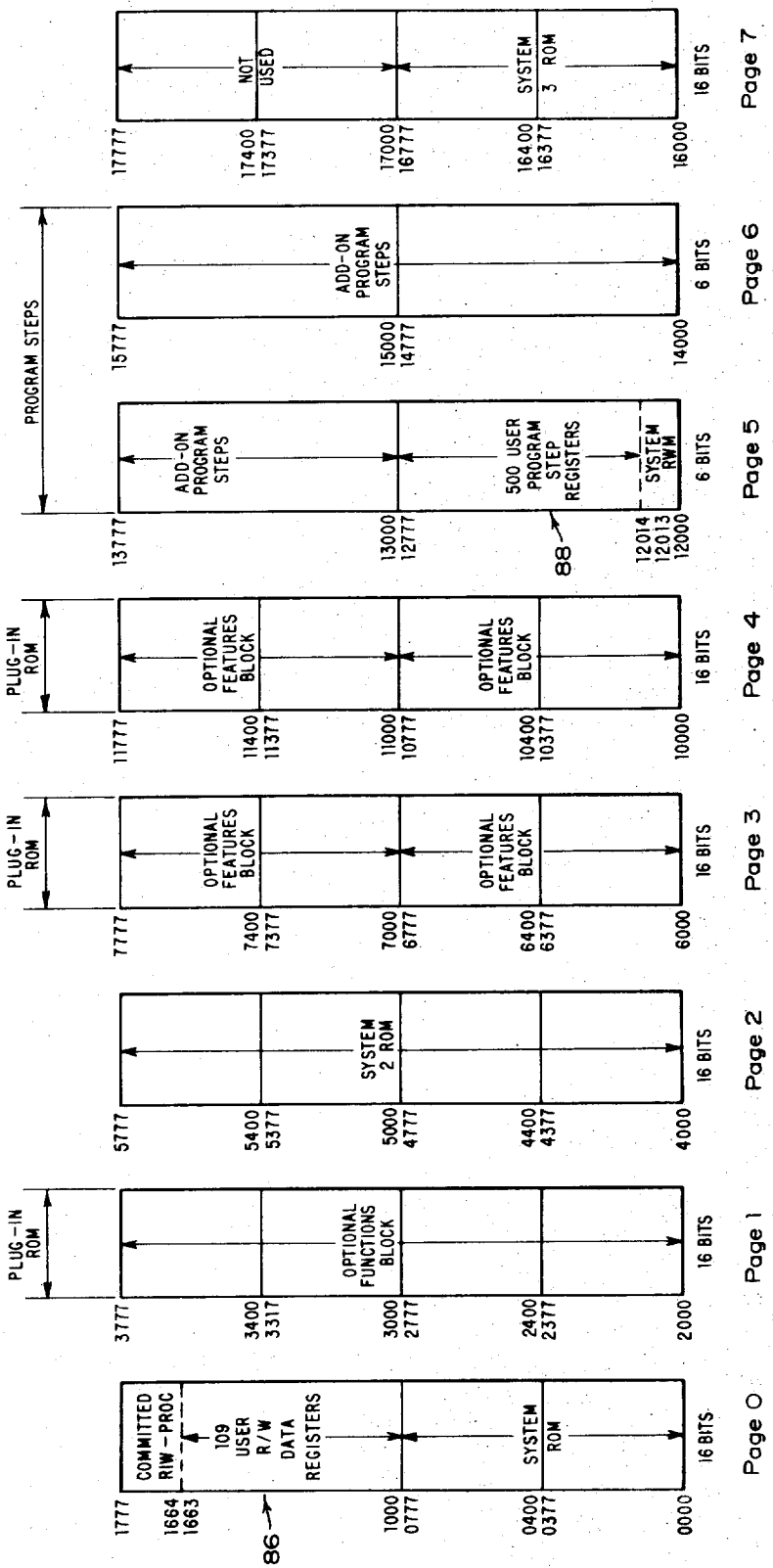

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
16 BITS
6 BITS
PLUG-IN ROM
PROGRAM STEPS
COMMITTED R/W - PROC
109 USER R/W DATA REGISTERS
SYSTEM ROM
OPTIONAL FUNCTIONS BLOCK
SYSTEM 2 ROM
OPTIONAL FEATURES BLOCK
ADD-ON PROGRAM STEPS
500 USER PROGRAM STEP REGISTERS
SYSTEM RWM
NOT USED
SYSTEM 3 ROM
86
88
1777
1664
1663
1000
0777
0400
0377
0000
3777
3400
3317
3000
2777
2400
2377
2000
5777
5400
5377
5000
4777
4400
4377
4000
7777
7400
7377
7000
6777
6400
6377
6000
11777
11400
11377
11000
10777
10400
10377
10000
13777
13000
12777
12014
12013
12000
15777
15000
14777
14000
17777
17400
17377
17000
16777
16400
16377
16000

Fig 4

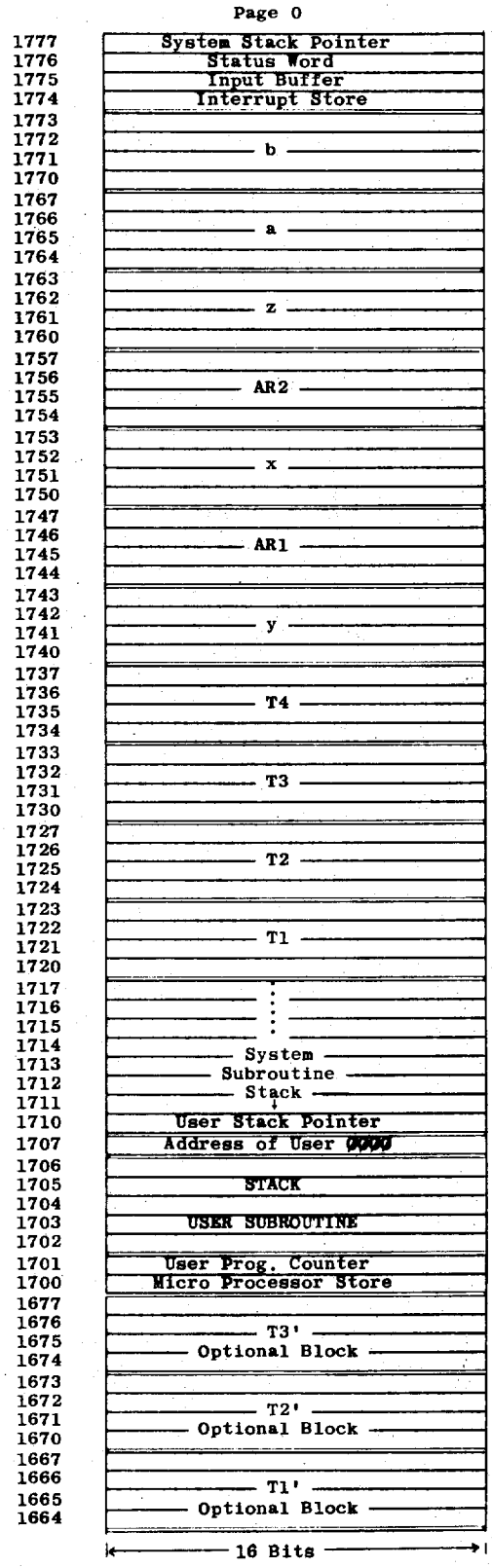

Page 0
1777 System Stack Pointer
1776 Status Word
1775 Input Buffer
1774 Interrupt Store
1773
1772
1771 b
1770
1767
1766
1765 a
1764
1763
1762
1761 z
1760
1757
1756
1755 AR2
1754
1753
1752
1751 x
1750
1747
1746
1745 AR1
1744
1743
1742
1741 y
1740
1737
1736
1735 T4
1734
1733
1732
1731 T3
1730
1727
1726
1725 T2
1724
1723
1722
1721 T1
1720
1717
1716
1715
1714
1713 System
1712 Subroutine
1711 Stack
1710 User Stack Pointer
1707 Address of User 0000
1706
1705 STACK
1704
1703 USER SUBROUTINE
1702
1701 User Prog. Counter
1700 Micro Processor Store
1677
1676
1675 T3'
1674 Optional Block
1673
1672
1671 T2'
1670 Optional Block
1667
1666
1665 T1'
1664 Optional Block
16 Bits

FIG 5

Page 5

12013	Security Word
12012	↑
12011	
12010	Available
12007	↓
12006	Print Code Buffer
12005	Temporary Display (Register Pointer)
12004	Temporary Display (Decimal Position)
12003	Temporary Display (Character Pointer)
12002	Print Flag
12001	Normalize Flag
12000	Automatic Execution Flag

← 6 Bits →

FIG 6

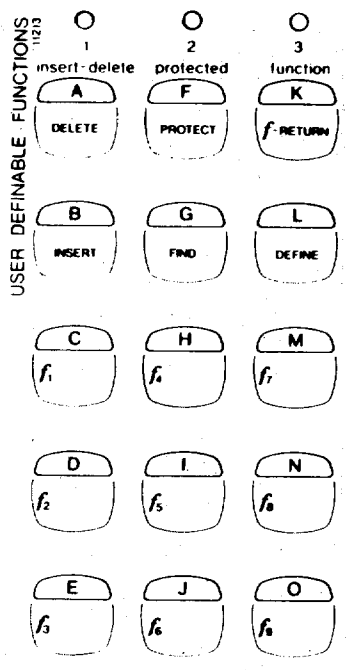


FIG 9

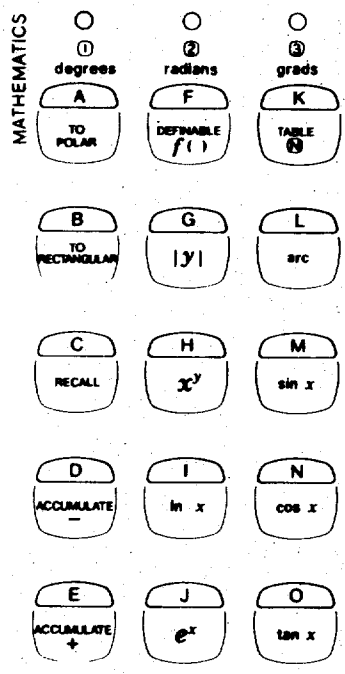


FIG 10

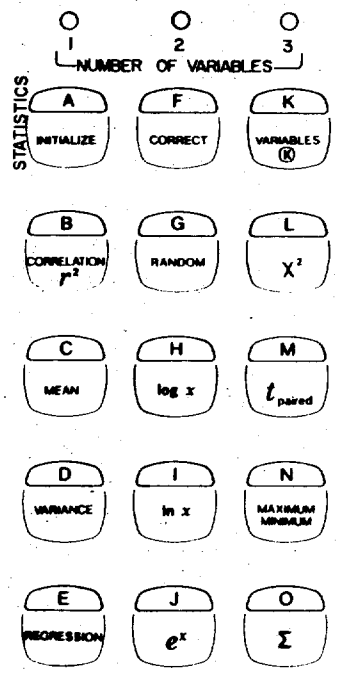


FIG 11

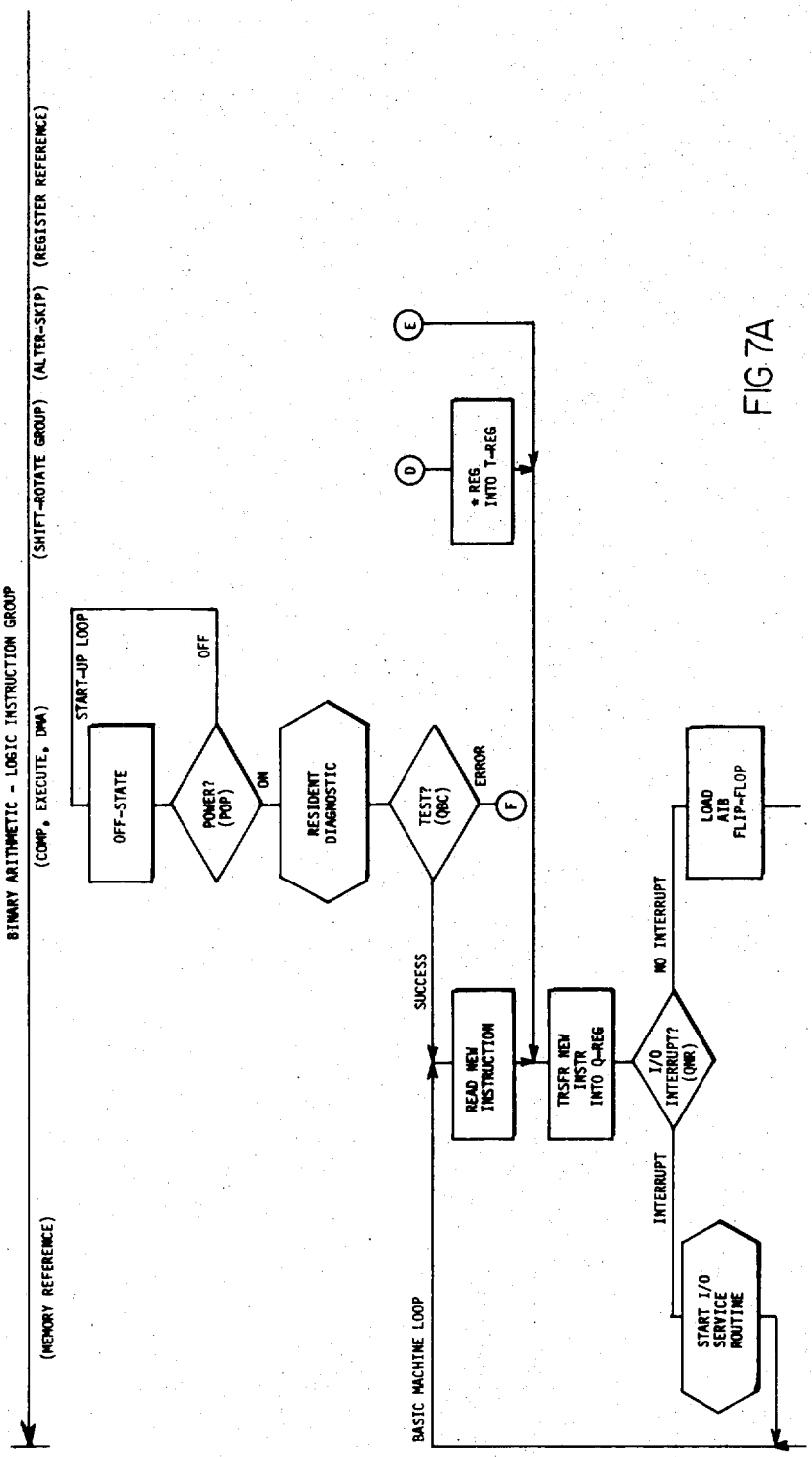

BINARY ARITHMETIC - LOGIC INSTRUCTION GROUP
(COMP, EXECUTE, DMA)
(SHIFT-ROTATE GROUP) (ALTER-SKIP) (REGISTER REFERENCE)
(MEMORY REFERENCE)
START-UP LOOP
OFF-STATE
POWER? (POP)
OFF
ON
RESIDENT DIAGNOSTIC
TEST? (OBC)
ERROR
E
D
* REG INTO T-REG
F
SUCCESS
READ MEM INSTRUCTION
TRSFR MEM INSTR INTO Q-REG
I/O INTERRUPT? (OMR)
NO INTERRUPT
LOAD AIB FLIP-FLOP
INTERRUPT
START I/O SERVICE ROUTINE
BASIC MACHINE LOOP

FIG 7A

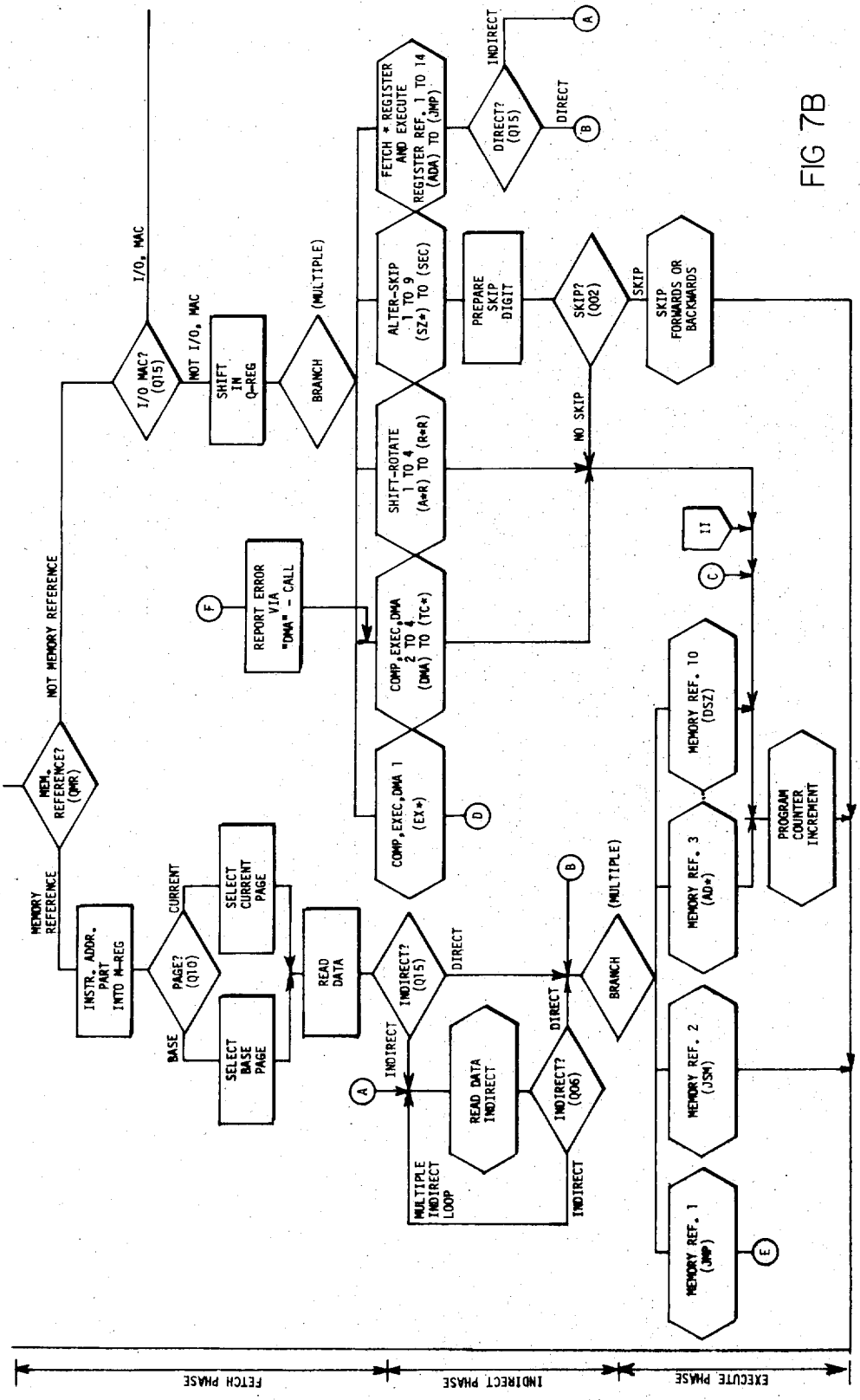
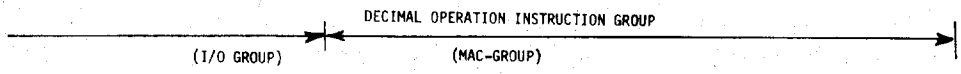

I/O, MAC
NOT I/O, MAC
I/O MAC? (Q15)
SHIFT IN Q-REG
BRANCH (MULTIPLE)
FETCH * REGISTER AND EXECUTE REGISTER REF. 1 TO 14 (ADA) TO (JMP)
DIRECT? (Q15)
INDIRECT
DIRECT
ALTER-SKIP 1 TO 9 (SZ*) TO (SEC)
PREPARE SKIP DIGIT
SKIP? (Q02)
SKIP
SKIP FORWARDS OR BACKWARDS
NO SKIP
SHIFT-ROTATE 1 TO 4 (A*R) TO (R*R)
COMP, EXEC, DMA 2 TO 4 (DMA) TO (TC*)
REPORT ERROR VIA "DMA" - CALL
COMP, EXEC, DMA 1 (EX*)
NOT MEMORY REFERENCE
MEM. REFERENCE? (QMR)
MEMORY REFERENCE
INSTR. ADDR. PART INTO M-REG
PAGE? (Q10)
CURRENT
BASE
SELECT CURRENT PAGE
SELECT BASE PAGE
READ DATA
INDIRECT? (Q15)
DIRECT
INDIRECT
READ DATA INDIRECT
INDIRECT? (Q06)
DIRECT
INDIRECT
MULTIPLE INDIRECT LOOP
BRANCH (MULTIPLE)
MEMORY REF. 10 (DSZ)
MEMORY REF. 3 (AD*)
MEMORY REF. 2 (JSM)
MEMORY REF. 1 (JMP)
PROGRAM COUNTER INCREMENT
FETCH PHASE
INDIRECT PHASE
EXECUTE PHASE

FIG 7B



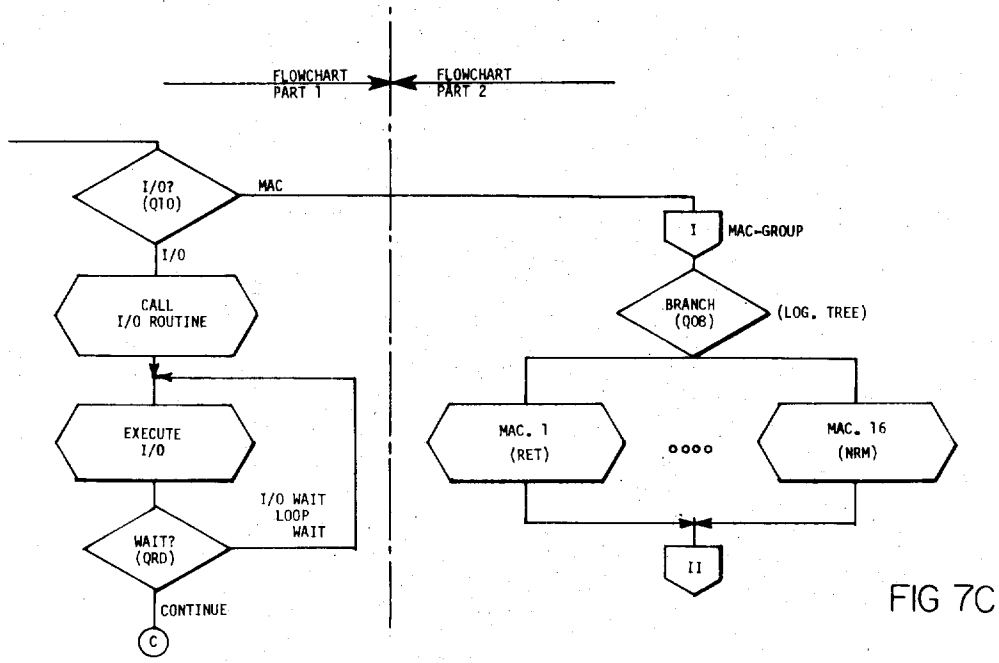


FIG 7C

NOTE: SUBROUTINES FOR INDIVIDUAL INSTRUCTION EXECUTION USE COMMON PATHS, COMMON LOOPS, COMMON BRANCHES, ETC. IN THE DETAIL FLOWCHART

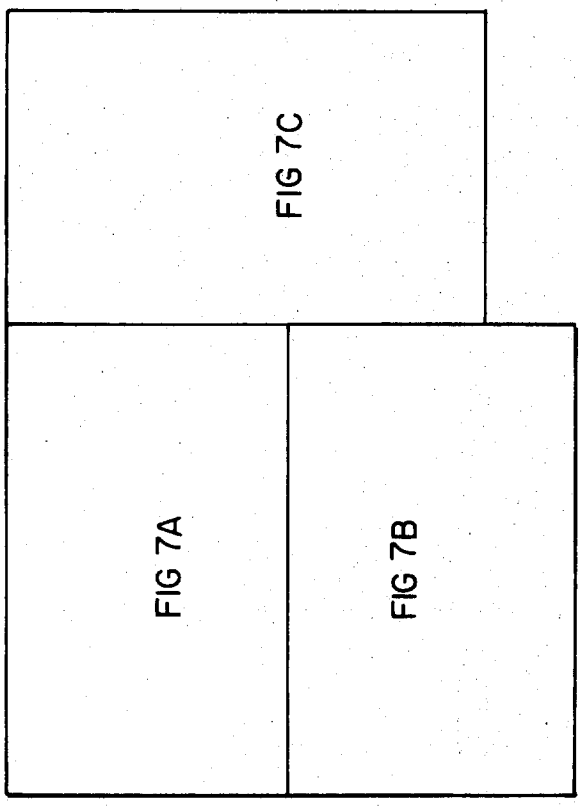


FIG 7'

BASIC KEYBOARD AND PRINTER ALPHA KEYBOARD

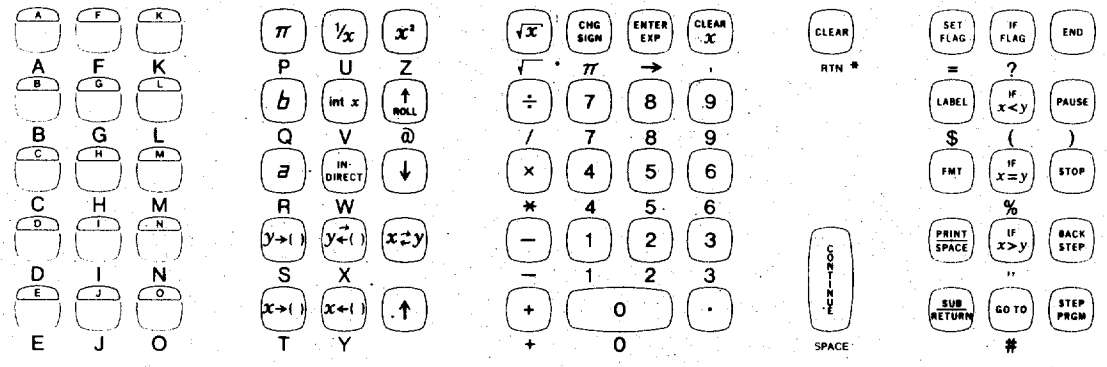


FIG 8

CALCULATOR KEYBOARD TO TYPEWRITER† UNSHIFTED AND SHIFTED

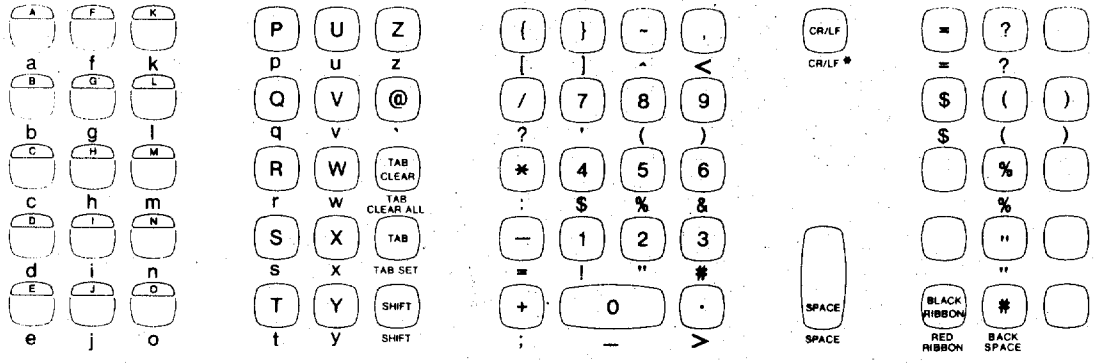


FIG 12

*CARRIAGE RETURN/LINE FEED

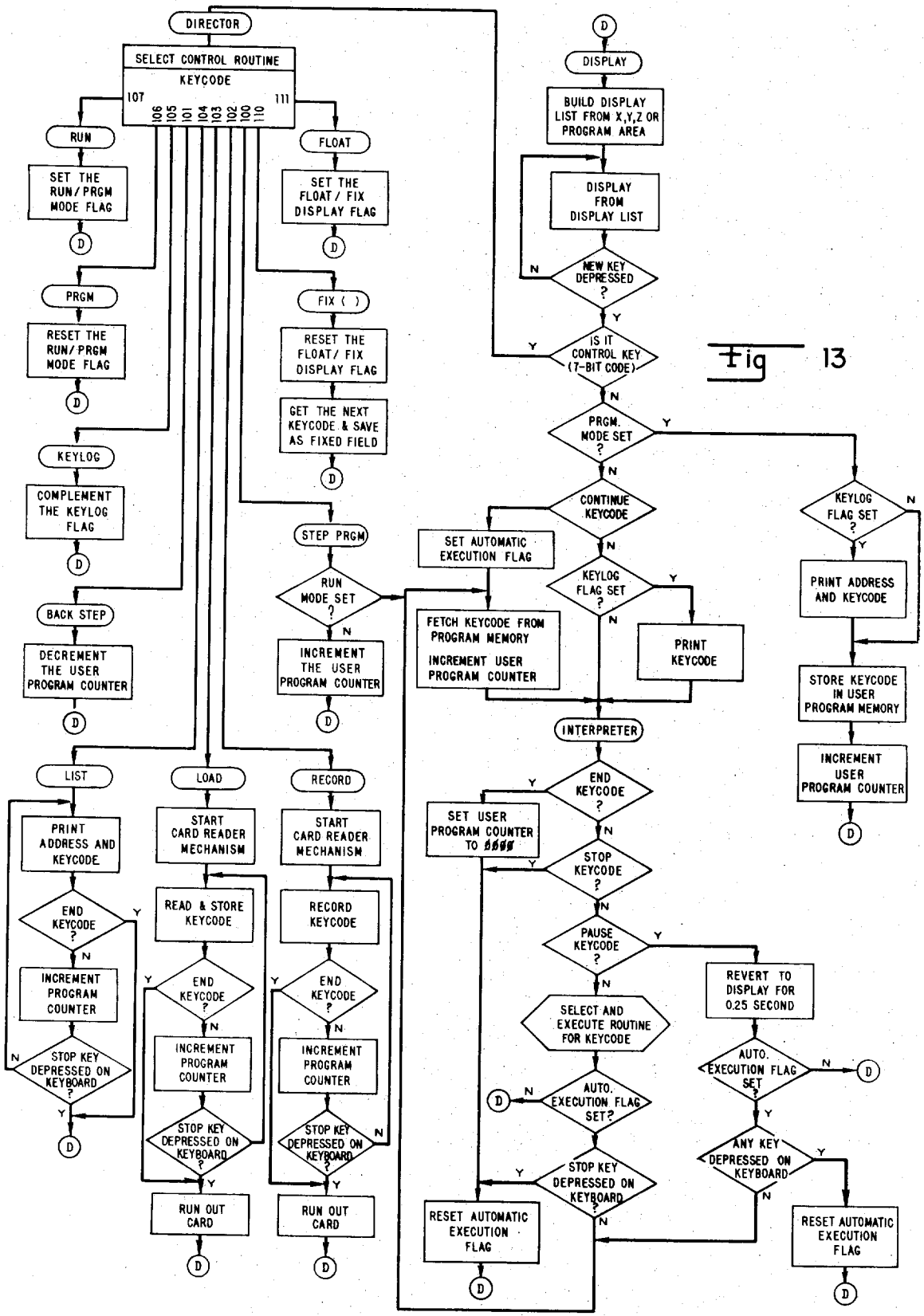

DIRECTOR
SELECT CONTROL ROUTINE
KEYCODE
107
111
106 105 101 104 103 102 100 110
RUN
SET THE RUN/PRGM MODE FLAG
PRGM
RESET THE RUN/PRGM MODE FLAG
KEYLOG
COMPLEMENT THE KEYLOG FLAG
BACK STEP
DECREMENT THE USER PROGRAM COUNTER
FLOAT
SET THE FLOAT/FIX DISPLAY FLAG
FIX ( )
RESET THE FLOAT/FIX DISPLAY FLAG
GET THE NEXT KEYCODE & SAVE AS FIXED FIELD
STEP PRGM
RUN MODE SET ?
INCREMENT THE USER PROGRAM COUNTER
LIST
PRINT ADDRESS AND KEYCODE
END KEYCODE ?
INCREMENT PROGRAM COUNTER
STOP KEY DEPRESSED ON KEYBOARD ?
LOAD
START CARD READER MECHANISM
READ & STORE KEYCODE
END KEYCODE ?
INCREMENT PROGRAM COUNTER
STOP KEY DEPRESSED ON KEYBOARD ?
RUN OUT CARD
RECORD
START CARD READER MECHANISM
RECORD KEYCODE
END KEYCODE ?
INCREMENT PROGRAM COUNTER
STOP KEY DEPRESSED ON KEYBOARD ?
RUN OUT CARD
DISPLAY
BUILD DISPLAY LIST FROM X,Y,Z OR PROGRAM AREA
DISPLAY FROM DISPLAY LIST
NEW KEY DEPRESSED ?
IS IT CONTROL KEY (7-BIT CODE)
PRGM. MODE SET ?
CONTINUE KEYCODE
SET AUTOMATIC EXECUTION FLAG
KEYLOG FLAG SET ?
FETCH KEYCODE FROM PROGRAM MEMORY
INCREMENT USER PROGRAM COUNTER
PRINT KEYCODE
INTERPRETER
END KEYCODE ?
SET USER PROGRAM COUNTER TO ####
STOP KEYCODE ?
PAUSE KEYCODE ?
SELECT AND EXECUTE ROUTINE FOR KEYCODE
AUTO. EXECUTION FLAG SET?
STOP KEY DEPRESSED ON KEYBOARD ?
RESET AUTOMATIC EXECUTION FLAG
REVERT TO DISPLAY FOR 0.25 SECOND
AUTO. EXECUTION FLAG SET ?
ANY KEY DEPRESSED ON KEYBOARD
RESET AUTOMATIC EXECUTION FLAG
KEYLOG FLAG SET ?
PRINT ADDRESS AND KEYCODE
STORE KEYCODE IN USER PROGRAM MEMORY
INCREMENT USER PROGRAM COUNTER

Fig 13

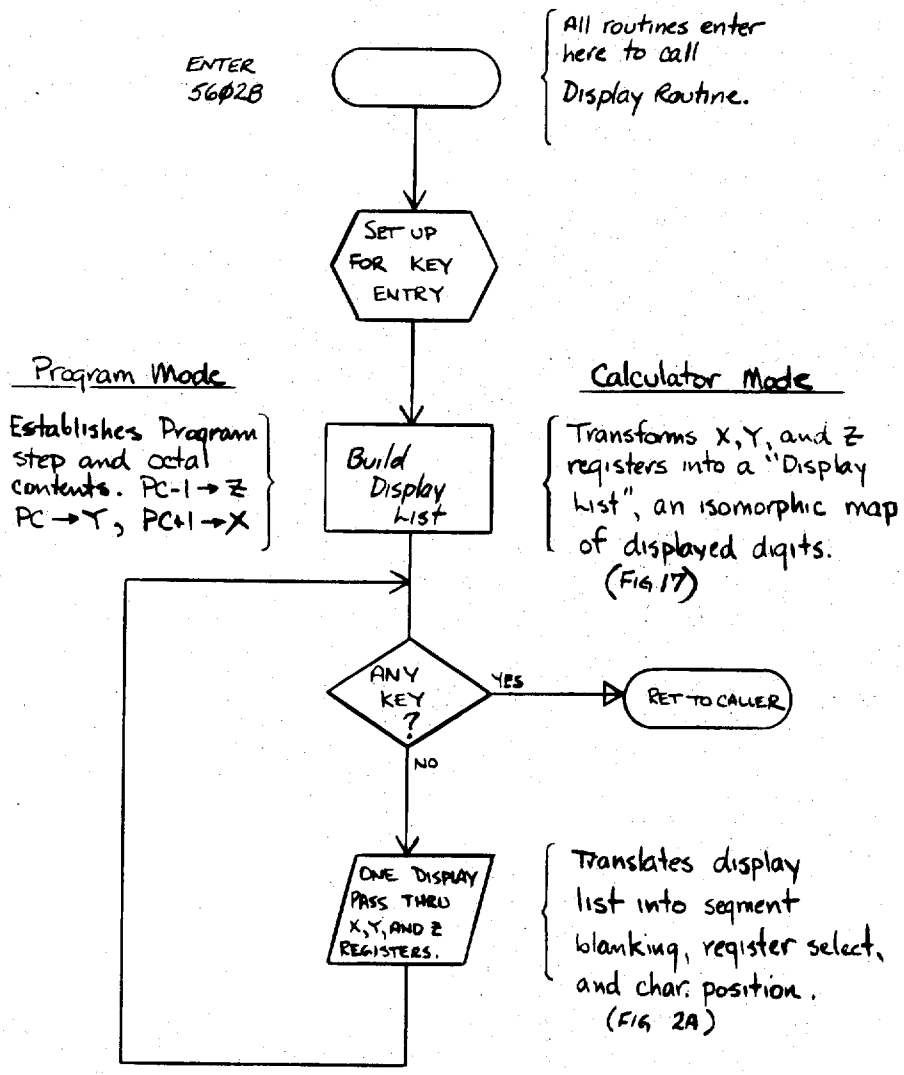


FIG 14

Basic Display Process.

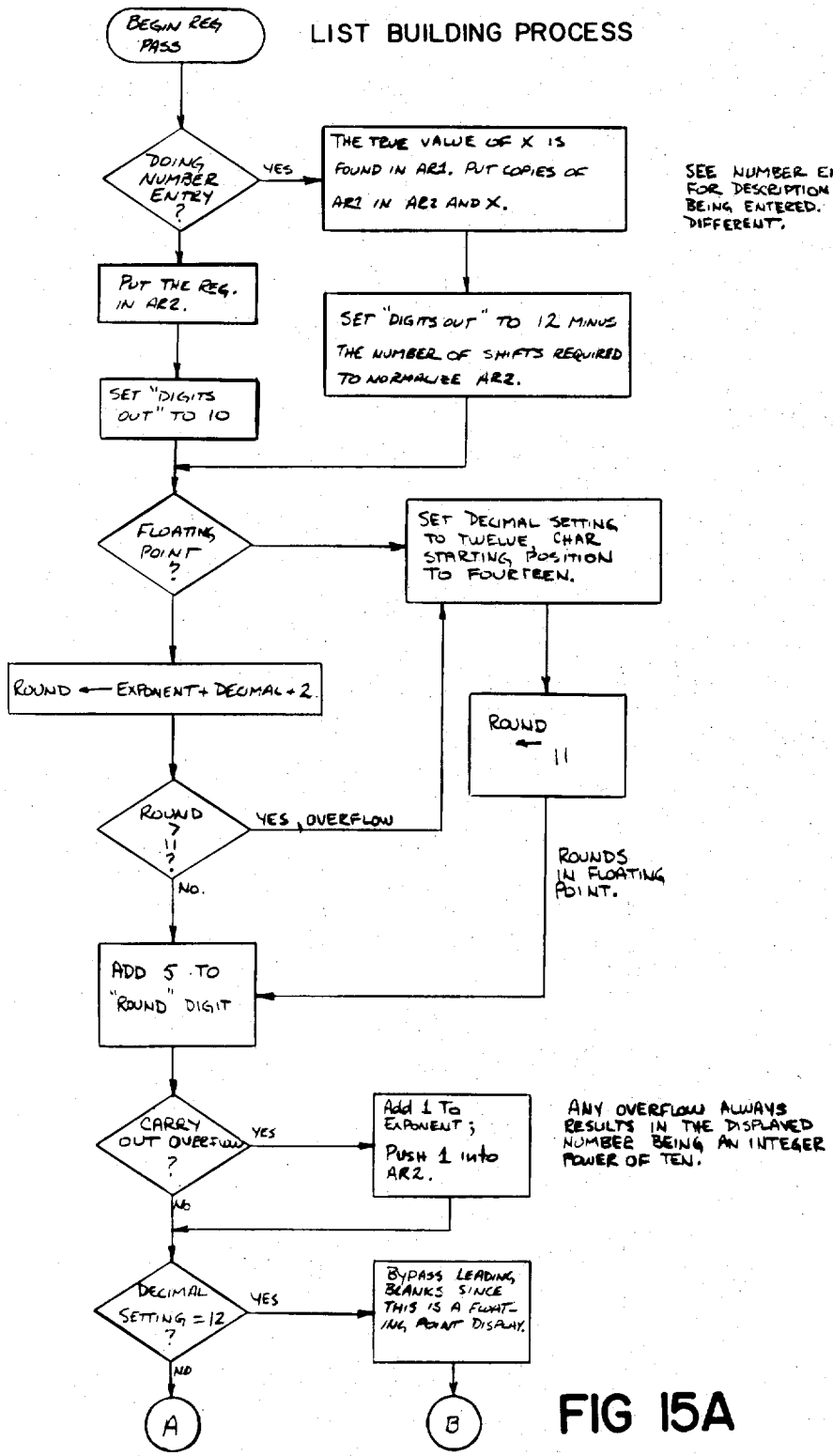

LIST BUILDING PROCESS
BEGIN REG PASS
DOING NUMBER ENTRY ?
YES
THE TRUE VALUE OF X IS FOUND IN AR1. PUT COPIES OF AR1 IN AR2 AND X.
SEE NUMBER ENTRY ROUTINE FOR DESCRIPTION OF NUMBER BEING ENTERED. FORMAT IS DIFFERENT.
PUT THE REG. IN AR2.
SET "DIGITS OUT" TO 12 MINUS THE NUMBER OF SHIFTS REQUIRED TO NORMALIZE AR2.
SET "DIGITS OUT" TO 10
FLOATING POINT ?
SET DECIMAL SETTING TO TWELVE, CHAR STARTING POSITION TO FOURTEEN.
ROUND ← EXPONENT + DECIMAL + 2
ROUND ← 11
ROUND = 11 ?
YES, OVERFLOW
ROUNDS IN FLOATING POINT.
No.
ADD 5 TO "ROUND" DIGIT
CARRY OUT OVERFLOW ?
YES
Add 1 To EXPONENT; PUSH 1 into AR2.
ANY OVERFLOW ALWAYS RESULTS IN THE DISPLAYED NUMBER BEING AN INTEGER POWER OF TEN.
No
DECIMAL SETTING = 12 ?
YES
BYPASS LEADING BLANKS SINCE THIS IS A FLOATING POINT DISPLAY.
NO
A
B

FIG 15A

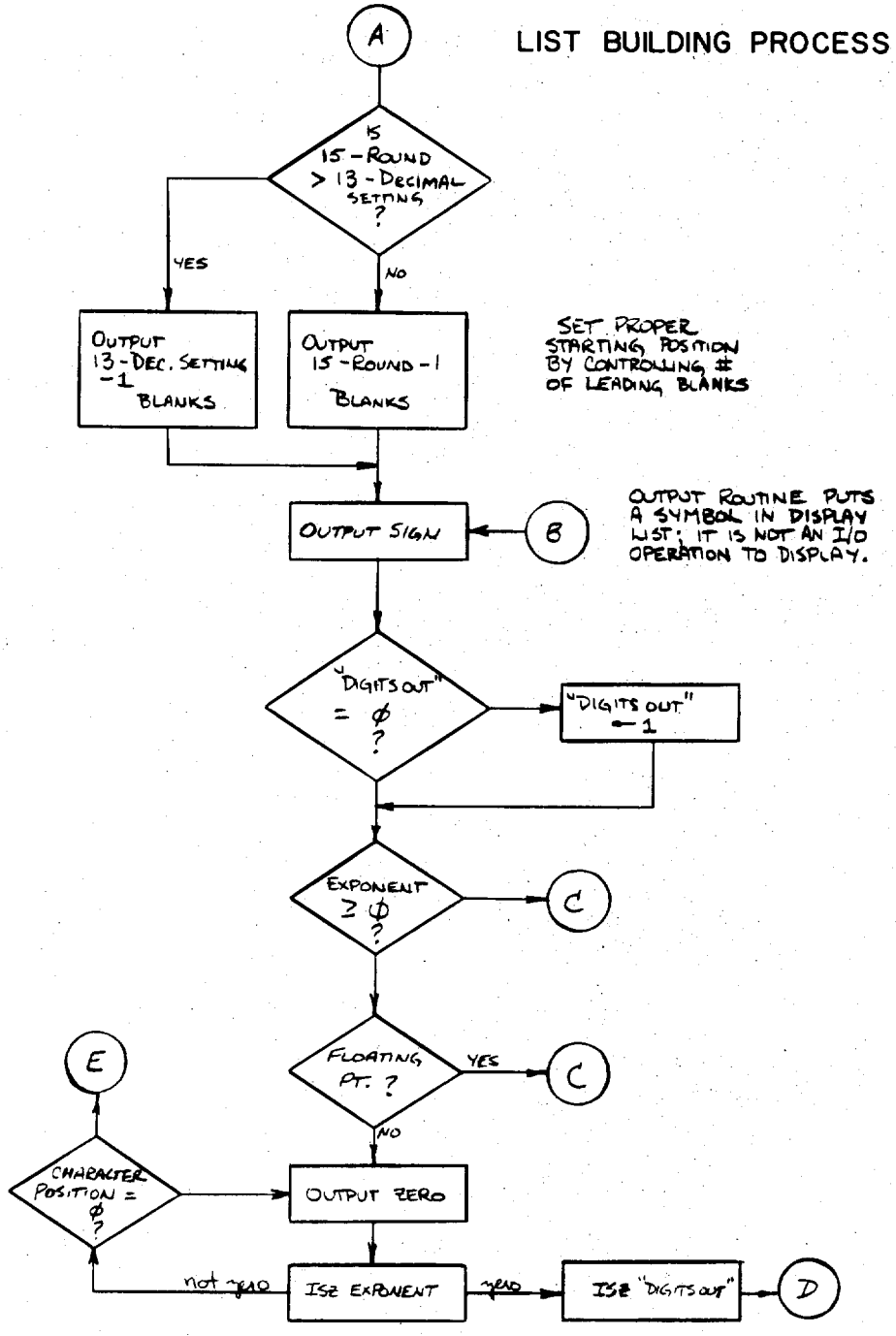

A
LIST BUILDING PROCESS
IS 15 - ROUND > 13 - DECIMAL SETTING ?
YES
NO
OUTPUT 13 - DEC. SETTING - 1 BLANKS
OUTPUT 15 - ROUND - 1 BLANKS
SET PROPER STARTING POSITION BY CONTROLLING # OF LEADING BLANKS
OUTPUT SIGN
B
OUTPUT ROUTINE PUTS A SYMBOL IN DISPLAY LIST; IT IS NOT AN I/O OPERATION TO DISPLAY.
"DIGITS OUT" = ϕ ?
"DIGITS OUT" ← 1
EXPONENT ≥ ϕ ?
C
FLOATING PT. ?
YES
C
NO
E
CHARACTER POSITION = ϕ ?
OUTPUT ZERO
not zero
ISZ EXPONENT
zero
ISZ "DIGITS OUT"
D

FIG 15B

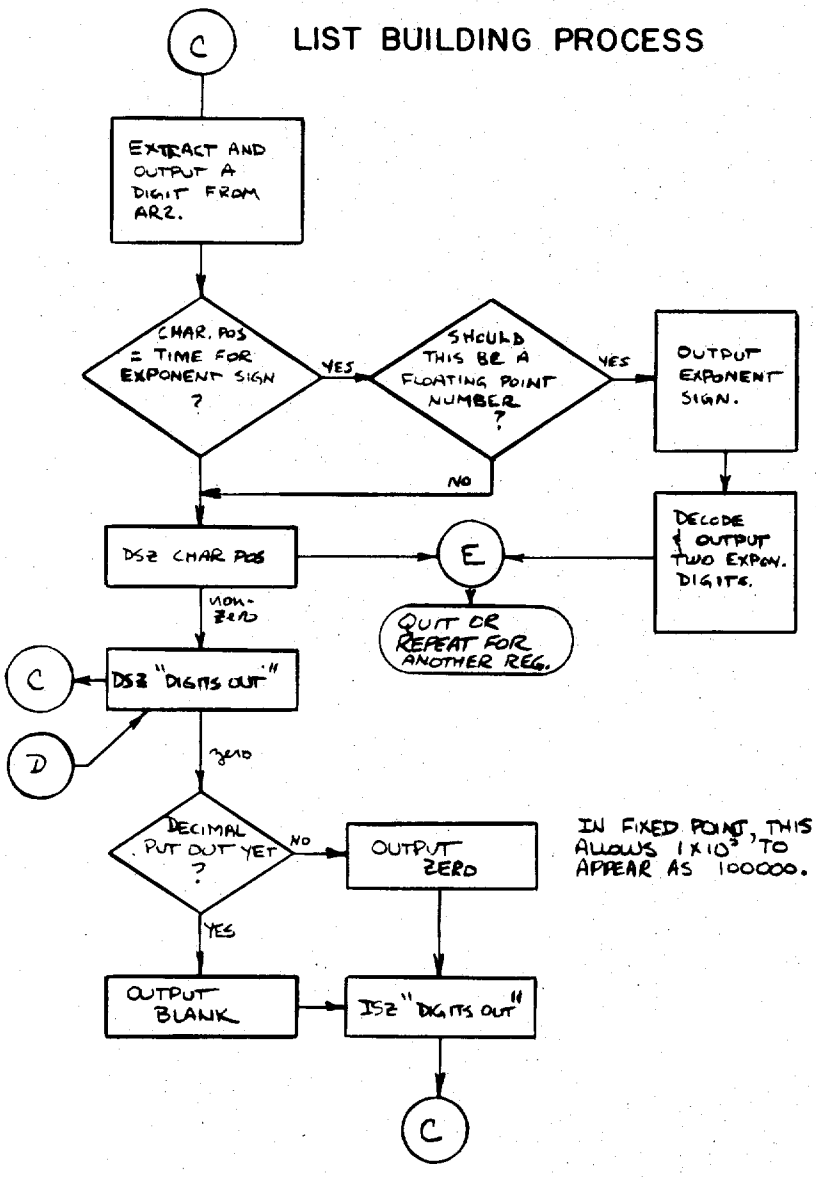
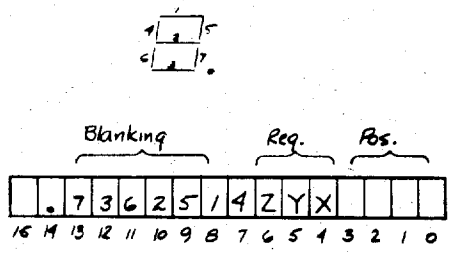
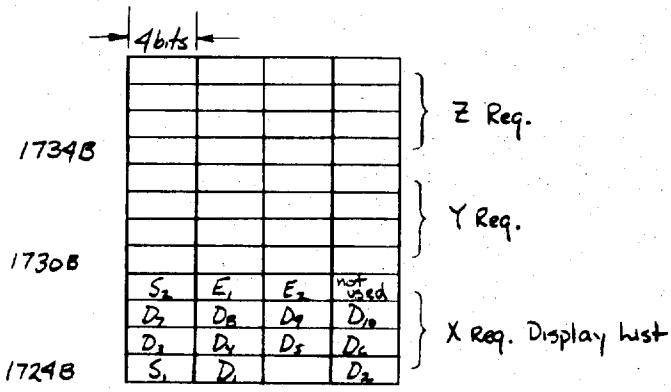

C
LIST BUILDING PROCESS
EXTRACT AND OUTPUT A DIGIT FROM AR2.
CHAR. POS = TIME FOR EXPONENT SIGN ?
YES
SHOULD THIS BE A FLOATING POINT NUMBER ?
YES
OUTPUT EXPONENT SIGN.
NO
DECODE & OUTPUT TWO EXPON. DIGITS.
DSZ CHAR POS
E
QUIT OR REPEAT FOR ANOTHER REG.
NON-ZERO
C
DSZ "DIGITS OUT"
D
ZERO
DECIMAL PUT OUT YET ?
NO
OUTPUT ZERO
IN FIXED POINT, THIS ALLOWS 1×10^5 TO APPEAR AS 100000.
YES
OUTPUT BLANK
ISZ "DIGITS OUT"
C

FIG 15C



Blanking: $\phi \Rightarrow$ Blank segment.
Req: $\phi \Rightarrow$ select register
Pos: $\phi \Rightarrow$ Leftmost char. in register.

FIG 16 Output bit Assignment for 7 segment display.



$-2.105764265-72$
$S_1 D_1 . D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10} S_2 E_1 E_2$

FIG 17 Display List built in Read/Write Area.

If the digit in the Display list is a:	The internal 4 bits are:
ϕ	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
-	1010
blank	1011
decimal	1100
asterisk (Printer only)	1101

Ex:
IF the displayed register is
2035 - - - - - - - - 46,
The display list must be

T+3	0100	0110	1011	1011
T+2	1010	1010	1010	1010
T+1	1010	1010	1010	1010
T	0010	0000	0011	0101

For some $T \in \{17248, 17308, 17348\}$

FIG 18 Internal Display List structure.

MONITOR FLOWCHART

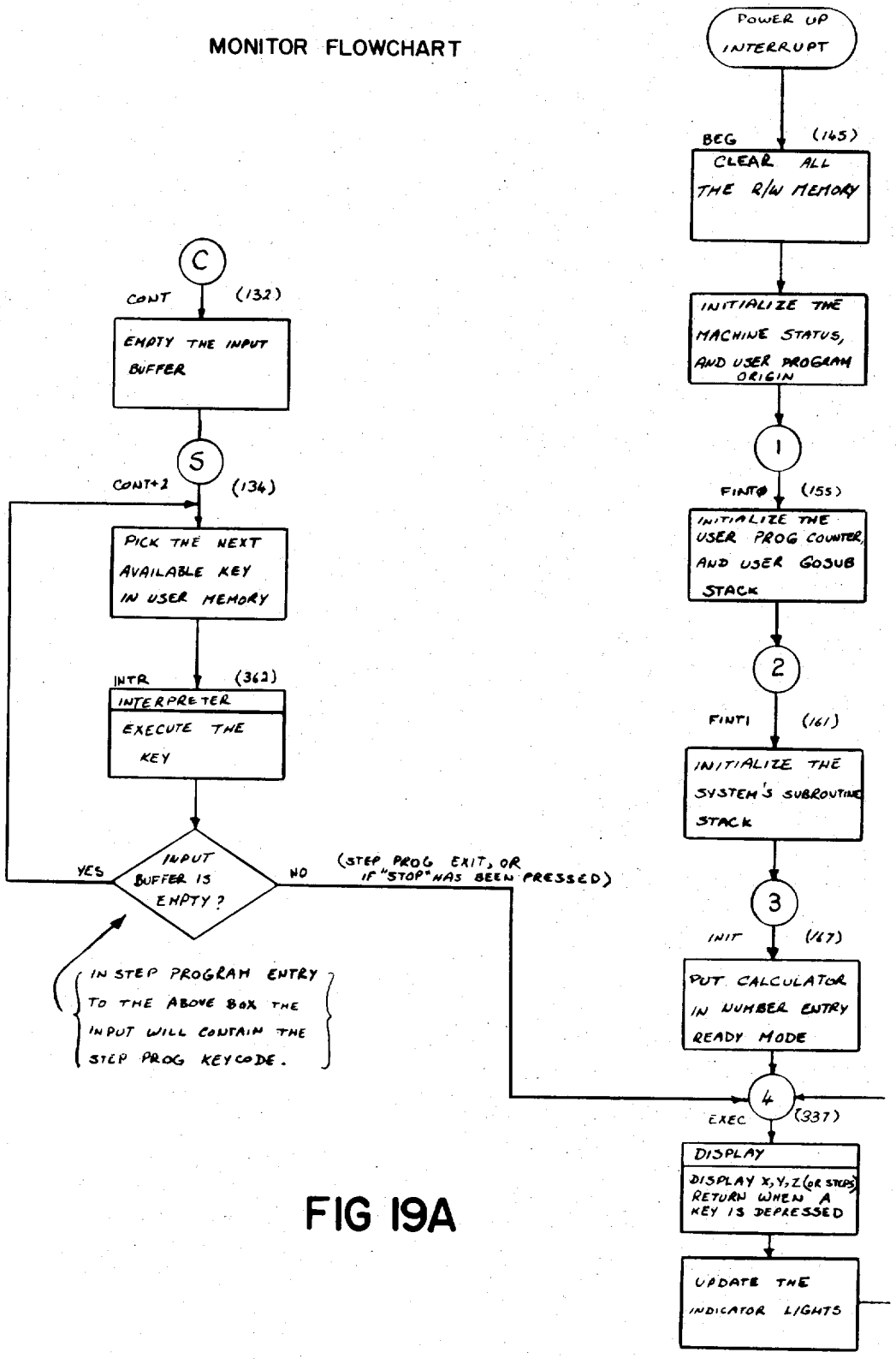


FIG 19A

MONITOR FLOWCHART

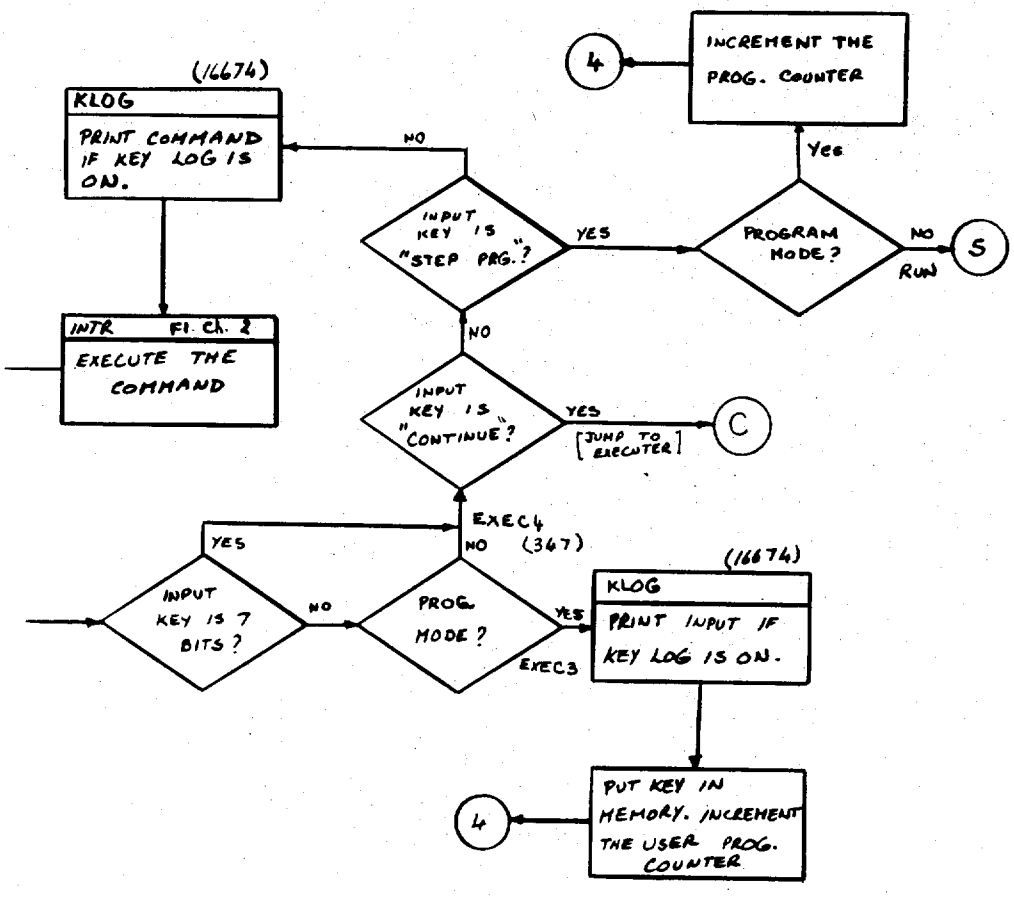
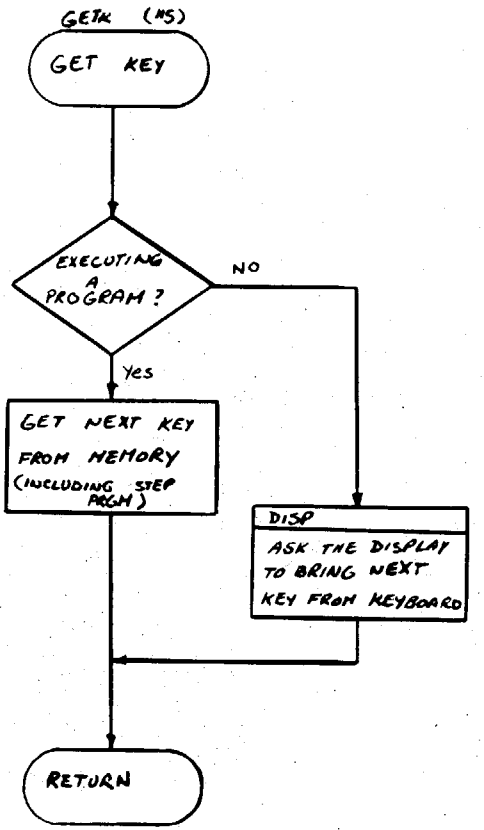


FIG 19B



GET NEXT KEY SUBROUTINE

FIG 20B

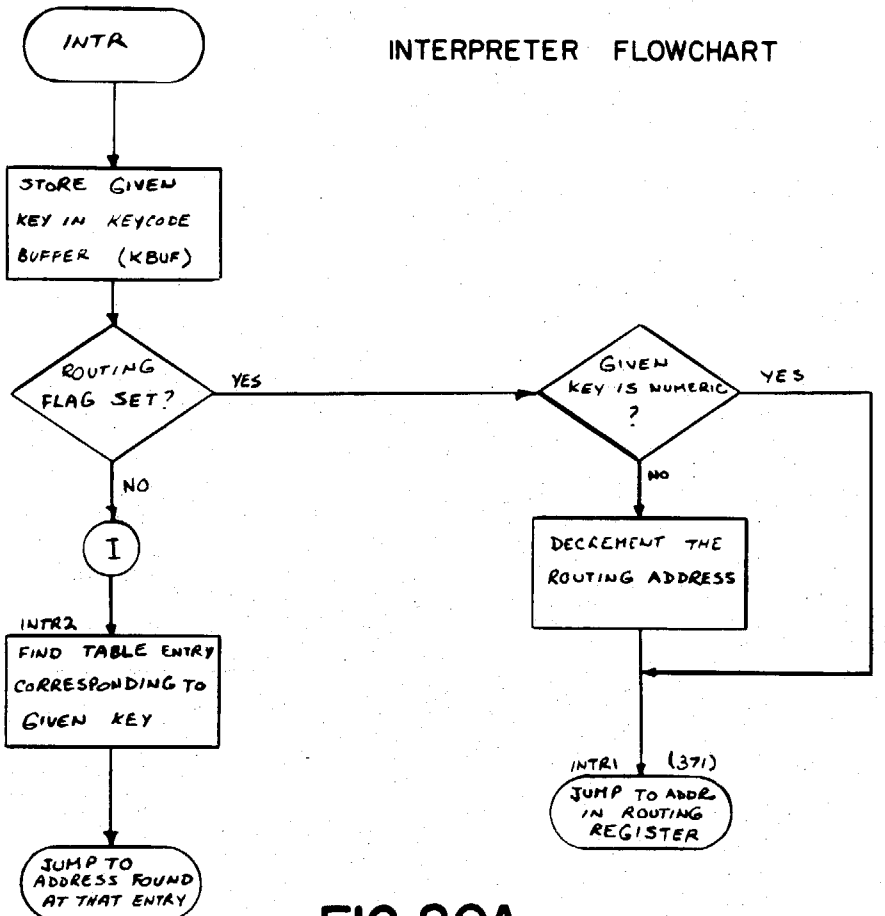
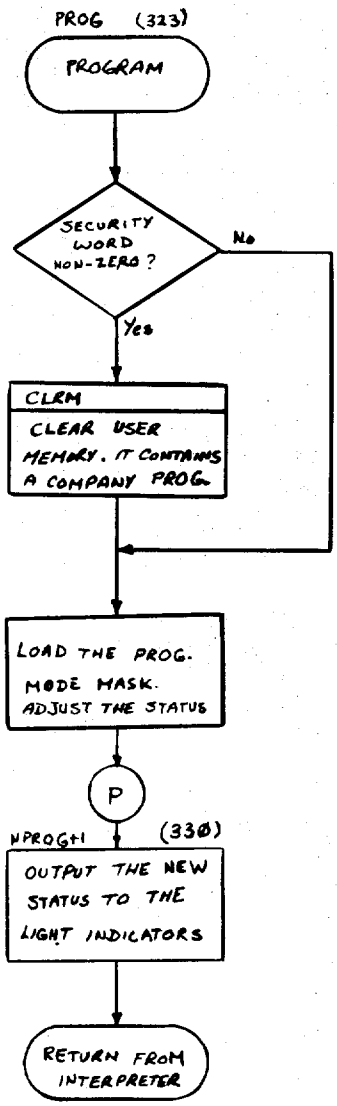


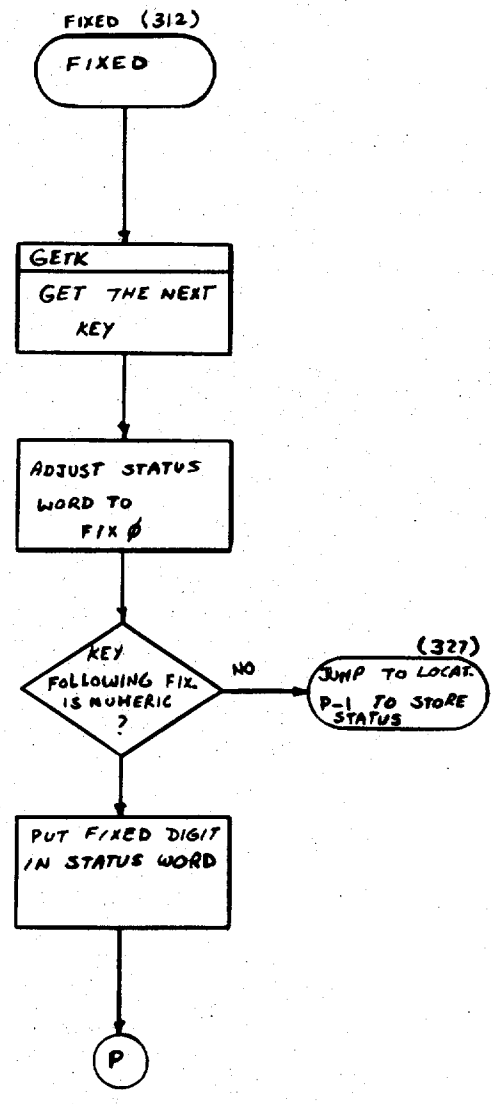
FIG 20A



DIRECTIVE TABLE ENTRY
No. 306

PROGRAM KEY PROCESSING
(KEYCODE 106)

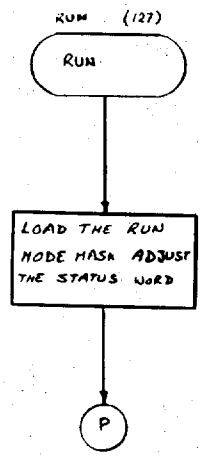
FIG 21A



DIRECTIVE TABLE ENTRY
No. 310

FIXED KEY PROCESSING
(KEYCODE 110)

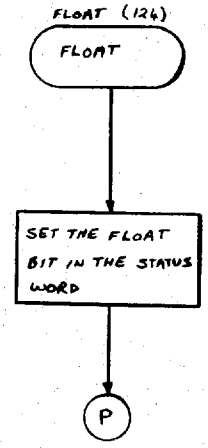
FIG 21D



DIRECTIVE TABLE ENTRY
No 307

RUN KEY PROCESSING
(KEYCODE 107)

FIG 21B



DIRECTIVE TABLE ENTRY
No. 311

FLOAT KEY PROCESSING
(KEYCODE 111)

FIG 21C

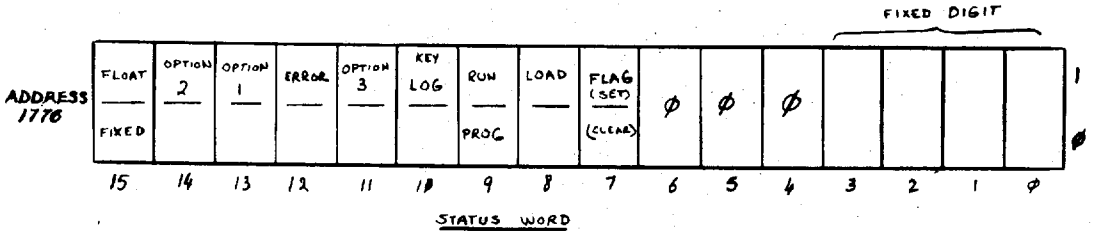
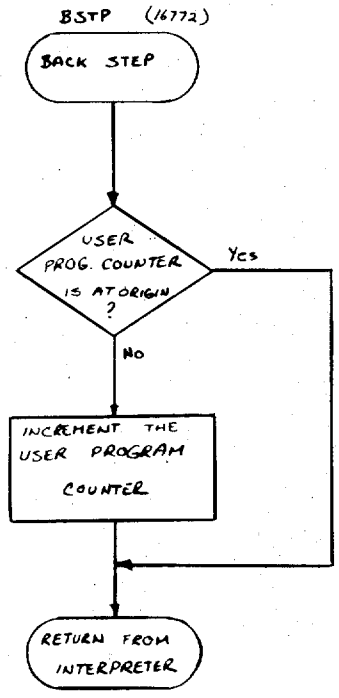


FIG 22

INTERPRETER

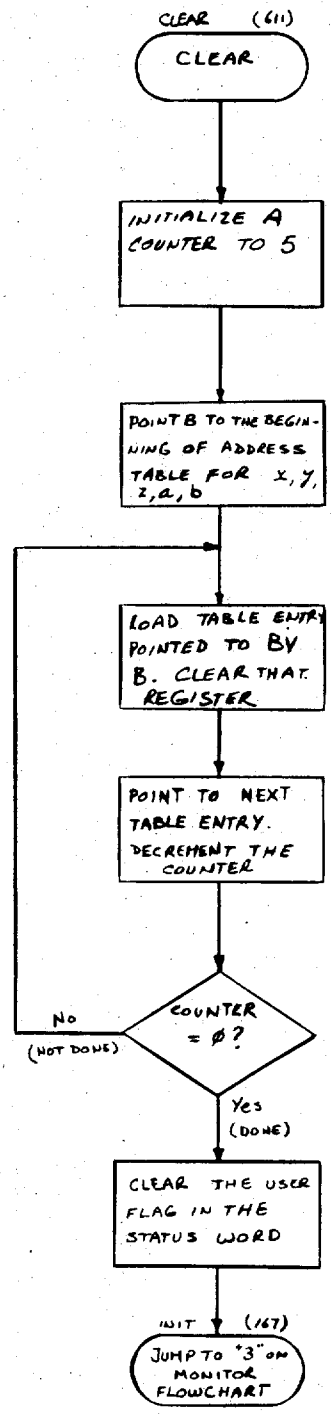
DIRECTIVE KEYS PROCESSING (7 bit Codes)



DIRECTIVE TABLE ENTRY
No. 301

BACK STEP KEY PROCESSING
(KEYCODE 101)
DIRECTIVE KEY

FIG 23

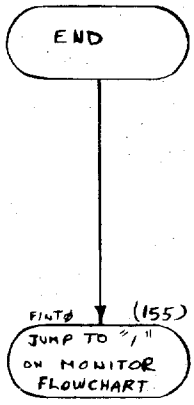


DIRECTIVE TABLE ENTRY
No. 220

CLEAR KEY PROCESSING
(KEYCODE 20)

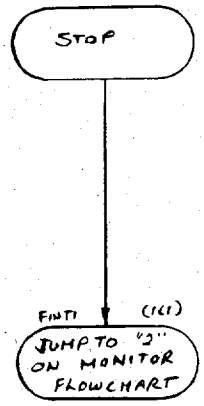
FIG 24

INTERPRETER FLOWCHARTS



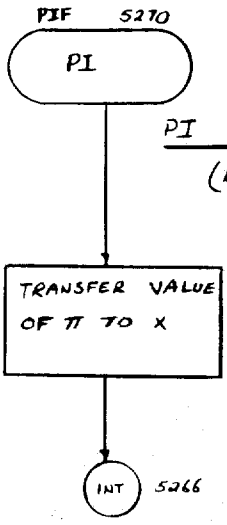
DIRECTIVE TABLE ENTRY
No 246

END KEY PROCESSING
(KEYCODE 46)
FIG 25



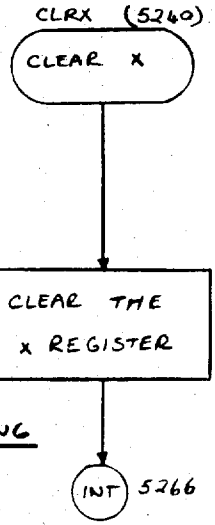
DIRECTIVE TABLE ENTRY
No. 241

STOP KEY PROCESSING
(KEYCODE 41)
FIG 26



PI PROCESSING
(KEYCODE 56)
FIG 28

DIRECTIVE TABLE ENTRY
No. 256

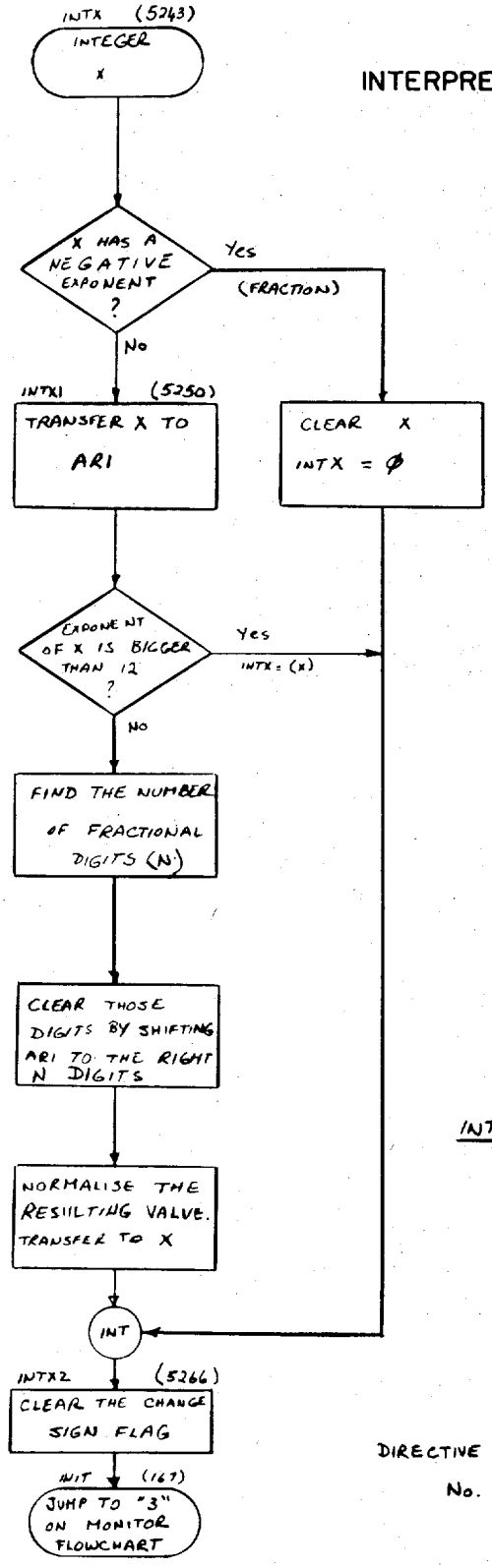


CLEAR X PROCESSING
(KEYCODE 37)
FIG 29

DIRECTIVE TABLE ENTRY
No. 237

INTERPRETER FLOWCHARTS

INTERPRETER FLOWCHART



INTEGER X PROCESSING

(KEYCODE 64)

FIG 27

DIRECTIVE TABLE ENTRY

No. 264

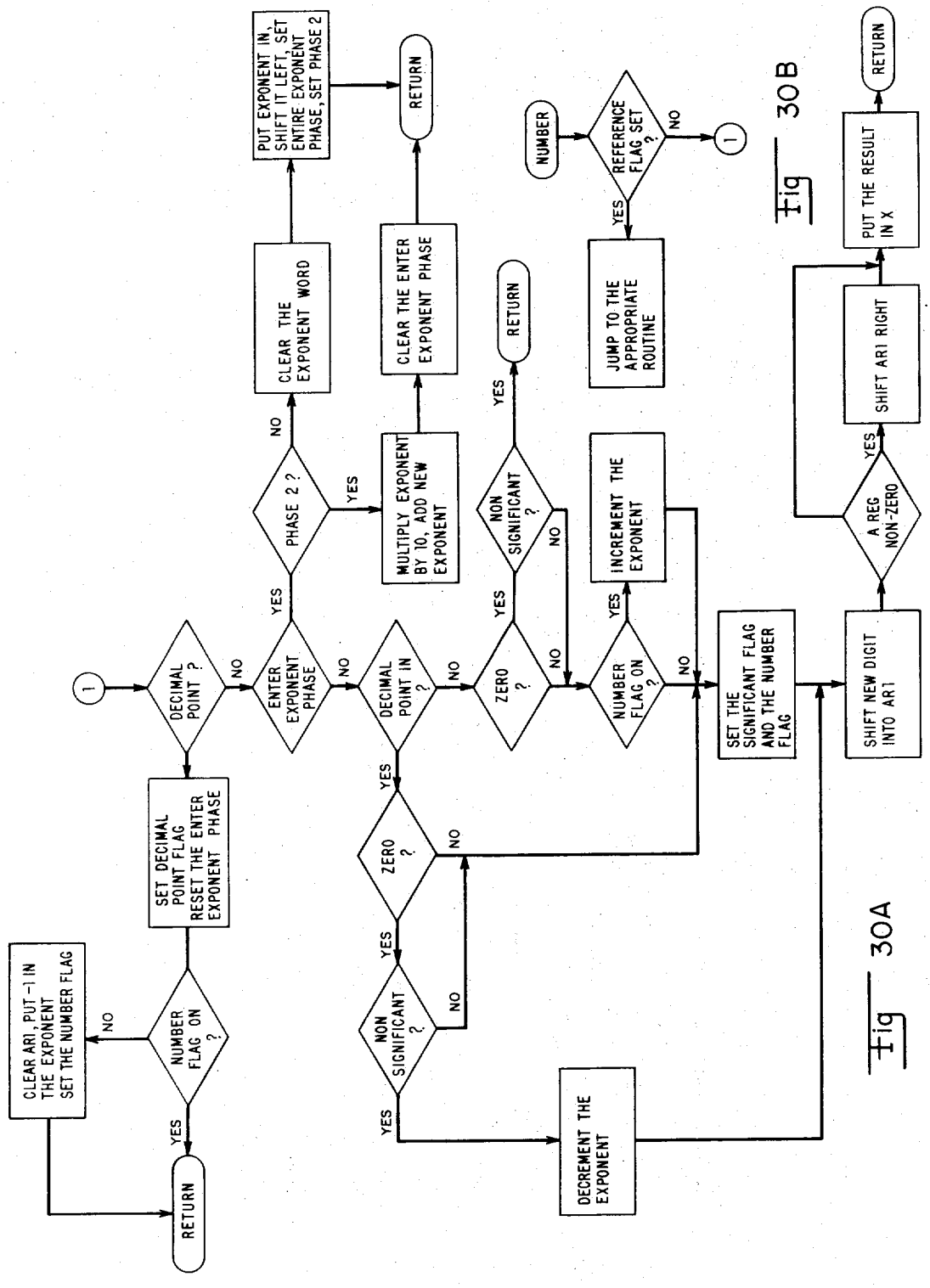


Fig 30A

Fig 30B

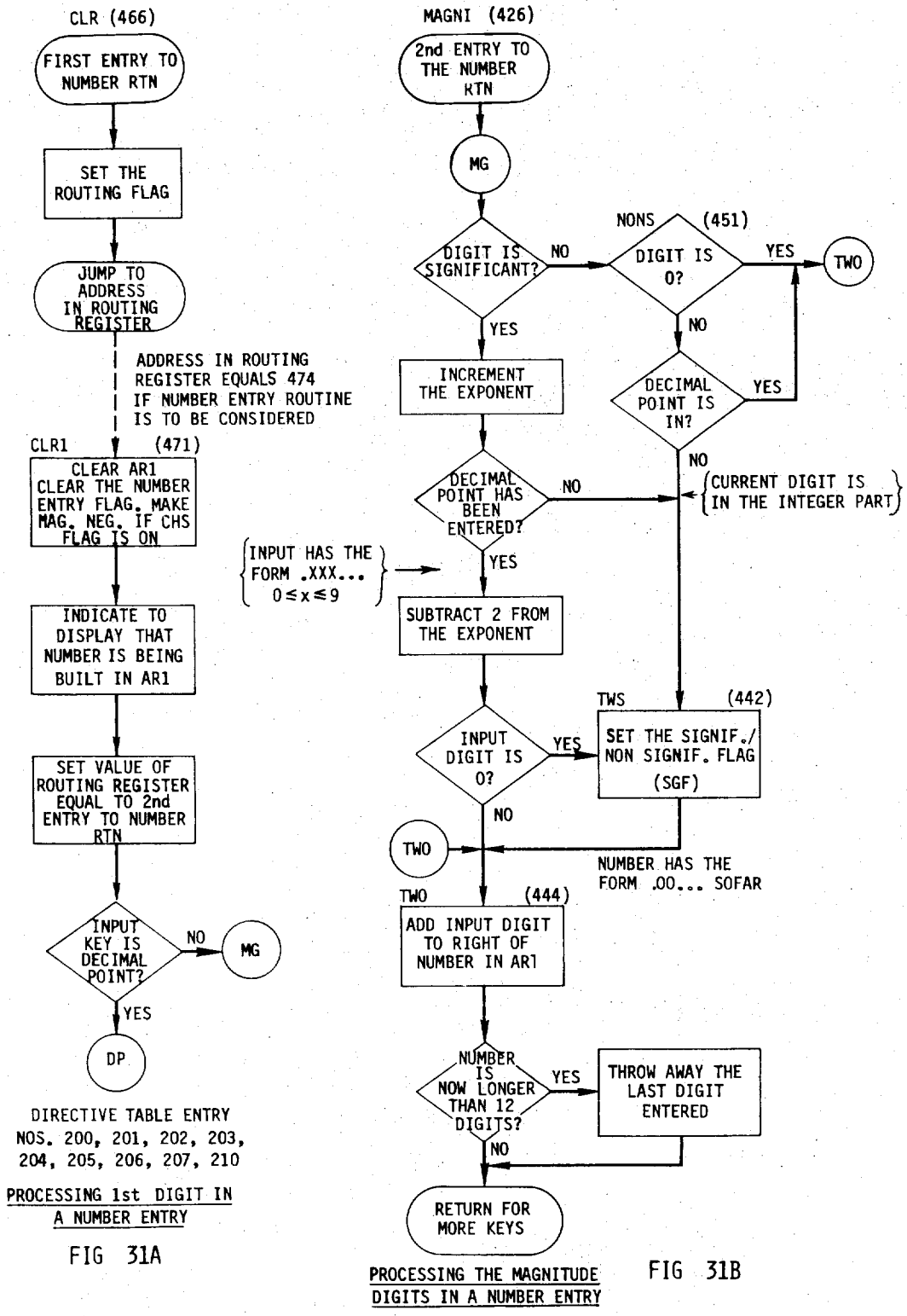
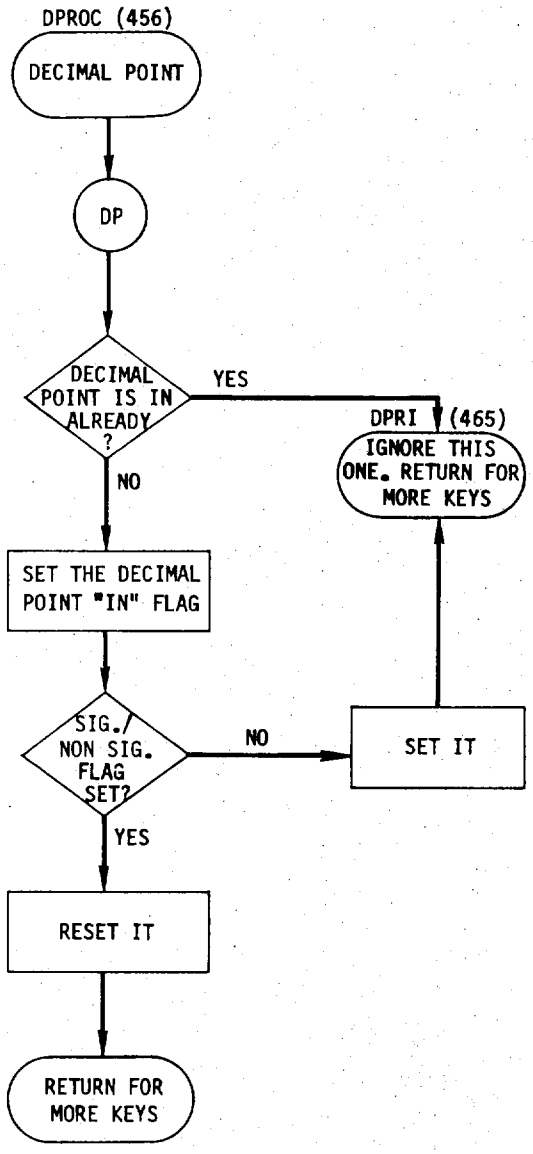


FIG 31A

FIG 31B

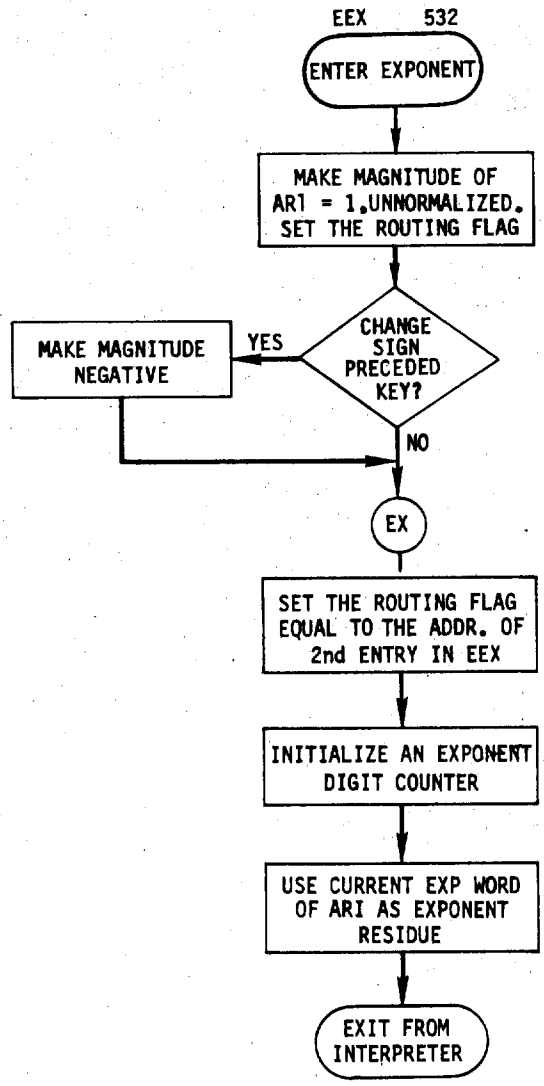


DIRECTIVE TABLE ENTRY NO. 221

DECIMAL POINT PROCESSING

(KEYCODE 21)

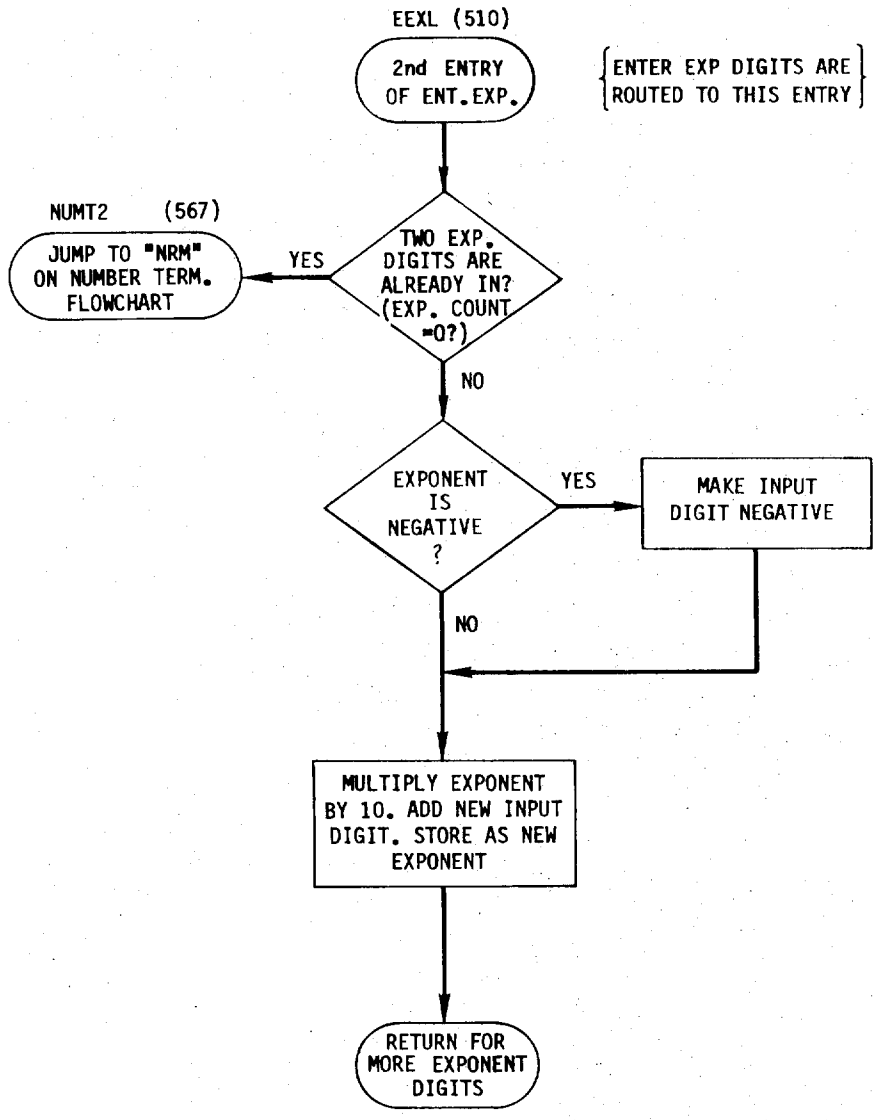
FIG 31C



DIRECTIVE TABLE ENTRY (NO. 226)

ENTER EXPONENT PROCESSING

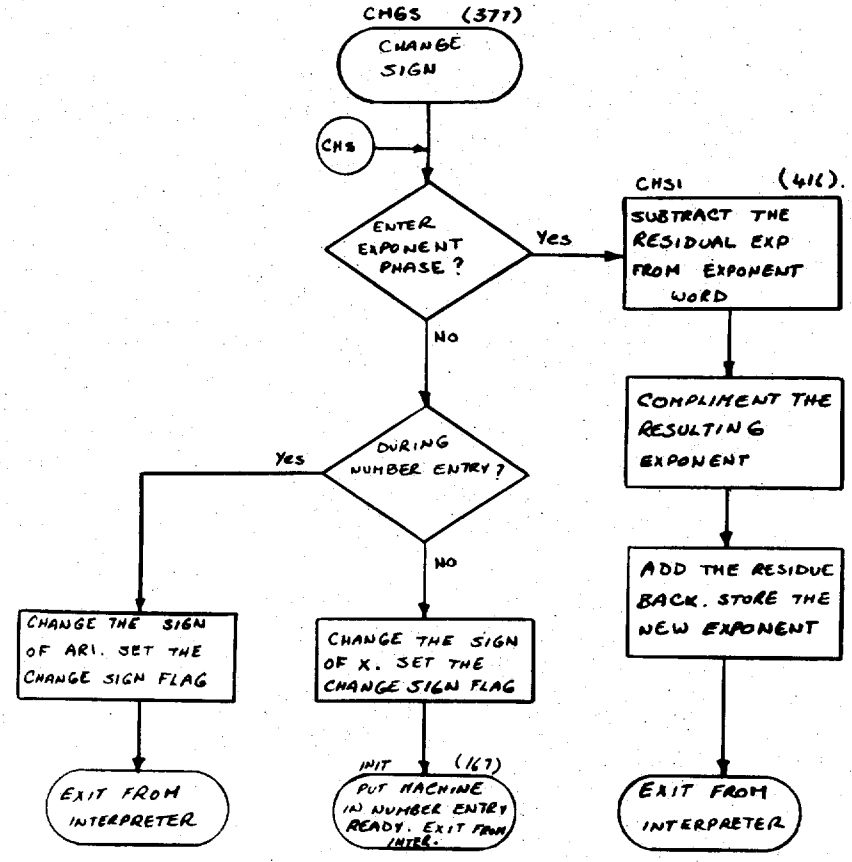
FIG 32A



ENTER EXPONENT PROCESSING

NOTE: NON NUMERIC KEYCODE FOLLOWING EEX. (OTHER THAN CHS) WILL CAUSE THE ROUTING REGISTER TO BE DECREMENTED, THUS JUMPING TO THE NUMBER ENTRY TERMINATOR INSTEAD OF EEXL.

FIG 32B



DIRECTIVE TABLE ENTRY
No. 232

CHANGE SIGN PROCESSING

FIG 33

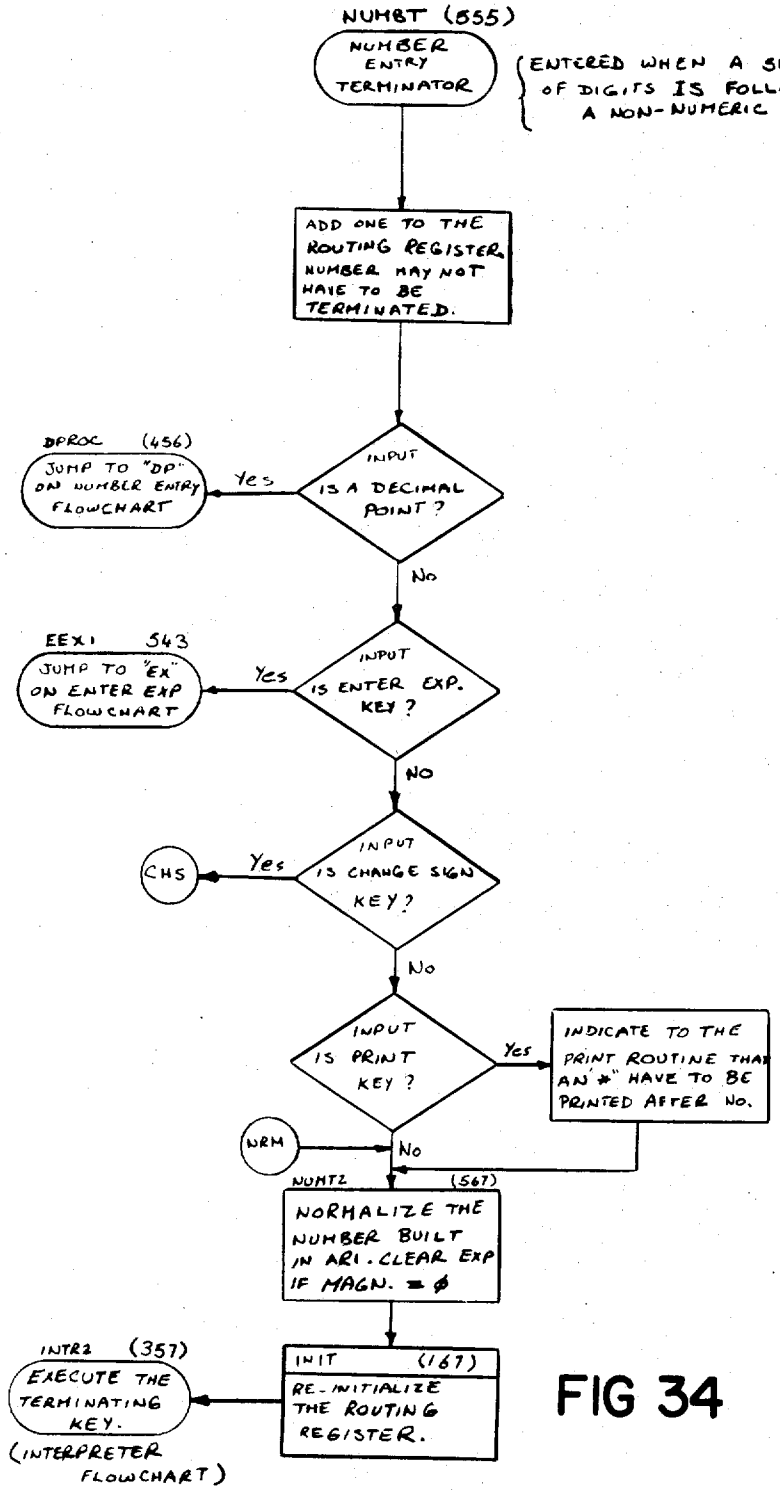
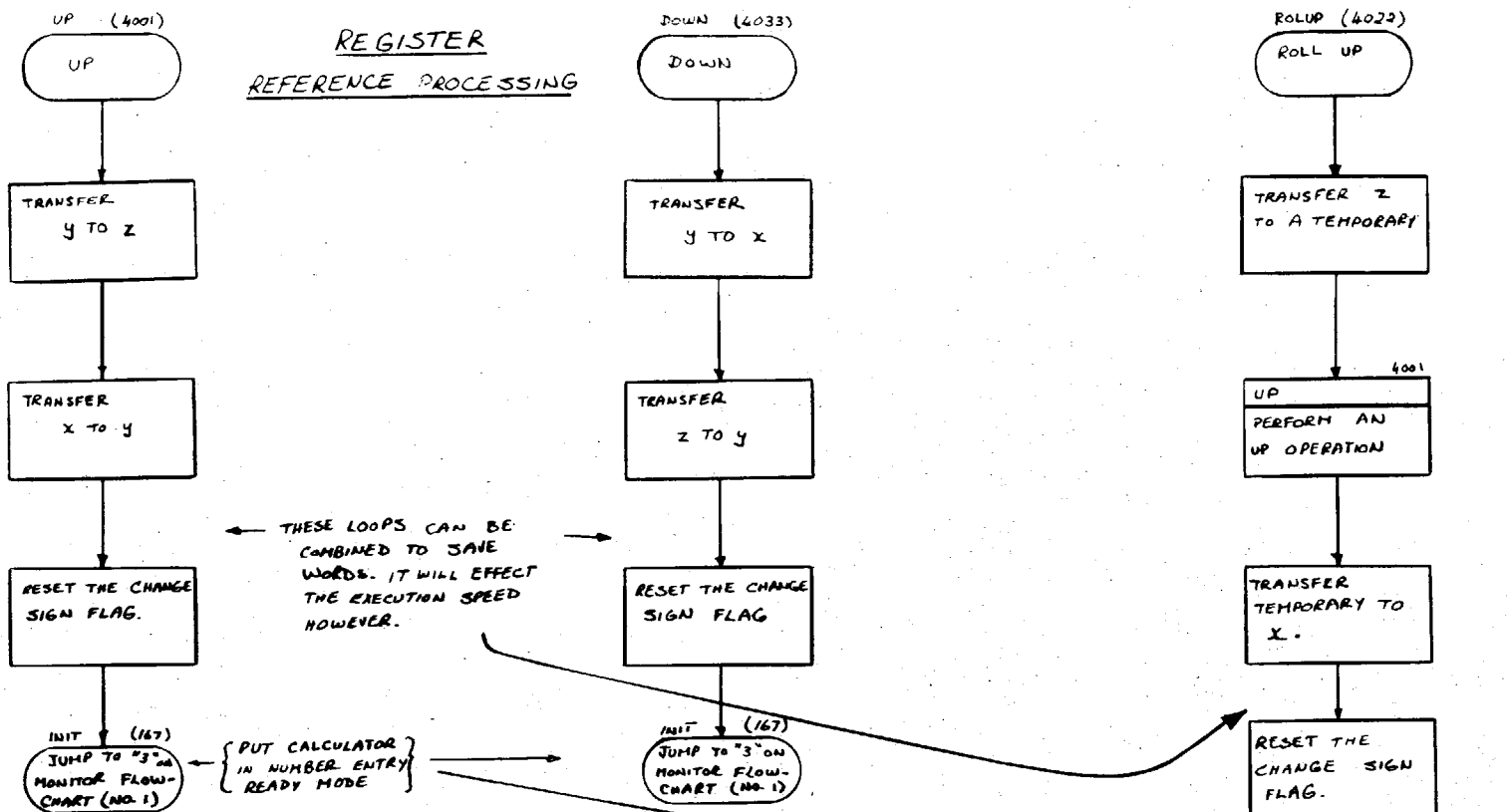


FIG 34

NUMBER ENTRY TERMINATOR



UP PROCESSING

(KEYCODE 27)

DIRECTIVE TABLE ENTRY No. 227

FIG 35

DOWN PROCESSING

(KEYCODE 25)

DIRECTIVE TABLE ENTRY NO. 225

FIG 36

ROLL-UP PROCESSING

(KEYCODE 22)

DIRECTIVE TABLE ENTRY NO. 222

FIG 37

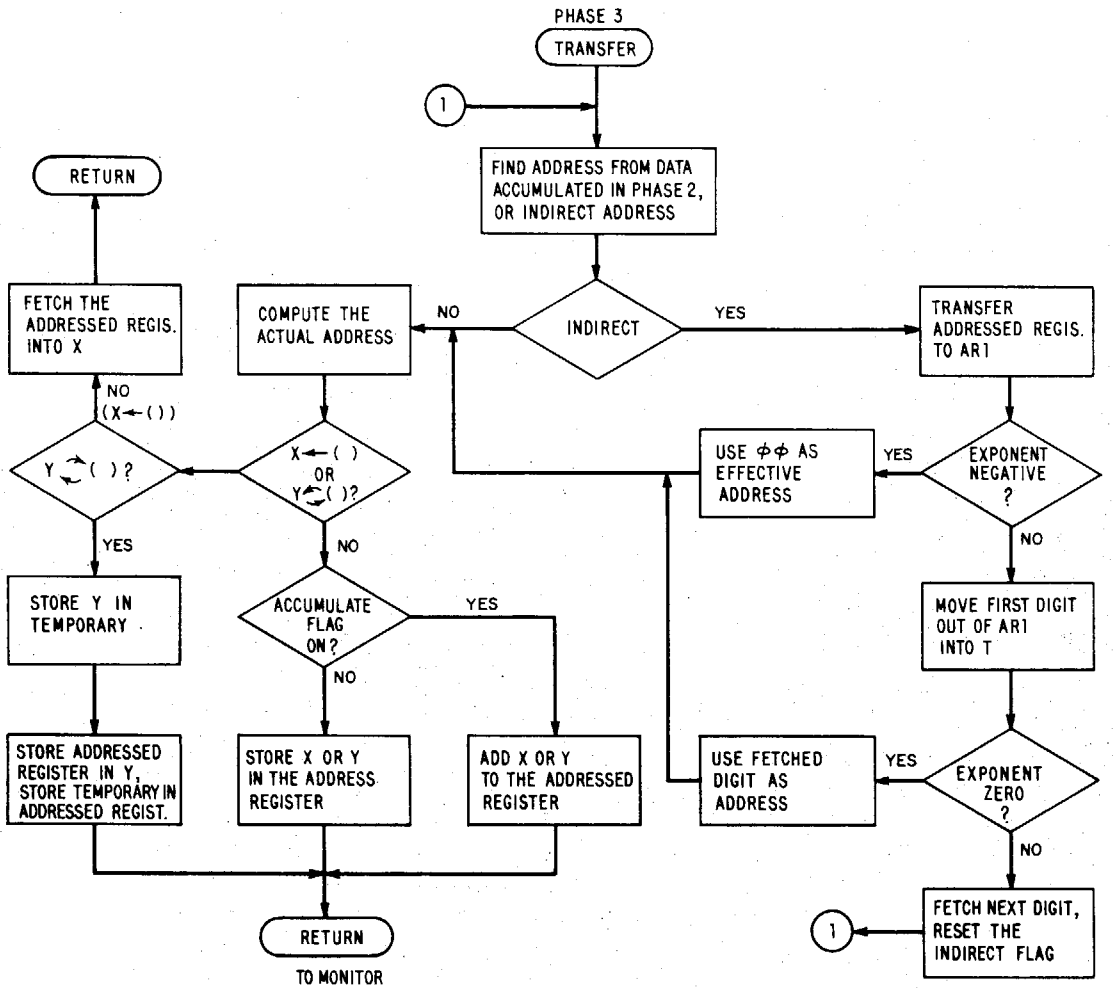


Fig 38A

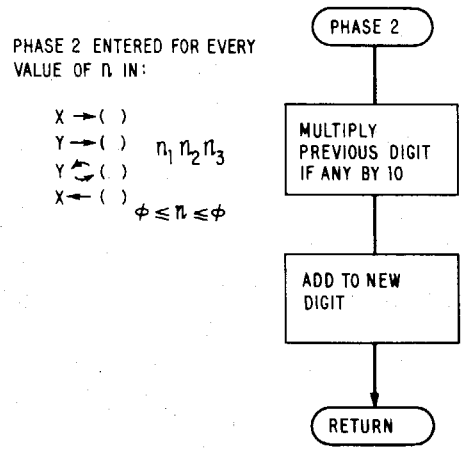


Fig 38B

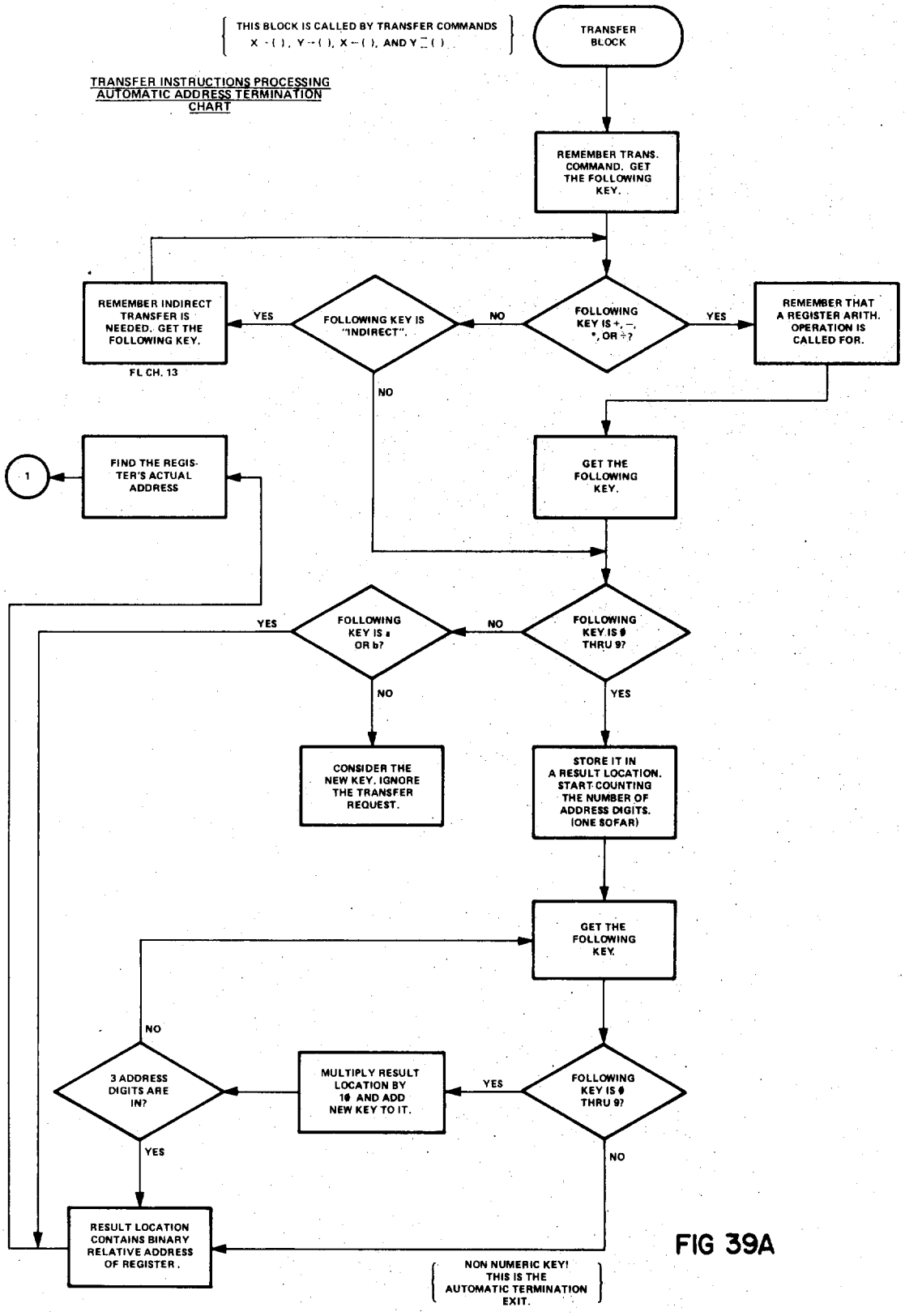

THIS BLOCK IS CALLED BY TRANSFER COMMANDS X←( ), Y←( ), X←( ), AND Y←( )
TRANSFER BLOCK
TRANSFER INSTRUCTIONS PROCESSING AUTOMATIC ADDRESS TERMINATION CHART
REMEMBER TRANS. COMMAND. GET THE FOLLOWING KEY.
REMEMBER INDIRECT TRANSFER IS NEEDED. GET THE FOLLOWING KEY.
FL CH. 13
YES
FOLLOWING KEY IS "INDIRECT".
NO
FOLLOWING KEY IS +, −, *, OR ÷?
YES
REMEMBER THAT A REGISTER ARITH. OPERATION IS CALLED FOR.
NO
1
FIND THE REGISTER'S ACTUAL ADDRESS
GET THE FOLLOWING KEY.
YES
FOLLOWING KEY IS a OR b?
NO
FOLLOWING KEY IS 0 THRU 9?
NO
YES
CONSIDER THE NEW KEY. IGNORE THE TRANSFER REQUEST.
STORE IT IN A RESULT LOCATION. START COUNTING THE NUMBER OF ADDRESS DIGITS. (ONE SOFAR)
GET THE FOLLOWING KEY.
NO
3 ADDRESS DIGITS ARE IN?
MULTIPLY RESULT LOCATION BY 10 AND ADD NEW KEY TO IT.
YES
FOLLOWING KEY IS 0 THRU 9?
YES
NO
RESULT LOCATION CONTAINS BINARY RELATIVE ADDRESS OF REGISTER.
NON NUMERIC KEY! THIS IS THE AUTOMATIC TERMINATION EXIT.

FIG 39A

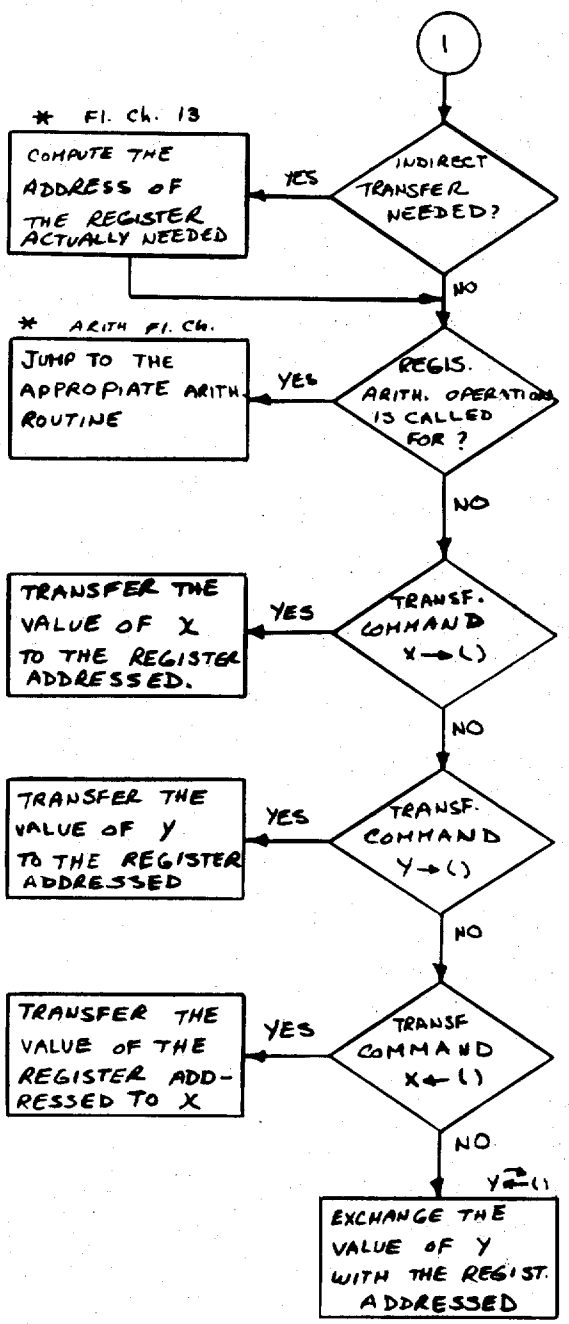
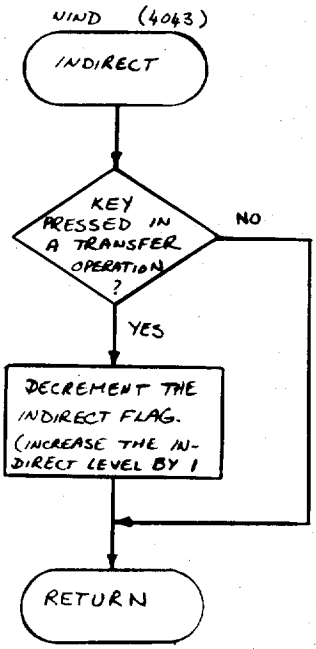


FIG 39B

* THIS OPERATION IS MORE DETAILED IN OTHER CHARTS



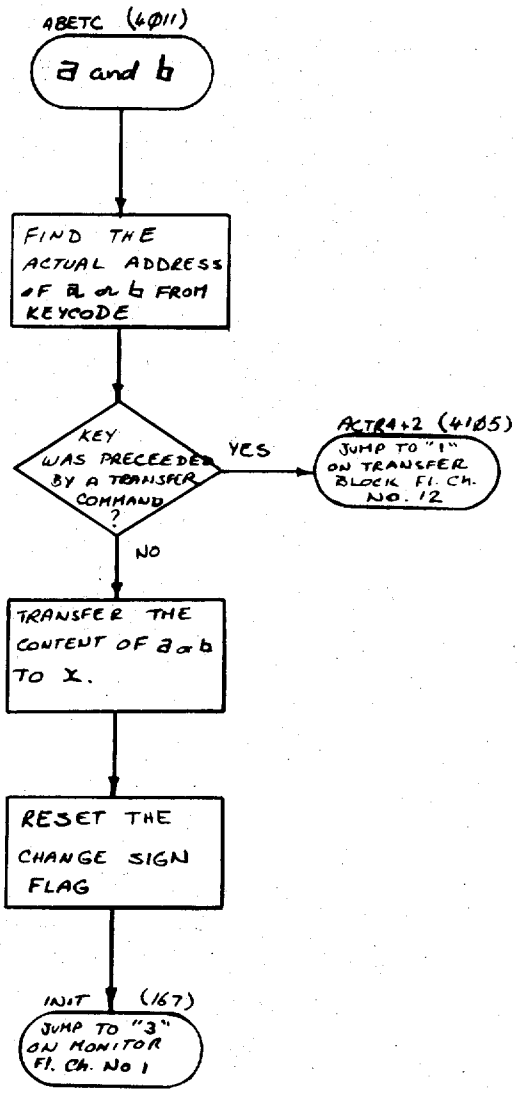
INDIRECT PROCESSING

(KEYCODE 31)

DIRECTIVE TABLE ENTRY

No. 231

FIG 40



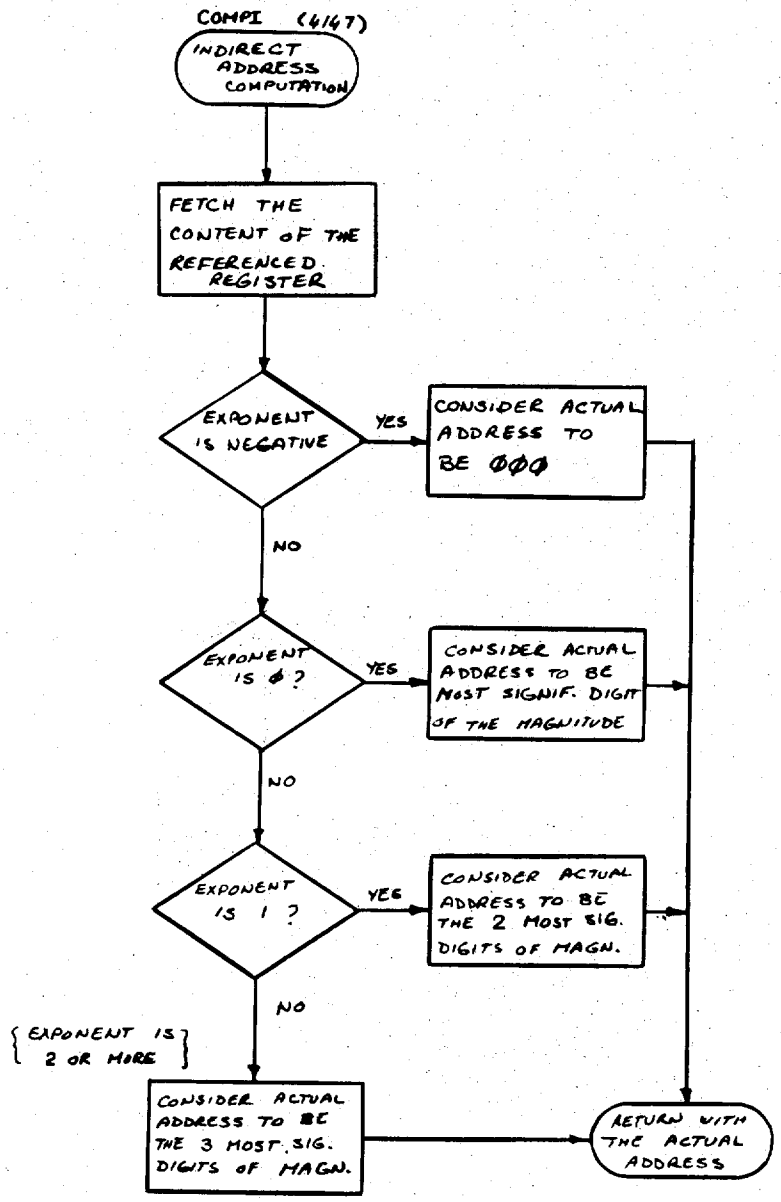
a and b PROCESSING

(KEYCODES 13, 14)

DIRECTNE TABLE ENTRY

NO. 213, 214

FIG 41



INDIRECT ADDRESS COMPUTATION

FIG 42

DECIMAL ADDRESS JUMPS, AND AUTO. ADDR. TERMINATION

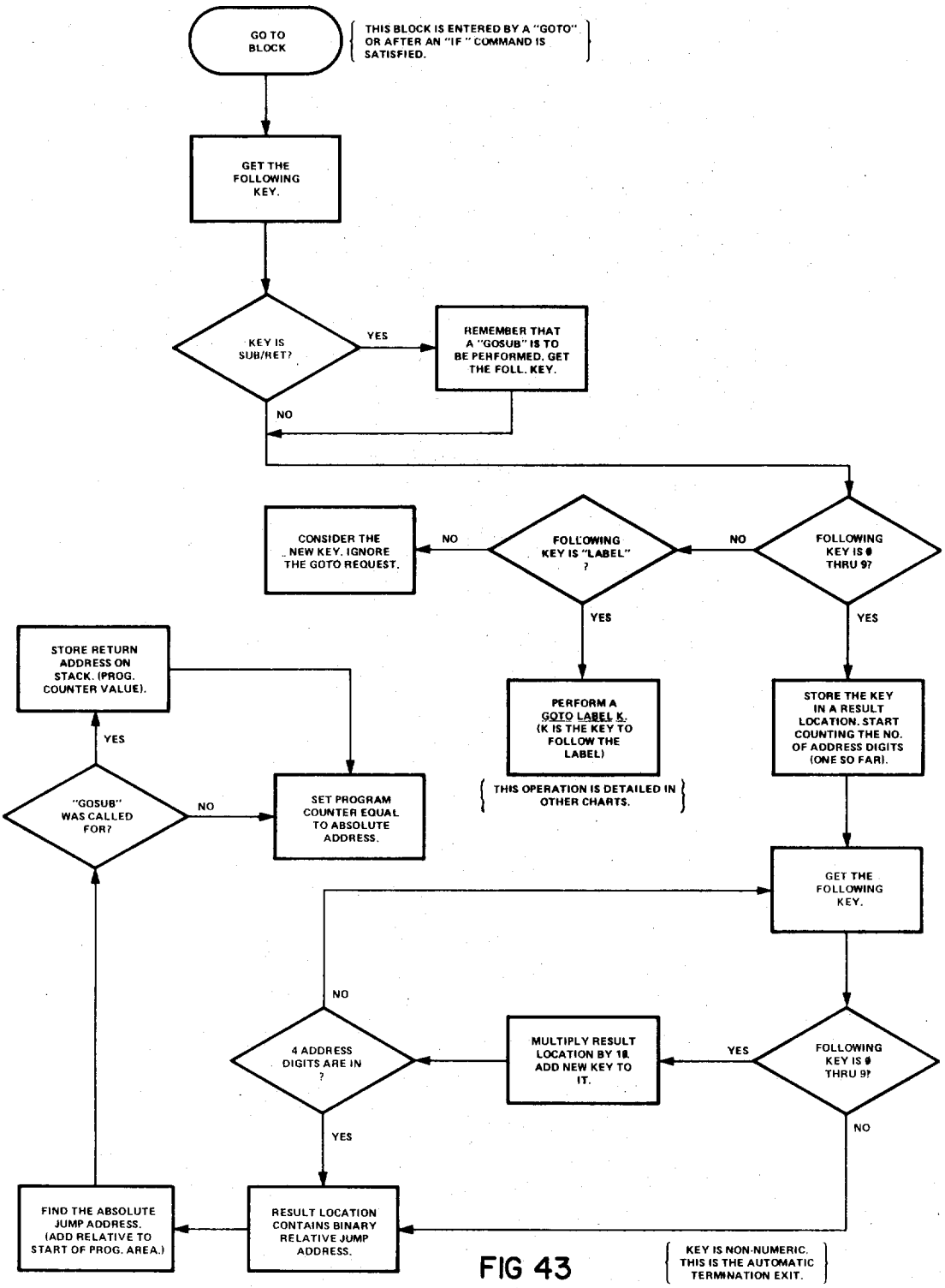


FIG 43

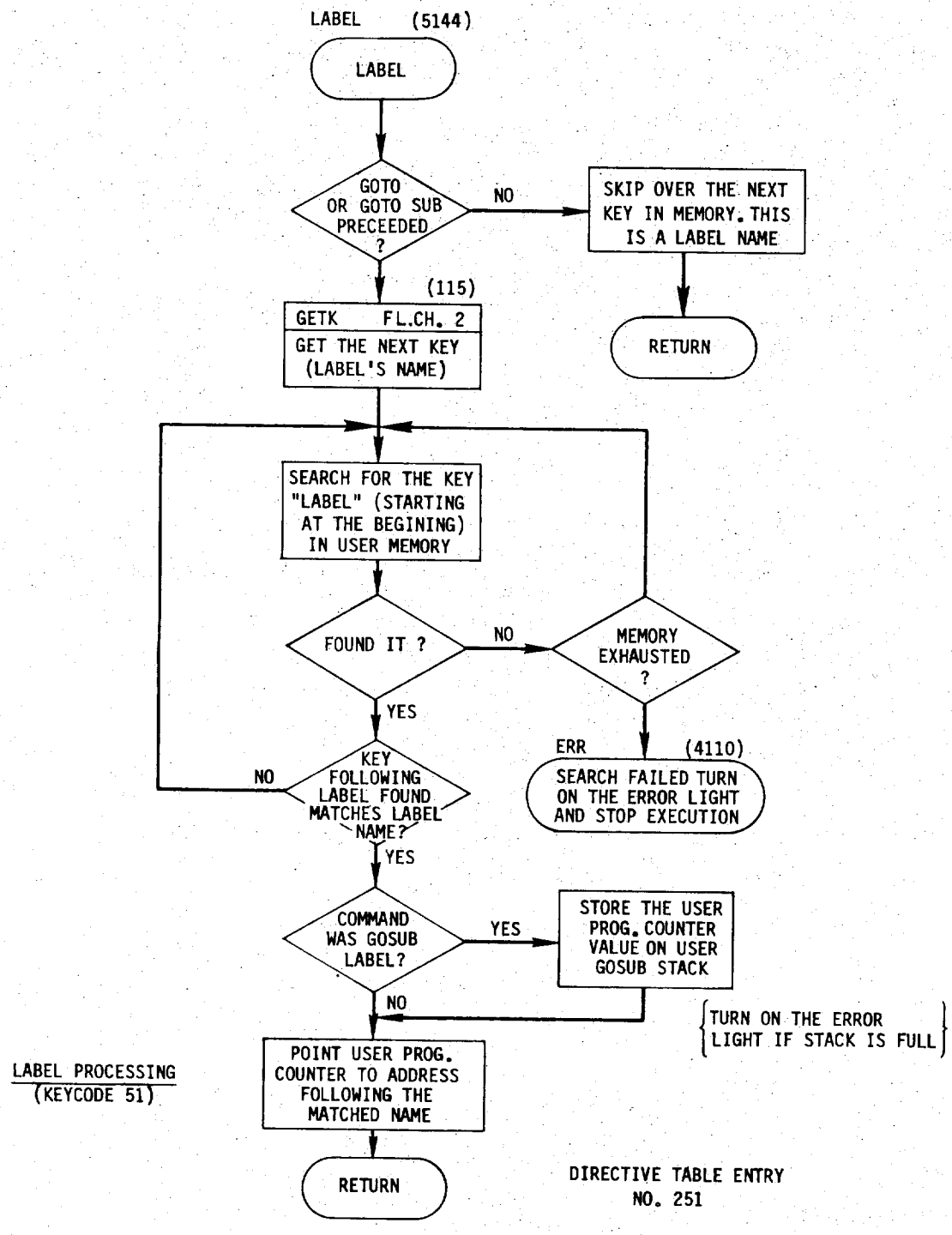

LABEL (5144)
LABEL
GOTO OR GOTO SUB PRECEEDED ?
NO
SKIP OVER THE NEXT KEY IN MEMORY. THIS IS A LABEL NAME
RETURN
(115)
GETK FL.CH. 2
GET THE NEXT KEY (LABEL'S NAME)
SEARCH FOR THE KEY "LABEL" (STARTING AT THE BEGINING) IN USER MEMORY
FOUND IT ?
NO
MEMORY EXHAUSTED ?
YES
ERR (4110)
SEARCH FAILED TURN ON THE ERROR LIGHT AND STOP EXECUTION
NO
KEY FOLLOWING LABEL FOUND MATCHES LABEL NAME?
YES
COMMAND WAS GOSUB LABEL?
YES
STORE THE USER PROG. COUNTER VALUE ON USER GOSUB STACK
NO
{TURN ON THE ERROR LIGHT IF STACK IS FULL}
POINT USER PROG. COUNTER TO ADDRESS FOLLOWING THE MATCHED NAME
RETURN
LABEL PROCESSING (KEYCODE 51)
DIRECTIVE TABLE ENTRY NO. 251

FIG 44

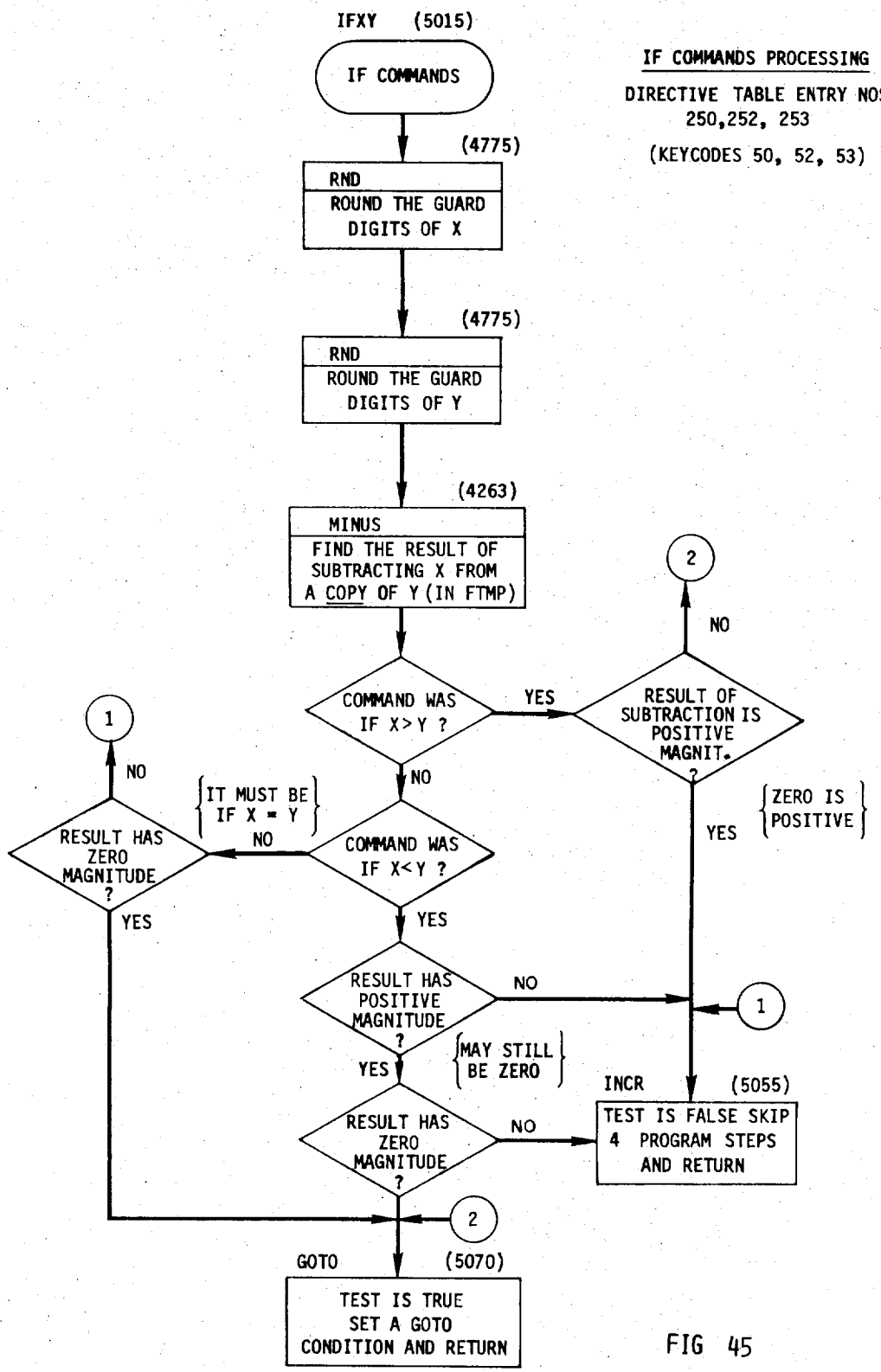
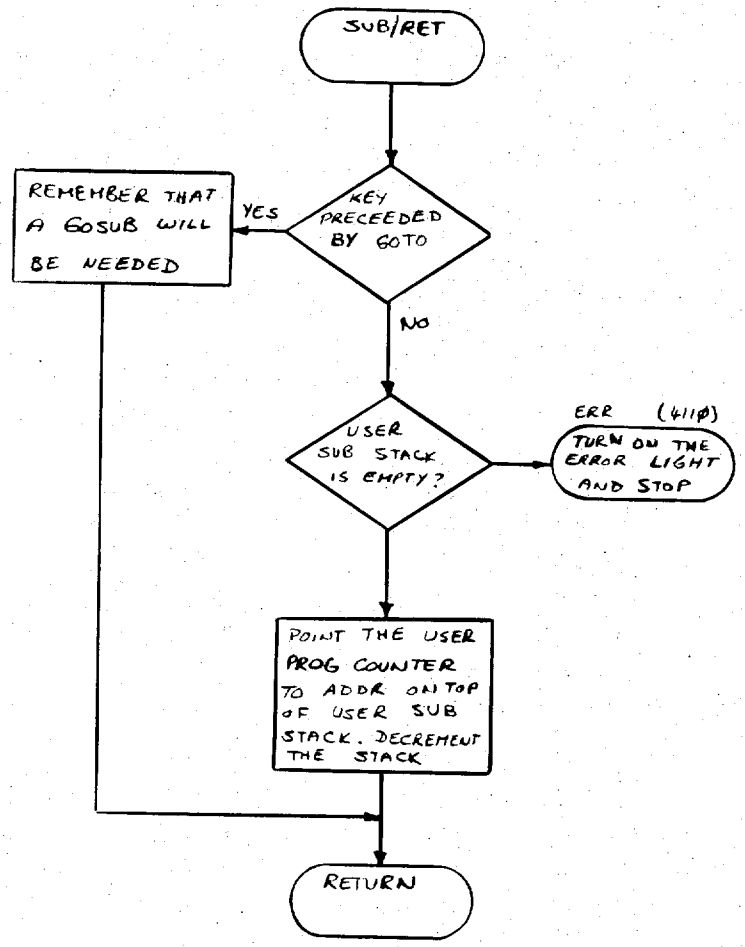

IFXY (5015)
IF COMMANDS
IF COMMANDS PROCESSING
DIRECTIVE TABLE ENTRY NOS. 250,252, 253
(KEYCODES 50, 52, 53)
(4775)
RND
ROUND THE GUARD DIGITS OF X
(4775)
RND
ROUND THE GUARD DIGITS OF Y
(4263)
MINUS
FIND THE RESULT OF SUBTRACTING X FROM A COPY OF Y (IN FTMP)
COMMAND WAS IF X>Y ?
YES
RESULT OF SUBTRACTION IS POSITIVE MAGNIT. ?
2
NO
ZERO IS POSITIVE
YES
NO
COMMAND WAS IF X<Y ?
IT MUST BE IF X = Y
NO
RESULT HAS ZERO MAGNITUDE ?
1
NO
YES
YES
RESULT HAS POSITIVE MAGNITUDE ?
NO
1
MAY STILL BE ZERO
YES
INCR (5055)
RESULT HAS ZERO MAGNITUDE ?
NO
TEST IS FALSE SKIP 4 PROGRAM STEPS AND RETURN
2
GOTO (5070)
TEST IS TRUE SET A GOTO CONDITION AND RETURN

FIG 45



SUB/RET PROCESSING

DIRECTIVE TABLE ENTRY
No. 277
(KEYCODE 77)

FIG 46

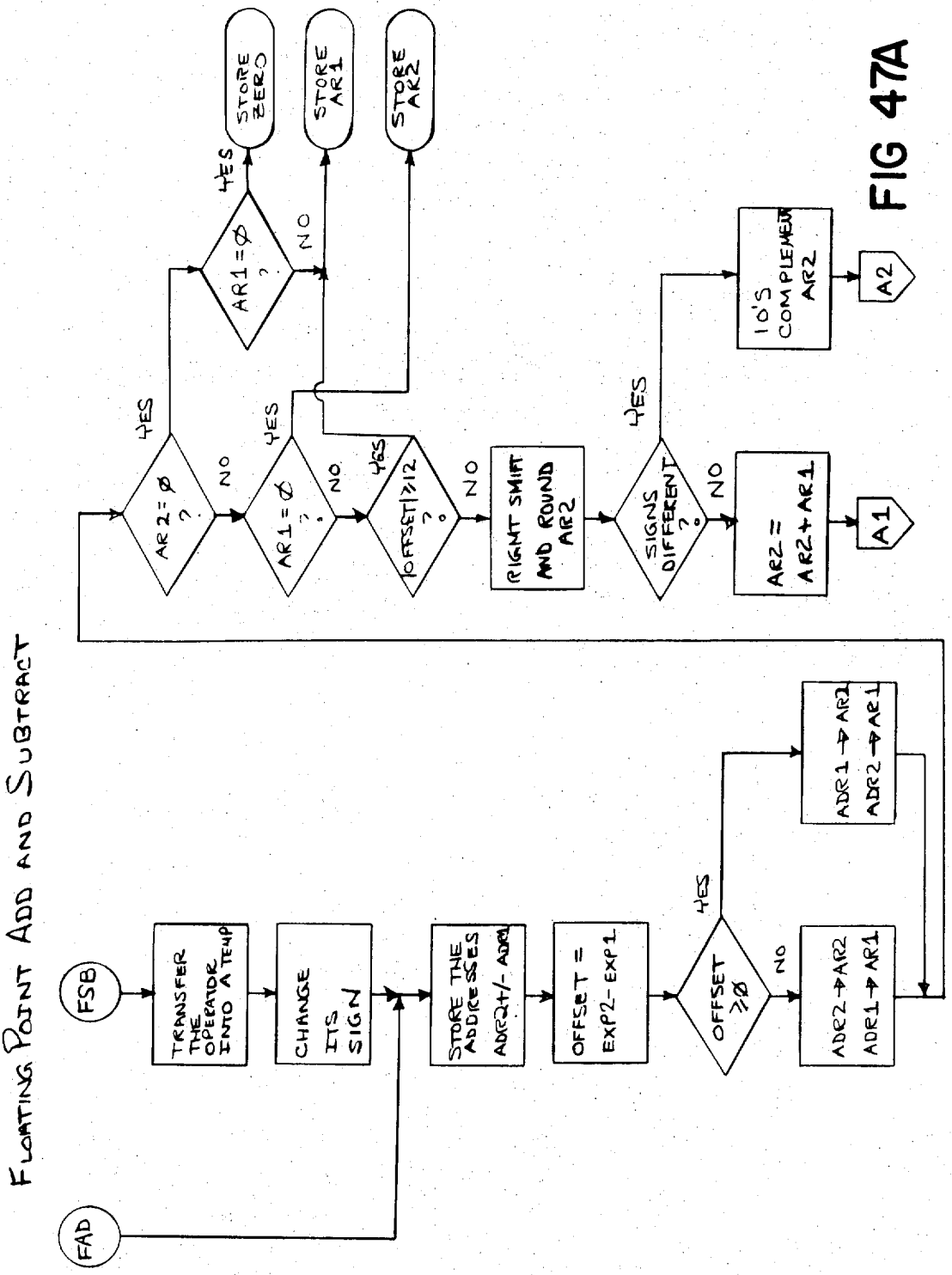

FLOATING POINT ADD AND SUBTRACT
FSB
FAD
TRANSFER THE OPERATOR INTO A TEMP
CHANGE ITS SIGN
STORE THE ADDRESSES ADR2+/- ADR1
OFFSET = EXP2 - EXP1
OFFSET ≥ Ø
YES
NO
ADR1 → AR2
ADR2 → AR1
ADR2 → AR2
ADR1 → AR1
SIGNS DIFFERENT ?
YES
NO
10'S COMPLEMENT AR2
A2
AR2 = AR2 + AR1
A1
RIGHT SHIFT AND ROUND AR2
|OFFSET| ≥ 12 ?
YES
NO
AR1 = Ø ?
YES
NO
STORE AR2
AR1 = Ø ?
YES
NO
STORE AR1
STORE ZERO
AR2 = Ø ?
YES
NO

FIG 47A

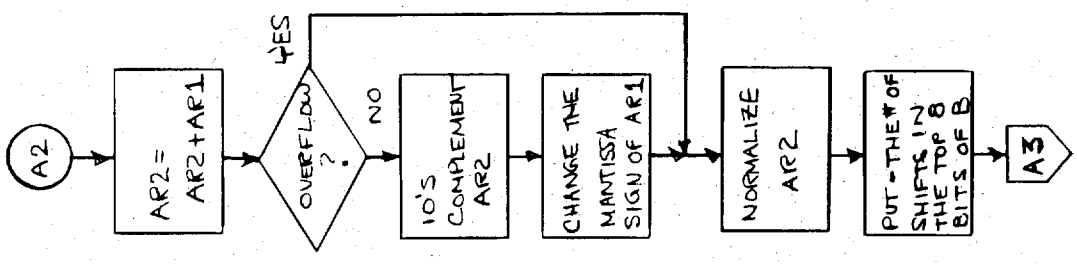


FIG 47C

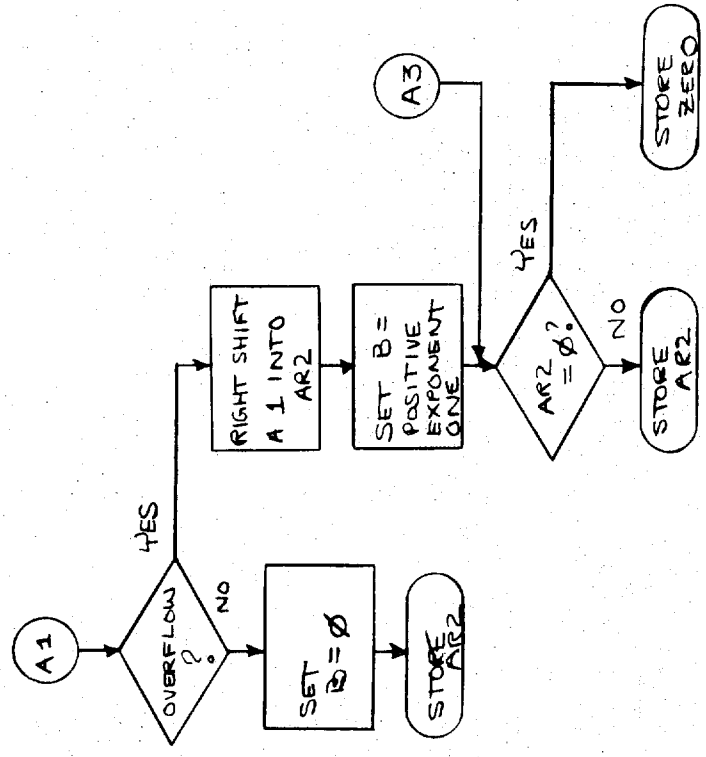


FIG 47B

FLOATING POINT MULTIPLY

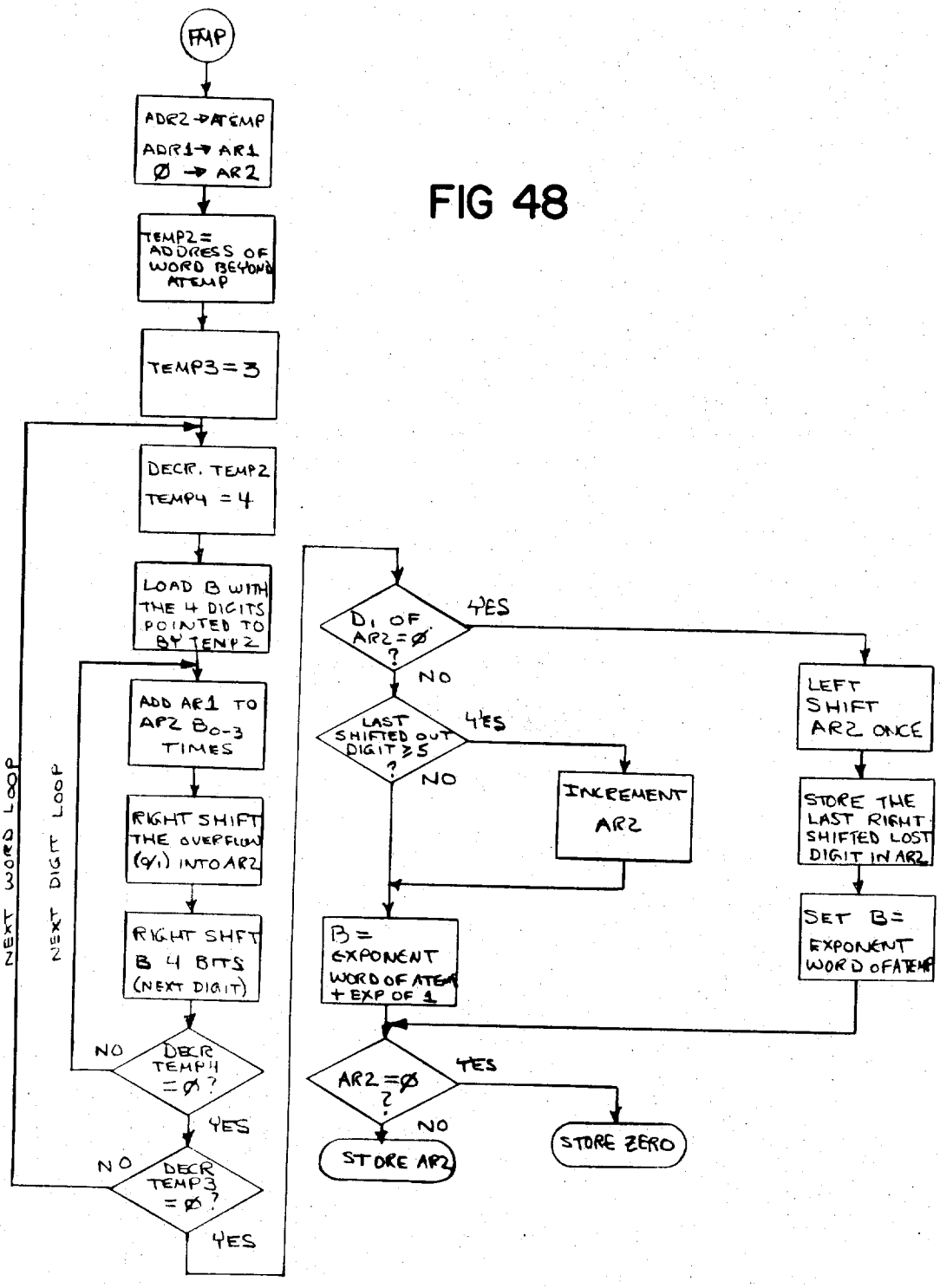


FIG 48

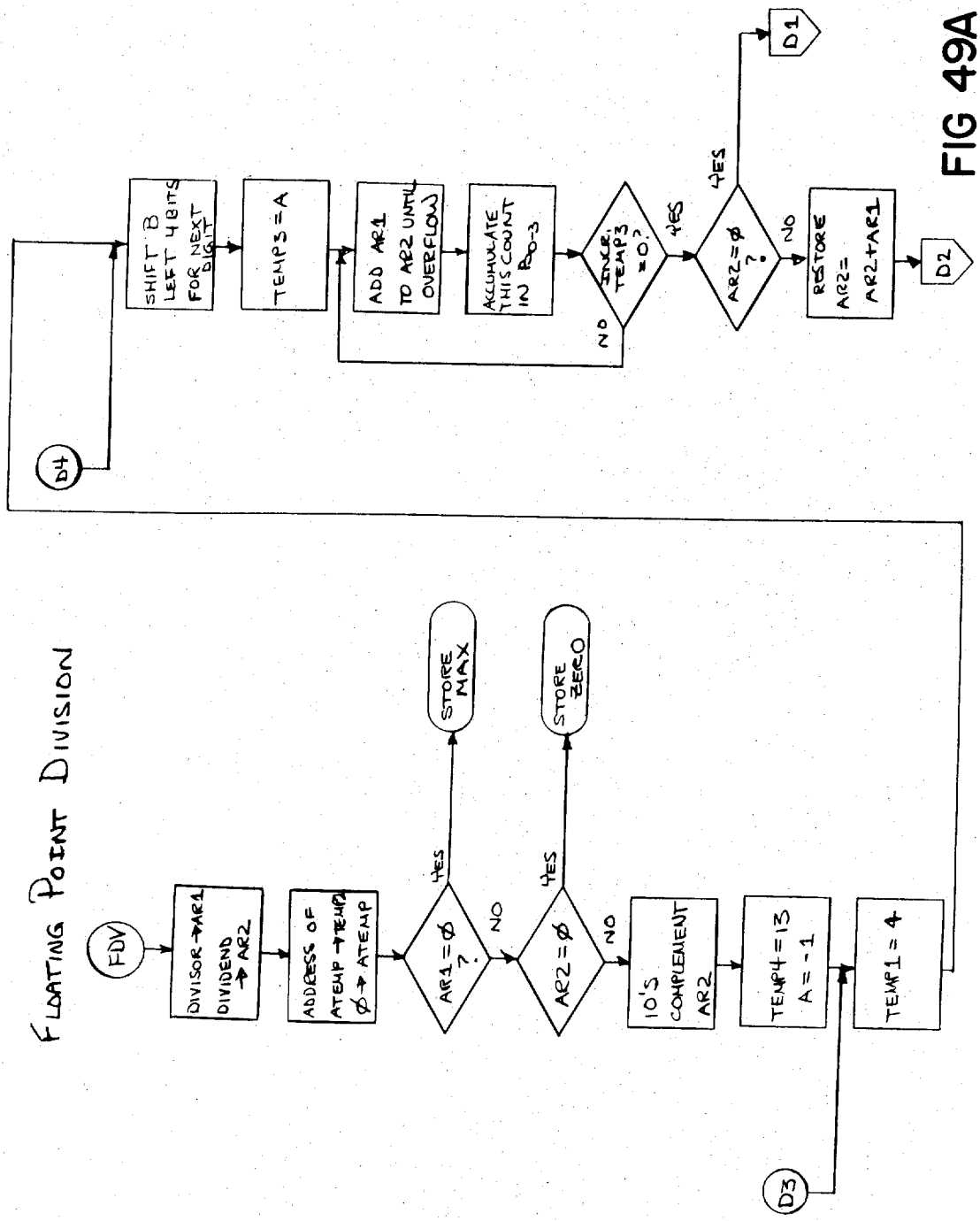

FLOATING POINT DIVISION
FDV
DIVISOR → AR1
DIVIDEND → AR2
ADDRESS OF ATEMP → TEMP
Ø → ATEMP
AR1 = Ø ?
YES
STORE MAX
NO
AR2 = Ø ?
YES
STORE ZERO
NO
10'S COMPLEMENT AR2
TEMP4 = 13
A = -1
TEMP1 = 4
D3
D4
SHIFT B LEFT 4 BITS FOR NEXT DIGIT
TEMP3 = A
ADD AR1 TO AR2 UNTIL OVERFLOW
ACCUMULATE THIS COUNT IN R0-3
INCR. TEMP3 = 0 ?
YES
NO
AR2 = Ø ?
YES
D1
NO
RESTORE AR2 = AR2 + AR1
D2

FIG 49A

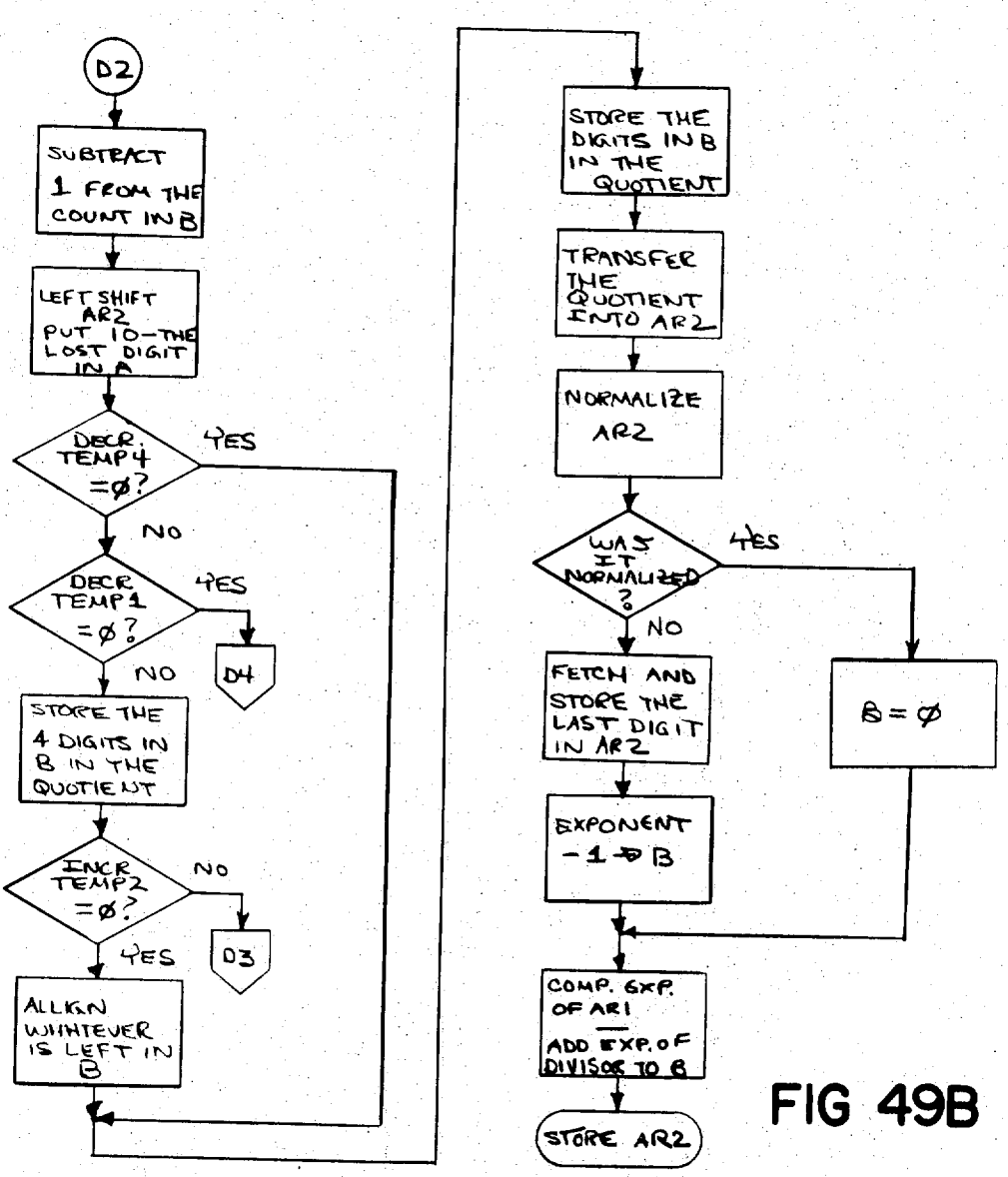

D2
SUBTRACT 1 FROM THE COUNT IN B
LEFT SHIFT AR2 PUT IO-THE LOST DIGIT IN A
DECR TEMP4 =ø?
YES
NO
DECR TEMP1 =ø?
YES
D4
NO
STORE THE 4 DIGITS IN B IN THE QUOTIENT
INCR TEMP2 =ø?
NO
D3
YES
ALLIGN WHITEVER IS LEFT IN B
STORE THE DIGITS IN B IN THE QUOTIENT
TRANSFER THE QUOTIENT INTO AR2
NORMALIZE AR2
WAS IT NORMALIZED ?
YES
NO
FETCH AND STORE THE LAST DIGIT IN AR2
B = ø
EXPONENT -1 → B
COMP. EXP. OF AR1
ADD EXP. OF DIVISOR TO B
STORE AR2

FIG 49B

FLOATING POINT SQUARE ROOT

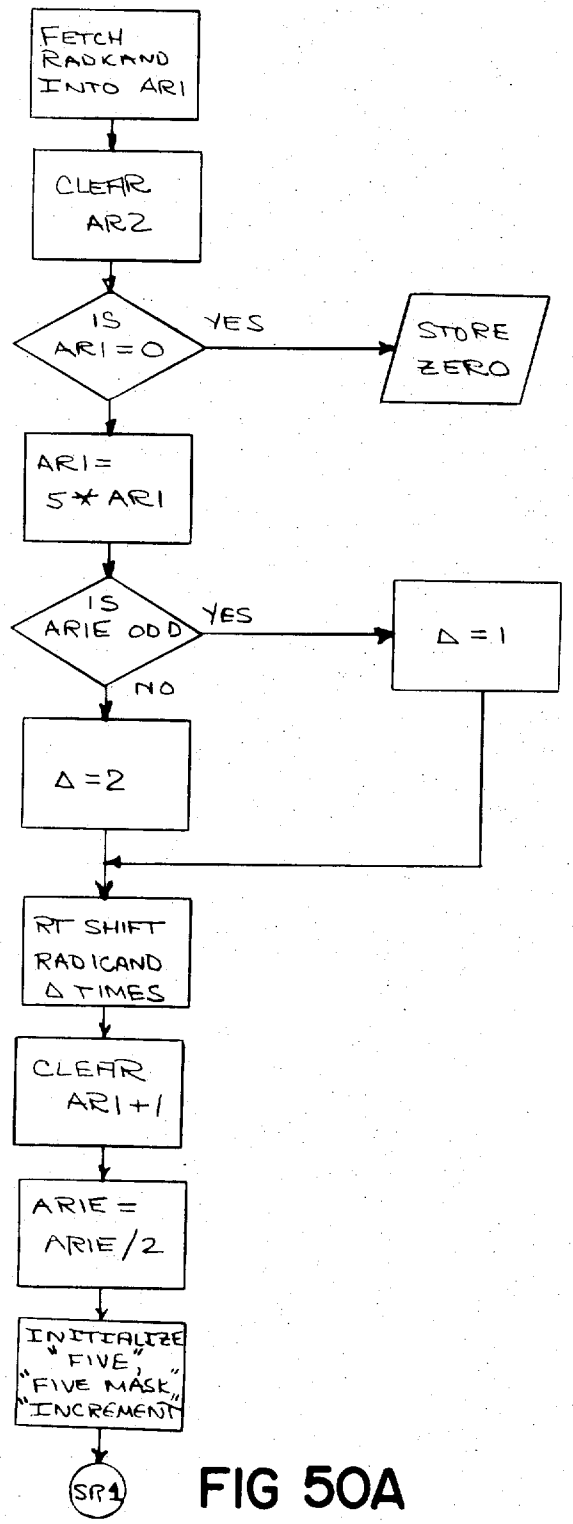


FIG 50A

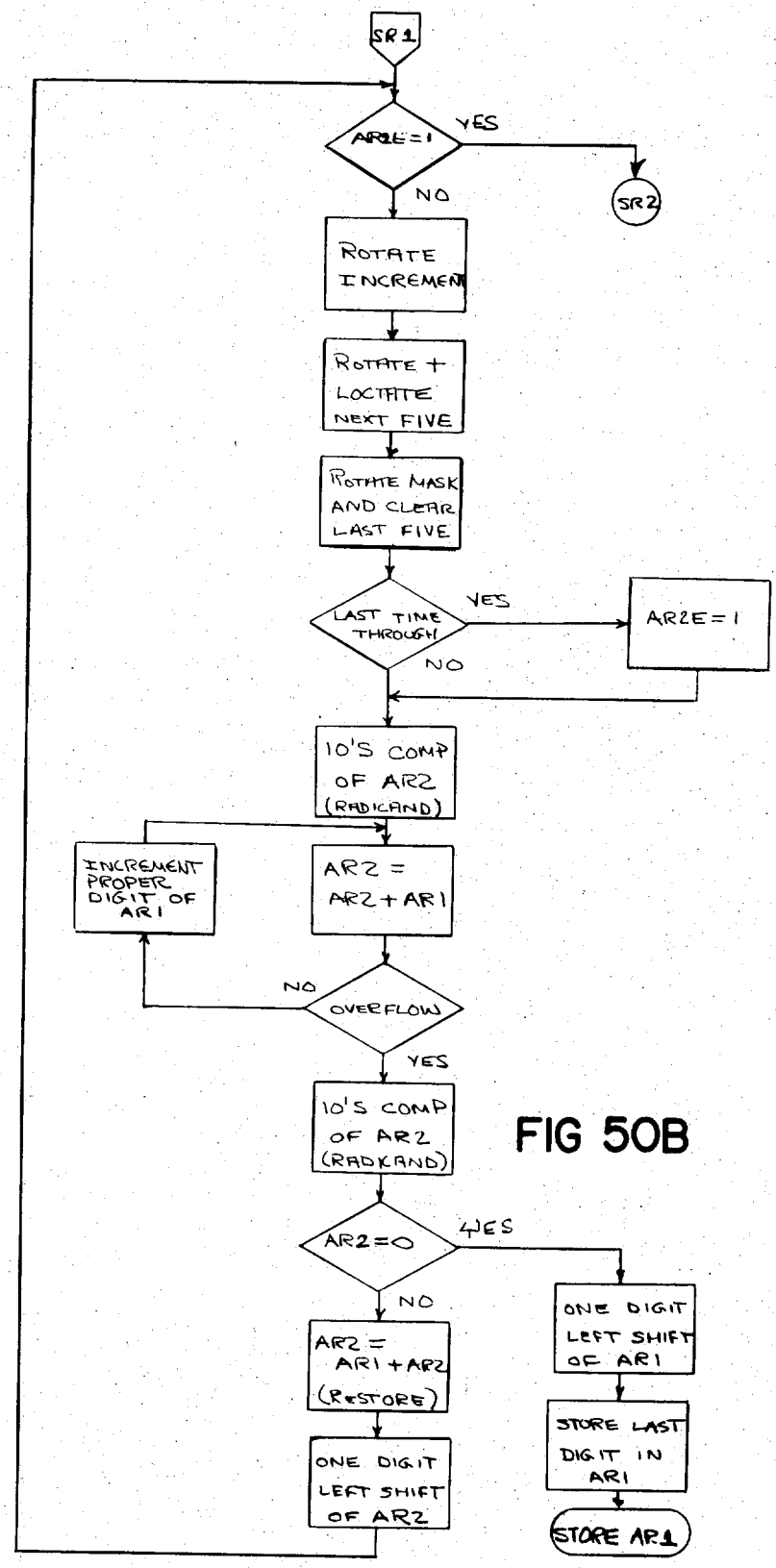

SR1
AR2E=1
YES
NO
SR2
ROTATE INCREMENT
ROTATE + LOCTATE NEXT FIVE
ROTATE MASK AND CLEAR LAST FIVE
LAST TIME THROUGH
YES
NO
AR2E=1
10'S COMP OF AR2 (RADICAND)
INCREMENT PROPER DIGIT OF AR1
AR2 = AR2+AR1
NO
OVERFLOW
YES
10'S COMP OF AR2 (RADKAND)
AR2=0
YES
NO
ONE DIGIT LEFT SHIFT OF AR1
AR2 = AR1+AR2 (RESTORE)
ONE DIGIT LEFT SHIFT OF AR2
STORE LAST DIGIT IN AR1
STORE AR1

FIG 50B

ROUND ROUTINE

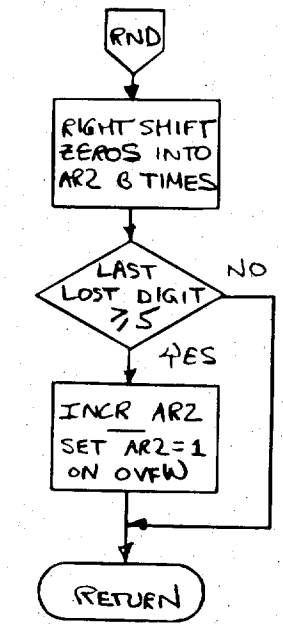


FIG 52

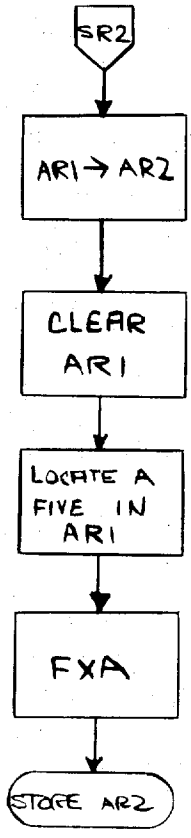


FIG 50C

X SQUARED

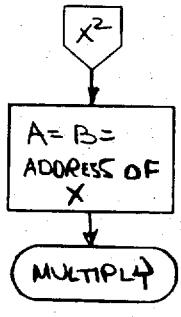


FIG 53

RECIPROCAL X

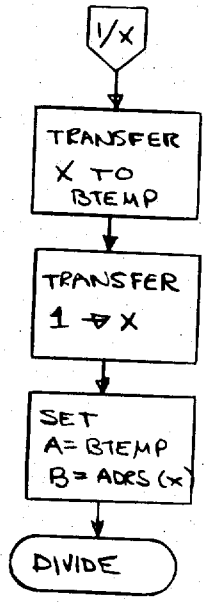
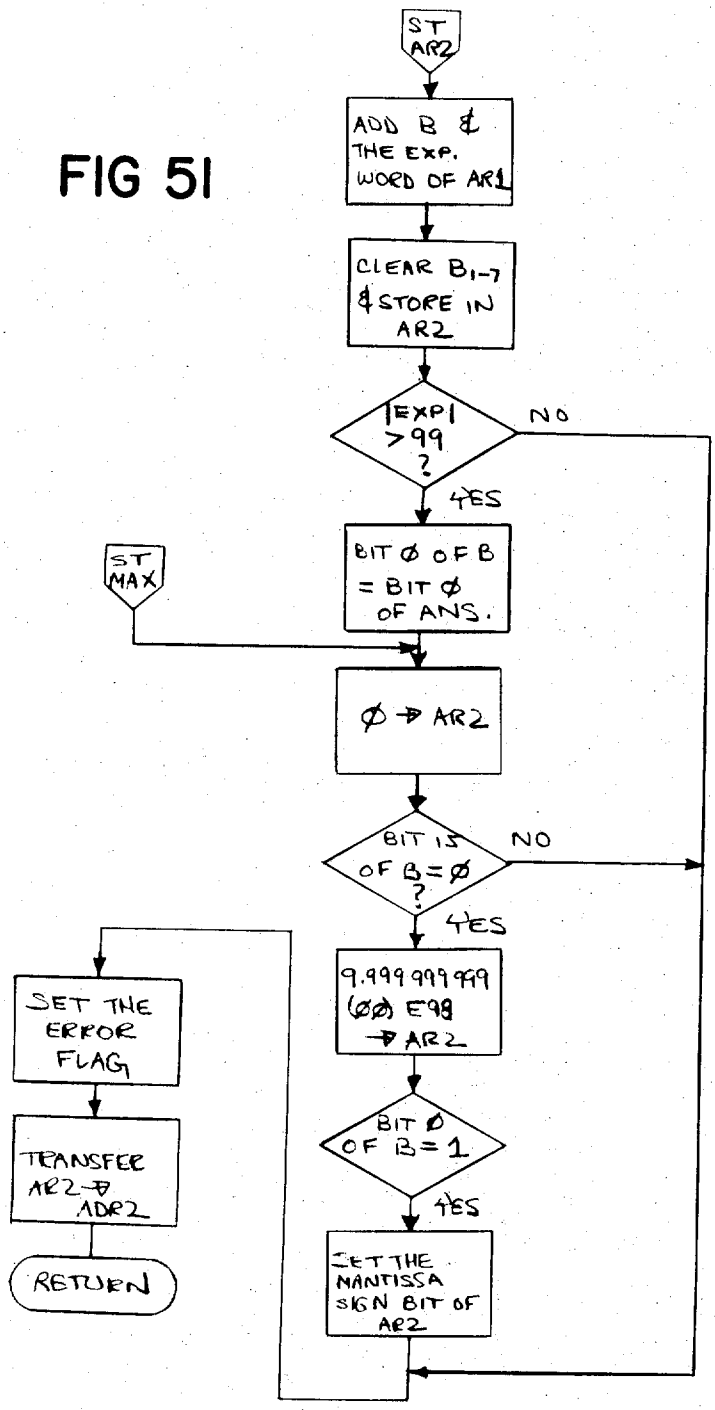


FIG 54

STORE ROUTINE

FIG 5I



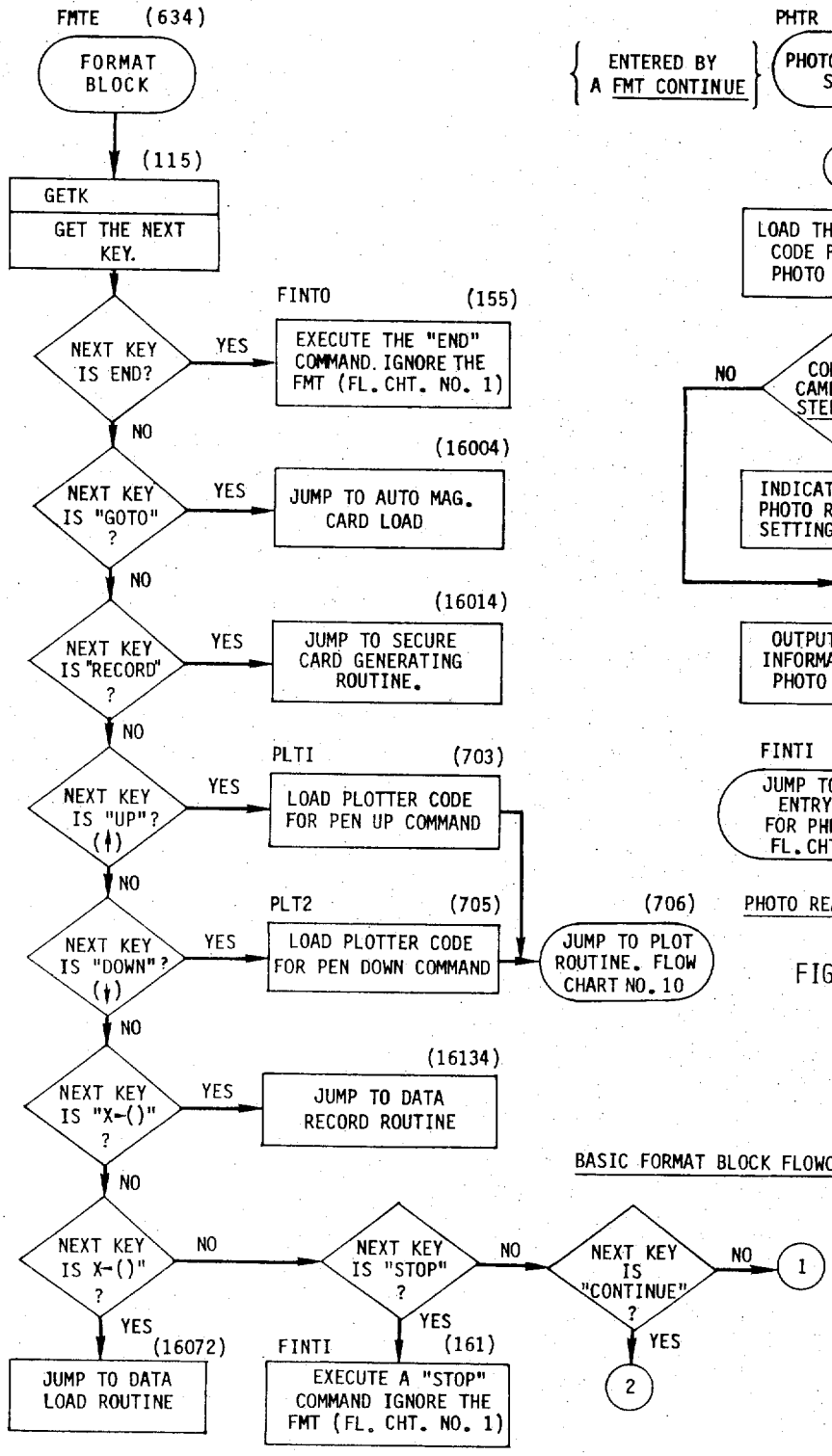


FIG 55A

FIG 55B

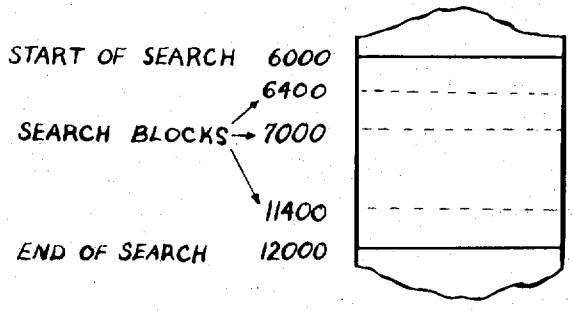
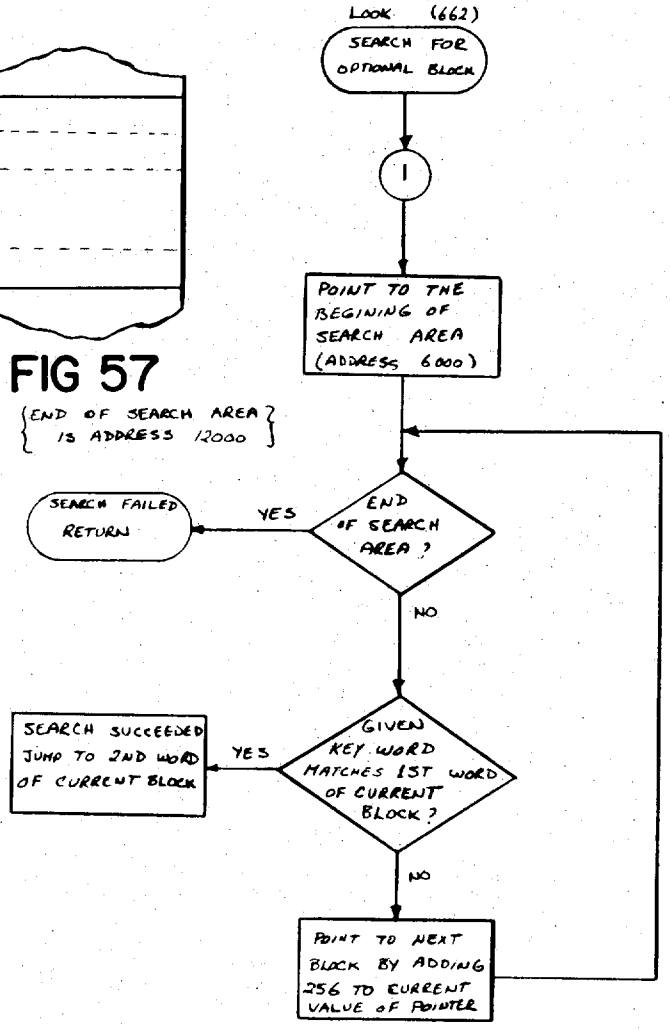


FIG 57

{END OF SEARCH AREA IS ADDRESS 12000}



OPTIONAL BLOCK SEARCH ROUTINE

FIG 56

BASIC FORMAT BLOCK FLOWCHART – CONT.
PLOTTER ROUTINE

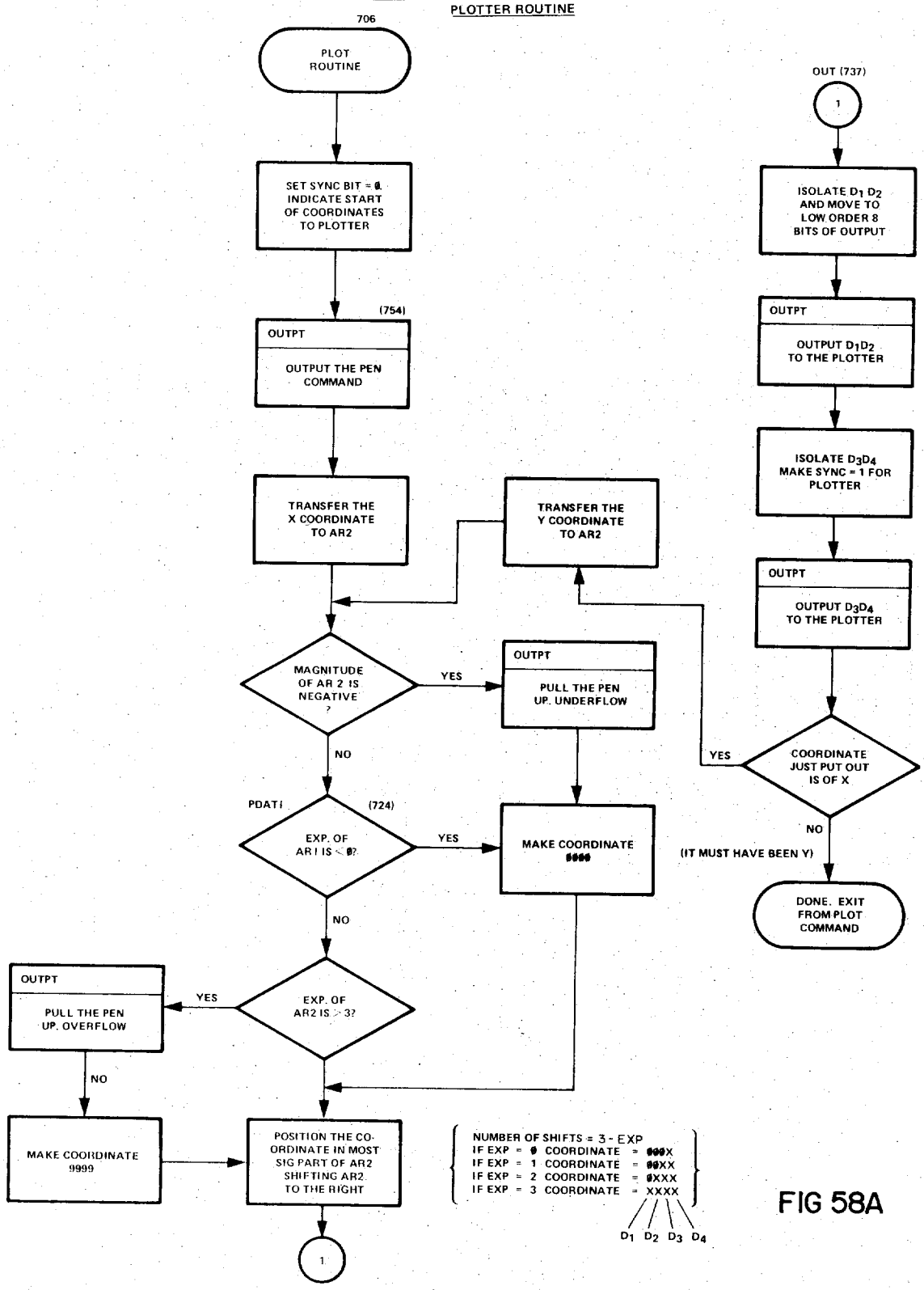


FIG 58A

PLOTTER OUTPUT ROUTINE

1755

D_1	D_2	D_3	D_4

FIG 59

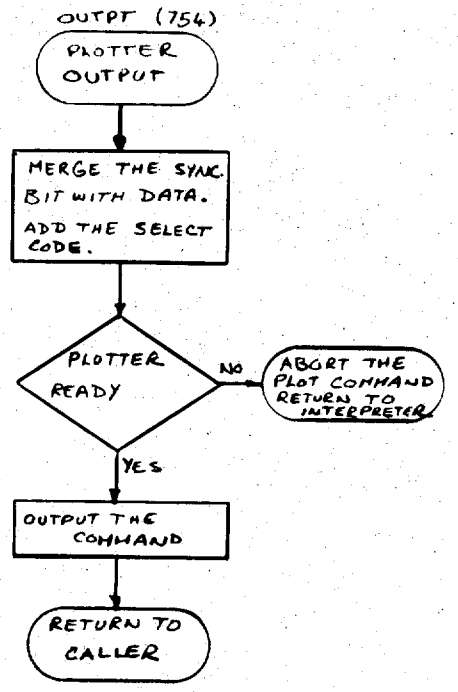
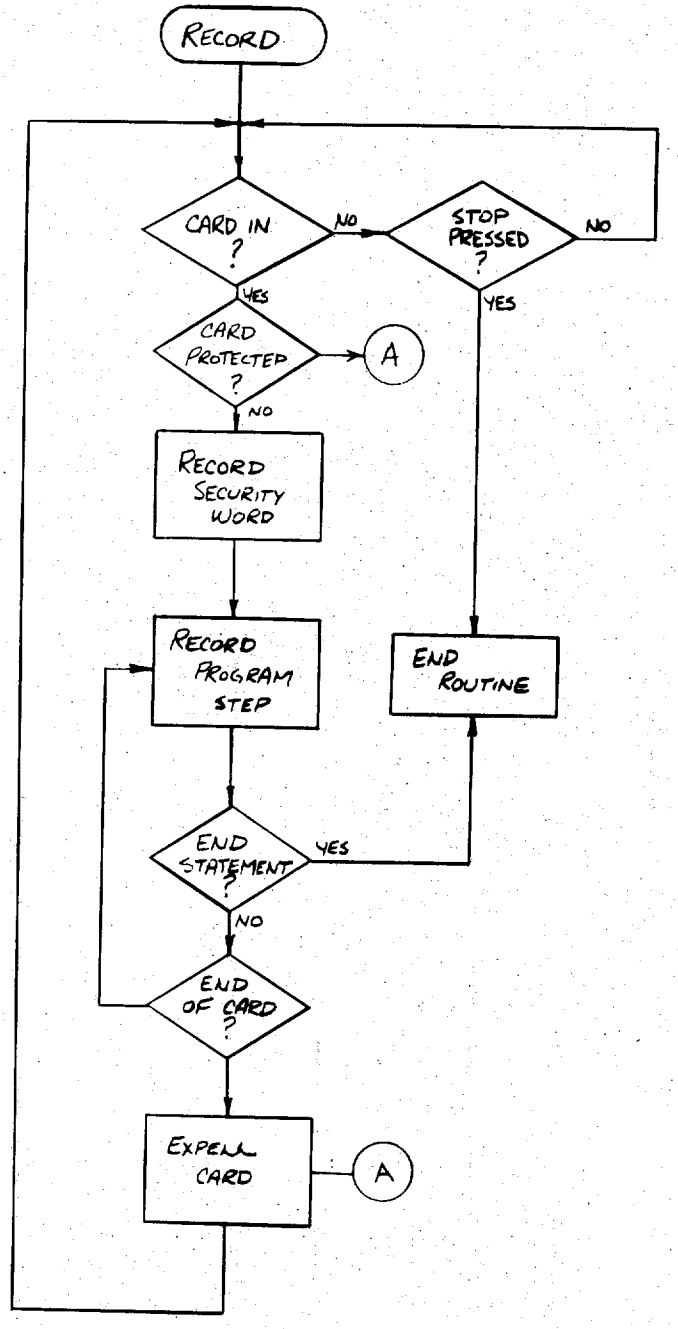


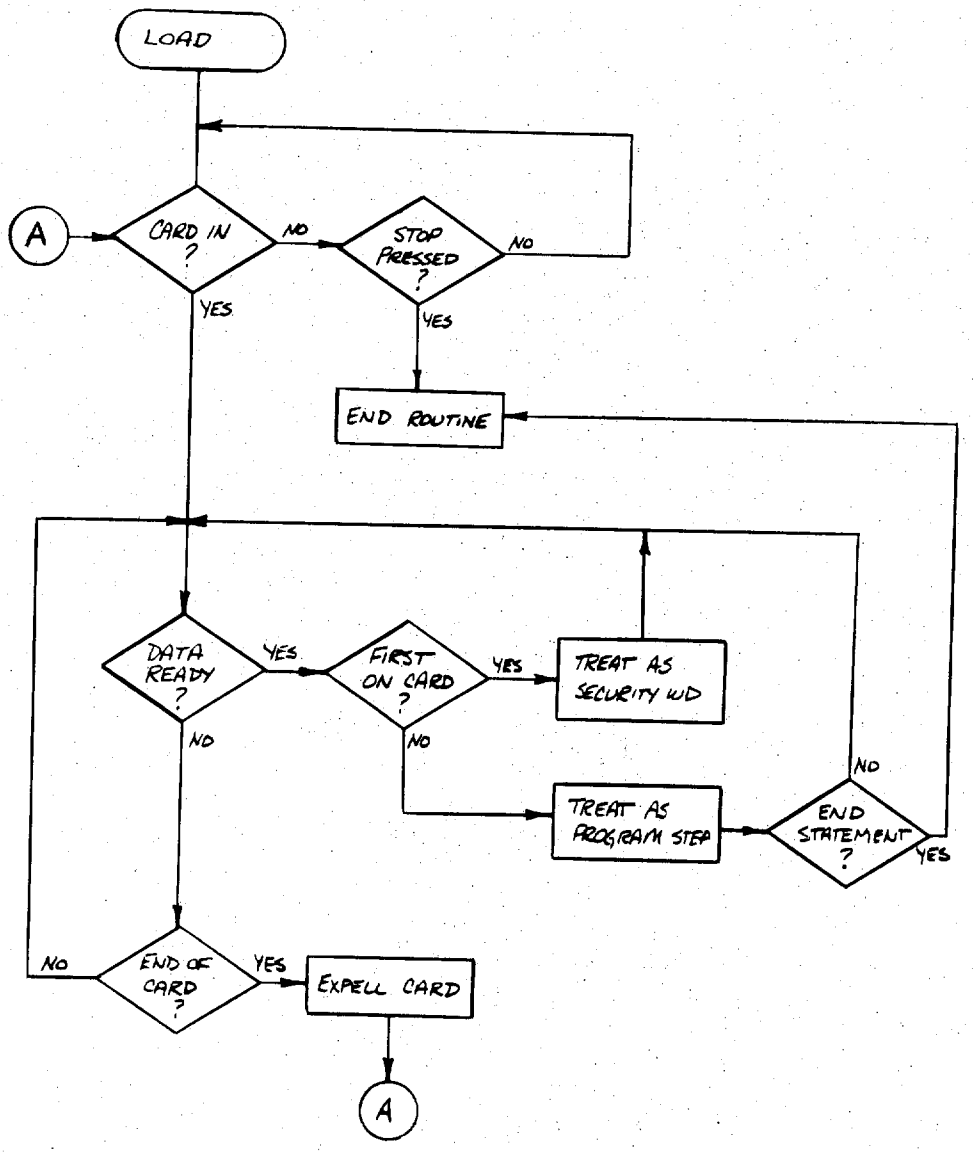
FIG 58B

PLOTTER IS NOT READY IF:
1. No Plotter is hooked up.
2. Plotter Power is not on



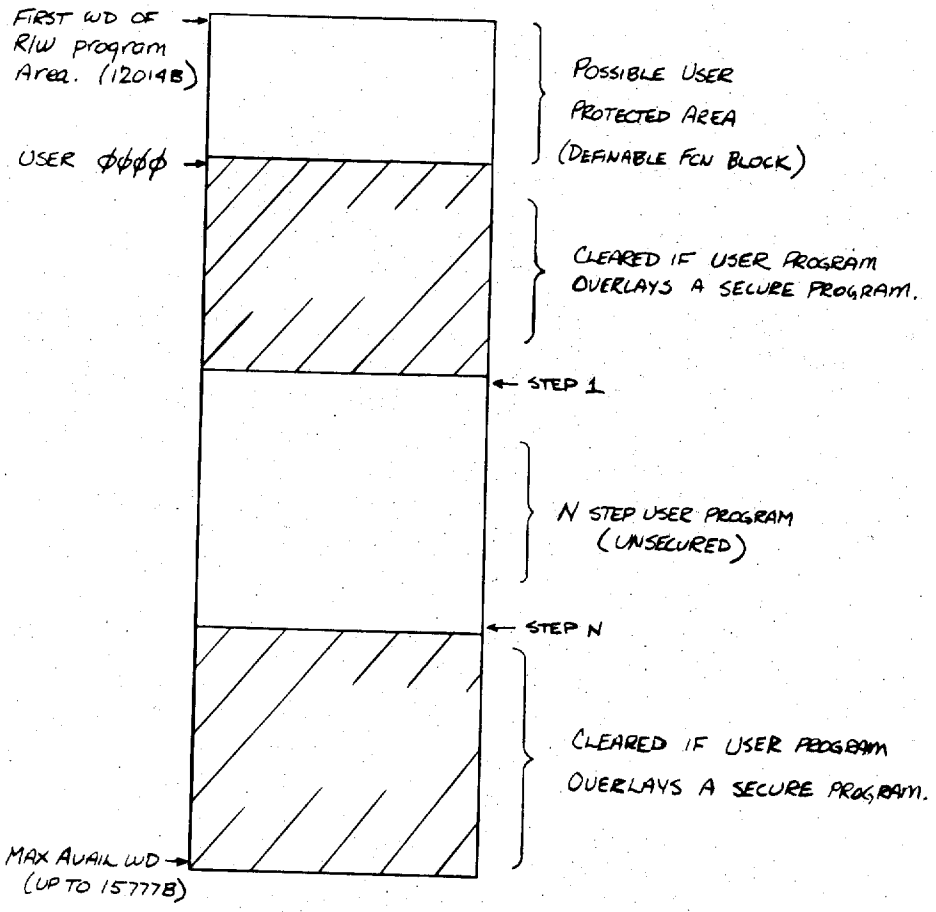
BASIC RECORD OPERATION
(PROGRAM STEPS)

FIG 60



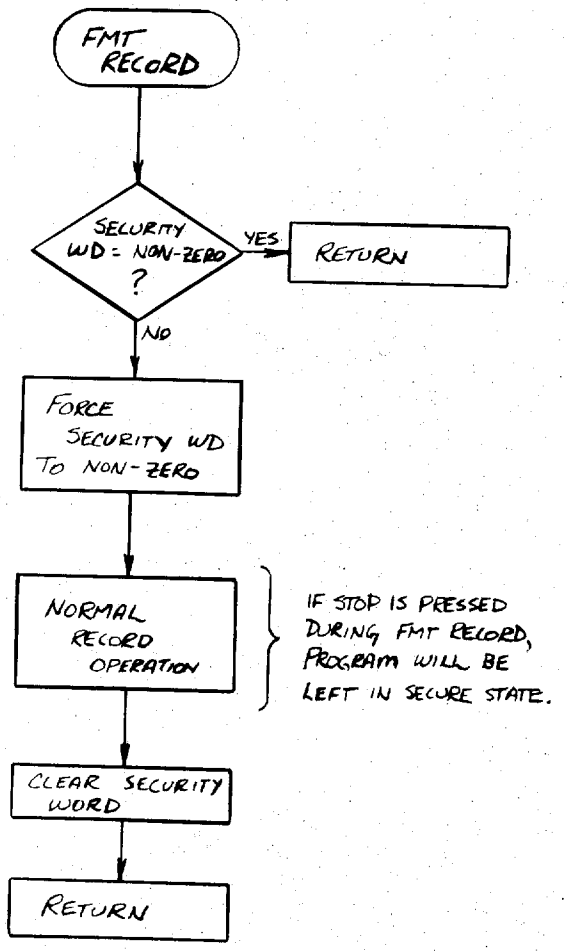
BASIC LOAD OPERATION
(PROGRAM STEPS)

FIG 61



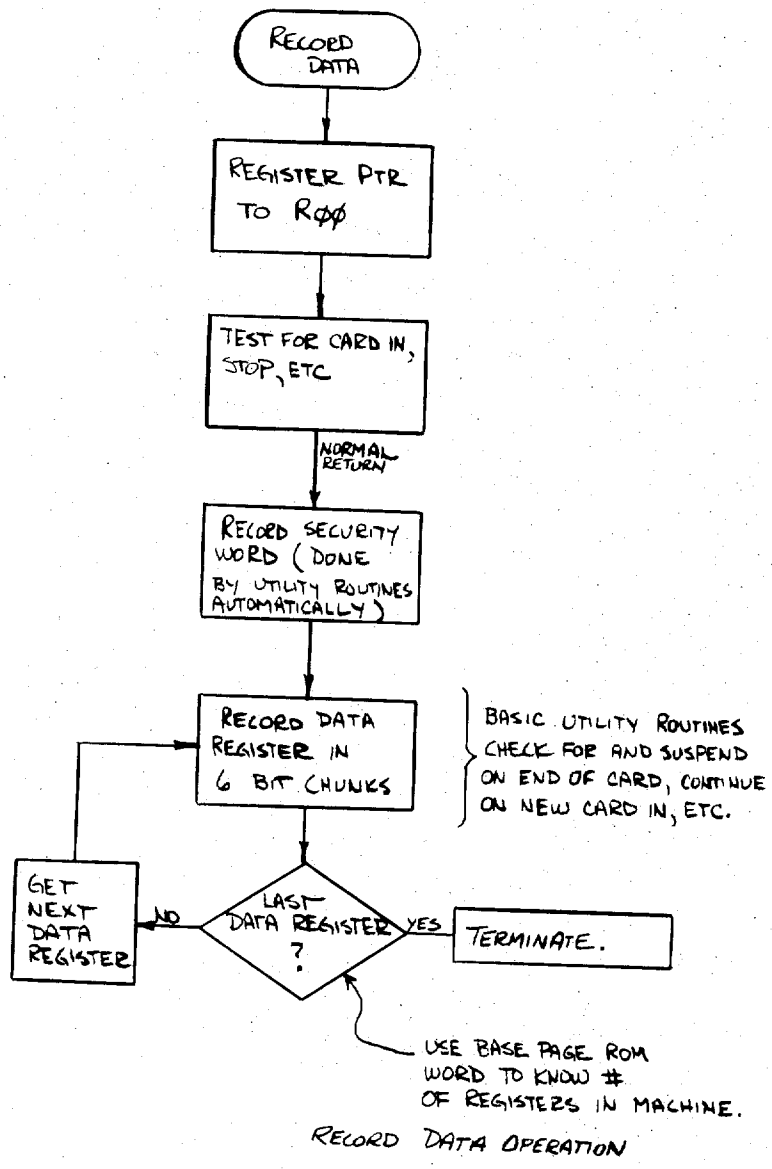
HOW A SECURE PROGRAM IS DESTROYED BY AN UNSECURED OVERLAY.

FIG 62



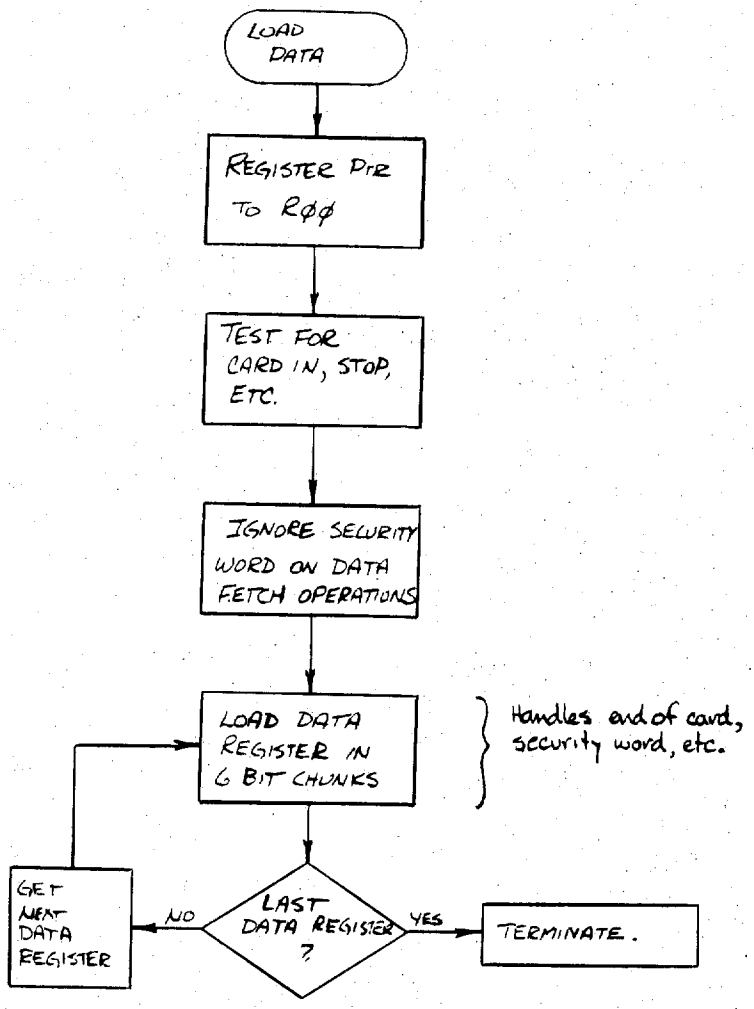
FMT RECORD OPERATION.

FIG 63



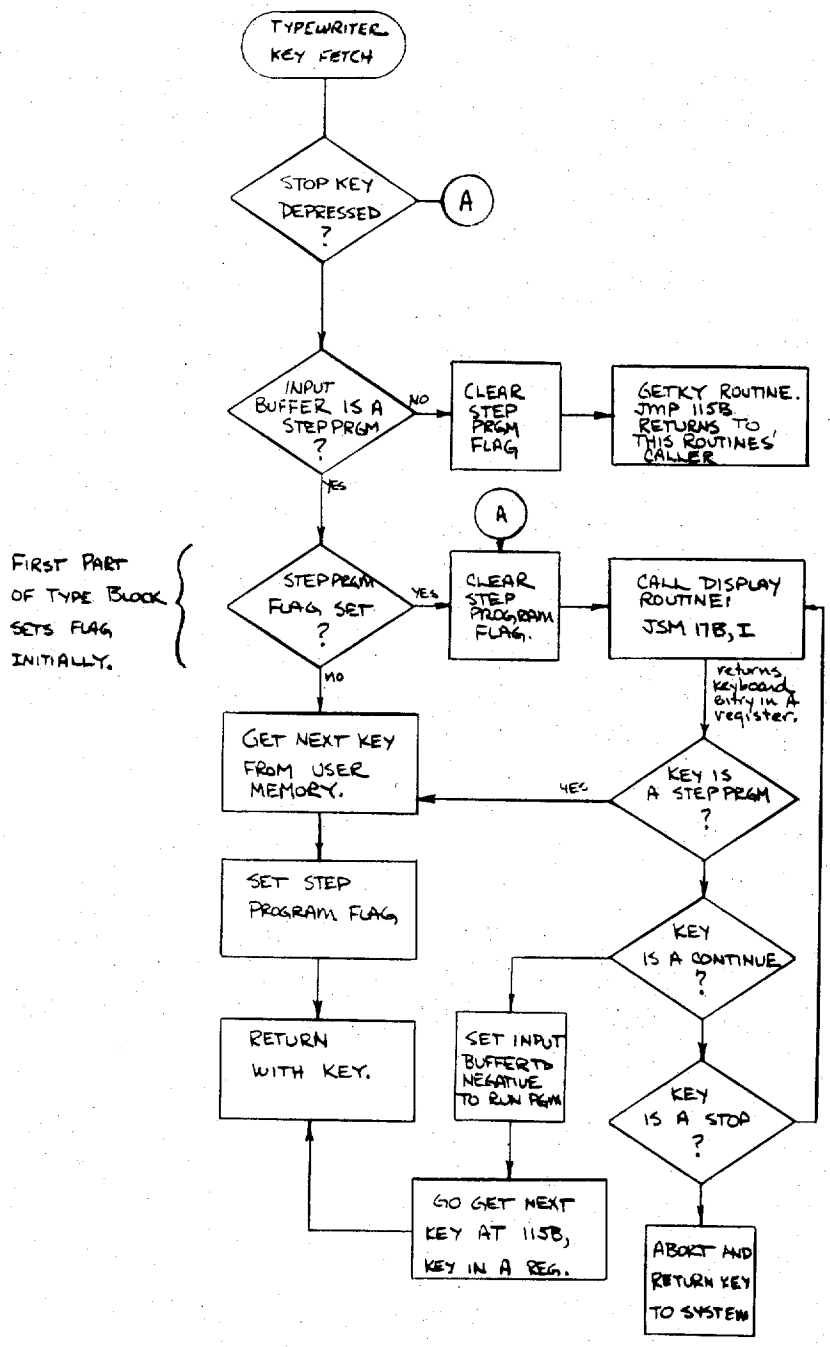
RECORD DATA OPERATION

FIG 64



LOAD DATA OPERATION

FIG 65



HOW AN OPTIONAL BLOCK CAN GET THE NEXT KEY, PAYING ATTENTION TO STEP PROGRAM

FIG 66

① The print routine stores 5x7 dot matrix information in a table like this: →

Notice that adjacent chars do not have identical structures; all odd keycode chars have one type of entry, while all even keycode chars have another.

x	1	2	3
x	4	5	6
x	7	—	1
x	2	3	4
x	5	6	7
x			
x			
x			

2 Characters

FIG 67A

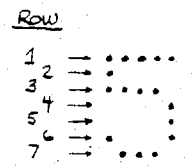


FIG 67B

	odd		even chars	
Row 1	Right φ	+0	Right 10	φ
2	Right 10	+1	Right 5	φ
3	Right 5	+1	Zero	φ
4	" 0	+1	" 10	+1
5	" 10	+2	" 5	+1
6	" 5	+2	" φ	+1
7	" φ	+2	" 10	+2
	Rotate	Offset	Rotate	Offset

FIG 67C

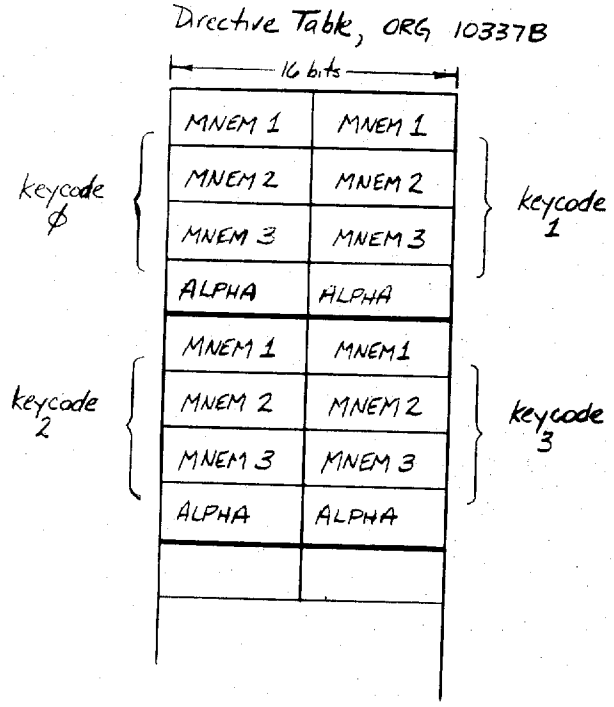
② Once a keycode reaches the print routine code access portion, the table entry is computed as $\frac{5*\text{char code}}{2}$ for even codes, and $\frac{(5*\text{char code})-1}{2}$ for odd codes.

Adding the table base gives the address of the word containing the first row of printer code for that character.

③ Having the address of the first word in the character group, it is necessary to decide how to get the particular row of information desired. An auxiliary table, whose entry point is the current row #, gives an offset to apply to the address in hand, as well as a rotation amount required to position the particular code.

④ Adding the offset, getting the word, and applying the rotation results in having the desired bits in a known location. Done.

HOW THE PRINT ROUTINE ACCESSES CODE INFORMATION



Entries in this table are pointers to the thermal printer code table, which contains 5×7 dot matrix information for every printable character.

ALPHA BLOCK POINTER STRUCTURE.
(Directive Table)

FIG 68A

Code Table, ORG 10540

15 bits

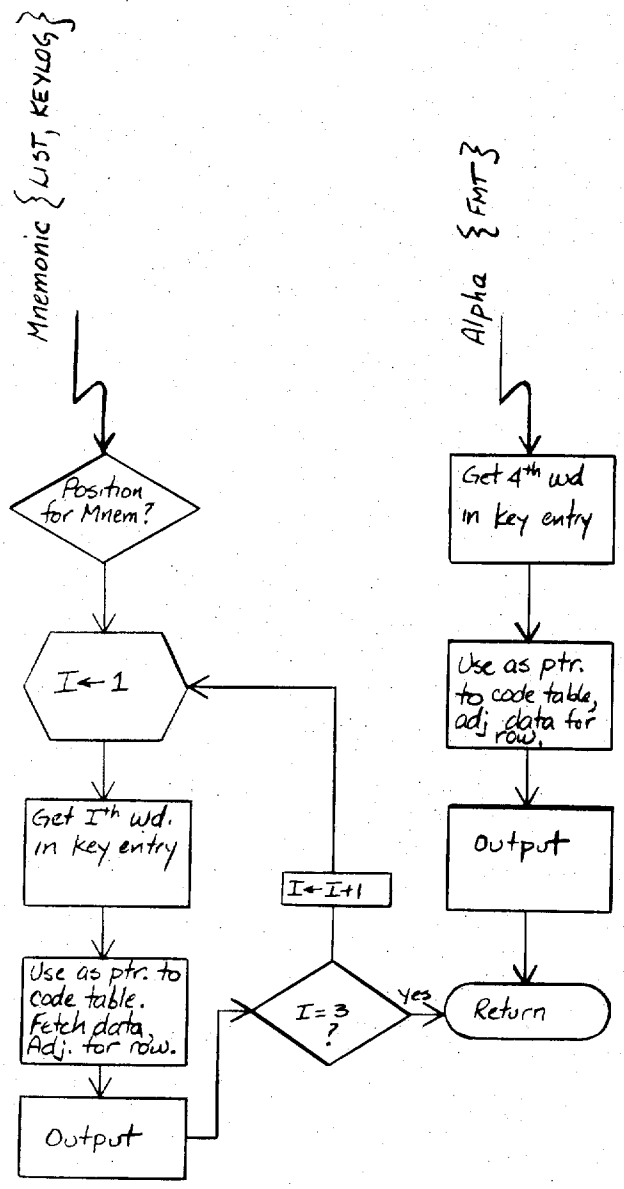
1ST Char.	Row 1	Row 2	Row 3	
	Row 4	Row 5	Row 6	
	Row 7	Not Used	Row 1	
	Row 2	Row 3	Row 4	2nd Char.
	Row 5	Row 6	Row 7	
3rd Char.	Row 1	Row 2	Row 3	
	Row 4	Row 5	Row 6	
	Row 7	Not Used	Row 1	
	Row 2	Row 3	Row 4	4th Char.
	Row 5	Row 6	Row 7	

Bit = 1 ⇒ Print a dot.
Bit = 0 ⇒ Don't print a dot.

ALPHA BLOCK CODE STRUCTURE.

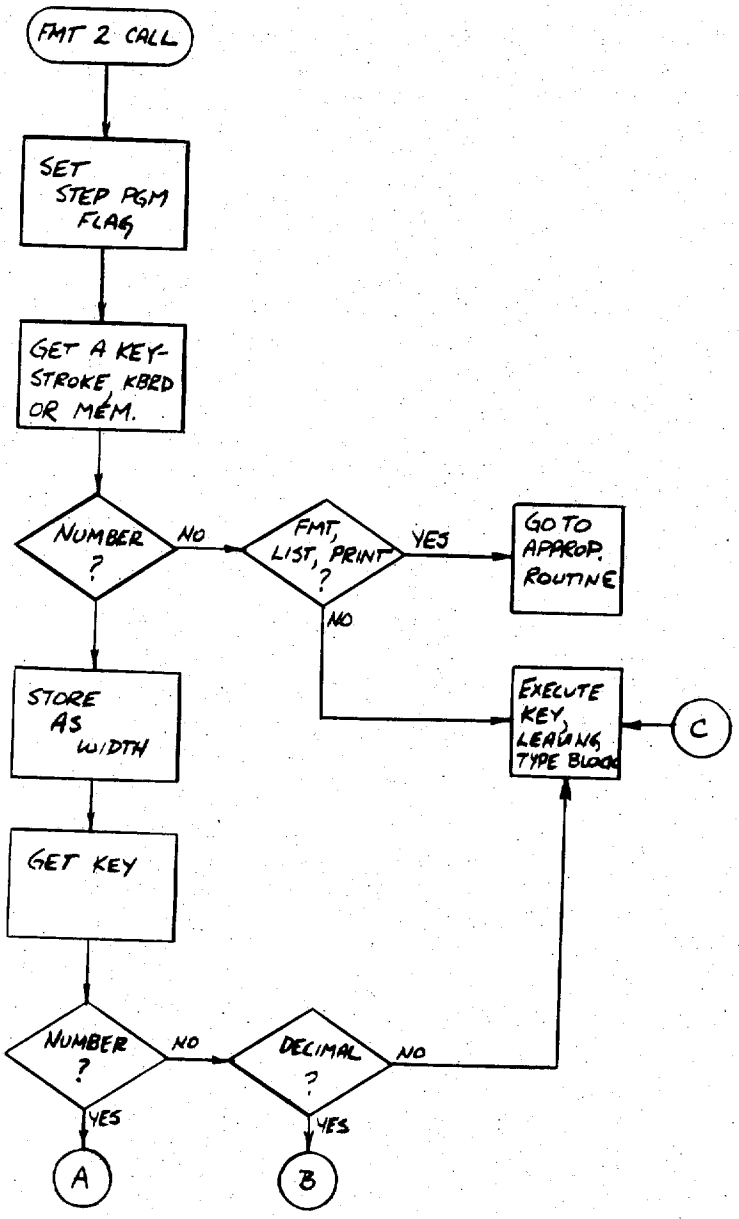
FIG 68B

The Alpha block interacts with the basic system in the following way: Whenever the print routine gets five bits of printer code, the search routine is called to look for the Alpha block. If it is not present, the print routine retains control and prints the 5 bits it already has. If the alpha block is present, control goes to it; the Alpha block looks at certain print routine variables to determine the current position in the line. If it is not time to print MNEMONICS, it returns to the basic print routine without affecting the 5 bit code. If it is time to print MNEMONICS, the alpha routine checks the subroutine stack to determine that it is not just a number print that was requested. If it was in fact a MNEMONIC call, Alpha reconstructs the keycode from the display list, accesses the printer code, and inserts it in place of the basic routines 5 bit code. Alpha returns, and the basic print routine continues.



ACCESSING PRINTER CODE.

FIG 69



TYPEWRITER FORMAT ROUTINE.

FIG 70A

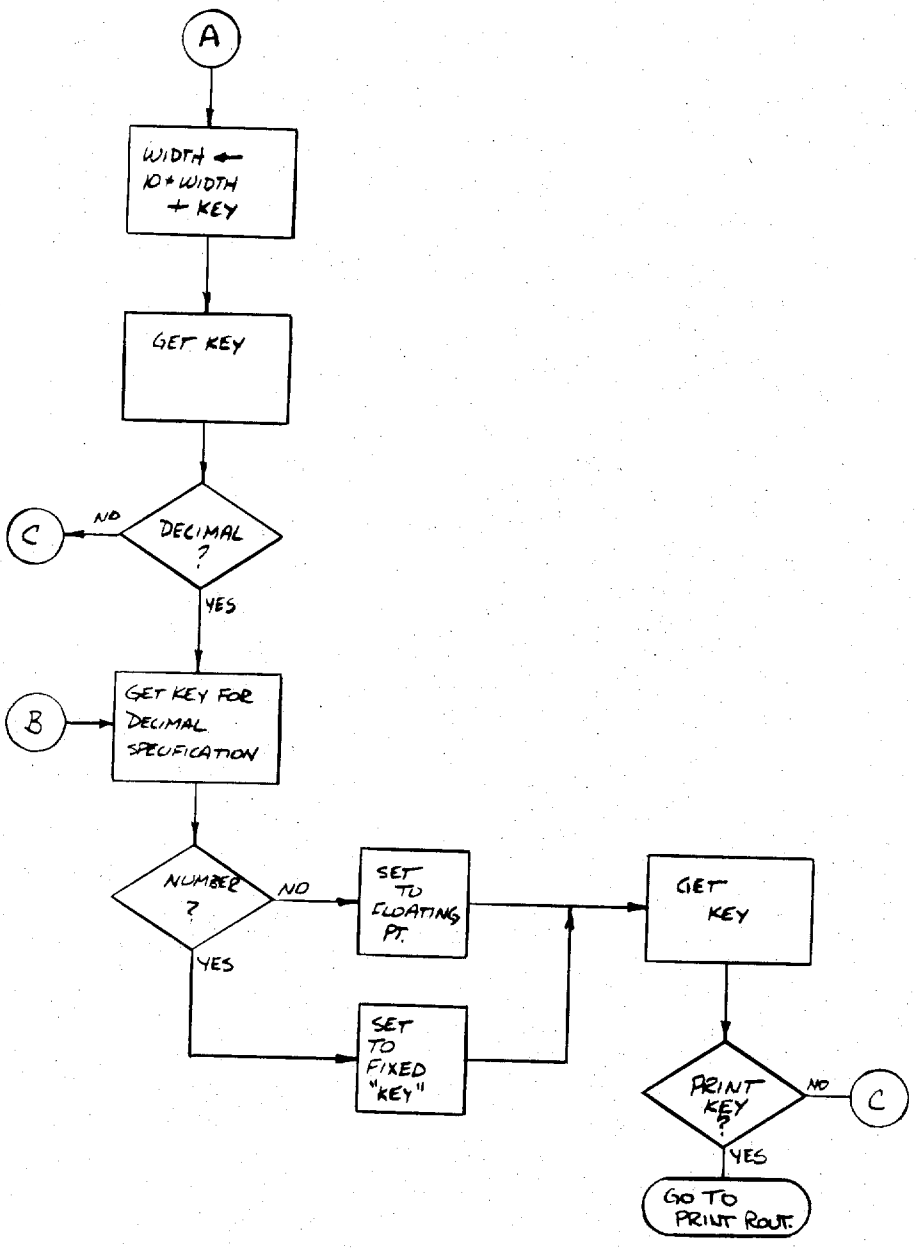
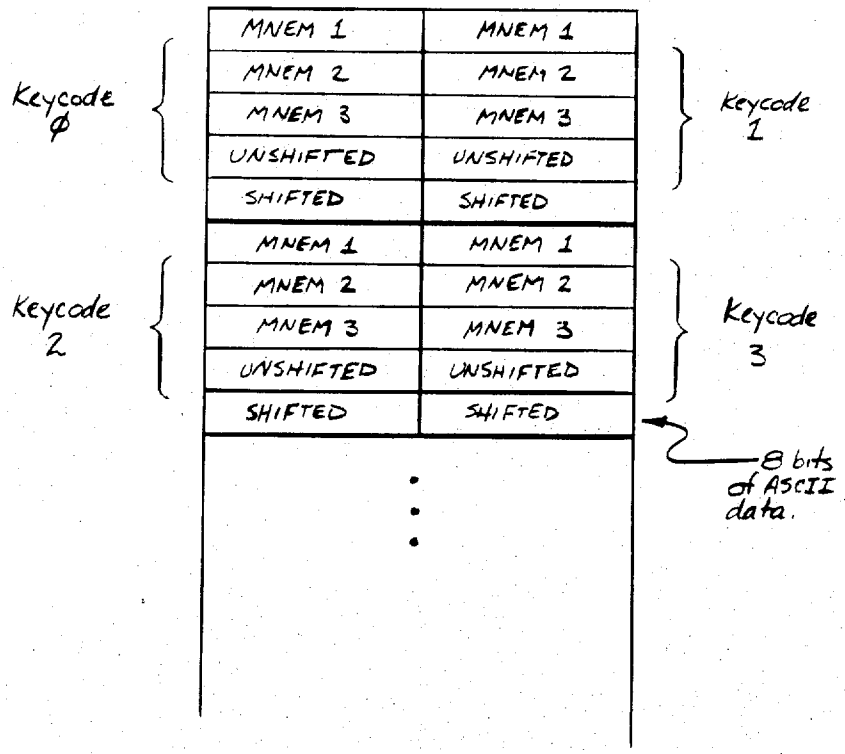
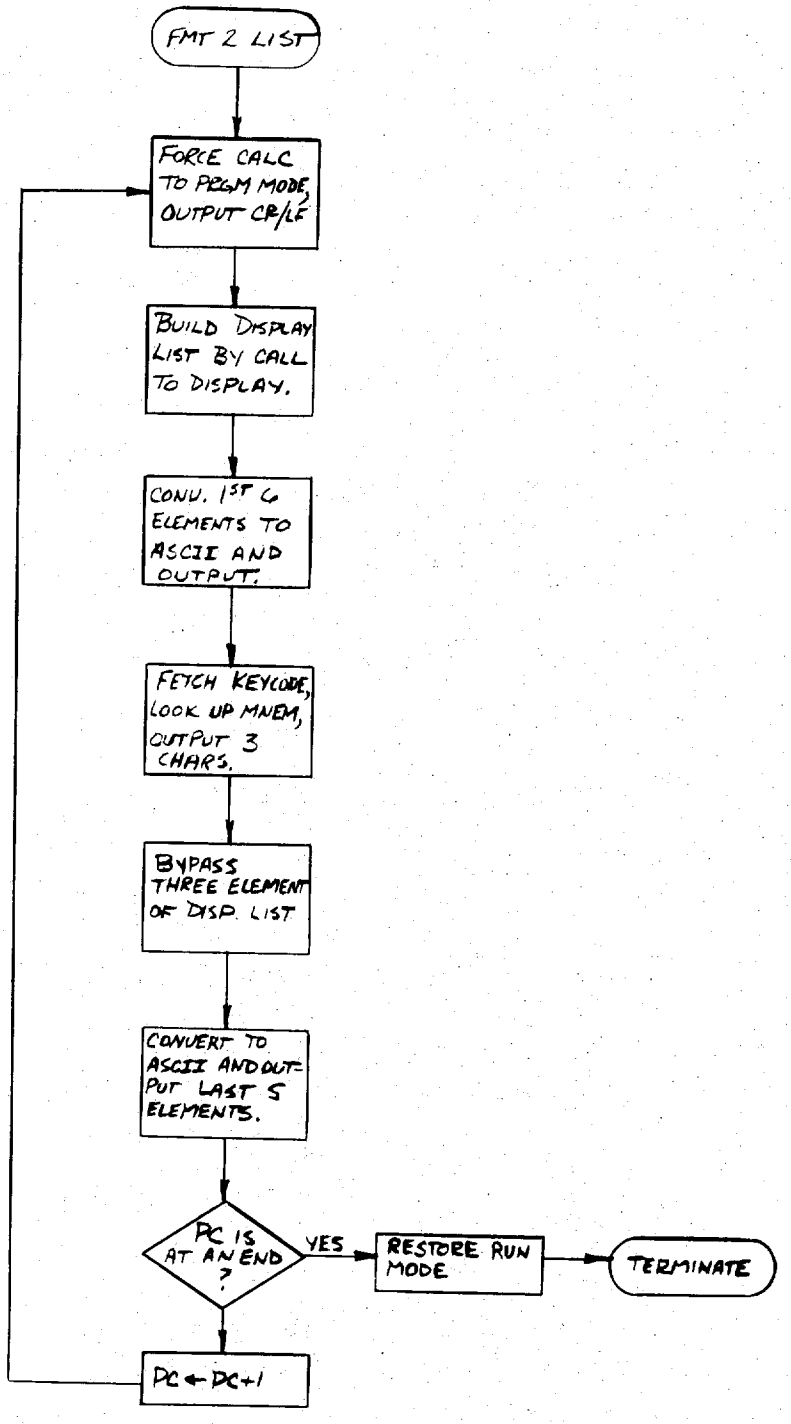

A
WIDTH ← 10*WIDTH + KEY
GET KEY
C
NO
DECIMAL ?
YES
B
GET KEY FOR DECIMAL SPECIFICATION
NUMBER ?
NO
SET TO FLOATING PT.
YES
SET TO FIXED "KEY"
GET KEY
PRINT KEY ?
NO
C
YES
GO TO PRINT ROUT.

FIG 70B



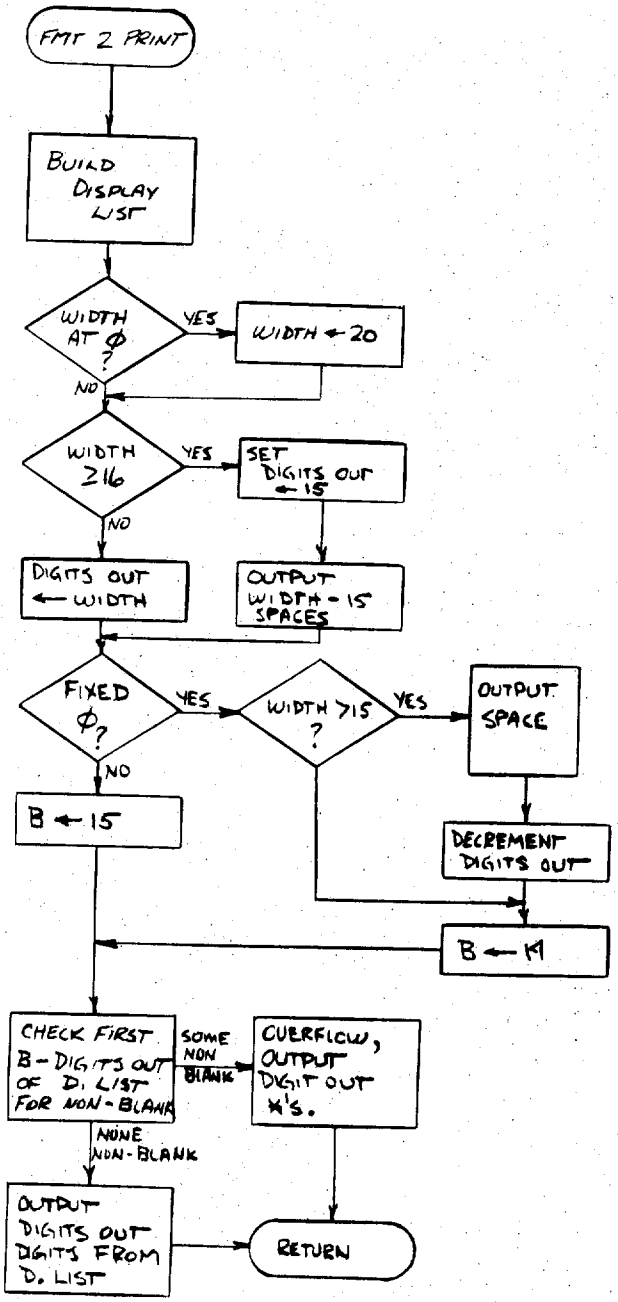
TYPEWRITER MNEMONIC TABLE STRUCTURE

FIG 71



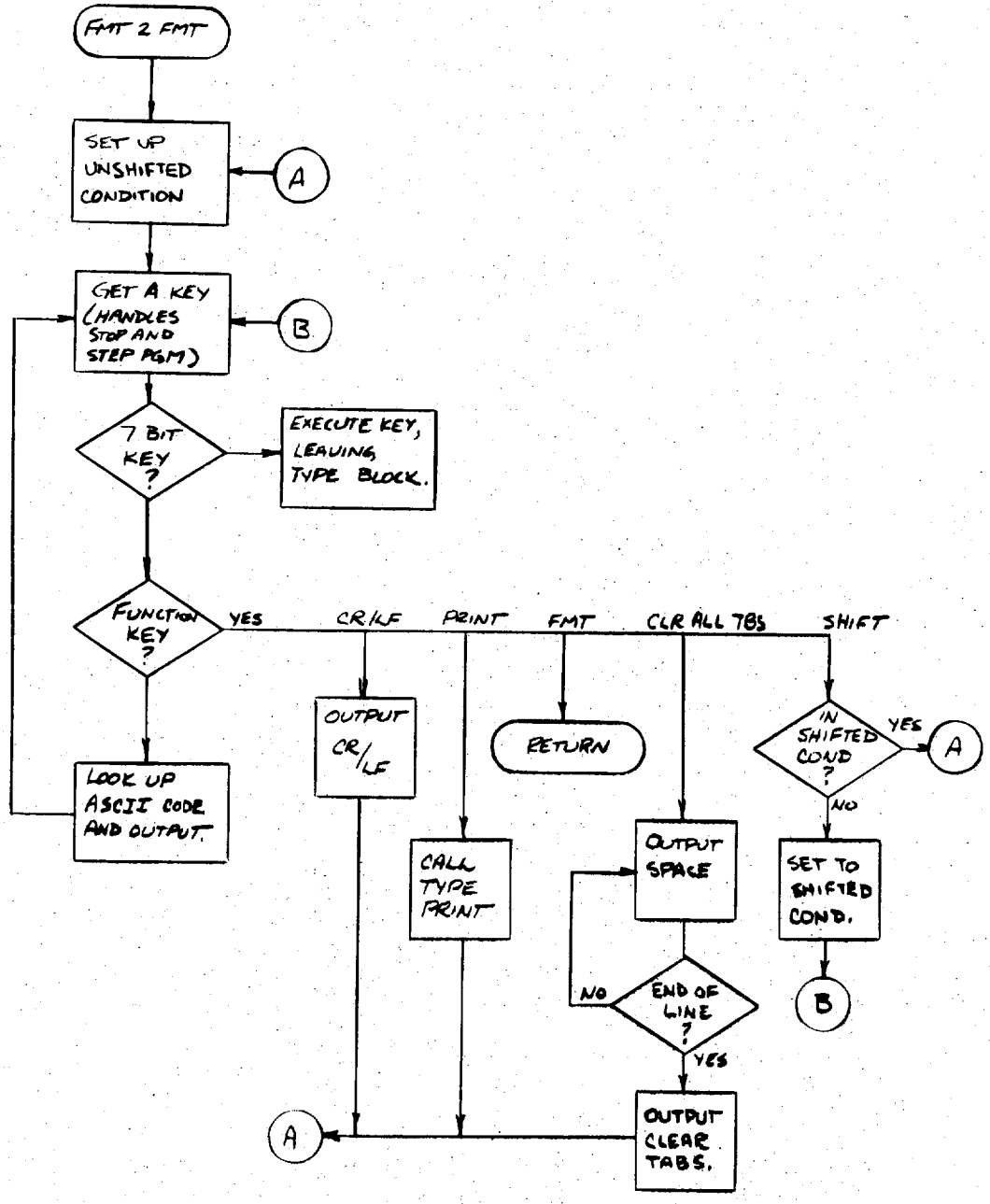
TYPEWRITER LIST ROUTINE.

FIG 72



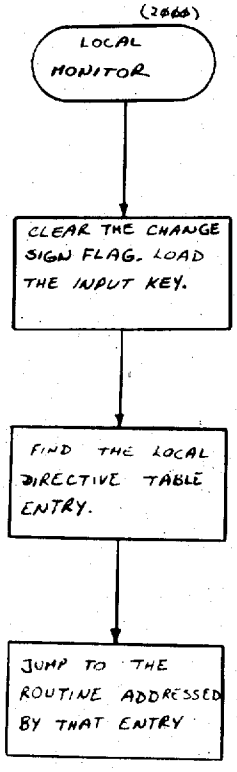
TYPEWRITER PRINT ROUTINE

FIG 73



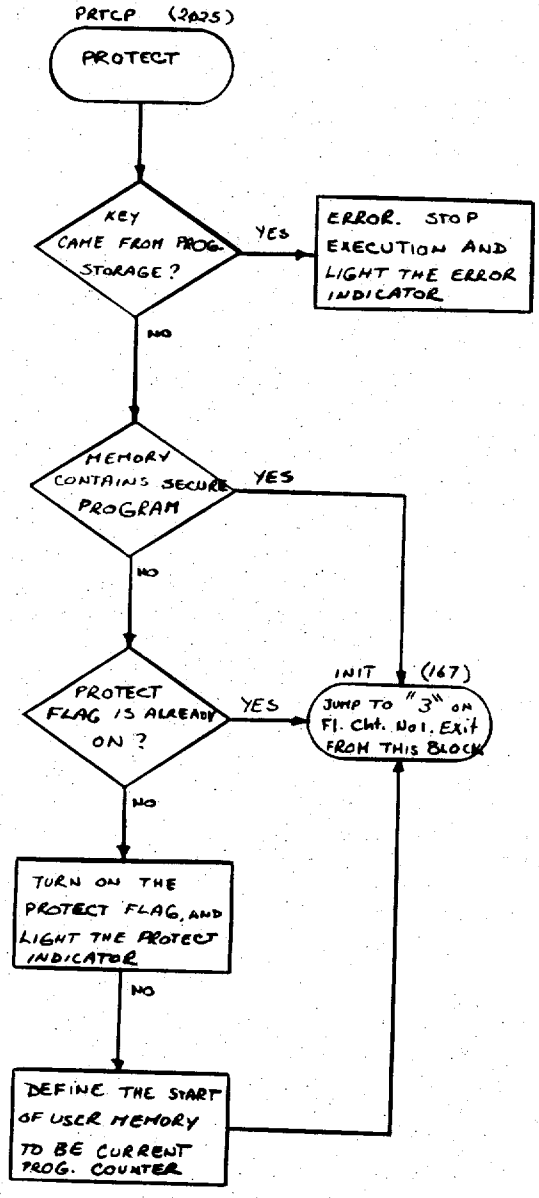
TYPEWRITER ALPHA ROUTINE

FIG 74



ALL KEYS ON OPTIONAL BLOCK ENTER HERE

FIG 75



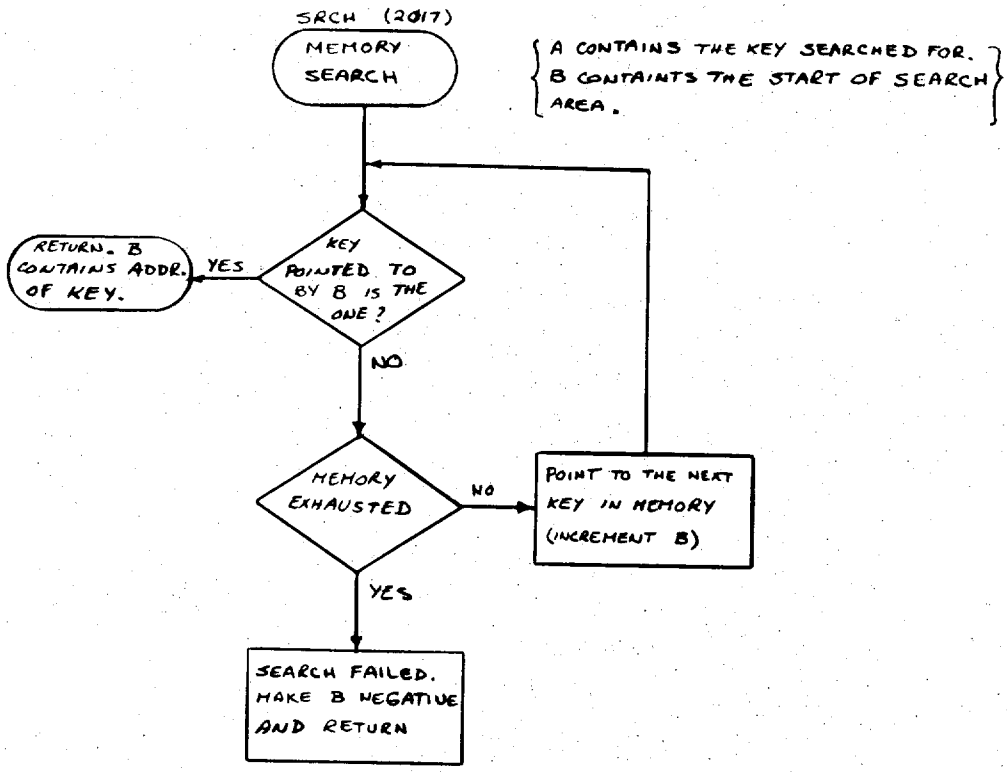
LOCAL DIRECTIVE TABLE ENTRY

No. 2016

PROTECT KEY PROCESSING

KEYCODE 16

FIG 76



BASIC MEMORY SEARCH ROUTINE

FIG 77

USER DEFINABLE FUNCTIONS BLOCK

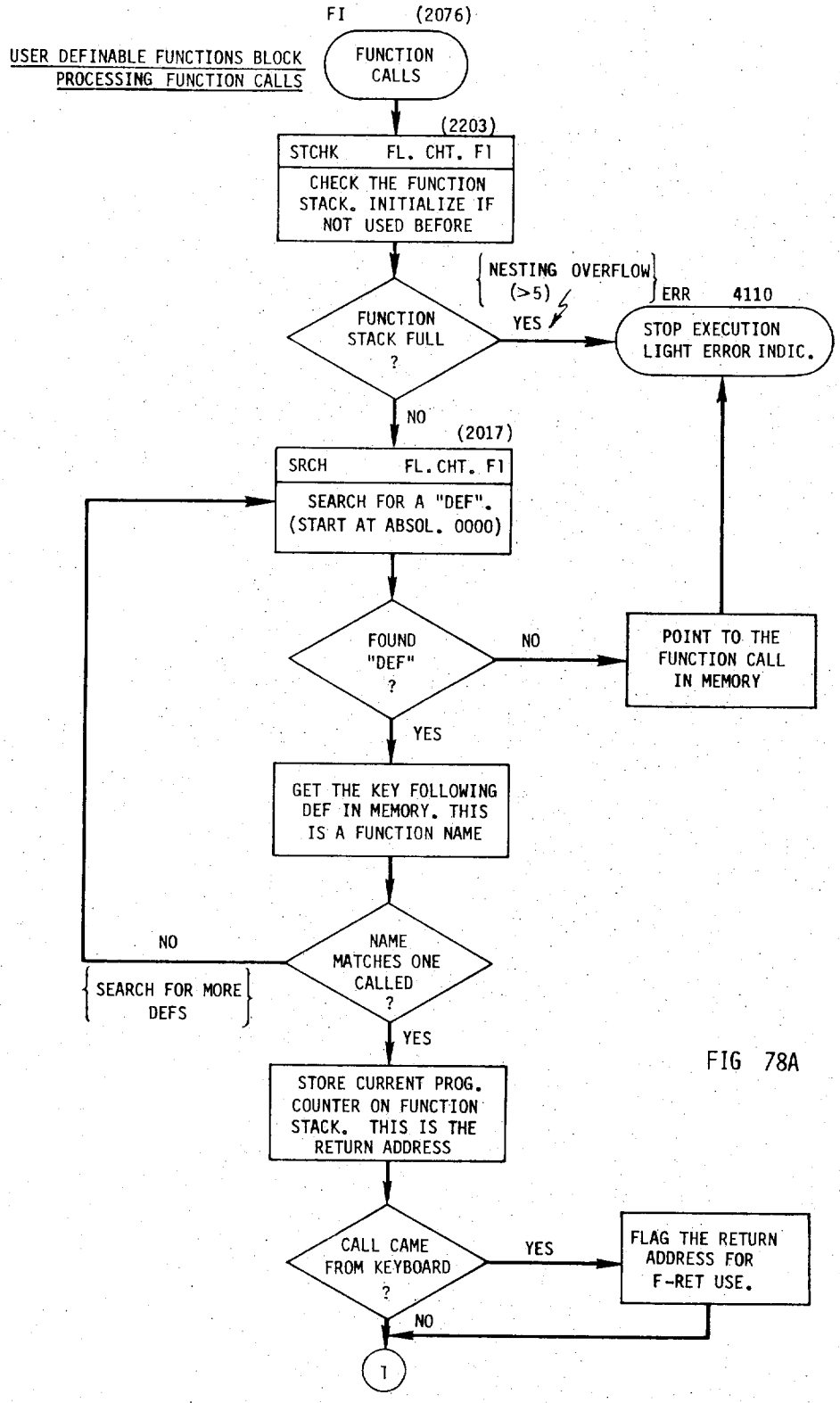
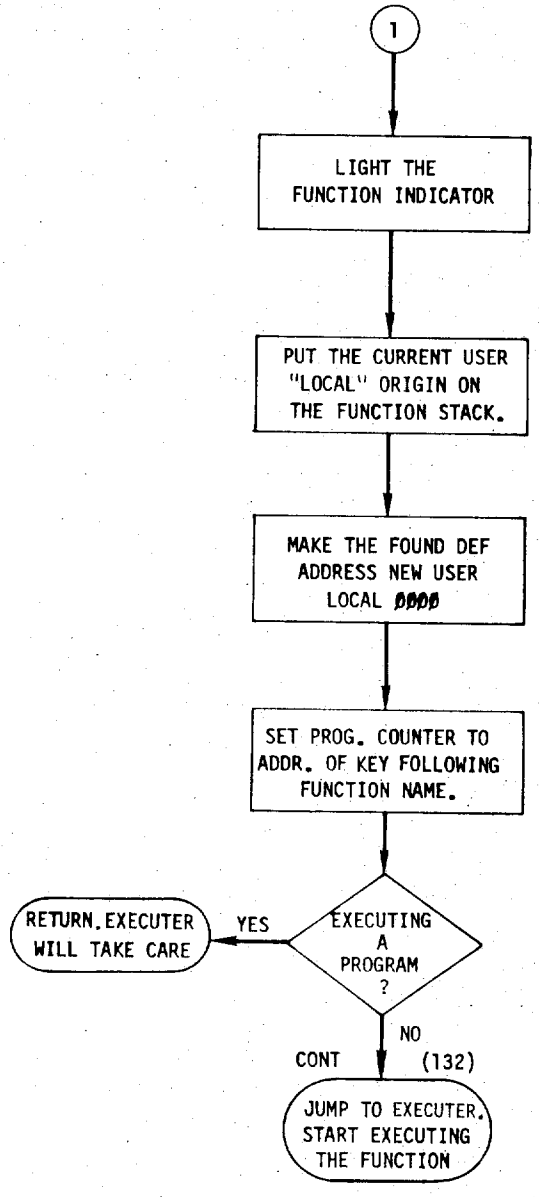

USER DEFINABLE FUNCTIONS BLOCK PROCESSING FUNCTION CALLS
FI (2076)
FUNCTION CALLS
(2203)
STCHK FL. CHT. F1
CHECK THE FUNCTION STACK. INITIALIZE IF NOT USED BEFORE
NESTING OVERFLOW (>5)
ERR 4110
FUNCTION STACK FULL ?
YES
STOP EXECUTION LIGHT ERROR INDIC.
NO
(2017)
SRCH FL.CHT. F1
SEARCH FOR A "DEF". (START AT ABSOL. 0000)
FOUND "DEF" ?
NO
POINT TO THE FUNCTION CALL IN MEMORY
YES
GET THE KEY FOLLOWING DEF IN MEMORY. THIS IS A FUNCTION NAME
NAME MATCHES ONE CALLED ?
NO
SEARCH FOR MORE DEFS
YES
STORE CURRENT PROG. COUNTER ON FUNCTION STACK. THIS IS THE RETURN ADDRESS
CALL CAME FROM KEYBOARD ?
YES
FLAG THE RETURN ADDRESS FOR F-RET USE.
NO
1

FIG 78A



USER DEFINABLE FUNCTIONS BLOCK PROCESSING FUNCTION CALLS

FIG 78B

LOCAL DIRECTIVE TABLE ENTRIES:		KEYCODE
2Ø66	(F1)	66
2Ø61	(F2)	61
2Ø63	(F3)	63
2Ø6Ø	(F4)	6Ø
2Ø75	(F5)	75
2Ø71	(F6)	71
2Ø73	(F7)	73
2Ø7Ø	(F8)	7Ø
2Ø72	(F9)	72

FIG 79

USER DEFINABLE FUNCTION BLOCK

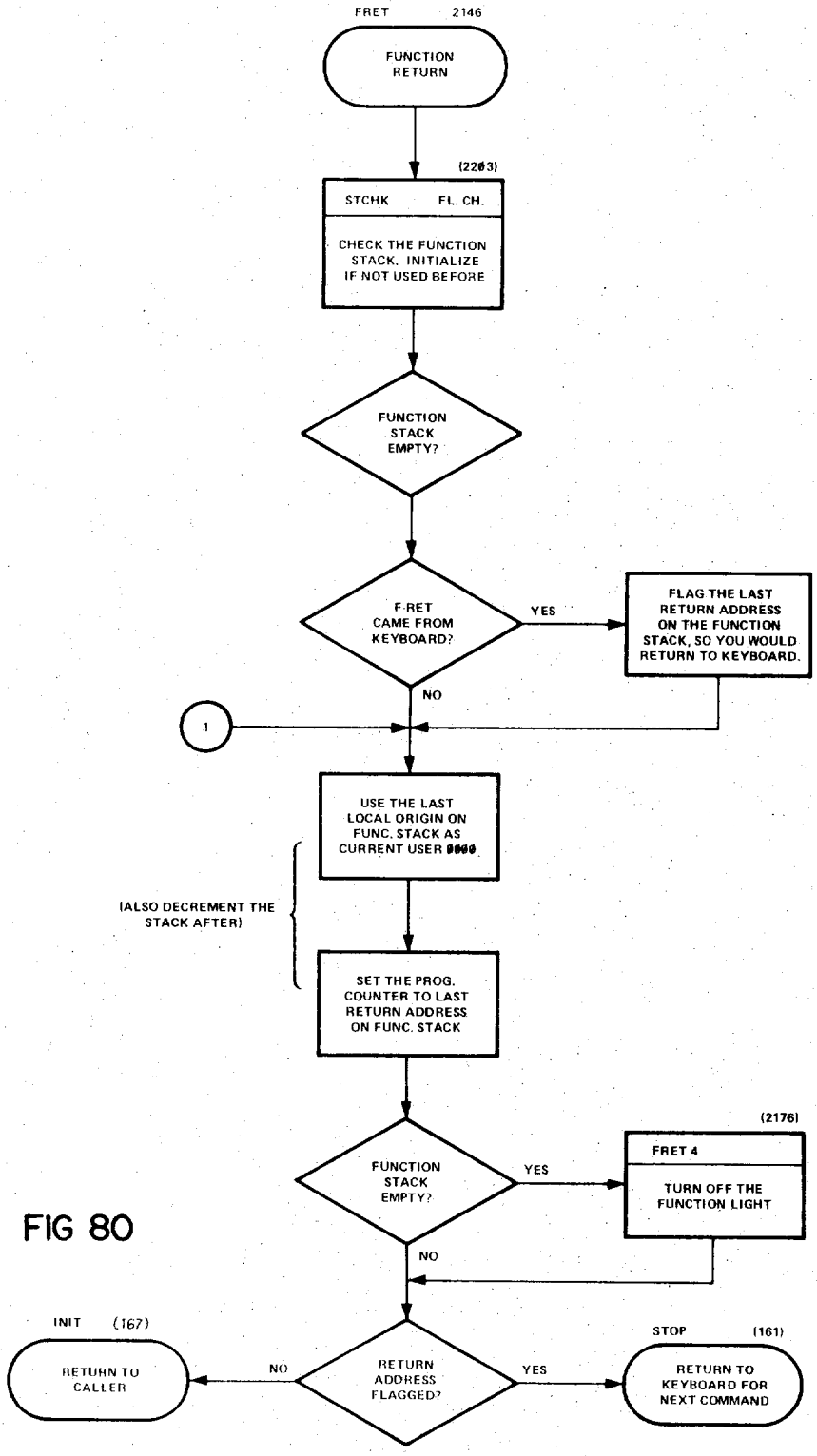
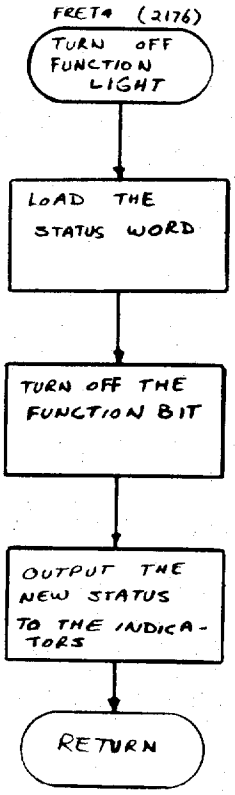


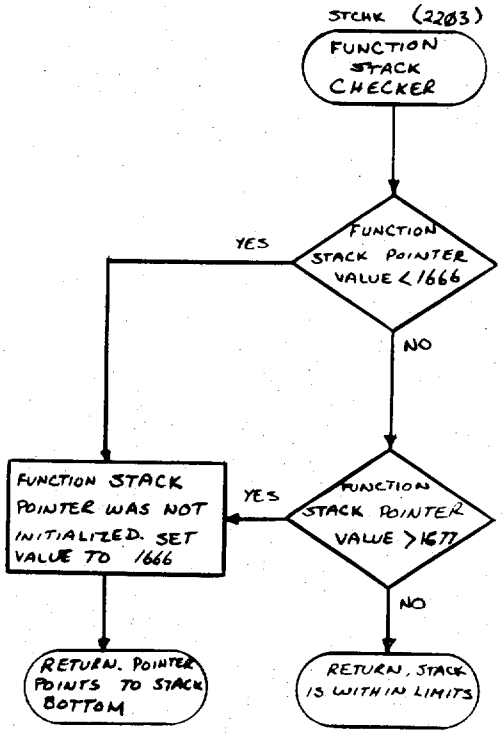
FIG 80

PROCESSING F-RET LOCAL DIRECTIVE ENTRY NO. 2055 (KEYCODE 55)



TURN OFF THE FUNCTION LIGHT

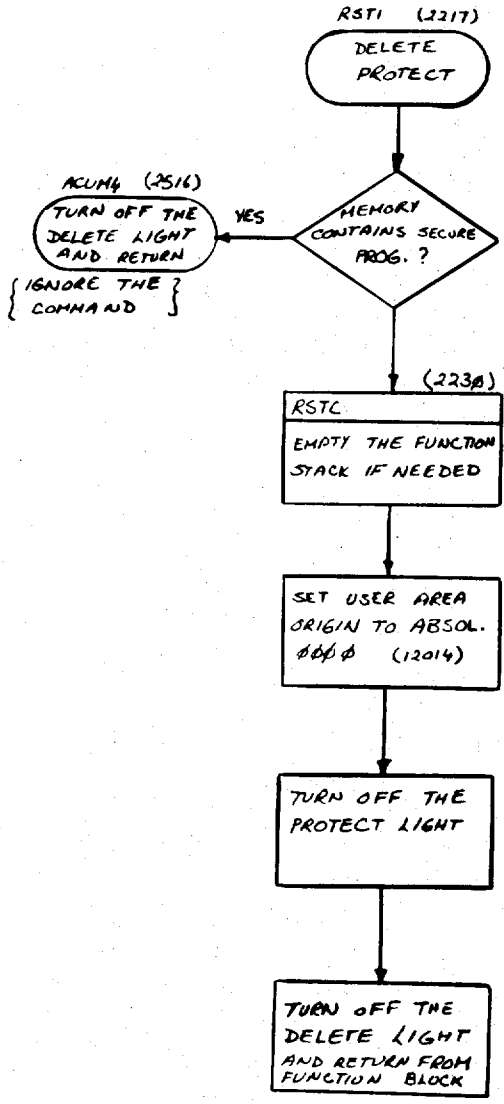
FIG 81



FUNCTION STACK CHECKING ROUTINE

FIG 82

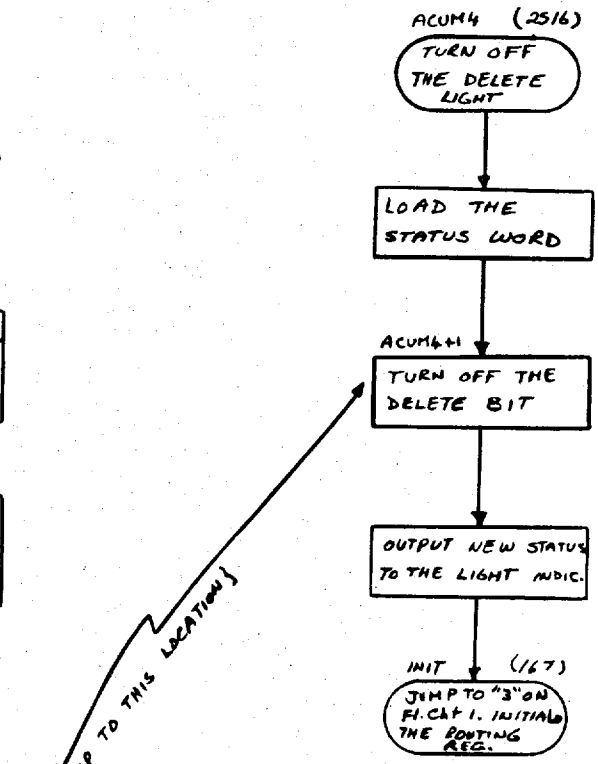
USER DEFINABLE FUNCTION BLOCK



PROCESSING DELETE-PROTECT

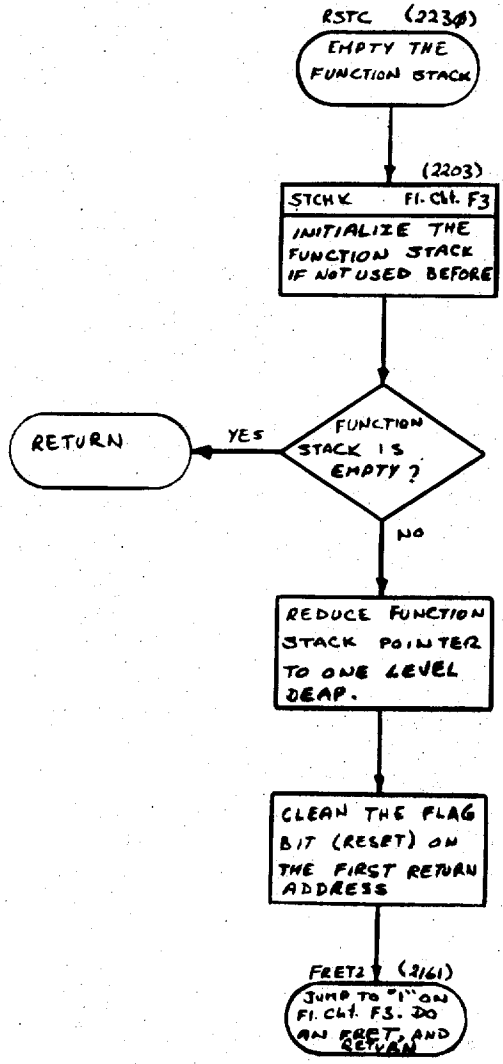
FIG 83

USER DEFINABLE FUNCTION BLOCK



TURN OFF THE DELETE LIGHT

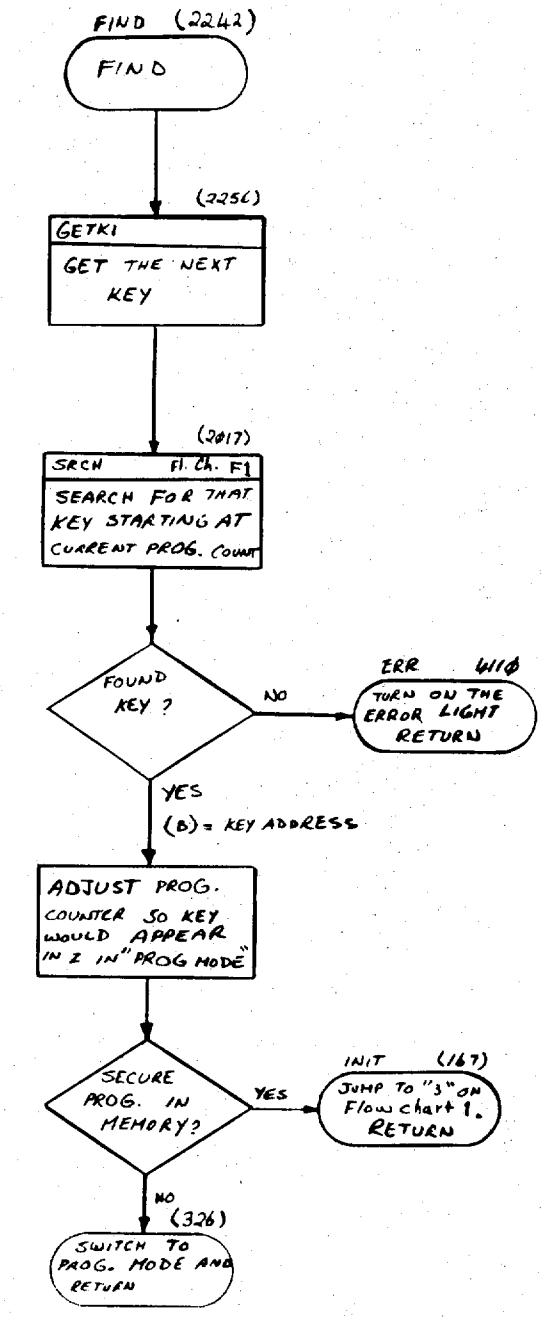
FIG 84



EMPTY THE FUNCTION STACK AND TURN OFF THE FUNC. LIGHT

FIG 85

USER DEFINABLE FUNCTION BLOCK

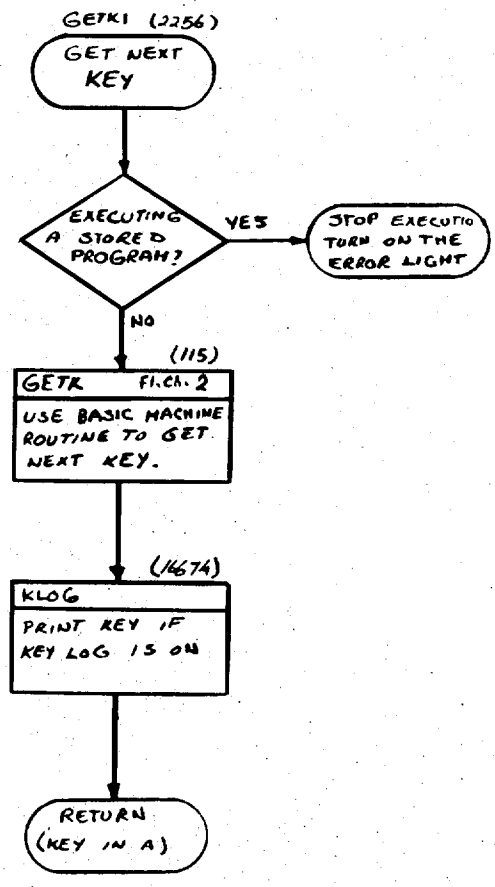


FIND PROCESSING

FIG 86

LOCAL DIRECTIVE TABLE ENTRY NO. 2074

(KEYCODE 74)



GET NEXT KEY

FIG 87

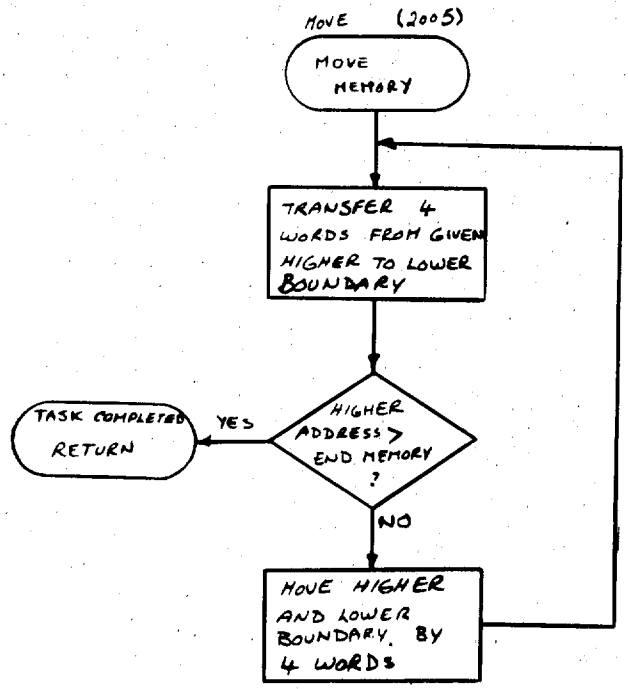
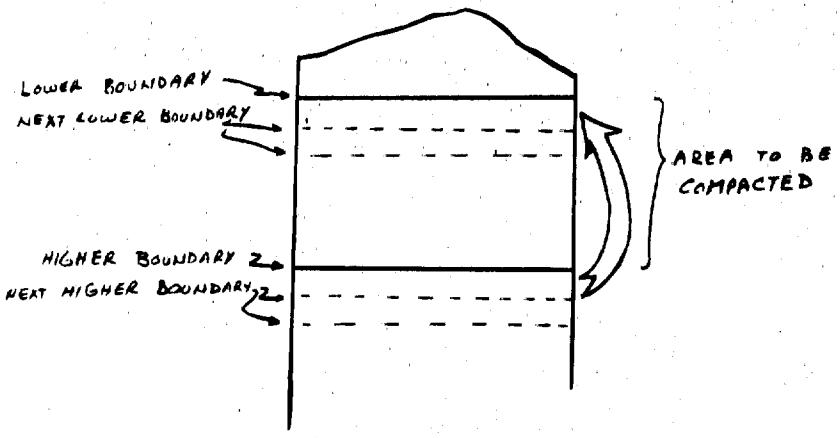


FIG 88



MEMORY COMPACTER

FIG 89

USER DEFINABLE FUNCTION BLOCK

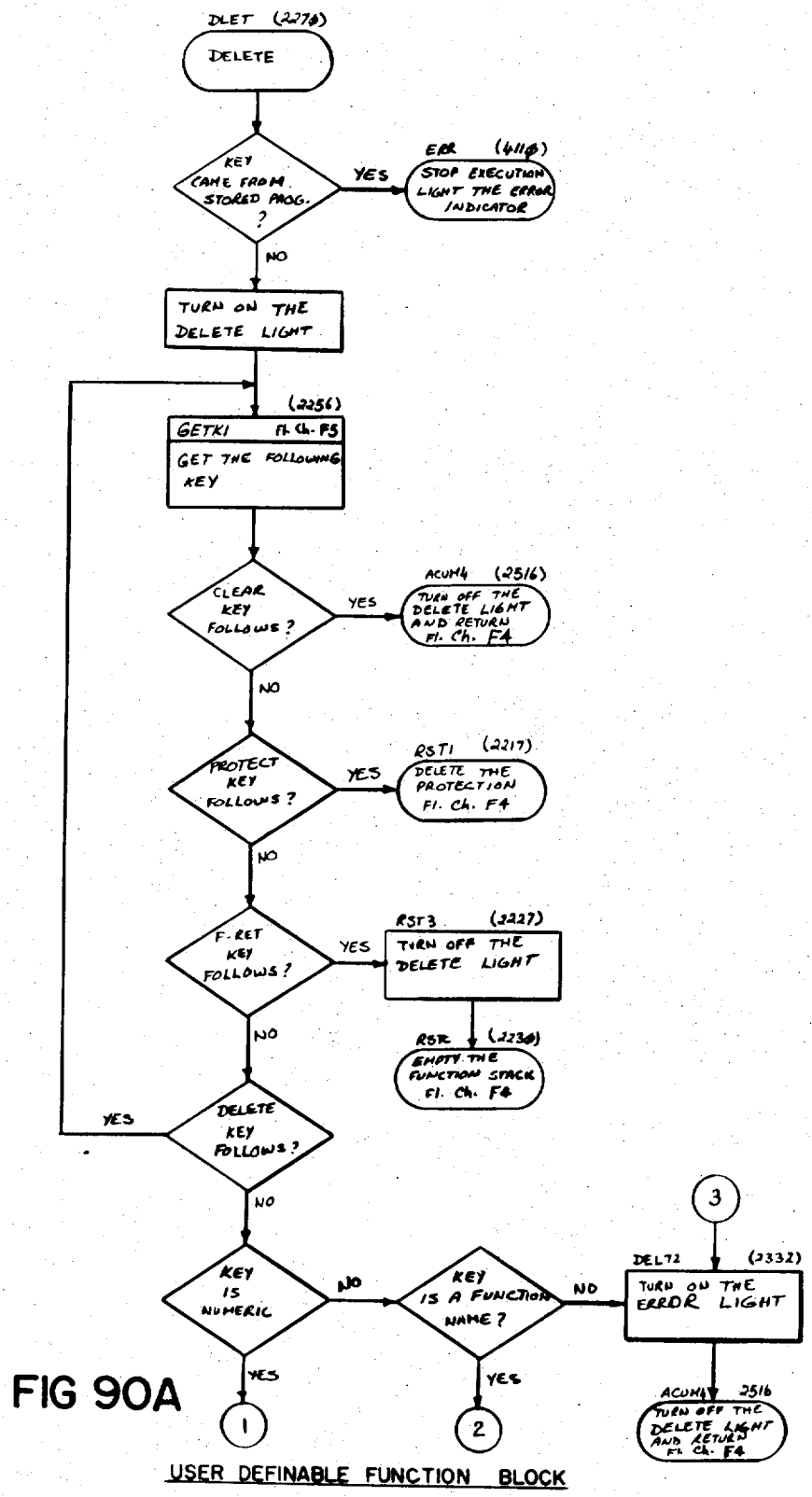


FIG 90A

USER DEFINABLE FUNCTION BLOCK

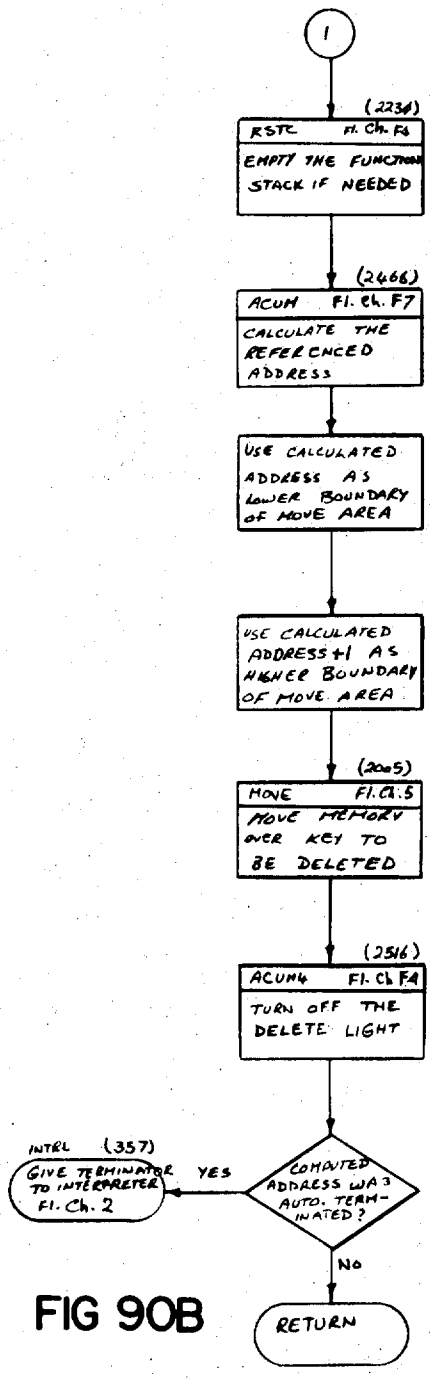


FIG 90B

DELETE PROCESSING

LOCAL DIRECTIVE TABLE ENTRY NO. 2062

(KEYCODE 62)

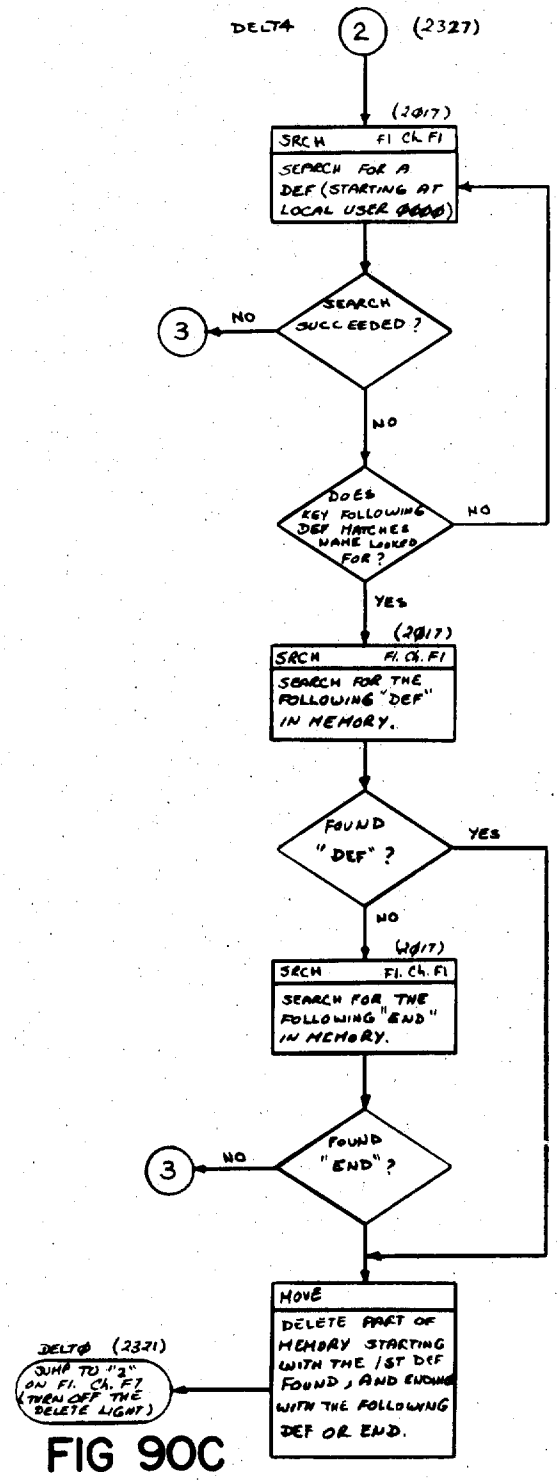


FIG 90C

USER DEFINABLE FUNCTION BLOCK

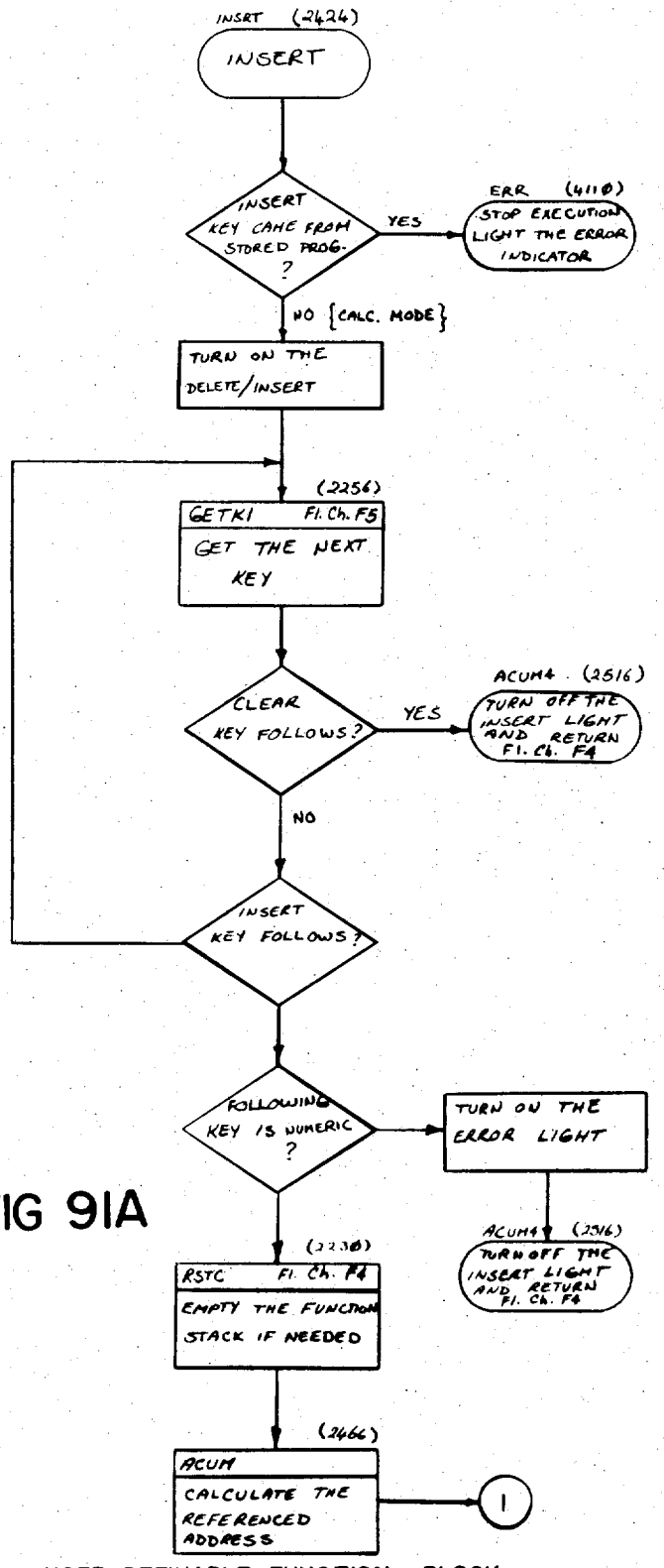


FIG 91A

USER DEFINABLE FUNCTION BLOCK

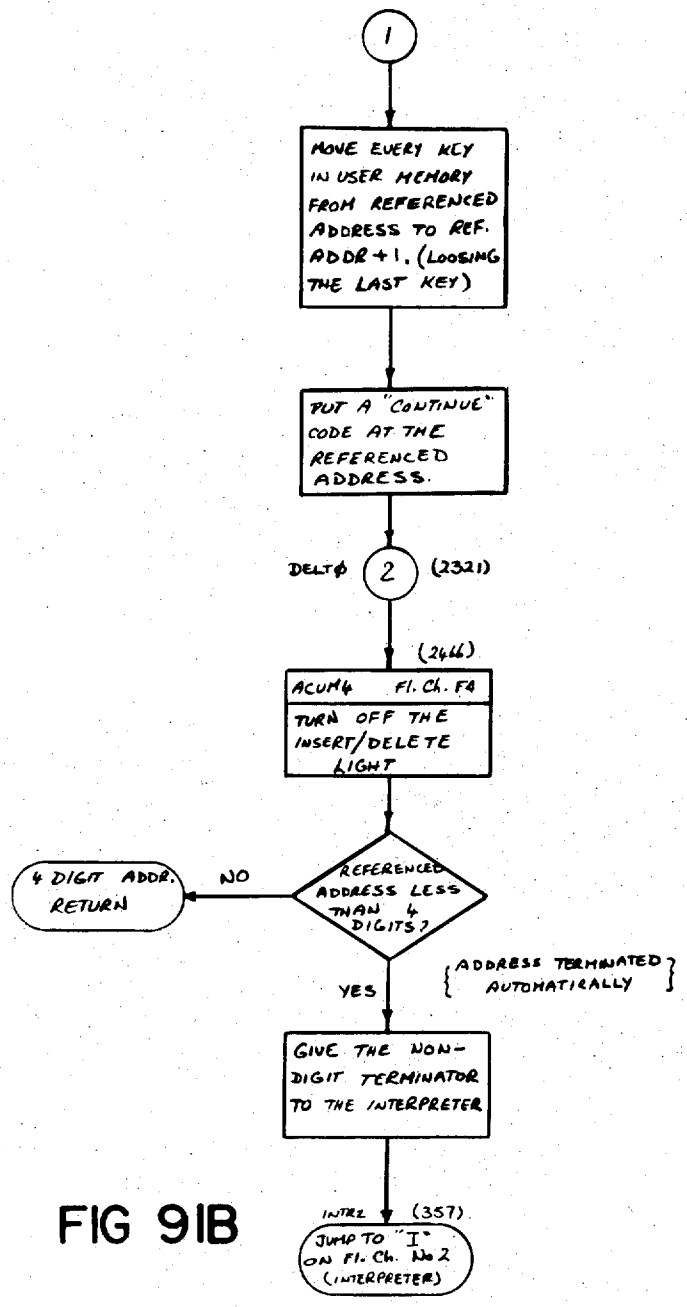


FIG 91B

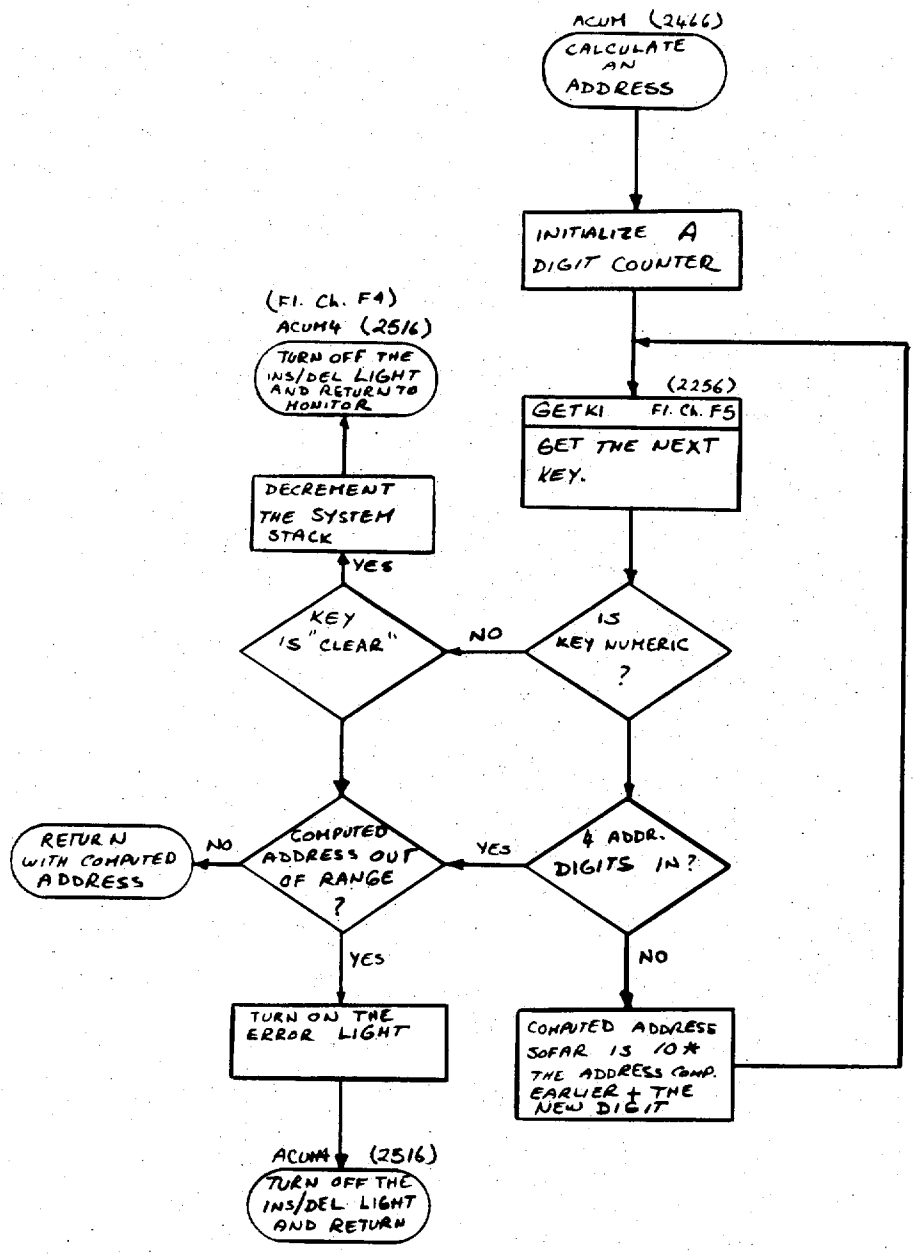
INSERT PROCESSING

LOCAL DIRECTIVE TABLE ENTRY

No. 2066

(KEYCODE 66)

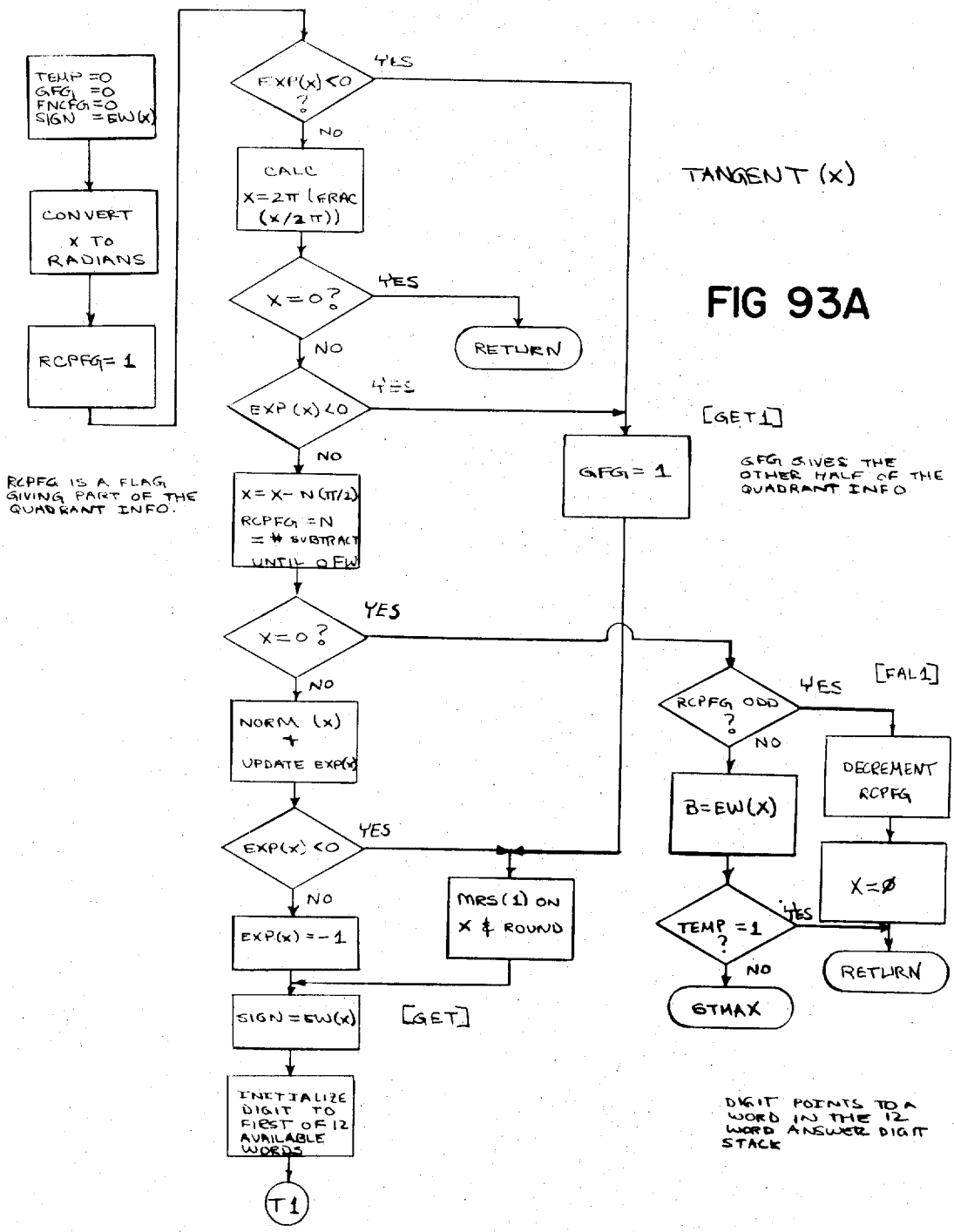
USER DEFINABLE FUNCTION BLOCK

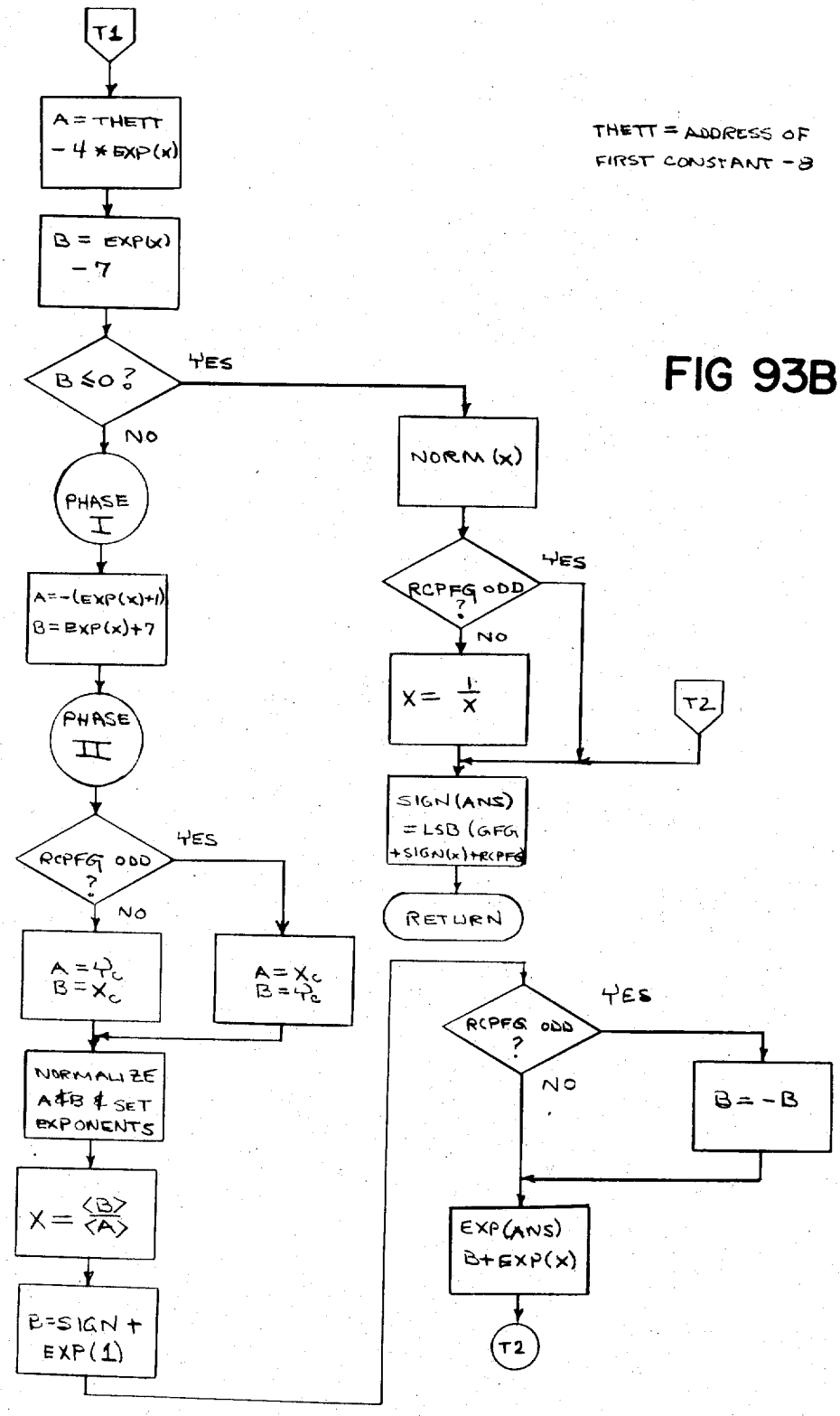


COMPUTING A REFERENCED ADDRESS

FIG 92

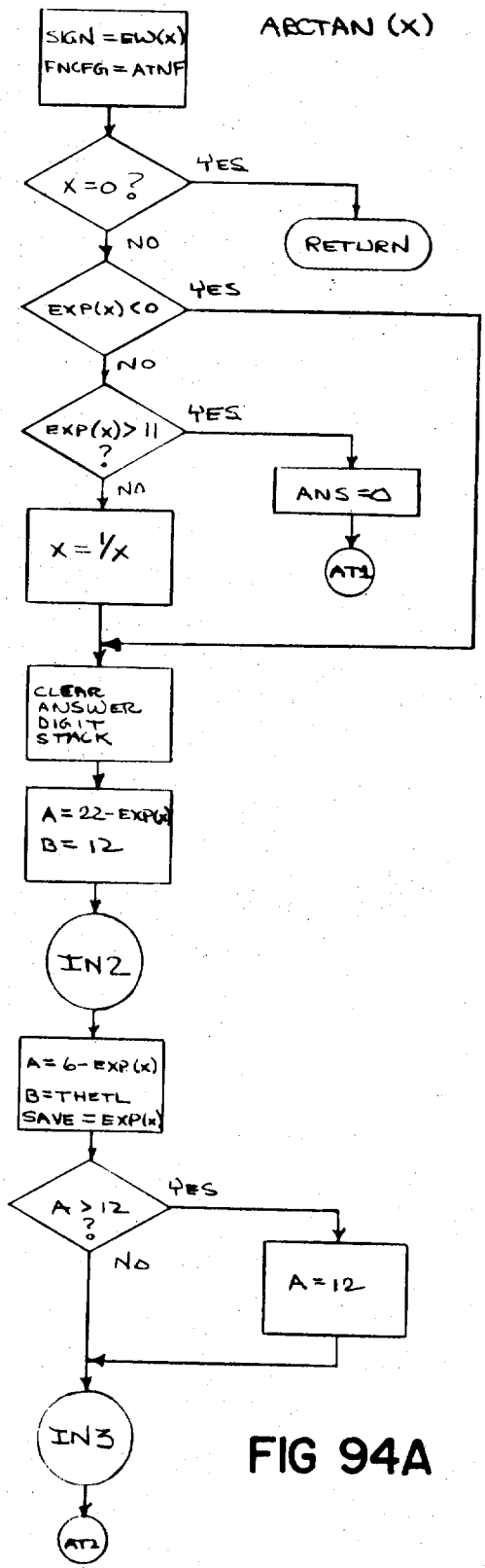
USER DEFINABLE FUNCTION BLOCK


TEMP = 0
GFG1 = 0
FNCFG1 = 0
SIGN = EW(X)
CONVERT X TO RADIANS
RCPFG = 1
EXP(X) < 0 ?
YES
NO
CALC X = 2π (FRAC (X/2π))
X = 0 ?
YES
RETURN
NO
EXP(X) < 0
YES
NO
X = X - N(π/2)
RCPFG = N = # SUBTRACT UNTIL 0 FW
X = 0 ?
YES
NO
NORM (X) + UPDATE EXP(X)
EXP(X) < 0
YES
NO
EXP(X) = -1
MRS(1) ON X & ROUND
SIGN = EW(X)
[GET]
INITIALIZE DIGIT TO FIRST OF 12 AVAILABLE WORDS
T1
[GET1]
GFG1 = 1
RCPFG ODD ?
YES
[FAL1]
NO
B = EW(X)
DECREMENT RCPFG
X = Ø
TEMP = 1 ?
YES
NO
RETURN
GTHAX
TANGENT (X)
FIG 93A
RCPFG IS A FLAG GIVING PART OF THE QUADRANT INFO.
GFG GIVES THE OTHER HALF OF THE QUADRANT INFO
DIGIT POINTS TO A WORD IN THE 12 WORD ANSWER DIGIT STACK



THETT = ADDRESS OF FIRST CONSTANT - 8

FIG 93B



ATNF IS THE ARCTAN FLAG

THETL = ADDRESS OF LAST CORDIC CONSTANT -4

FIG 94A

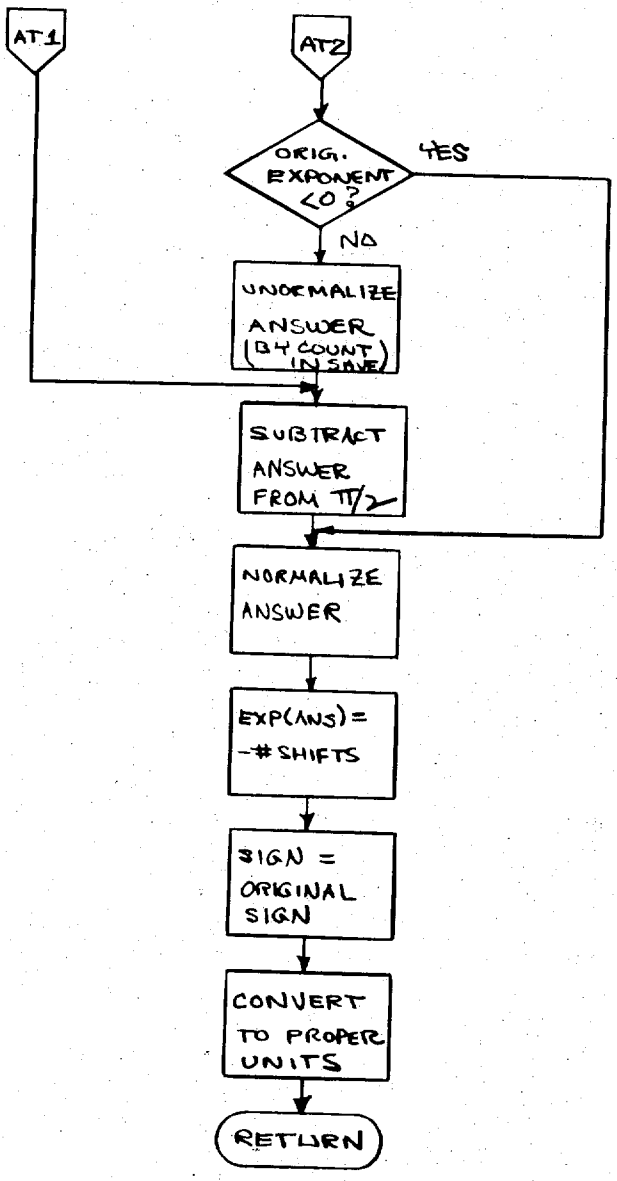
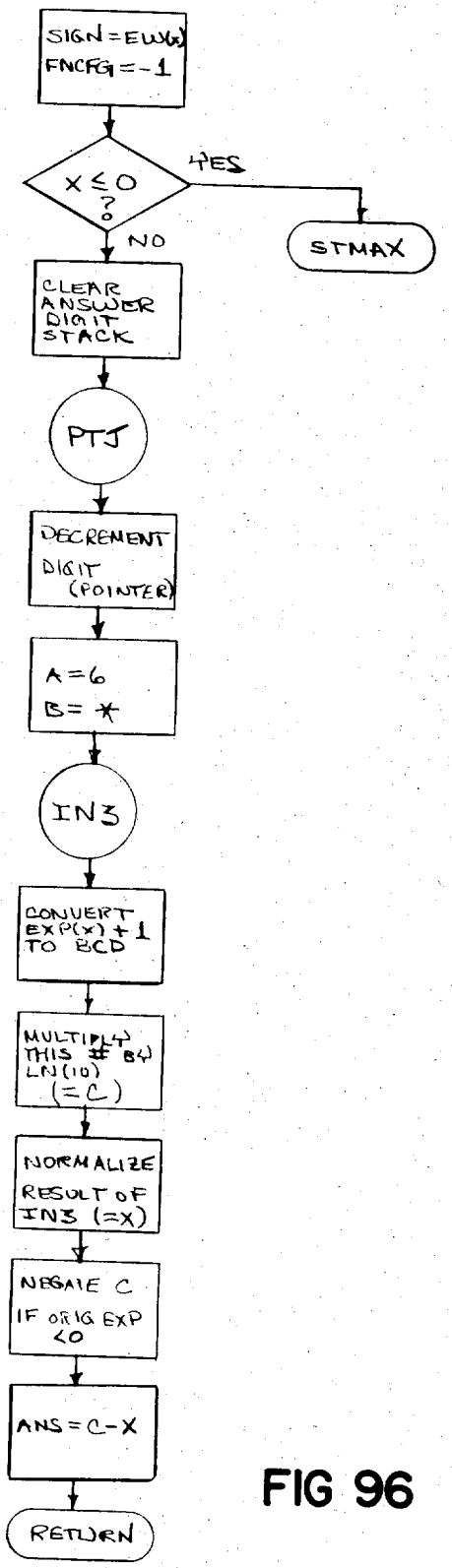

AT 1
AT 2
ORIG. EXPONENT <0 ?
YES
NO
UNNORMALIZE ANSWER (B4 COUNT IN SAVE)
SUBTRACT ANSWER FROM TT/2
NORMALIZE ANSWER
EXP(ANS) = -# SHIFTS
SIGN = ORIGINAL SIGN
CONVERT TO PROPER UNITS
RETURN

FIG 94B

FIG 95A

FIG 95B

LN (X)



FNCFG = FUNCTION FLAG

DIGIT IS A POINTER FOR THE ANSWER DIGIT STACK

* IS ADDRESS OF LAST CORDIC CONSTANT - 4

FIG 96

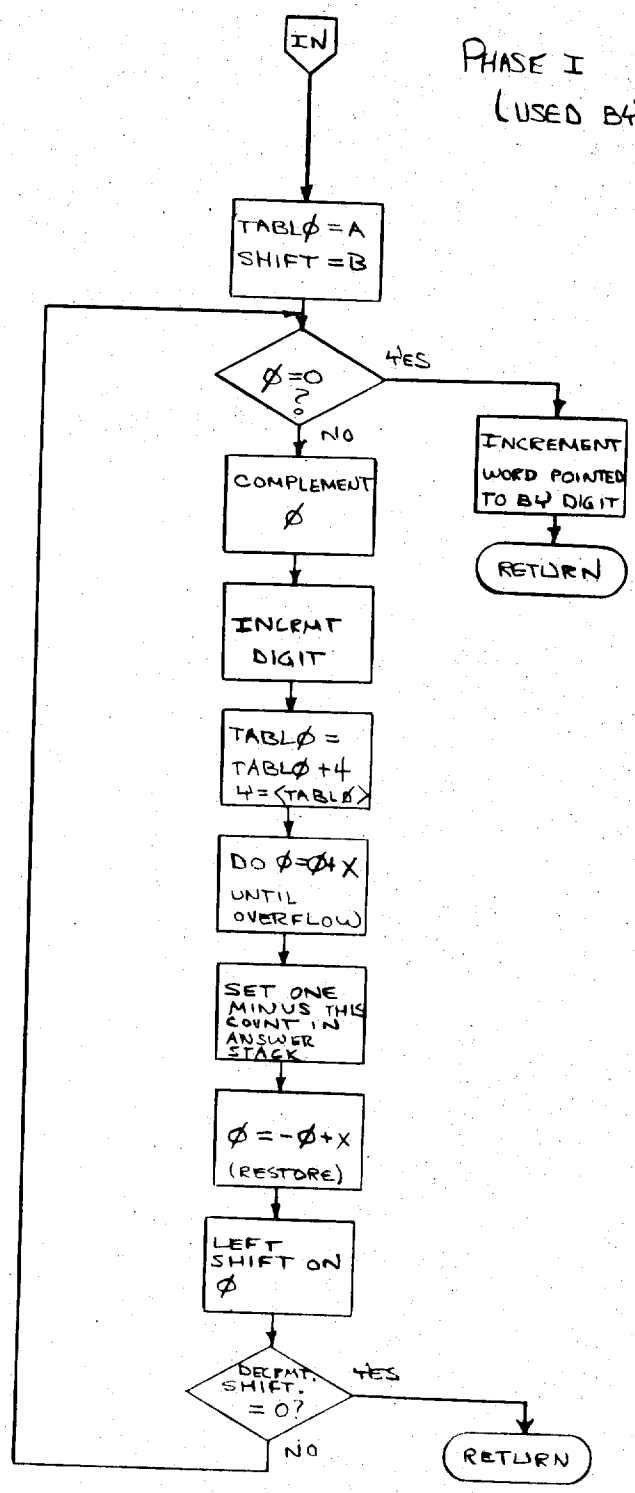
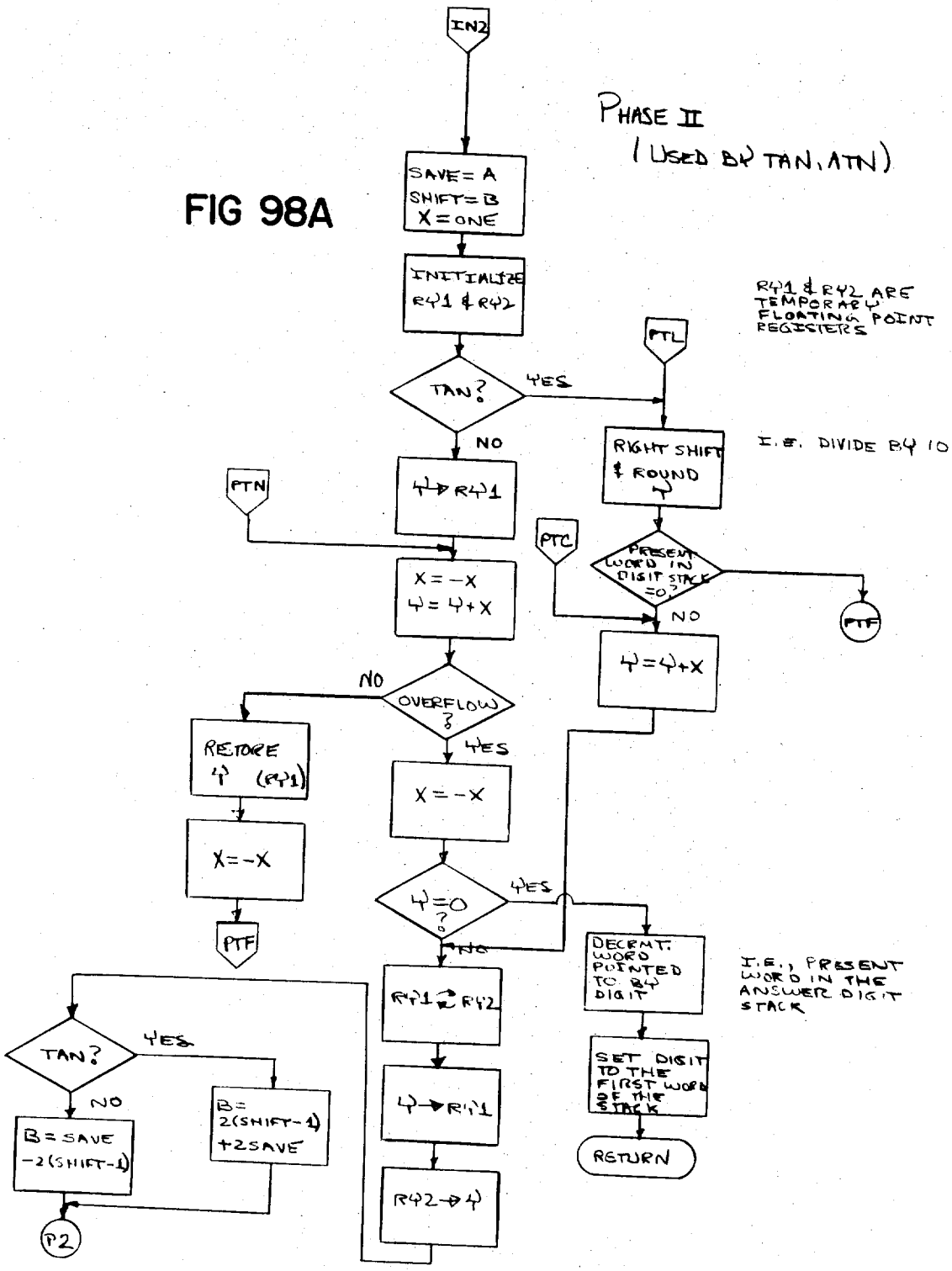

IN
PHASE I
(USED BY TAN, EXP)
TABLØ = A
SHIFT = B
TABLØ HOLDS THE ADDRESS OF A CORDIC CONSTANT
SHIFT = # TIMES TO GO THRU THE MAIN LOOP
Ø = 0 ?
YES
NO
INCREMENT WORD POINTED TO BY DIGIT
RETURN
COMPLEMENT Ø
INCRMT DIGIT
TABLØ = TABLØ + 4
4 = <TABLØ>
SET X = TO NEXT CORDIC CONSTANT
DO Ø = Ø + X UNTIL OVERFLOW)
SET ONE MINUS THIS COUNT IN ANSWER STACK
THE COUNT IS LOCATED IN THE STACK VIA DIGIT
Ø = -Ø + X (RESTORE)
LEFT SHIFT ON Ø
I.E., MULTIPLY Ø BY 10
DECRMT. SHIFT. = 0?
YES
NO
RETURN

FIG 97

FIG 98A


IN2
PHASE II
(USED BY TAN, ATN)
SAVE = A
SHIFT = B
X = ONE
INITIALIZE RY1 & RY2
R4Y1 & RY2 ARE TEMPORARY FLOATING POINT REGISTERS
TAN?
YES
NO
PTL
RIGHT SHIFT & ROUND Y
I.E. DIVIDE BY 10
Y → R4Y1
PTN
PTC
X = -X
Y = Y + X
PRESENT WORD IN DIGIT STACK = 0?
PTF
NO
Y = Y + X
OVERFLOW?
NO
YES
RESTORE Y (RY1)
X = -X
X = -X
Y = 0?
YES
NO
PTF
DECRMT. WORD POINTED TO BY DIGIT
I.E., PRESENT WORD IN THE ANSWER DIGIT STACK
R4Y1 ⇄ R4Y2
SET DIGIT TO THE FIRST WORD OF THE STACK
TAN?
YES
NO
B = 2(SHIFT-1) + 2SAVE
B = SAVE - 2(SHIFT-1)
R4Y1 → Y
RETURN
R4Y2 → Y
P2

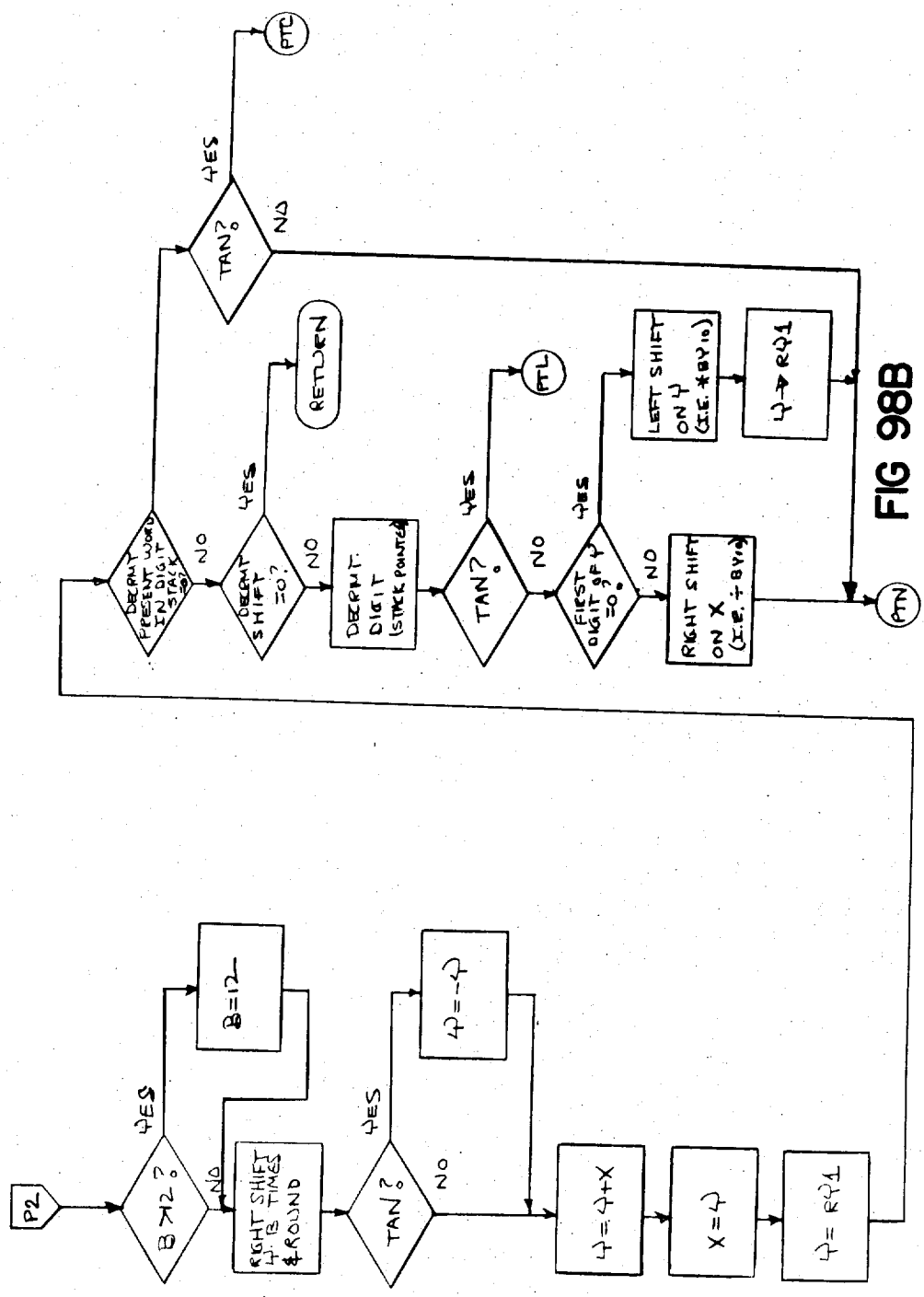
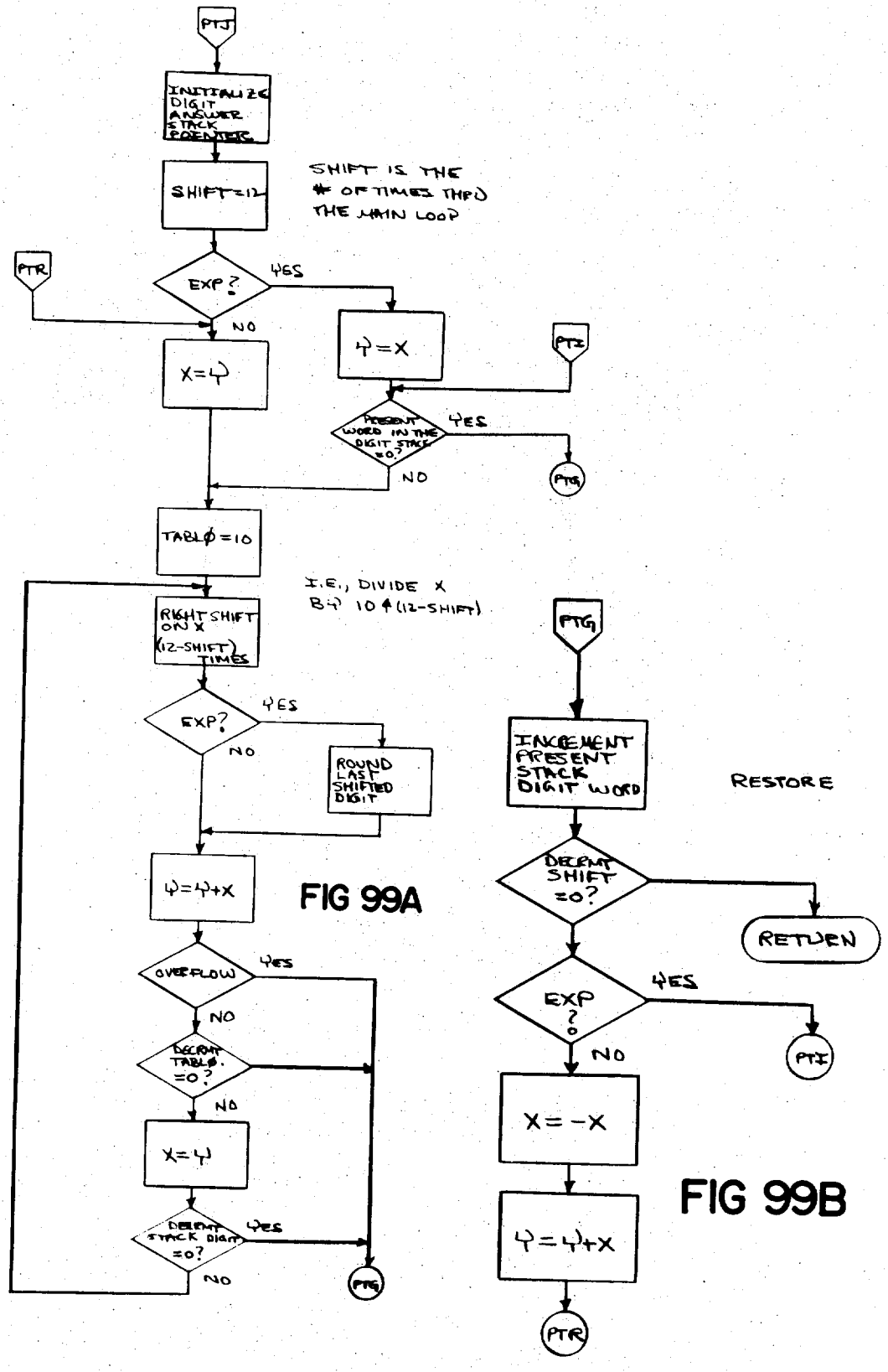

P2
B>12?
YES
NO
B=12
RIGHT SHIFT Y, B TIMES & ROUND
TAN?
YES
NO
Y=-Y
Y=Y+X
X=Y
Y=RY1
PTC
TAN?
YES
NO
RETURN
DECRMT PRESENT WORD IN DIGIT STACK
DECRMT SHIFT =0?
DECRMT DIGIT (STACK POINTER)
PTL
TAN?
FIRST DIGIT OF Y =0?
LEFT SHIFT ONLY (I.E. *BY10)
Y→RY1
RIGHT SHIFT ON X (I.E. ÷BY10)
PTN

FIG 98B


PTJ
INITIALIZE DIGIT ANSWER STACK POINTERS
SHIFT=12
SHIFT IS THE # OF TIMES THRU THE MAIN LOOP
PTR
EXP?
YES
NO
Y=X
PTZ
X=Y
PRESENT WORD IN THE DIGIT STACK =0?
YES
NO
PTG
TABLØ=10
I.E., DIVIDE X BY 10↑(12-SHIFT)
RIGHT SHIFT ON X (12-SHIFT) TIMES
EXP?
YES
NO
ROUND LAST SHIFTED DIGIT
Y=Y+X
OVERFLOW
YES
NO
DECRMT TABLØ. =0?
NO
X=Y
DECRMT STACK DIGIT =0?
YES
NO
PTG
FIG 99A
PTG
INCREMENT PRESENT STACK DIGIT WORD
RESTORE
DECRMT SHIFT =0?
RETURN
EXP ?
YES
NO
PTI
X=-X
Y=Y+X
PTR
FIG 99B

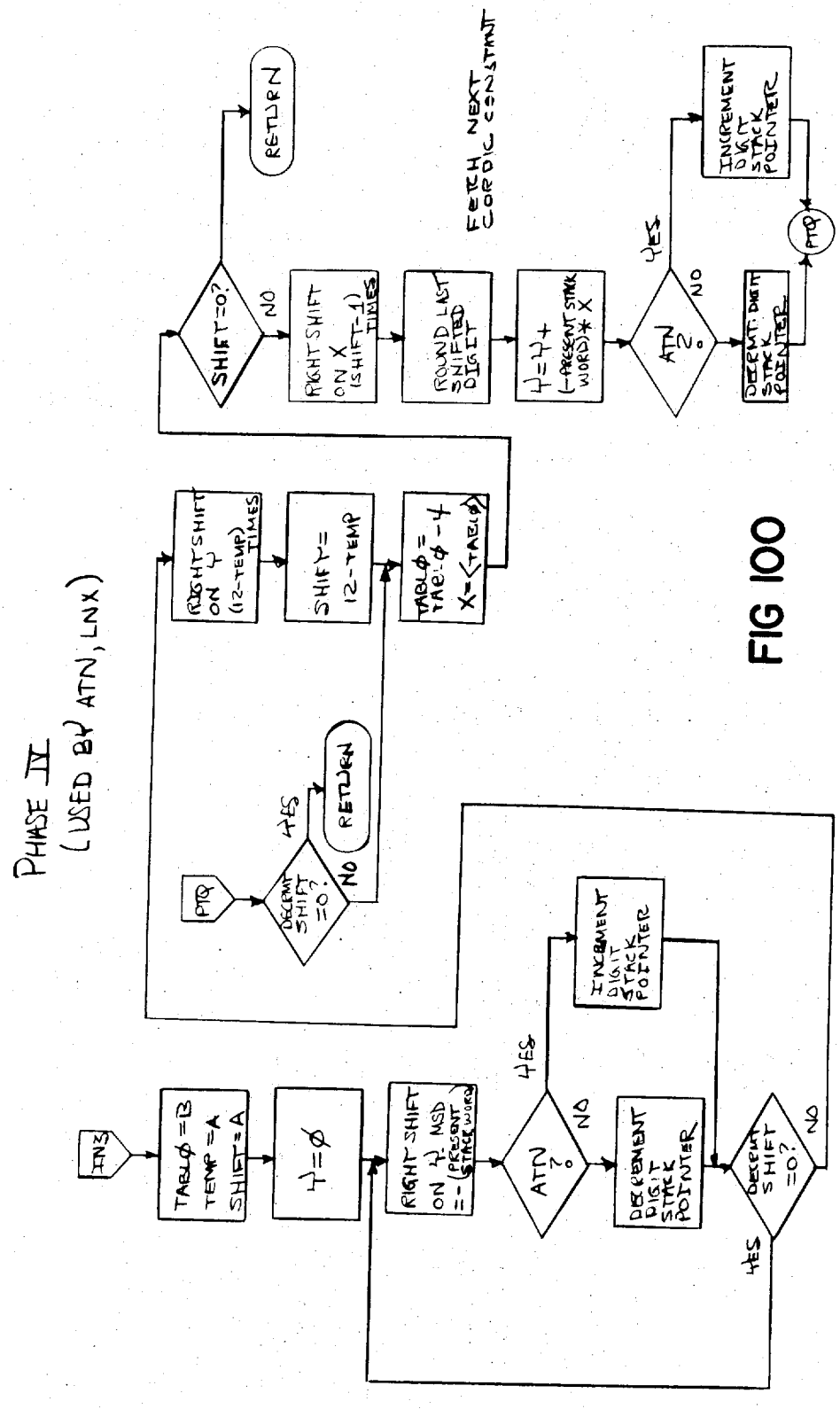
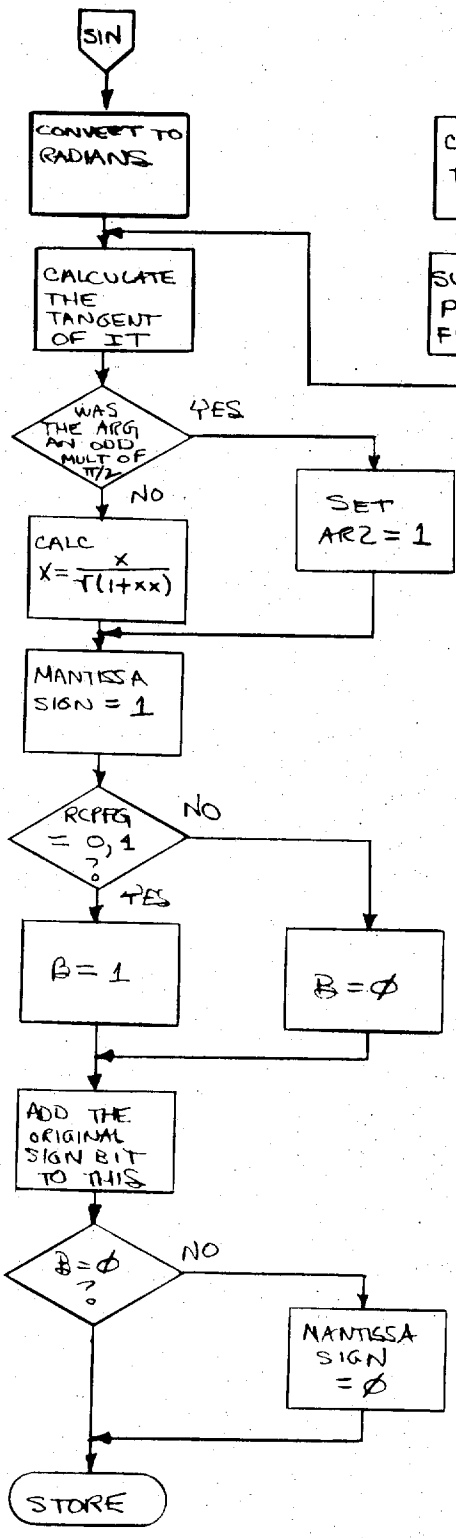

PHASE IV
(USED BY ATN, LNX)
INS
TABLΦ = B
TEMP = A
SHIFT = A
Y = Φ
RIGHT SHIFT ON Y NSD = - (PRESENT STACK WORD)
ATN ?
YES
NO
INCREMENT DIGIT STACK POINTER
DECREMENT DIGIT STACK POINTER
DECRMT SHIFT =0?
YES
NO
PTR
DECRMT SHIFT =0?
YES
NO
RETURN
RIGHT SHIFT ON Y (12-TEMP) TIMES
SHIFT = 12-TEMP
TABLΦ = TABLΦ - 4
X = <TABLΦ>
FETCH NEXT CORDIC CONSTANT
SHIFT=0?
NO
YES
RETURN
RIGHT SHIFT ON X (SHIFT-1) TIMES
ROUND LAST SHIFTED DIGIT
Y = Y + (PRESENT STACK WORD) * X
ATN ?
YES
NO
INCREMENT DIGIT STACK POINTER
DECRMT DIGIT STACK POINTER
PTR

FIG 100

SINE AND COSINE ROUTINES



RCPFG IS THE NO. OF QUARTER CIRCLE SUBTRACTS FROM THE TANGENT ROUTINE

FIG 101

TABLE f(x) AND ARC

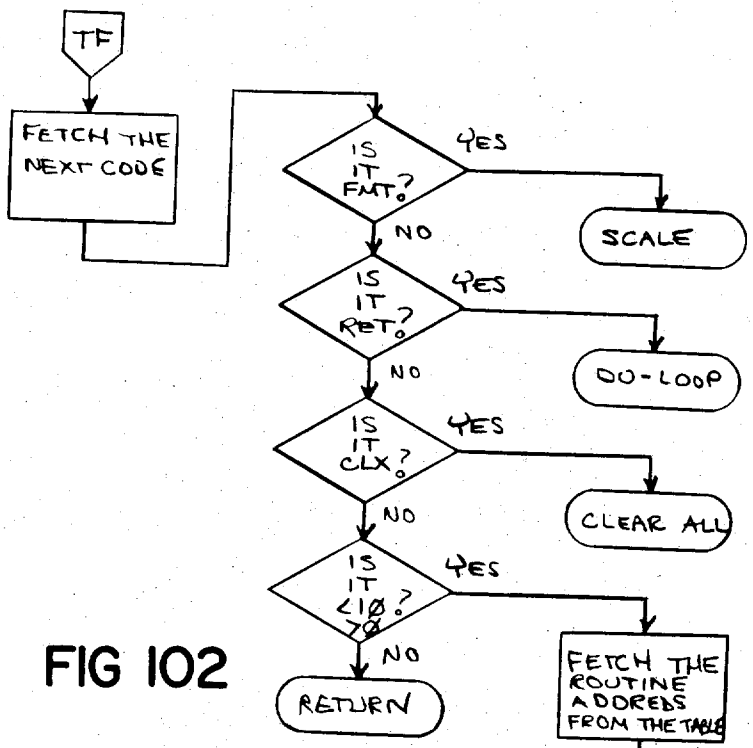


FIG 102

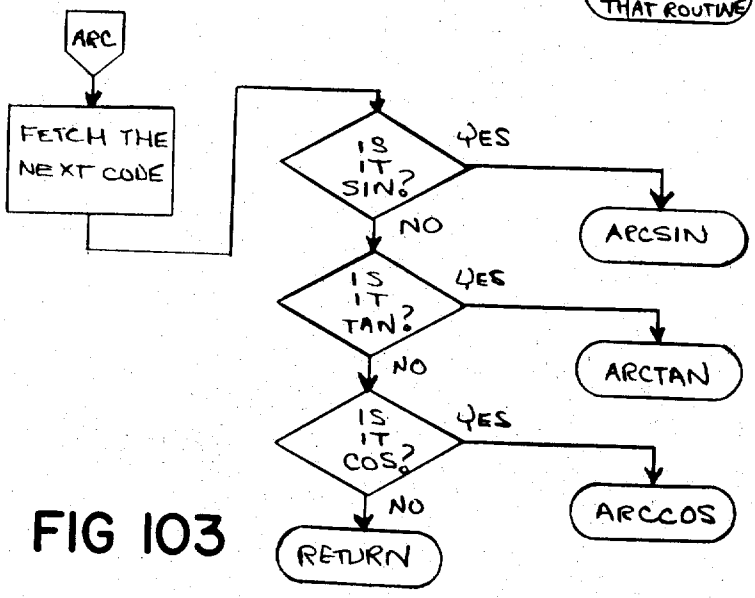


FIG 103

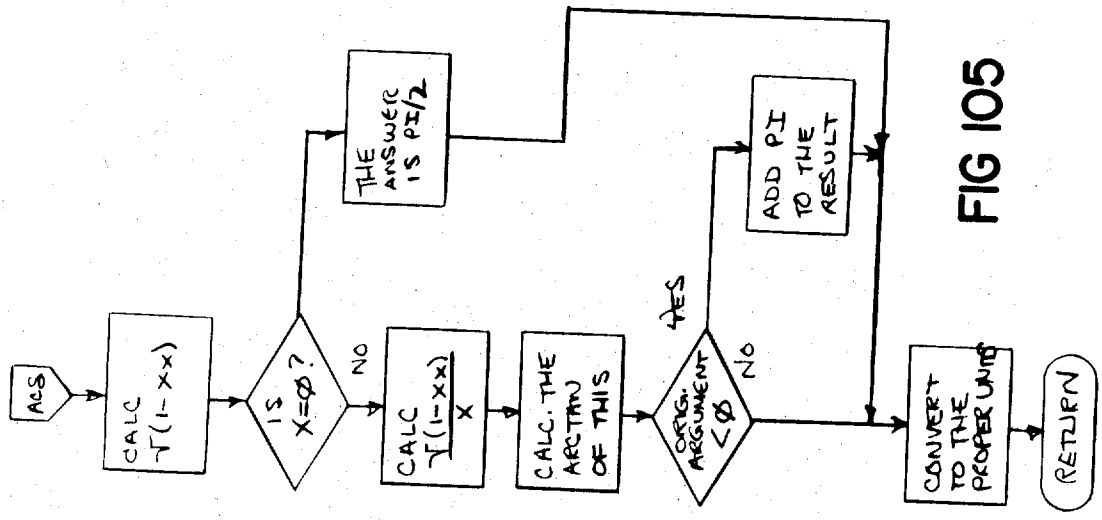


FIG 105

ARCSINE AND ARCCOSINE

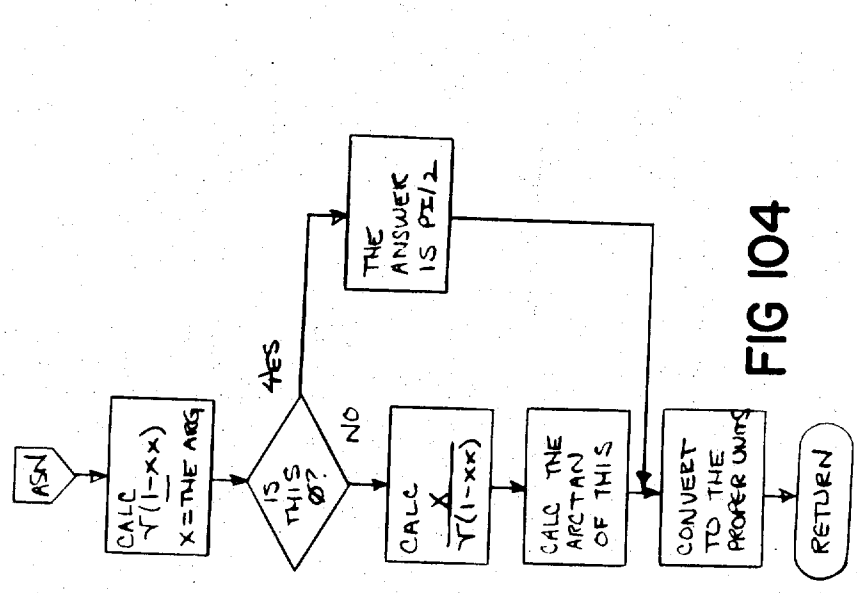


FIG 104

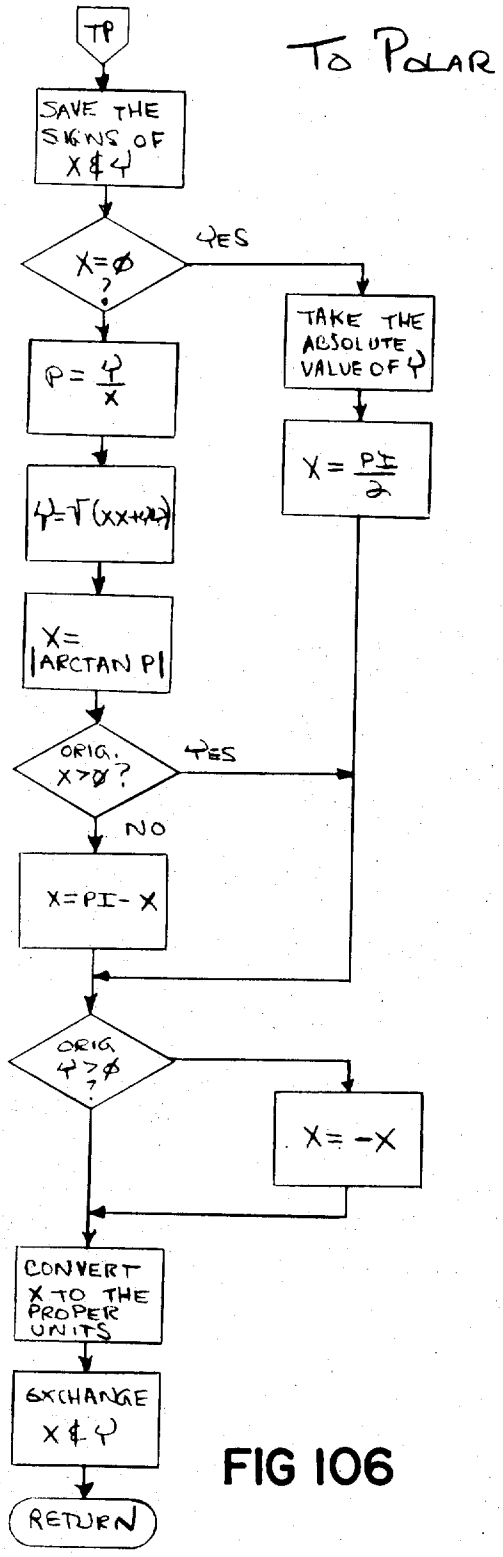

TP
To Polar
SAVE THE SIGNS OF X & Y
X=Ø ?
YES
TAKE THE ABSOLUTE VALUE OF Y
P = Y/X
X = PI/2
Y=T(XX+YY)
X= |ARCTAN P|
ORIG. X>Ø ?
YES
NO
X=PI- X
ORIG Y>Ø ?
X= -X
CONVERT X TO THE PROPER UNITS
EXCHANGE X & Y
RETURN

FIG 106

TO RETANGULAR AND 10^X AND |Y|

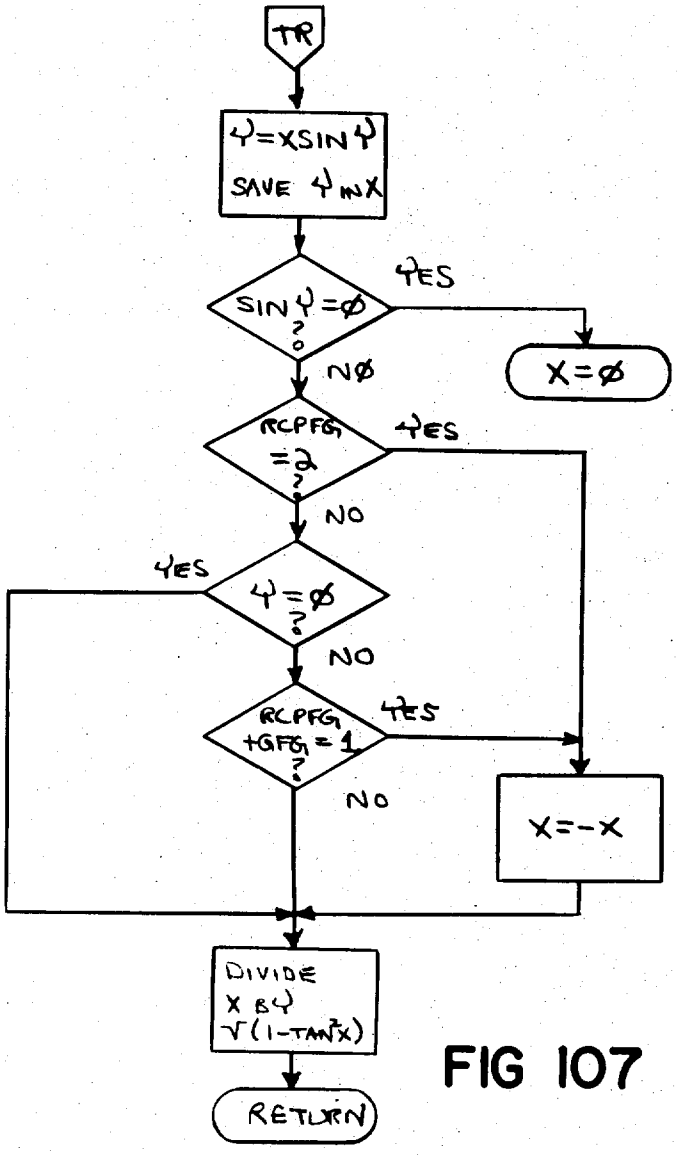


FIG 107

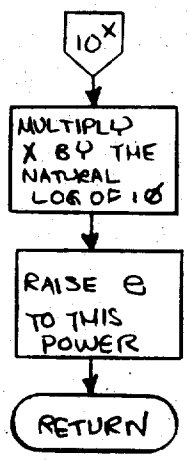


FIG 108

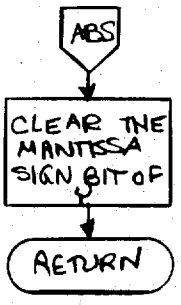


FIG 109

X^Y AND LOG_{10}

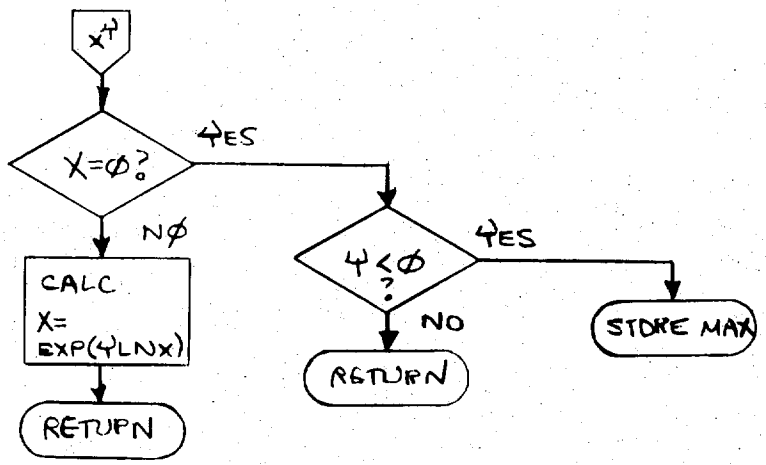
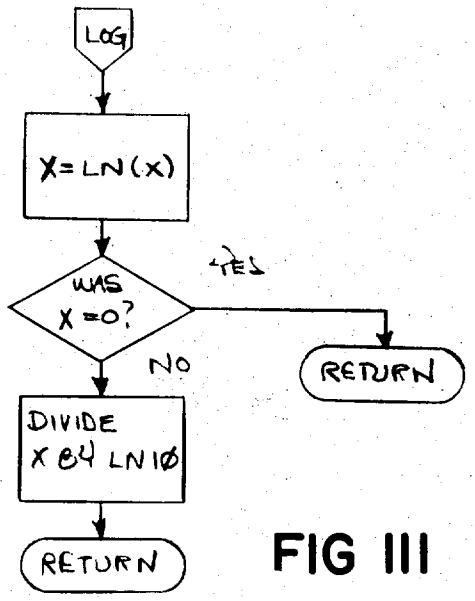


FIG 110



(WITH -9.999 999 999 E98)

FIG 111

DEGREES, MINUTES, SECONDS ⇄ DEGREES

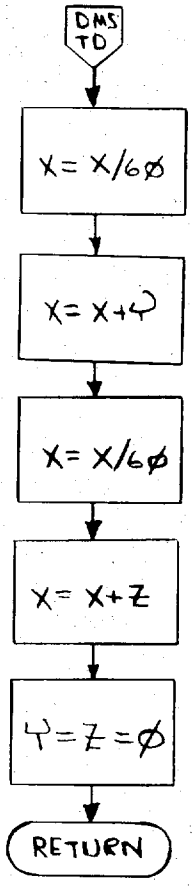


FIG 112

FIG 113

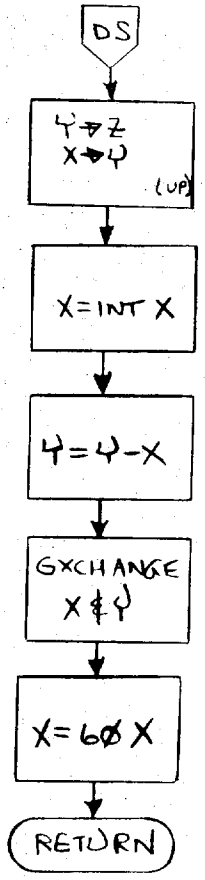


FIG 114

X FACTORIAL, ACCUMULATE PLUS & MINUS, RECALL

FIG 115

FIG 116

FIG 117

FIG 118

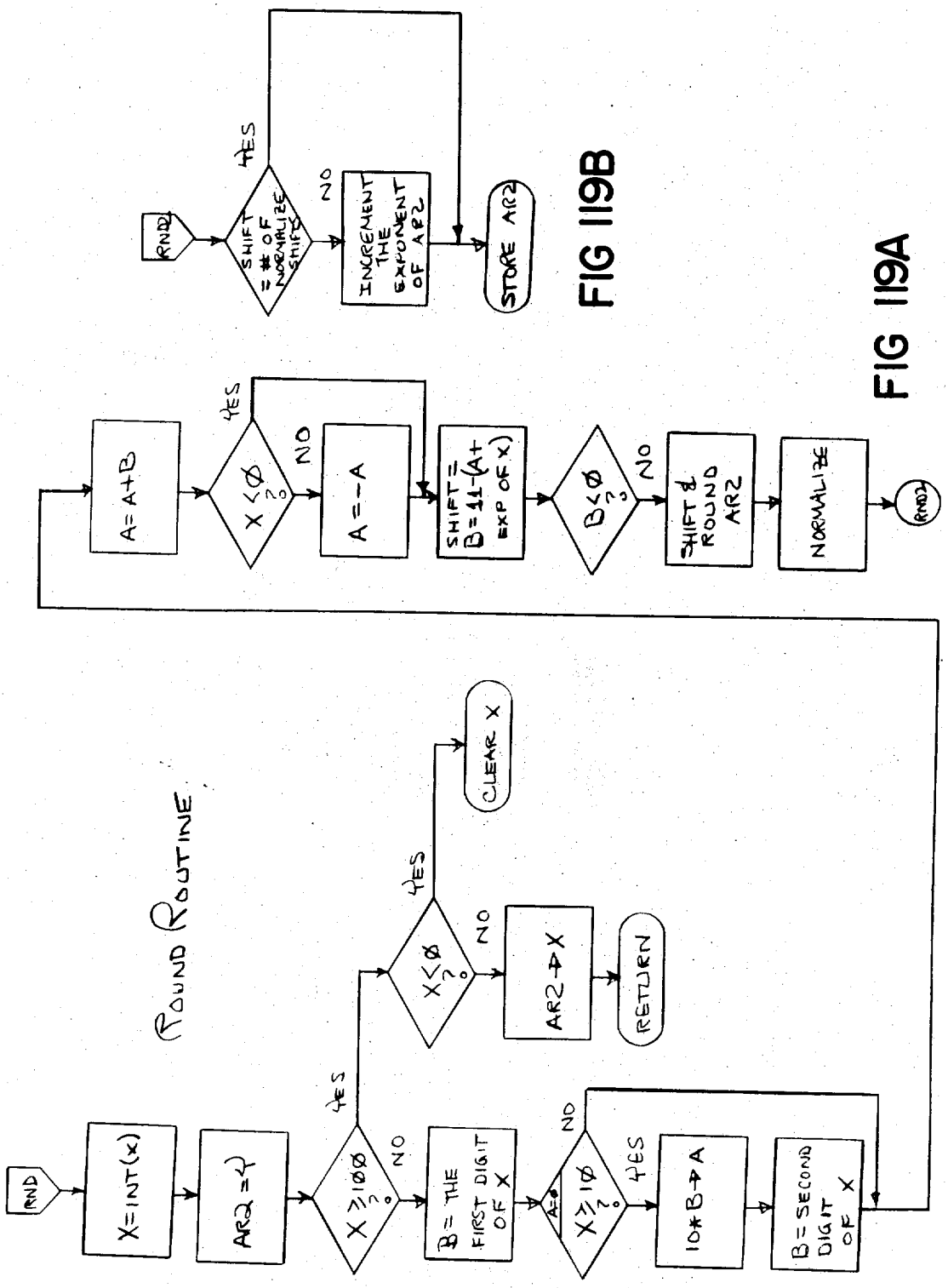


FIG 119A

FIG 119B

SCALE & CLEAR ROUTINES

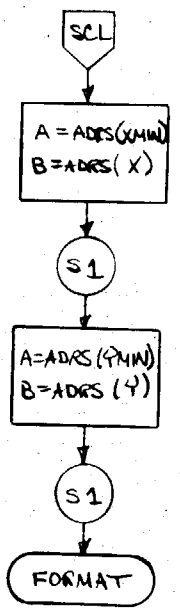


FIG 120

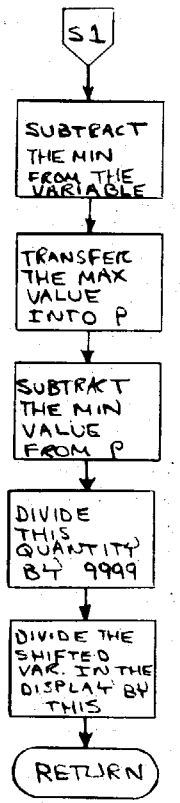


FIG 121

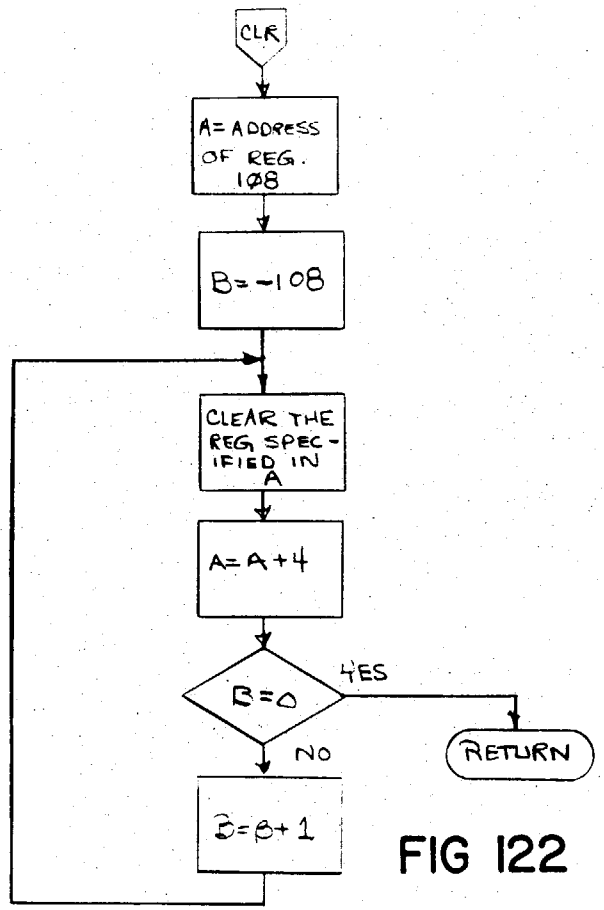


FIG 122

DEFINABLE F(X) AND DO-LOOP

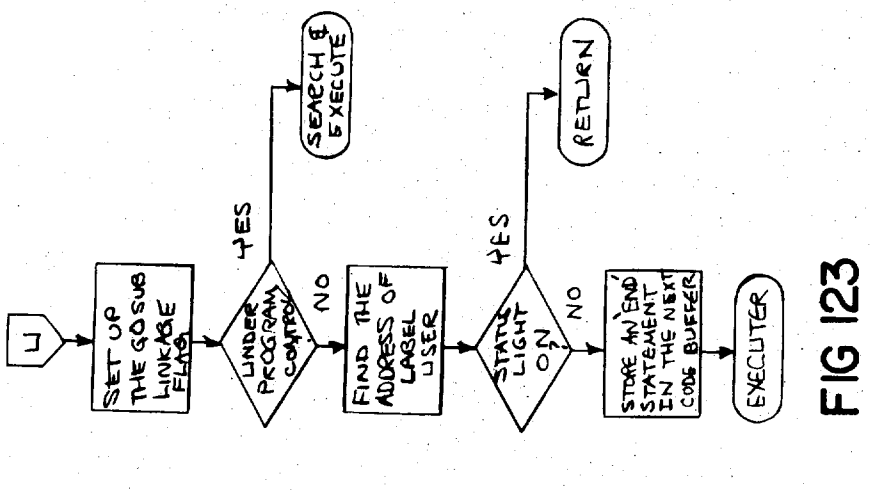


FIG 123

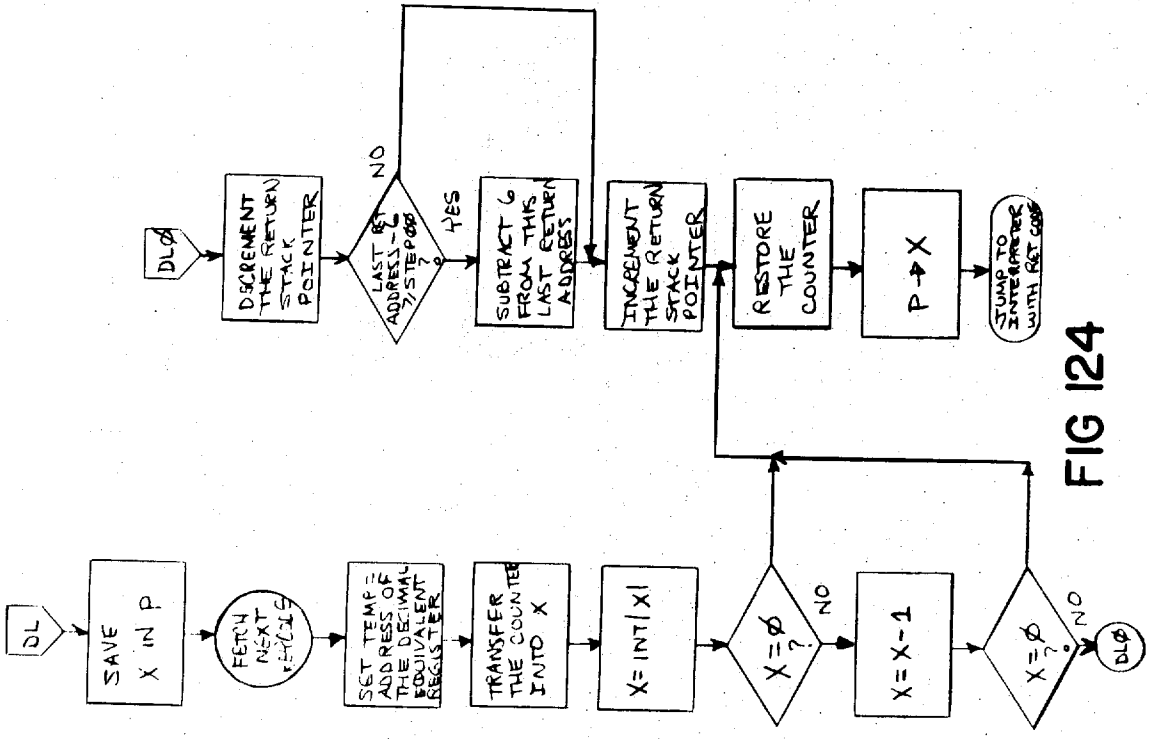


FIG 124

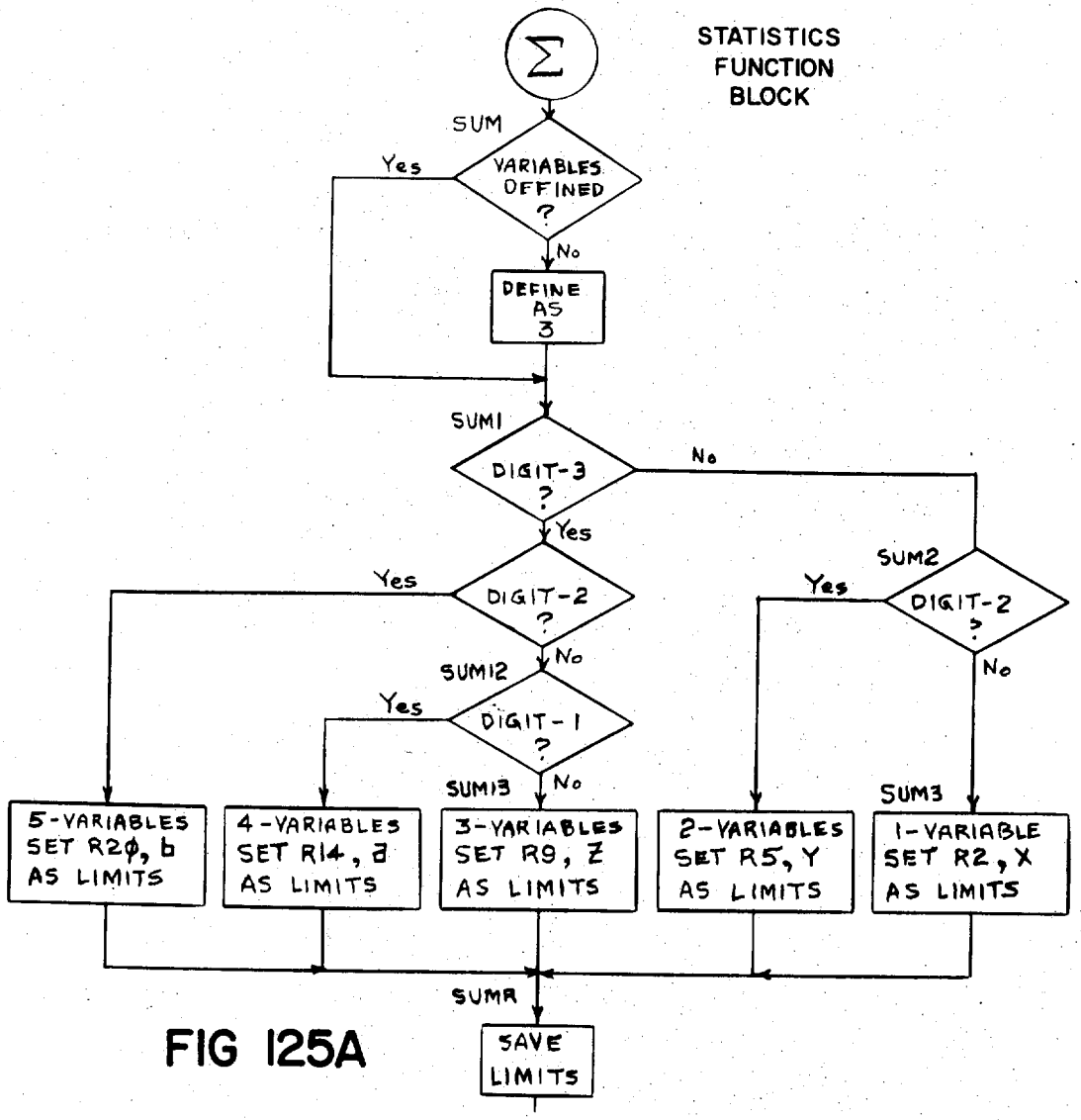

STATISTICS FUNCTION BLOCK
Σ
SUM
VARIABLES DEFINED ?
Yes
No
DEFINE AS 3
SUM1
DIGIT-3 ?
No
Yes
SUM2
DIGIT-2 ?
Yes
No
DIGIT-2 ?
Yes
No
SUM12
DIGIT-1 ?
Yes
No
SUM13
SUM3
5-VARIABLES SET R2ϕ, b AS LIMITS
4-VARIABLES SET R14, d AS LIMITS
3-VARIABLES SET R9, Z AS LIMITS
2-VARIABLES SET R5, Y AS LIMITS
1-VARIABLE SET R2, X AS LIMITS
SUMR
SAVE LIMITS

FIG 125A

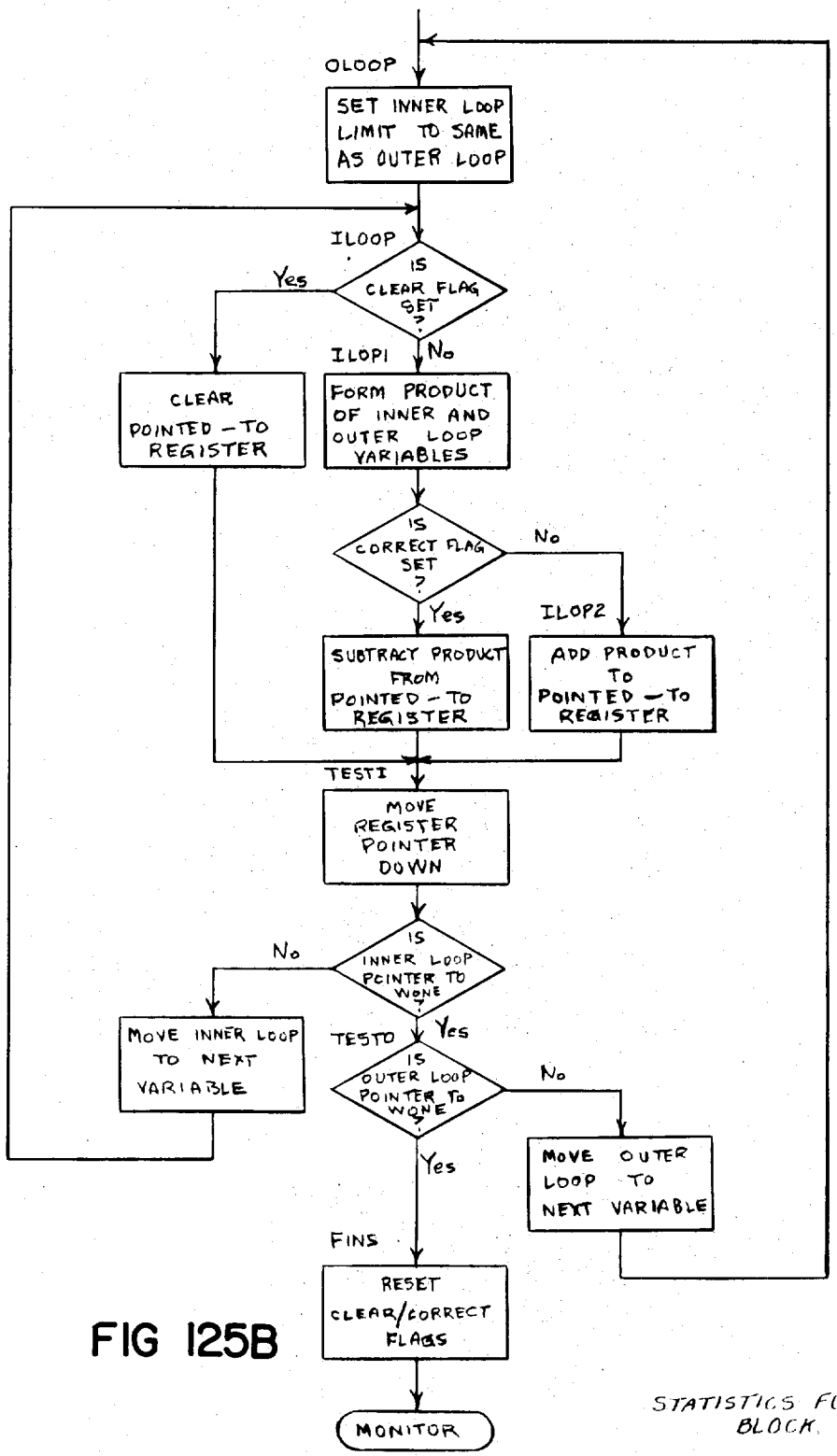


FIG 125B

STATISTICS FUNCTION BLOCK.

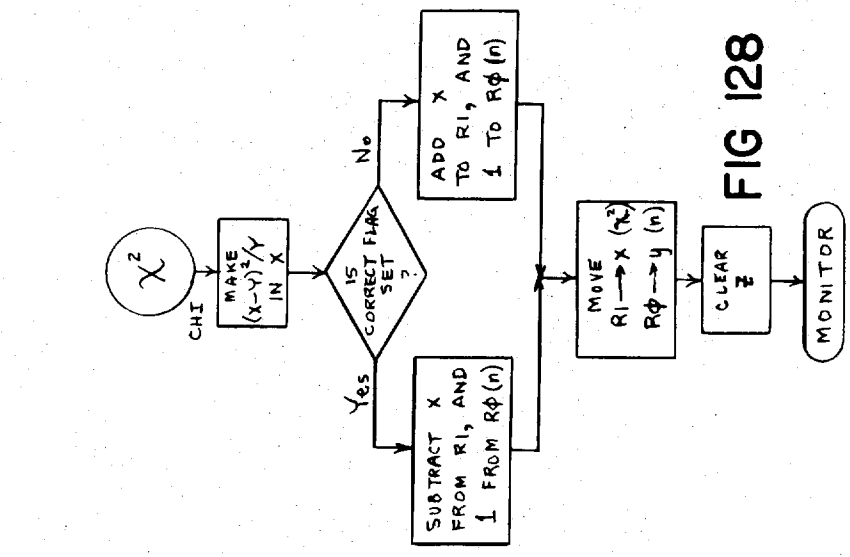


FIG 128

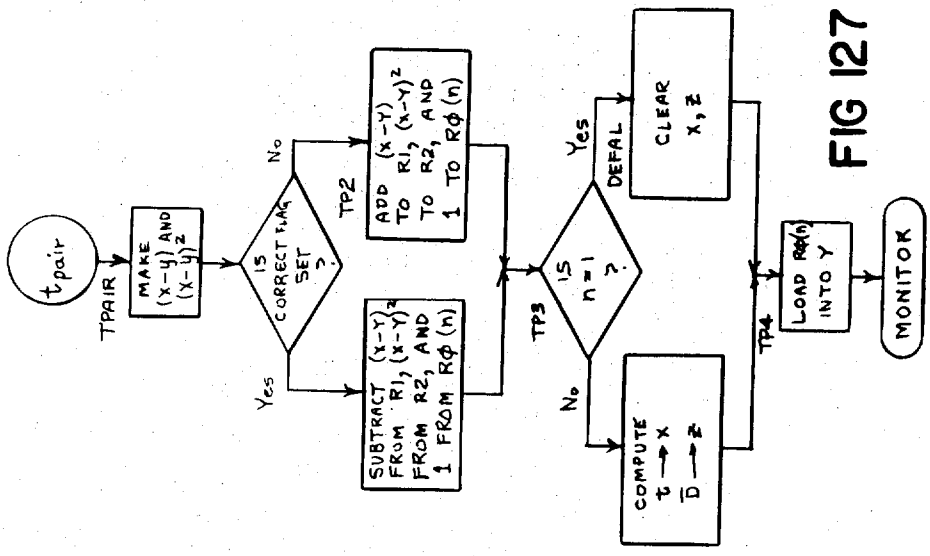


FIG 127

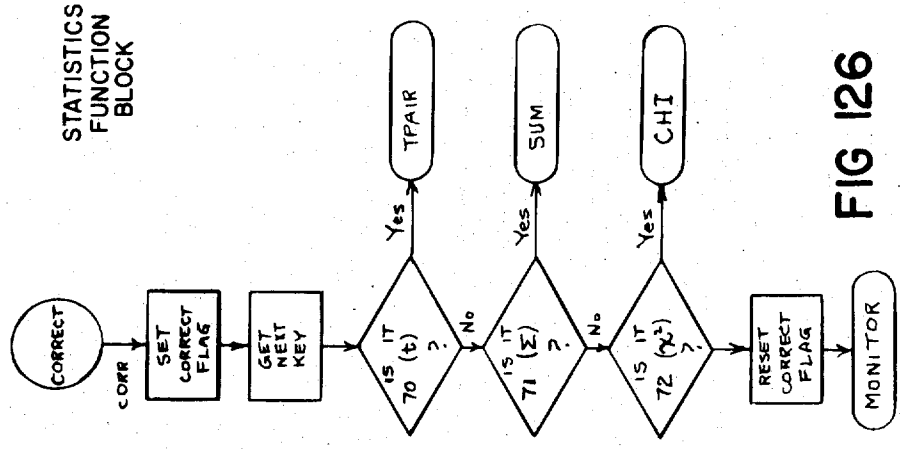


FIG 126

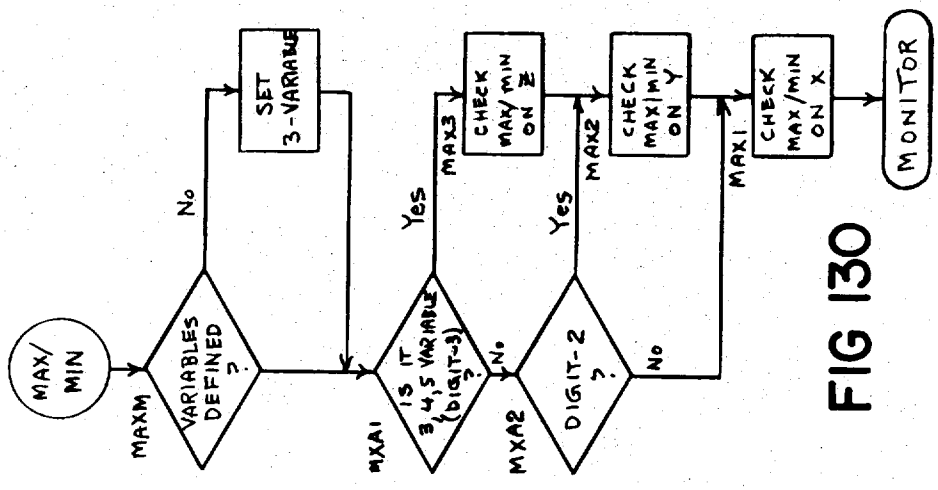


FIG 130

STATISTICS FUNCTION BLOCK

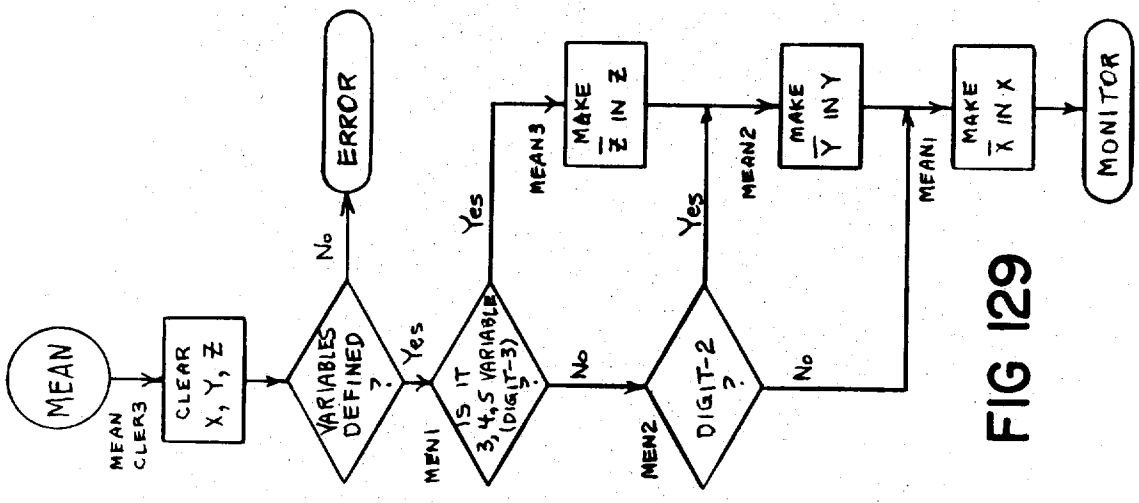


FIG 129

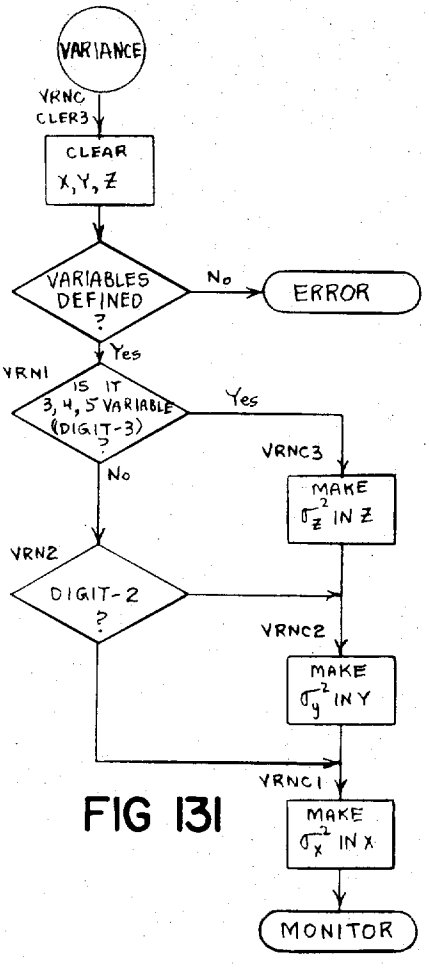


FIG 131

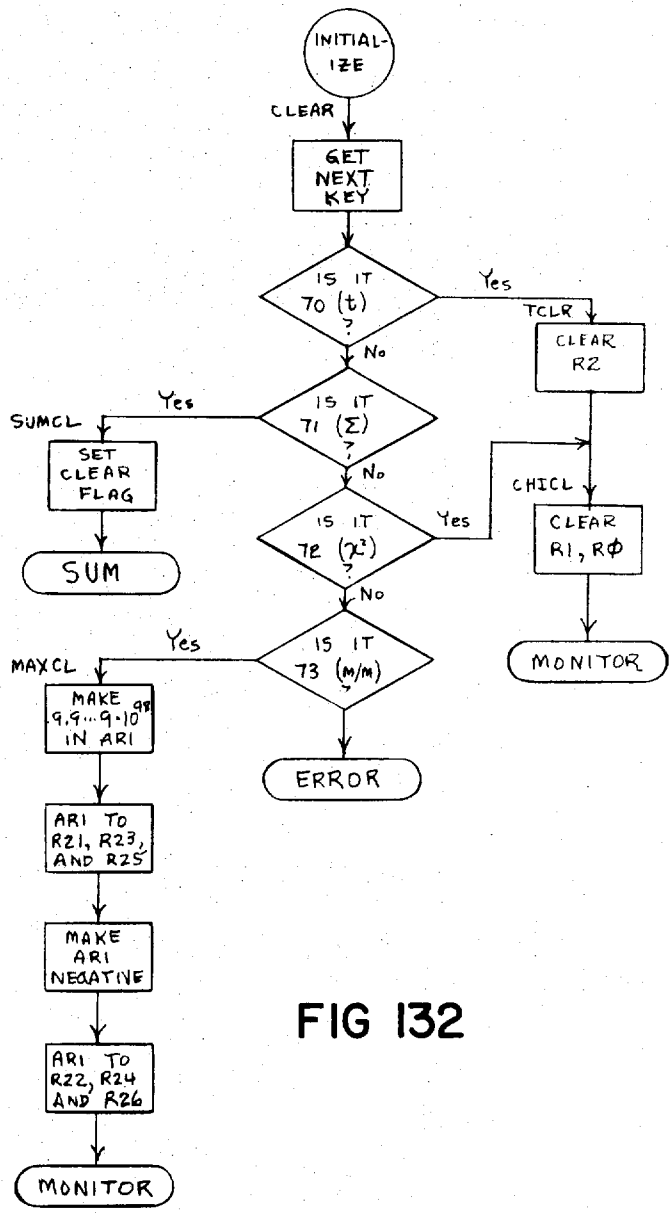


FIG 132

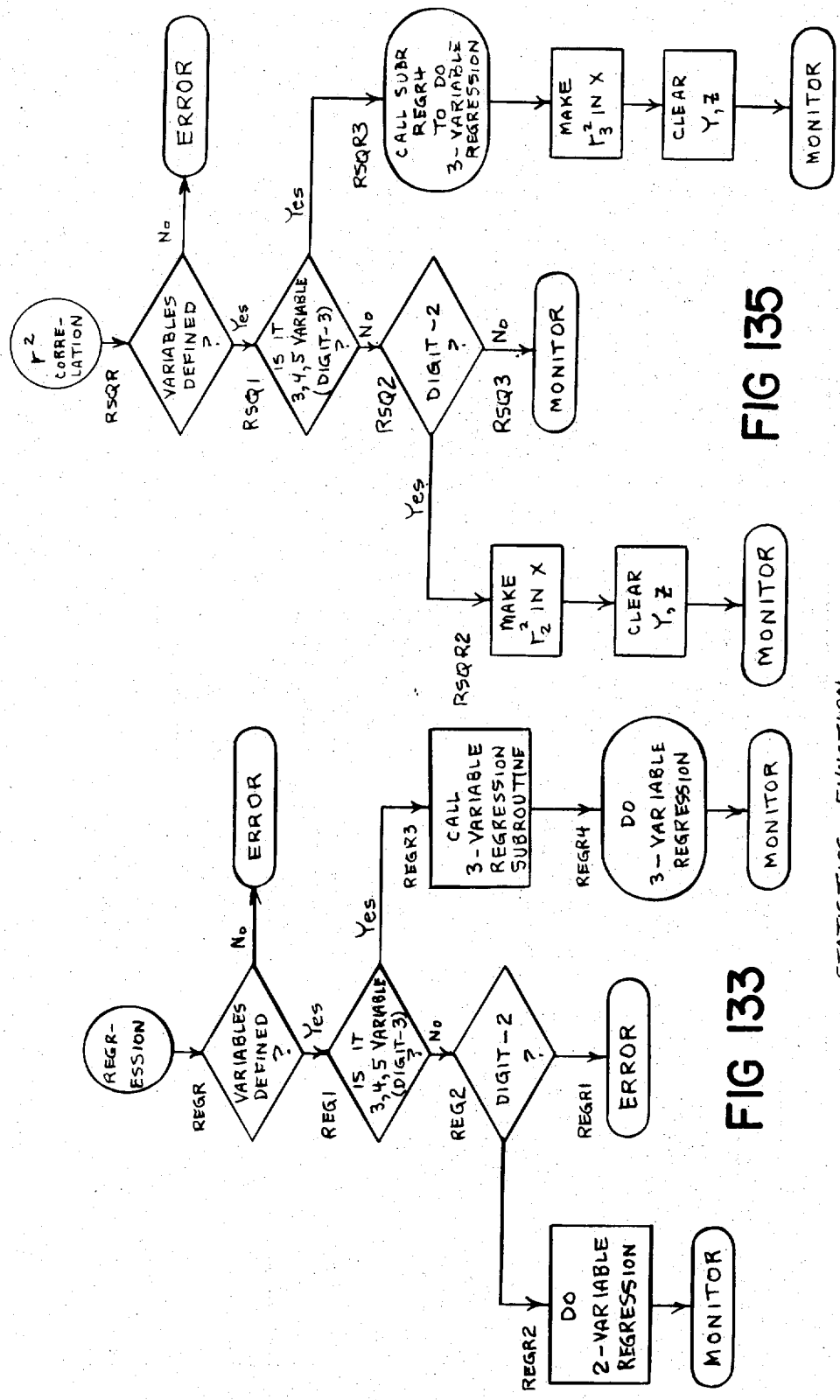


FIG 135

FIG 133

STATISTICS FUNCTION BLOCK

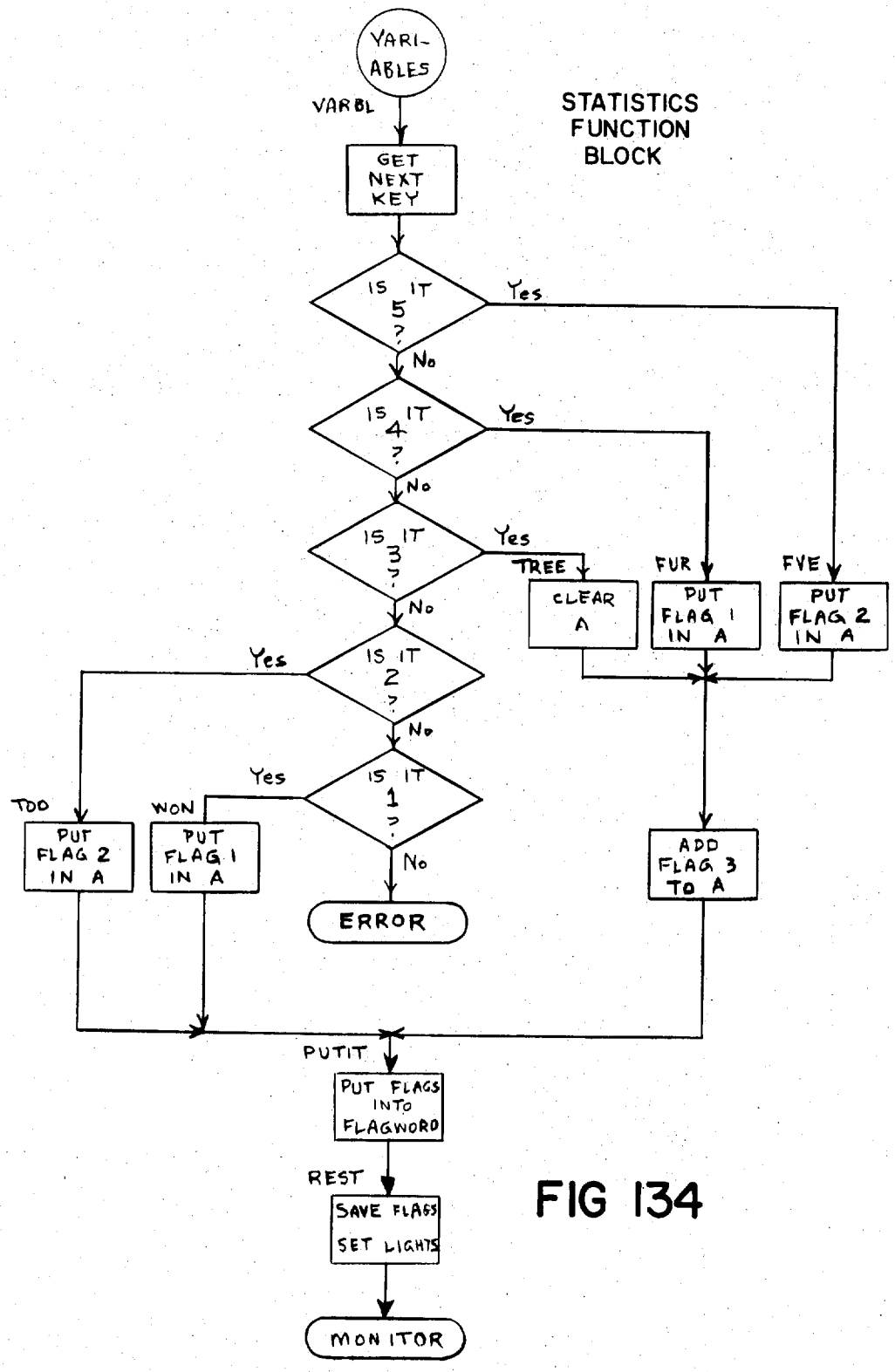
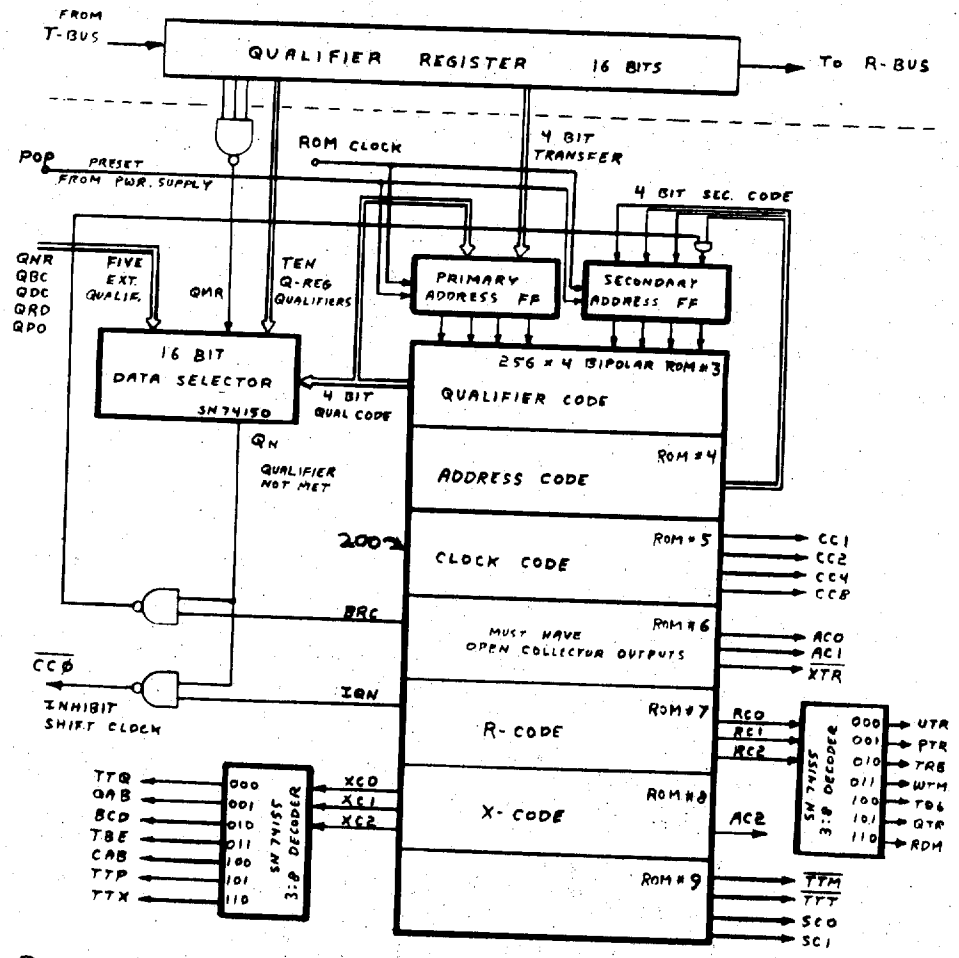

VARI-ABLES
VARBL
GET NEXT KEY
STATISTICS FUNCTION BLOCK
IS IT 5 ?
Yes
No
IS IT 4 ?
Yes
No
IS IT 3 ?
Yes
No
IS IT 2 ?
Yes
No
IS IT 1 ?
Yes
No
TREE
FUR
FVE
CLEAR A
PUT FLAG 1 IN A
PUT FLAG 2 IN A
TOO
WON
PUT FLAG 2 IN A
PUT FLAG 1 IN A
ADD FLAG 3 TO A
ERROR
PUTIT
PUT FLAGS INTO FLAGWORD
REST
SAVE FLAGS SET LIGHTS
MONITOR

FIG 134



ROM ORGANIZATION: 256 WORDS x 28 BITS EXPANDED TO 36 μ-INSTRUCTIONS

FIG 136

MICRO PROCESSOR

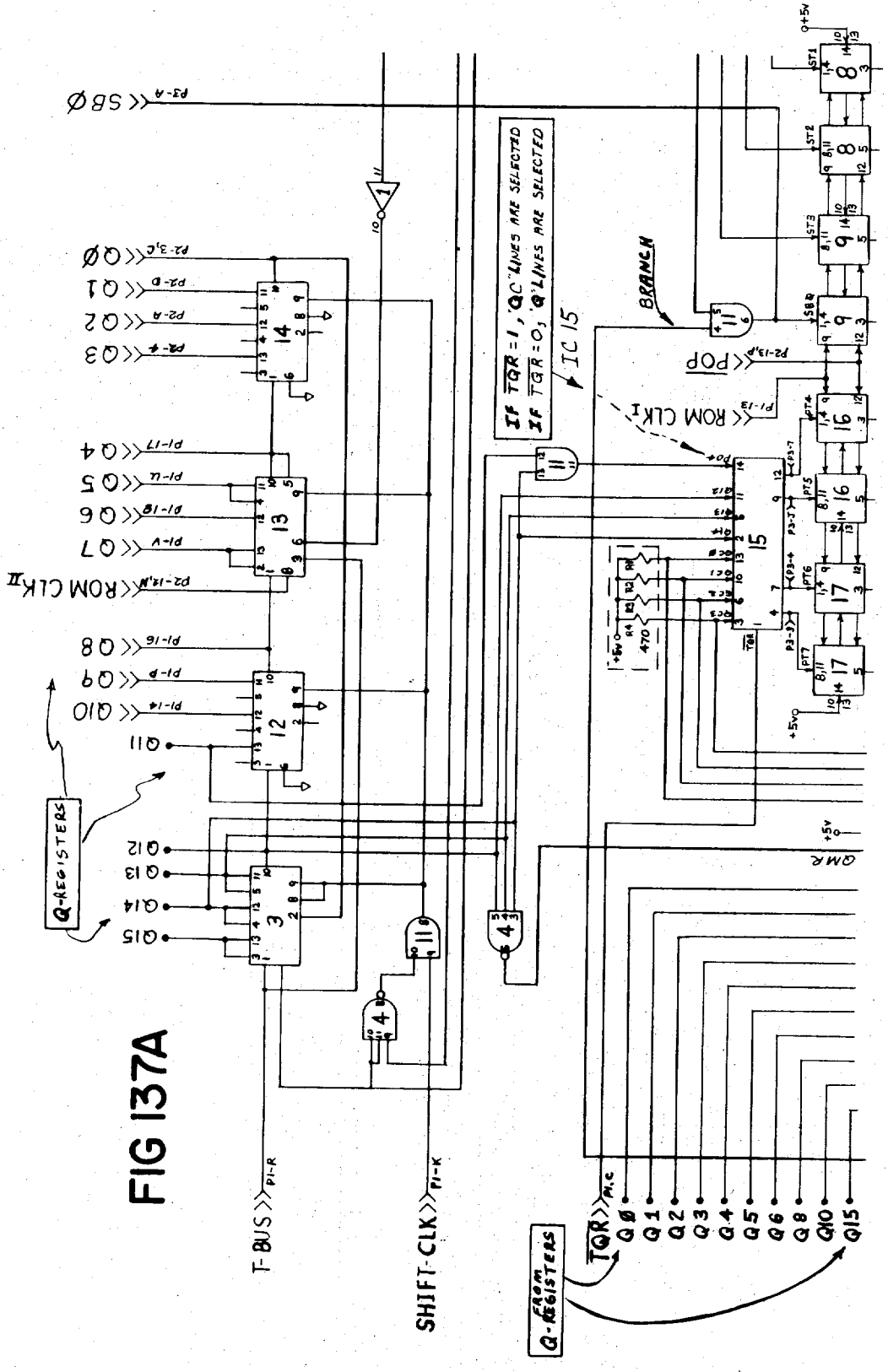

SBØ
QØ
Q1
Q2
Q3
Q4
Q5
Q6
Q7
ROM CLK II
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q-REGISTERS
IF TQR = 1, 'QC' LINES ARE SELECTED
IF TQR = 0, 'Q' LINES ARE SELECTED
IC 15
BRANCH
POP
ROM CLK I
QMR
T-BUS
SHIFT-CLK
TQR
FROM Q-REGISTERS

FIG 137A

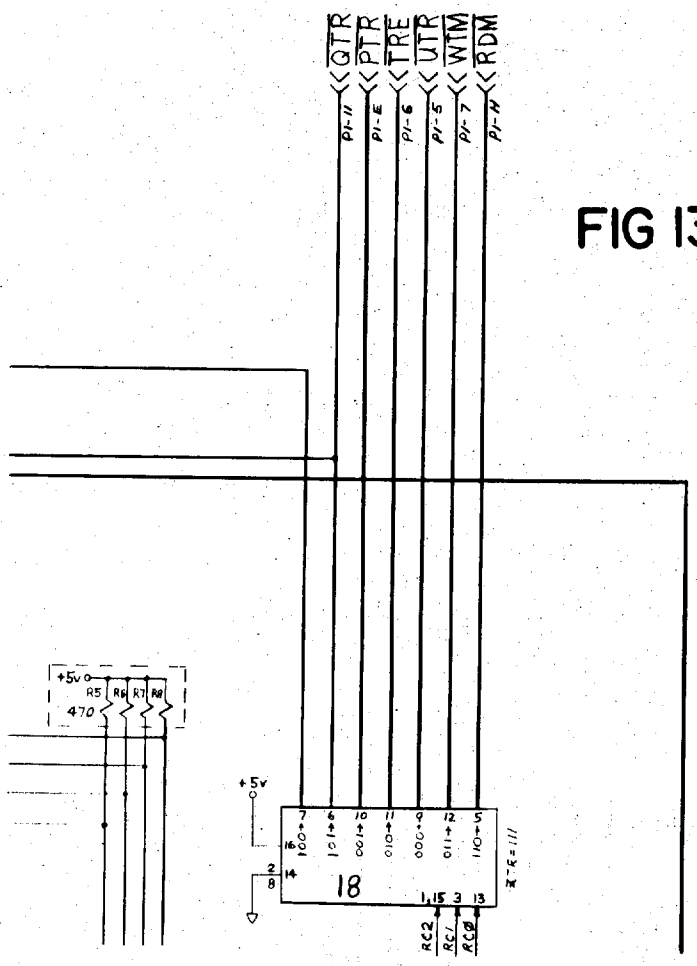
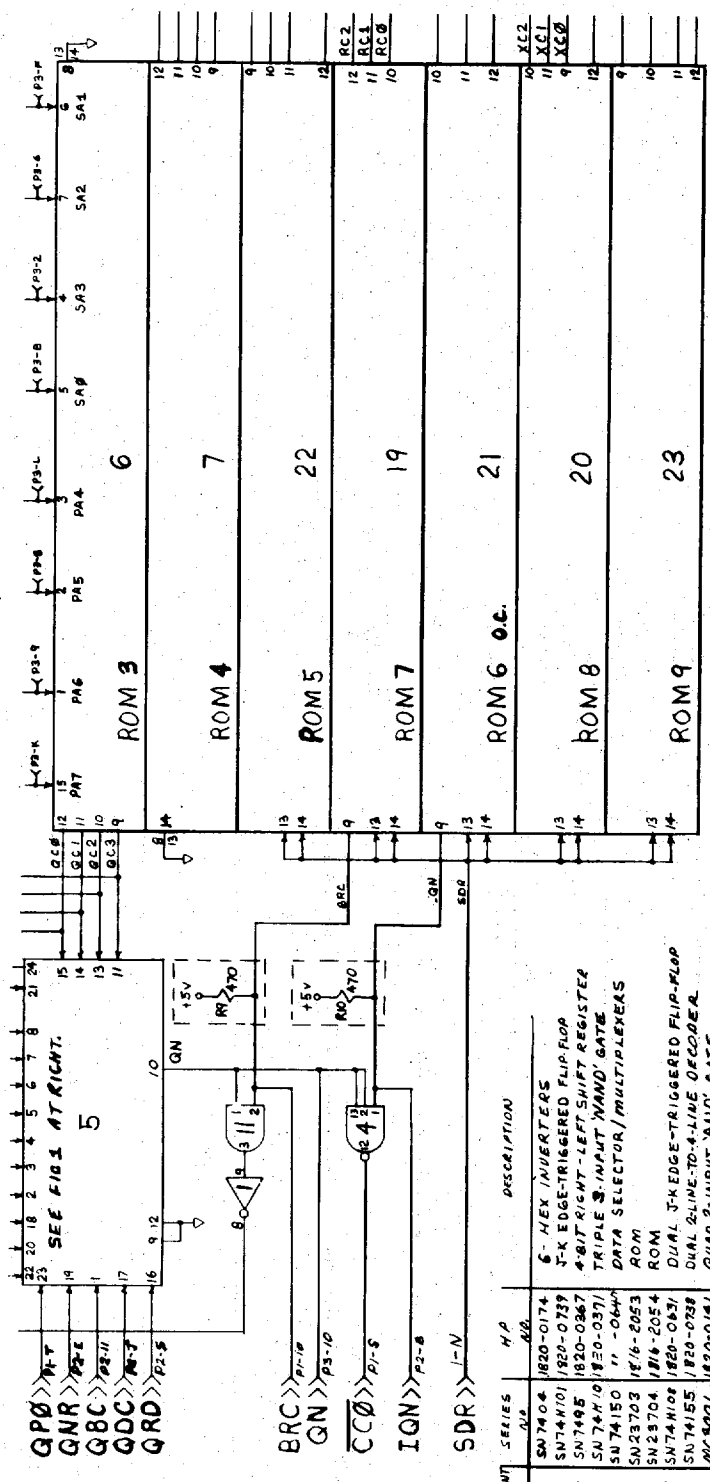

QTR
PTR
TRE
UTR
WTM
RDM
Pi-11
Pi-E
Pi-6
Pi-5
Pi-7
Pi-4
+5v
R5
R6
R7
R8
470
18
RC2
RC1
RC0

FIG 137B



COMPONENT IC NO.	SERIES NO.	HP NO.	DESCRIPTION
1	SN7404	1820-0174	6- HEX INVERTERS
2	SN74H101	1820-0739	J-K EDGE-TRIGGERED FLIP-FLOP
3,12,13,14	SN7495	1820-0367	4-BIT RIGHT-LEFT SHIFT REGISTER
4	SN74H10	1820-0371	TRIPLE 3-INPUT 'NAND' GATE
5	SN74150	" -0640	DATA SELECTOR/MULTIPLEXERS
6	SN23703	1816-2053	ROM
7	SN23704	1816-2054	ROM
8,9,16,17	SN74H108	1820-0631	DUAL J-K EDGE-TRIGGERED FLIP-FLOP
10,18	SN74155	1820-0738	DUAL 2-LINE-TO-4-LINE DECODER
11	MC3001	1820-0141	QUAD 2-INPUT 'AND' GATE
15	F932259	1820-0616	QUAD DATA SELECTOR
19	SN23707	1816-2057	ROM
20	SN23708	1816-2058	ROM
21	SN23706	" -2056	ROM
22	SN23705	" -2055	ROM
23	SN23709	" -2059	ROM

FIG 137C

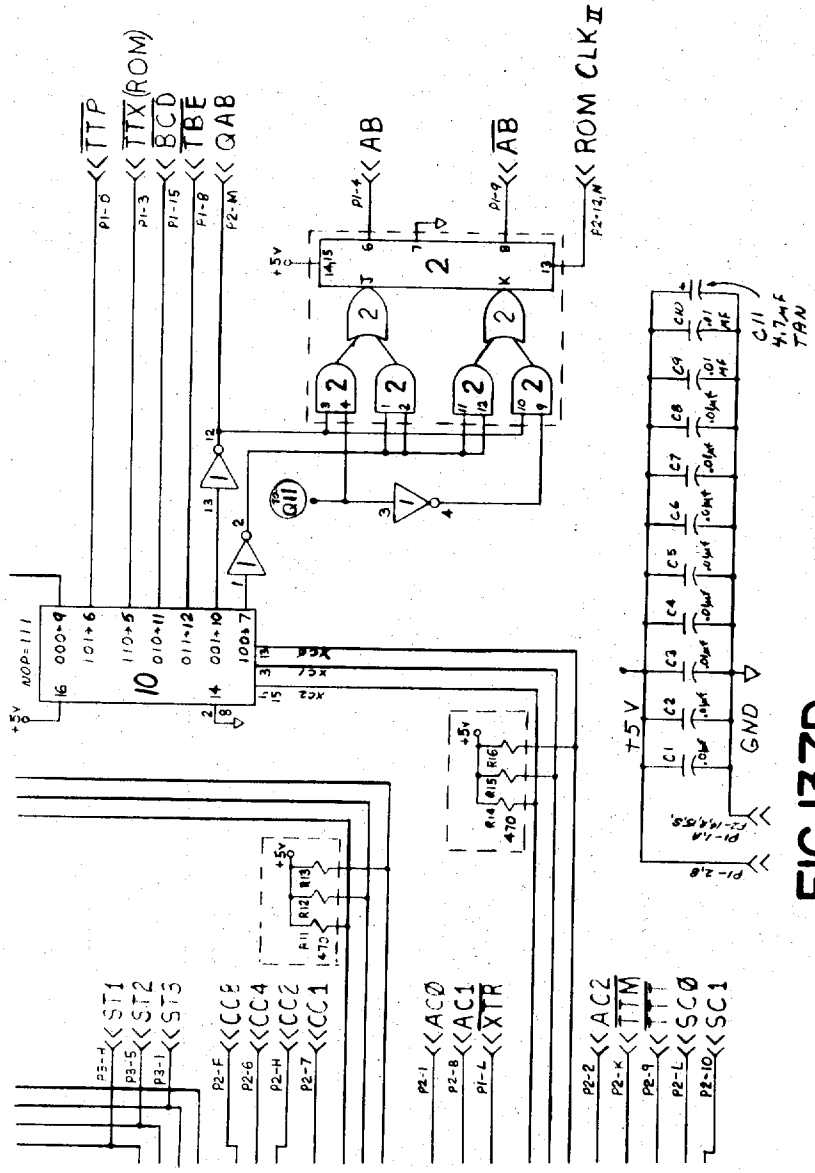

ROM CLK II
AB
AB
TTP
TTX (ROM)
BCD
TBE
QAB
+5V
GND
ST1
ST2
ST3
CC8
CC4
CC2
CC1
ACØ
AC1
XTR
AC2
TTM
SCØ
SC1

FIG 137D

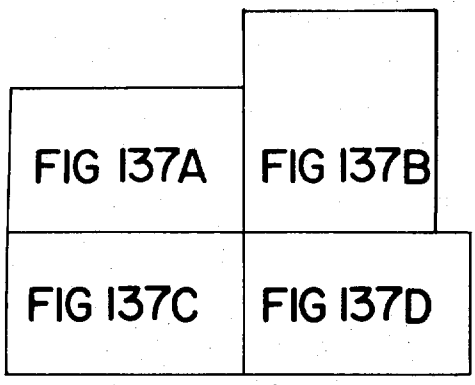


FIG 137'

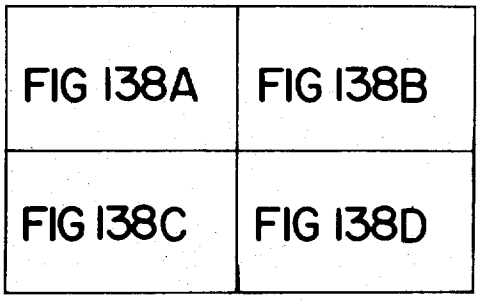


FIG. 138'

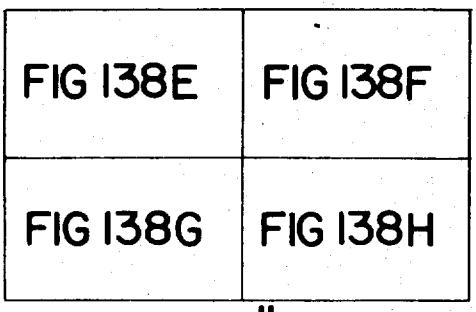


FIG 138"

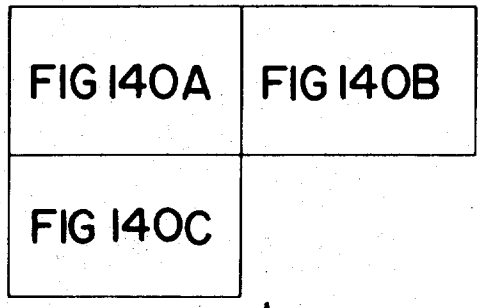


FIG 140'

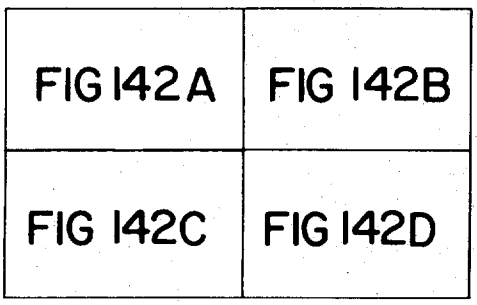


FIG 142'

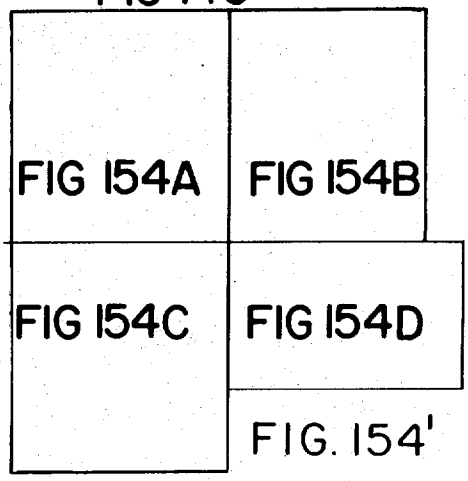


FIG. 154'

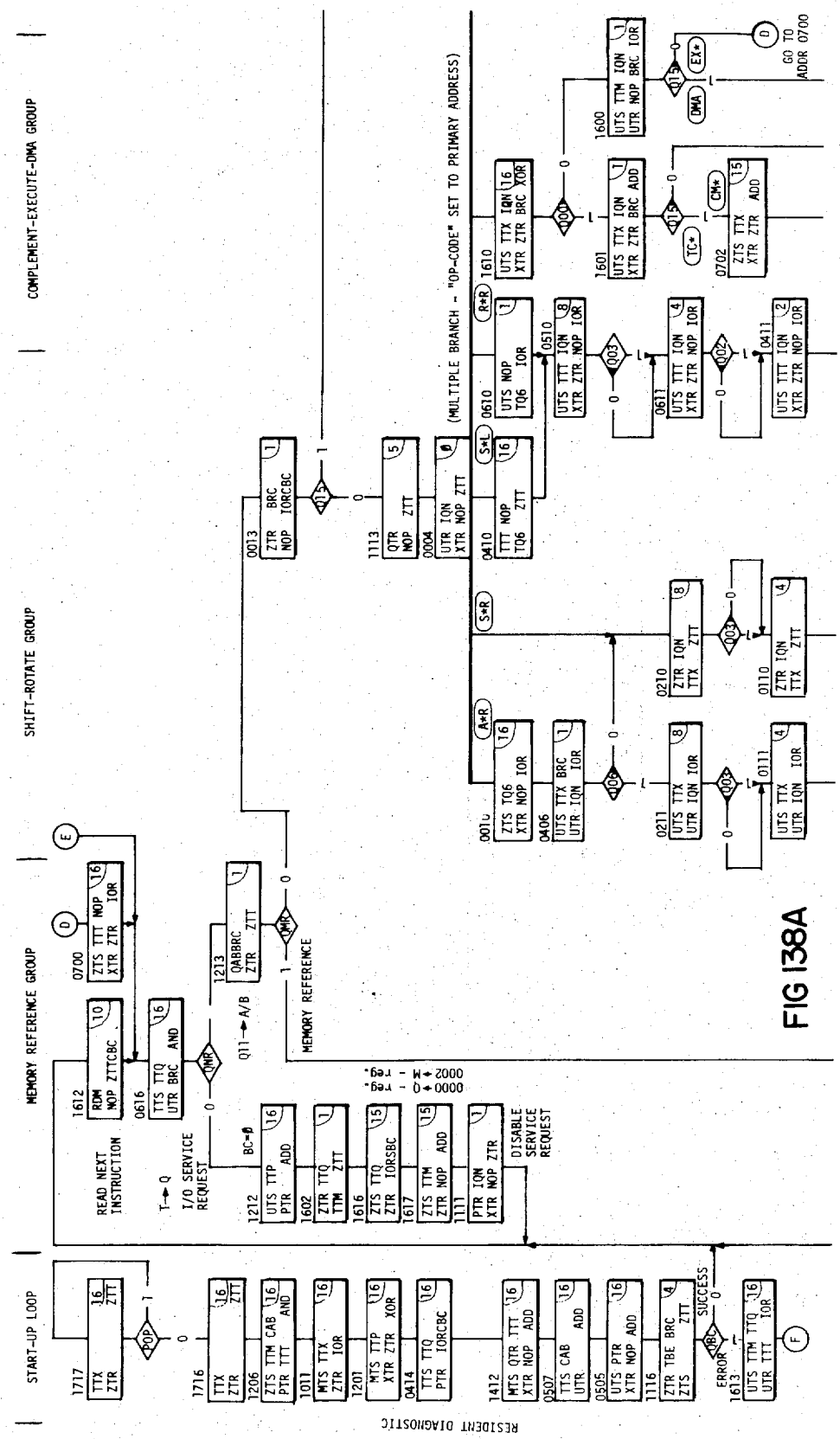

START-UP LOOP
SHIFT-ROTATE GROUP
COMPLEMENT-EXECUTE-DMA GROUP
MEMORY REFERENCE GROUP
RESIDENT DIAGNOSTIC
(MULTIPLE BRANCH - "OP-CODE" SET TO PRIMARY ADDRESS)
MEMORY REFERENCE
READ NEXT INSTRUCTION
I/O SERVICE REQUEST
DISABLE SERVICE REQUEST
GO TO ADDR 0700
ERROR
SUCCESS
0000 → 0 - reg.
0002 → M - reg.

FIG 138A

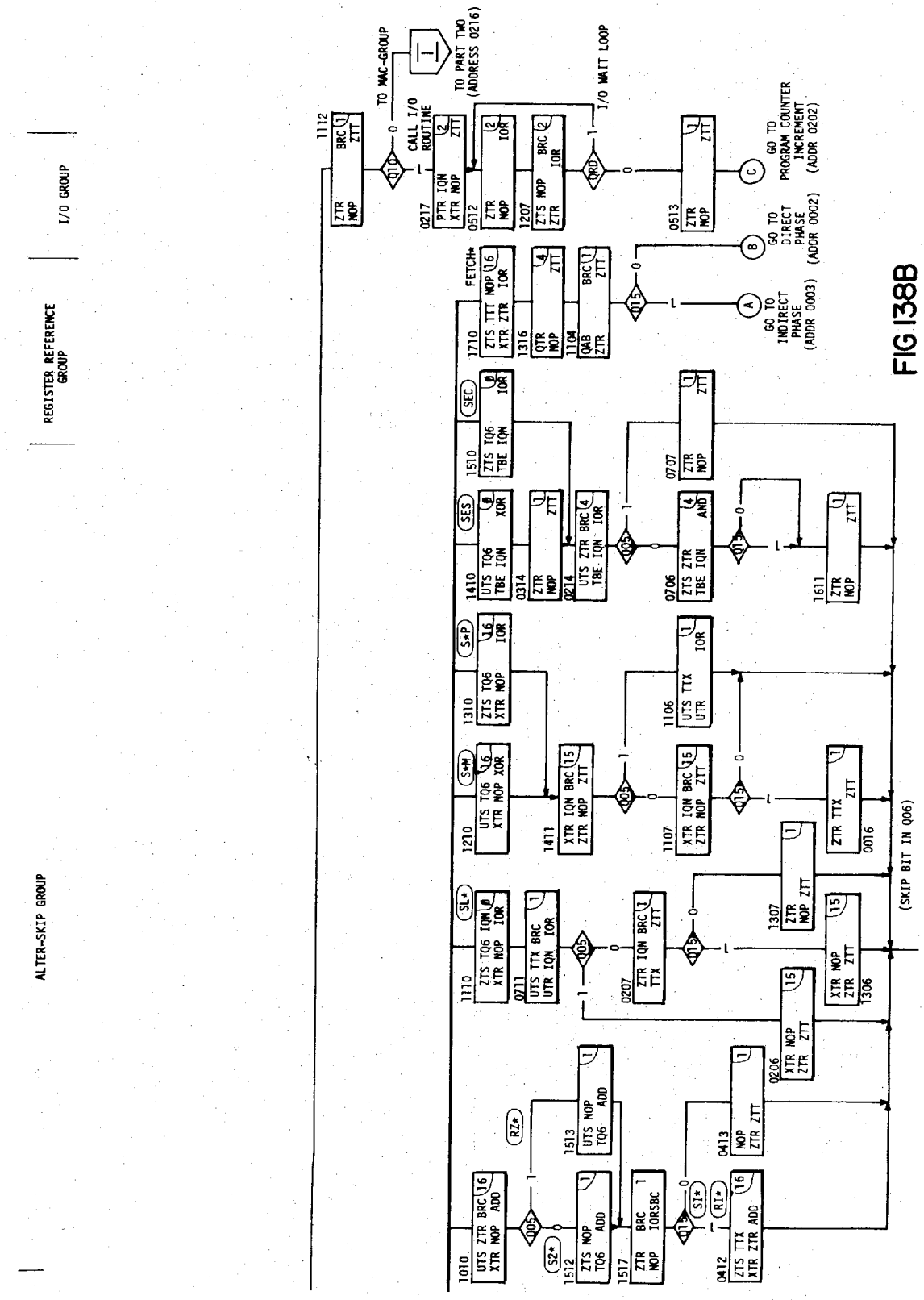

REGISTER REFERENCE GROUP
I/O GROUP
ALTER-SKIP GROUP
TO MAC-GROUP
CALL I/O ROUTINE
TO PART TWO (ADDRESS 0216)
I/O WAIT LOOP
GO TO PROGRAM COUNTER INCREMENT (ADDR 0202)
GO TO DIRECT PHASE (ADDR 0002)
GO TO INDIRECT PHASE (ADDR 0003)
FETCH+
(SKIP BIT IN Q06)

FIG 138

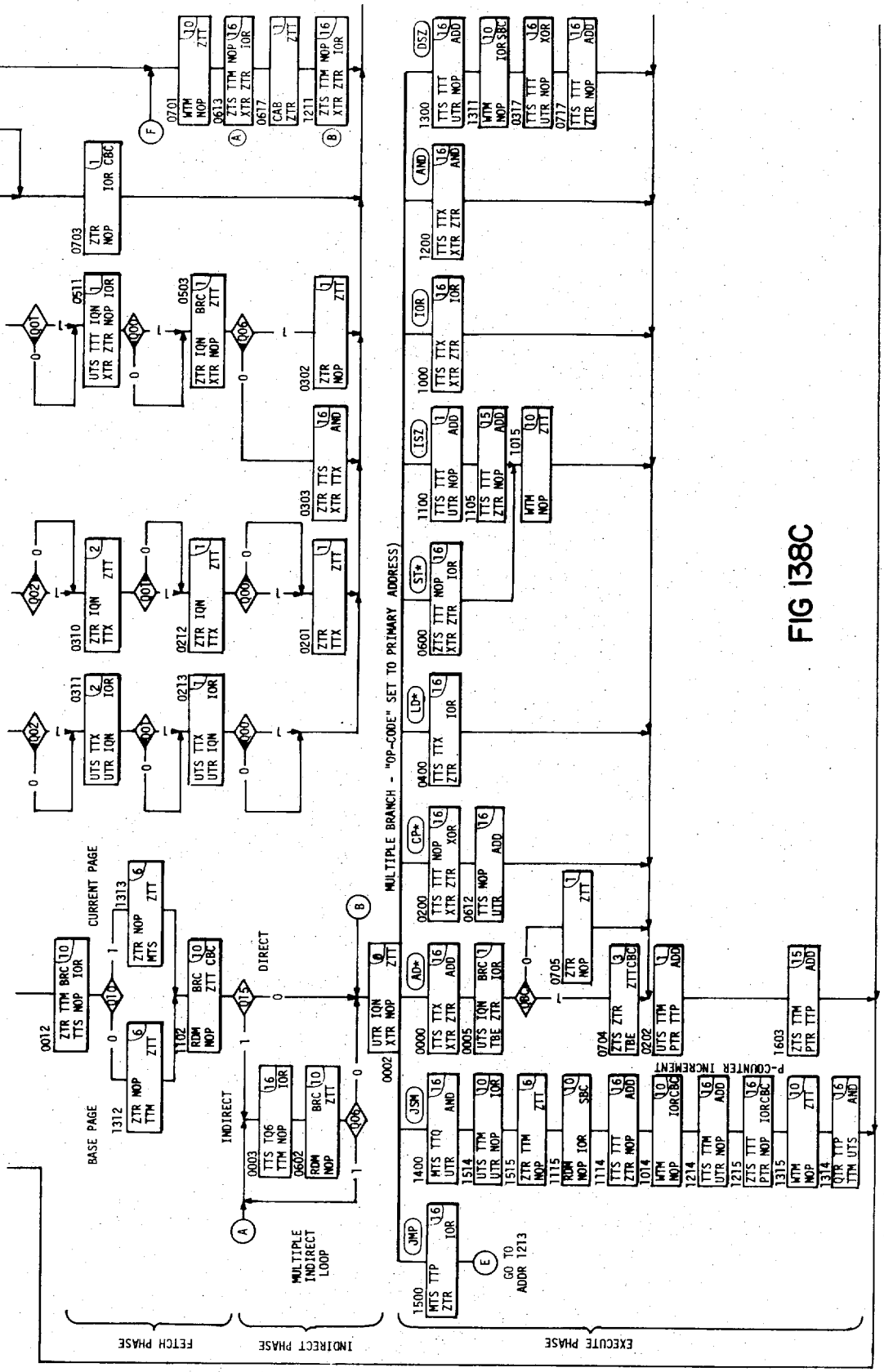

FETCH PHASE
INDIRECT PHASE
EXECUTE PHASE
MULTIPLE BRANCH - "OP-CODE" SET TO PRIMARY ADDRESS)
P-COUNTER INCREMENT
CURRENT PAGE
BASE PAGE
INDIRECT
DIRECT
MULTIPLE INDIRECT LOOP
GO TO ADDR 1213

FIG 138C

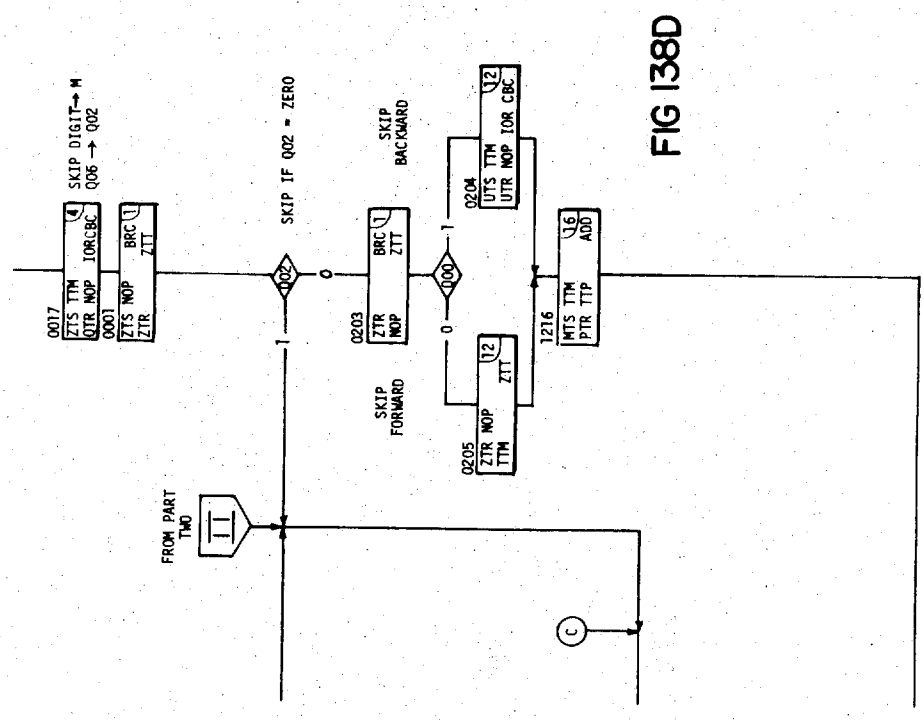

SKIP DIGIT→M
Q06 → Q02
SKIP IF Q02 = ZERO
SKIP BACKWARD
SKIP FORWARD
FROM PART TWO
0017
ZTS TTM
QTR NOP
IOR CBC
0001
ZTS NOP
ZTR
BRC 1
ZTT
Q02
0
1
0203
ZTR
NOP
BRC 1
ZTT
Q00
0204
UTS TTM
UTR NOP
IOR CBC
12
0205
ZTR NOP
TTM
ZTT
12
1216
MTS TTM
PTR TTP
ADD
16
C

FIG 138D

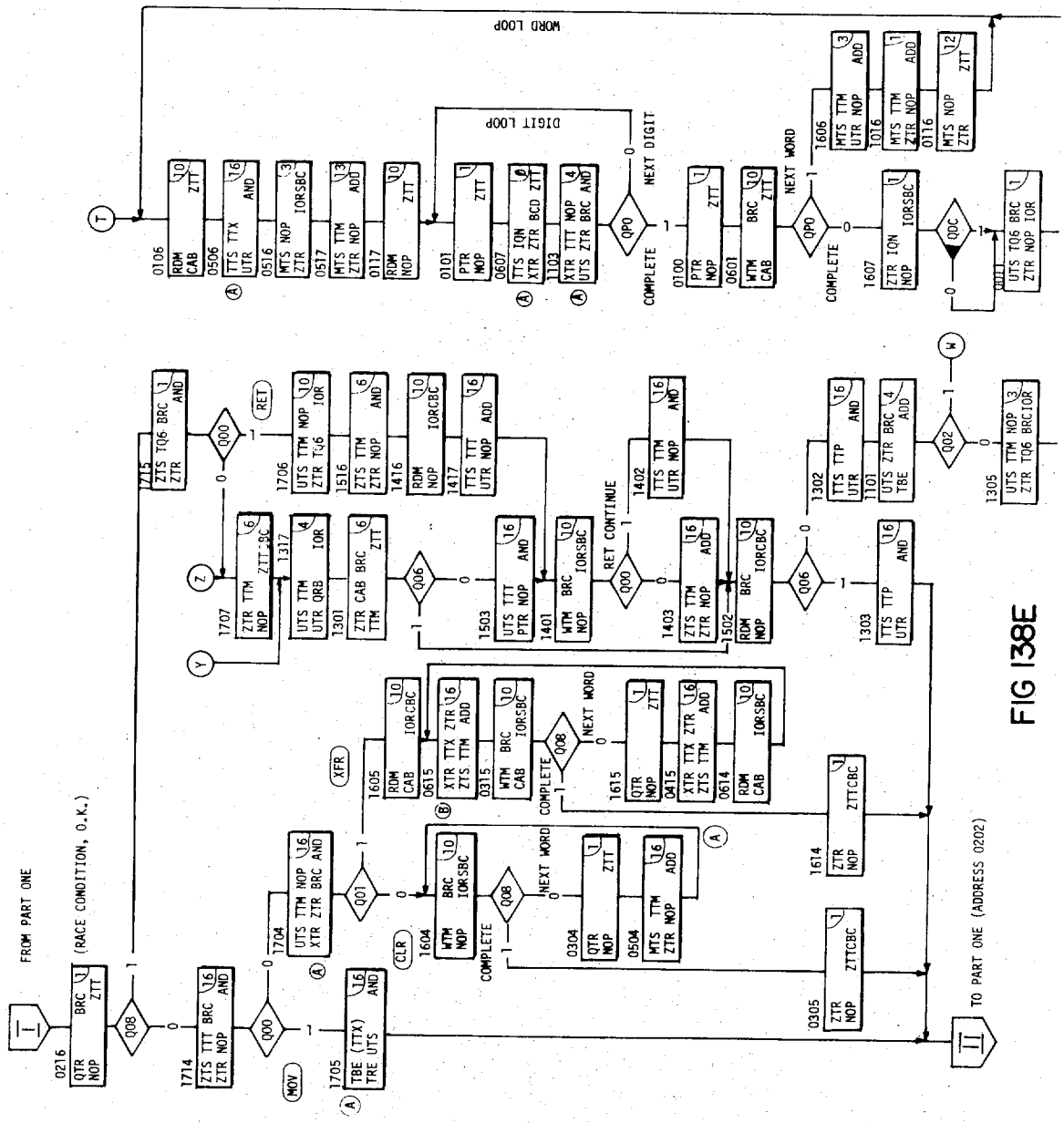

WORD LOOP
DIGIT LOOP
NEXT DIGIT
COMPLETE
NEXT WORD
RET CONTINUE
FROM PART ONE
(RACE CONDITION, O.K.)
TO PART ONE (ADDRESS 0202)

FIG 138E

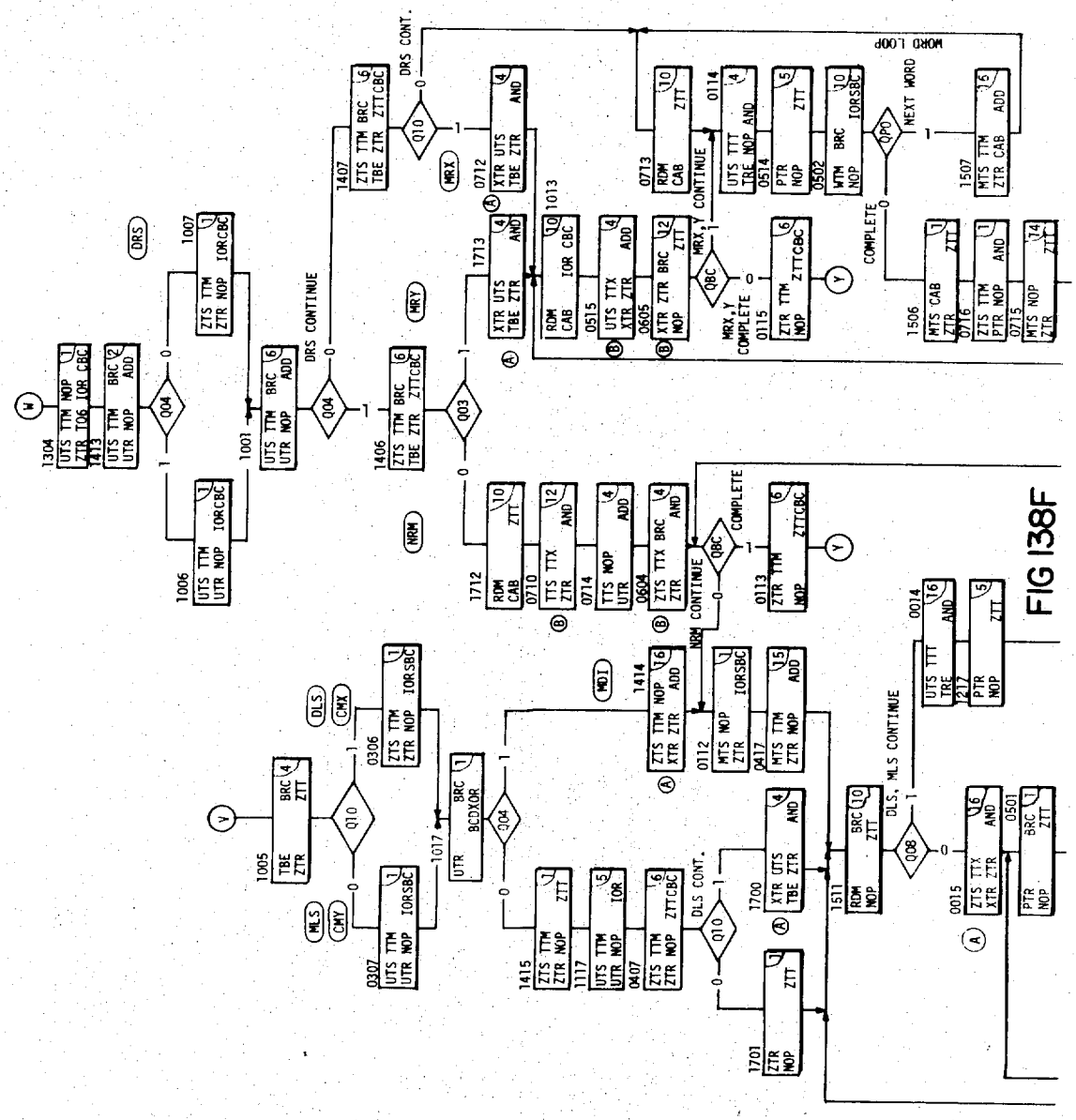

DRS
DRS CONTINUE
DRS CONT.
MRX
MRX, Y CONTINUE
MRX, Y COMPLETE
NRM
CONTINUE
COMPLETE
WORD LOOP
NEXT WORD
MDI
DLS
CMX
MLS
CMY
DLS CONT.
DLS, MLS CONTINUE

FIG 138F

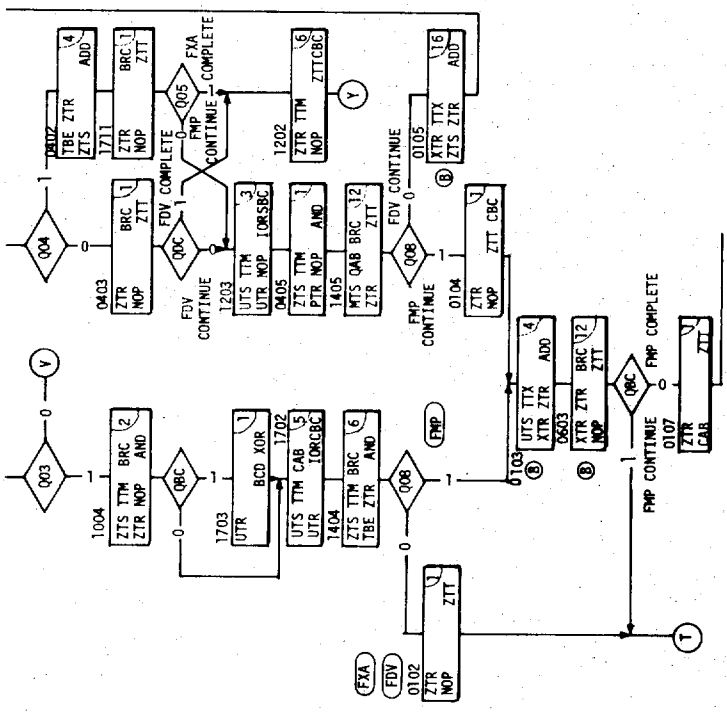
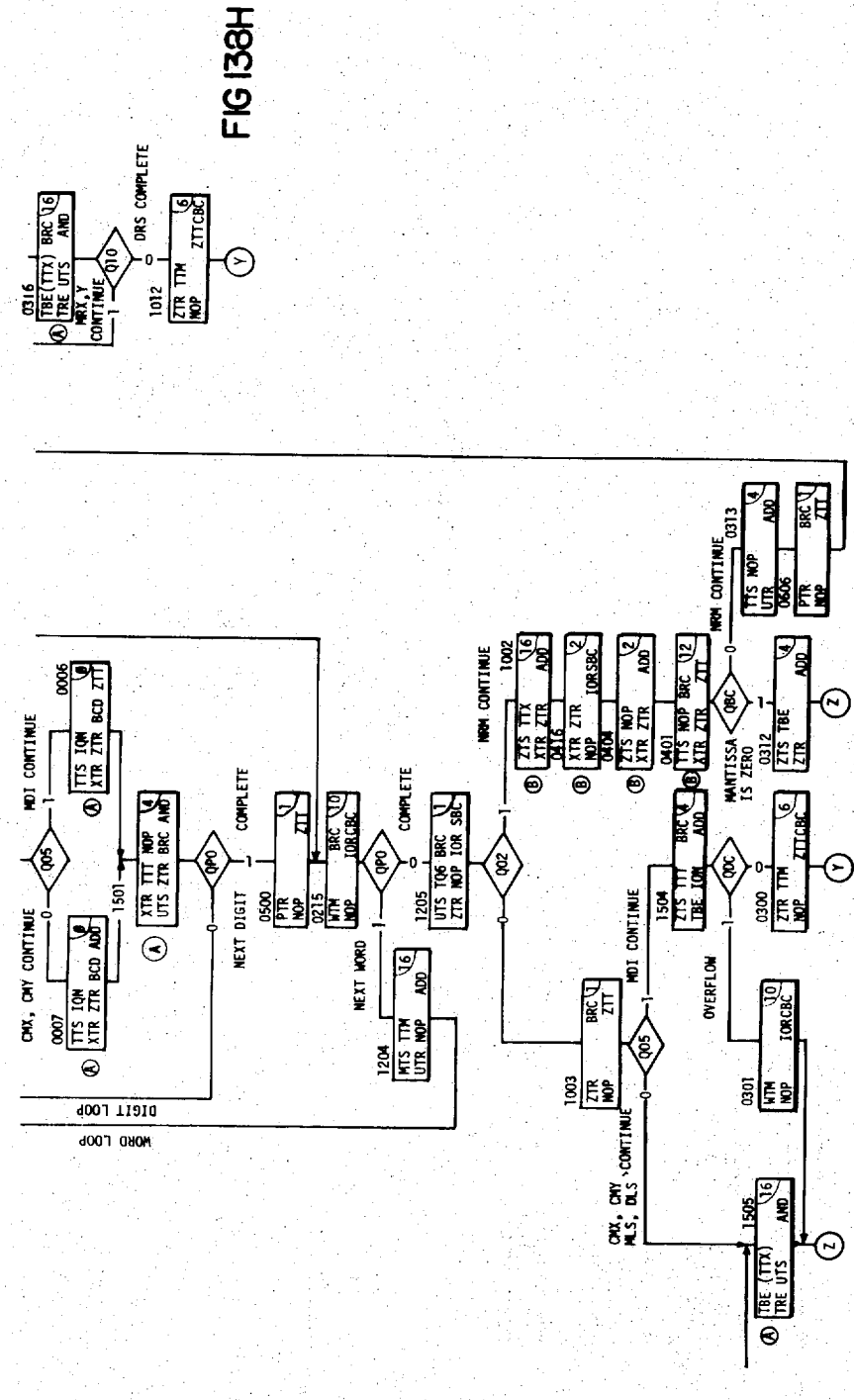

Q04
BRC 1 ZTT
ZTR NOP
0403
FDV COMPLETE
Q0C
FDV CONTINUE
1203
UTS TTM UTR NOP
3 IORSBC
0405
ZTS TTM PTR NOP
1 AND
1405
MTS QAB ZTR
BRC 12 ZTT
FDV CONTINUE
Q08
0104
ZTR NOP
1 ZTT CBC
FMP CONTINUE
0105
XTR TTX ZTS ZTR
16 ADD
Q05
FMP CONTINUE
FXA COMPLETE
1202
ZTR TTM NOP
6 ZTTCBC
0107
ZTS TBE ZTR
14 ADD
1711
BRC 1 ZTT
ZTR NOP
Q03
1004
ZTS TTM ZTR NOP
2 BRC AND
Q0C
1703
UTR
1 BCD XOR
1702
UTS TTM CAB UTR
5 IORCBC
1404
ZTS TTM BRC TBE ZTR
6 AND
Q08
FMP
0103
UTS TTX XTR ZTR
4 ADD
0603
XTR ZTR NOP
BRC 12 ZTT
Q0C
FMP COMPLETE
FMP CONTINUE
0107
ZTR CAR
7 ZTT
FXA
FDV
0102
ZTR NOP
7 ZTT

FIG 138G

FIG 138H


0316
TBE(TTX) TRE UTS
BRC 16 AND
MIX, Y CONTINUE
Q10
ORS COMPLETE
1012
ZTR TTM NOP
ZTTCBC
Y
MDI CONTINUE
Q05
0006
TTS ION XTR ZTR BCD ZTT
0007
TTS ION XTR ZTR BCD ADD
CMX, CMY CONTINUE
1501
XTR TTT NOP UTS ZTR BRC AND
A
Q00
COMPLETE
NEXT DIGIT
0500
PTR NOP
ZTT
0215
WTM NOP
BRC 10 IORCBC
Q00
COMPLETE
1205
UTS T06 ZTR NOP
BRC 1 IOR SBC
Q02
NEXT WORD
1204
MTS TTM UTR NOP
16 ADD
DIGIT LOOP
WORD LOOP
NORM CONTINUE
1002
ZTS TTX XTR ZTR
16 ADD
B
0416
XTR ZTR NOP WTM
IORSBC
ZTS NOP XTR ZTR
2 ADD
0401
TTS NOP XTR ZTR
BRC 12 ZTT
MANTISSA IS ZERO
QBC
0312
ZTS TBE ZTR
4 ADD
Z
0313
TTS NOP UTR
4 ADD
0606
PTR NOP
BRC 1 ZTT
1504
ZTS TTT TBE ION
BRC 4 ADD
QBC
0300
ZTR TTM NOP
6 ZTTCBC
Y
OVERFLOW
1003
ZTR NOP
BRC 1 ZTT
MDI CONTINUE
Q05
0301
WTM NOP
10 IORCBC
CMX, CMY MLS, DLS CONTINUE
1505
TBE (TTX) TRE UTS
16 AND
Z

1 MAC-INSTRUCTION CODING TABLE:

MNEMONIX	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OCTAL
RET	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	170402
MOV	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	170002
CLR	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	170000
XFR	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	170004
MRY	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	0	174470
MLS	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	171400
DRS	1	1	1	1	0	0	0	1	0	0	0	0	1	0	0	0	170410
DLS	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	175400
FXA	1	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0	170560
FMP	1	1	1	1	0	0	1	1	0	0	1	1	0	0	0	0	171460
FDV	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	170420
CMX	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	174400
CMY	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	170400
MDI	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	170540
NRM	1	1	1	1	0	0	1	1	0	0	1	0	1	0	0	0	171450
MRX	1	1	1	1	1	0	0	1	0	0	0	1	1	0	0	0	174430

2 USED DATA FORMAT:

ADDRESS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	se	exponent														sm
M+1	D_1				D_2				D_3				D_4			
M+2	D_5				D_6				D_7				D_8			
M+3	D_9				D_{10}				D_{11}				D_{12}			

D_1 ... MOST SIGNIFICANT DIGIT

D_{12} .. LEAST SIGNIFICANT DIGIT

se ... SIGN OF EXPONENT

sm ... SIGN OF MANTISSA

DECIMAL POINT IS LOCATED BETWEEN D_1 AND D_2

3 USED ADDRESS AND CONSTANTS (OCTAL)

0001 COUNTER CONSTANT ADDRESS

1700 P-REGISTER CONTENTS STORAGE

1777 STACK-POINTER ADDRESS

1744-47 AR1
1754-57 AR2 } ARITHMETIC PSEUDOREGISTERS

043060 COUNTER CONSTANT

FIG 138 I

INSTRUCTIONS

ADD - BINARY ADD = CARRY FLIP-FLOP + R_{BUS} + S_{BUS} TO T_{BUS} - CARRY OUT TO CARRY FLIP-FLOP.

AND - LOGICAL AND = R·S TO T_{BUS}

BCD - BCD ARITHMETIC MODE OF ALU

BRC - BRANCH INHIBIT SOO IF QNM

CAB - COMPLEMENT AB FLIP-FLOP

IOR - INCLUSIVE OR R+S TO T_{BUS}

IOR·CBC - IOR, CLEAR BINARY CARRY FLIP-FLOP

IOR·SBC - IOR, SET BINARY CARRY FLIP-FLOP

IOS = PTR·XTR - INITIATES TRANSFER OF CONTROL TO I/O

IQN - INHIBIT SHIFT CLOCK IF QNM

MTS - TRANSFER M_{REG} TO S_{BUS}

NOP - NO OPERATION ON XC INSTRUCTIONS

PTR - TRANSFER P_{REG} TO R_{BUS}

QAB - Q_{REG} (BIT 11) TO AB FLIP-FLOP

QTR - TRANSFER Q_{REG} TO R_{BUS}

RDM - READ MEMORY (M) TO T_{REG}

TBE - TRANSFER T_{BUS} TO E_{REG} TO R_{BUS}

TQR = UTR·XTR - TRANSFER Q_{REG} TO PRIMARY ADDRESS

Q14·Q11	→ P04
Q12	→ P05
Q13	→ P06
Q14	→ P07

TRE - TRANSFER T_{REG} TO E_{REG} TO R_{BUS}

TTM - TRANSFER T_{BUS} TO M_{REG}

TTP - TRANSFER T_{BUS} TO P_{REG}

TTQ - TRANSFER T_{BUS} TO Q_{REG}

TTS - TRANSFER T_{REG} TO S_{BUS}

TTT - TRANSFER T_{BUS} TO T_{REG}

TTX - TRANSFER T_{BUS} TO A/B_{REG}

TO6·ZTR - TRANSFER T_{BUS} TO Q_{REG} (BIT 6)

UTR - ONE TO R_{BUS}

UTS - ONE TO S_{BUS}

WTM - STORE T_{REG} IN MEMORY

XOR - EXCLUSIVE OR R+S TO T_{BUS}

XTR - A/B_{REG} TO R_{BUS}

ZTR - ZERO TO R_{BUS}

ZTS - ZERO TO S_{BUS}

ZTT - ZERO TO T_{BUS}

ZTT·CBC - ZERO TO T_{BUS}, CLEAR BINARY CARRY FLIP-FLOP

FIG 138 J

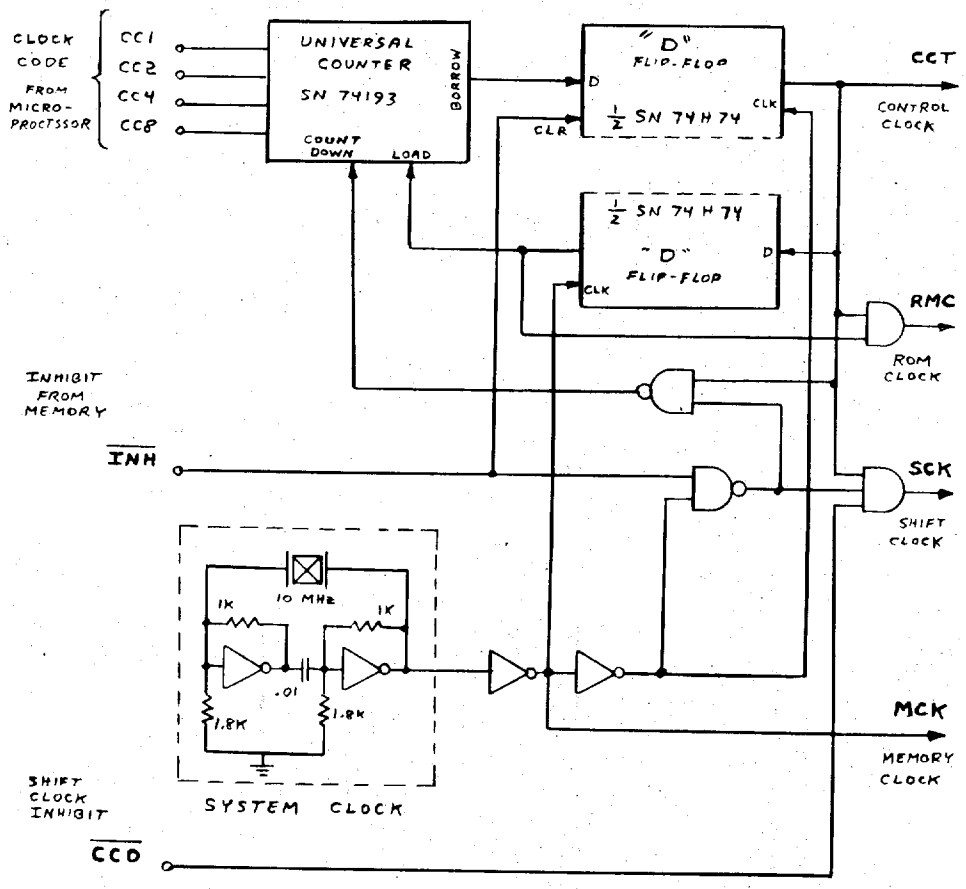


FIG 139 PROGRAMMABLE CLOCK

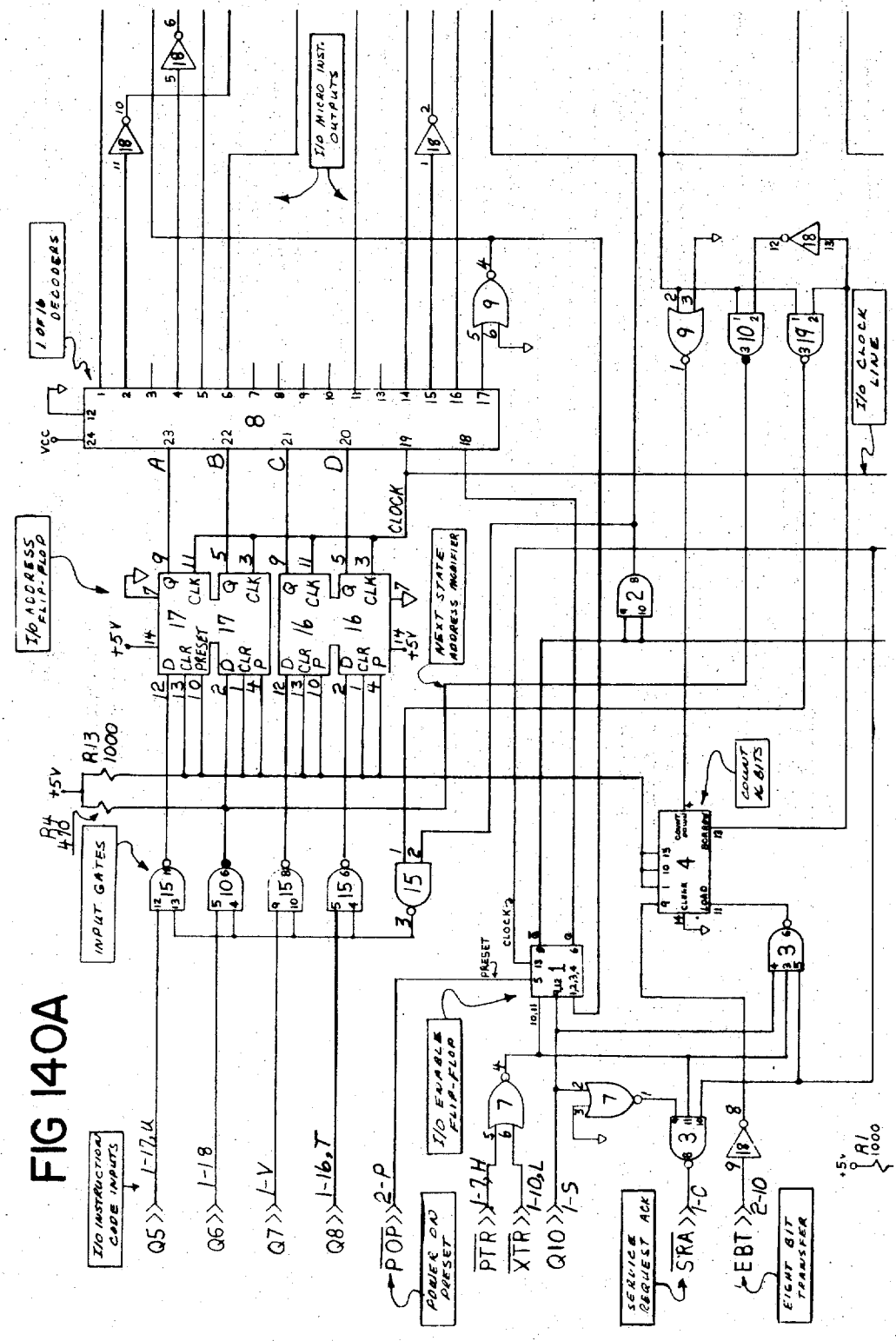

I/O INSTRUCTION CODE INPUTS
INPUT GATES
I/O ADDRESS FLIP-FLOP
I/O MICRO INST. OUTPUTS
I OF 16 DECODERS
NEXT STATE ADDRESS MODIFIER
I/O CLOCK LINE
COUNT 4 BITS
I/O ENABLE FLIP-FLOP
POWER ON PRESET
SERVICE REQUEST ACK
EIGHT BIT TRANSFER
CLOCK
PRESET
+5V
R13 1000
R14 470
R1 1000
Q5>>
Q6>>
Q7>>
Q8>>
POP>>
PTR>>
XTR>>
Q10>>
SRA>>
EBT>>

FIG 140A

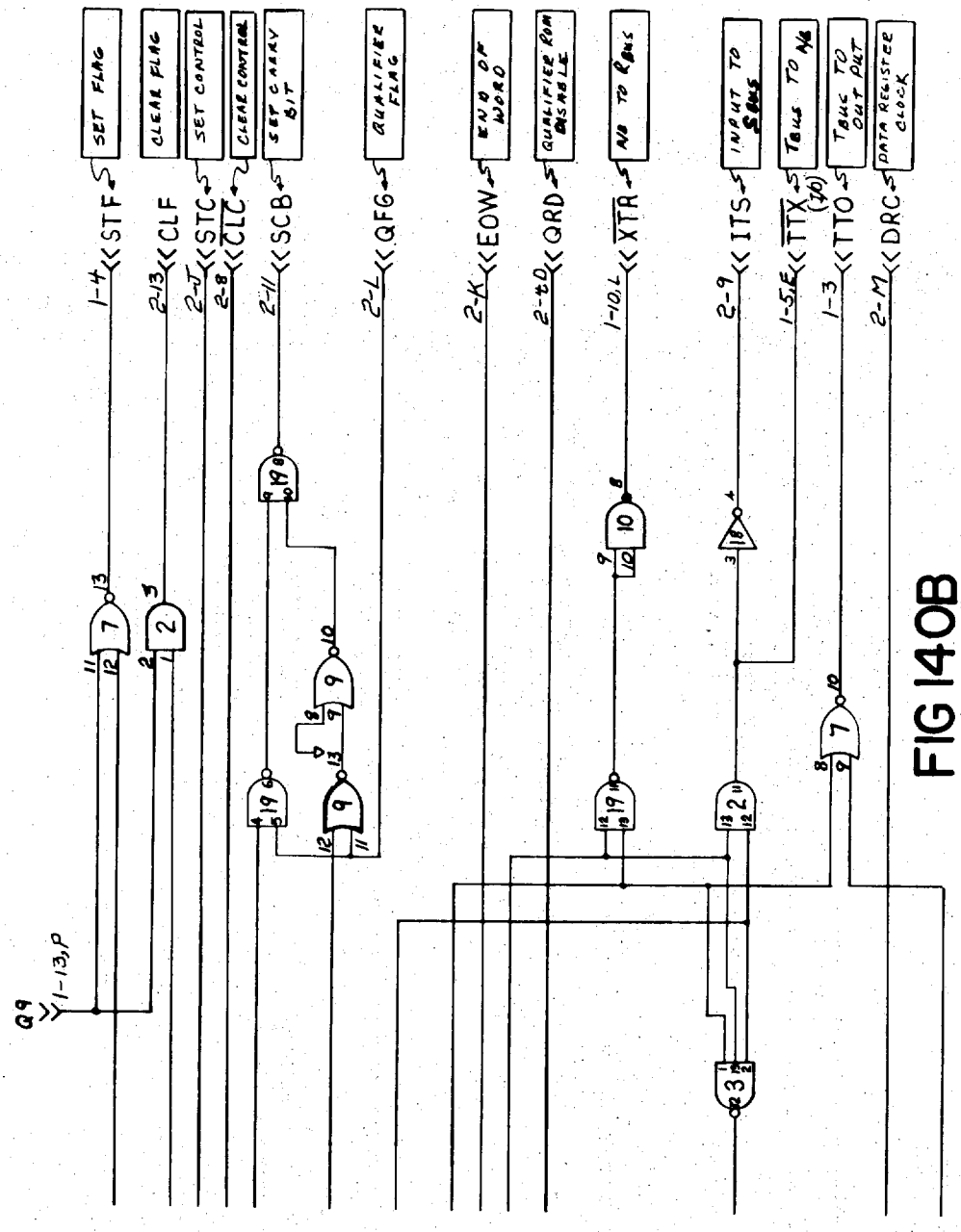

SET FLAG
CLEAR FLAG
SET CONTROL
CLEAR CONTROL
SET C MEM BIT
QUALIFIER FLAG
END OF WORD
QUALIFIER ROM ENABLE
NO TO RBUS
INPUT TO SBUS
TBUS TO NB
TBUS TO OUT PUT
DATA REGISTER CLOCK
STF
CLF
STC
CLC
SCB
QFG
EOW
QRD
XTR
ITS
TTX
TTO
DRC
Q9
1-13,P

FIG 140B

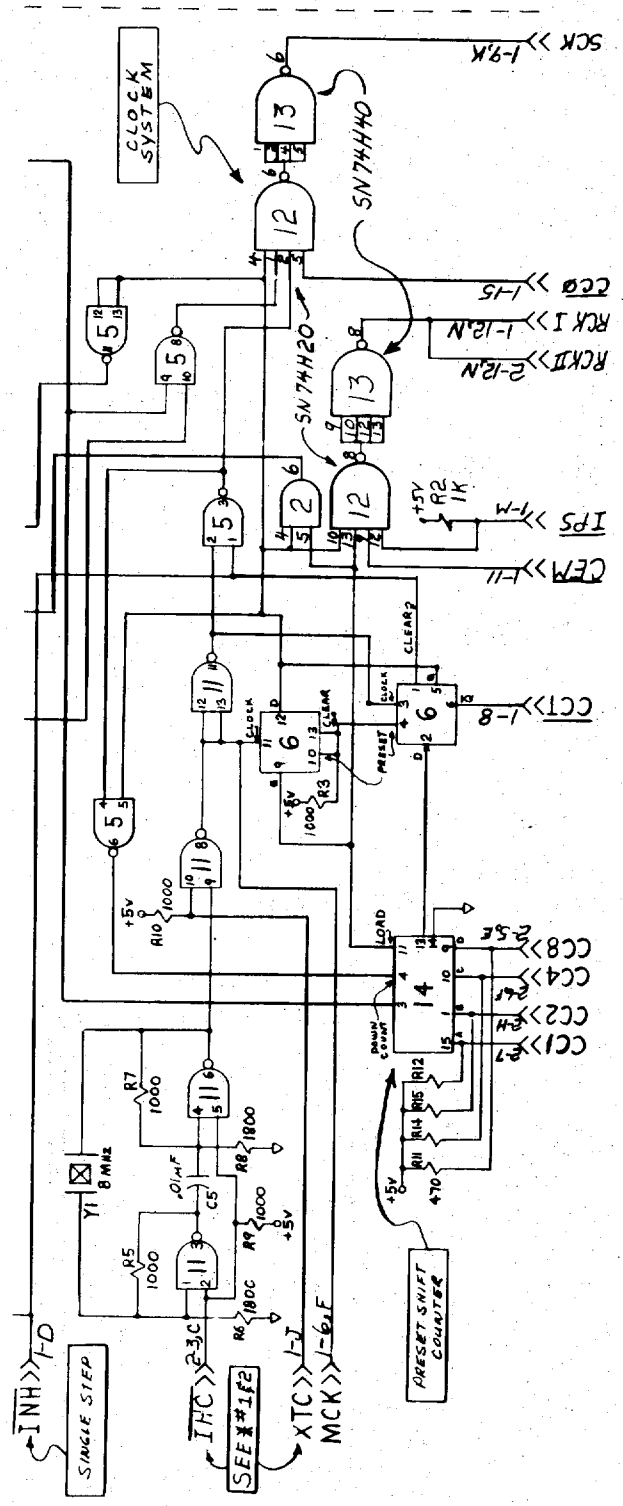
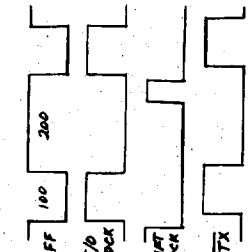


FIG 140C



COMPONENT NO.	SERIES NO.	DESCRIPTION
3	SN7410	TRIPLE 3-INPUT NAND GATES
2	MC3001P	QUAD 2-INPUT AND GATES
4,14	SN74193	SYN. 4-BIT UP/DOWN COUNTER
5,11	SN7400L	QUAD 2-INPUT NAND GATE
6	SN74H74	DUAL EDGE-TRIGGERED F-F
7,9	MC3002P	QUAD 2-INPUT NOR GATE
8	SN74154	4-LINE TO 16 LINE DECODER
10	SN7403	QUAD 2-INPUT NAND GATE OC
12,13	MC3026	DUAL 4 INPUT AND POWER GATE
15,18	SN7400N	QUAD 2-INPUT NAND GATE
16	MC4015	QUAD D FLIP-FLOPS
17	SN7404	6 HEX INVERTERS
1	SN74101	J-K EDGE-TRIGGERED FLIP FLOP (HIGH SPEED)

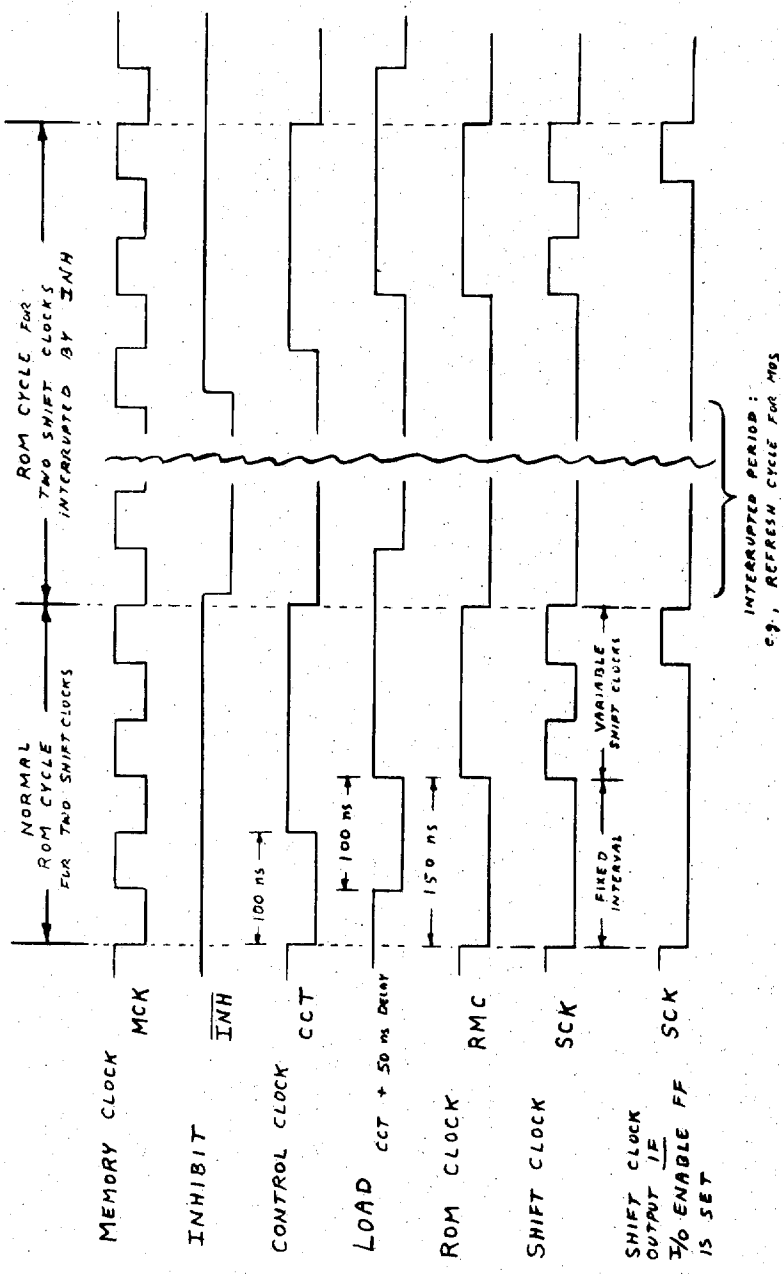


FIG 141 TIMING WAVEFORMS — CENTRAL PROCESSOR

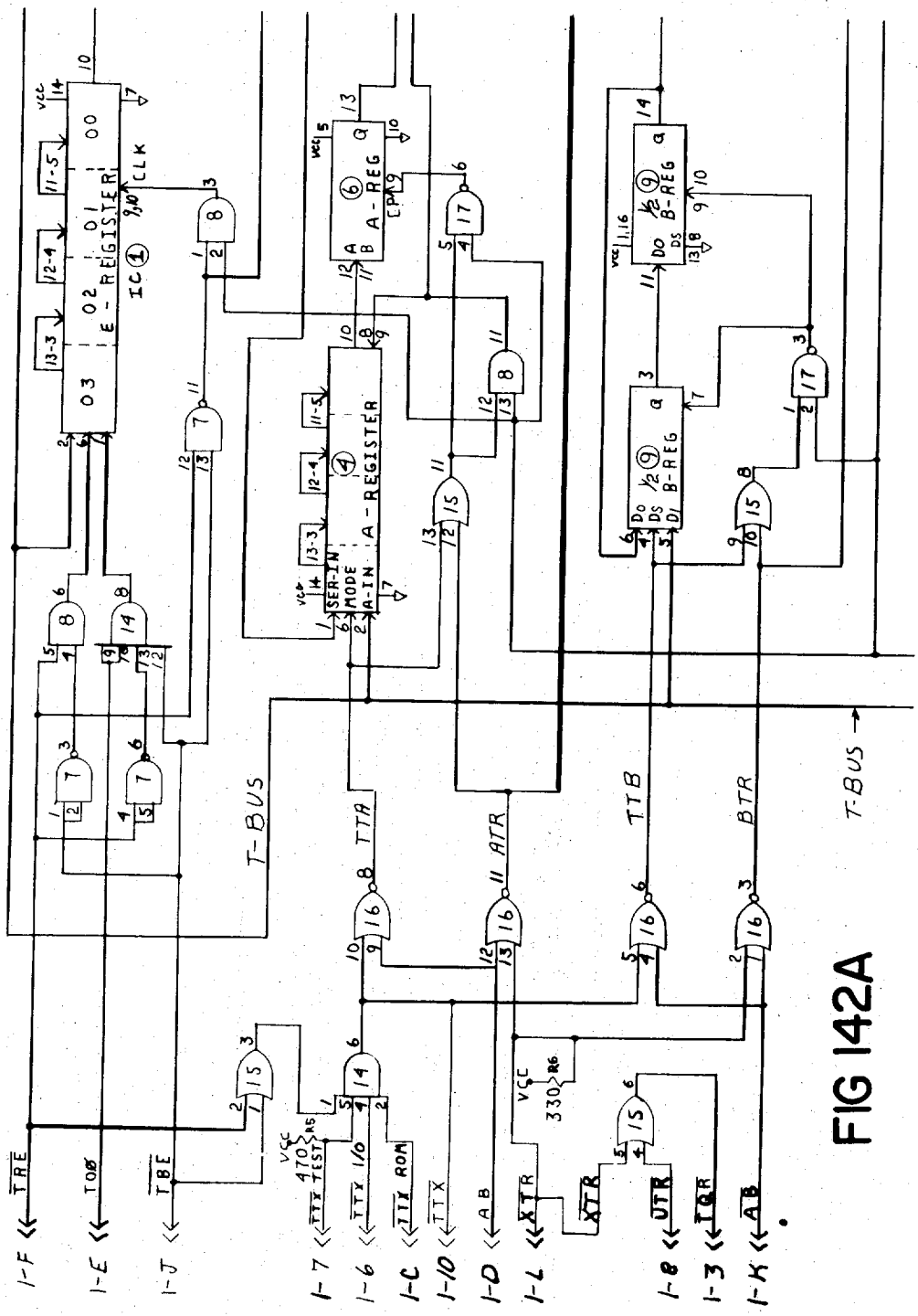
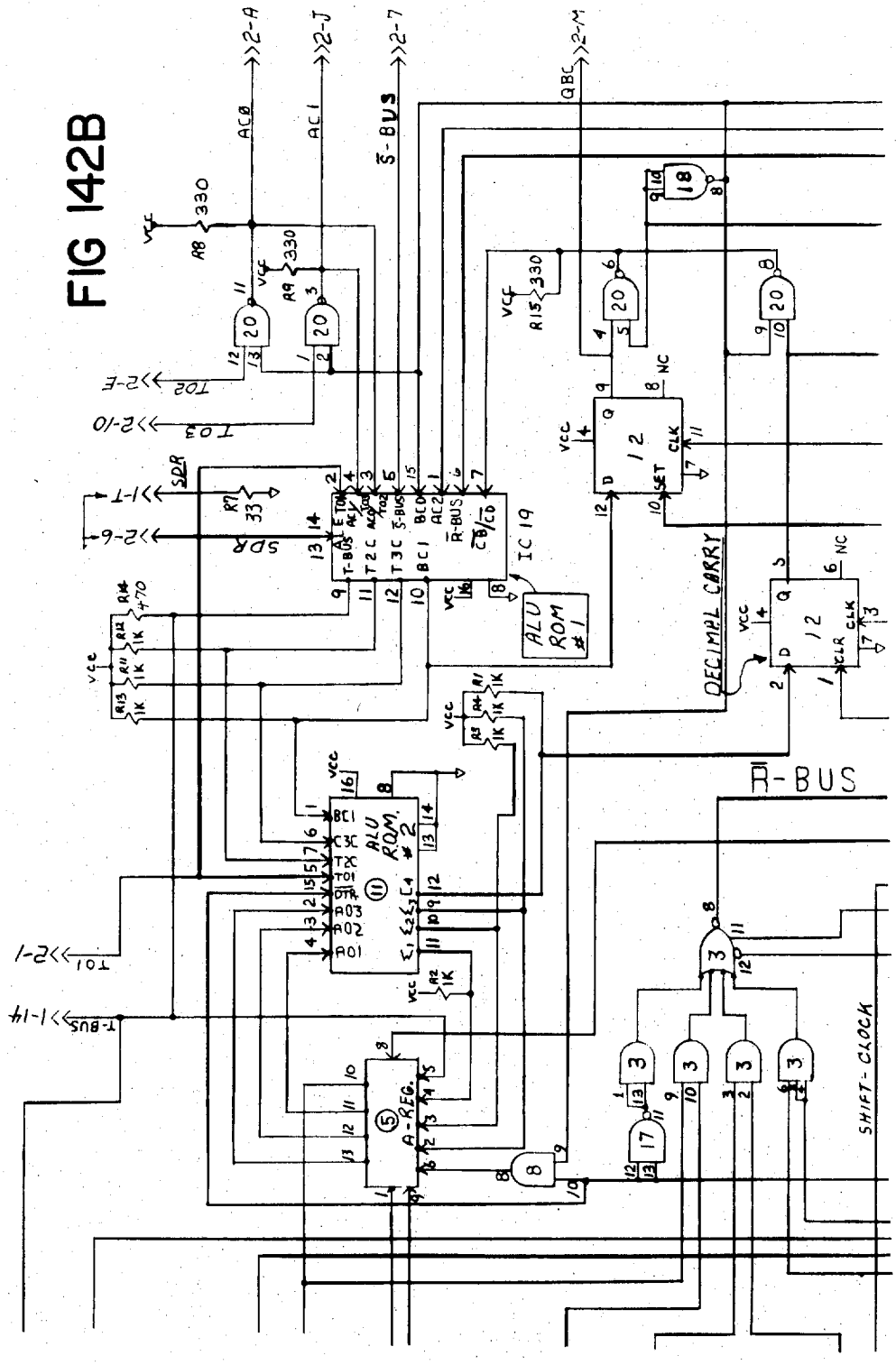

E - REGISTER
IC 1
A - REG
A - REGISTER
B-REG
B-REG
SER-IN
MODE
A-IN
CLK
T-BUS
TTA
ATR
TTB
BTR
T-BUS
TRE
TOØ
TBE
TTX TEST
TTX I/O
TTX ROM
TTX
AB
XTR
XTR
UTR
TQR
AB
1-F
1-E
1-J
1-7
1-6
1-C
1-10
1-D
1-L
1-8
1-3
1-K

FIG 142A

FIG 142B


ALU ROM #1
ALU ROM #2
A-REG.
IC 19
DECIMAL CARRY
R̄-BUS
SHIFT-CLOCK
S̄-BUS
SDR
T-BUS
T01
T02
T03
QBC
ACØ
AC1
>2-A
>2-J
>2-7
>2-M
>2-E
>2-10
>1-T
>2-6
>2-1
>1-14

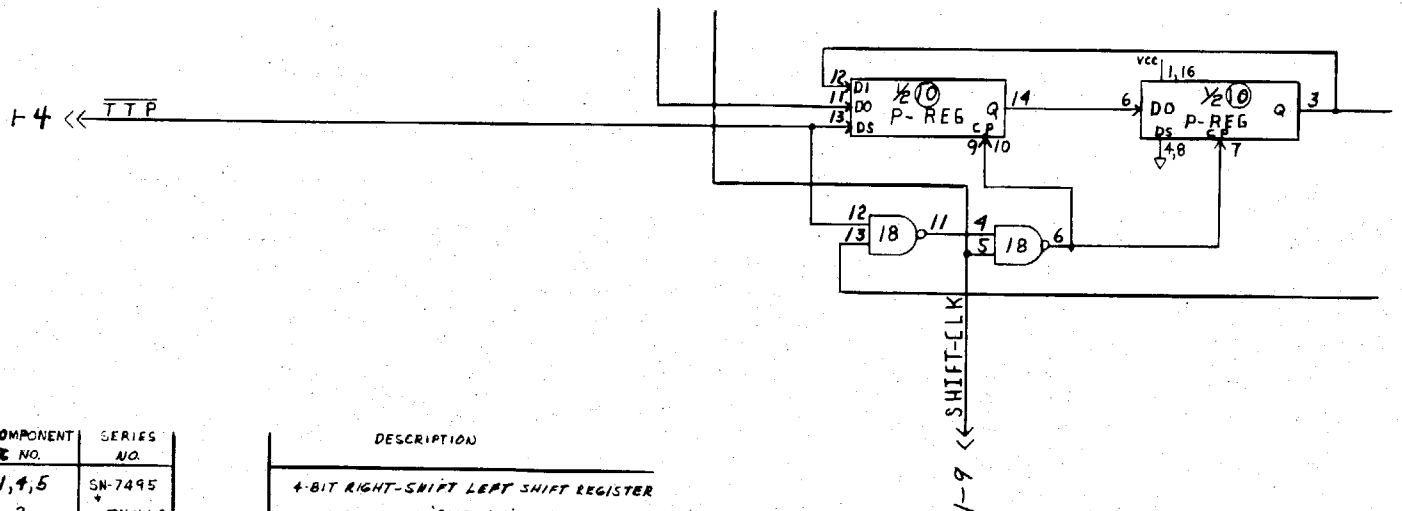


FIG 142C

COMPONENT IC NO.	SERIES NO.	DESCRIPTION
1,4,5	SN-7495	4-BIT RIGHT-SHIFT LEFT SHIFT REGISTER
2	-74H62	3-2-2-3-INPUT 'AND-OR' EXPANDER
3	-74H53	EXPANDABLE 2-2-2-3-INPUT 'AND-OR' INVERTED GATE
6	-7491	8 BIT SHIFT REGISTER
7, 13, 17, 18	-74H00	QUAD 2-INPUT 'NAND' GATES
8	MC-3001P	QUAD 2-INPUT 'AND' GATES
9,10	F932859	8-BIT SHIFT REGISTER, DUAL
11		BI-POLAR 'ROM'
12	SN-7474	DUAL, EDGE-TRIGGERED FLIP-FLOPS
14	SN-74H21	DUAL, 4-INPUT 'AND' GATES
15	MC3003P	QUAD, 2-INPUT 'OR' GATES
16	MC3002P	QUAD, 2-INPUT 'NOR' GATES
19		BI-POLAR 'ROM'
20	SN74H01	QUAD 2-INPUT 'NAND' GATE (OPEN COLLECTOR OUTPUT)

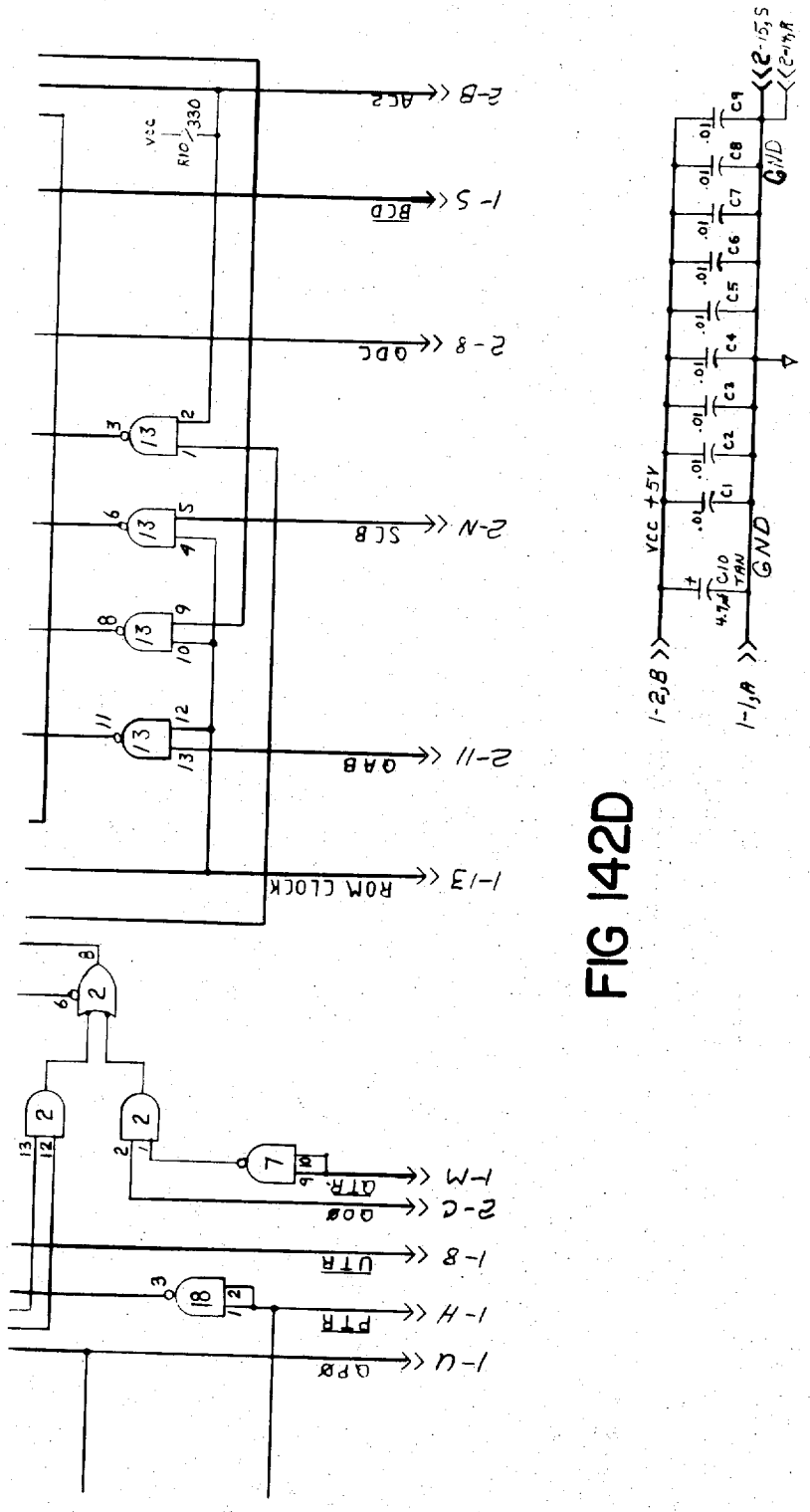

2-B
ACZ
1-S
BCD
2-8
QDC
2-N
SCB
2-11
QAB
1-13
ROM CLOCK
1-M
QTR
2-C
QDØ
1-8
UTR
1-H
PTR
1-U
QPØ
VCC
R10
330
VCC +5V
GND
1-2,B
1-1,A
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
TAN
.01
4.7µf

FIG 142D

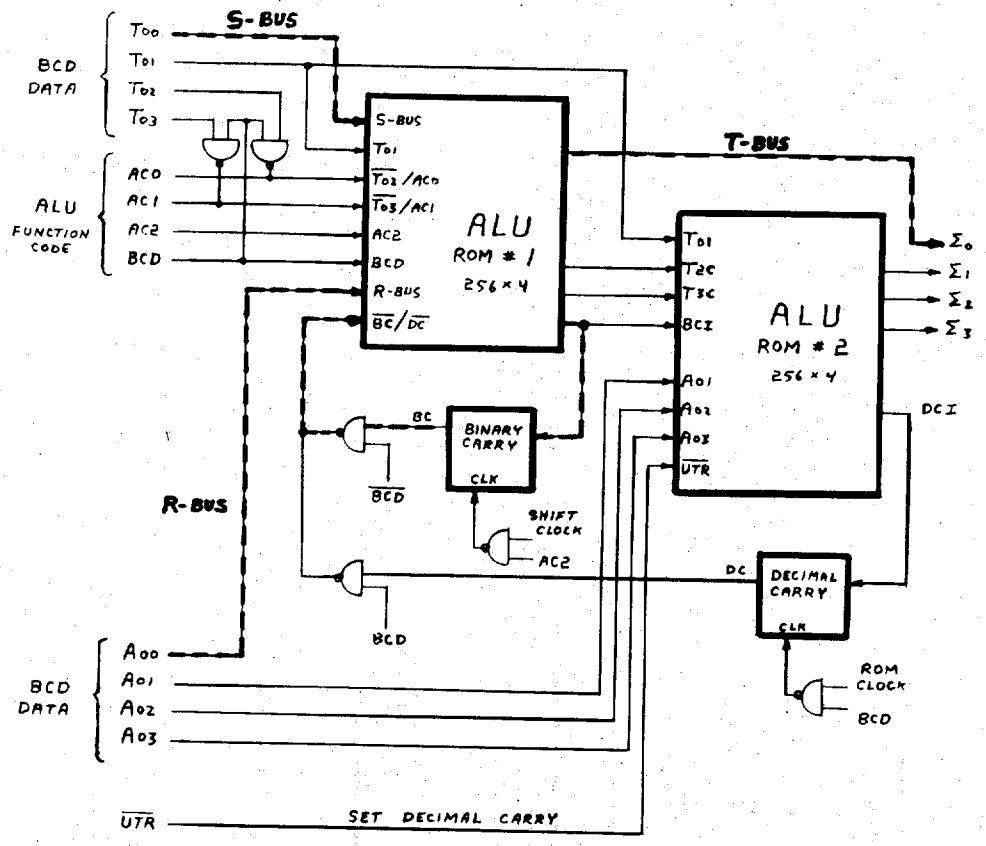


FIG 143

BINARY / BCD ARITH LOGIC UNIT

ALU CODE					ALU FUNCTION	DESCRIPTION
	BCD	AC2	AC1	AC0		
Binary Functions	0	0	0	0	XOR	Exclusive OR .... R ⊕ S → T
	0	0	0	1	AND	Logical AND .... R · S → T
	0	0	1	0	IOR	Inclusive OR .... R + S → T
	0	0	1	1	ZTT	Zero → T-BUS
	0	1	0	0	ZTT·CBC·	Zero → T-BUS, Clear Binary Carry
	0	1	0	1	IOR·CBC	Inclusive OR, Clear Binary Carry
	0	1	1	0	IOR·SBC	Inclusive OR, Set Binary Carry
	0	1	1	1	ADD	Binary ADD....R + S + BC → T,C
BCD Functions	1	0	1	1	BCD ADD	BCD ADD.... $T_{0-3} + A_{0-3} \rightarrow \Sigma_{0-3}$
	1	1	1	1	BCD COMP/ADD	IO's Complement and BCD ADD

FIG 144 ALU FUNCTION CODE ASSIGNMENTS

-HP- PART #	DESCRIPTION	REFERENCE
1820-9007 #1	1024-bit ROM	-HP- Dwg. #A-1820-9007-1A ROM #1
1820-9007 #2	1024-bit ROM	-HP- Dwg. #A-1820-9007-1A ROM #2
1820-0077	DUAL D-FLIP FLOP	SN7474 - T.I. I/C Catalog
1820-0370	Quad-2 NAND Gate	SN74H00 - T.I. I/C Catalog
1820-	Quad-2 NAND (D.C.)	SN74H01 - T.I. I/C Catalog

FIG 145 HARDWARE REQUIREMENTS: ALU

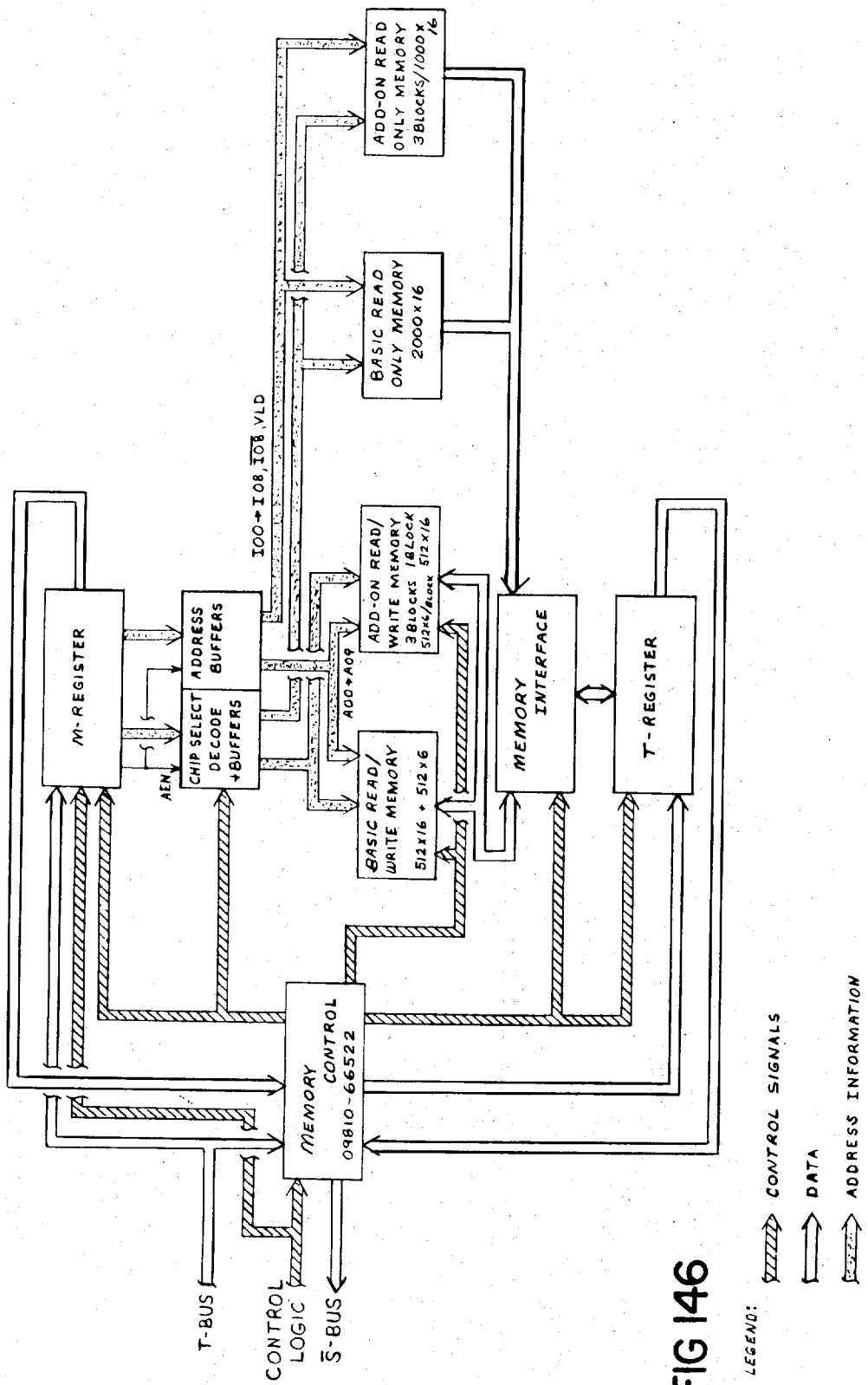

ADD-ON READ ONLY MEMORY 3 BLOCKS/1000 x 16
BASIC READ ONLY MEMORY 2000 x 16
I00→I08, I08, VLD
ADD-ON READ/ WRITE MEMORY 3 BLOCKS 1 BLOCK 512x6/block 512 x 16
M-REGISTER
ADDRESS BUFFERS
CHIP SELECT DECODE + BUFFERS
AEN
A00→A09
BASIC READ/ WRITE MEMORY 512 x 16 + 512 x 6
MEMORY INTERFACE
T-REGISTER
MEMORY CONTROL 09810-66522
T-BUS
CONTROL LOGIC
S-BUS
LEGEND:
CONTROL SIGNALS
DATA
ADDRESS INFORMATION

FIG 146

SIGNALS GENERATED OUTSIDE MEMORY I/O CONNECTOR

$\overline{CCT}$ – CONTROL CLOCK-NOT THE INVERTED ENVELOPE OF SCK

SCK– SHIFT CLOCK

MCK– MEMORY CLOCK, A CONTINUOUS PULSE TRAIN, USED BY THE MEMORY CONTROL FOR TIMING. OF THE MEMORY AND REFRESH CYCLES

IOD – I/O DATA. GOES TO CONTRL BOARD TO BE GATED TO S-BUS

ITS – I/O TO S-BUS, THE SIGNAL WHICH GATES IOD TO S-BUS

SCO & – CODED SIGNALS

SCI WHICH GENERATE UTS-UNITS TO S-BUS

ZTS-ZERO " "

MTS-M-RIG " "

TTS-T-REG " "

$\overline{TTT}$ – T-BUS TO T-REG, 0V = TRUE

T-BUS– DATA ON THIS BUS ACTS AS INPUTS TO M&T REGISTERS

S-BUS– DATA FROM M & T-REGISTERS, AS WELL AS I/O IS INVERTED AND OUTPUTED ON THIS BUS.

$\overline{RDM}$ – READ MEMORY, NEGATIVE TRUE. LASTS FOR 12 CLOCK PULSES

$\overline{WTM}$ – WRITE MEMORY, NEGATIVE TRUE. LASTS FOR 12 CLOCK PULSES

$\overline{INH}$ – INHIBIT, NEGATIVE TRUE. THE PROCESSOR IS STOPPED WHENEVER $\overline{INH}$ IS AT ZERO VOLTS. THE MEMORY CONTROL GENERATES THIS SIGNAL WHILE A R/W MEMORY REFRESH CYCLE IS PRESENT. I/O ALSO GENERATES IT.

OTHER SIGNALS AT I/O CONNECTOR

NAME	SOURCE
T00	T-REGISTER
T01	" "
T02	" "
T03	" "

SIGNALS GENERATED ON READ/WRITE MEMORY CARDS.

RWD(XX)– READ/WRITE DATA. OUTPUT FROM THE 1103 MEMORY, 600μa INTO 150Ω = "1" 0 CURRENT = 0

OTHER SIGNALS USED BY (R/WM)

NAME	SOURCE
A00-A04	CONTROL
A05-A09	M-REG
CSB(XX)	M-REG
CSA(XX)	M-REG
RWE(XX)	T-REG
YBL	CONTROL
PCG	CONTROL

SIGNALS GENERATED ON T-REGISTER CARD

T00-T15– T-REGISTER DATA BITS. USED AS DATA INTO MEMORY T00-T03 ARE ALSO OUTPUTS TO THE CPU. (4 BIT PROCESSING)

RWI(XX) READ/WRITE INPUTS. T-REGISTER DATA GATED WITH WE1, WE2 TO BECOME INPUT DATA TO MEMORY. +16V → GND.

OTHER SIGNALS USED BY T-REGISTER

NAME	SOURCE
ROD(XX)	ROM
TRI	CONTROL
TSC	CONTROL
TMC	CONTROL
TPC	CONTROL
WE1	CONTROL
WE2	CONTROL
RWD(XX)	R/W MEM
RWE	M-REG

SIGNALS GENERATED ON M-REGISTER CARD

M00-M15– M-REGISTER DATA BITS. USED TO GENERATE ADDRESS AND CHIP SELECT INFORMATION. (M00 ALSO IS GATED OUT ON S-BUS BY MTS)

I00-I07– ROM ADDRESS BITS. DECODES DOWN TO TWO BITS AVAILABLE AT ROM OUTPUT BUFFERS.

I08-$\overline{I08}$– SELECTS WHICH ROM OUTPUT BUFFERS ARE ENABLED

CEB(XX)– CHIP ENABLE, BASIC MACHINE. SELECTS WHICH ROM CHIPS ARE TURNED ON. (+12V=ON)

CEA(XX)– CHIP ENABLE, ADD-ON FUNCT.

AEN– ADDRESS ENABLE, AEN = ROM+WTM

A05-A09– ADDRESS BITS FOR R/W MEMORY. (+16V & GND)

CSB(XX)– CHIP SELECT, BASIC MACHINE A NEGATIVE TRUE CLOCK WHICH SELECTS WHICH R/W CHIPS ARE TURNED ON. (+16V & GND)

CSA(XX)– CHIP SELECT, ADD-ON.

RWE – READ/WRITE ENABLE. A +5V SIGNAL ANY TIME A R/W CHIP IS ADDRESSED FOR A MACHINE MEMORY CYCLE.

OTHER SIGNALS USED BY THE M-REGISTER

NAME	SOURCE
T-BUS	PROCESSOR
SCK	"
$\overline{RDM}$	"
$\overline{WTM}$	"
$\overline{TTT}$	"
VLD	CONTROL
MTS	"
$\overline{CSL}$	"
$\overline{REF}$	"

SIGNALS GENERATED ON CONTROL BOARD

VLD–V– LOAD, A SIGNAL GENERATED HALF WAY THROUGH THE MEMORY CYCLE TO DISABLE THE ACTIVE PULL UP DEVICES ON THE ROM OUTPUTS

TRI–T-REGISTER INPUT. T-BUS AND TTT OR T00 AND TTS

MTS–M-REGISTER TO S-BUS. GENERATED FROM SCO·$\overline{SCI}$

A00-A04– ADDRESS BITS FOR R/W MEMORY ALSO USED DURING MEMORY REFRESH.

TSC– T-REGISTER SERIES CLOCK. TSC=SCK·(TTT+TTS)

TPC – T-REGISTER PARALLEL CLOCK. (STROBES IN DATA FROM MEMORY)

WE1– WRITE ENABLE 1 GATES ODD T-REG DATA TO MEMORY DURING 1ST HALF OF MEMORY CYCLE

WE2– WRITE ENABLE 2 GATES EVEN T-REG DATA TO MEMORY DURING THE LAST HALF OF A MEMORY CYCLE

YBL–Y–ENABLE. A CLOCK WHICH LEFT AT +16V FOR A READ, AND CLOCKED TO GND DURING A WRITE. (R/W MEMORY ONLY)

PCG– PRECHARGE. THE 3RD 16V CLOCK REQUIRED BY THE 1103 R/W MEMORY CHIPS

CSL– CHIP SELECT CLOCK. THE SIGNAL WHICH IS ANDED WITH M-REGISTER DATA TO PROVIDE CHIP SELECTS.

$\overline{REF}$– REFRESH. 0V WHEN THE MEMORY IS IN A REFRESH CYCLE.

OTHER SIGNALS USED BY THE CONTROL

NAME	SOURCE
$\overline{CCT}$	PROCESSOR
T00	T-REG
IOD	PROCESSOR
ITS	"
SCO	"
$\overline{SCI}$	"
$\overline{TTT}$	"
T-BUS	"
$\overline{S\text{-}BUS}$	"
SCK	"
$\overline{RDM}$	"
$\overline{WTM}$	"
MCK	"
$\overline{INH}$	"
M00-M03	M-REG
AEN	M-REG

SIGNALS GENERATED ON ROM BOARD

ROD(XX)–READ ONLY DATA

OTHER SIGNALS USED BY ROM

NAME	SOURCE
CEA(XX)	M-REG
CEB(XX)	M-REG
I00-I07	M-REG
I08, $\overline{I08}$	M-REG
VLD	CONTROL

FIG 147

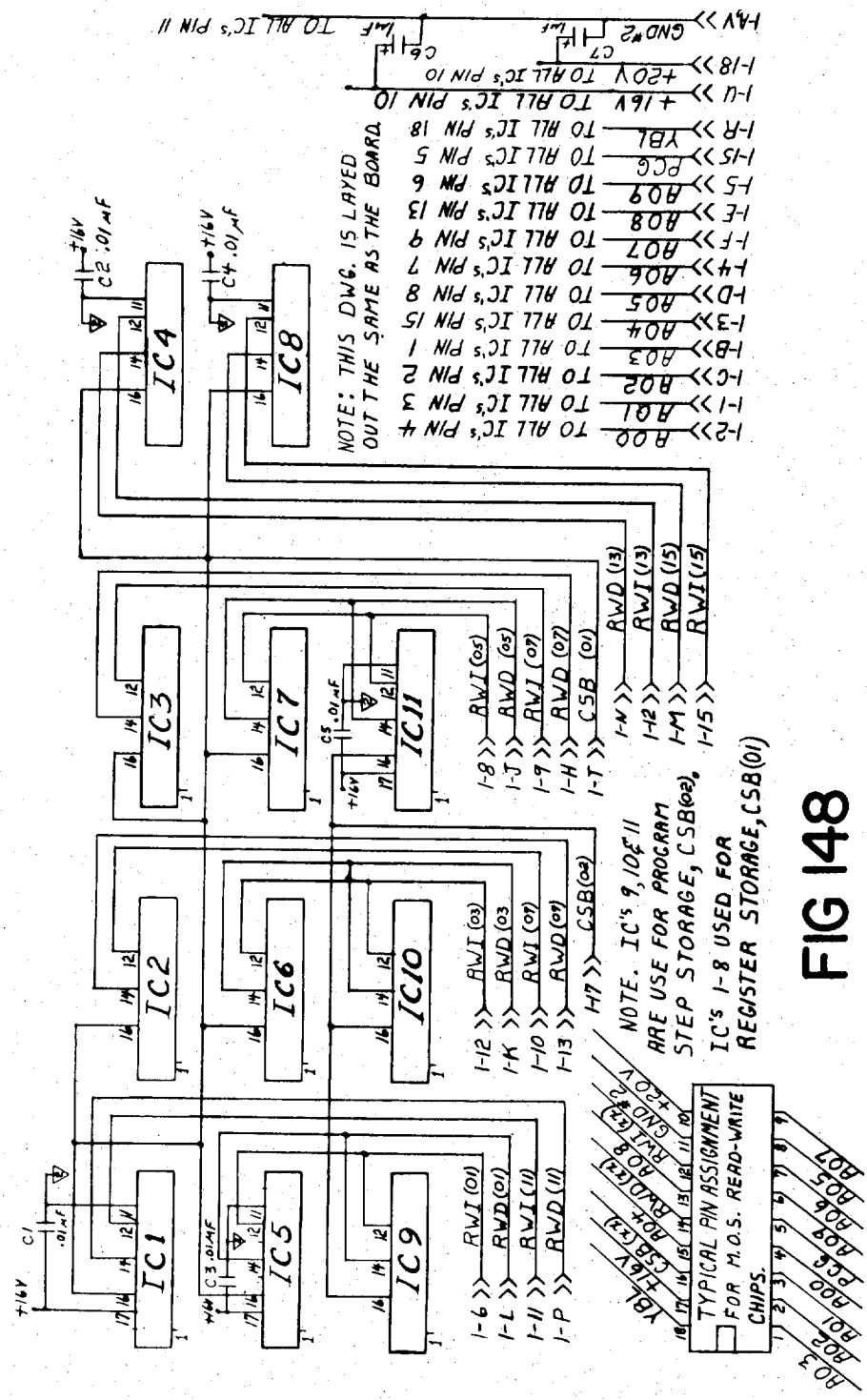

NOTE: THIS DWG. IS LAYED OUT THE SAME AS THE BOARD.
+16V TO ALL IC's PIN 10
+20V TO ALL IC's PIN 10
GND #2
TO ALL IC's PIN 11
I-2 >> A00 TO ALL IC's PIN 4
I-1 >> A01 TO ALL IC's PIN 3
I-C >> A02 TO ALL IC's PIN 2
I-B >> A03 TO ALL IC's PIN 1
I-3 >> A04 TO ALL IC's PIN 15
I-D >> A05 TO ALL IC's PIN 8
I-4 >> A06 TO ALL IC's PIN 7
I-F >> A07 TO ALL IC's PIN 9
I-E >> A08 TO ALL IC's PIN 13
I-5 >> A09 TO ALL IC's PIN 6
I-15 >> PCG TO ALL IC's PIN 5
I-R >> YBL TO ALL IC's PIN 18
I-U >> +16V
I-18 >> +20V
I-AV >> GND #2
IC1
IC2
IC3
IC4
IC5
IC6
IC7
IC8
IC9
IC10
IC11
C1 .01 mf
C2 .01 mf
C3 .01 mf
C4 .01 mf
C5 .01 mf
C6
C7
+16V
I-6 >> RWI (01)
I-7 >> RWD (01)
I-11 >> RWI (11)
I-P >> RWD (11)
I-12 >> RWI (03)
I-K >> RWD (03)
I-10 >> RWI (07)
I-13 >> RWD (07)
I-17 >> CSB (04)
I-8 >> RWI (05)
I-J >> RWD (05)
I-9 >> RWI (07)
I-H >> RWD (07)
I-T >> CSB (01)
I-W >> RWD (13)
I-12 >> RWI (13)
I-M >> RWD (15)
I-15 >> RWI (15)
NOTE. IC's 9, 10 & 11 ARE USE FOR PROGRAM STEP STORAGE, CSB(02), IC's 1-8 USED FOR REGISTER STORAGE, CSB(01)
TYPICAL PIN ASSIGNMENT FOR M.O.S. READ-WRITE CHIPS.
YBL
+16V
CSB (x)
A04 (x)
RWD (x)
A08 (x)
RWI (x)
GND (x)
+20V
A07
A09
PCG
A00
A01
A06
A03
A05
A02

FIG 148

CONNECTOR PIN ASSIGNMENTS (P1)

A	GND	1	A01
B	A03	2	A00
C	A02	3	A04
D	A05	4	A06
E	A08	5	A09
F	A07	6	RWI Ø1
H	RWD Ø7	7	RWI Ø3
J	RWD Ø5	8	RWI Ø5
K	RWD Ø3	9	RWI Ø7
L	RWD Ø1	10	RWI Ø9
M	RWD 15	11	RWI 11
N	RWD 13	12	RWI 13
P	RWD 11	13	RWD Ø9
R	YBL	14	RWI 15
S	CSA Ø1	15	PCG
T	CSA Ø3	16	CSA Ø2
U	+VSS(+16V)	17	CSA Ø4
V	GND	18	+VCC(+20V)

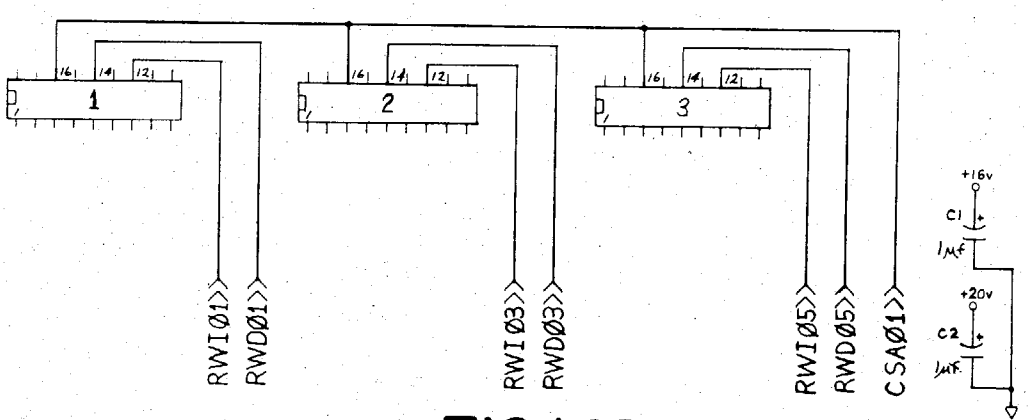
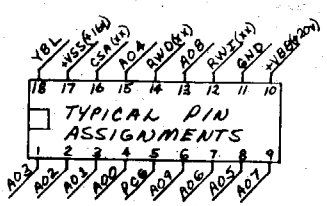


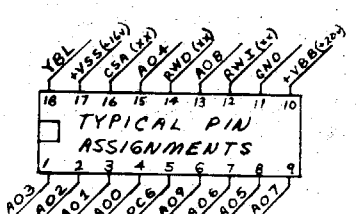
FIG 149



NOTE: PIN ASSIGNMENTS ARE COMMON TO ALL IC'S, UNLESS OTHERWISE SHOWN.

CONNECTOR PIN ASSIGNMENTS (P1)

A	GND	1	AO1
B	AO3	2	AOO
C	AO2	3	AO4
D	AO5	4	AO6
E	AO8	5	AO9
F	AO7	6	RWI Ø1
H	RWDØ7	7	RWI Ø3
J	RWDØ5	8	RWI Ø5
K	RWDØ3	9	RWI Ø7
L	RWDØ1	10	RWI Ø9
M	RWD 15	11	RWI 11
N	RWD 13	12	RWI 13
P	RWD 11	13	RWD Ø9
R	YBL	14	RWI 15
S	CSA Ø1	15	PCG
T	CSA Ø3	16	CSA Ø2
U	+VSS(-16V)	17	CSA Ø4
V	GND	18	VCC (+20V)



NOTE: PIN ASSIGNMENTS ARE COMMON TO ALL IC'S, UNLESS OTHERWISE SHOWN.

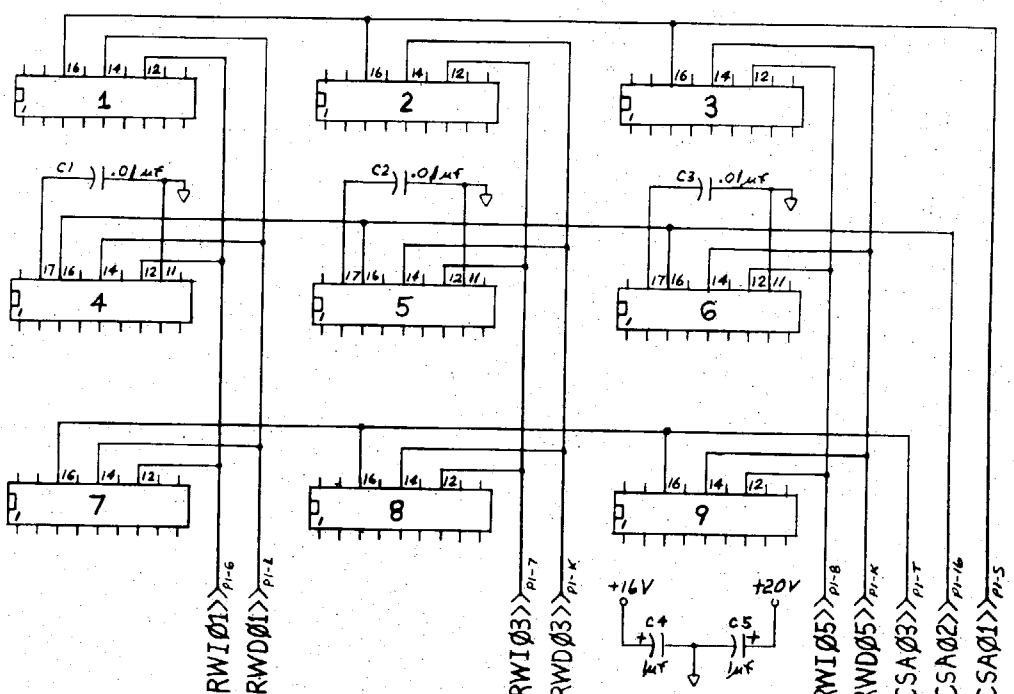


FIG 150

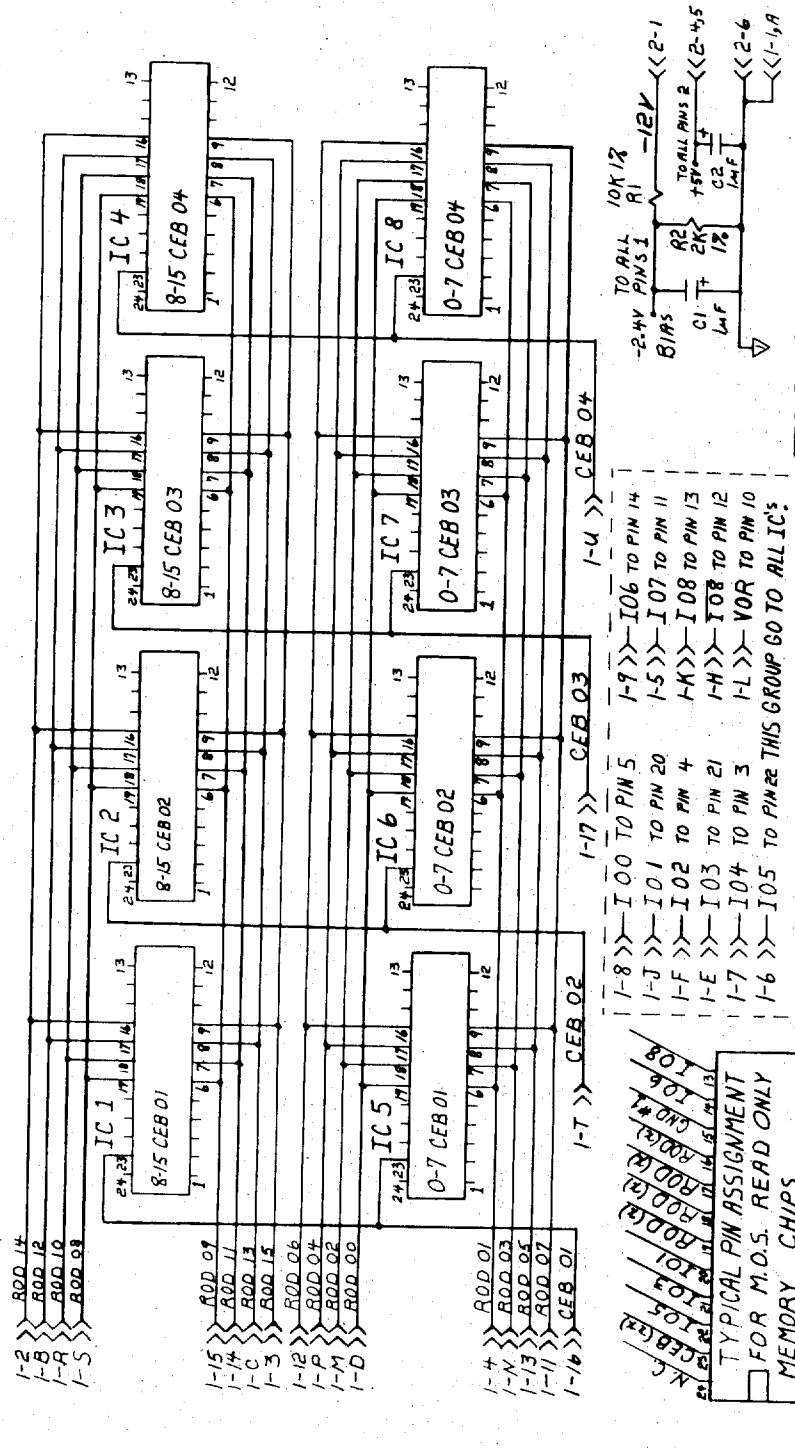

IC 1
8-15 CEB 01
IC 2
8-15 CEB 02
IC 3
8-15 CEB 03
IC 4
8-15 CEB 04
IC 5
0-7 CEB 01
IC 6
0-7 CEB 02
IC 7
0-7 CEB 03
IC 8
0-7 CEB 04
CEB 01
CEB 02
CEB 03
CEB 04
TO ALL PINS 1
TO ALL PINS 2
-2.4V BIAS
-12V
10K 1%
R1
R2 2K 1%
C1 1mF
C2 1mF
+5V
TYPICAL PIN ASSIGNMENT FOR M.O.S. READ ONLY MEMORY CHIPS.
I-6 >> I05 TO PIN 22 THIS GROUP GO TO ALL IC's
I-7 >> I04 TO PIN 3
I-E >> I03 TO PIN 21
I-F >> I02 TO PIN 4
I-J >> I01 TO PIN 20
I-8 >> I00 TO PIN 5
I-L >> VOR TO PIN 10
I-H >> I08 TO PIN 12
I-K >> I08 TO PIN 13
I-5 >> I07 TO PIN 11
I-9 >> I06 TO PIN 14

FIG 151

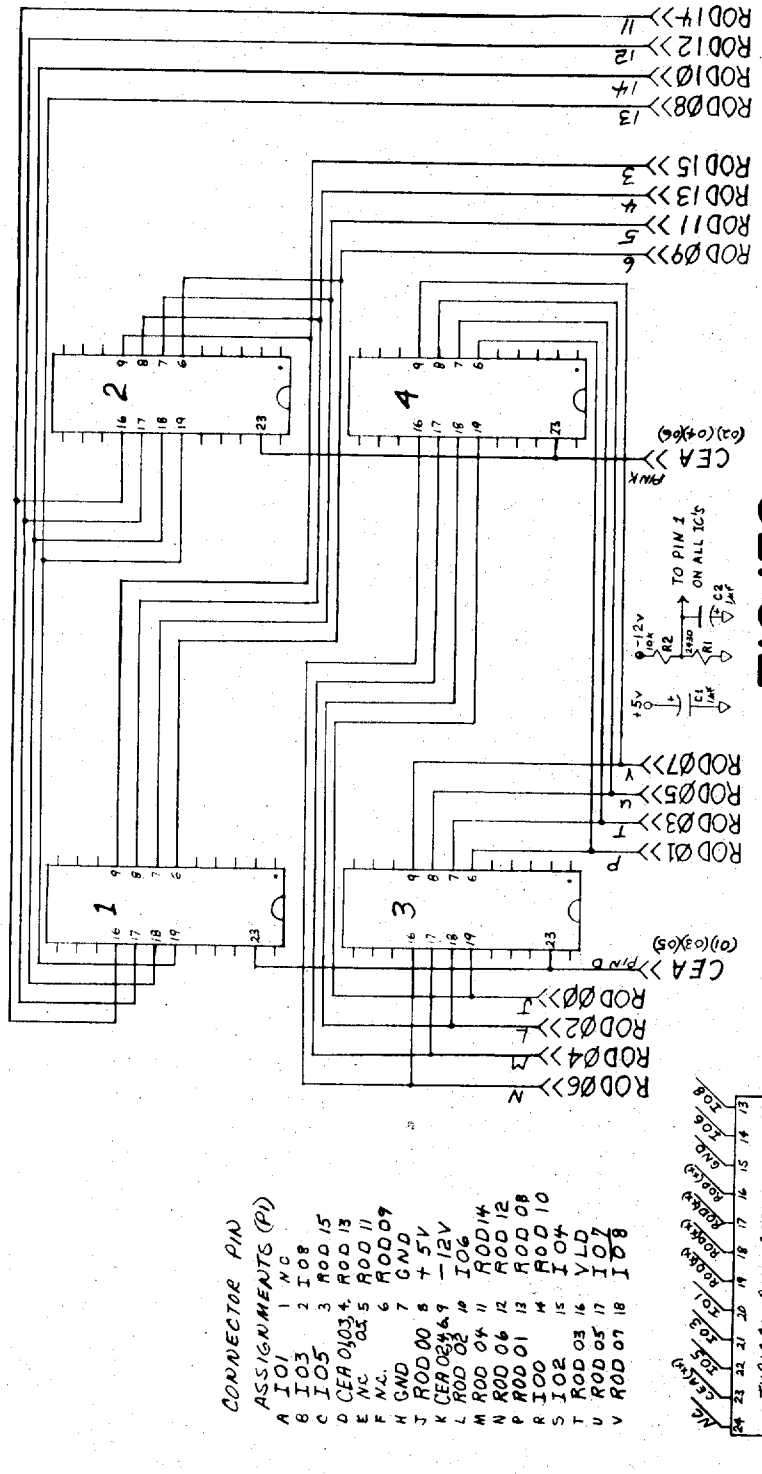

ROD14
ROD12
ROD10
ROD08
ROD15
ROD13
ROD11
ROD09
CEA (02)(04)(06)
ROD07
ROD05
ROD03
ROD01
CEA (01)(03)(05)
ROD00
ROD02
ROD04
ROD06
+5V
-12V
TO PIN 1 ON ALL ICS
SLOT #1 = MATH BLOCK CEA 01, 02
SLOT #2 = STATISTICAL/ TYPEWRITER, CALCULABLE FUNCTION CEA 03, 04.
SLOT #3 ALPHA PRINTER CEA 05, 06
TYPICAL PIN ASSIGNMENT
CONNECTOR PIN ASSIGNMENTS (P1)
A IO1 1 N.C.
B IO3 2 IO8
C IO5 3 ROD 15
D CEA 01,03,4 ROD 13
E N.C. 05 5 ROD 11
F N.C. 6 ROD 09
H GND 7 GND
J ROD 00 8 +5V
K CEA 02,04,6,9 -12V
L ROD 02 10 IO6
M ROD 04 11 ROD 14
N ROD 06 12 ROD 12
P ROD 01 13 ROD 08
R IO0 14 ROD 10
S IO2 15 IO4
T ROD 03 16 VLD
U ROD 05 17 IO7
V ROD 07 18 IO8

FIG 152

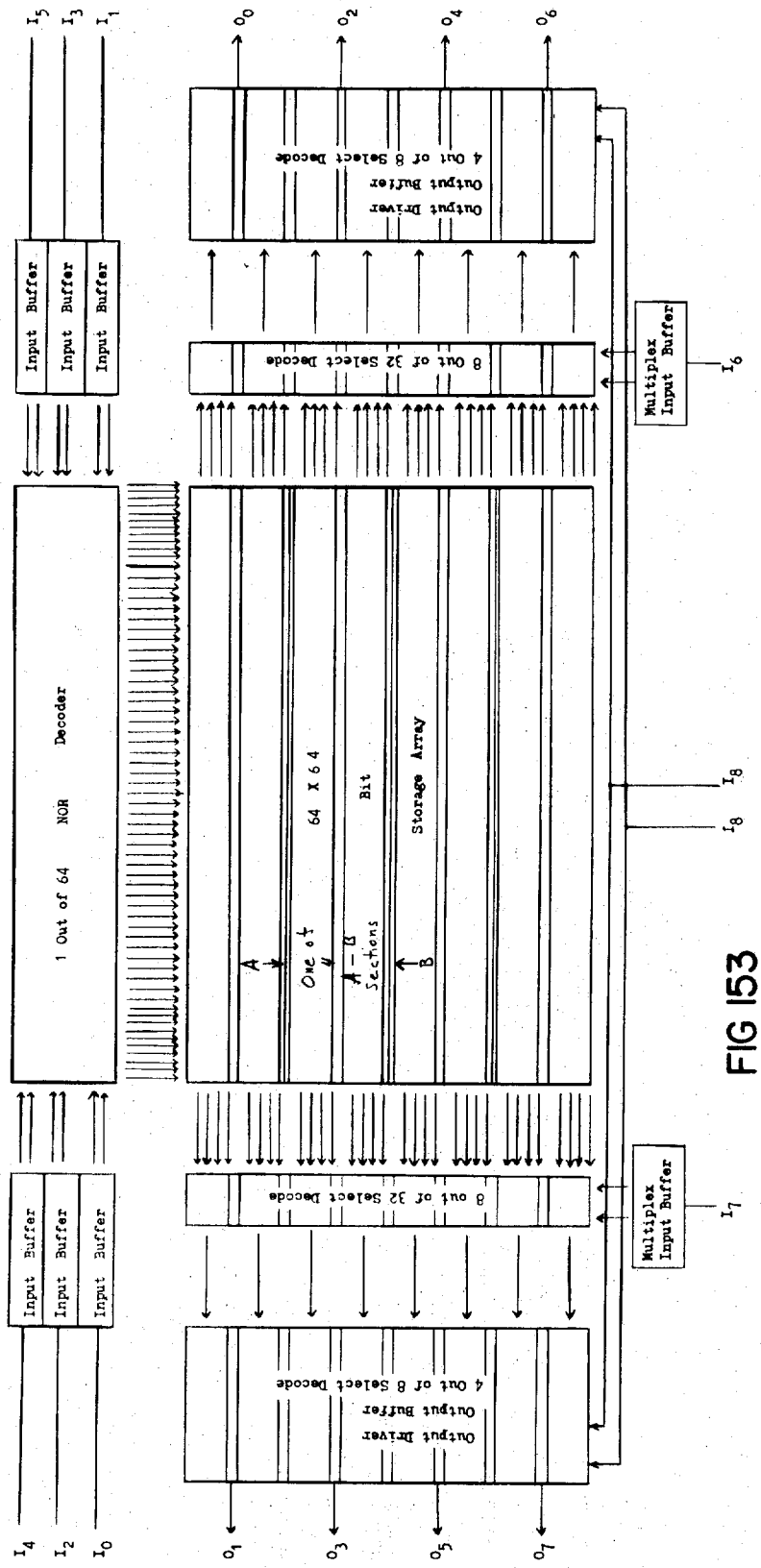
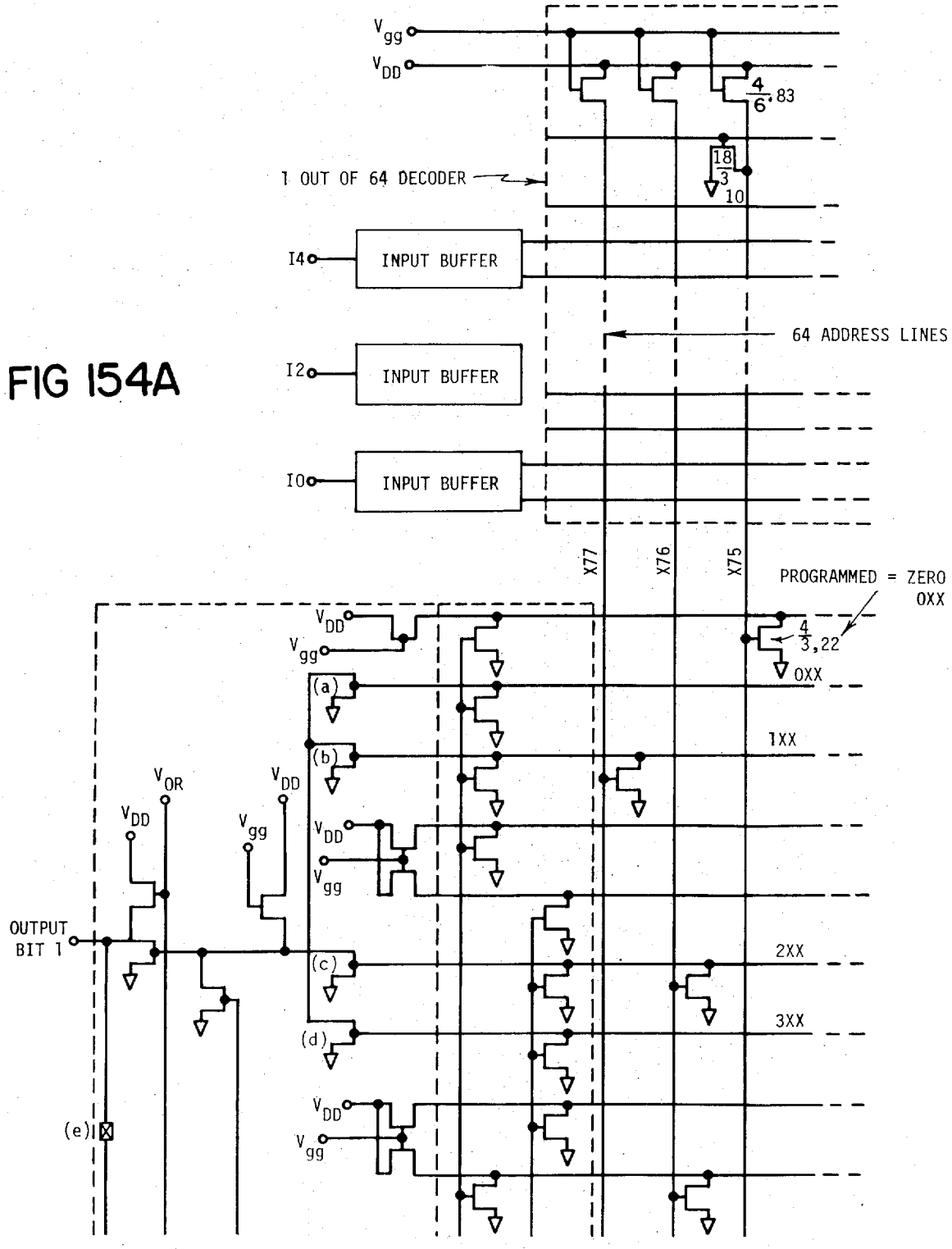

I5
I3
I1
O0
O2
O4
O6
Input Buffer
Input Buffer
Input Buffer
Output Driver
Output Buffer
4 Out of 8 Select Decode
8 Out of 32 Select Decode
Multiplex Input Buffer
I6
1 Out of 64 NOR Decoder
A
One of
A - B Sections
64 X 64 Bit Storage Array
B
I8
I8
8 out of 32 Select Decode
Multiplex Input Buffer
I7
Output Driver
Output Buffer
4 Out of 8 Select Decode
I4
I2
I0
O1
O3
O5
O7

FIG 153

FIG 154A


V_{gg}
V_{DD}
4/6,83
1 OUT OF 64 DECODER
18/3/10
I4
INPUT BUFFER
64 ADDRESS LINES
I2
INPUT BUFFER
I0
INPUT BUFFER
X77
X76
X75
PROGRAMMED = ZERO
0XX
4/3,22
0XX
1XX
2XX
3XX
V_{OR}
OUTPUT BIT 1
(a)
(b)
(c)
(d)
(e)

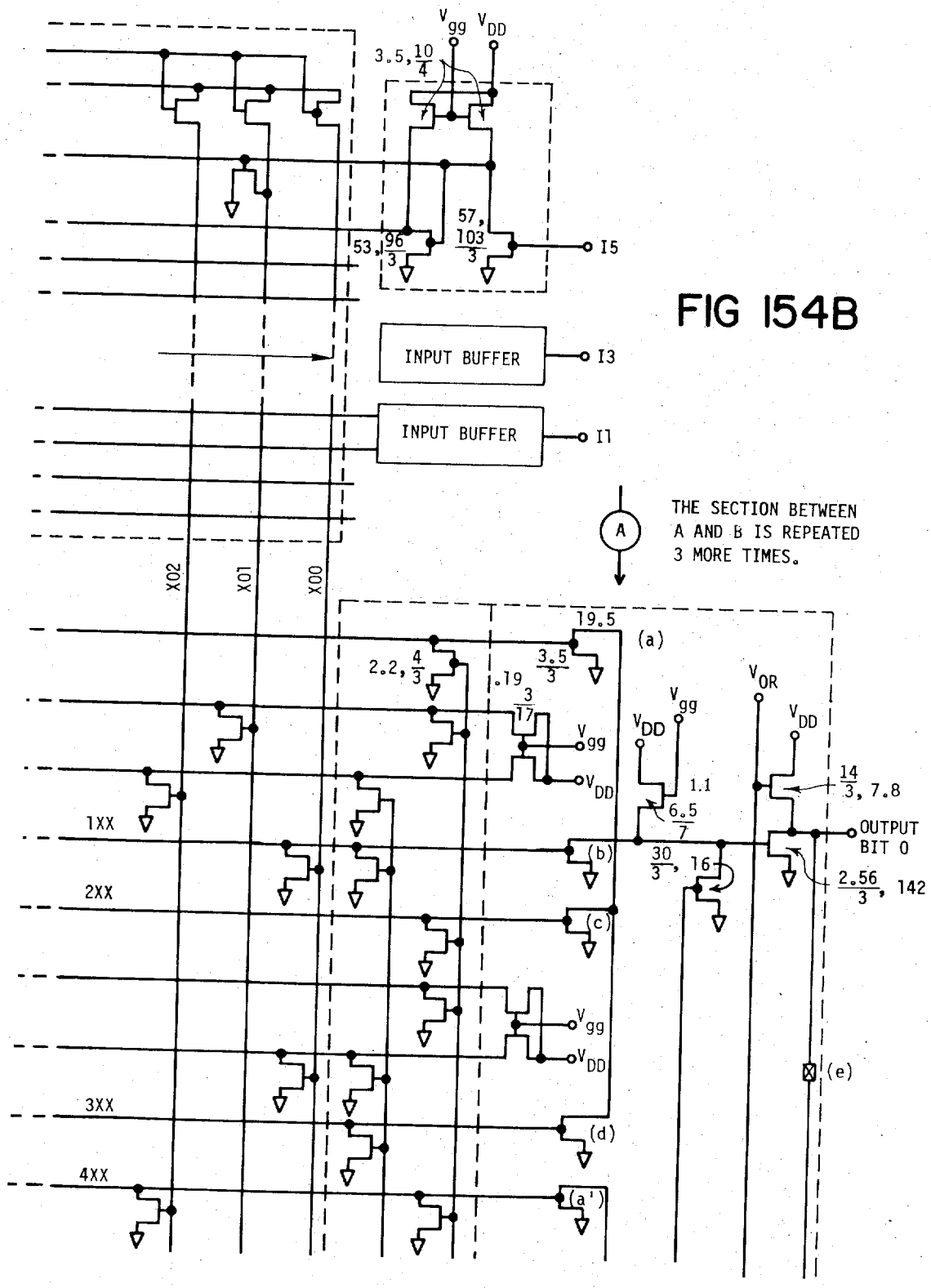

V_{gg} V_{DD}
3.5, 10/4
57, 103/3
53, 96/3
I5
INPUT BUFFER
I3
INPUT BUFFER
I1
A
THE SECTION BETWEEN A AND B IS REPEATED 3 MORE TIMES.
X02
X01
X00
19.5
(a)
2.2, 4/3
.19
3.5/3
3/17
V_{gg}
V_{DD}
1XX
2XX
3XX
4XX
(b)
(c)
(d)
(a')
V_{OR}
1.1
6.5/7
30/3, 16
14/3, 7.8
OUTPUT BIT 0
2.56/3, 142
(e)

FIG 154B

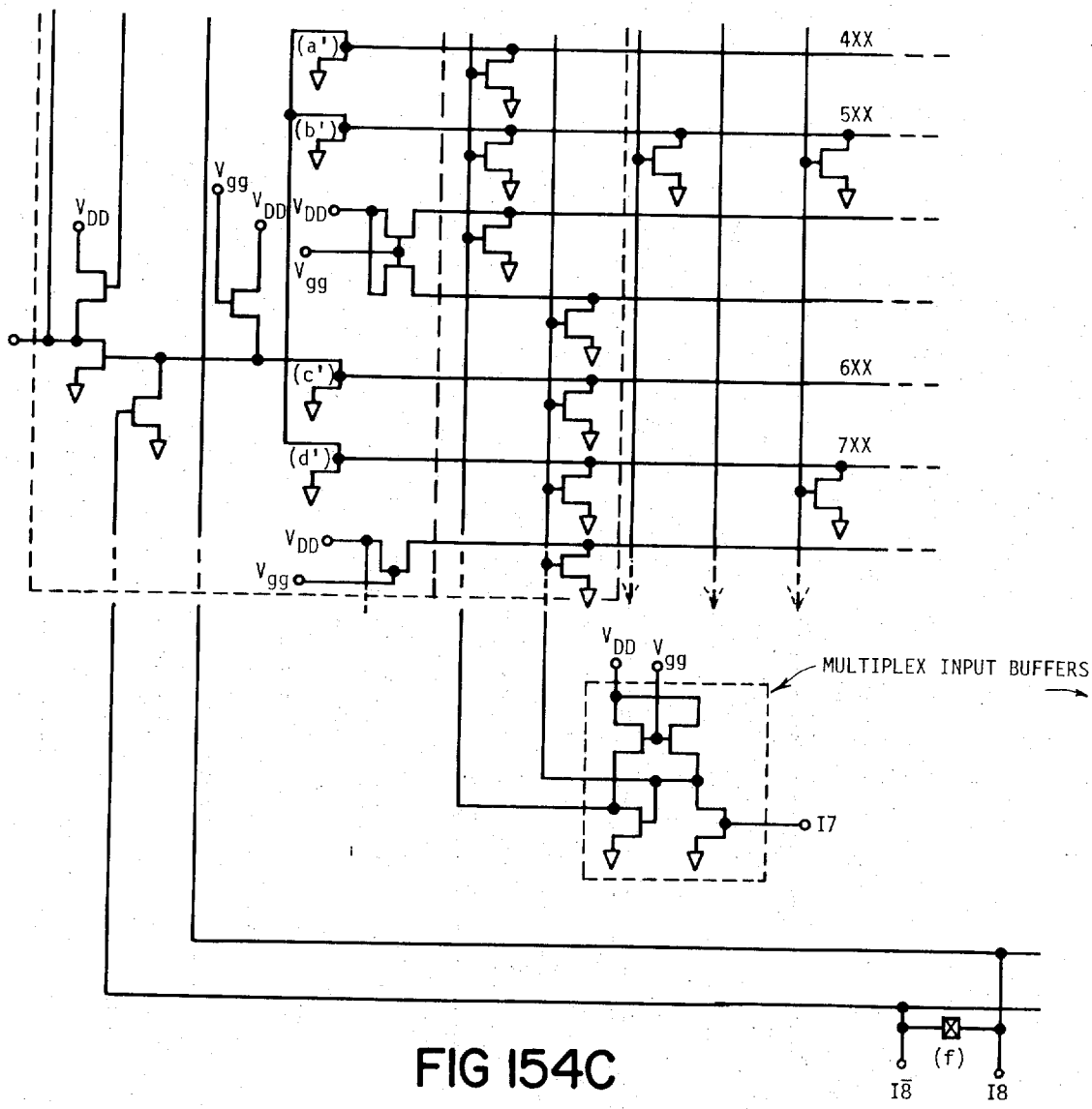

(a')
(b')
(c')
(d')
4XX
5XX
6XX
7XX
V_{DD}
V_{gg}
MULTIPLEX INPUT BUFFERS
I7
(f)
$\overline{I8}$
I8

FIG 154C

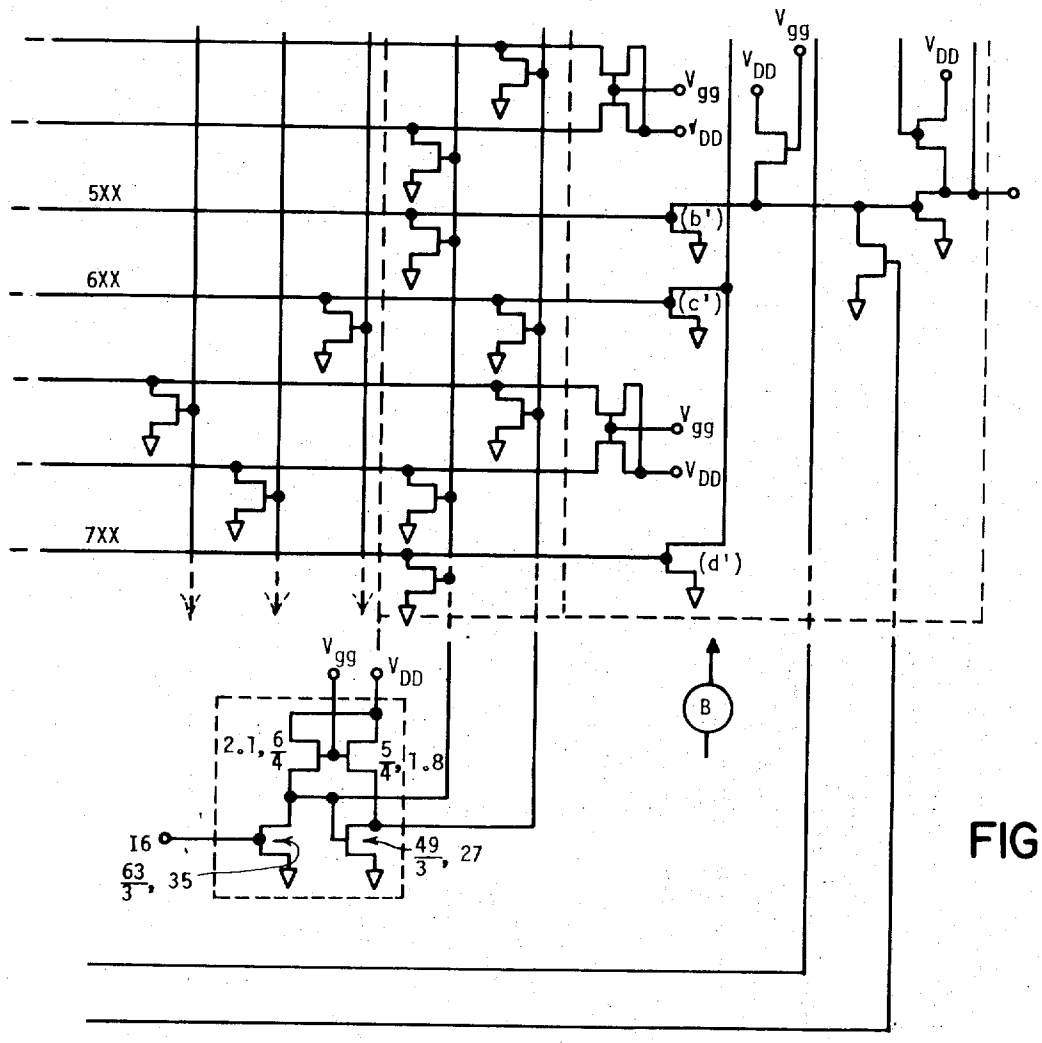


FIG 154D

NOTES ⟶

THIS SYMBOL INDICATES CONNECTIONS WHICH CAN BE PROGRAMMED OPEN OR SHORT. ⊠

(1) FOR A 512 x 8:
- (a) TIE OUTPUT PAIRS TOGETHER
- (b) DO NOT TIE I8 AND $\overline{I8}$ TOGETHER

(2) FOR A 256 x 16:
- (a) DO NOT TIE OUTPUT PAIRS TOGETHER
- (b) TIE I8 AND $\overline{I8}$ TOGETHER

(3) V_{DD} = +5V V_{gg} = +12V (SWITCHED)
(POSITIVE TRUE LOGIC)

(4) RESISTANCES ASSUME 10Ω □
ALL W/l's ASSUME A 3μ TOTAL SIDE DIFFUSION

ALL CONTACTS SHOWN AS (⊠) ARE PROGRAMMABLE WITH GATE MASK

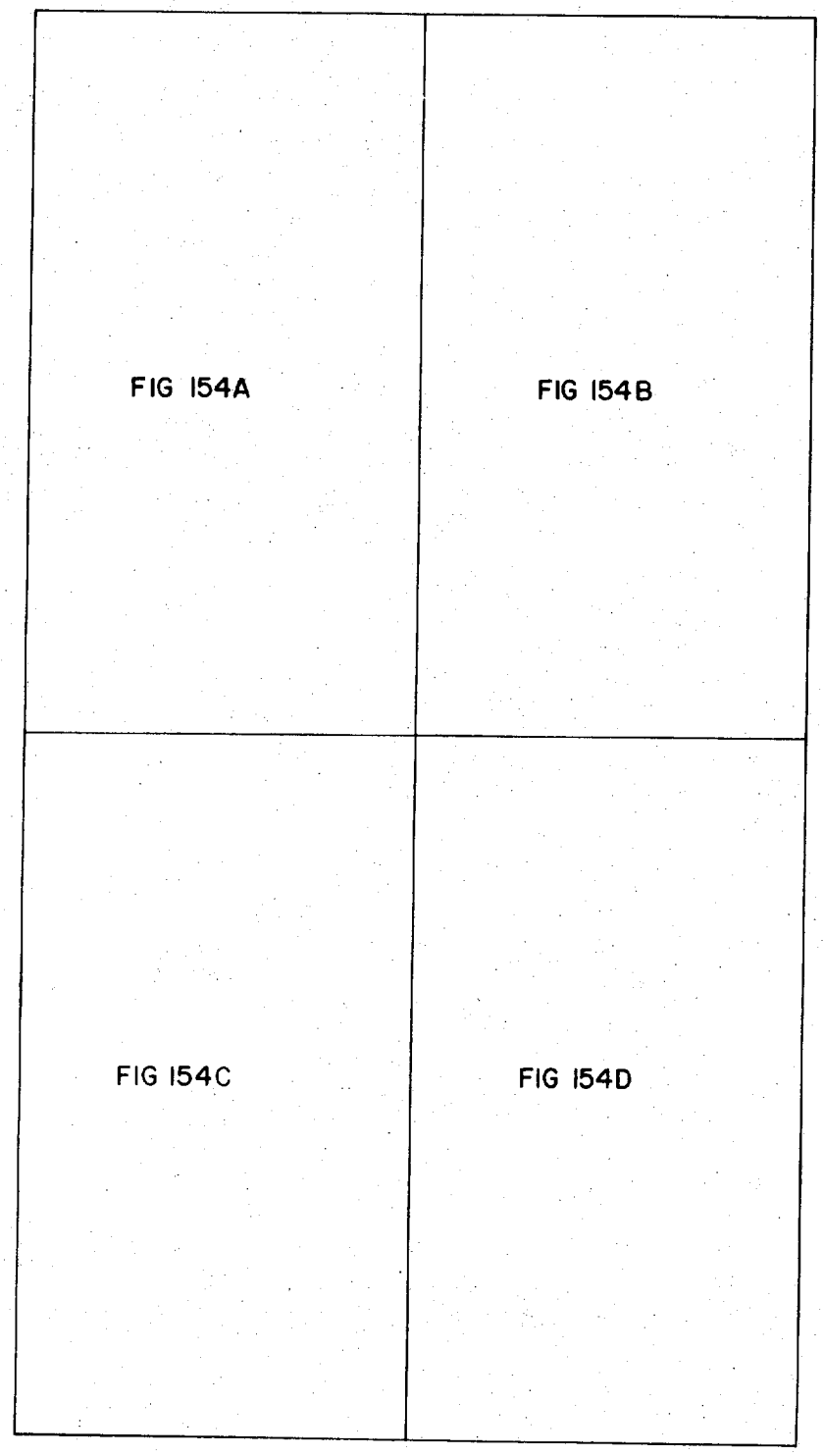


FIG 154'

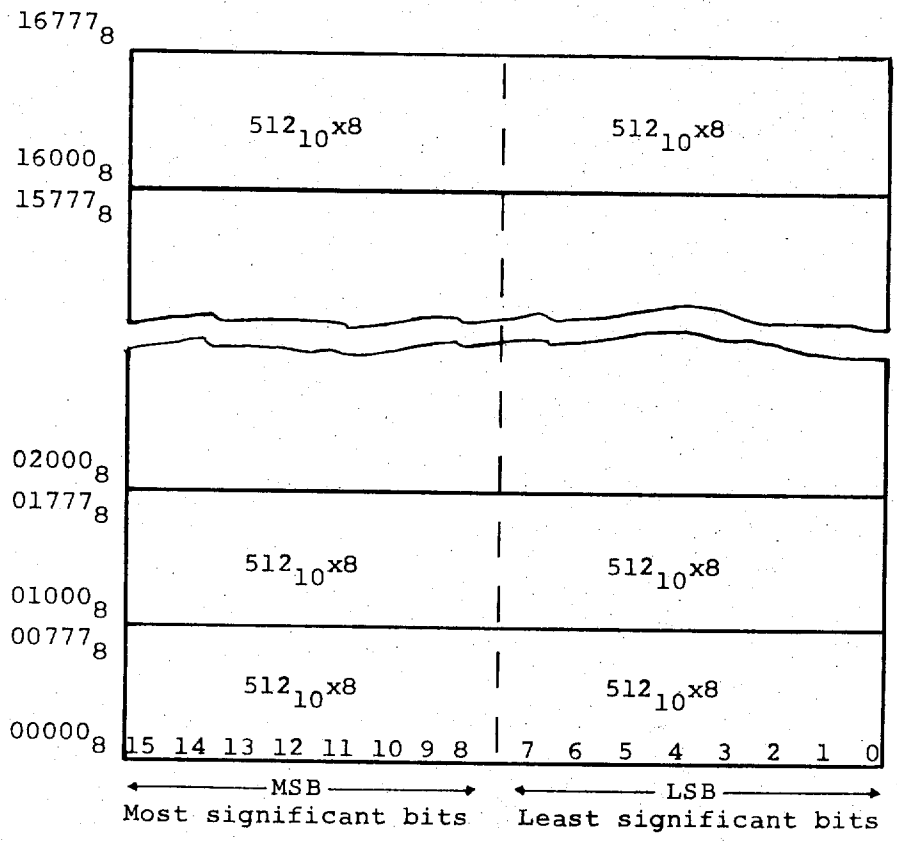

16777_8
16000_8
15777_8
02000_8
01777_8
01000_8
00777_8
00000_8
512_{10}x8
512_{10}x8
512_{10}x8
512_{10}x8
512_{10}x8
512_{10}x8
15 14 13 12 11 10 9 8
7 6 5 4 3 2 1 0
MSB
Most significant bits
LSB
Least significant bits

FIG 155

Bit Numbers	Actual Bit
1	15 or 7
2	14 or 6
3	13 or 5
4	12 or 4
5	11 or 3
6	10 or 2
7	9 or 1
8	8 or 0

FIG 157

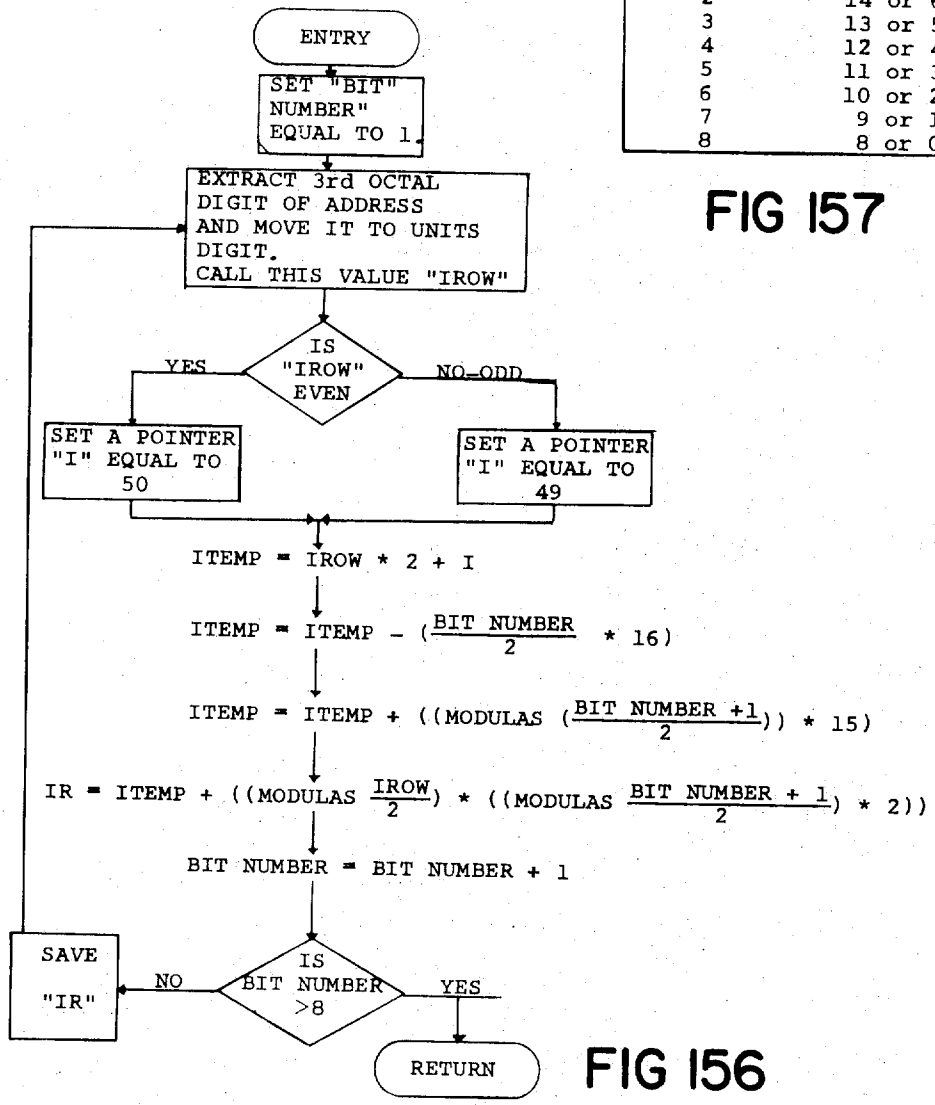


FIG 156

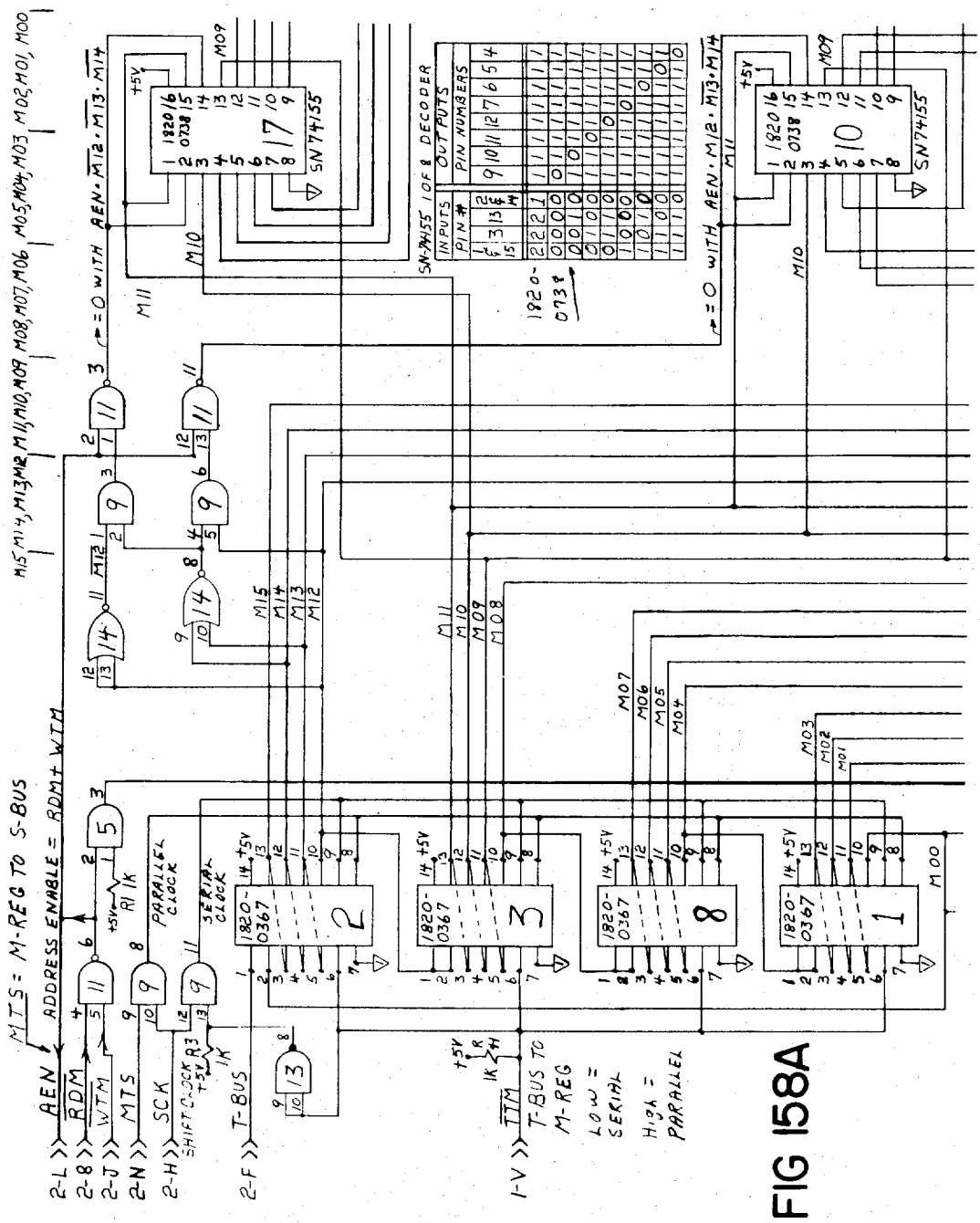

MTS = M-REG TO S-BUS
ADDRESS ENABLE = RDM+WTM
AEN
RDM
WTM
MTS
SCK
SHIFT CLOCK
T-BUS
TTM
T-BUS TO M-REG
LOW = SERIAL
HIGH = PARALLEL
PARALLEL CLOCK
SERIAL CLOCK
SN74155
1820-0738
1820-0367
SN-74155 1 OF 8 DECODER
INPUTS
OUTPUTS
PIN #
PIN NUMBERS
= 0 WITH AEN · M12 · M13 · M14
= 0 WITH AEN · M12 · M13 · M14
M15 M14, M13 M12 M11, M10, M09 M08, M07, M06 M05, M04, M03 M02, M01, M00

FIG 158A

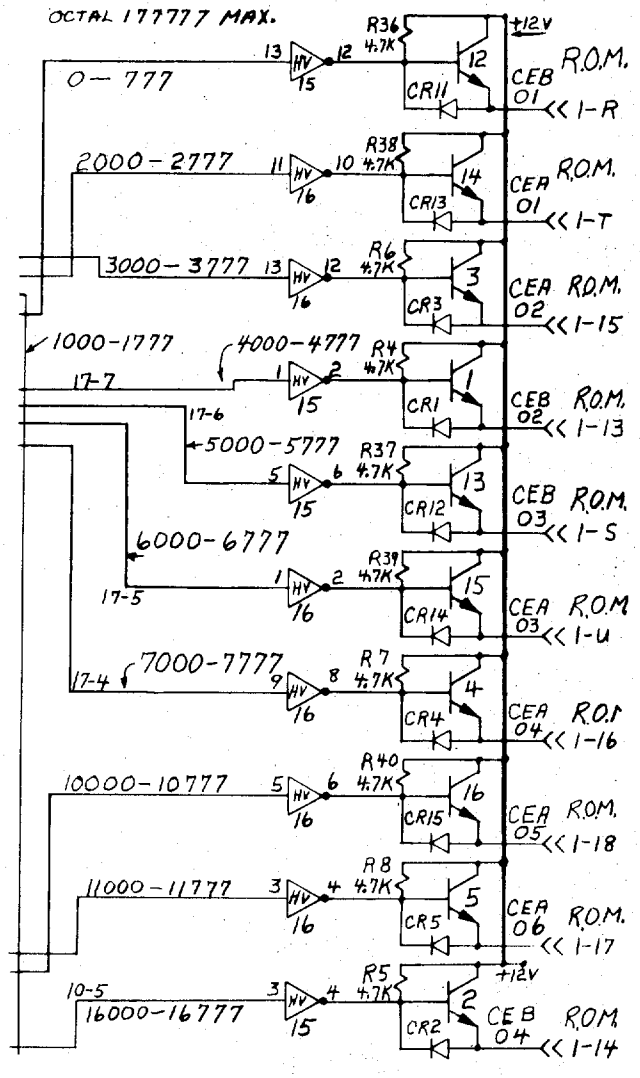

OCTAL 177777 MAX.
0 — 777
2000 — 2777
3000 — 3777
1000-1777
4000-4777
17-7
17-6
5000-5777
6000-6777
17-5
7000-7777
17-4
10000-10777
11000-11777
10-5
16000-16777
+12V
R36 4.7K
R38 4.7K
R6 4.7K
R4 4.7K
R37 4.7K
R39 4.7K
R7 4.7K
R40 4.7K
R8 4.7K
R5 4.7K
CR11
CR13
CR3
CR1
CR12
CR14
CR4
CR15
CR5
CR2
CEB 01 R.O.M. << 1-R
CEA 01 R.O.M. << 1-T
CEA 02 R.O.M. << 1-15
CEB 02 R.O.M. << 1-13
CEB 03 R.O.M. << 1-S
CEA 03 R.O.M. << 1-U
CEA 04 R.O.M. << 1-16
CEA 05 R.O.M. << 1-18
CEA 06 R.O.M. << 1-17
CEB 04 R.O.M. << 1-14

FIG 158B

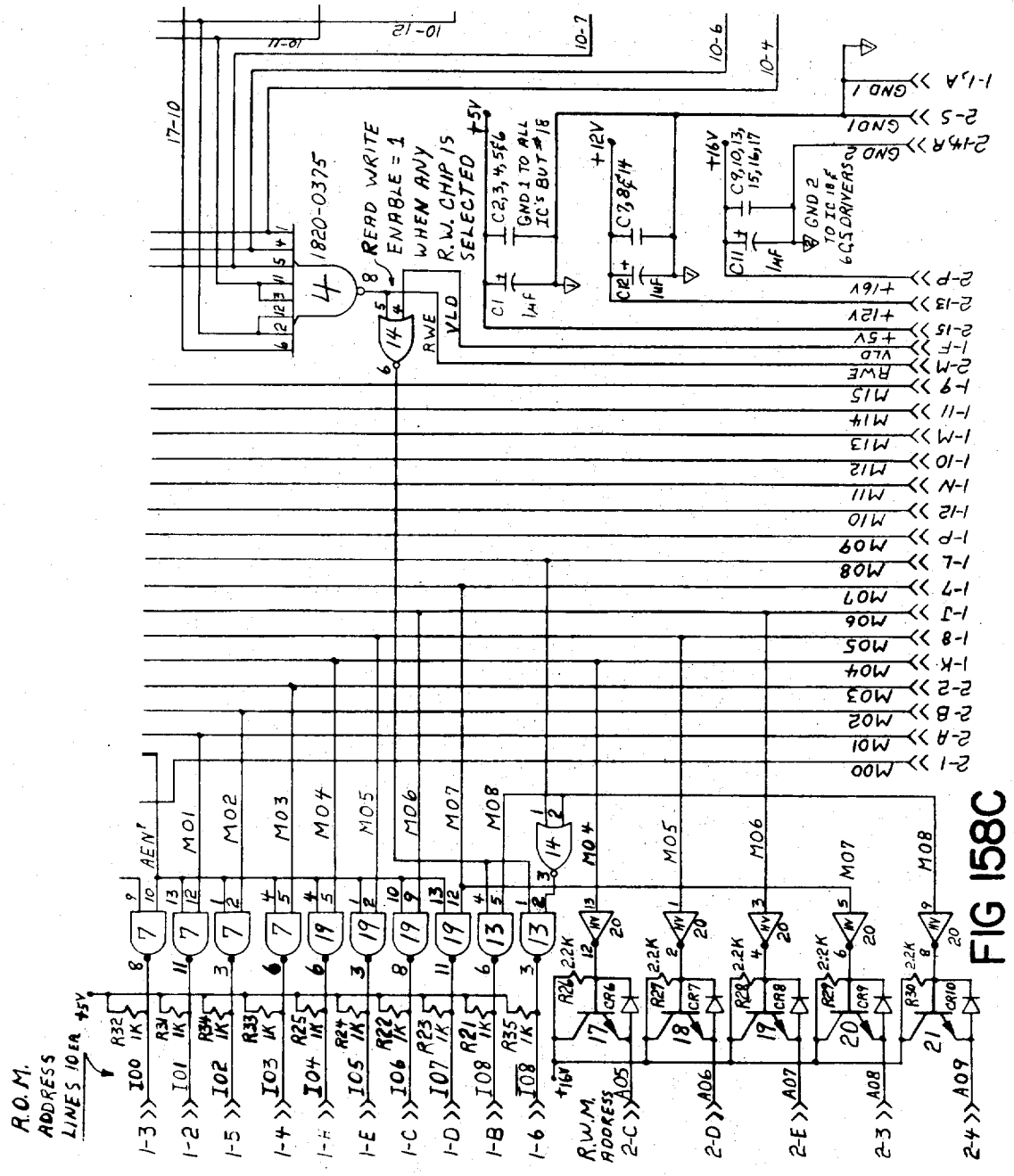

R.O.M. ADDRESS LINES 10 EA
+5V
AEN'
I00
I01
I02
I03
I04
I05
I06
I07
I08
I09
M01
M02
M03
M04
M05
M06
M07
M08
R.W.M. ADDRESS
A05
A06
A07
A08
A09
1820-0375
READ WRITE ENABLE = 1 WHEN ANY R.W. CHIP IS SELECTED
RWE
VLD
+5V
+12V
+16V
GND 1
GND 2
TO IC 18 & CS DRIVERS
M00
M01
M02
M03
M04
M05
M06
M07
M08
M09
M10
M11
M12
M13
M14
M15
RWE
VLD
+5V
+12V
+16V
GND 1
GND 2

FIG 158C

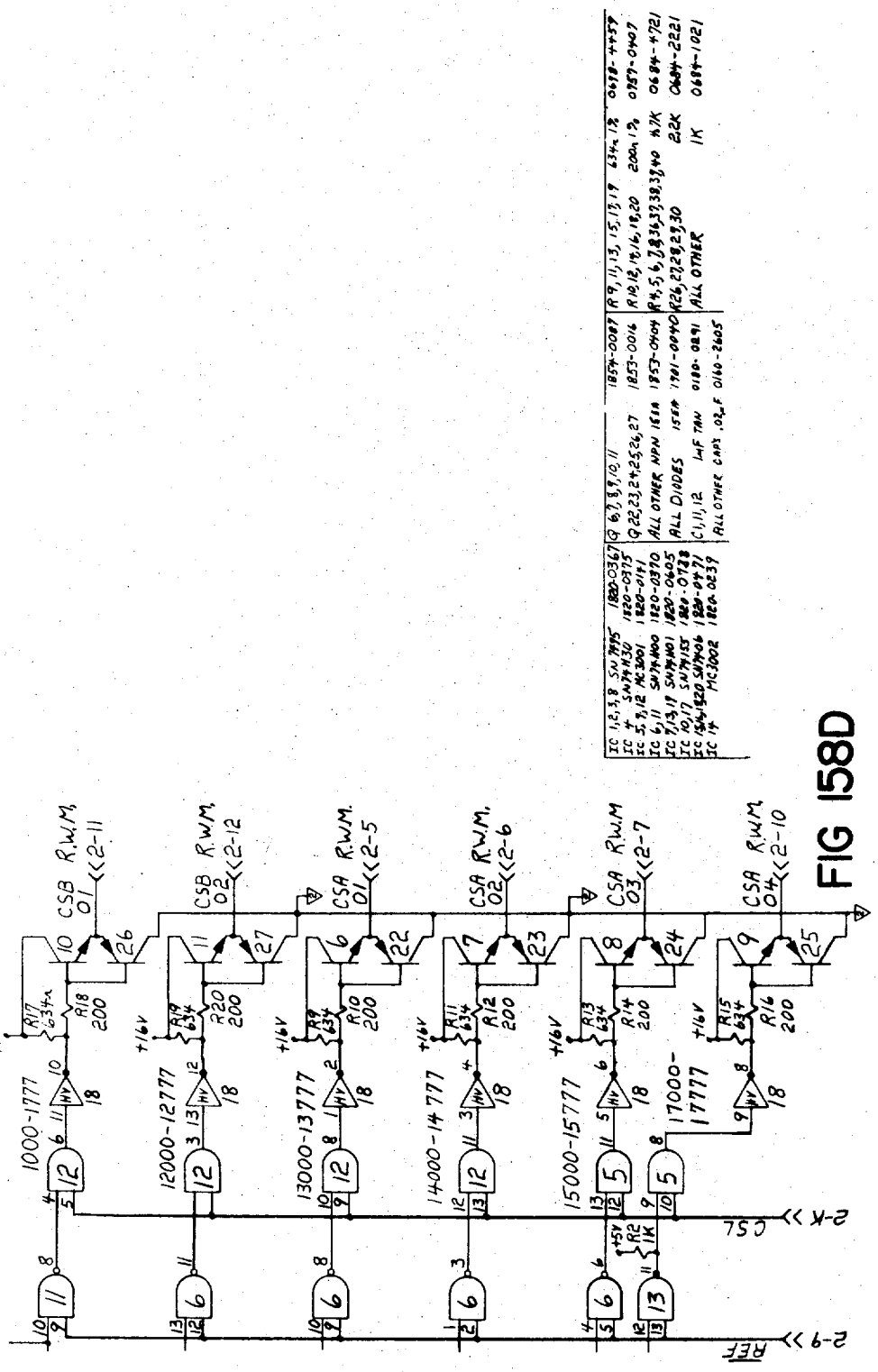

IC 1,2,3,8 SN7495 1820-0367
IC 4 SN74H30 1820-0375
IC 5,9,12 MC3001 1820-0141
IC 6,11 SN74H00 1820-0370
IC 7,13,19 SN74H01 1820-0605
IC 10,17 SN74155 1820-0728
IC 15,16,18,20 SN7406 1820-0471
IC 14 MC3002 1820-0239
Q 6,7,8,9,10,11 1854-0087
Q 22,23,24,25,26,27 1853-0016
ALL OTHER NPN 15EA 1853-0404
ALL DIODES 15EA 1901-0040
C1,11,12 1μF TAN 0180-0291
ALL OTHER CAPS .02μF 0160-2605
R 9,11,13,15,17,19 634Ω 1% 0698-4459
R 10,12,14,16,18,20 200Ω 1% 0757-0407
R 4,5,6,7,8,36,37,38,39,40 4.7K 0684-4721
R 26,27,28,29,30 22K 0684-2221
ALL OTHER 1K 0684-1021
CSB R.W.M. 01 2-11
CSB R.W.M. 02 2-12
CSA R.W.M. 01 2-5
CSA R.W.M. 02 2-6
CSA R.W.M. 03 2-7
CSA R.W.M. 04 2-10
1000-1777
12000-12777
13000-13777
14000-14777
15000-15777
17000-17777
+16V
R17 634Ω
R18 200
R19 634
R20 200
R9 634
R10 200
R11 634
R12 200
R13 634
R14 200
R15 634
R16 200
+5V
R2 1K
CSL
2-K
REF
2-9

FIG 158D

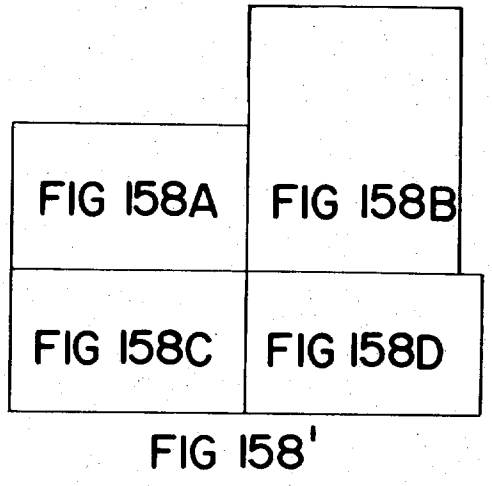


FIG 158'

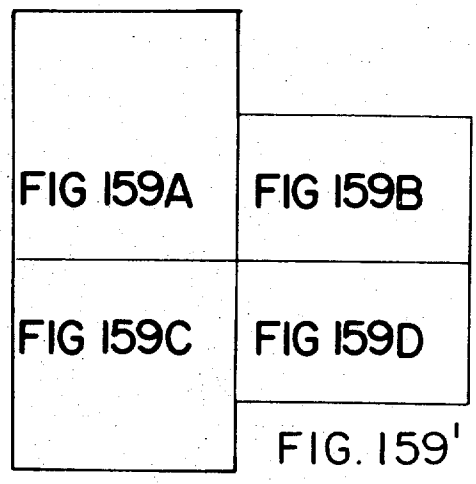


FIG. 159'

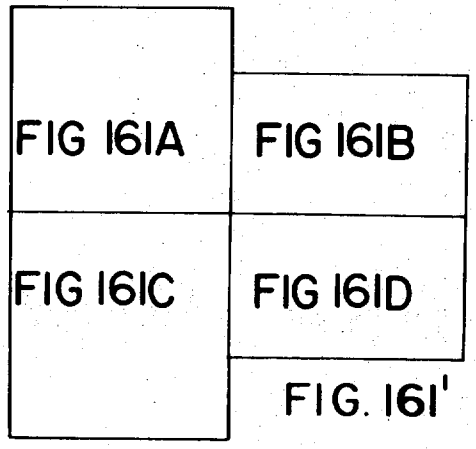


FIG. 161'

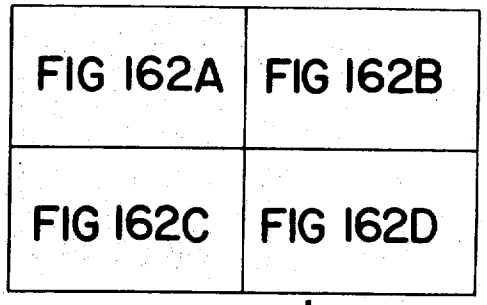


FIG 162'

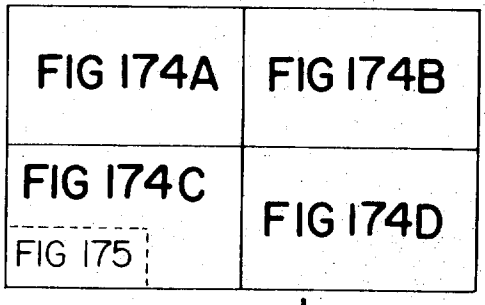


FIG 174'

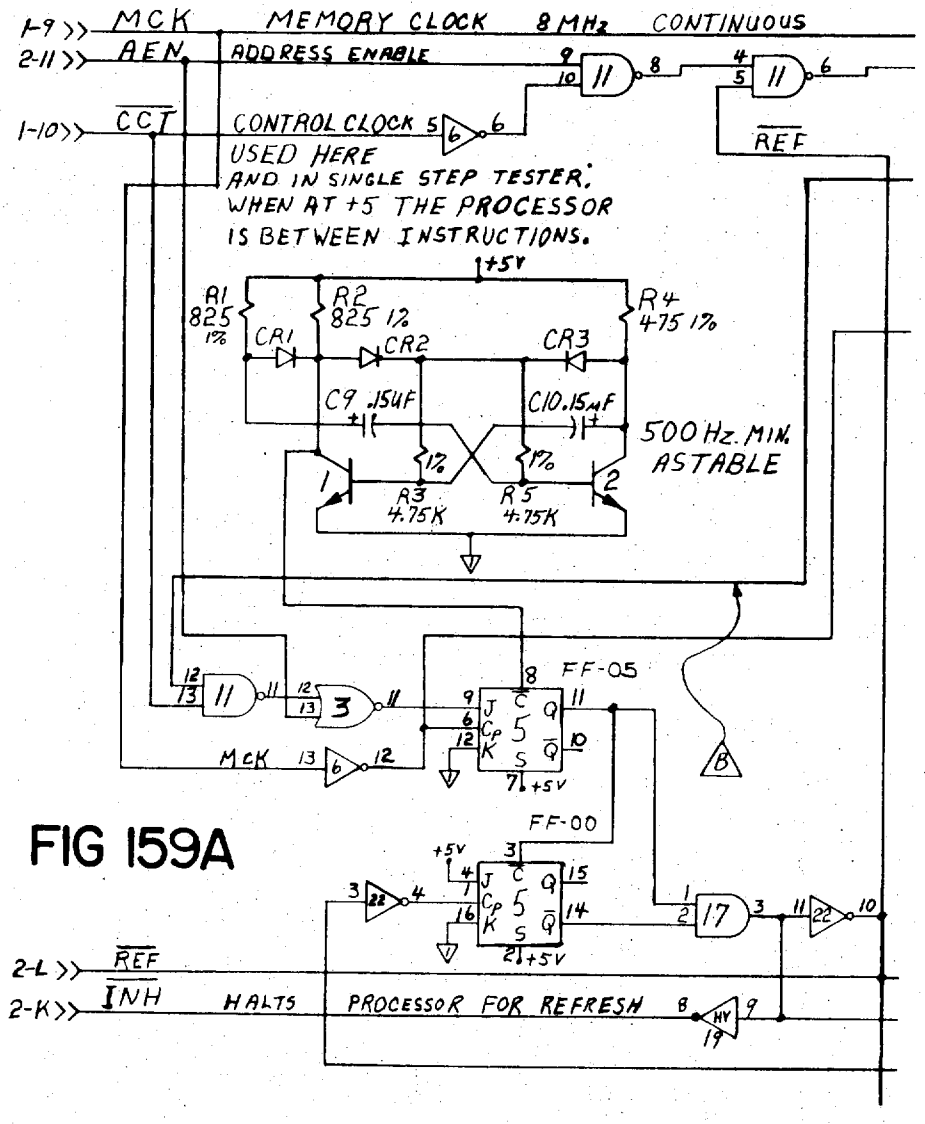

1-9 >> MCK
MEMORY CLOCK 8 MHz CONTINUOUS
2-11 >> AEN
ADDRESS ENABLE
1-10 >> CCT
CONTROL CLOCK
USED HERE
AND IN SINGLE STEP TESTER;
WHEN AT +5 THE PROCESSOR
IS BETWEEN INSTRUCTIONS.
REF
+5V
R1 825 1%
R2 825 1%
R4 4.75 1%
CR1
CR2
CR3
C9 .15μF
C10. 15μF
500 Hz. MIN. ASTABLE
1%
R3 4.75K
R5 4.75K
FF-05
MCK
FF-00
2-L >> REF
2-K >> INH
HALTS PROCESSOR FOR REFRESH

FIG 159A

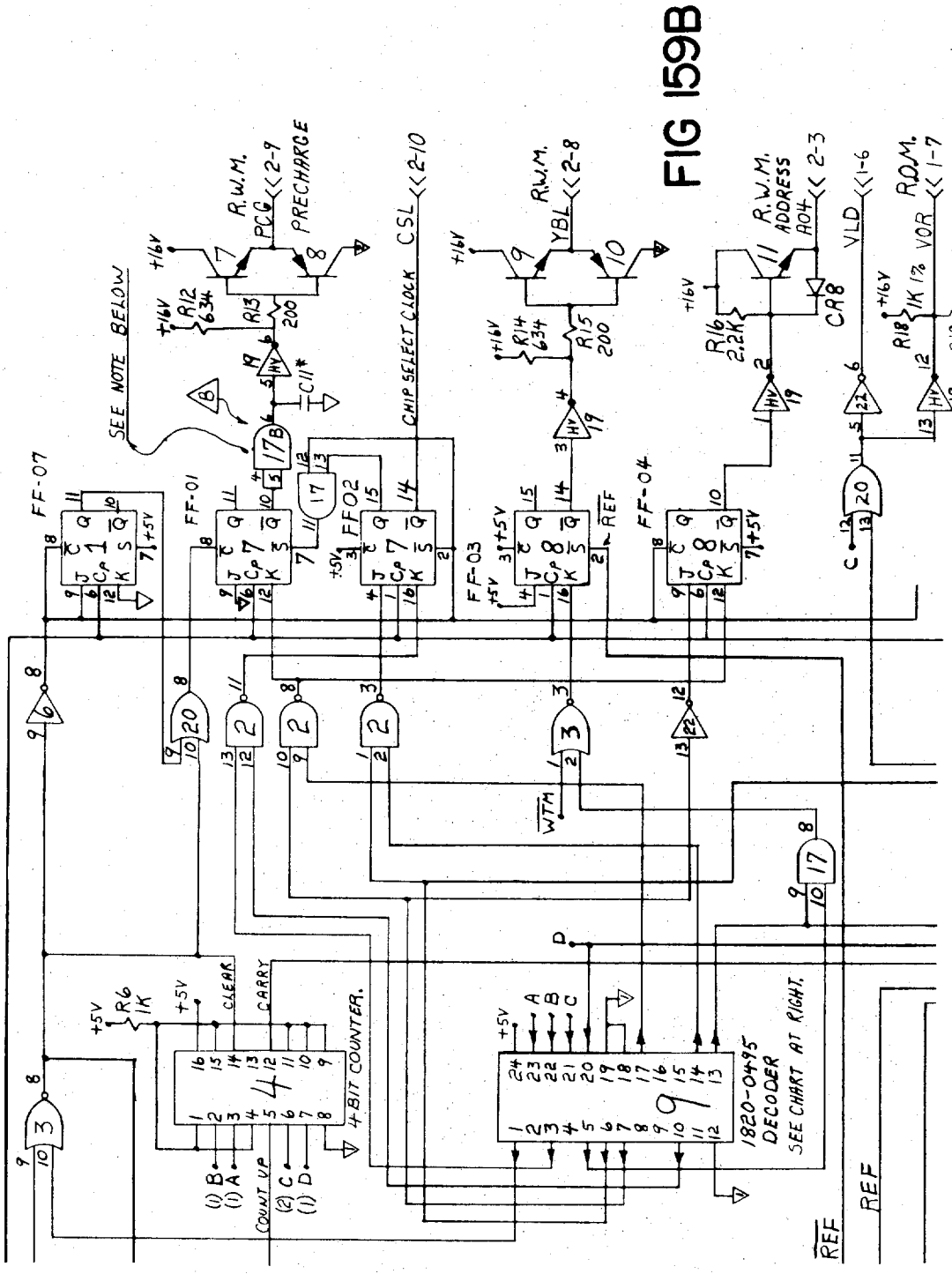
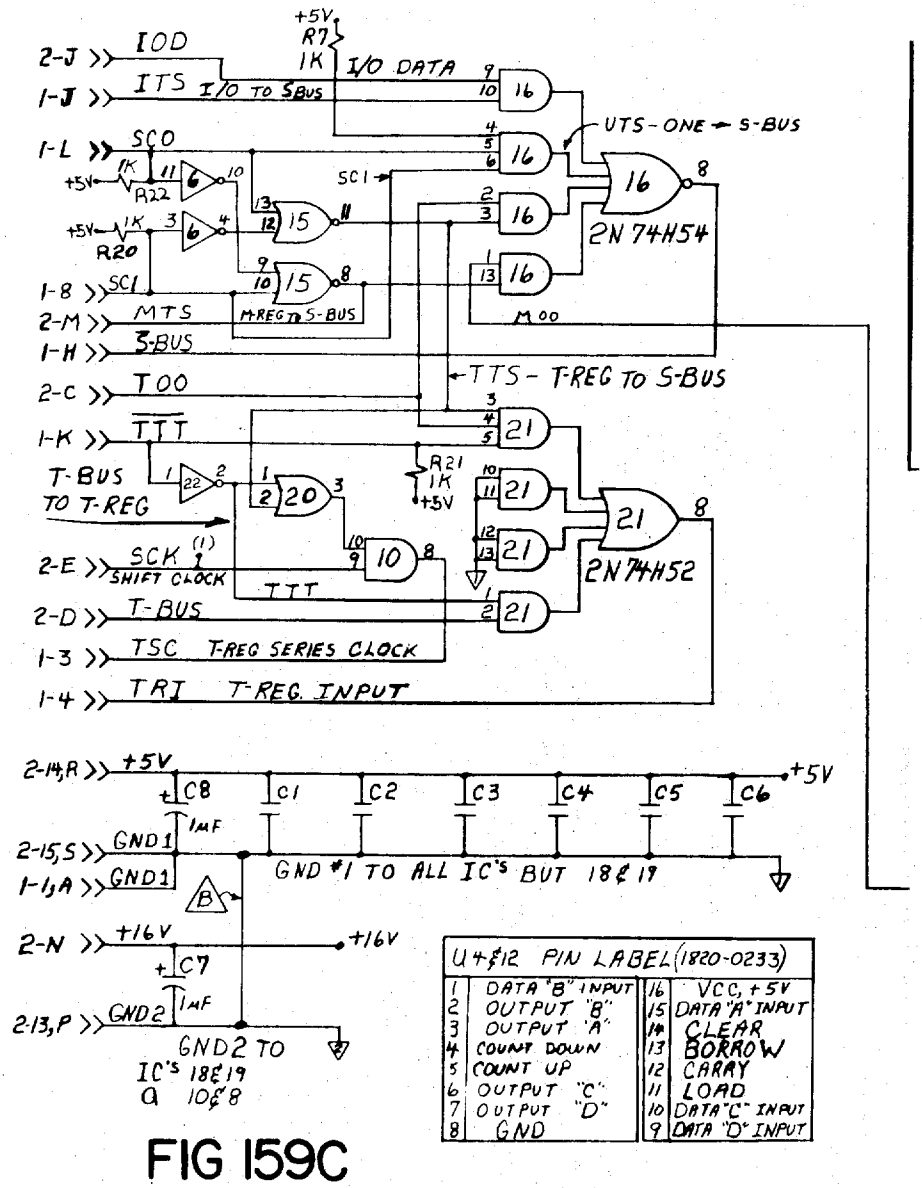


FIG 159B
SEE NOTE BELOW
R.W.M. PCG <<2-9 PRECHARGE
R.W.M. YBL <<2-8
R.W.M. ADDRESS A04 <<2-3
VLD <<1-6
R.D.M. VOR <<1-7
CHIP SELECT CLOCK CSL <<2-10
+16V
R12 634
R13 200
R14 634
R15 200
R16 2.2K
R18 1K 1%
R19 3.09K
CR8
C11*
FF-07
FF-01
FF02
FF-03
FF-04
REF
WTM
R6 1K
+5V
CLEAR
CARRY
COUNT UP
4-BIT COUNTER.
1820-0495 DECODER
SEE CHART AT RIGHT.
A
B
C
D



U4 & 12 PIN LABEL (1820-0233)			
1	DATA "B" INPUT	16	VCC, +5V
2	OUTPUT "B"	15	DATA "A" INPUT
3	OUTPUT "A"	14	CLEAR
4	COUNT DOWN	13	BORROW
5	COUNT UP	12	CARRY
6	OUTPUT "C"	11	LOAD
7	OUTPUT "D"	10	DATA "C" INPUT
8	GND	9	DATA "D" INPUT

FIG 159C

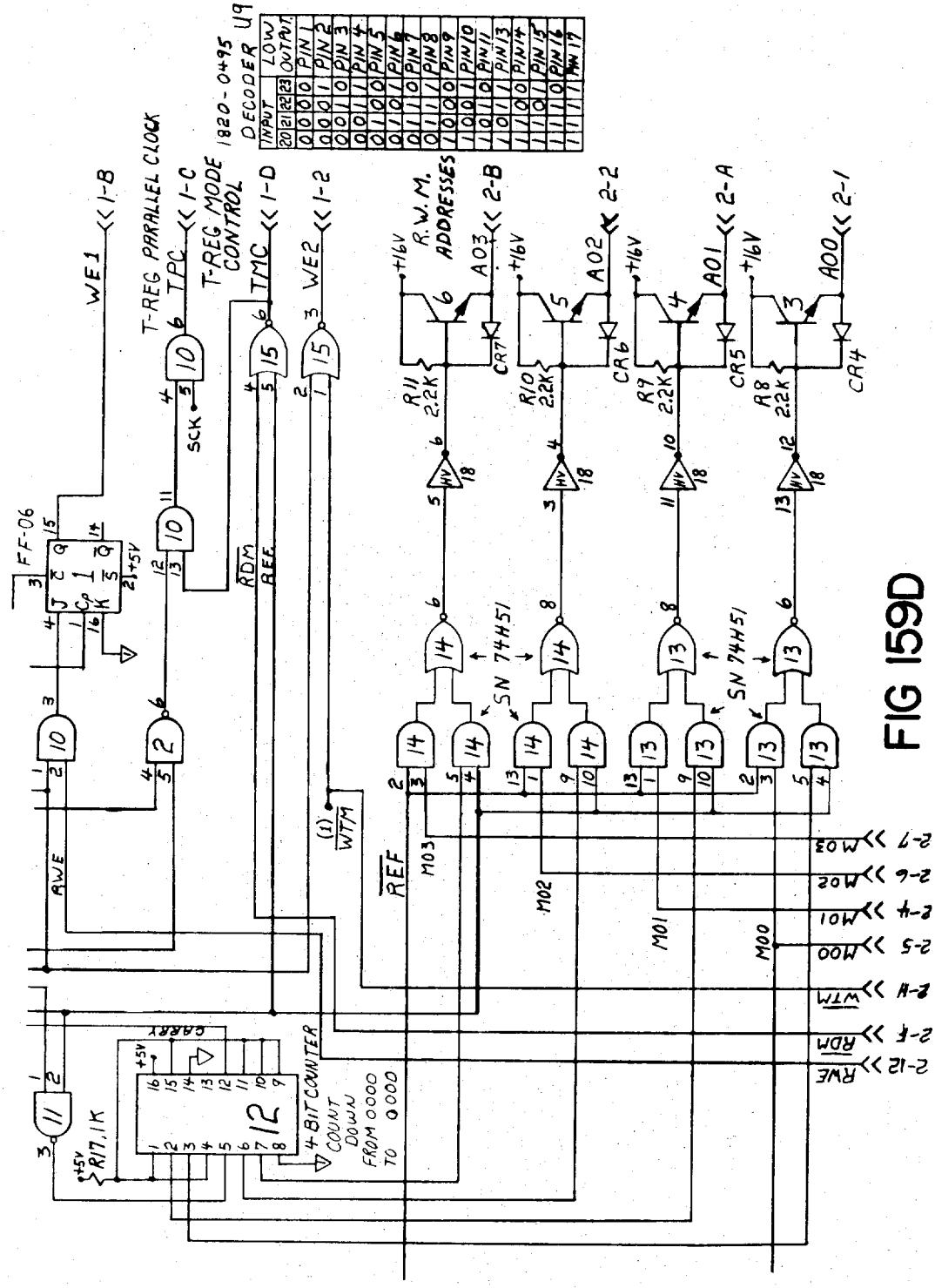

DECODER U9
1820-0495
INPUT
20 21 22 23
LOW OUTPUT
T-REG PARALLEL CLOCK
T-REG MODE CONTROL
R.W.M. ADDRESSES
WE1
WE2
TPC
TMC
SCK
RDM
REF
WTM
AWE
FF-06
SN 74H51
4-BIT COUNTER
COUNT DOWN FROM 0000 TO 0000
CARRY
R17, 1K
R11 2.2K
R10 2.2K
R9 2.2K
R8 2.2K
CR7
CR6
CR5
CR4
A03
A02
A01
A00
M03
M02
M01
M00
+16V
+5V

FIG 159D

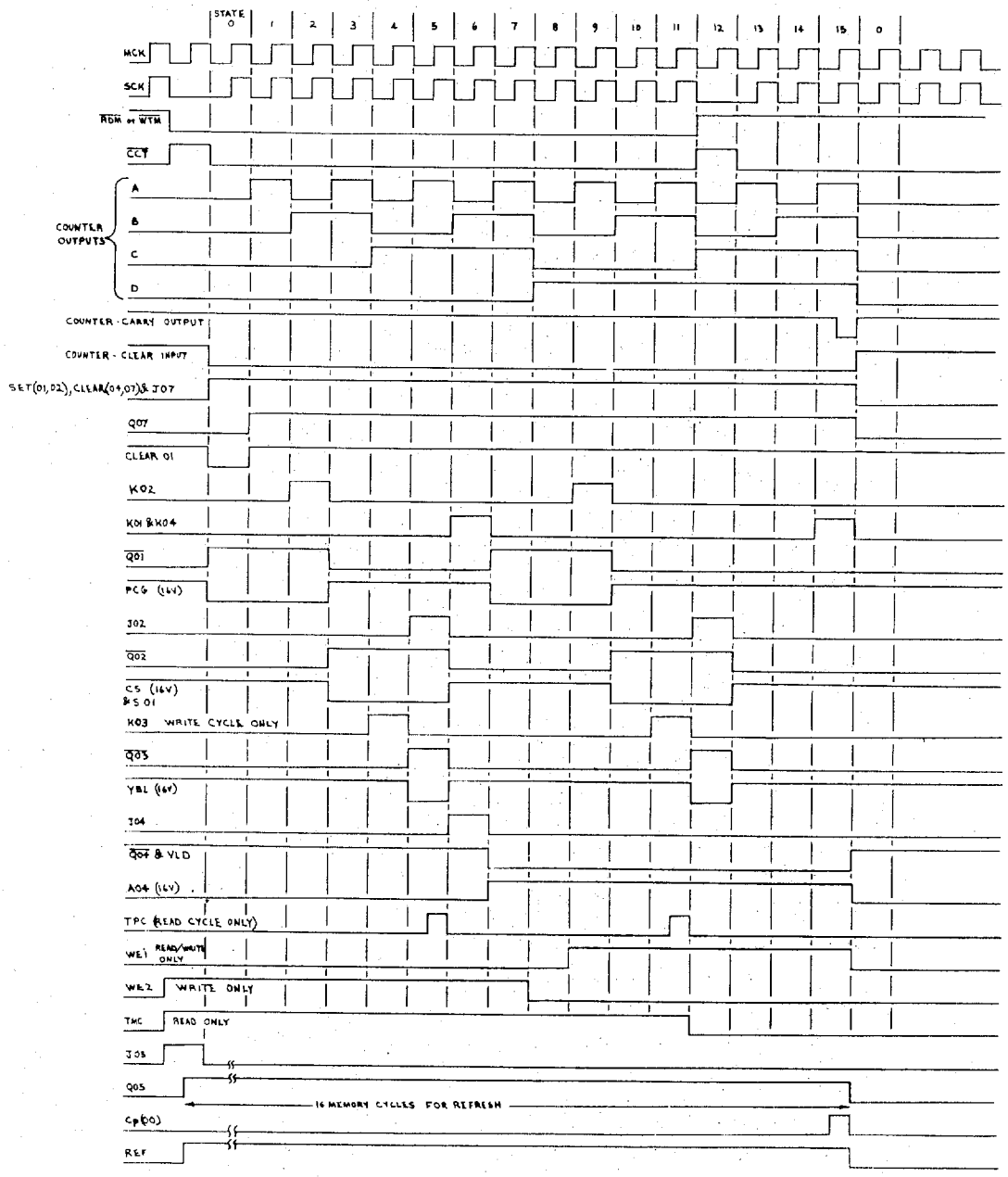

STATE
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
MCK
SCK
ROM or WTM
CCT
COUNTER OUTPUTS
A
B
C
D
COUNTER - CARRY OUTPUT
COUNTER - CLEAR INPUT
SET(01,02), CLEAR(04,07) & J07
Q07
CLEAR 01
K02
K01 & K04
Q01
PCG (16V)
J02
Q02
C5 (16V) & S01
K03 WRITE CYCLE ONLY
Q03
YBL (16V)
J04
Q04 & VLD
A04 (16V)
TPC (READ CYCLE ONLY)
WE1 READ/WRITE ONLY
WE2 WRITE ONLY
TMC READ ONLY
J05
Q05
16 MEMORY CYCLES FOR REFRESH
Cp(00)
REF

FIG 160

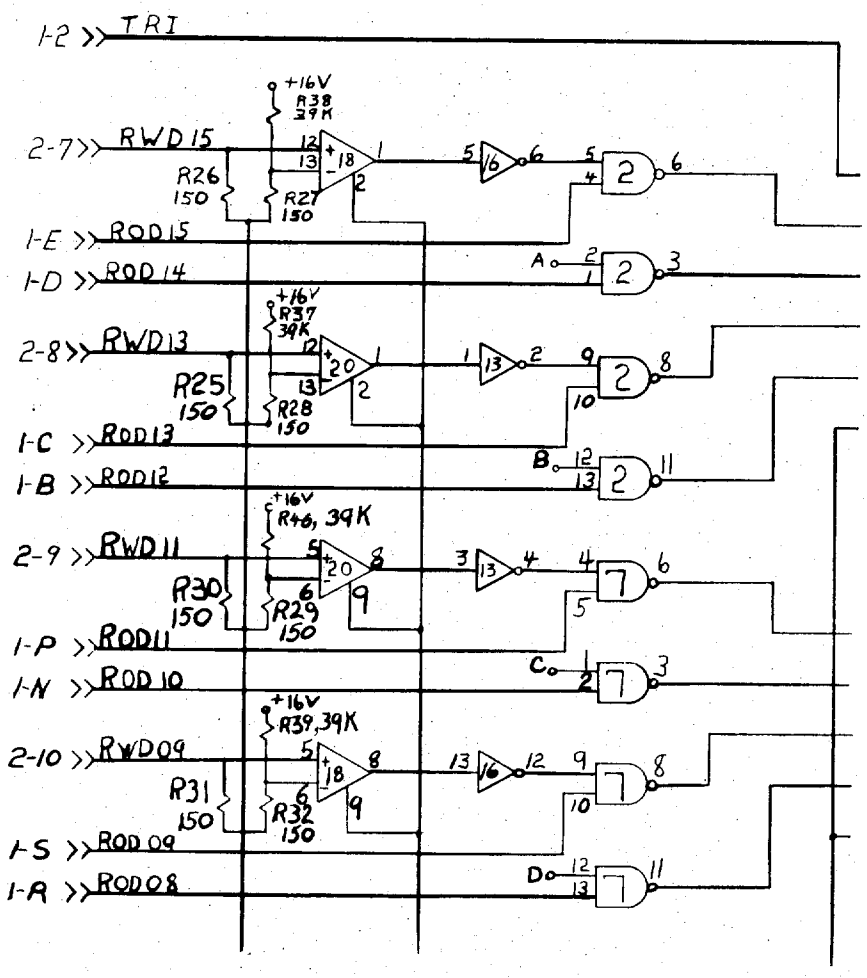

1-2 >> TRI
+16V
R38 39K
2-7 >> RWD 15
R26 150
R27 150
1-E >> ROD 15
1-D >> ROD 14
+16V
R37 39K
2-8 >> RWD 13
R25 150
R28 150
1-C >> ROD 13
1-B >> ROD 12
+16V
R46, 39K
2-9 >> RWD 11
R30 150
R29 150
1-P >> ROD 11
1-N >> ROD 10
+16V
R39, 39K
2-10 >> RWD 09
R31 150
R32 150
1-S >> ROD 09
1-R >> ROD 08
A
B
C
D

FIG 161A

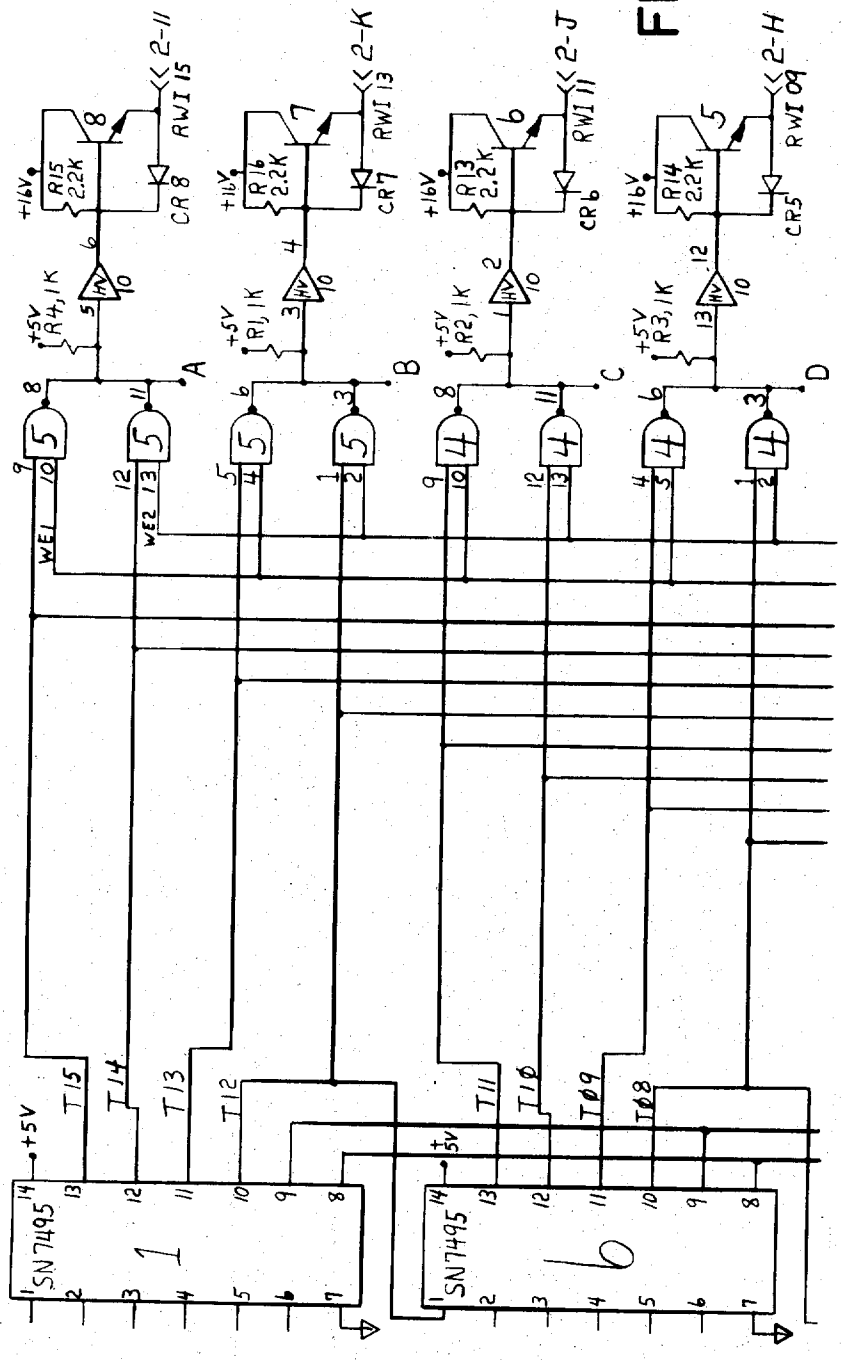

+16V
R15 2.2K
R16 2.2K
R13 2.2K
R14 2.2K
CR8
CR7
CR6
CR5
RWI 15
RWI 13
RWI 11
RWI 09
2-11
2-K
2-J
2-H
+5V
R4, 1K
R1, 1K
R2, 1K
R3, 1K
A
B
C
D
WE1
WE2
T15
T14
T13
T12
T11
T10
T09
T08
SN7495
1
0

FIG 161B

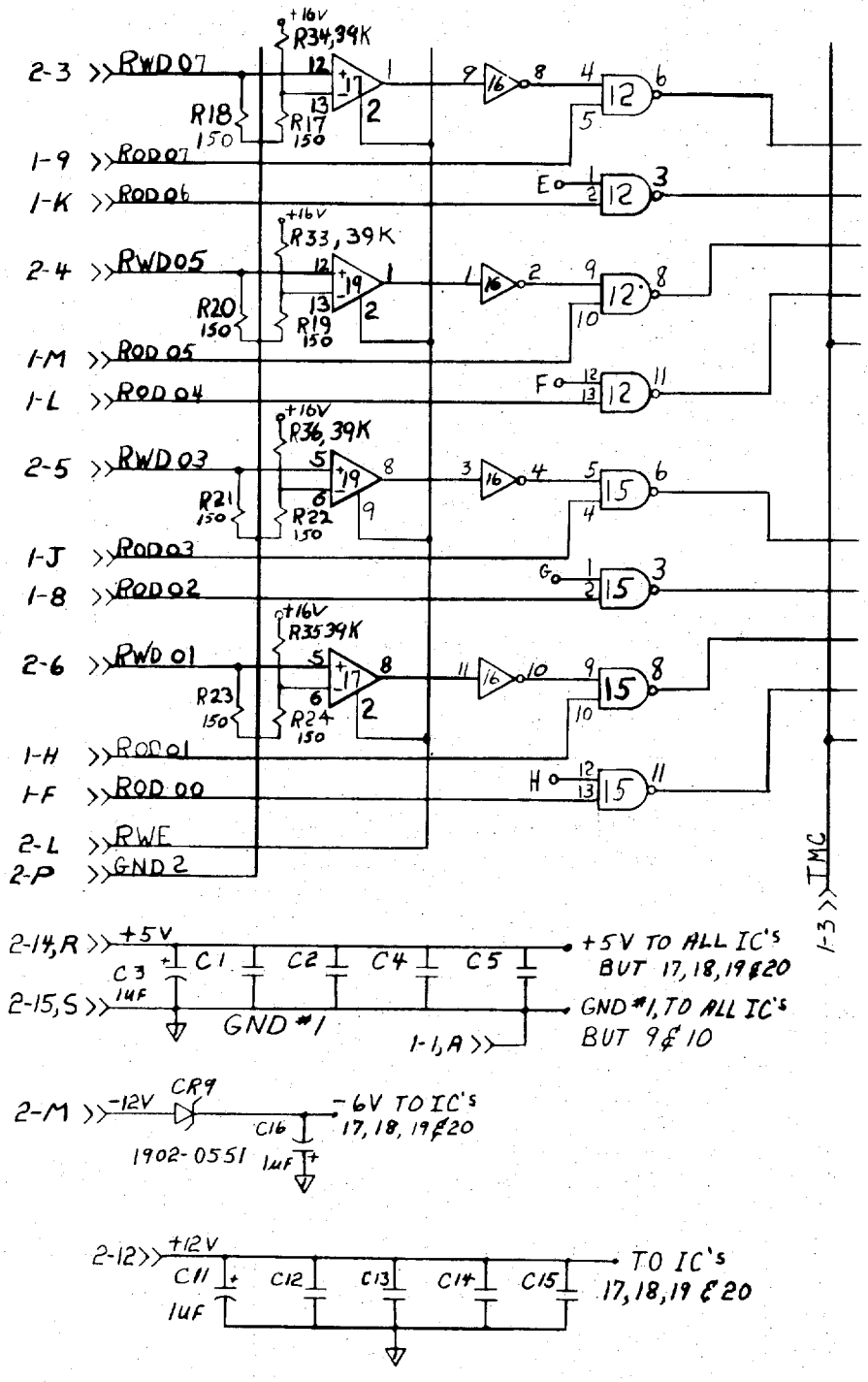

+16V
R34, 39K
2-3 >> RWD 07
R18 150
R17 150
1-9 >> ROD 07
1-K >> ROD 06
R33, 39K
2-4 >> RWD 05
R20 150
R19 150
1-M >> ROD 05
1-L >> ROD 04
R36, 39K
2-5 >> RWD 03
R21 150
R22 150
1-J >> ROD 03
1-8 >> ROD 02
R35 39K
2-6 >> RWD 01
R23 150
R24 150
1-H >> ROD 01
1-F >> ROD 00
2-L >> RWE
2-P >> GND 2
E
F
G
H
1-3 >> IMC
2-14,R >> +5V
C3 14F
C1
C2
C4
C5
+5V TO ALL IC's BUT 17, 18, 19 & 20
2-15,S >> GND #1
GND #1, TO ALL IC's BUT 9 & 10
1-1,A >>
2-M >> -12V
CR9 1902-0551
C16 1uF
-6V TO IC's 17, 18, 19 & 20
2-12 >> +12V
C11 1uF
C12
C13
C14
C15
TO IC's 17, 18, 19 & 20

FIG 161C

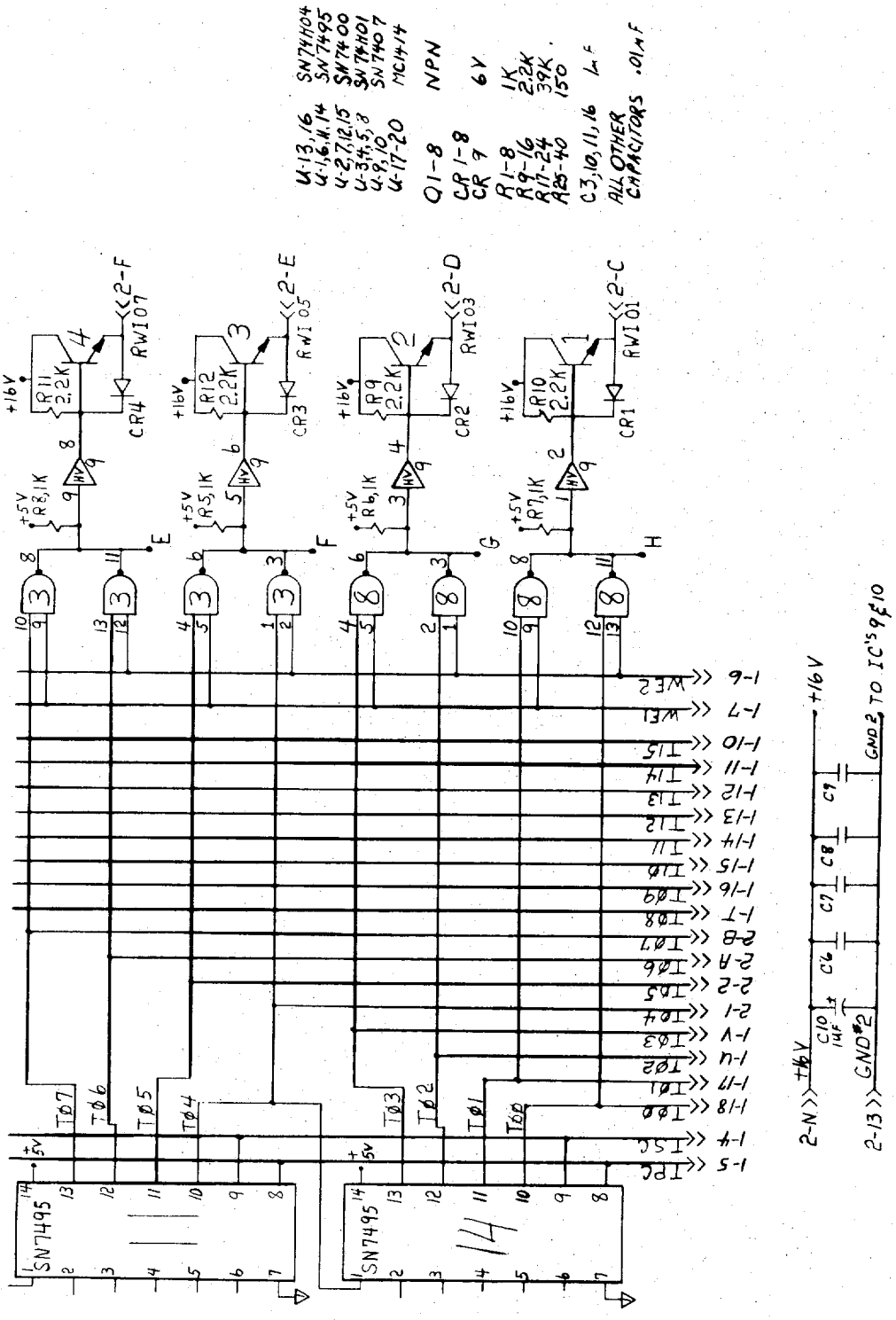

U-13,16 SN74H04
U-1,6,11,14 SN7495
U-2,7,12,15 SN7400
U-3,4,5,8 SN74H01
U-9,10 SN7407
U-17-20 MC1414
Q1-8 NPN
CR1-8
CR9 6V
R1-8 1K
R9-16 2.2K
R17-24 39K
R25-40 150
C3,10,11,16 1μF
ALL OTHER CAPACITORS .01μF
SN7495
SN7495
+16V
+5V
GND#2
TO IC'S 9 & 10

FIG 161D

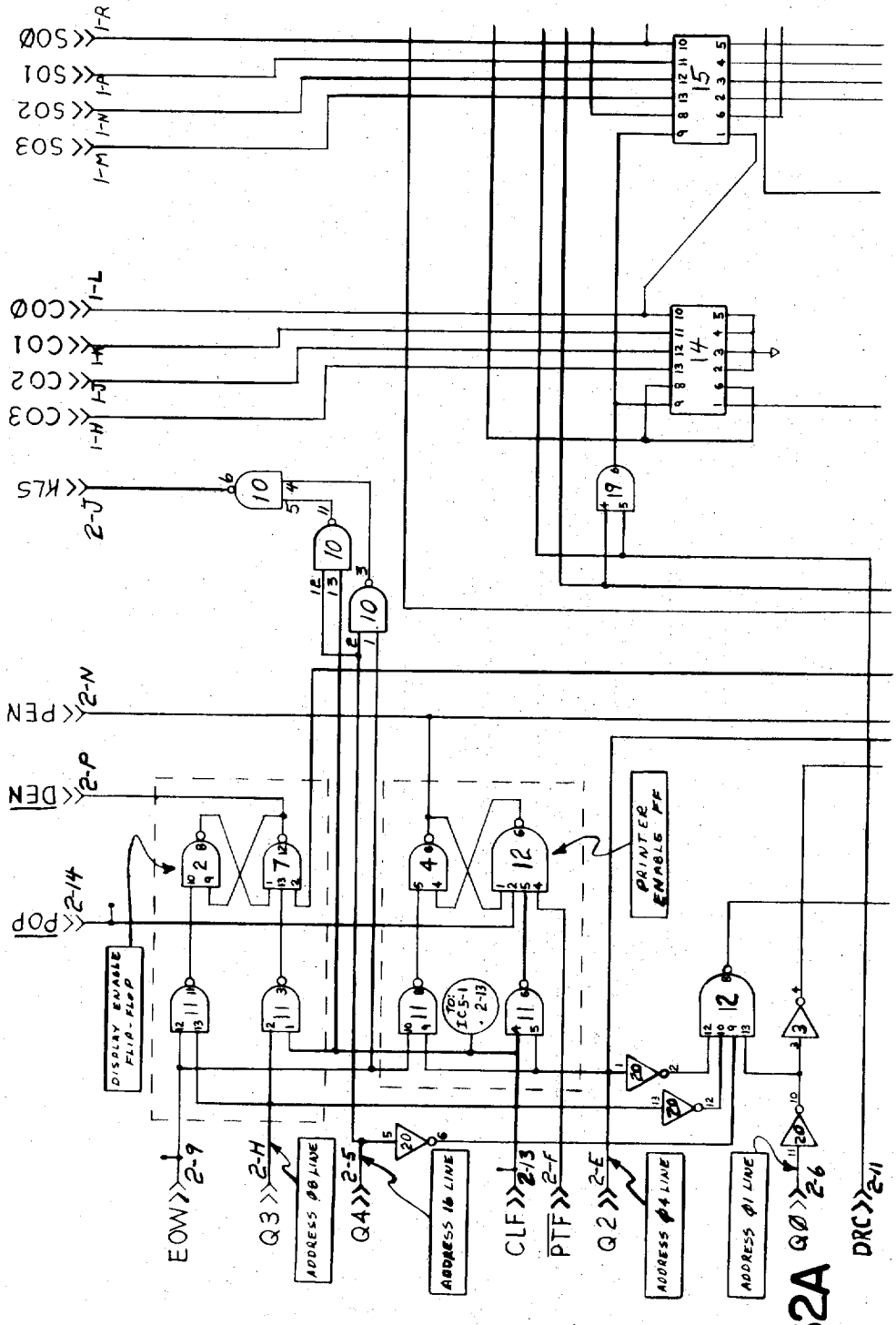

ØOS
SO1
SO2
SO3
ØOC
CO1
CO2
CO3
KLS
PEN
DEN
POP
DISPLAY ENABLE FLIP-FLOP
PRINTER ENABLES FF
EOW
Q3
Q4
CLF
PTF
Q2
Q0
DRC
ADDRESS Ø8 LINE
ADDRESS 16 LINE
ADDRESS Ø4 LINE
ADDRESS Ø1 LINE

FIG 162A

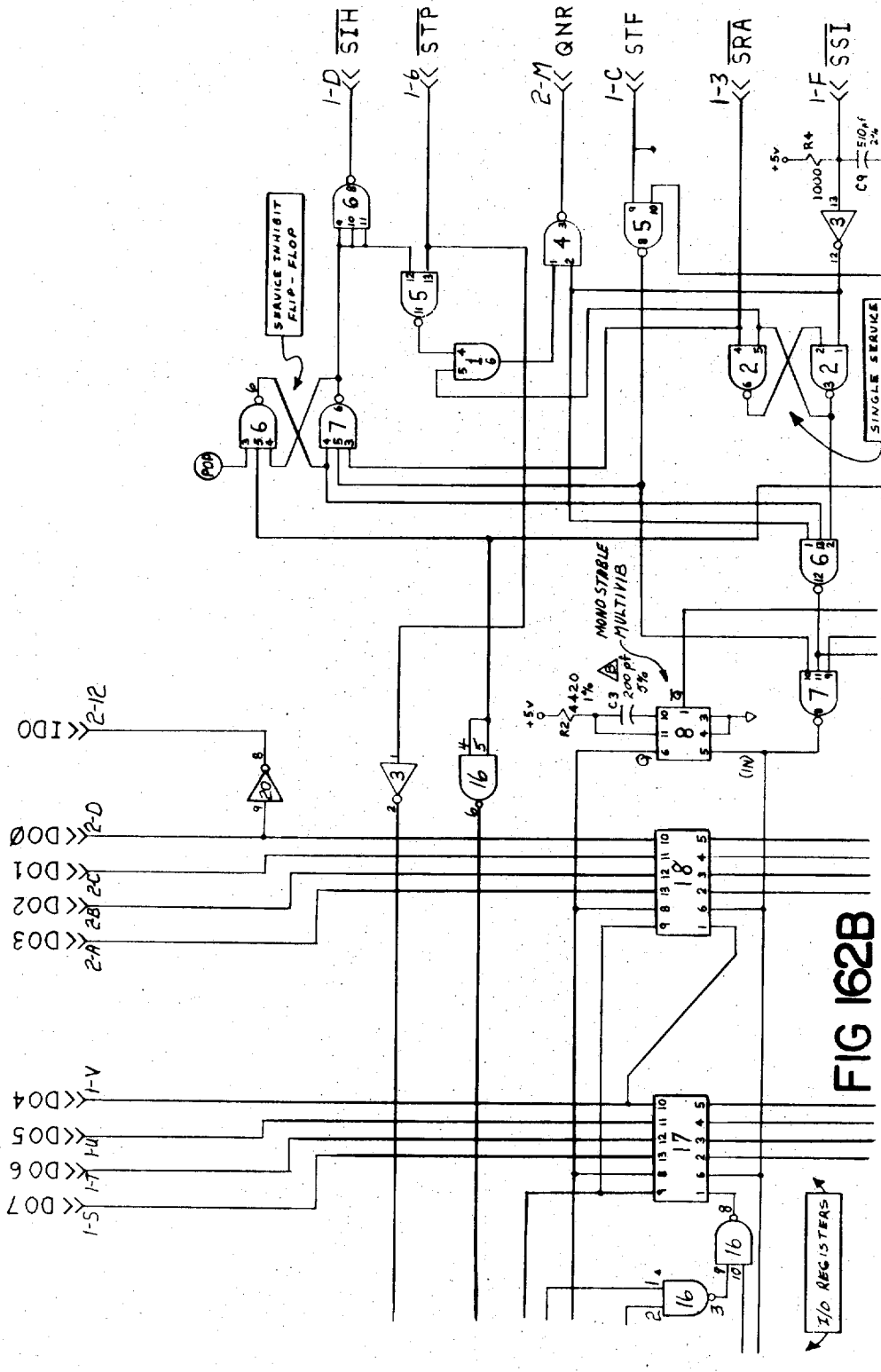

SIH
STP
QNR
STF
SRA
SSI
1-D
1-6
2-M
1-C
1-3
1-F
SERVICE INHIBIT FLIP-FLOP
SINGLE SERVICE FLIP-FLOP
MONO STABLE MULTIVIB
POA
IDO
DOØ
DO1
DO2
DO3
DO4
DO5
DO6
DO7
2-12
2-D
2-C
2-B
2-A
1-V
1-U
1-T
1-S
I/O REGISTERS

FIG 162B

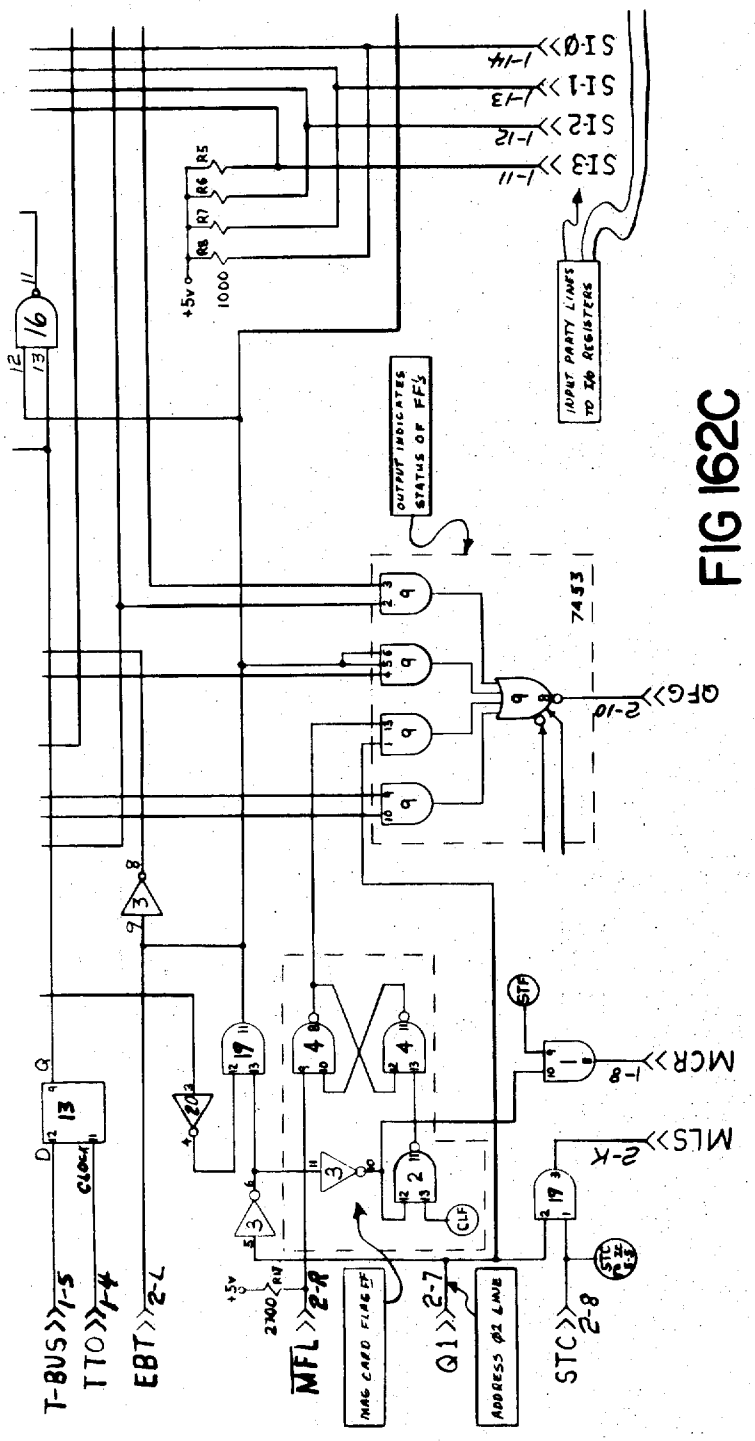


FIG 162C

COMPONENT IC NO.	SERIES NO.	DESCRIPTION
1,19	MC-3001P	QUAD 2-INPUT 'AND' GATES
2,4,5,10,11,16	SN-7400	QUAD 2-INPUT 'NAND' GATES
3,20	-7404	6-PK HEX INVERTERS
6,17	-7410	TRIPLE 3-INPUT 'NAND' GATES
8	-74121	MONOSTABLE MULTIVIBRATOR
9	-7453	EXPANDABLE 4-WIDE 2-INPUT 'AND-OR-INVERTED GATES
12	-7420	DUAL 4-INPUT 'NAND' GATES
13	-7474	DUAL EDGE-TRIGGERD FLIP-FLOPS
14,15,17,18	-7495	4-BIT RIGHT SHIFT, LEFT SHIFT REGISTER

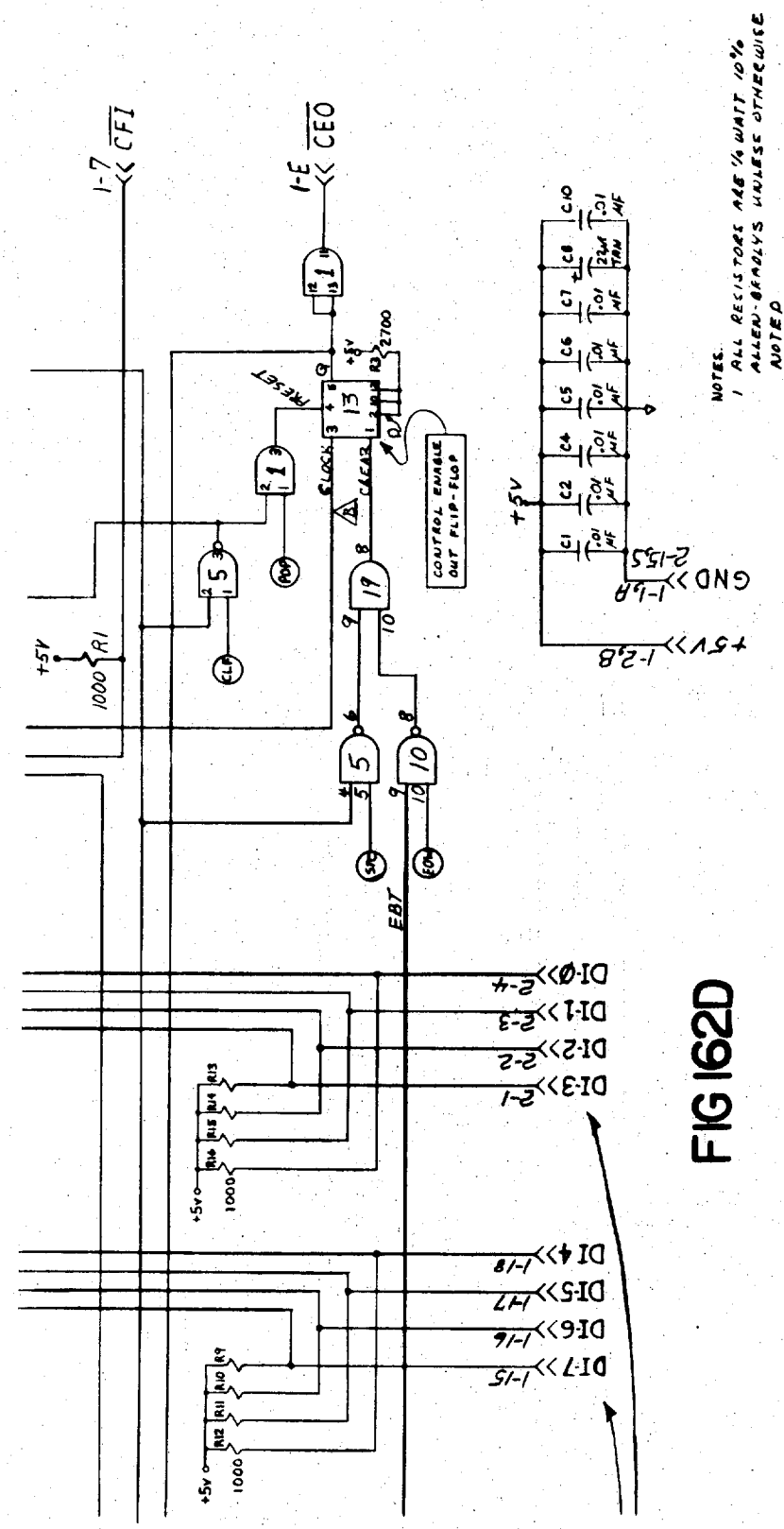

1-7 CFI
1-E CEO
CONTROL ENABLE OUT FLIP-FLOP
CLOCK
CLEAR
PRESET
+5V
R1
1000
R3
2700
EBT
DI-Ø
DI-1
DI-2
DI-3
DI-4
DI-5
DI-6
DI-7
GND
+5V
NOTES.
1 ALL RESISTORS ARE 1/4 WATT 10%
ALLEN-BRADLYS UNLESS OTHERWISE
NOTED

FIG 162D

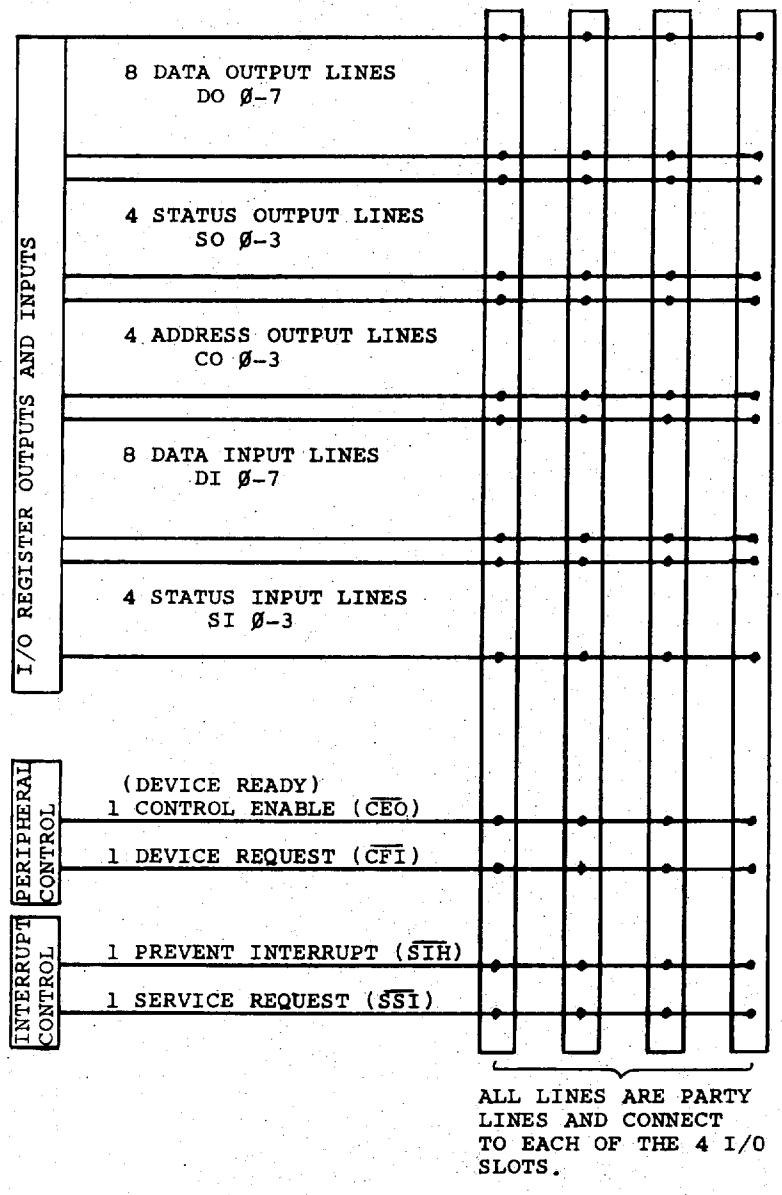
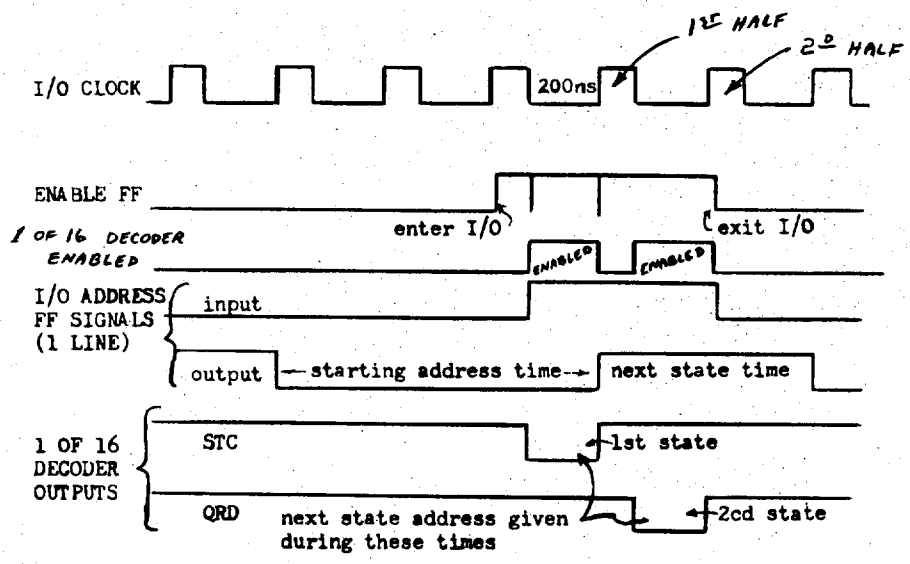


FIG 163 Relationship of the I/O Lines.



I/O Control Board Timing Diagrams SHOWING EXECUTION OF STC

FIG 164

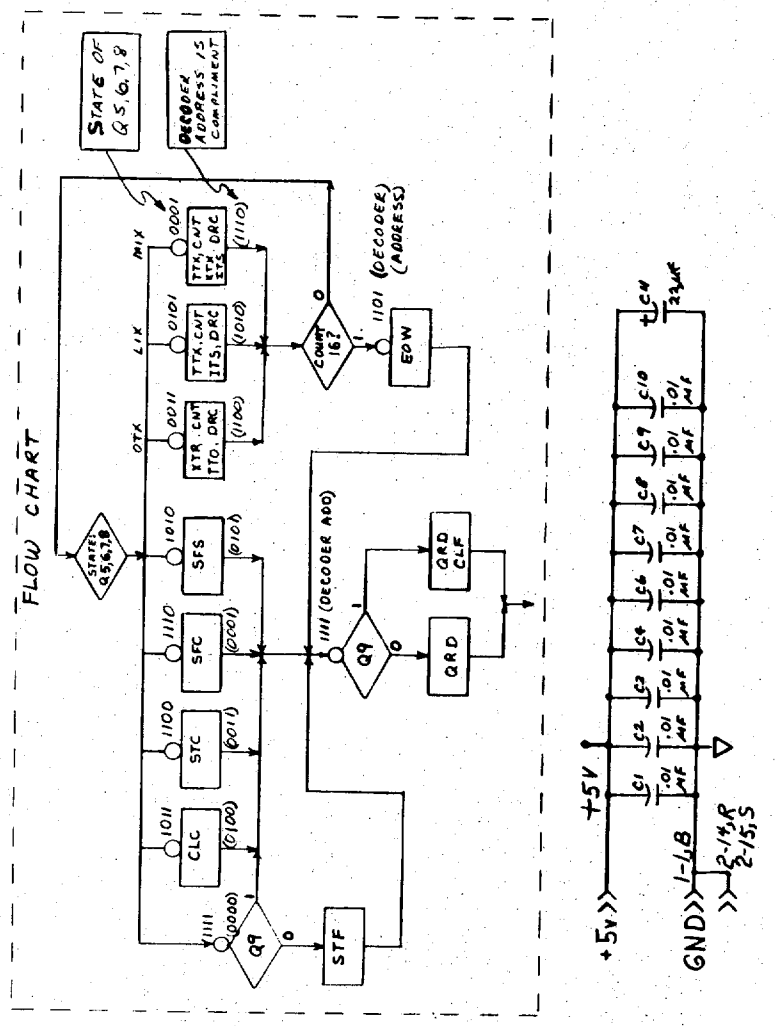

FLOW CHART
STATE OF Q5,6,7,8
DECODER ADDRESS IS COMPLIMENT
STATE: Q5,6,7,8
MIX
LIX
OTX
0001
0101
0011
1010
1110
1100
1011
1111
TTX, CNT XTX ITS DRC
TTX, CNT ITS, DRC
XTR, CNT TTO, DRC
SFS
SFC
STC
CLC
Q9
STF
COUNT 16?
1101 (DECODER) (ADDRESS)
EOW
1111 (DECODER ADD)
QRD CLF
QRD
+5V
GND
C1
C2
C3
C4
C6
C7
C8
C9
C10
C11
.01 μF
22μF
1-1,B
2-14,R
2-15,S

FIG 165

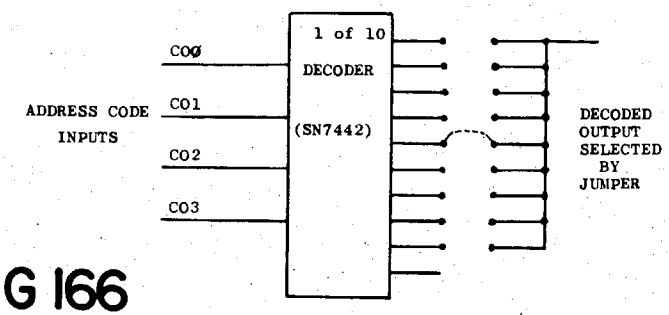


FIG 166

FIG 167

I/O Line Specification Chart

	Name of Line	Line Definition	Direction	Load/Loading	Voltage High	Voltage Low	# of Lines
1	Address Data (CO Ø-3)	Transmits a 4 bit address from the I/O Register to be recognized by an interface card. (Data = High)	Out	1 TTL (1.6ma) allowed per interface.	≥ 2.4v	≤ .4v	4
2	Device Ready $\overline{CEO}$	Indicates calculator is ready for information interchange with an addressed peripheral. (Active State = Low)	Out	1 TTL (1.6ma) allowed per interface.	≥ 2.4v	≤ .4v	1
3	Device Request ($\overline{CFI}$)	Acknowledges receipt of data by a peripheral from the calculator or indicates data is to be input to the calculator. (Active State = Low)	In	Loading of 6.6 ma to the interface card.	1k resistor to + 5v, use open collector	Must be driven below .4v.	1
4	Halt Status ($\overline{STP}$)	Indicates stop key has been depressed. (Active State = Low)	Out	1 TTL (1.6ma) Allowed/interface.	≥ 2.4v	≤ .4	1
5	Input Data (DI Ø-7, SI Ø-3)	Receives input data to I/O register. (DATA = LOW)	In	Loading of 6.1ma	1k Res. To + 5v	Driven ≤ .4v	12
6	Output Data (DO Ø-7, SO Ø-3)	Transmits Data from the I O register. (Data = High)	Out	1 TTL (1.6 ma) Allowed/interface.	≥ 2.4v	≤ .4v	12
7	Prevent Interrupt ($\overline{3IN}$)	Indicates data cannot be entered under service request. (Interrupt) (Active State = Low)	Out	1 TTL (1.6 ma) Allowed/interface	≥ 2.4v	≤ .4	1
8	Service Request (Lo) ($\overline{SSI}$)	Indicates a CPU interrupt is to take place to allow data to enter. (Active State = Low)	In	Loading of 6.6ma	1k Res. to + 5v	Driven ≤ .4	1

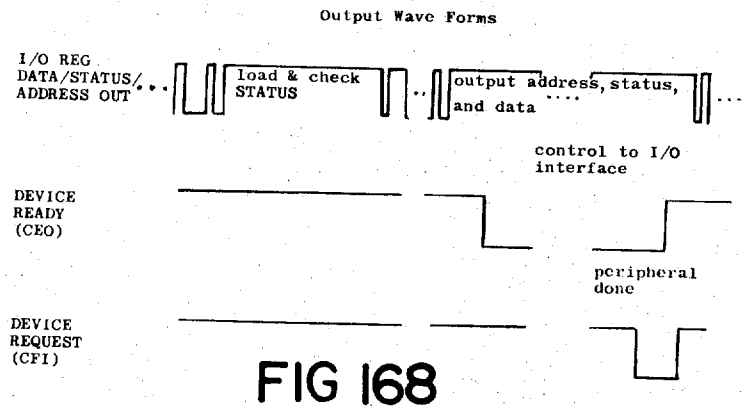


FIG 168

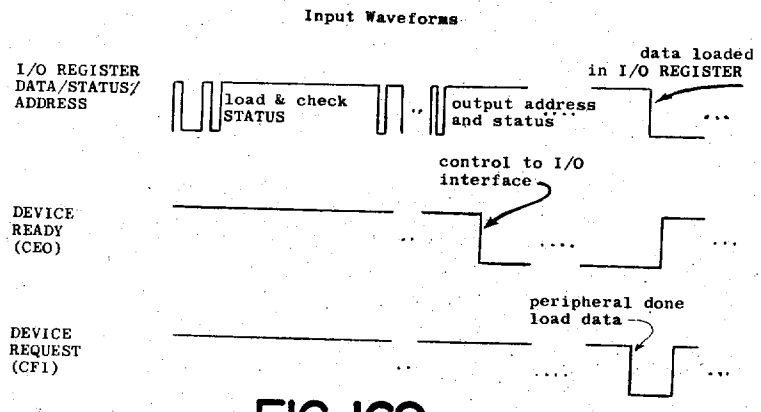


FIG 169

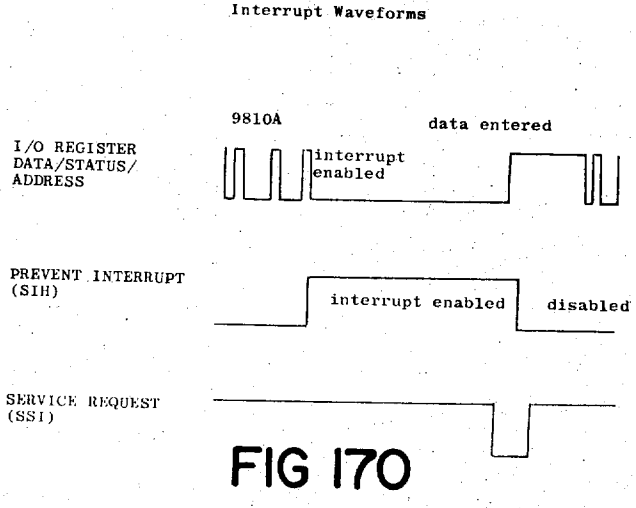
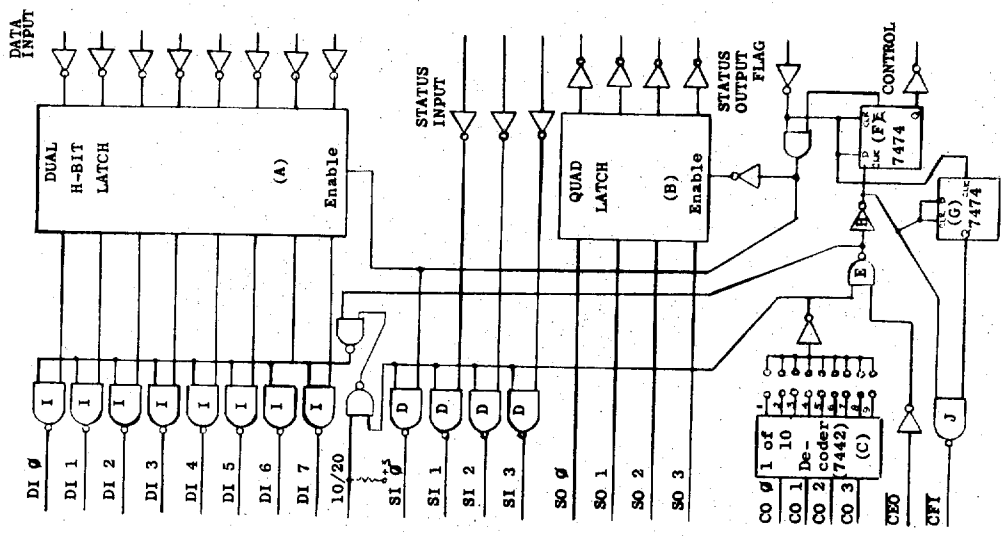
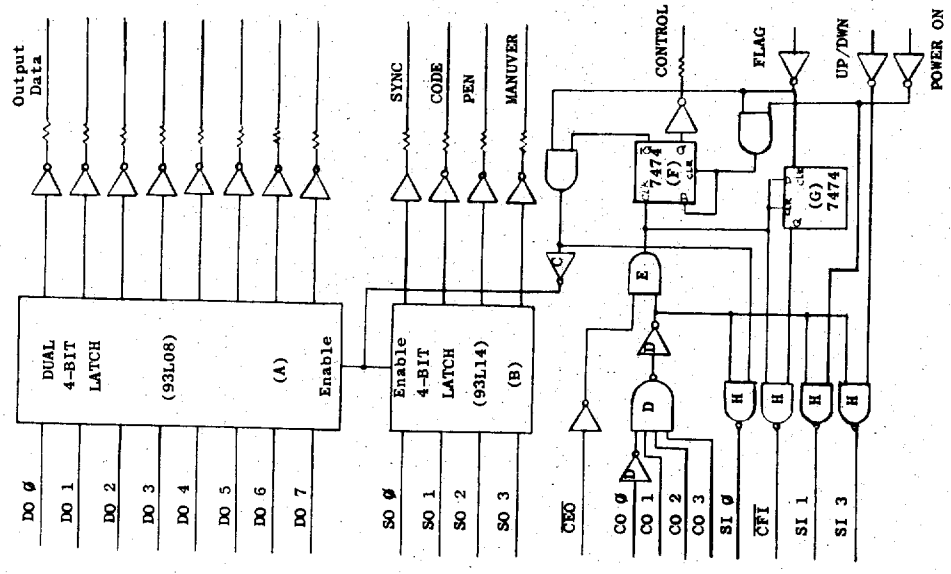


FIG 170



General Purpose Input Card.

FIG 172



Interface Logic to Plotter

FIG 171

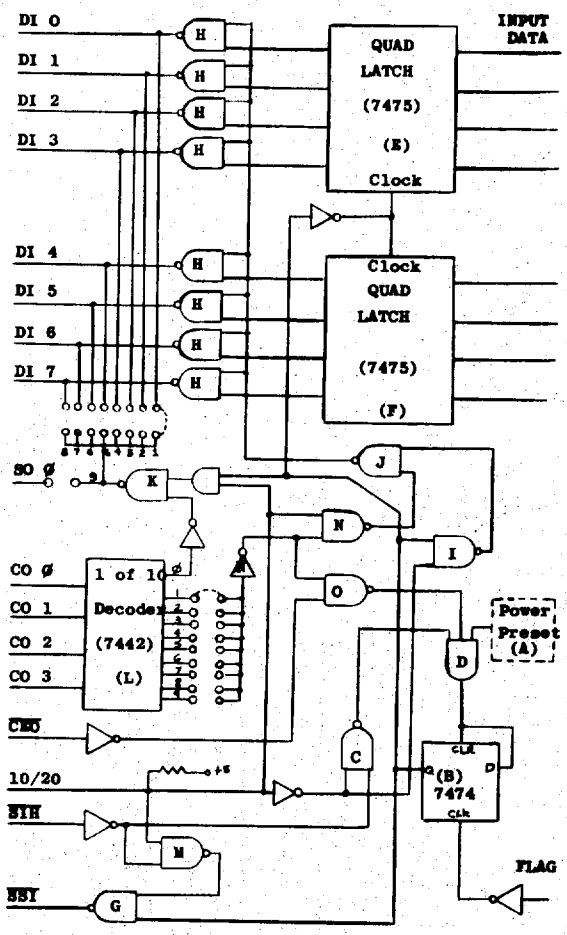


FIG 173 General I/O Board Interrupt Logic.

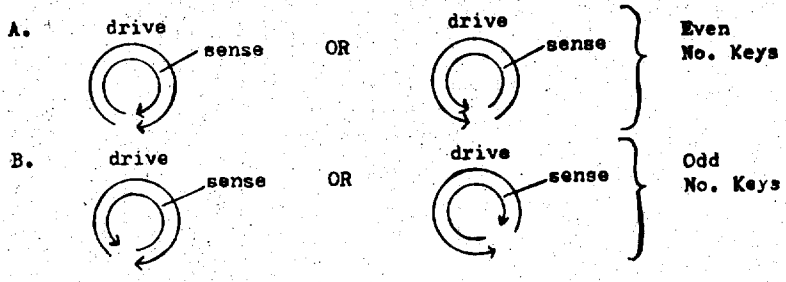


FIG 179

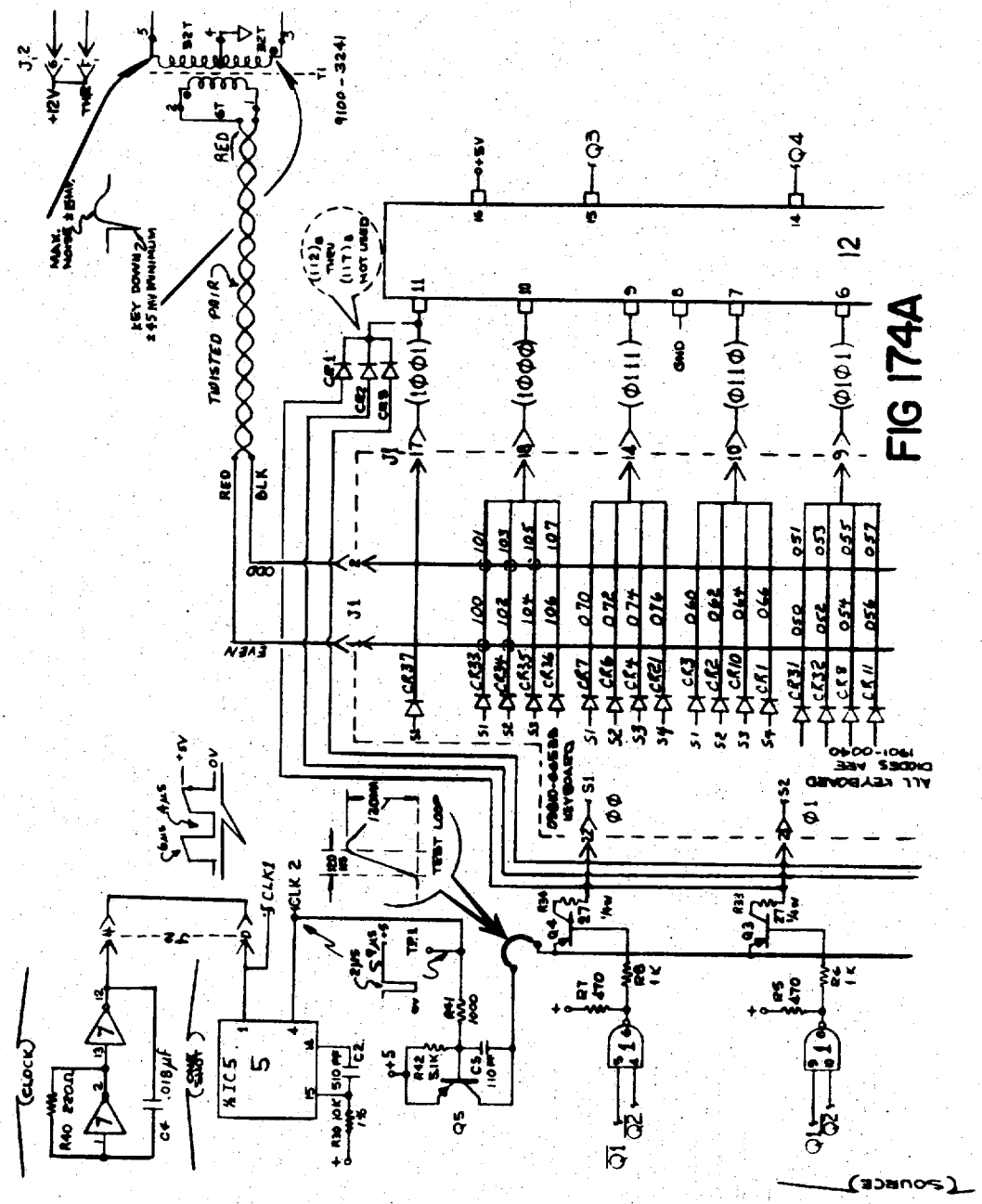


FIG 174A

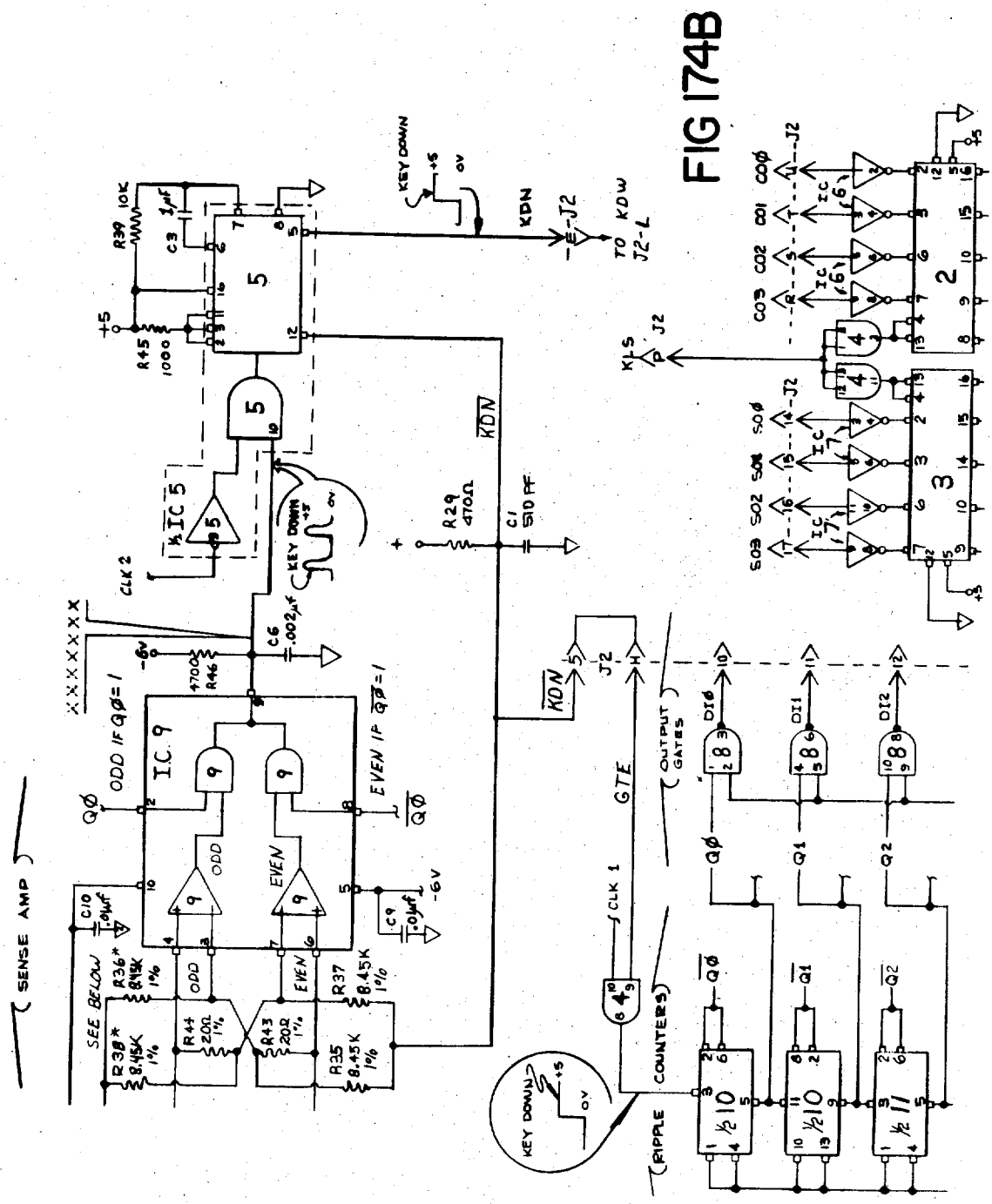

SENSE AMP
SEE BELOW
ODD IF QØ=1
EVEN IF $\overline{QØ}$=1
IC 9
ODD
EVEN
R38* 8.45K 1%
R36* 8.45K 1%
R44 20Ω 1%
R43 20Ω 1%
R35 8.45K 1%
R37 8.45K 1%
C10 .01μf
C9 .01μf
-6V
R46 4700
C6 .002μf
CLK 2
½ IC 5
KEY DOWN
R29 470Ω
C1 510 PF
KDN
J2
GTE
CLK 1
OUTPUT GATES
COUNTERS
RIPPLE
DIØ
DI1
DI2
QØ
Q1
Q2
KLS
CO3
CO2
CO1
COØ
SO3
SO2
SO1
SOØ
TO KDW J2-L
R39 10K
C3 1μf
R45 1000
+5
0V

FIG 174B

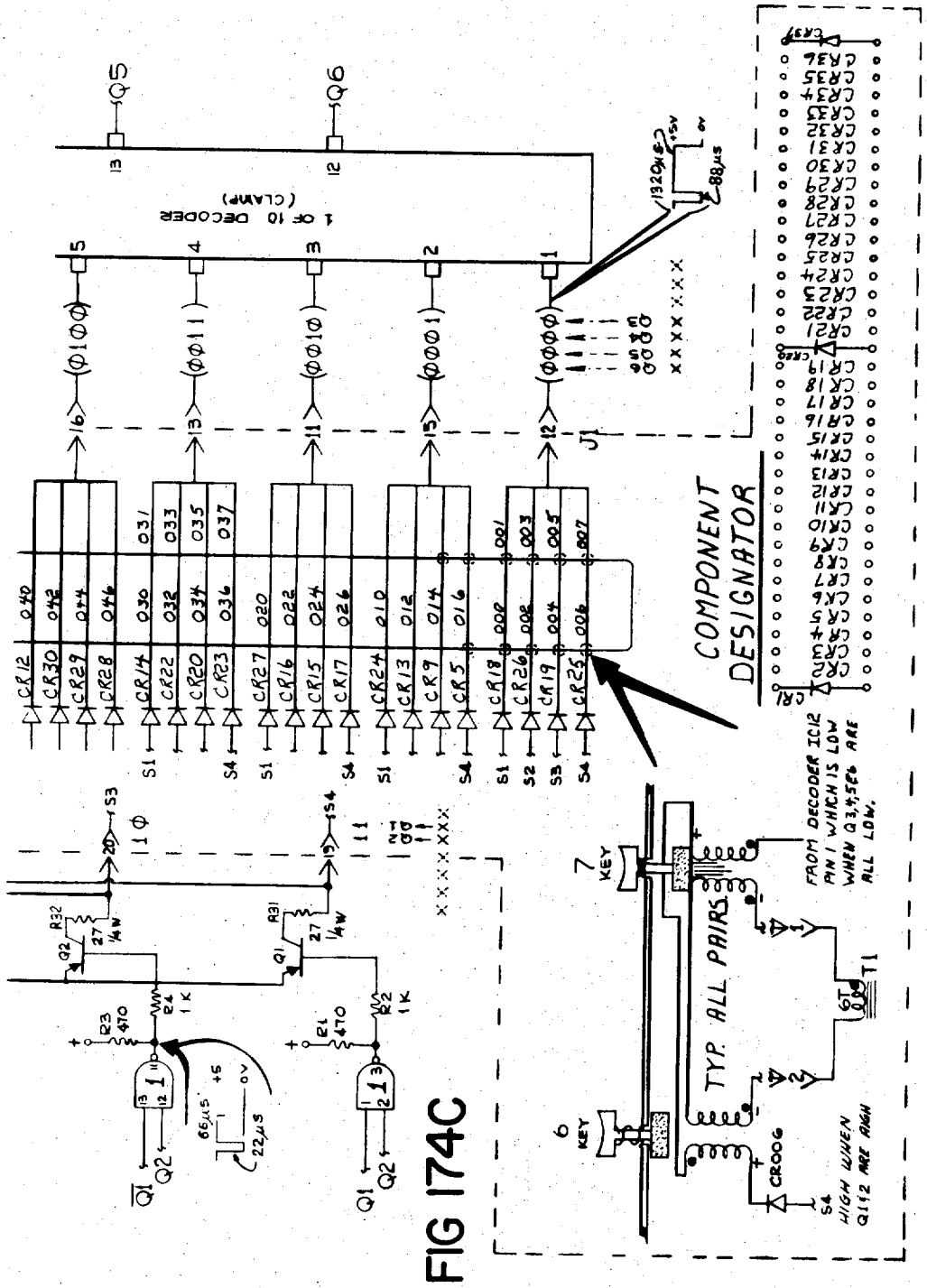
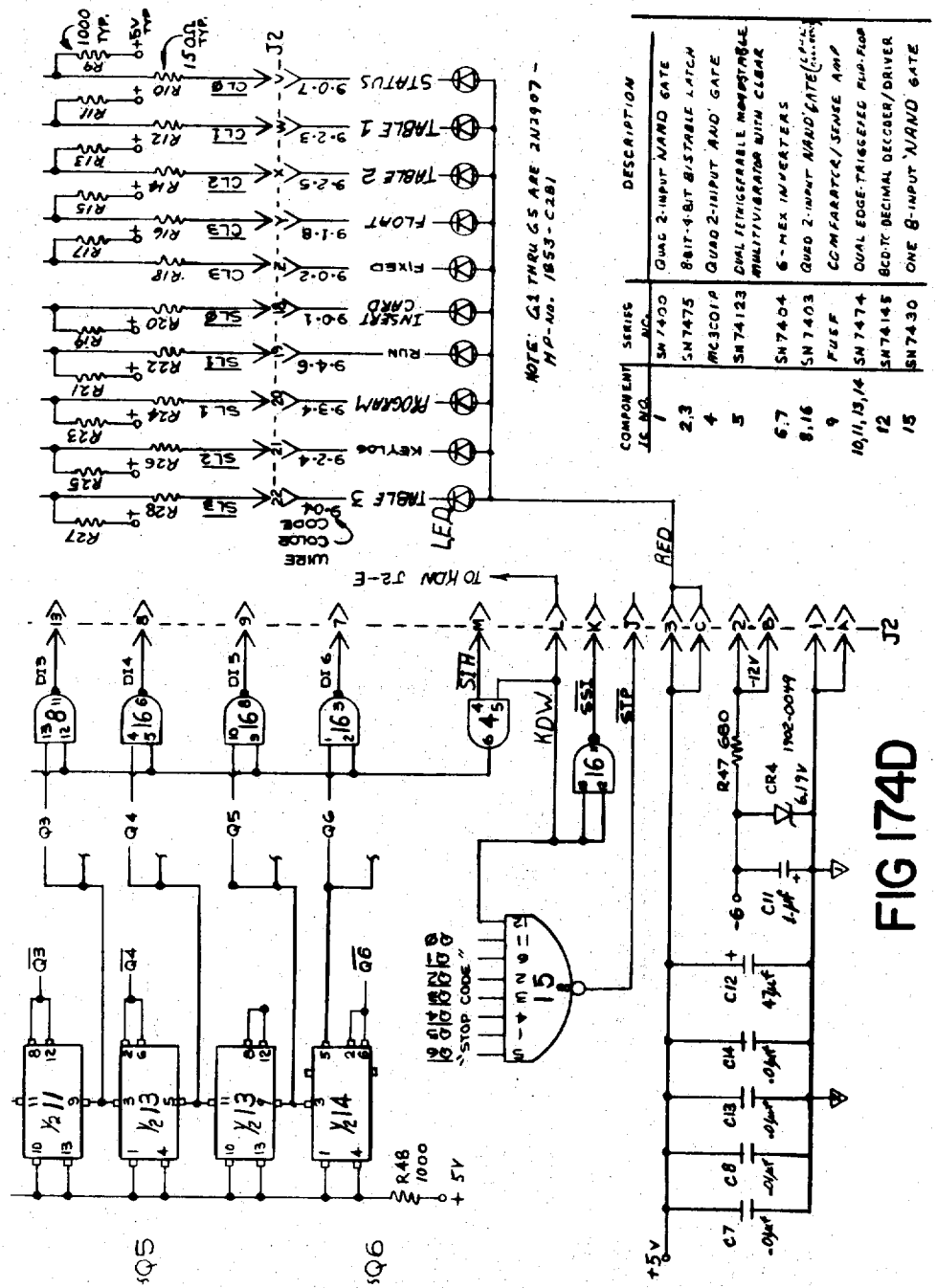


FIG 174C

FIG 175



COMPONENT IC NO.	SERIES NO.	DESCRIPTION
1	SN7400	QUAD 2-INPUT NAND GATE
2,3	SN7475	8BIT-4-BIT BISTABLE LATCH
4	MC3001P	QUAD 2-INPUT AND GATE
5	SN74123	DUAL RETRIGGERABLE MONOSTABLE MULTIVIBRATOR WITH CLEAR
6,7	SN7404	6-HEX INVERTERS
8,16	SN7403	QUAD 2-INPUT NAND GATE(OPEN COLL.)
9	FUSE	
10,11,13,14	SN7474	COMPARATOR/SENSE AMP
12	SN74145	DUAL EDGE-TRIGGERED FLIP-FLOP
15	SN7430	BCD-TO-DECIMAL DECODER/DRIVER

NOTE: Q1 THRU Q5 ARE 2N3907 - MP-NO. 1853-0281

FIG 174D

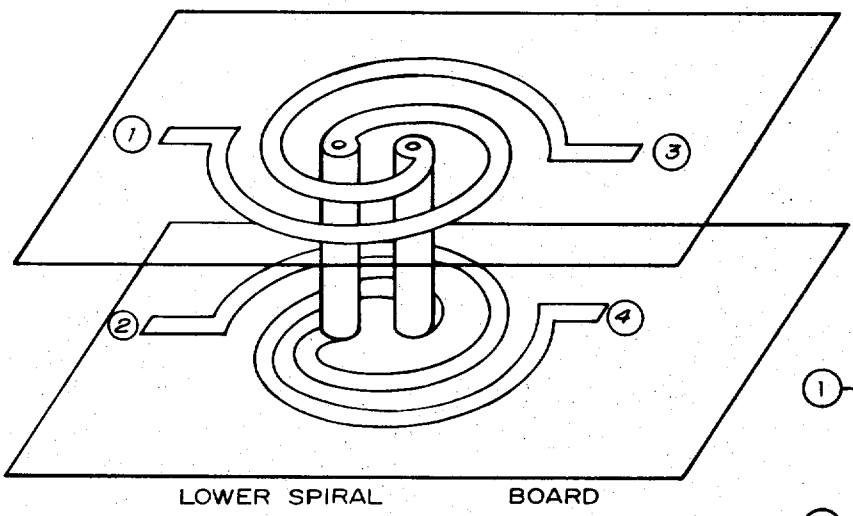


Figure 176

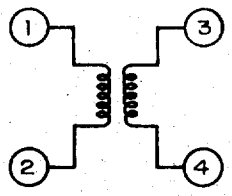
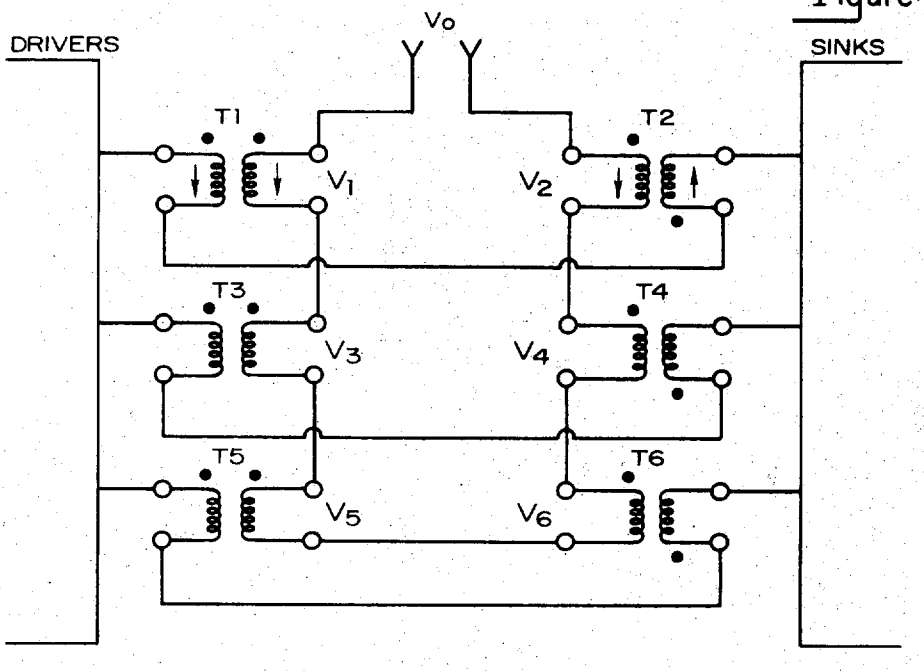


Figure 177



$$V_O = V_1 - V_2 + V_3 - V_4 + V_5 - V_6$$

Figure 178

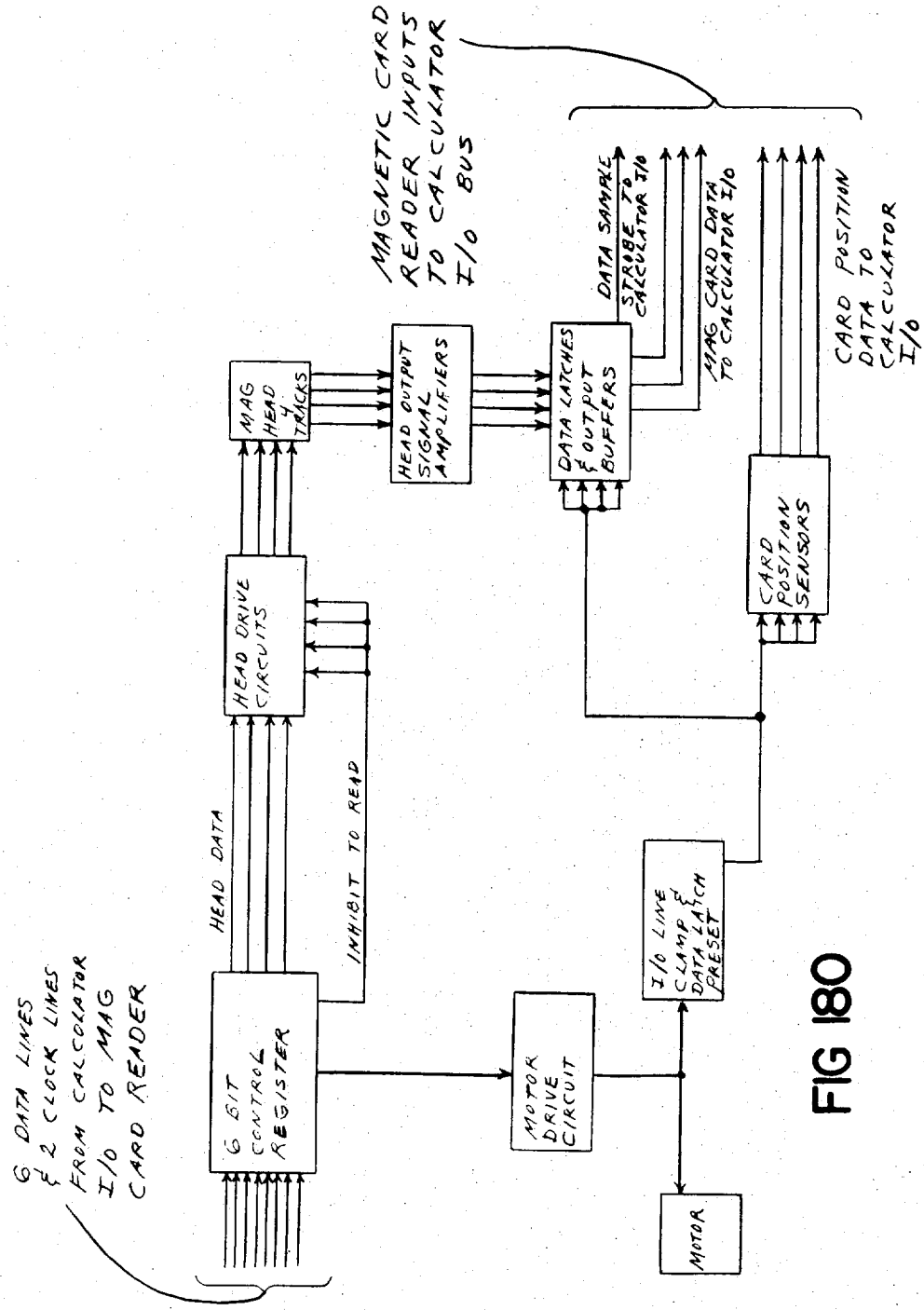

MAGNETIC CARD READER INPUTS TO CALCULATOR I/O BUS
DATA SAMPLE STROBE TO CALCULATOR I/O
MAG CARD DATA TO CALCULATOR I/O
CARD POSITION DATA TO CALCULATOR I/O
MAG HEAD 4 TRACKS
HEAD OUTPUT SIGNAL AMPLIFIERS
DATA LATCHES & OUTPUT BUFFERS
CARD POSITION SENSORS
HEAD DRIVE CIRCUITS
HEAD DATA
INHIBIT TO READ
6 DATA LINES & 2 CLOCK LINES FROM CALCULATOR I/O TO MAG CARD READER
6 BIT CONTROL REGISTER
MOTOR DRIVE CIRCUIT
I/O LINE CLAMP & DATA LATCH PRESET
MOTOR

FIG 180

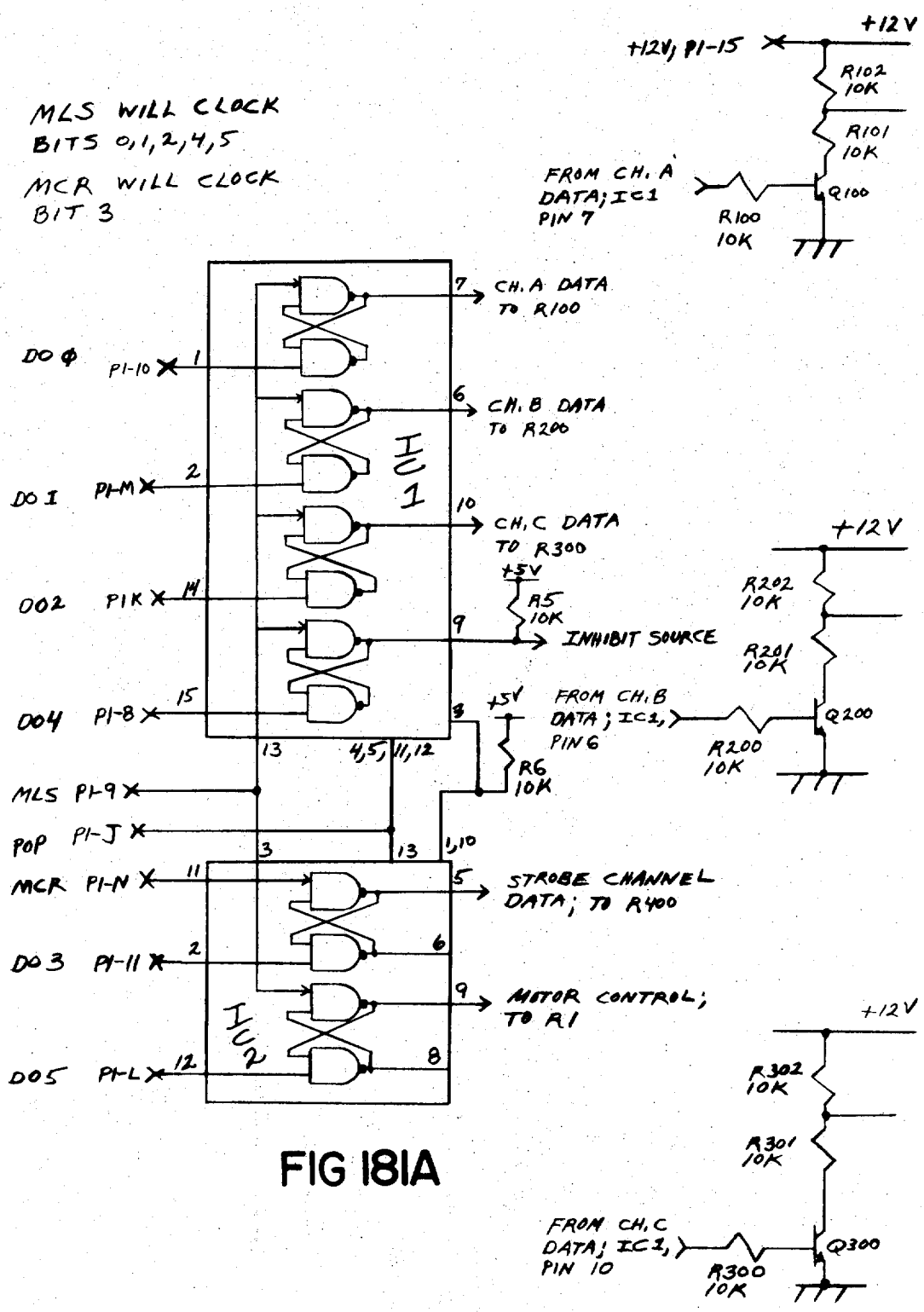
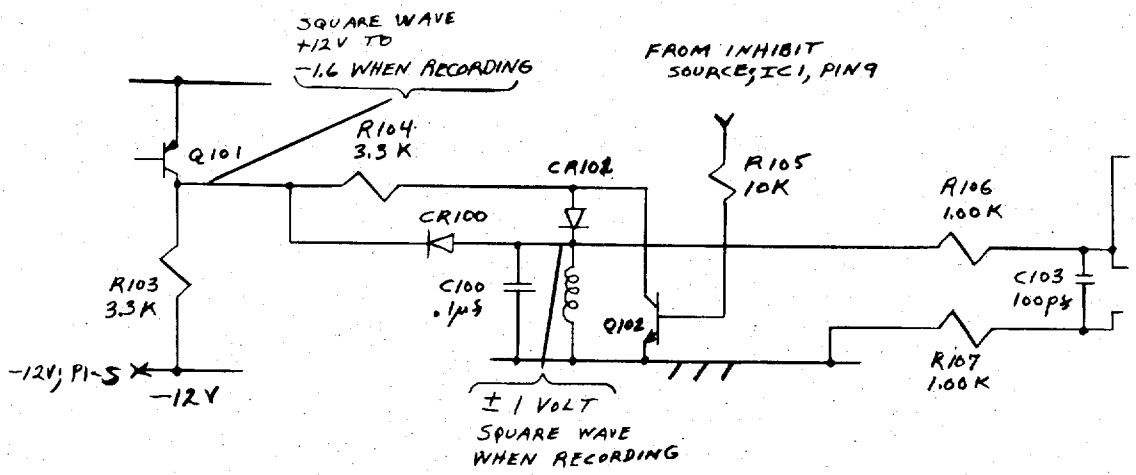
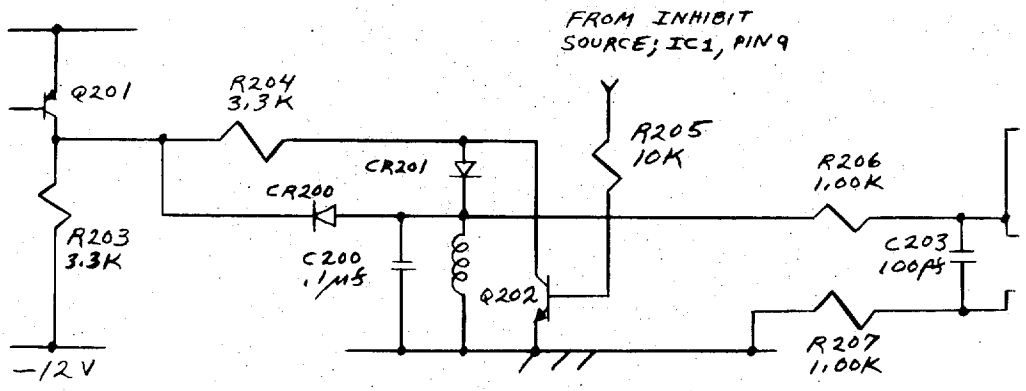

MLS WILL CLOCK BITS 0,1,2,4,5
MCR WILL CLOCK BIT 3
+12V
+12V, P1-15
R102 10K
R101 10K
FROM CH. A DATA; IC1 PIN 7
R100 10K
Q100
IC 1
IC 2
D0 Φ
P1-10
D0 1
P1-M
D0 2
P1-K
D0 4
P1-8
MLS P1-9
POP P1-J
MCR P1-N
D0 3
P1-11
D0 5
P1-L
CH. A DATA TO R100
CH. B DATA TO R200
CH. C DATA TO R300
+5V
R5 10K
INHIBIT SOURCE
FROM CH. B DATA; IC1, PIN 6
R6 10K
R202 10K
R201 10K
Q200
R200 10K
STROBE CHANNEL DATA; TO R400
MOTOR CONTROL; TO R1
R302 10K
R301 10K
FROM CH. C DATA; IC1, PIN 10
Q300
R300 10K

FIG 181A


SQUARE WAVE
+12V TO
-1.6 WHEN RECORDING
FROM INHIBIT
SOURCE; IC1, PIN 9
Q101
R104
3.3K
CR102
R105
10K
R106
1.00K
CR100
R103
3.3K
C100
.1μf
Q102
C103
100pf
R107
1.00K
-12V; PI-5
-12V
± 1 VOLT
SQUARE WAVE
WHEN RECORDING


FROM INHIBIT
SOURCE; IC1, PIN 9
Q201
R204
3.3K
R205
10K
CR201
R206
1.00K
CR200
R203
3.3K
C200
.1μf
C203
100pf
Q202
R207
1.00K
-12V

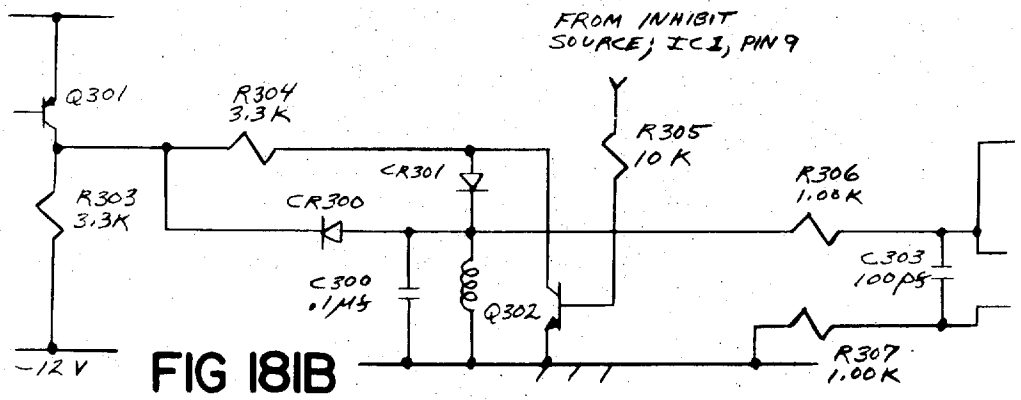

FROM INHIBIT
SOURCE; IC1, PIN 9
Q301
R304
3.3K
R305
10K
CR301
R306
1.00K
R303
3.3K
CR300
C303
100pf
C300
.1μf
Q302
R307
1.00K
-12V

FIG 181B

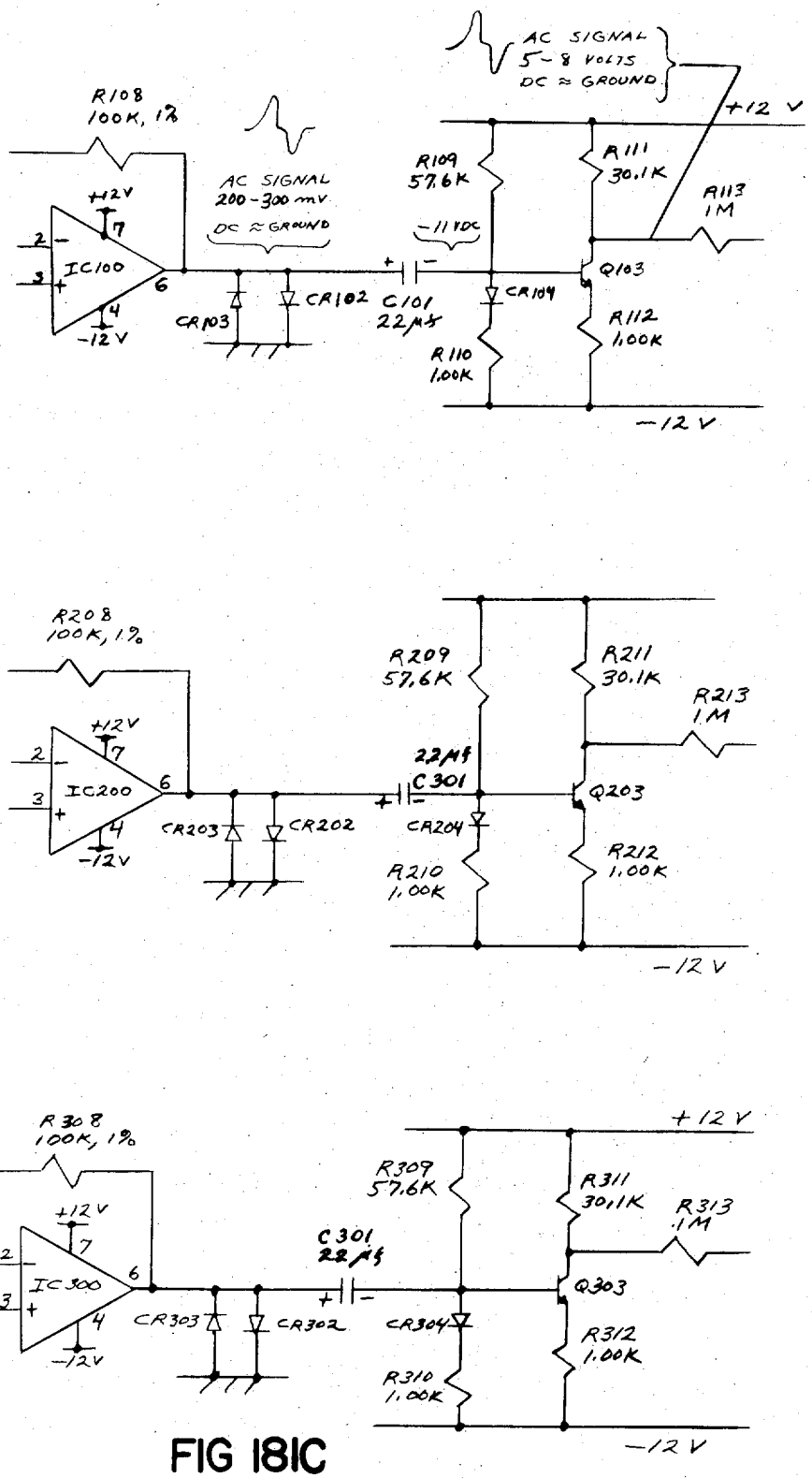

AC SIGNAL
5-8 VOLTS
DC ≈ GROUND
+12 V
R108
100K, 1%
AC SIGNAL
200-300 mV
DC ≈ GROUND
R109
57.6K
-11 VDC
R111
30.1K
R113
1M
+12V
IC100
-12V
CR103
CR102
C101
22μF
CR104
Q103
R112
1.00K
R110
1.00K
-12 V
R208
100K, 1%
R209
57.6K
R211
30.1K
R213
1M
+12V
IC200
-12V
CR203
CR202
22μF
C301
CR204
Q203
R212
1.00K
R210
1.00K
-12 V
R308
100K, 1%
+12 V
R309
57.6K
R311
30.1K
R313
.1M
+12V
IC300
-12V
C301
22μF
CR303
CR302
CR304
Q303
R312
1.00K
R310
1.00K
-12 V

FIG 181C

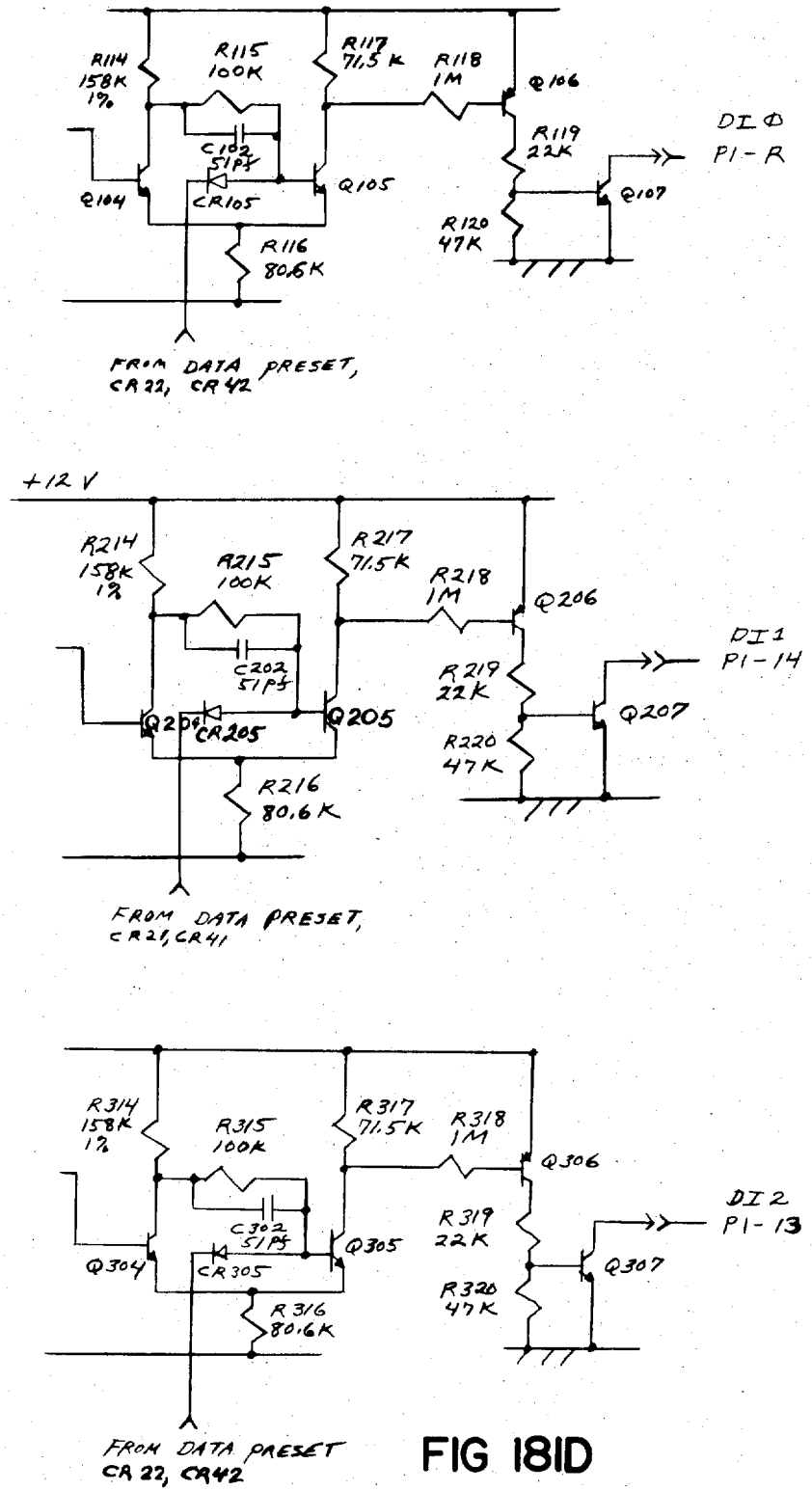

R114
158K
1%
R115
100K
R117
71.5K
R118
1M
Q106
R119
22K
DI Ø
P1-R
C102
51PF
Q104
CR105
Q105
Q107
R120
47K
R116
80.6K
FROM DATA PRESET,
CR22, CR42
+12 V
R214
158K
1%
R215
100K
R217
71.5K
R218
1M
Q206
DI 1
P1-14
C202
51PF
Q204
CR205
Q205
R219
22K
Q207
R220
47K
R216
80.6K
FROM DATA PRESET,
CR21, CR41
R314
158K
1%
R315
100K
R317
71.5K
R318
1M
Q306
DI 2
P1-13
C302
51PF
Q304
CR305
Q305
R319
22K
Q307
R320
47K
R316
80.6K
FROM DATA PRESET
CR22, CR42

FIG 18ID

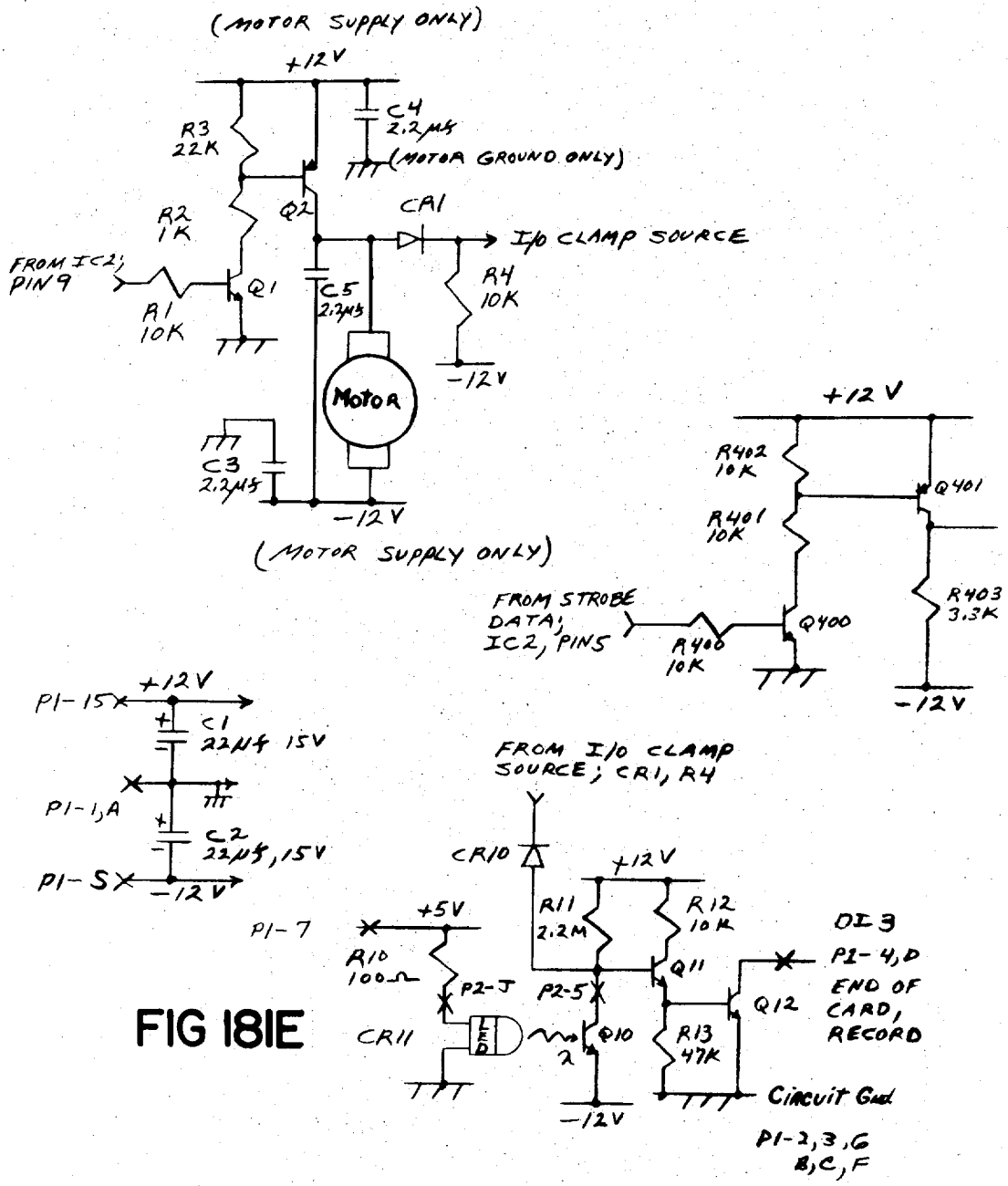

(MOTOR SUPPLY ONLY)
+12V
R3 22K
C4 2.2μF
(MOTOR GROUND ONLY)
R2 1K
Q2
CR1
I/O CLAMP SOURCE
FROM IC2; PIN 9
Q1
R1 10K
C5 2.2μF
R4 10K
-12V
Motor
C3 2.2μF
-12V
(MOTOR SUPPLY ONLY)
+12V
R402 10K
Q401
R401 10K
R403 3.3K
FROM STROBE DATA; IC2, PIN5
R400 10K
Q400
-12V
P1-15
+12V
C1 22μF 15V
P1-1,A
C2 22μF, 15V
P1-5
-12V
P1-7
FROM I/O CLAMP SOURCE; CR1, R4
CR10
+12V
R11 2.2M
R12 10K
+5V
R10 100Ω
P2-J
P2-5
Q11
DI 3
P1-4,D
END OF CARD, RECORD
Q12
CR11
LED
Q10
R13 47K
Circuit Gnd
P1-2,3,G,B,C,F
-12V

FIG 181E

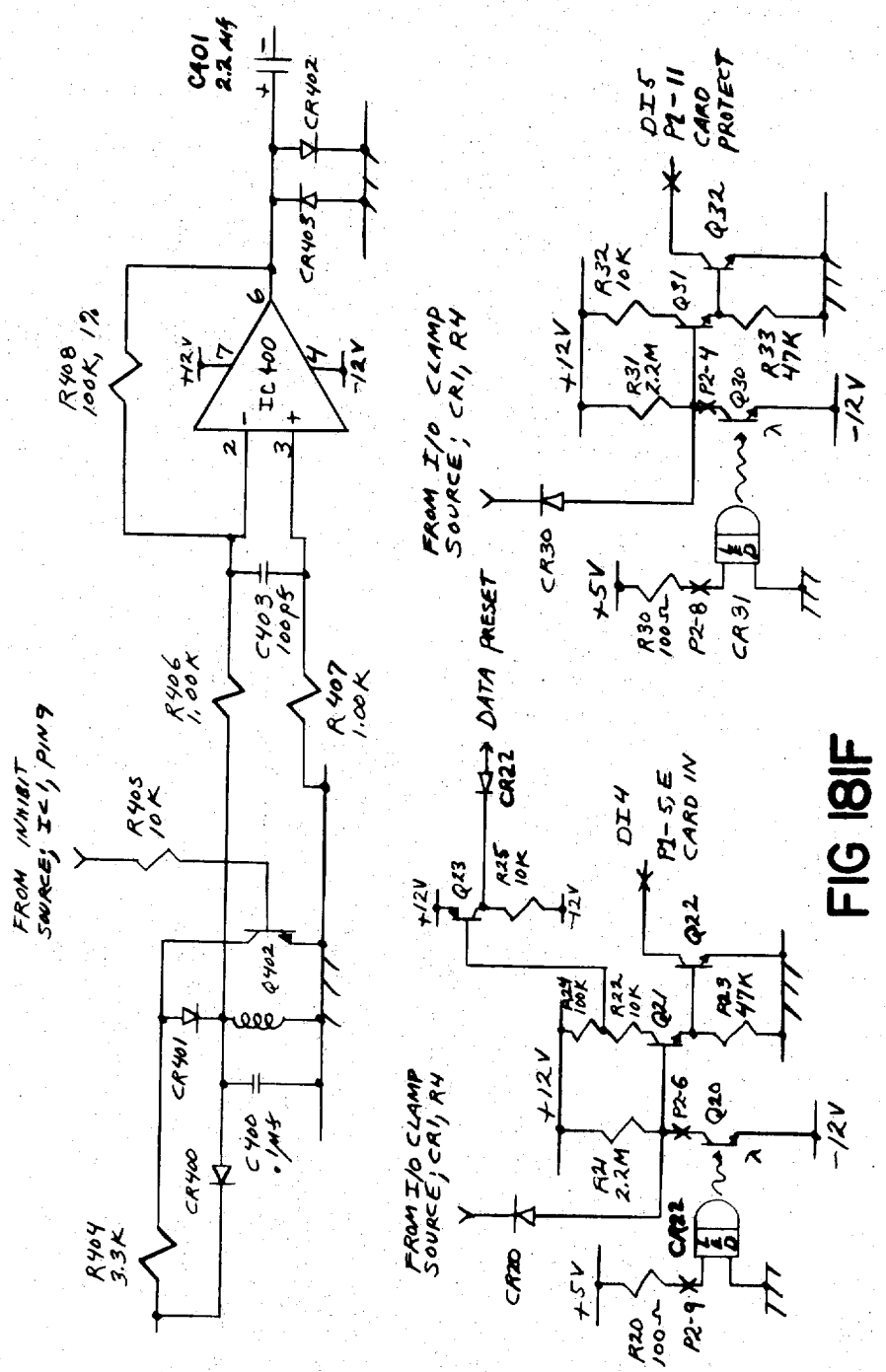

C401
2.2 MF
CR402
CR405
R408
100K, 1%
+12V
IC 400
-12V
C403
100PF
R406
1.00K
R407
1.00K
FROM INHIBIT
SOURCE; IC1, PIN 9
R405
10K
Q402
CR401
CR400
C400
.1MF
R404
3.3K
DATA PRESET
CR22
R25
10K
Q23
+12V
-12V
DI4
P1-5 E
CARD IN
Q22
R24
100K
R22
10K
Q21
R23
47K
+12V
P2-6
Q20
-12V
FROM I/O CLAMP
SOURCE; CR1, R4
R21
2.2M
CR20
CR21
+5V
R20
100Ω
P2-9
DI5
P1-11
CARD
PROTECT
Q32
R32
10K
Q31
+12V
R31
2.2M
P2-4
Q30
R33
47K
-12V
FROM I/O CLAMP
SOURCE; CR1, R4
CR30
+5V
R30
100Ω
P2-8
CR31

FIG 18F

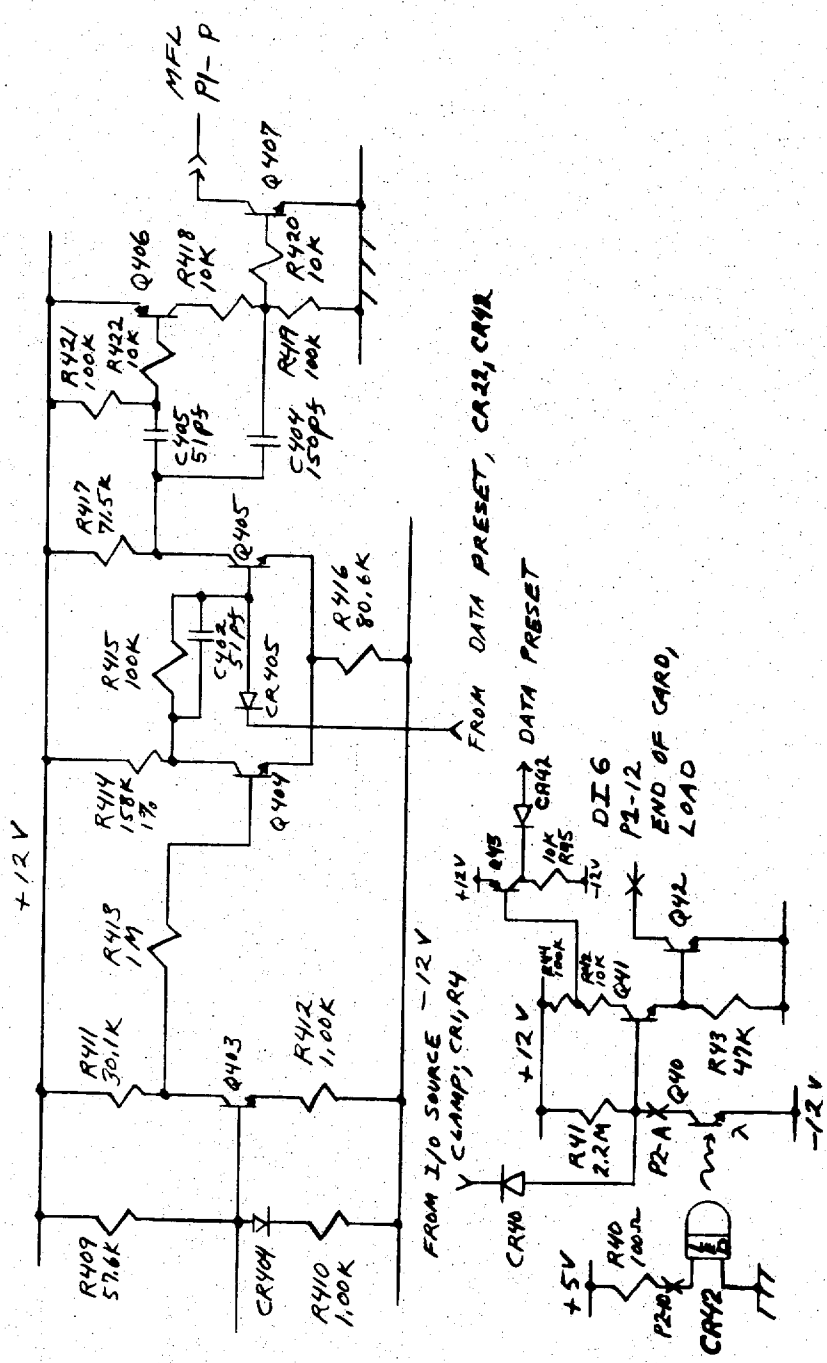

MFL
P1-P
Q407
Q406
R418
10K
R420
10K
R421
100K
R422
10K
R419
100K
C405
51pf
C404
150pf
R417
71.5K
Q405
R416
80.6K
R415
100K
C402
51pf
CR405
FROM DATA PRESET, CR22, CR42
DATA PRESET
R414
158K
1%
Q404
+12V
R413
1M
R411
30.1K
Q403
R412
1.00K
-12V
R409
57.6K
CR404
R410
1.00K
FROM I/O SOURCE CLAMP; CR1, R4
CR40
R41
2.2M
P2-A
Q40
R43
47K
Q41
Q42
DI6
P1-12
END OF CARD, LOAD
+5V
R40
100Ω
P2-10
CR42

FIG 181G

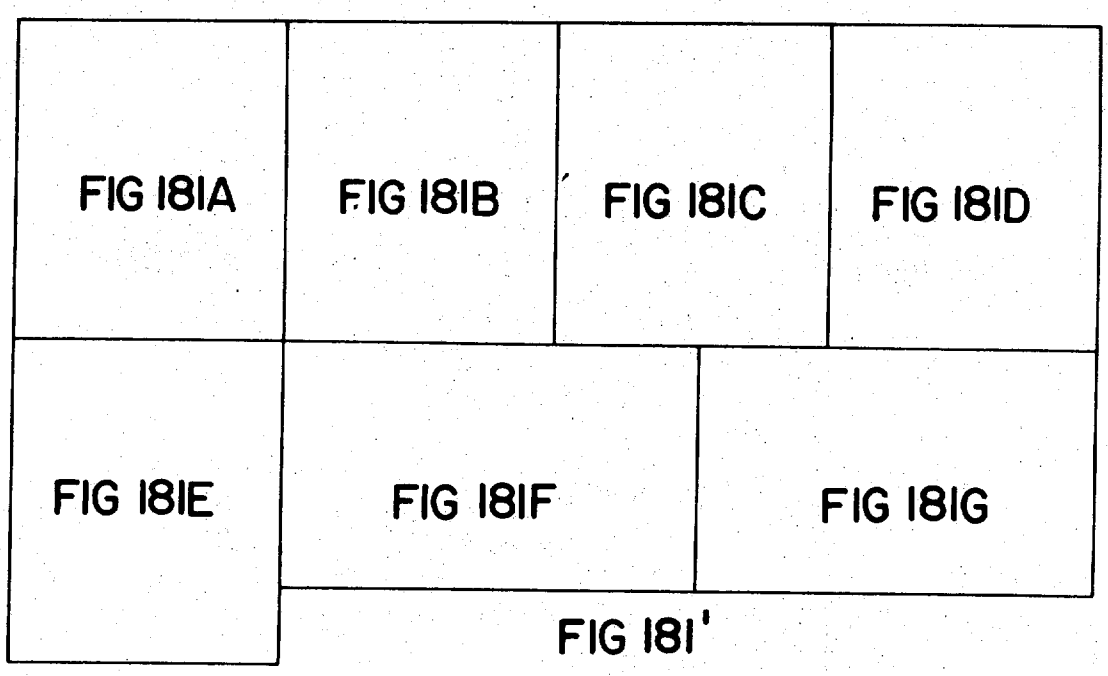


FIG 181'

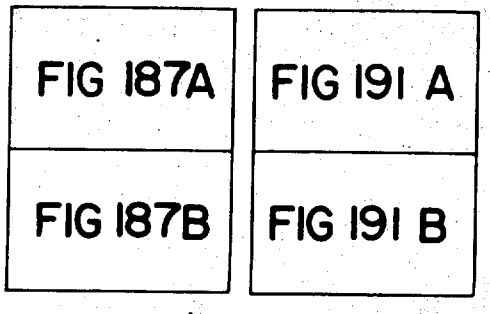


FIG 187' FIG 191

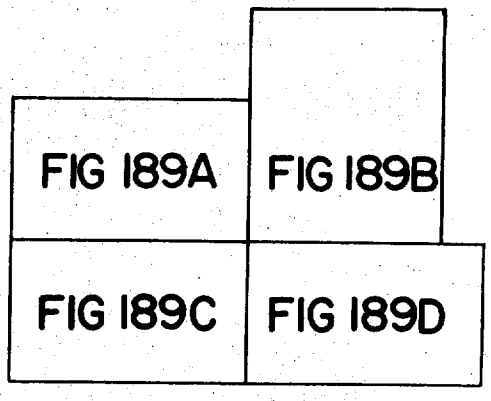


FIG 190

FIG. 199'

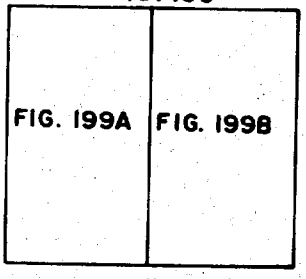


FIG. 184'

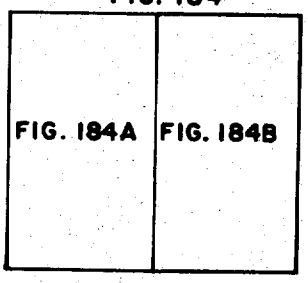
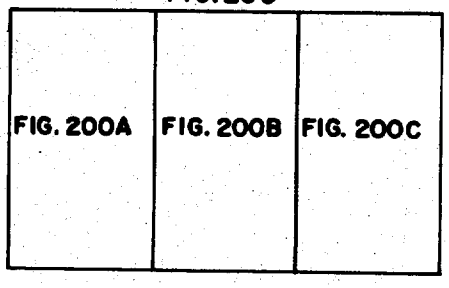


FIG. 200'



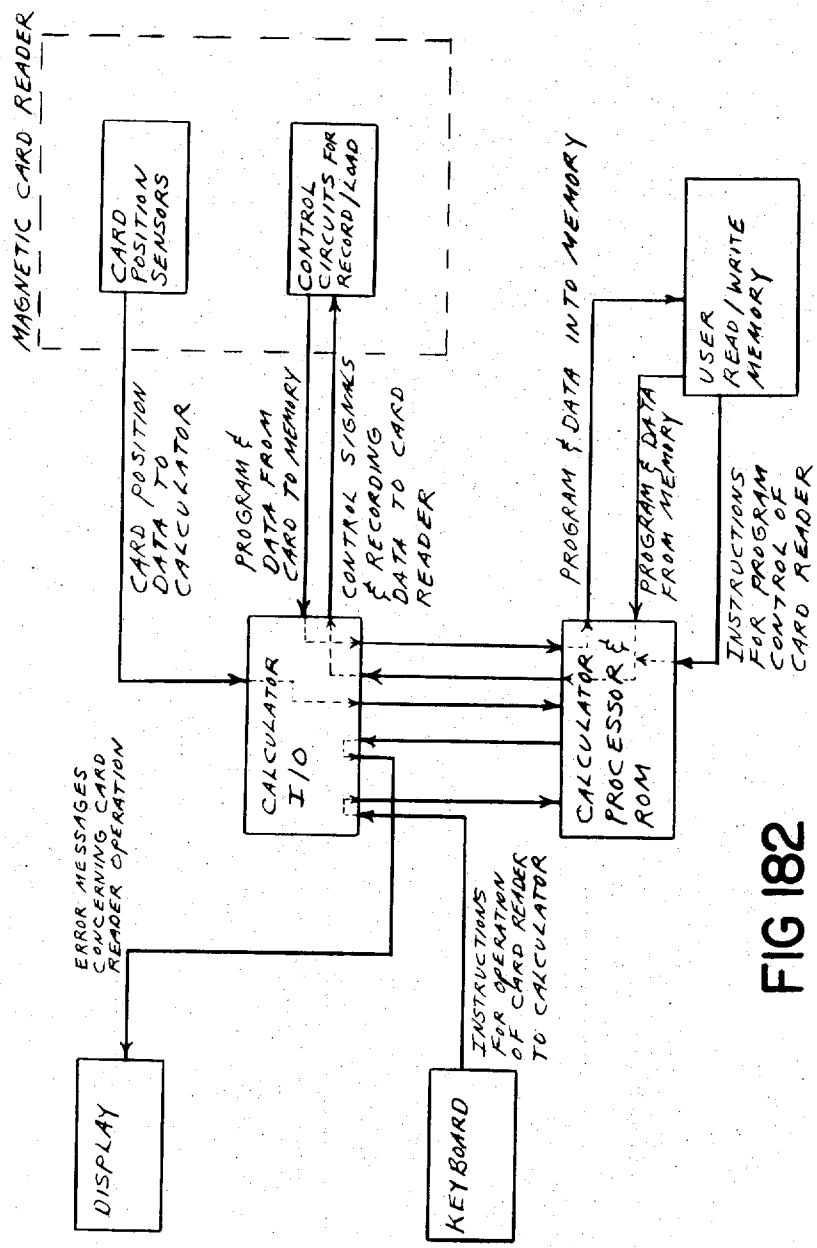

MAGNETIC CARD READER
CARD POSITION SENSORS
CONTROL CIRCUITS FOR RECORD/LOAD
CARD POSITION DATA TO CALCULATOR
PROGRAM & DATA FROM CARD TO MEMORY
CONTROL SIGNALS & RECORDING DATA TO CARD READER
PROGRAM & DATA INTO MEMORY
USER READ/WRITE MEMORY
PROGRAM & DATA FROM MEMORY
INSTRUCTIONS FOR PROGRAM CONTROL OF CARD READER
CALCULATOR I/O
CALCULATOR PROCESSOR & ROM
ERROR MESSAGES CONCERNING CARD READER OPERATION
INSTRUCTIONS FOR OPERATION OF CARD READER TO CALCULATOR
DISPLAY
KEYBOARD

FIG 182

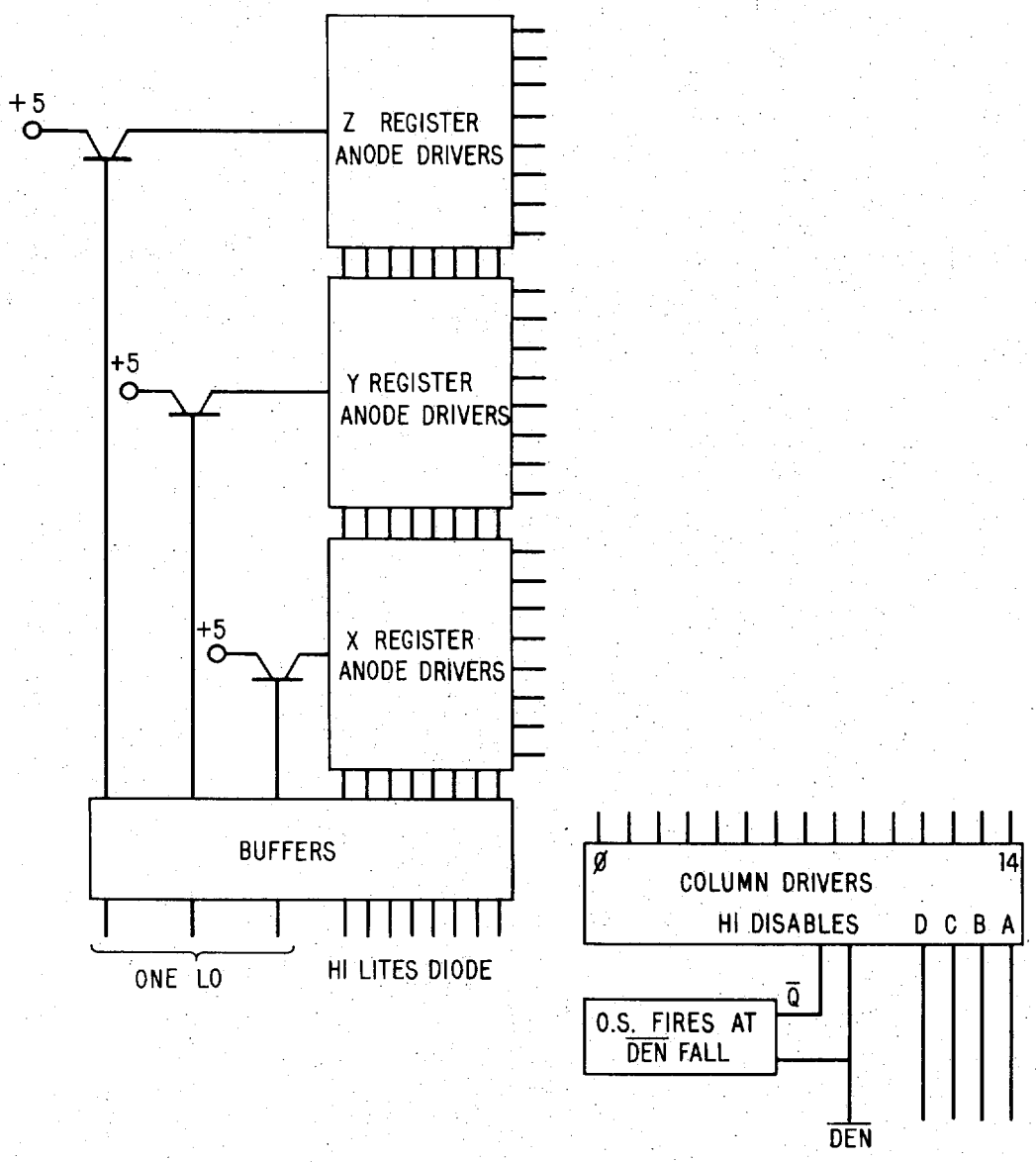
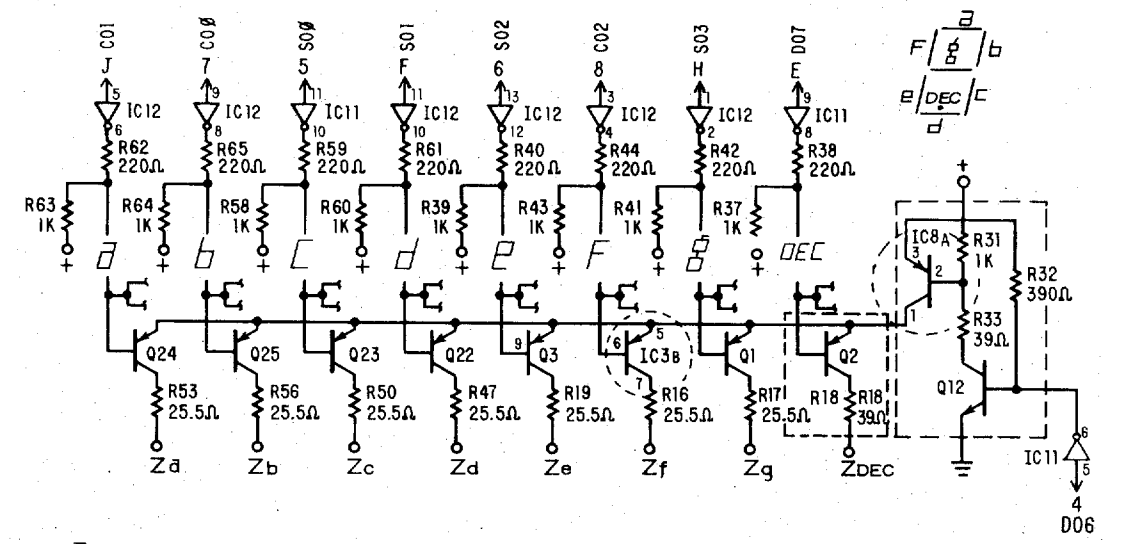
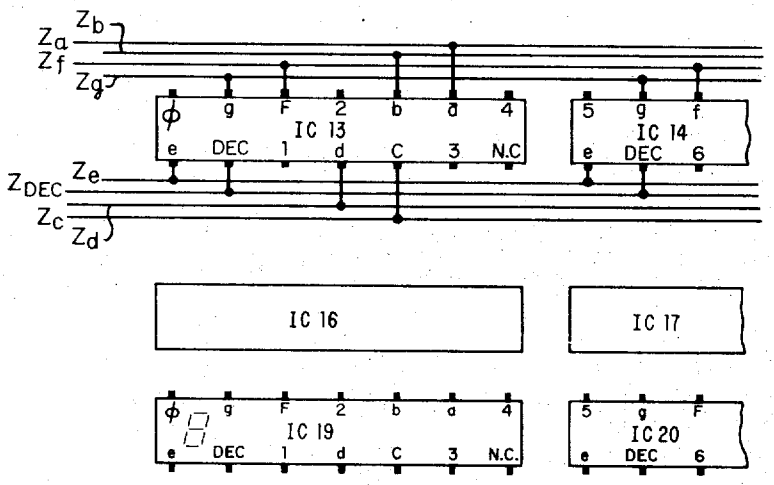

+5
Z REGISTER ANODE DRIVERS
+5
Y REGISTER ANODE DRIVERS
+5
X REGISTER ANODE DRIVERS
BUFFERS
ONE LO
HI LITES DIODE
ø
COLUMN DRIVERS
14
HI DISABLES
D C B A
Q̄
O.S. FIRES AT DEN FALL
DEN

Fig 183


CO1
CO0
SO0
SO1
SO2
CO2
SO3
DO7
IC12
IC11
R62 220Ω
R65 220Ω
R59 220Ω
R61 220Ω
R40 220Ω
R44 220Ω
R42 220Ω
R38 220Ω
R63 1K
R64 1K
R58 1K
R60 1K
R39 1K
R43 1K
R41 1K
R37 1K
Q24
Q25
Q23
Q22
Q3
Q1
Q2
IC3B
IC8A
R31 1K
R32 390Ω
R33 39Ω
Q12
R53 25.5Ω
R56 25.5Ω
R50 25.5Ω
R47 25.5Ω
R19 25.5Ω
R16 25.5Ω
R17 25.5Ω
R18 39Ω
Za
Zb
Zc
Zd
Ze
Zf
Zg
ZDEC
IC11
DO6


Za
Zb
Zf
Zg
Ze
ZDEC
Zc
Zd
IC 13
IC 14
IC 16
IC 17
IC 19
IC 20
DEC
N.C.

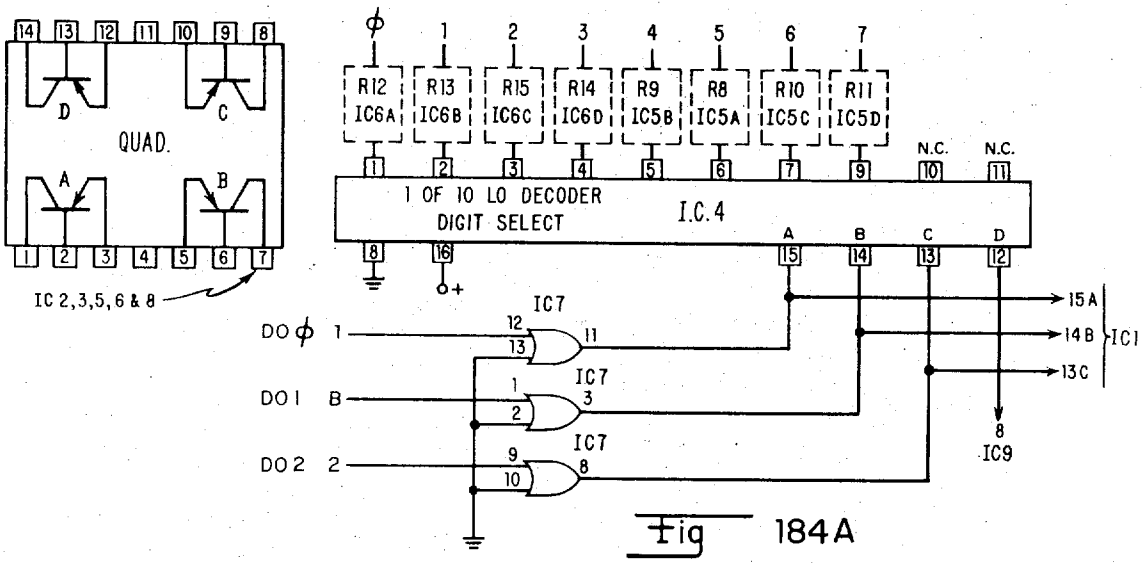
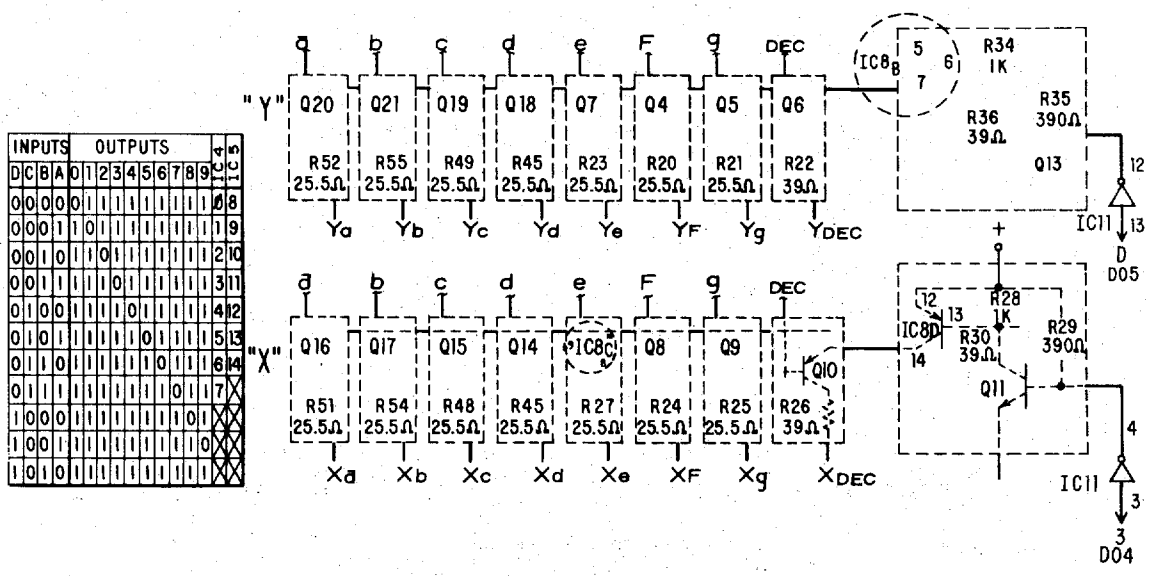
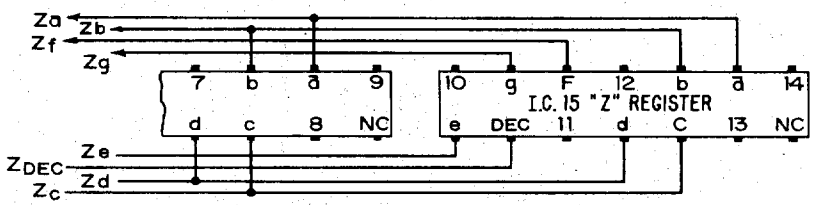

QUAD.
IC 2,3,5,6 & 8
R12 IC6A
R13 IC6B
R15 IC6C
R14 IC6D
R9 IC5B
R8 IC5A
R10 IC5C
R11 IC5D
1 OF 10 LO DECODER
DIGIT SELECT
I.C.4
N.C.
IC7
DO φ
DO 1
DO 2
15A
14B
13C
IC1
IC9

Fig 184A





HEX INVERTER IC 11,12
QUAD 2-INPUT "OR" IC 7,9
8601 O.S. IC 10
1 OF 10 DECODER IC 1,4
NPN XISTER Q11, 12, 13
PNP XISTER Q1-10, 14-25
PNP QUAD PACKS IC 2,3,5,6,8

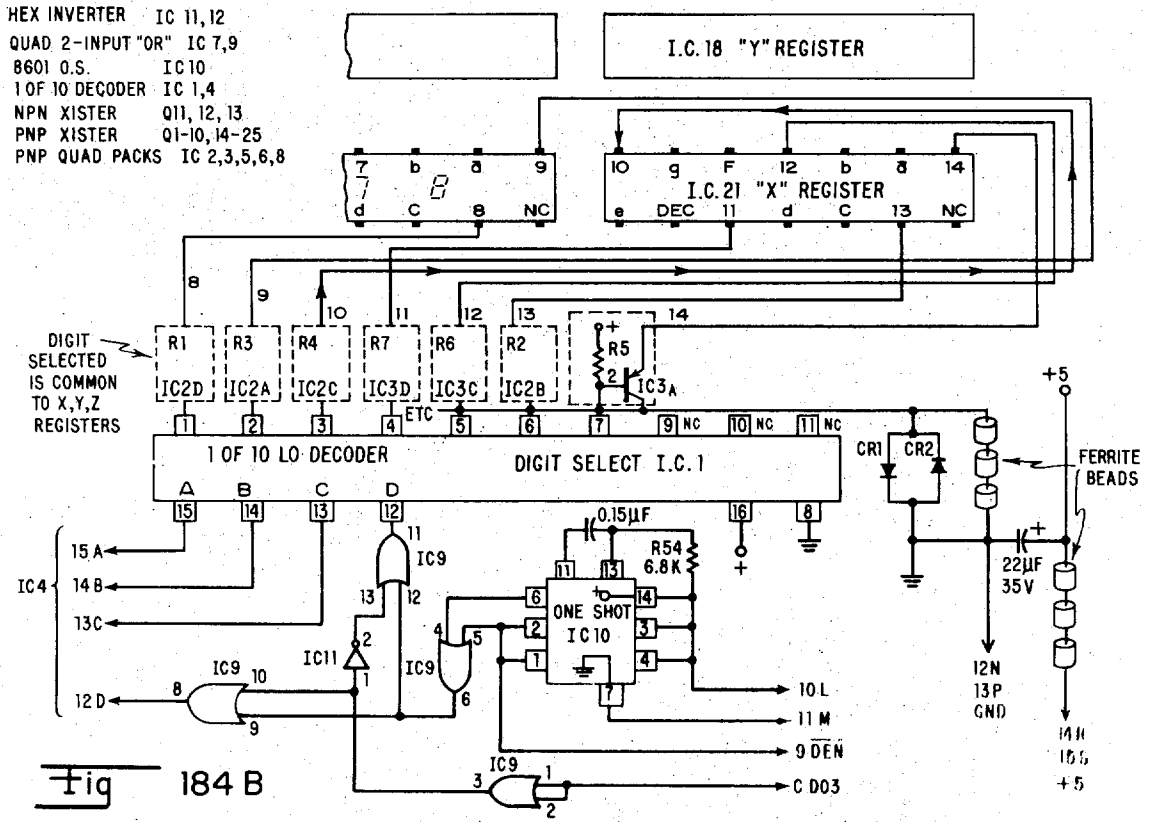


Fig 184 B

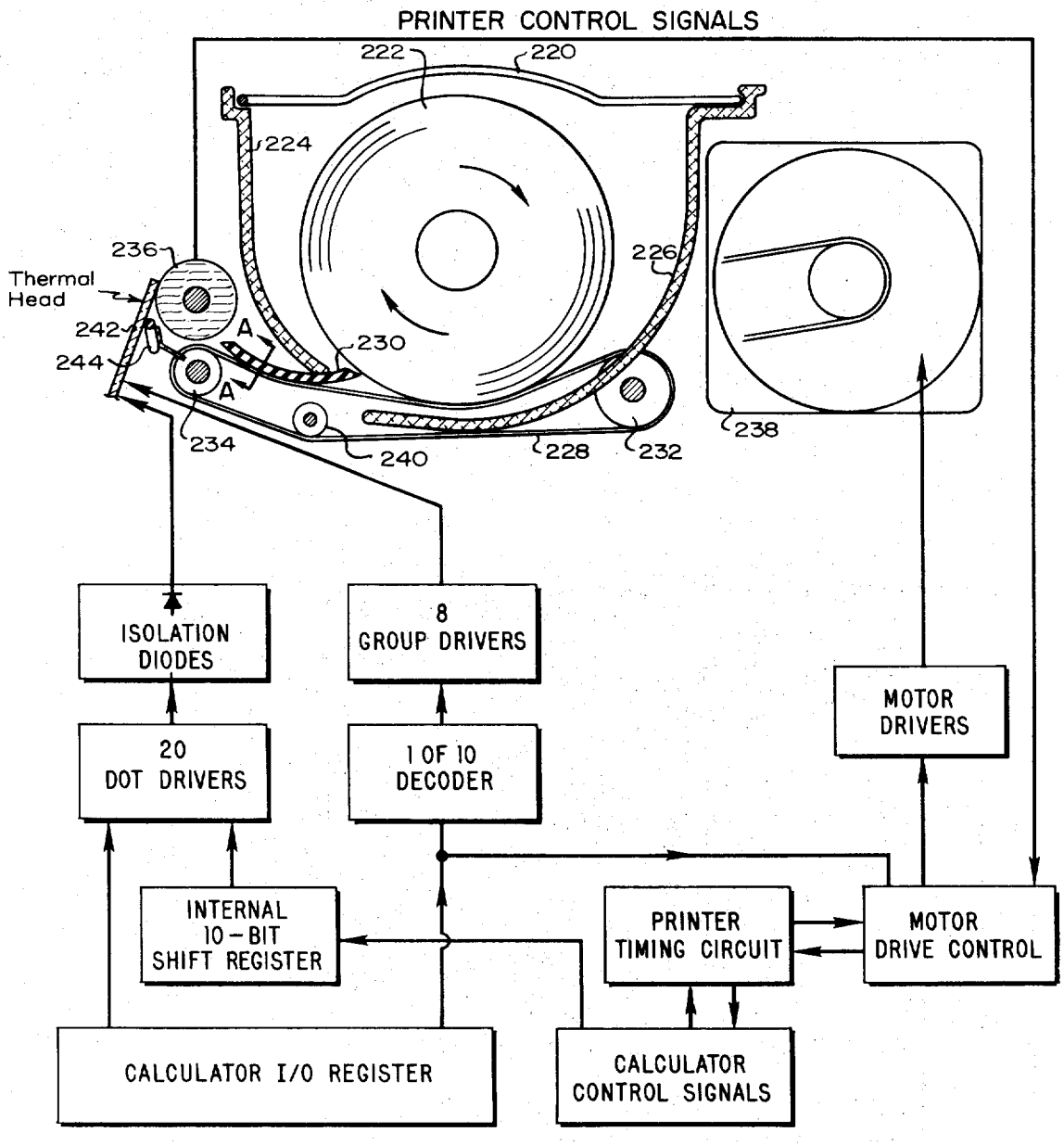


Figure 185

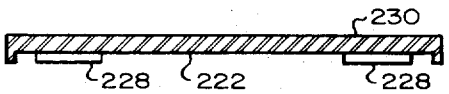


Fig 186

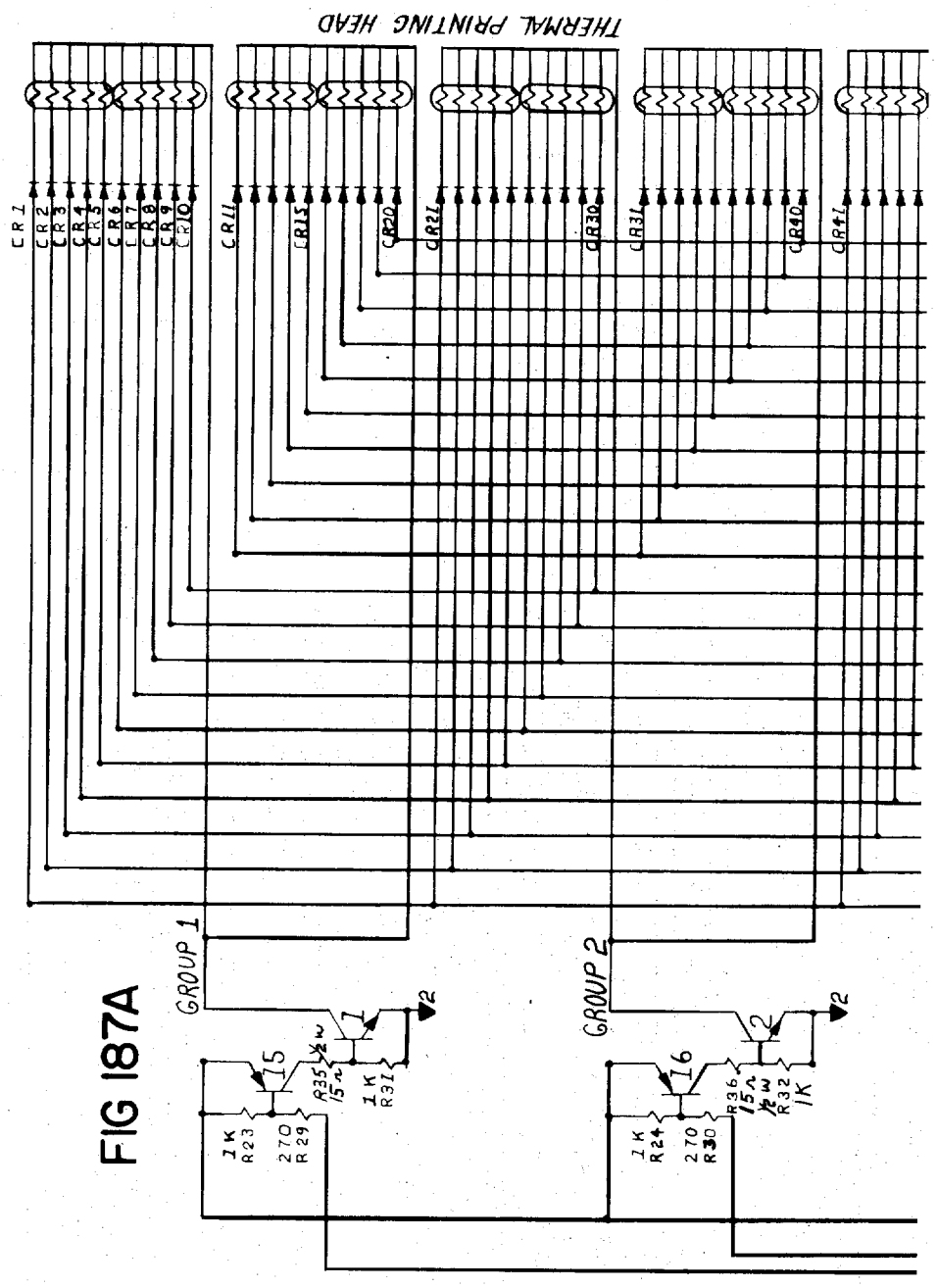

THERMAL PRINTING HEAD
CR1
CR2
CR3
CR4
CR5
CR6
CR7
CR8
CR9
CR10
CR11
CR15
CR20
CR21
CR30
CR31
CR40
CR41
GROUP 1
GROUP 2
I5
I6
1
2
R35 15Ω ½W
1K R31
1K R23
270 R29
R36 15Ω ½W
R32 1K
1K R24
270 R30

FIG 187A

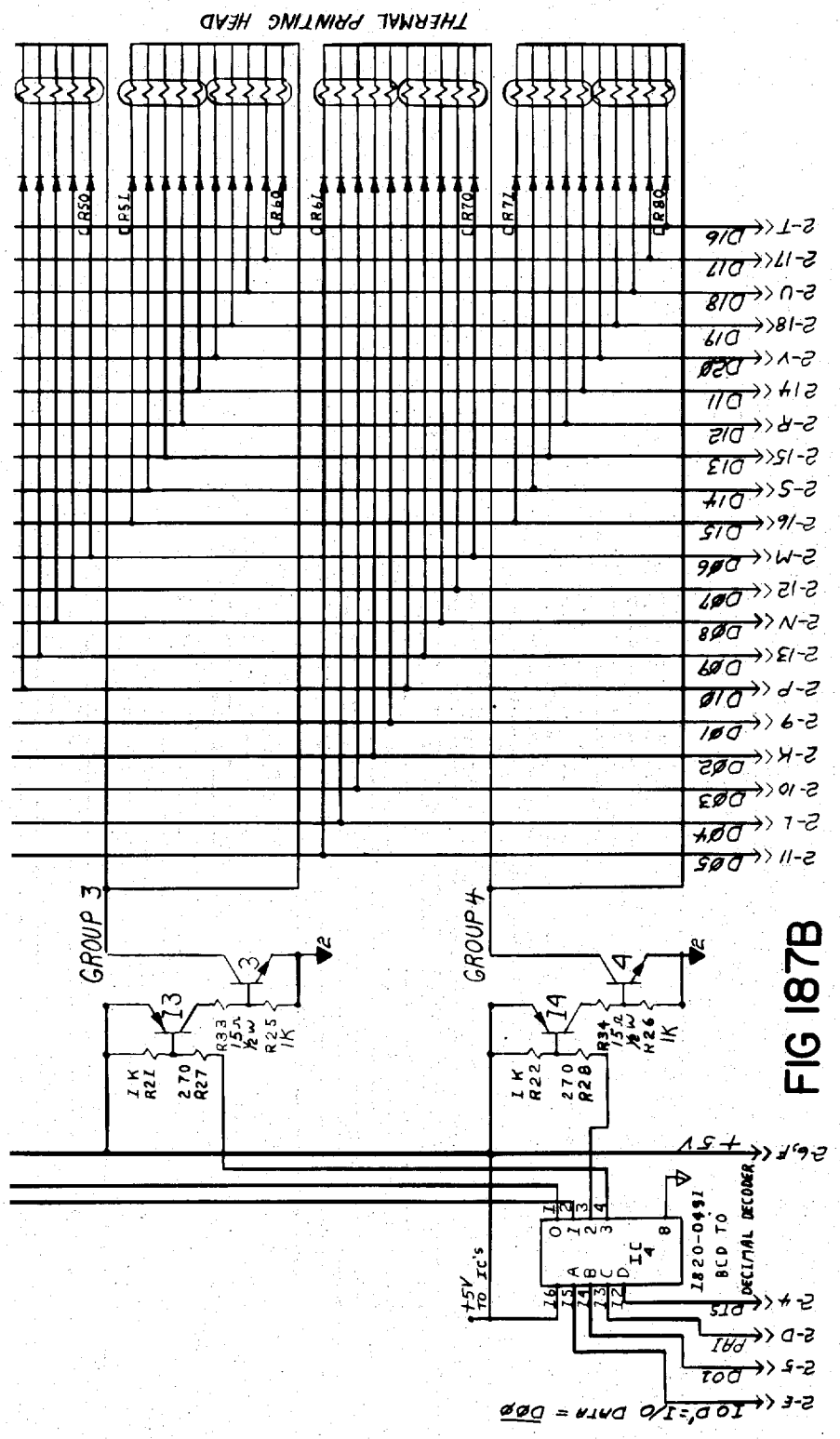

THERMAL PRINTING HEAD
GROUP 3
GROUP 4
1820-0491
BCD TO DECIMAL DECODER
+5V TO IC's
IOD'=I/O DATA = DØØ

FIG 187B

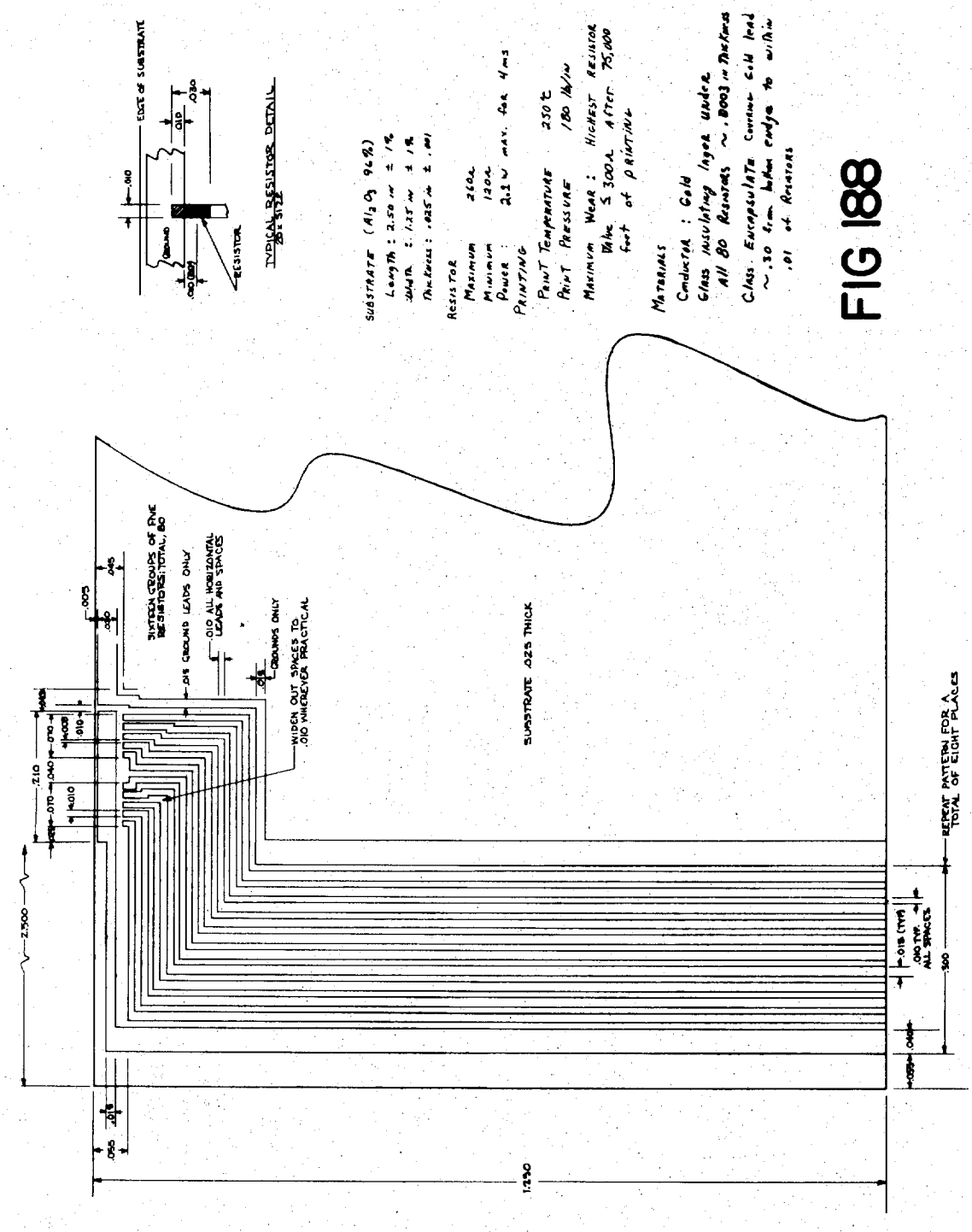

TYPICAL RESISTOR DETAIL
EDGE OF SUBSTRATE
GROUND
RESISTOR
SUBSTRATE (Al_2O_3 96%)
Length : 2.50 in ± 1%
Width : 1.25 in ± 1%
Thickness : .025 in ± .001
Resistor
Maximum 260Ω
Minimum 120Ω
Power : 2.1 W max. for 4 ms
Printing
Print Temperature 230 t
Print Pressure 180 lb/in
Maximum Wear : Highest Resistor Value ≤ 300Ω after 75,000 feet of printing
Materials
Conductor : Gold
Glass insulating layer under All 80 Resistors ~ .0003 in Thickness
Glass Encapsulate Covers Gold lead ~.30 from bottom edge to within .01 of Resistors
SIXTEEN GROUPS OF FIVE RESISTORS; TOTAL, 80
.015 GROUND LEADS ONLY
.010 ALL HORIZONTAL LEADS AND SPACES
GROUNDS ONLY
WIDEN OUT SPACES TO .010 WHEREVER PRACTICAL
SUBSTRATE .025 THICK
REPEAT PATTERN FOR A TOTAL OF EIGHT PLACES
2.500
1.250
.500

FIG 188

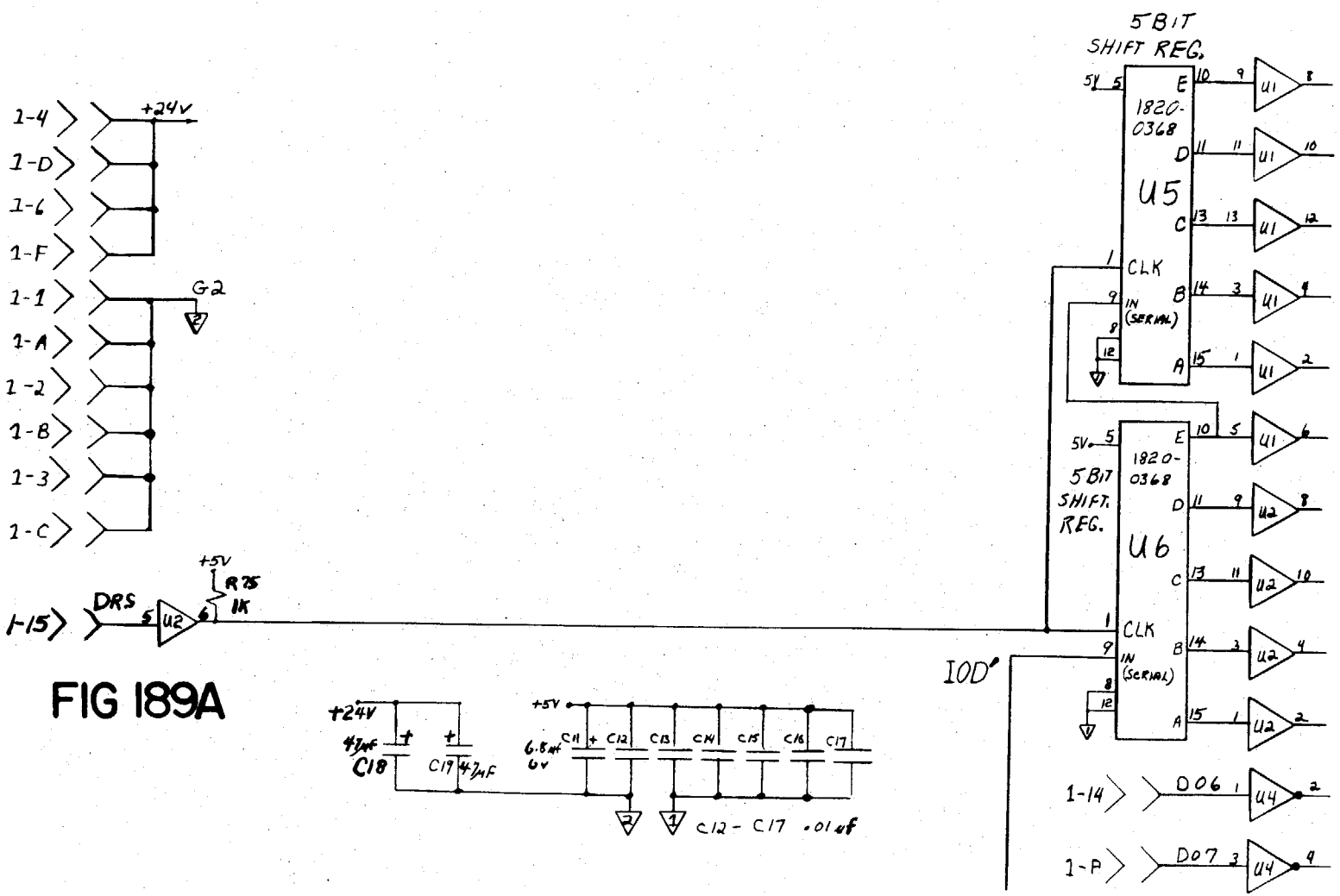

1-4
1-D
1-6
1-F
1-1
1-A
1-2
1-B
1-3
1-C
+24V
G2
1-15
DRS
U2
+5V
R25
1K
5 BIT SHIFT REG.
U5
1820-0368
E
D
C
B
A
CLK
IN (SERIAL)
U1
5V
5 BIT SHIFT REG.
U6
1820-0368
U2
IOD'
1-14
D06
U4
1-P
D07
U4
+24V
47μF
C18
C19 47μF
+5V
6.8μF 6V
C11
C12
C13
C14
C15
C16
C17
C12 - C17 .01μF

FIG 189A

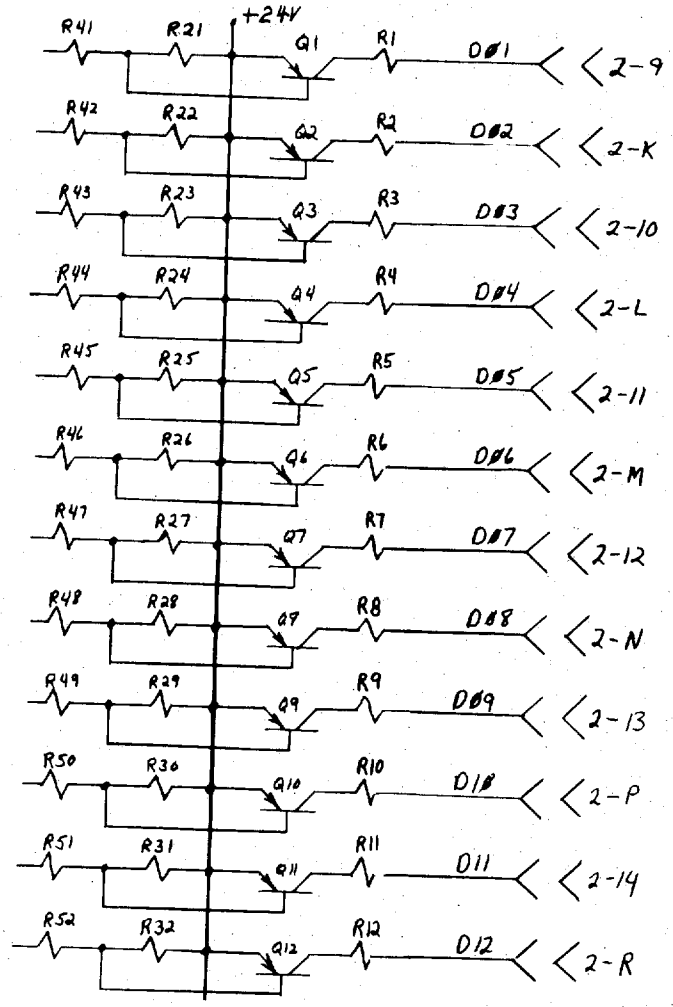

+24V
R41
R21
Q1
R1
D#1
2-9
R42
R22
Q2
R2
D#2
2-K
R43
R23
Q3
R3
D#3
2-10
R44
R24
Q4
R4
D#4
2-L
R45
R25
Q5
R5
D#5
2-11
R46
R26
Q6
R6
D#6
2-M
R47
R27
Q7
R7
D#7
2-12
R48
R28
Q8
R8
D#8
2-N
R49
R29
Q9
R9
D#9
2-13
R50
R30
Q10
R10
D10
2-P
R51
R31
Q11
R11
D11
2-14
R52
R32
Q12
R12
D12
2-R

FIG 189B

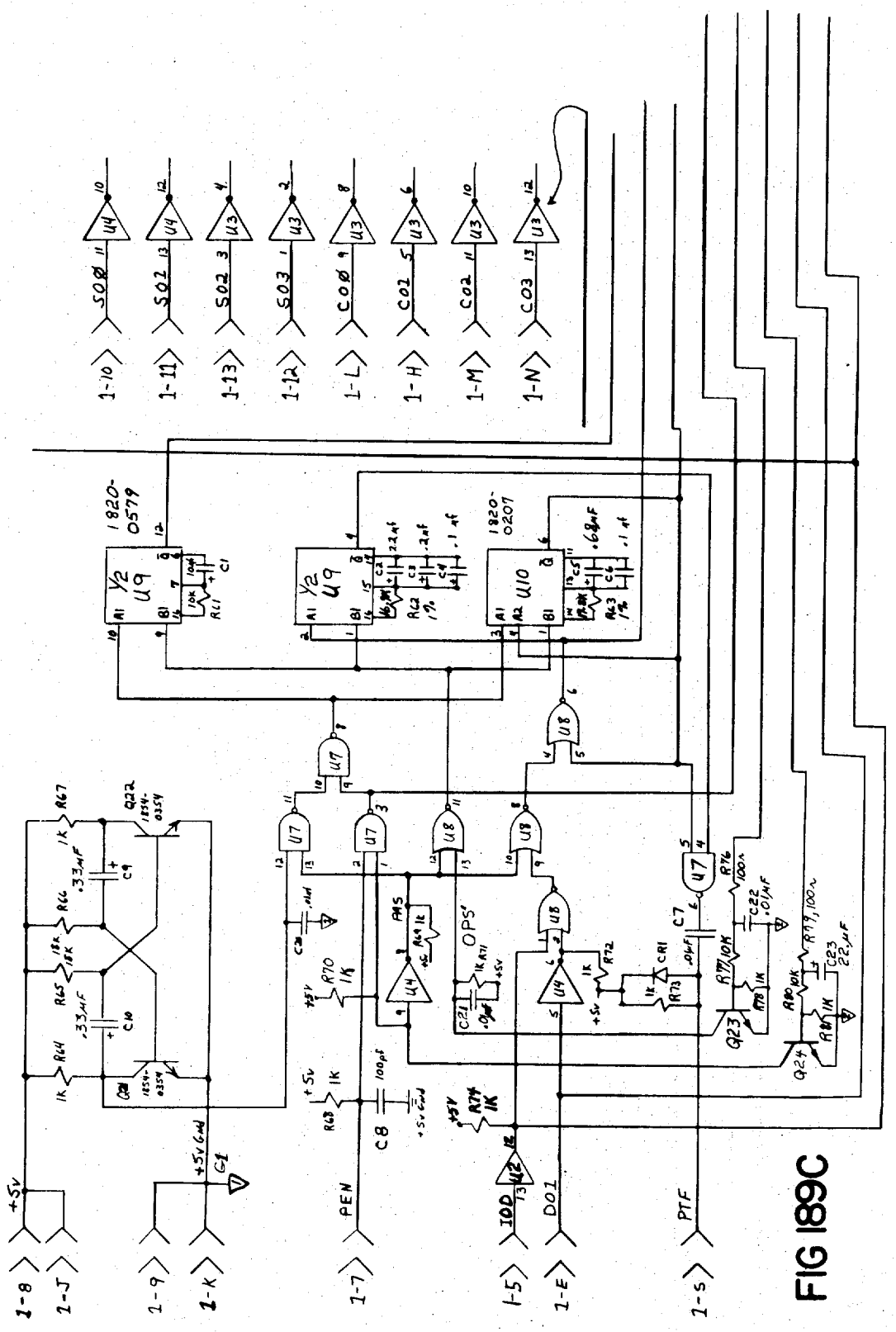

1-10
1-11
1-13
1-12
1-L
1-H
1-M
1-N
SO0
SO1
SO2
SO3
CO0
CO1
CO2
CO3
1820-0579
1820-0207
U9
U10
U7
U8
U4
U3
U2
Q22
Q23
Q24
+5v
+5v Gnd
G1
PEN
IOD
DO1
PTF
1-8
1-J
1-9
1-K
1-7
1-5
1-E
1-S

FIG 189C

R1 - R20 68 Ω ½W 5%
R21 - R40 10 KΩ ¼W 10%
R41 - R60 2.2KΩ ¼W 10%
All other Resistors ¼W 10%
Unless otherwise Noted
+24V
+5V
G2
2-3
2-C
2-6
2-F
2-7
2-H
2-8
2-J
MDI
MSC
OTS
PAI
OPS
PAS
OOI
IOD'
2-A
2-B
2-4
2-D
2-1
2-2
2-5
2-E
D13
D14
D15
D16
D17
D18
D19
D20
2-15
2-5
2-16
2-T
2-17
2-4
2-18
2-V

FIG 189D

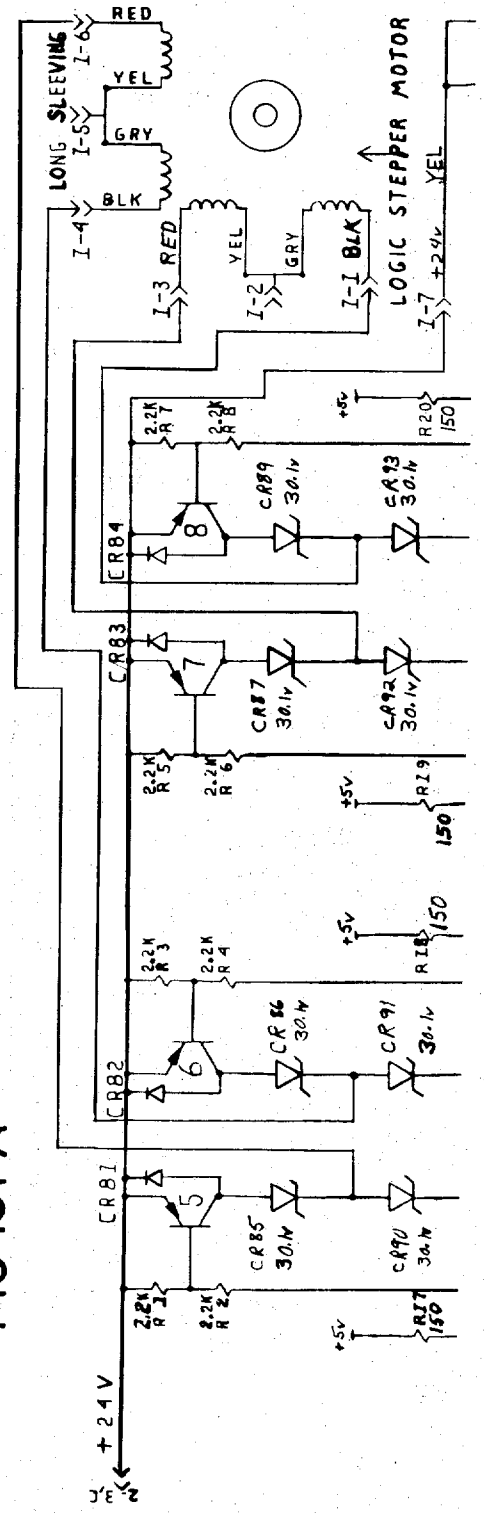
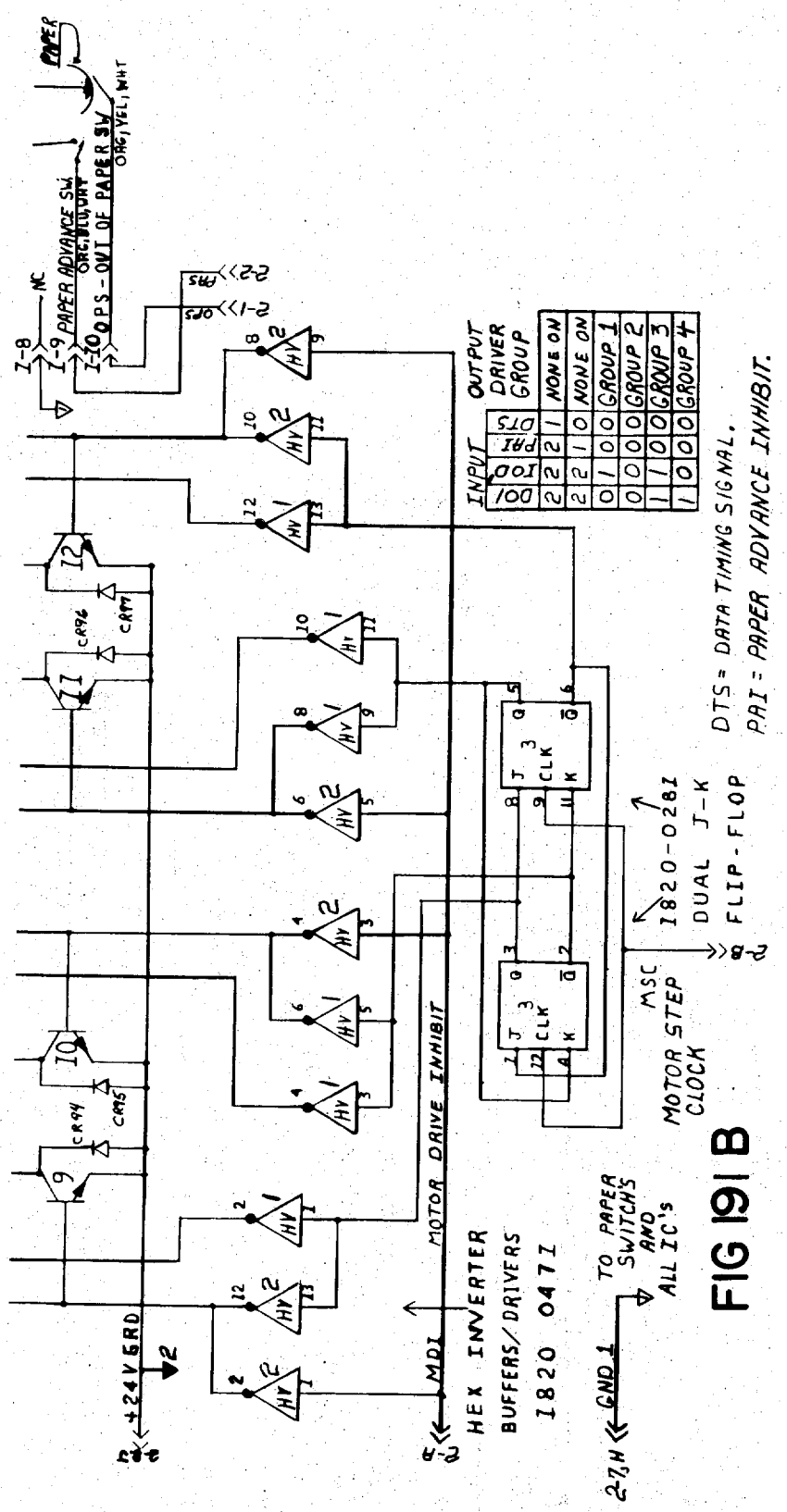

LONG SLEEVING
RED
YEL
GRY
BLK
I-6
I-5
I-4
I-3
I-2
I-1
I-7
LOGIC STEPPER MOTOR
+24V
CR81
CR82
CR83
CR84
CR85
CR86
CR87
CR89
CR90
CR91
CR92
CR93
30.1v
2.2K
R1
R2
R3
R4
R5
R6
R7
R8
R17
R18
R19
R20
150
+5V
+24V

FIG 191 A



INPUT DOI	INPUT IOD	INPUT PAI	INPUT DTS	OUTPUT DRIVER GROUP
2	2	2	1	NONE ON
2	2	1	0	NONE ON
0	1	0	0	GROUP 1
0	0	0	0	GROUP 2
1	1	0	0	GROUP 3
1	0	0	0	GROUP 4

FIG 191 B

Flow Diagram of Printing Operation

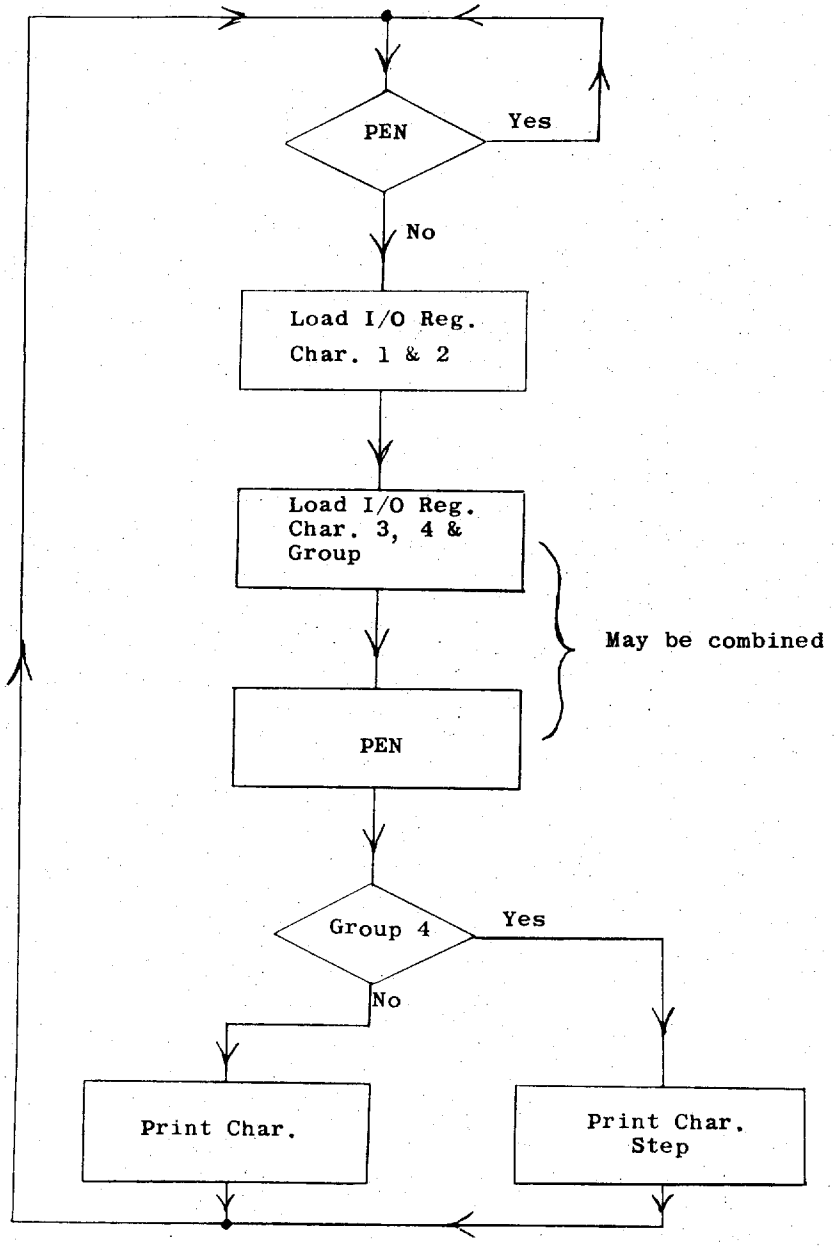
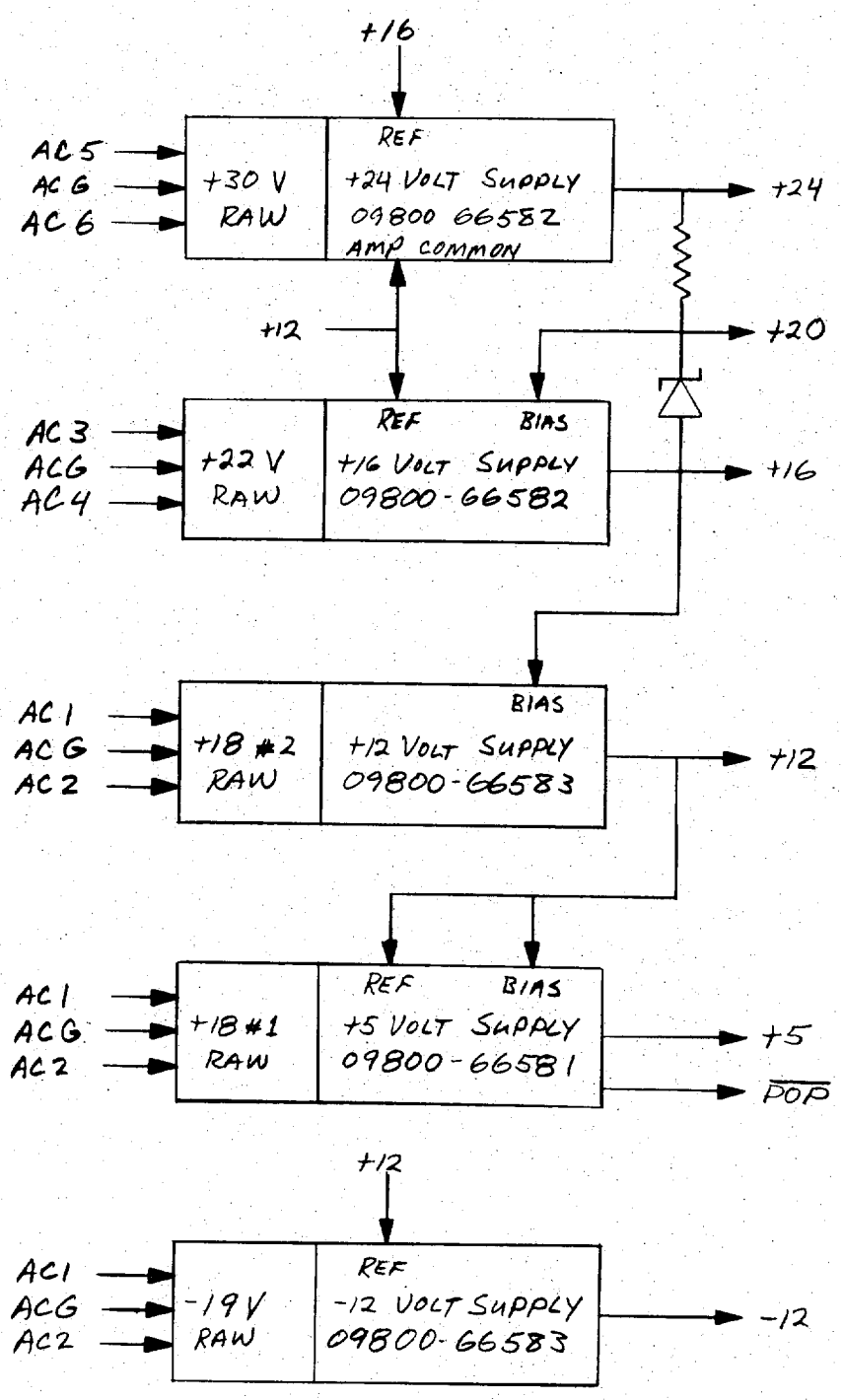


FIG 193



POWER SYSTEM BLOCK DIAGRAM

FIG 194

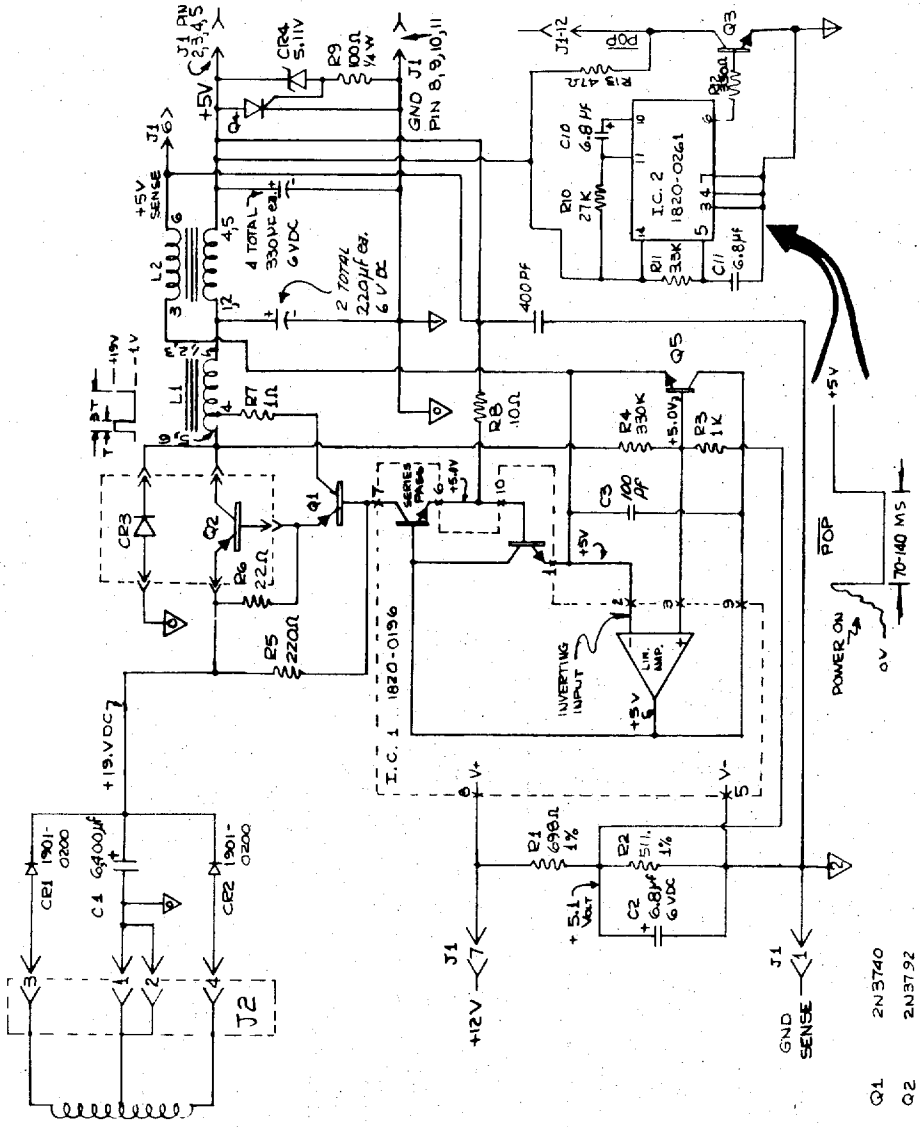

J1 PIN 2,3,4,5
CR4 5.11V
R9 100Ω 1/4W
GND J1 PIN 8,9,10,11
+5V
J1 6
+5V SENSE
L2
4,5
4 TOTAL 330μF ea. 6VDC
2 TOTAL 220μF ea. 6 VDC
400PF
J1-12
POP
Q3
R13 47Ω
R12 330Ω
C10 6.8μF
R10 27K
I.C. 2 1820-0261
R11 33K
C11 6.8μF
L1
R7 1Ω
R8 10Ω
Q5
R4 330K
+5.0V
R3 1K
C3 100 pF
CR3
Q2
Q1
R6 22Ω
R5 220Ω
SERIES PASS
+5.4V
I.C. 1 1820-0196
INVERTING INPUT
LIN. AMP.
+5V
POP
70-140 MS
POWER ON
0V
+19.VDC
CR1 1901-0200
C1 6,400μF
CR2 1901-0200
J2
R1 698Ω 1%
R2 511Ω 1%
C2 6.8μF 6VDC
+5.1 VOLT
J1 7
+12V
J1 1
GND SENSE
Q1 2N3740
Q2 2N3792
L1 180μh
L2 40μh

FIG 195

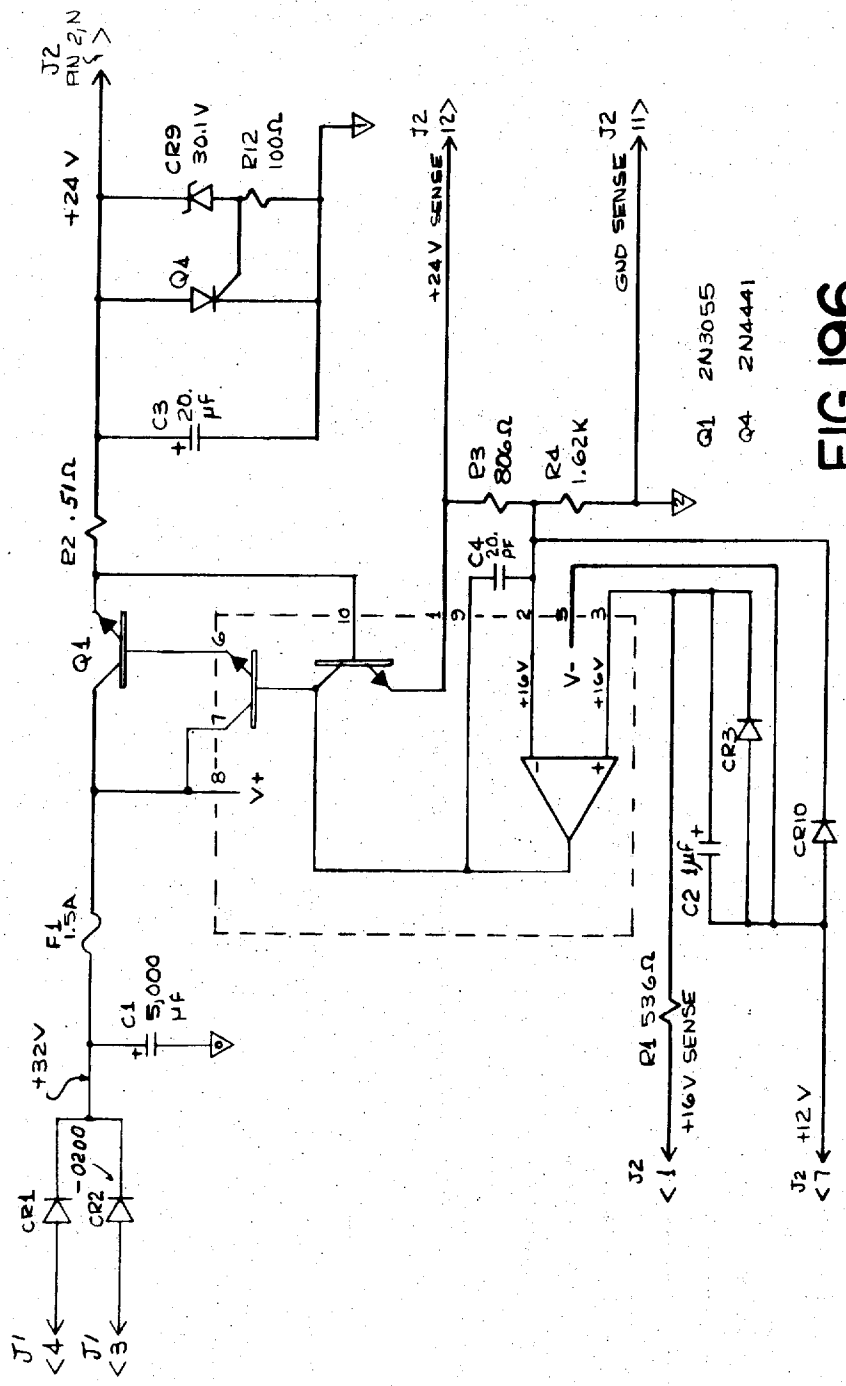

J2 PIN 2,N
+24 V
CR9 30.1 V
R12 100Ω
Q4
C3 20. µF
R2 .51Ω
Q1
F1 1.5A
C1 5,000 µF
+32V
CR1
CR2 -0200
J1 <4
J1 <3
J2 <12>
+24 V SENSE
J2 <11>
GND SENSE
R3 806Ω
R4 1.62K
C4 20. pF
+16V
V-
+16V
V+
CR3
C2 1µF
CR10
R1 536Ω
+16V SENSE
J2 <1
J2 +12 V
J2 <7
Q1 2N3055
Q4 2N4441

FIG 196

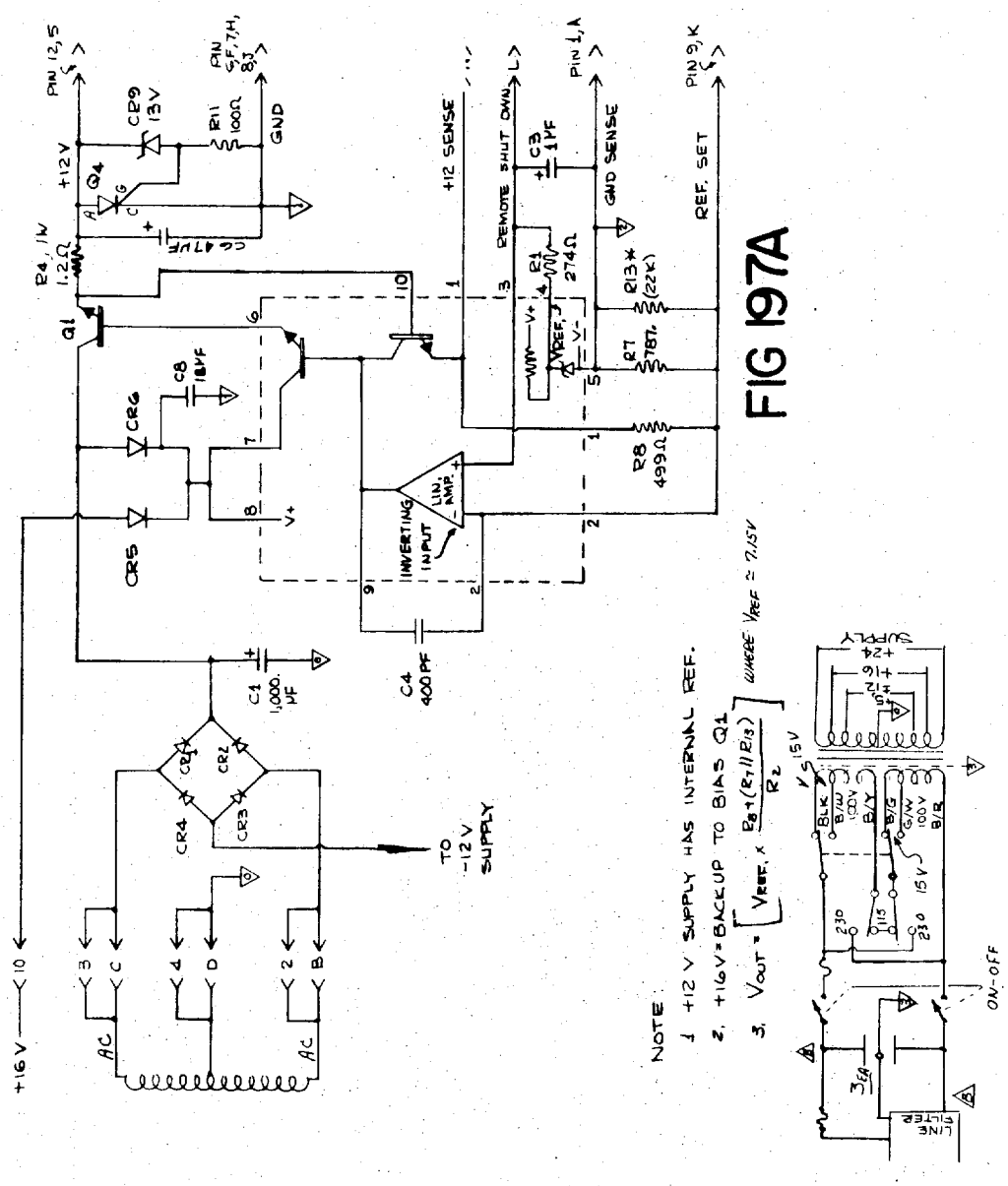

PIN 12,5
PIN 6,F,7,H, 8,J
+12V
CR9 13V
R11 100Ω
GND
Q4
R4, 1W 1.2Ω
C6 47μF
Q1
C8 18μF
CR6
CR5
+12 SENSE
REMOTE SHUT DWN
C3 1μF
PIN 1,A
GND SENSE
PIN 9,K
REF. SET
R1 274Ω
R13* (22K)
R7 787
R8 499Ω
INVERTING INPUT
LIN. AMP.
C4 400PF
C1 1,000 μF
CR1
CR2
CR3
CR4
TO -12V SUPPLY
+16V
AC
NOTE:
1. +12 V SUPPLY HAS INTERNAL REF.
2. +16V = BACKUP TO BIAS Q1
3. $V_{OUT} = \left[V_{REF.} \times \frac{R_8 + (R_7 \| R_{13})}{R_2} \right]$ WHERE $V_{REF} \approx 7.15V$
SUPPLY
+24
+16
+12
15V
BLK
B/W 100V
B/Y
B/G
G/W 100V
B/R
230
115
ON-OFF
LINE FILTER

FIG 197A

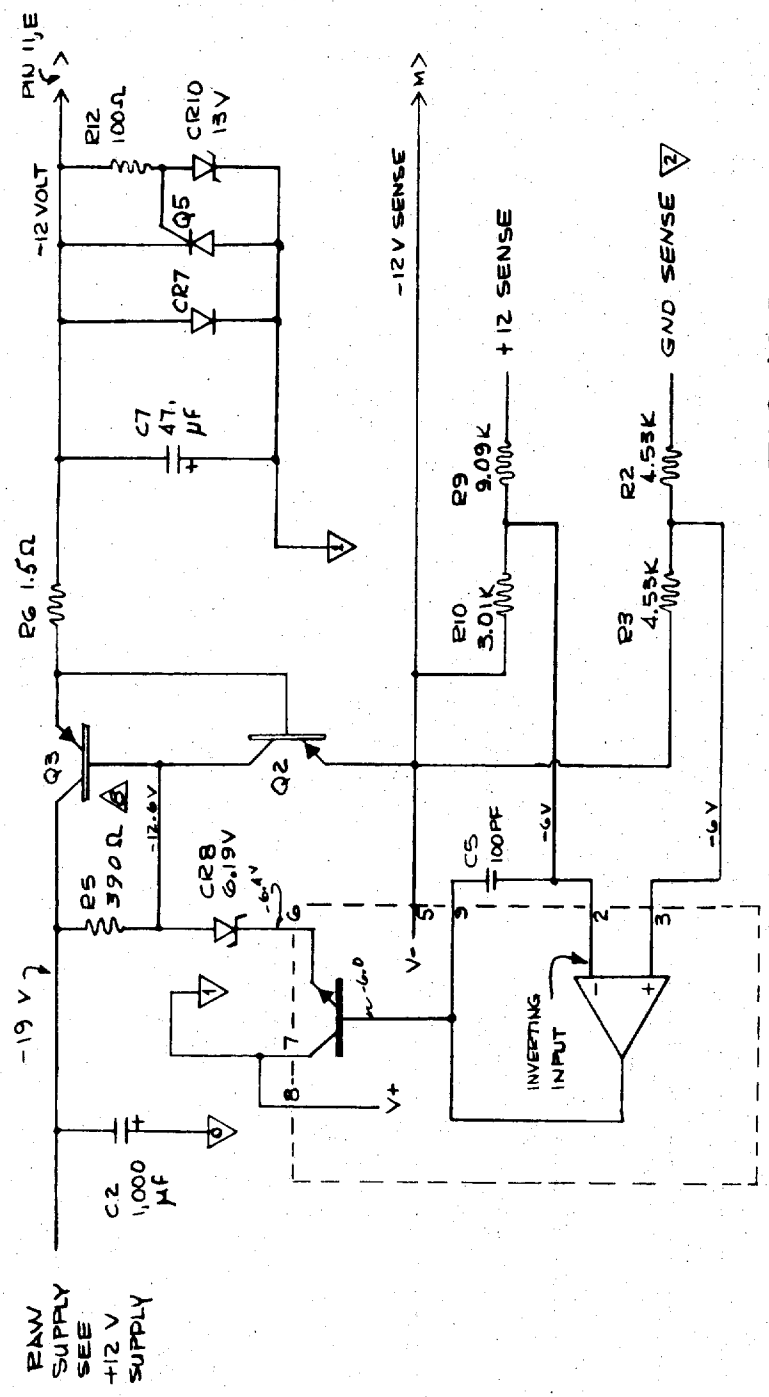

PIN 11,E
-12 VOLT
R12 100Ω
CR10 13V
Q5
CR7
C7 47, μF
R6 1.5Ω
Q3
Q2
R5 390Ω
CR8 6.19V
C2 1,000 μF
-19 V
RAW SUPPLY SEE +12 V SUPPLY
-12 V SENSE
+12 SENSE
GND SENSE
R9 9.09K
R10 3.01K
R2 4.53K
R3 4.53K
C5 100PF
-6V
V-
V+
INVERTING INPUT

FIG 197B

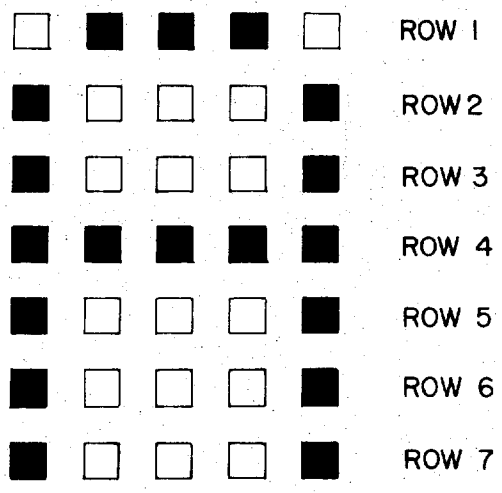


FIG 192

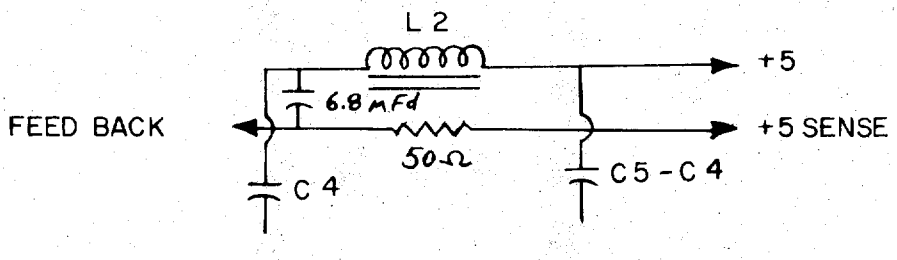


FIG 198

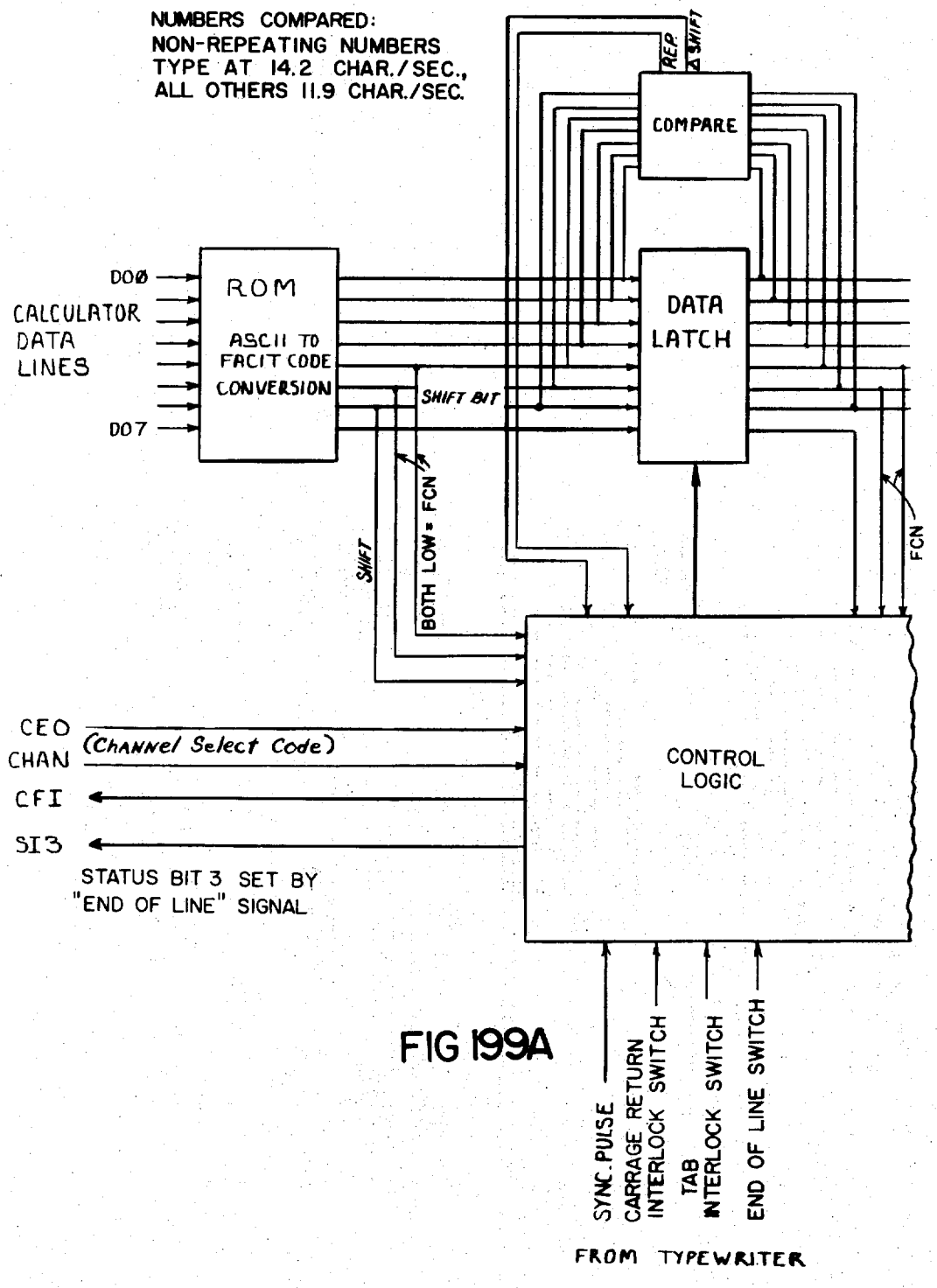

NUMBERS COMPARED:
NON-REPEATING NUMBERS
TYPE AT 14.2 CHAR./SEC.,
ALL OTHERS 11.9 CHAR./SEC.
REP
Δ SHIFT
COMPARE
DATA
LATCH
ROM
ASCII TO
FACIT CODE
CONVERSION
DOØ
CALCULATOR
DATA
LINES
DO7
SHIFT BIT
SHIFT
BOTH LOW = FCN
FCN
CEO
CHAN
(Channel Select Code)
CFI
SI3
STATUS BIT 3 SET BY
"END OF LINE" SIGNAL
CONTROL
LOGIC
SYNC. PULSE
CARRAGE RETURN
INTERLOCK SWITCH
TAB
INTERLOCK SWITCH
END OF LINE SWITCH
FROM TYPEWRITER

FIG 199A

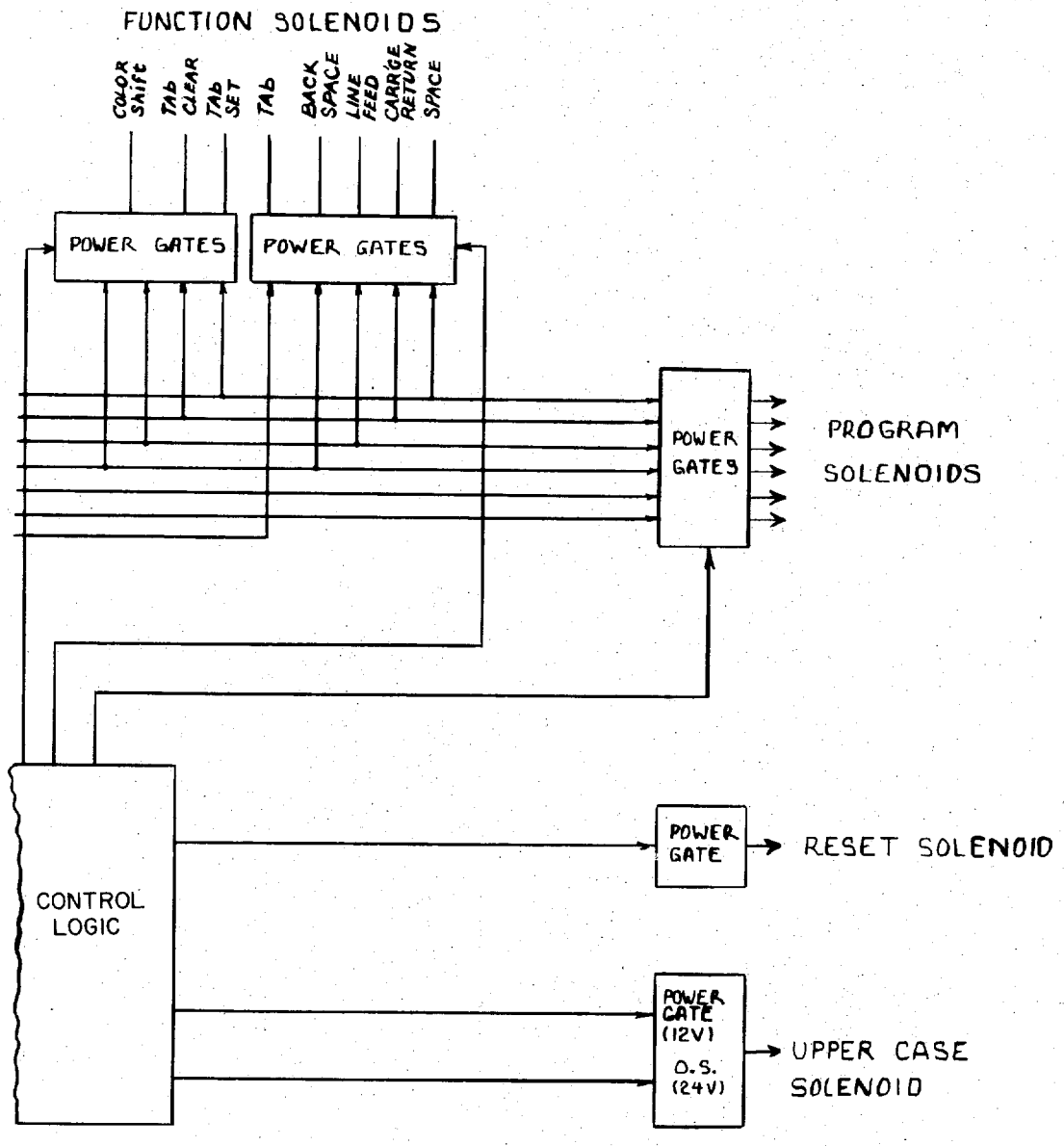


FIG 199B

POWER SUPPLIES: 5V, 24V

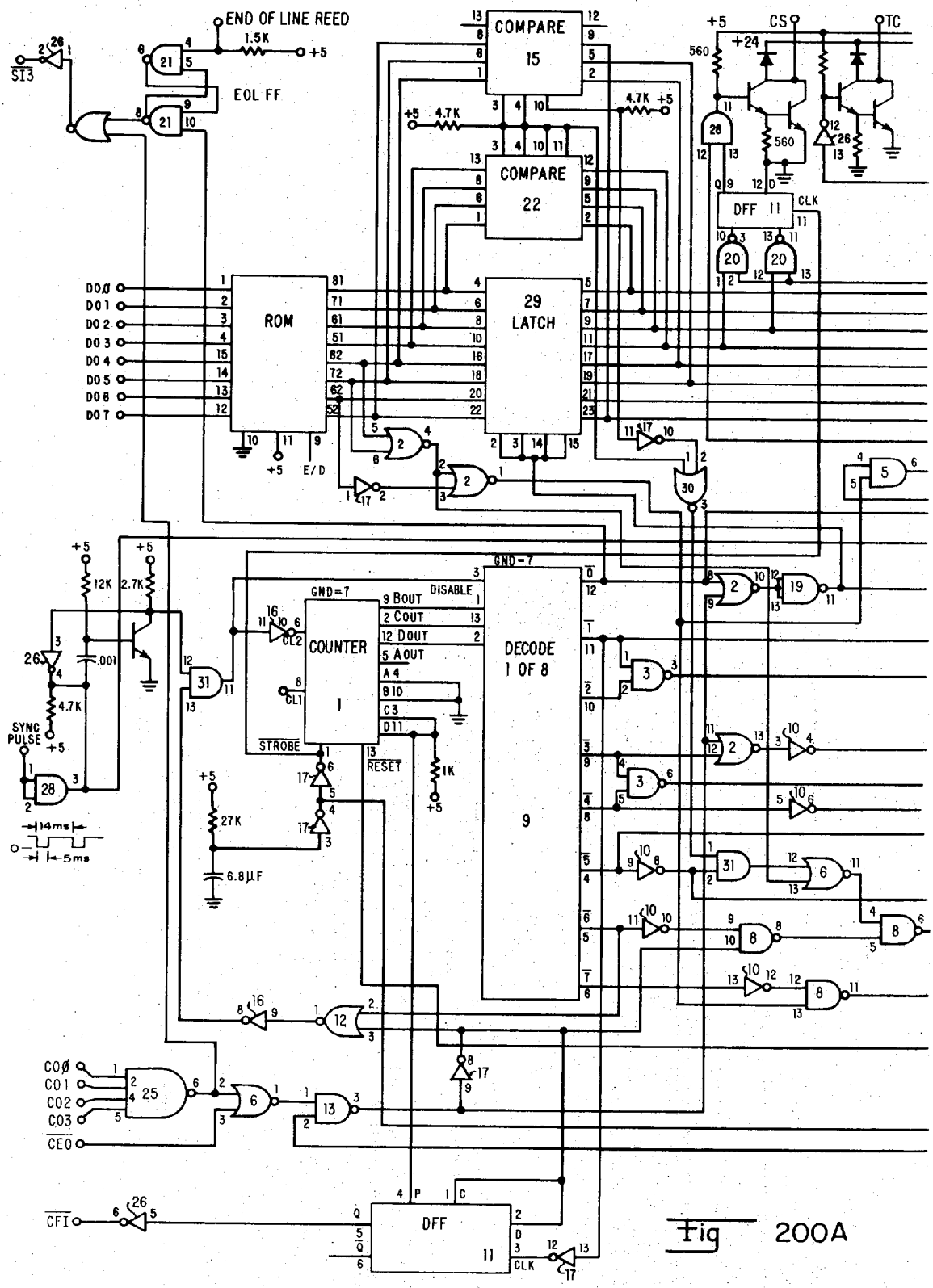

END OF LINE REED
1.5K
+5
EOL FF
SI3
COMPARE 15
COMPARE 22
4.7K
+5
+24
CS
TC
560
DFF 11
CLK
ROM
29 LATCH
DO0
DO1
DO2
DO3
DO4
DO5
DO6
DO7
E/D
30
GND=7
DISABLE
9 BOUT
2 COUT
12 DOUT
5 AOUT
A4
B10
C3
D11
COUNTER 1
CL2
CL1
STROBE
RESET
DECODE 1 OF 8
9
12K
2.7K
.001
4.7K
SYNC PULSE
14ms
5ms
27K
6.8µF
1K
CO0
CO1
CO2
CO3
CEO
CFI
DFF 11

Fig 200A

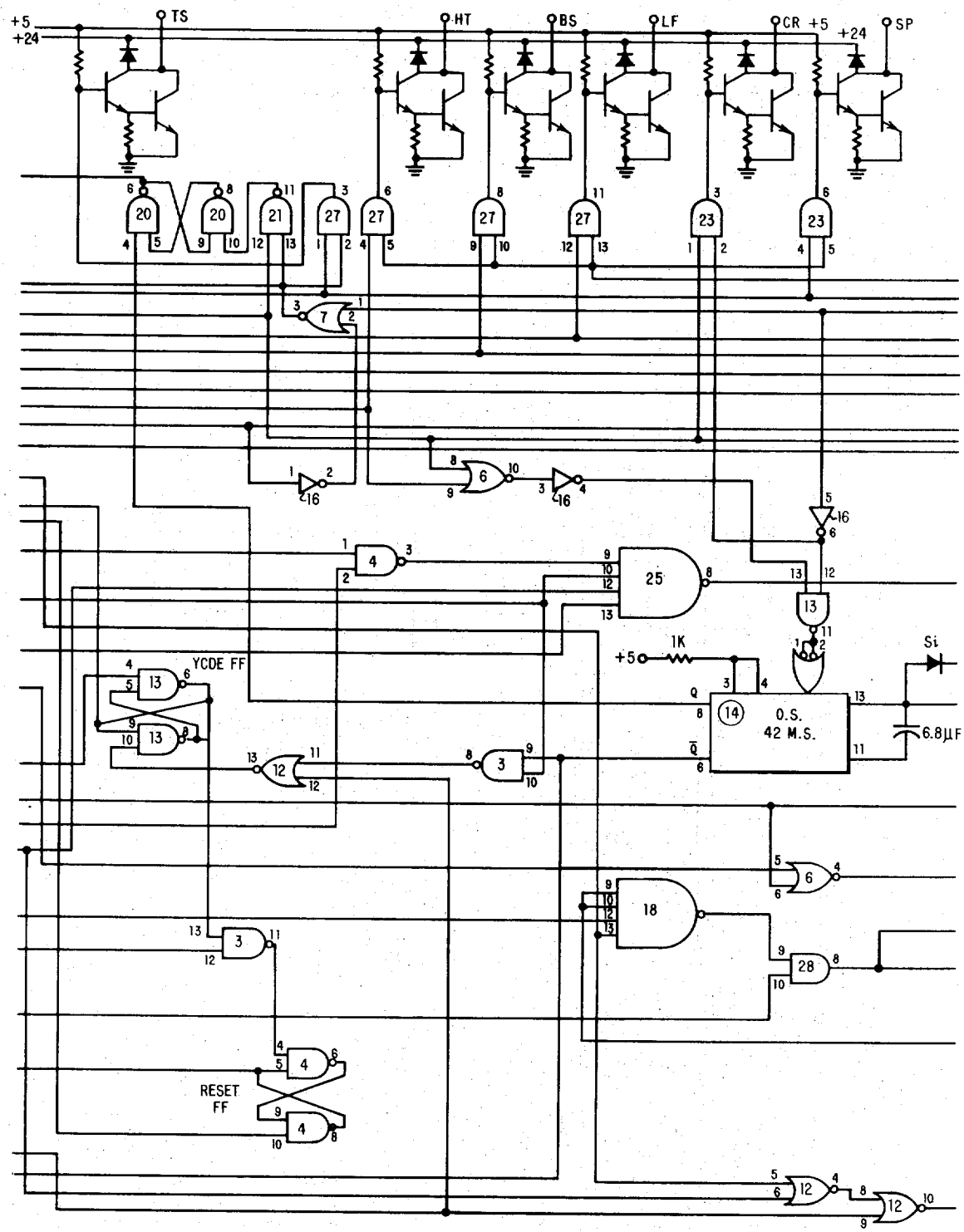

+5
+24
TS
HT
BS
LF
CR
+5
+24
SP
20
20
21
27
27
27
27
23
23
7
16
16
6
4
25
13
1K
+5
Si
YCDE FF
13
13
12
3
14
O.S.
42 M.S.
6.8µF
18
6
28
3
RESET
FF
4
4
12
12

Fig 200B

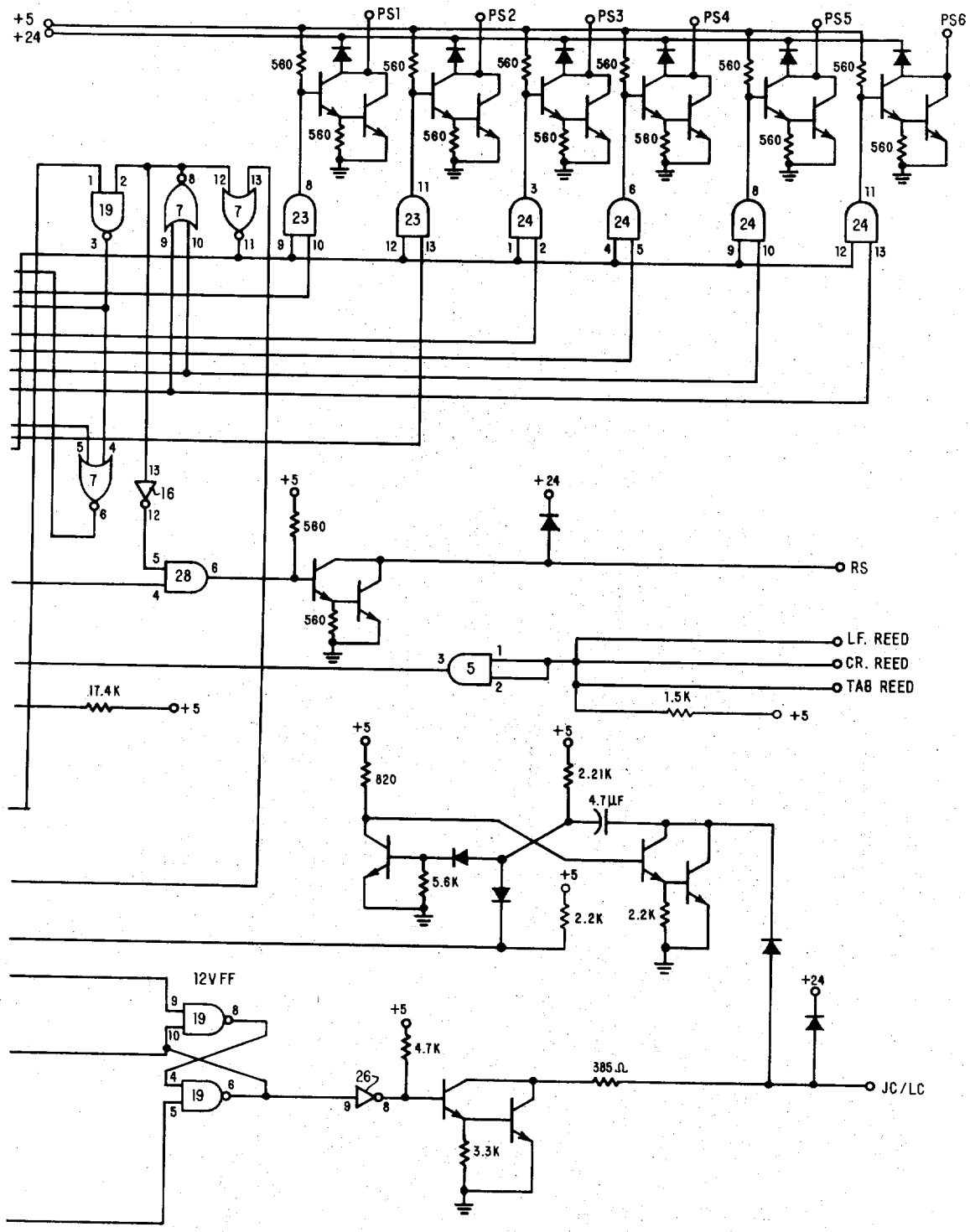

+5
+24
PS1
PS2
PS3
PS4
PS5
PS6
560
19
7
23
24
28
16
RS
LF. REED
CR. REED
TAB REED
17.4K
1.5K
820
2.21K
4.7μF
5.6K
2.2K
12V FF
4.7K
385Ω
26
3.3K
JC/LC

Fig 200 C

PROGRAMMABLE CALCULATOR

BACKGROUND OF THE INVENTION

This invention relates generally to calculators and improvements therein and more particularly to programmable calculators that may be controlled both manually from the keyboard input unit and automatically by a stored program loaded into the calculator from the keyboard input unit or an external record member.

Conventional programmable calculators generally have less capability and flexibility than is required to meet the needs of many users. For example, they typically cannot be readily expanded and adapted by the user to separately increase the amount of program and data storage memory or to perform special keyboard functions oriented toward the environment of the user. They also typically cannot perform indirectly addressed numeric data register transfers and arithmetic without utilizing available working registers for addresses rather than data. This seriously limits their ability to efficiently perform complex operations such as file manipulations or matrix arithmetic. Moreover, they typically have a very limited capability for performing direct arithmetic between working and storage memory registers and little or no capability for performing indirect arithmetic between working and storage memory registers.

In some conventional programmable calculators a program stored within the calculator can be recorded onto an external magnetic record member and can later be reloaded back into the calculator from the magnetic record member. However, data and programs stored within these calculators typically cannot be separately recorded onto an external magnetic record member and later separately reloaded back into the calculator therefrom. Moreover, these calculators have no provision for making a program secure when it is recorded onto an external magnetic record member. Any user may therefore re-record the program or obtain an indication of the individual program steps once the program is reloaded into the calculator.

Conventional programmable calculators with self-contained output printer units typically have a very limited alpha capability of only a few selected characters confined to certain columns of the printer. They are therefore typically unable to print out both a numeric and a distinct mnemonic representation of every program step of every program stored within the calculator. Furthermore, they are typically unable to print out labels for inputs to and outputs from the calculator or messages informing the user how to run programs with which he may be unfamiliar. Such features would be very helpful to the user both in editing programs and in simplifying their use.

In some conventional programmable calculators a program stored within the calculator may be edited by single stepping forward through the program while viewing an output display representing the last-encountered program step and its associated address and, in one case, also the presently-encountered program step and its associated address. However, these calculators typically cannot single step backward through the program or display the next program step to be encountered and its associated address. Moreover, they typically have no provision for inserting program steps into the program without reloading portions

of the program and no provision for finding every occurrence of any designated program step. Such features would also be very helpful to the user in editing programs.

Conventional computer systems have or may be programmed to have much more capability than conventional programmable calculators. However, they are larger, more expensive, and less efficient in calculating elementary mathematical functions than conventional programmable calculators. Moreover, a skilled programmer is typically required to utilize them. Due to these factors, conventional computer systems are best suited for handling large amounts of data or making highly iterative or very complex mathematical calculations.

SUMMARY OF THE INVENTION

The principal object of this invention is to provide an improved programmable calculator that has more capability and flexibility than conventional programmable calculators and that is smaller, less expensive, more efficient in calculating elementary mathematical functions, and easier to utilize than conventional computer systems.

Another object of this invention is to provide a programmable calculator in which the amount of program and data storage memory available to the user may be separately expanded and in which additional program and data storage memory made available to the user is automatically accommodated by the calculator and the user informed when he has exceeded the capacity of either the program or data storage memory.

Another object of this invention is to provide a programmable calculator in which the functions performed by the calculator may be readily expanded by the user and oriented toward the environment of the user and in which the added functions are automatically accommodated by the calculator.

Another object of this invention is to provide a programmable calculator in which the user may define keyboard functions to be performed by the calculator and may protect them from subsequently being inadvertently altered or destroyed.

Another object of this invention is to provide a programmable calculator capable of employing extensive indirect addressing to permit efficient manipulation of files and matrix operations.

Another object of this invention is to provide a programmable calculator in which the user may employ either absolute or symbolic addressing and in which program jumps or subroutine calls may be made to absolute or symbolic addresses.

Another object of this invention is to provide a programmable calculator capable of performing both direct and indirect storage, recall, and exchange between working and storage memory registers.

Another object of this invention is to provide a programmable calculator capable of performing both direct and indirect arithmetic bidirectionally between working and storage memory registers.

Another object of this invention is to provide a programmable calculator capable of performing serial binary arithmetic, parallel-by-digit binary-coded-decimal arithmetic, and logic operations.

Another object of this invention is to provide a programmable calculator in which either programs or data stored within the calculator may be separately re-

corded on an external magnetic record member and, subsequently, separately reloaded back into the calculator therefrom.

Another object of this invention is to provide a programmable calculator in which the user may designate any program stored within the calculator as being secure when it is recorded onto an external magnetic record member for subsequent re-entry into the calculator and in which the user is prevented from re-recording any secure program or obtaining any indication of its individual program steps once it is reloaded into the calculator.

Another object of this invention is to provide a programmable calculator capable of printing out every alphabetic and numeric character and many other symbols individually and in messages.

Another object of this invention is to provide a programmable calculator capable of printing out both a numeric and a mnemonic representation of each keyboard entry in a key-log listing.

Another object of this invention is to provide a programmable calculator capable of printing out a numeric representation of each numeric keyboard entry and calculated numeric result and a mark distinguishing each numeric keyboard entry from each calculated numeric result.

Another object of this invention is to provide a programmable calculator capable of printing out both a numeric and mnemonic representation of each step in a program stored within the calculator and a numeric indication of the address of each program step in a program listing.

Another object of this invention is to provide a programmable calculator capable of editing programs stored within the calculator more efficiently than conventional programmable calculators.

Another object of this invention is to provide a programmable calculator in which the user may single step either forward or backward through an internally stored program in a program entering mode to check the program and may single step forward through the program in a program-controlled operating mode to check the program execution.

Another object of this invention is to provide a programmable calculator in which the last program step encountered, the program step presently encountered, and the next program step to be encountered, while single stepping either forward or backward through a program stored within the calculator, may be displayed.

Another object of this invention is to provide a programmable calculator in which the user may insert program steps and automatically locate every occurrence of a designated program step in a program stored within the calculator.

Other and incidental objects of this invention will become apparent from a reading of this specification and an inspection of the accompanying drawings.

These objects are accomplished according to the illustrated preferred embodiment of this invention by employing a keyboard input unit, a magnetic card reading and recording unit, a solid state output display unit, an output printer unit, an input-output control unit, a memory unit and a central processing unit to provide an adaptable programmable calculator having manual operating, automatic operating, program entering, magnetic card reading, magnetic card recording, and alphameric printing modes. The keyboard input unit includes a group of data keys for entering numeric data into the calculator, a group of control keys for controlling the various modes and operations of the calculator and the format of the output display, and a group of definable keys for controlling additional functions that may be added by the user. All of the data keys and nearly all of the control keys may also be employed for programming the calculator, many of the control keys being provided solely for this purpose. The keyboard input unit also includes a group of indicator lights for informing the user of the status of the calculator. These indicator lights and all of the keys are mounted on a front panel of a housing for the calculator.

The magnetic card reading and recording unit includes a reading and recording head, a drive mechanism for driving a magnetic card from an input receptacle in the front panel of the calculator housing past the reading and recording head to an output receptacle in the front panel, and reading and recording drive circuits coupled to the reading and recording head for bi-directionally transferring information between the magnetic card and the calculator as determined by the control keys of the keyboard input unit. It also includes a pair of detectors and an associated control circuit for disabling the recording drive circuit whenever a notch is detected in the leading edge of the magnetic card to prevent information recorded on the magnetic card from being inadvertently destroyed. Such a notch may be provided in any magnetic card the user desires to protect by simply pushing out a perforated portion thereof.

The solid state output display unit includes three rows of light emitting diode arrays and associated drive circuits for selectively displaying three separate lines of numeric information. Numeric data may be displayed in either a fixed or a floating point format as determined by the control keys of the keyboard input unit.

The output printer unit includes a stationary thermal printing head with a row of resistive heating elements, a drive circuit for selectively energizing each heating element, and a stepping mechanism for driving a strip of thermal-sensitive recording paper past the stationary thermal printing head in seven steps for each line of alphameric information to be printed out. Every alphabetic and numeric character and many other symbols may be printed out individually or in messages as determined by the control keys of the keyboard input unit or by a program stored within the calculator.

The input-output unit includes a 16-bit universal shift register serving as an input-output register into which information may be transferred serially from the central processing unit or in parallel from the keyboard input and magnetic card reading and recording units and from which information may be transferred serially to the central processing unit or in parallel to the keyboard indicator lights and to the solid state output display, magnetic card reading and recording, and output printer units. It also includes control logic responsive to the central processing unit for controlling the transfer of information between these units. The input-output control unit may also be employed to perform the same functions between the central processing unit and peripheral units including, for example, a digitizer, a marked card reader, an X-Y plotter, a magnetic tape unit, and a typewriter. A plurality of peripheral units may be connected at the same time to the input-output control unit by simply plugging interface modules asso-

ciated with the selected peripheral units into receptacles provided therefore in a rear panel of the calculator housing.

The memory unit may employ both direct and indirect decimal and symbolic addressing. It includes a modular random-access read-write memory having a program storage section for storing a plurality of program steps and having a separate data storage section including a plurality of working registers, a plurality of associated display registers, and a plurality of storage registers for manipulating and storing data. These program and data storage sections of the read-write memory may be separately expanded without increasing the overall dimensions of the calculator by the addition of program storage modules or by the alteration of the data storage memory control. Additional read-write memory made available to the user is automatically accommodated by the calculator, and the user is automatically informed when the program or data storage capacity of the read-write memory has been exceeded.

The memory unit also includes a modular read-only memory in which routines and subroutines of basic instructions for performing the various functions of the calculator are stored. These routines and subroutines of the read-only memory may be expanded and adapted by the user to perform additional functions oriented toward the specific needs of the user. This is accomplished by simply plugging additional read-only memory modules into receptacles provided therefor in the top panel of the calculator housing. Added read-only memory modules are automatically accommodated by the calculator and may be associated with the definable keys of the keyboard input unit or employed to expand the operations associated with other keys. An overlay is employed with each added read-only memory module associated with the definable keys of the keyboard input unit to identify the additional functions that may then be performed by the calculator.

Plug-in read-only memory modules including, for example, an alpha module, a mathematics module, a statistics module, a definable functions module, and a typewriter module may be added to the read-only memory. The alpha module enables the calculator to print out every alphabetic character individually or in messages. It employs addressing enabling it to redefine most of the keys of the keyboard input unit so that it may be employed at the same time as other plug-in read-only memory modules. The mathematics module enables the calculator to perform trigonometric functions, coordinate transformations, vector arithmetic, and many other mathematical functions. Similarly, the statistics module enables the calculator to perform random number generations, accumulations of sums, sums of products and sums of squares for up to five variables, linear and multiple linear regressions, and many other statistical functions. It also permits the use of a correct key, included among the definable keys of the keyboard input unit, to automatically delete data from a statistical analysis. The definable functions module enables the user to store programs of his own choosing in the program storage section of the read-write memory, associate them with some of the definable keys of the keyboard input unit, and protect them from subsequently being inadvertently altered or destroyed. It also permits the use of an insert key and a find key, included among the definable keys of the keyboard input unit, to insert program steps in a program stored in the read-write memory and to find evey occurrence of any designated program step in the stored program. The typewriter module enables the calculator to control the entire keyboard of a properly interfaced typewriter.

The memory unit further includes a pair of recirculating 16-bit serial shift registers. One of these registers serves as a memory address register for serially receiving information from an arithmetic-logic unit included in the central processing unit, for parallel addressing any memory location designated by the received information, and for serially transferring the received information back to the arithmetic-logic unit. The other of these registers serves as a memory access register for serially receiving information from the arithmetic-logic unit, for writing information in parallel into any addressed memory location, for reading information in parallel from any addressed memory location, and for serially transferring information to the arithmetic-logic unit. It also serves as a four-bit parallel shift register for transferring four bits of binary-coded-decimal information in parallel to the arithmetic-logic unit.

The central processing unit includes four recirculating 16-bit serial shift registers, a four-bit serial shift register, the arithmetic logic unit, a programmable clock, and a microprocessor. Two of these 16-bit serial shift registers serve as accumulator registers for serially receiving information from and serially transferring information to the arithmetic-logic unit. The accumulator register employed is designated by a control flip-flop. One of the accumulator registers also serves as a four-bit parallel shift register for receiving four bits of binary-coded-decimal information in parallel from and transferring four bits of such information in parallel to the arithmetic-logic unit. The two remaining 16-bit serial shift registers serve as a program counter register and a qualifier register, respectively. They are also employed for serially receiving information from and serially transferring information to the arithmetic-logic unit. The four-bit serial shift register serves as an extend register for serially receiving information from either the memory access register or the arithmetic-logic unit and for serially transferring information to the arithmetic-logic unit.

The arithmetic-logic unit is employed for performing one-bit serial binary arithmetic, four-bit parallel binary-coded-decimal arithmetic, and logic operations. It may also be controlled by the microprocessor to perform bi-directional direct and indirect arithmetic between any of a plurality of the working registers and any of the storage registers of the data storage section of the read-write memory.

The programmable clock is employed to supply a variable number of shift clock pulses to the arithmetic-logic unit and to the serial shift registers of the input-output, memory, and central processing units. It is also employed to supply clock control signals to the input-output control logic and to the microprocessor.

The microprocessor includes a read-only memory in which a plurality of microinstructions and codes are stored. These microinstructions and codes are employed to perform the basic instructions of the calculator. They include a plurality of coded and non-coded microinstructions for transferring control to the input-output control logic, for controlling the addressing and accessing of the memory unit, and for controlling the operation of the two accumulator registers, the program counter register, the extend register and the arith-

metic-logic unit. They also include a plurality of clock codes for controlling the operation of the programmable clock, a plurality of qualifier selection codes for selecting qualifiers and serving as primary address codes for addressing the read-only memory of the microprocessor, and a plurality of secondary address codes for addressing the read-only memory of the microprocessor. In response to a control signal from a power supply provided for the calculator, control signals for the programmable clock, and qualifier-control signals from the central-processing and input-output control units, the microprocessor issues the microinstructions and codes stored in the read-only memory of the microprocessor as required to process either binary or binary-coded-decimal information entered into or stored in the calculator.

In the manual operating mode, the calculator is controlled by keycodes sequentially entered into the calculator from the keyboard input unit by the user. The solid state output display unit displays a numeric representation of the contents of three of the working registers and their associated display registers. These working registers and their associated display registers may contain the last-entered numeric operand and two previously entered or calculated numeric operands or results or three previously entered or calculated numeric operands or results. The output printer unit may be controlled by the user to selectively print out a numeric representation of any numeric data entered into the calculator from the keyboard input unit, a numeric representation of any result calculated by the calculator, and a mark distinguishing numeric data entries from calculated numeric results. If the alpha read-only memory module is plugged into the calculator, the output printer unit may also be controlled by the user to print out labels for inputs to and outputs from the calculator and any other alphabetic information that may be desired.

When the calculator is in the manual operating mode, it may also be operated in a key-log alphameric printing mode. The output printer unit then prints out a numeric representation of each keycode as it is entered by the user. If the alpha read-only memory module is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each such keycode.

In the automatic operating mode, the calculator is controlled by automatically obtaining keycodes stored as steps of a program in the program storage section of the read-write memory. During automatic operation of the calculator, data may be obtained from the memory unit as designated by the program or may be entered from the keyboard input unit by the user while the operation of the calculator is stopped for data either by the program or by the user. The solid state output display unit displays the final contents of the three registers and their associated display registers. This may include the final calculated numeric result and two previously entered or calculated numeric operands or results or three previously entered or calculated numeric operands or results. The output printer unit prints out calculated numeric results and other numeric information designated by the program. If the alpha read-only memory module is plugged into the calculator, the output printer unit also prints out any alphabetic information designated by the program.

When the calculator is in the automatic operating mode, the user may also employ a step program control key of the keyboard input unit to single step forward through the program being executed. This enables the user to check the execution of the program step by step in order to determine whether the program, as entered into the calculator, does in fact carry out the desired sequence of operations.

In the program entering mode, keycodes are sequentially entered by the user into the calculator from the keyboard input unit and are stored as steps of a program in the program storage section of the read-write memory. The program may include sequences of program steps that will be intrepreted, when the program is executed, as alphabetic information to be printed out by the output printer unit if the alpha read-only memory module is plugged into the calculator. This alphabetic information may include labels for inputs to and outputs from the calculator, alphabetic messages for facilitating the use of the program and the operation of the calculator, or any other alphabetic information that may be desired. While the user is entering a program into the calculator in the program entering mode, the solid state output display unit displays a numeric representation of the last-entered program step and its associated address and the addresses of the next two program steps to be entered and the present contents of those addresses.

In the program entering mode, the user may also employ the step program control key and a back step control key of the keyboard input unit, to single step either forward or backward through any sequence of program steps stored in the program storage section of the read-write memory. While the user is single stepping forward or backward through a sequence of program steps, the solid state output display unit displays a numeric representation of the last-encountered program step, the program step presently encountered, the next program step to be encountered, and the addresses of these program steps. If the definable functions read-only memory module is plugged into the calculator, the user may also employ the insert and find keys described above by switching to the manual operating mode. These features greatly facilitate the editing of programs stored in the program storage section of the read-write memory.

When the calculator is in the program entering mode, it may also be operated in a key-log alphameric printing mode. The output printer unit then prints out a numeric representation of each program step and its associated address as it is entered into the calculator from the keyboard input unit by the user. If the read-only memory module is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each such program step.

When the calculator is in the program entering mode, it may also be operated in a list alphameric printing mode. The output printer unit then prints out a numeric representation of every program step then stored in the program storage section of the read-write memory and a numeric representation of the addresses of those program steps. If the alpha read-only memory module is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each such program step.

In the magnetic card reading mode, the magnetic card reading and recording unit may be employed by the user to separately load either data or programs into

the calculator from one or more external magnetic cards. Data and programs so loaded are separately stored in the data storage and program storage sections of the read-write memory.

In the magnetic card recording mode, the magnetic card reading and recording unit may be employed by the user to separately record either data or programs, separately stored in the data storage and program storage sections of the read-write memory, onto one or more external magnetic cards. Programs stored in the program storage section of the read-write memory may be coded by the user as being secure when they are recorded onto one or more external magnetic cards. The calculator detects such programs when they are reloaded into the calculator and prevents the user from re-recording them or obtaining any listing or other indication of the individual program steps.

DESCRIPTION OF THE DRAWINGS

FIG. **1** is a front perspective view of an adaptable programmable calculator according to the preferred embodiment of this invention.

FIG. **2** is a rear perspective view of the adaptable programmable calculator of FIG. **1.**

FIGS. **3A–B** are simplified block diagrams of the adaptable programmable calculator of FIGS. **1** and **2.**

FIG. **4** is a memory map of the memory unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B.**

FIG. **5** is a detailed memory map of the dedicated portion of the data storage section of the read-write memory employed in the memory unit of FIGS. **3A–B** and **4.**

FIG. **6** is a detailed memory map of the dedicated portion of the program storage section of the read-write memory employed in the memory unit of FIGS. **3A–B** and **4.**

FIGS. **7A–C** are simplified operational logic flow charts illustrating the operation of the microprocessor employed in the central processing unit of FIGS. **3A–B.**

FIG. **7'** is a figure map showing how the operational flow charts of FIGS. **7A–C** fit together.

FIG. **8** is a plan view of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B** showing how the keyboard input unit may be redefined by an alpha plug-in read-only memory module that may also be employed in the adaptable programmable calculator.

FIG. **9** is a plan view of the definable keys of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B** and the overlay associated with a definable functions plug-in read-only memory module that may be employed in the adaptable programmable calculator.

FIG. **10** is a plan view of the definable keys of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B** and the overlay associated with a mathematics plug-in read-only memory module that may be employed in the adaptable programmable calculator.

FIG. **11** is a plan view of the definable keys of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B** and the overlay associated with a statistics plug-in read-only memory module that may be employed in the adaptable programmable calculator.

FIG. **12** is a plan view of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B** showing how the keyboard input unit may be redefined by a typewriter plug-in read-only memory module that may also be employed in the adaptable programmable calculator.

FIG. **13** is a simplified flow chart of the overall control sequence employed for keycode processing in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B.**

FIG. **14** is a flow chart of the display routine of FIG. **13.**

FIGS. **15** A–C are flow charts of the display list building routine of FIG. **14.**

FIG. **16** is a map of the output bit assignment for The seven-segment display employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B.**

FIG. **17** is a map of the display list built in the read-write memory of FIGS. **3A–B, 4,** and **5.**

FIG. **18** is a map of the internal structure of the display list of FIG. **17.**

FIGS. **19A–B** are flow charts of the monitor routine employed in the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B.**

FIGS. **20A–B** are flow charts of the interpreter routine of FIG. **13.**

FIGS. **21** A–D are flow charts of the program mode key, run mode key, floating-point display key, and fixed-point display key processing routines, respectively, of FIG. **13.**

FIG. **22** is a map of the status word register of FIG. **5** illustrating how it is employed by some of the routines of FIGS. **21A–D.**

FIG. **23** is a flow chart of the back step key processing routine of FIG. **13.**

FIG. **24** is a flow chart of the clear key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **25** is a flow chart of the end key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **26** is a flow chart of the stop key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **27** is a flow chart of the integer *x* key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **28** is a flow chart of the pi key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **29** dis a flow chart of the clear *x* key processing routine selectable by the interpreter of FIG. **13.**

FIGS. **30A–B** are simplified flow charts of the number entry processing routine selectable by the interpreter of FIG. **13.**

FIGS. **31A–C** are flow charts of portions of the magnitude processing portions of the number entry routine of FIGS. **30A–B.**

FIGS. **32A–B** are flow charts of exponent processing portions of the number entry routine of FIGS. **30A–B.**

FIG. **33** is a flow chart of the change-sign processing portion of the number entry routine of FIGS. **30A–B.**

FIG. **34** is a flow chart of the termination portion of the number entry routine of FIGS. **30A–B.**

FIG. **35** is a flow chart of the up-key processing routine selectable by the interpreter of FIGS. **13.**

FIG. **36** is a flow chart of the down-key processing routine selectable by the interpreter routine of FIG. **13.**

FIG. **37** is a flow chart of the roll-up key processing routine selectable by the interpreter routine of FIG. **13.**

FIGS. 38A–B are simplified flow charts of the register transfer and register arithmetic routines selectable by the interpreter routine of FIG. 13.

FIGS. 39A–B are flow charts of the register transfer and automatic address termination routine employed in connection with the register transfer arithmetic key processing routine of FIGS. 38A–B.

FIG. 40 is a flow chart of the indirect-key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 41 is a flow chart of the *a*- and *b*-register transfer key processing routines selectable by the interpreter routine of FIG. 13.

FIG. 42 is a flow chart of the indirect address computation routine employed in connection with the indirect-key processing routine of FIG. 40.

FIG. 43 is a flow chart of the decimal address jump and automatic address termination routine employed in connection with the go-to and if key processing routines selectable by the interpreter routine of FIG. 13.

FIG. 44 is a flow chart of the label-key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 45 is a flow chart of the if key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 46 is a flow chart of the subroutine/return key processing routine selectable by the interpreter routine of FIG. 13.

FIGS. 47A–C are flow charts of the floating point add and subtract key processing routines selectable by the interpreter routine of FIG. 13.

FIG. 48 is a flow chart of the floating point multiply key processing routine selectable by the interpreter routine of FIG. 13.

FIGS. 49A–B are flow charts of the floating point division key processing routine of FIG. 13.

FIGS. 50A–C are flow charts of the floating point square root key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 51 is a flow chart of the store routine of FIG. 13.

FIG. 52 is a flow chart of the rounding routine employed in connection with several of the routines selectable by the interpreter routine of FIG. 13.

FIG. 53 is a flow chart of the *x*-squared key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 54 is a flow chart of the reciprocal-*x* key processing routine selectable by the interpreter routine of FIG. 13.

FIGS. 55A–B are flow charts of the basic format key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 56 is a flow chart of a search routine that may be employed in connection with the basic format key processing routine of FIGS. 55–B.

FIG. 57 is a memory map illustrating the use of the search routine of FIG. 56.

FIGS. 58A–B are flow charts of a plotter routine employed in connection with the basic format key processing routine of FIGS. 55A–B.

FIG. 59 is a map of a memory register employed in connection with the plotter routine of FIGS. 58A–B.

FIG. 60 is a flow chart of the program-step recording portion of the record key processing routine of FIG. 13.

FIG. 61 is a flow chart of the program-step loading portion of the load key processing routine of FIG. 13.

FIG. 62 is a memory map of the program storage section of the read-write memory of FIGS. 3A–B and 4–6 illustrating how a secure program is destroyed by an unsecure program.

FIG. 63 is a flow chart of the format recording portion of the record key processing routine of FIG. 13.

FIG. 64 is a flow chart of the data recording portion of the record key processing routione of FIG. 13.

FIG. 65 is a flow chart of the data loading portion of the load key processing routine of FIG. 13.

FIG. 66 is a flow chart showing how a plug-in read-only memory module employed in the adaptable programmable calculator of FIGS. 1, 2, and 3A–B can obtain keycodes from a program stored in the program storage section of the read-write memory of FIGS. 3A–B and 4.

FIGS. 67A–C show how the print routine accesses code information stored in tables in the data storage section of the read-write memory of FIGS. 3A–B and 4.

FIGS. 68A–B are memory maps of a directive table and a code table respectively, in an alpha plug-in read-only memory module that may be employed in the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 69 is a flow chart showing how the printer code is accessed by an alpha plug-in read-only memory module employed in the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIGS. 70A–B are flow charts of the typewriter format routine employed in connection with the format key processing routine selectable by the interpreter routine of FIG. 13.

FIG. 71 is a memory map of a typewriter mnemonic table in a typewriter plug-in read-only memory module that may be employed in the adaptable programmable calculator of FIGS. 1, 2, and 1, 2, and B.

FIG. 72 is a flow chart of a typewriter list routine that may be performed when the typewriter read-only memory module is plugged into the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 73 is a flow chart of a typewriter print routine that may be performed when the typewriter read-only memory module is plugged into the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 74 is a flow chart of a typewriter alpha routine that may be performed when the typewriter read-only memory module is plugged into the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 75 is a flow chart of a definable keycode entry routine employed when the definable functions read-only memory module is plugged into the calculator.

FIG. 76 is a flow chart of a protect key processing routine that may be performed when the definable functions read-only memory module is plugged into the calculator.

FIG. 77 is a flow chart of a memory search routine that may be performed when the definable functions read-only memory module is plugged into the calculator.

FIGS. 78A–B are flow charts of a definable function calls routine that may be performed when the definable functions read-only memory module is plugged into the calculator.

FIG. 79 is a directive table of the function calls that may be performed by the definable function calls routine of FIGS. 78A–B.

FIG. **80** is a flow chart of a definable function return routine employed with the definable function calls routine of FIGS. **78A–B**.

FIG. **81** is a flow chart of a turn-off-the-function light subroutine employed by the definable function return routine of FIG. **80**.

FIG. **82** is a flow chart of a function stack checking subroutine employed by the definable function calls routine and the definable function return routine of FIGS. **78A–B** and **80**, respectively.

FIG. **83** is a flow chart of a delete-protect routine that may be performed when the definable functions read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **84** is a flow chart of a turn-off-the-delete-light subroutine employed by the delete-protect routine of FIG. **83**.

FIG. **85** is a flow chart of an empty-the-function-stack subroutine employed by the delete-protect routine of FIG. **83**.

FIG. **86** is a flow chart of a find routine that may be performed when the definable functions read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **87** is a flow chart of a get-the-next-key subroutoine employed by the find routine of FIG. **86**.

FIG. **88** is a flow chart of a memory compactor routine that may be performed when the definable functions read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1,2,**and **3A–B**

FIG. **89** is a memory map illustrating the use of the memory compactor subroutine of FIGS. **88**.

FIGS. **90A–C** are flow charts of a delete routine that may be performed when the definable functions read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIGS. **91A–B** are flow charts of an insert routine that may be performed when the definable functions read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **92** is a flow chart of an address calculating subroutine employed by the insert routine of FIGS. **91A–B**.

FIGS. **93A–B** are flow charts of a tangent *x* routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIGS. **94A–B** are flow charts of an arctangent *x* routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIGS. **95A–B** are flow charts of an *e*-to-the-*x*-power that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **96** is a flow chart of a natural logarithm *x* routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **97** is a flow chart of a subroutine employed by the tangent *x* and the *e*-to-the-*x*-power routines of FIGS. **93A–B** and **95A–B**, respectively.

FIG. **98A–B** are flow charts of a subroutine employed by the tangent *x* and arctangent *x* routines of FIGS. **93A–B** and **94A–B**, respectively.

FIGS. **99A–B** are flow charts of a subroutine employed by the *e*-to-the-*x*-power and natural logarithm *x* routines of FIGS. **95A–B** and **96**, respectively.

FIG. **100** is a flow chart of a subroutine employed by the arctangent *x* and natural logarithm *x* routines of FIGS. **94A–B** and **96**, respectively.

FIG. **101** is a flow chart of sine and cosine routines that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **102** is a flow chart of a table $f(x)$ routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **103** is a flow chart of an arc routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **104** is a flow chart of an arcsine routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **105** is a flow chart of an arccosine routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **106** is a flow chart of a conversion to polar coordinates routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **107** is a flow chart of a conversion to rectangular coordinates routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **108** is a flow chart of a 10-to-the-*x*-power routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **109** is a flow chart of an absolute value *y* routine that may be performed when the mathematics read-only memory module is plugged into the adaptable progammable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **110** is a flow chart of an *x*-to-the-*y*-power routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **111** is a flow chart of a logarithm-to-the-base-10 routine that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **112** is a flow chart of a routine for converting degrees, minutes, and seconds to degrees that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **113** is a flow chart of a routine for converting degrees to degrees, minutes, and seconds that may be performed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1, 2,** and **3A–B**.

FIG. **114** is a flow chart of a subroutine employed by the routine of FIG. **113** for converting degrees to degrees, minutes, and seconds.

FIG. **115** is a flow chart of an *x* factorial routine that may be employed when the mathematics read-only

memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **116** is a flow chart of a recall routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **117** is a flow chart of an accumulate plus routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **118** is a flow chart of an accumulate minus routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIGS. **119A–B** are flow charts of a rounding routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **120** is a flow chart of a scale routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **121** is a flow chart of a subroutine employed by the scale routine of FIG. **120**.

FIG. **122** is a flow chart of a clear routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **123** is a flow chart of a definable $f(x)$ routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **124** is a flow chart of a do-loop routine that may be employed when the mathematics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIGS. **125A–B** are flow charts of a summation routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **126** is a flow chart of a correct routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **127** is a flow chart of a *t*-pair routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **128** is a flow chart of an *x*-squared routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **129** is a flow chart of a mean value routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **130** is a flow chart of a maximum/minimum routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **131** is a flow chart of a variance routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **132** is a flow chart of an initialize routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **133** is a flow chart of a regression routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **134** is a flow chart of a variables routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **135** is a flow chart of an r^2-correlation routine that may be performed when the statistics read-only memory module is plugged into the adaptable programmable calculator of FIGS. **1**, **2**, and **3A–B**.

FIG. **136** is a block diagram of the microprocessor of FIGS. **3A–B**.

FIGS. **137A–D** are detailed schematic diagrams of the microprocessor of FIGS. **34A–B** and **136**.

FIG. **137'** is a figure map showing how the detailed schematic diagrams of FIGS. **137A–D** fit together.

FIGS. **138A–H** are detailed flow charts illustrating the operation of the microprocessor of FIGS. **3A–B**, **136**, and **137A–D**.

FIG. **138'** is a figure map showing how the detailed flow charts of FIGS. **138A–D** fit together.

FIG. **138''** is a figure map showing how the detailed flow charts of FIGS. **138E–H** fit together.

FIG. **138I** shows the macro-instruction coding table, the used data format, and the used address and constants for the micro-processor of FIGS. **3A–B** and **136**.

FIG. **138J** is an instruction table for the microprocessor of FIGS. **3A–B** and **136**.

FIG. **139** is a block diagram of the programmable clock of FIGS. **3A–B**.

FIGS. **140A–C** are detailed schematic diagrams of the programmable clock of FIGS. **3A–B** and **139** and of a portion of the input-output control unit of FIG. **3A–B**.

FIG. **140'** is a figure map showing how the detailed schematic diagrams of FIGS. **149A–C** fit together.

FIG. **141** is a waveform diagram illustrating the operation of the programmable clock of FIGS. **3A–B 139**, and **140A–C**.

FIGS. **142A–D** are detailed schematic diagrams of the shift register and arithmetic logic units of FIGS. **3A–B**.

FIG. **142'** is a figure map showing how the detailed schematic diagrams of FIGS. **142A–D** fit together.

FIG. **143** is a block diagram of the arithmetic-logic unit of FIGS. **3A–B**.

FIG. **144** is a table of the function code assignments of the arithmetic-logic unit of FIGS. **142A–D** and **143**.

FIG. **145** is a table of integrated circuits that may be employed to construct the ALU of FIGS. **142A–D** and **143**.

FIG. **146** is a block diagram of the memory unit of FIGS. **3A–B**.

FIG. **147** is a table of mnemonics used in the memory unit of FIG. **3**.

FIG. **148** is a schematic diagram of the read-write memory of FIGS. **3A–B 4**, and **146**.

FIG. **149** is a schematic diagram of one of the add-on read-write memory modules of FIG. **146** that may be plugged into the calculator to increase the amount of program storage memory available to the user.

FIG. **150** is a schematic diagram of the other add-on read-write memory module of FIG. **146** that may be

plugged into the calculator to further increase the amount of program storage memory available to the user.

FIG. **151** is a schematic diagram of the read-only memory of FIGS. 3A–B **4**, and **146**.

FIG. **152** is a schematic diagram of the add-on read-only memory modules of FIG. **146** that may be pluged into the calculator to increase the number of functions available to the user.

FIG. **153** is a block diagram of one of the read-only memory chips of FIGS. **151–152**.

FIGS. **154**A–D are schematic diagrams of one of the read-only memory chips of FIGS. **151–152**.

FIG. **154'** is a figure map showing how the detailed schematic diagrams of FIGS. **154**A–D fit together.

FIG. **155** is a memory map of the memory unit of FIGS 3A–B and **4** illustrating how it is partitioned into the read-only and read-write memory chips of FIGS. **148–153** and **154**A–D.

FIG. **156** is a flow chart illustrating how the row numbers of the lists stored in the read-only memory chips are computed.

FIG. **157** is a table of bit mumbers of actual bits used in connection with the flow chart of FIG. **156**.

FIGS. **158**A–D are detailed schematic diagrams of the memory address register of FIGS. 3A–B and **146**.

FIG. **158'** is a figure map showing how the detailed schematic diagrams of FIGS. **158**A–D fit together.

FIGS. **159**A–D are detailed schematic diagrams of the control circuitry of FIGS. **3** and **146**.

FIG. **159'** is a figure map showing how the detailed schematic diagrams of FIGS. **159**A–D fit together.

FIG. **160** is a waveform diagram illustrating the operation of the control circuitry of FIGS. **159**A–D.

FIGS. **161**A–D are detailed schematic diagrams of the memory access register of FIGS. 3A–B and **146**.

FIG. **161'** is a figure map showing how the detailed schematic diagrams of FIGS. **161**A–D fit together.

FIGS. **162**A–D are detailed schematic diagrams of the input-output register and gating control circuits employed in the input-output control unit of FIG. **3**.

FIG. **162'** is a figure map showing how the detailed schematic diagrams of FIGS. **162**A–D fit together.

FIG. **163** is a schematic diagram of the source and and relationship of the input-output party lines connected to the peripheral interface module receptacles of FIG. **2**.

FIG. **164** is a waveform diagram illustrating the operation of the control section of the input-output control unit FIGS. 3A–B and **140**A–C.

FIG. **165** is a flow chart illustrating the operation of the control section of the input-output control unit of FIGS. 3A–B and **140**A–C.

FIG. **166** is a schematic diagram of the address code decoding for the output section of the interface modules employed with the input-output control unit of FIGS. 3A–B.

FIG. **167** is a chart listing and defining all of the input-output lines of the input-output control unit of FIGS. 3A–B.

FIG. **168** is a waveform diagram of some of the output signals employed by the input-output control unit and associated interface modules of FIGS. 3A–B.

FIG. **169** is a waveform diagram of some of the input signals employed by the input-output control unit and associated interface modules of FIGS. 3A–B.

FIG. **170** is a waveform diagram illustrating the operation of the interrupt mode of operation of the input-output control unit of FIGS. 3A–B.

FIG. **171** is a schematic diagram of logic that may be used to interface an output peripheral to the input-output control unit of FIGS. 3A–B.

FIG. **172** is a schematic diagram of logic that may be used to interface an input peripheral to the input-output control unit of FIGS. 3A–B.

FIG. **173** is a schematic diagram of logic that may be used to interface an interrupting peripheral to the input-output control unit of FIGS. 3A–B.

FIGS. **174**A–D are detailed schematic diagrams of the keyboard input unit employed in the adaptable programmable calculator of FIGS. **1**, **2**, and 3A–B.

FIG. **174'** is a figure map showing how the detailed schematic diagrams of FIGS. **174**A–D fit together.

FIG. **175** is a schematic diagram of one of the transformer pairs employed in the keyboard input of FIGS. **174**A–D.

FIG. **176** is a pictorial view of a transformer employed in the keyboard input unit of FIG. **3**.

FIG. **177** is a schematic diagram of the transformer of FIG. **176**.

FIG. **178** is a schematic diagram of a portion of the keyboard input unit of FIGS. **174**A–D.

FIG. **179** illustrates the required polarities for the drive and sense lines employed in the keyboard input unit of FIG. **3** and **176**.

FIG. **180** is a block diagram of the magnetic card reading and recording unit employed in the calculator of FIGS. **1**, **2**, and 3A–B.

FIGS. **181**A–G are schematic diagrams of the magnetic card reading and recording unit of FIG.**180**.

FIG. **181'** is a figure map showing how the detailed schematic diagrams of FIGS. **181**A–G fit together.

FIG. **182** is a block diagram illustrating how the magnetic card reading and recording unit of FIGS. **180** and **181**A–G interacts with the calculator of FIGS. **1**, **2**, and 3A–B.

FIG. **183** is a block diagram of the output display unit employed in the adaptable programmable calculator of FIGS. **1**, **2**, and 3A–B.

FIGS. **184**A–B are detailed schematic diagrams of the output display unit of FIG. **183**.

FIG. **184'** is a figure map showing how the detailed schematic diagrams of FIGS. **184**A–B fit together.

FIG. **185** is a block diagram of the output printer unit employed in the adaptable programmable calculator of FIGS. **1**, **2**, and 3A–B.

FIG. **186** is a cross-sectional view taken along the line A—A in FIG. **185**.

FIGS. **187**A–B are detailed schematic diagrams of the thermal printing head, isolation diodes, four group drivers, and one-of-10 decoder of FIG. **185**.

FIG. **187'** is a figure map showing how the detailed schematic diagrams of FIGS. **187**A–B fit together.

FIG. **188** is a partial plan view of the thermal printing head of FIG. **185**.

FIGS. **189**A–D are a detailed schematic diagrams of the 20 dot drivers the internal 10-bit shift register, the motor drivers, the motor drive control circuit, and the printer timing circuit of the motor drivers of FIG. **185**.

FIG. **190** is a figure map showing how the detailed schematic diagrams of FIGS. **189**A–D fit together.

FIGS. **191**A–B are detailed schematic diagrams of the motor drivers of FIG. **185**.

FIG. 191' is a figure map showing how the detailed schematic diagrams of FIGS. 191A–B fit together.

FIG. 192 illustrates how the output printer unit of FIGS. 185–191 prints out each character.

FIG. 193 is a flow chart illustrating the printing operation of the output printer unit of FIGS. 186, 187A–B, 188, 189A–D, and 191A–B.

FIG. 194 is a block diagram of the power supply system employed in the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 195 is a detailed schematic of the 5 volt power supply of FIG. 194.

FIG. 196 is detailed schematic diagram of the 16, 20, and 24 volt power supplies of FIG. 194.

FIGS. 197A–B are detailed schematic diagrams of the positive and negative 12 volt power supplies of FIG. 194.

FIG. 198 is a schematic diagram of an alternative ripple signal circuit that may be employed in the power supply of FIG. 195.

FIGS. 199A–B are block diagrams of an interface module that may be employed to interface a typewriter to the adaptable programmable calculator of FIGS. 1, 2, and 3A–B.

FIG. 199' is a figure map showing how the block diagrams of FIGS. 199A–B fit together.

FIGS. 200A–C are schematic diagrams of the interface module of FIGS. 199A–B.

FIG. 200' is a figure map showing how the schematic diagrams of FIGS. 200A–C fit together.

DESCRIPTION OF THE PREFERRED EMBODIMENT

GENERAL DESCRIPTION

Referring to FIGS. 1 and 2, there is shown an adaptable programmable calculator 10 including both a keyboard input unit 12 for entering information into and controlling the operation of the calculator and a magnetic card reading and recording unit 14 for recording information stored within the calculator onto one or more external magnetic cards 16 and for subsequently loading the information recorded on these and other similar magnetic cards back into the calculator. The calculator also includes a solid state output display unit 18 for displaying three lines of numeric information stored within the calculator and a group of indicator lights 19, serving as part of the keyboard input unit, for indicating the status of the calculator. It may also include an output printer unit 20 for printing out alphameric information on a strip of thermal-sensitive recording paper 22. All of these input and output units are mounted within a single calculator housing 24 adjacent to a curved front panel 26 thereof.

As shown in FIG. 2, a plurality of peripheral input and output units 28 including, for example, a digitizer, a marked card reader, an X-Y plotter, and a typewriter may be connected to the calculator at the same time by simply inserting interface modules 30 associated with the selected peripheral units into any of four receptacles 32 provided therefor in a rear panel 34 of the calculator housing. As each interface module 30 is inserted into one of these receptacles, a spring-loaded door 38 at the entrance of the receptacle swings down allowing passage of the interface module. Once the interface module is fully inserted, a printed-circuit terminal board 40 contained within the interface module plugs into a mating edge connector mounted inside the calculator. If any of the selected peripheral units require AC line power, their power cords may be plugged into any of three AC power outlets 42 provided therefor at the rear panel of calculator housing 24.

Referring to the simplified block diagram shown in FIGS. 3A–B, it may be seen that the calculator also includes an input-output control unit 44 (hereinafter referred to as the I/O control unit) for controlling the transfer of information to and from the input and output units, a memory unit 46 for storing and manipulating information entered into the calculator and for storing routines and subroutines of basic instructions performed by the calculator, and a central processing unit 48 (hereinafter referred to as the CPU) for controlling the execution of the routines and subroutines of basic instructions stored in the memory unit as required to process information entered into or stored within the calculator. The calculator also includes a bus system comprising an $\overline{S}$-bus 50, a T-bus 52, and an $\overline{R}$-bus 54 for transferring information from the memory and I/O control units to the CPU, from the CPU to the memory and I/O control units, and between different portions of the CPU. It further comprises a power supply for supplying DC power to the calculator and peripheral units employed therewith and for issuing a control signal $\overline{POP}$ when power is supplied to the calculator.

The I/O control unit 44 includes an input-output register 56 (hereinafter referred to as the I/O register), associated I/O gating control circuitry 58, and input-output control logic 60 (hereinafter referred to as the I/O control). I/O register 56 comprises a universal 16-bit shift register into which information may be transferred either bit-serially from CPU 48 via T-bus 52 or in parallel from keyboard input unit 12, magnetic card reading and recording unit 14, and peripheral input units 28 such as the marked card reader via twelve input party lines 62. Information may also be transferred from I/O register 56 either bit-serially to CPU 48 via S-bus 50 or in parallel to magnetic card reading and recording unit 14, solid state output display unit 18, indicator lights 19, output printer unit 20, and peripheral output units 28 such as the X-Y plotter or the typewriter via 16 output party lines 64.

I/O gating control circuitry 58 includes control circuits for controlling the transfer of information into and out of I/O register 56 in response to selected I/O qualifier control signals from CPU 48 and selected I/O control instructions from I/O control 60. It also includes an interrupt control circuit 65, a peripheral control circuit 66, a magnetic card control circuit 67, a printer control circuit 68, a display control circuit 69, and an indicator control circuit 70 for variously controlling the input and output units and issuing control signals QFG and EBT to I/O control 60 via two output lines 71 and 72. These last mentioned control circuits variously perform their control functions in response to control signal $\overline{POP}$ from the power supply, I/O qualifier control signals from CPU 48, I/O control instructions from I/O control 60, and control signals from keyboard input unit 12. Interrupt control circuit 65 initiates the transfer of information into I/O register 56 from keyboard input unit 12 or interrupting peripheral input units 28 such as the marked card reader and issues a qualifier control signal QNR to CPU 48 via output lines 73. Peripheral control circuit 66 enables interface modules 30 plugged into the calculator to respond to information from I/O register 56, control associated peripheral

units **28**, transfer information to and/or receive information from associated peripheral units **28**, and in some cases initiate the transfer of information to I/O register **56** from the interface modules themselves. Magnetic card control circuit **67** enables magnetic card reading and recording unit **14** to respond to information in I/O register **56** and either read information into I/O register **56** from a magnetic card **16** or record information onto a magnetic card **16** from I/O register **56**. Printer control circuit **68**, display control circuit **69**, and indicator control circuit **70** enable output display unit **18**, output printer unit **20**, and indicator lights **19**, respectively, to respond to information from I/O register **56**.

When a basic I/O instruction obtained from memory unit **46** is to be executed, CPU **48** transfers control to I/O control **60** by issuing a pair of I/O microinstructions $\overline{PTR}$ and $\overline{XTR}$ thereto. In response to these I/O microinstructions from CPU **48**, control signal $\overline{POP}$ from the power supply, control signals QFG and EBT from I/O gating control circuitry **58**, and I/O qualifier and clock control signals from CPU **48**, I/O control **60** selectively issues one or more I/O control instructions to gating control circuitry **58** as required to execute the basic I/O instruction designated by CPU **48** and issues control signals $\overline{TTX}$, $\overline{XTR}$, QRD, and SCB to CPU **48** via output lines **74–77**. The I/O qualifier control signals issued to I/O control **60** and gating control circuitry **58** by CPU **48** are derived from the basic I/O instruction to be executed. Those qualifier control signals issued to I/O control **60** designate the specific I/O control instructions to be issued by I/O control **60**, while those issued to gating control circuitry **58** designate selected control circuits to be employed in executing the basic I/O instruction.

Memory unit **46** includes a modular random-access read-write memory **78** (hereinafter referred to as the RWM), a modular read-only memory **80** (hereinafter referred to as the ROM), a memory address register **82** (hereinafter referred to as the M-register), a memory access register **84** (hereinafter referred to as the T-register), and control circuitry **85** for these memories and registers RWM **78** and ROM **80** comprise MOS-type semiconductor memories. As shown in the memory map of FIG. **4**, they are organized into eight 1,024-word pages. The basic RWM **78** contains a data storage section **86** of 512 16-bit words extending from address **1000** to address **1777** on page **0** and a separate program storage section **88** of 512 six-bit words extending from address **12000** to address **12777** on page **5**. All addresses on the memory map are represented in octal form.

Data storage section **86** contains 49 four-word storage registers available to the user (as user addresses **000–048**) for manipulating and storing data, 60 additional four-word storage registers that may be made available to the user (as user addresses **049–108**) for the same purpose, and 76 words dedicated for use by CPU **48**. The 60 additional four-word storage registers may be made available to the user by altering a control routine in the ROM section **80**. This is accomplished by removing a top panel **90** of the calculator housing shown in FIG. **1**, removing a printed circuit board containing the control routine to be altered, and substituting another printed circuit board containing the altered control routine. Additional data storage modules made available to the user are automatically accommodated by the calculator.

As shown in the more detailed memory map of FIG. **5**, the dedicated portion of data storage section **86** includes 12 words (addresses **1750–1753, 1740–1743,** and **1760–1763**) employed as "X", "Y", and "Z" four-word working registers available to the user and 8 words (addresses **1764–1773**) employed as *a* and *b* four-word storage registers available to the user. It also includes eight words (addresses **1744–1747** and **1754–1757**) employed as "AR1" and "AR2" four-word working registers for performing binary-coded-decimal arithmetic; 12 words (addresses **1664–1677**) employed as three ("T1'", "T2', and "T3'") or more temporary storage registers in connection with a definable section **91** of keyboard input unit **12** (see FIG. **1**); 16 words (addresses **1720–1737**) employed as four ("T1", "T2", "T3", and "T4") or more temporary storage registers in connection with the remaining sections of the keyboard input unit; seven words (addresses **1711–1717**) employed as a variable-length "system subroutine stack" for storing return addresses required by programs stored in ROM **80** and as temporary storage for housekeeping information required by CPU **48**; one word (address **1777**) employed as a "system stack pointer"; five words (addresses **1702–1706**) employed as a "user subroutine stack" for storing return addresses required by programs entered into program storage section **88** of RWM **78** by the user; one word (address **1710** employed as a "user stack pointer"; one word (address **1701**) employed as a "user program counter" for program steps entered into program storage section **88** by the user; one word (address **1707**) employed as an "address of user **0000**" register for storing the first address available to the user in program storage section **88**; one word (address **1700**) employed as a "microprocessor store" register for storing housekeeping information required by CPU **48**; one word (address **1774**) employed as an "interrupt store" register for storing information displaced from CPU **48** by some keyboard entries; one word (address **1775**) employed as an "input buffer" register for storing keyboard information awaiting entry into CPU **48**, and one word (address **1776**) employed as a "status word" register for storing information regarding the present state of the calculator.

As shown in the memory map of FIG. **4**, program storage section **88** of RWM **78** contains 500 one-word program step registers available to the user (as user addresses **0000–0499**) for storing programs and 12 words dedicated for use by CPU **48**. An additional 1,536 one-word program-step registers may be made available to the user (as user addresses **0500–2035**) in steps of 512 words (addresses **13000–13777**) and 1,024 words (addresses **14000** to **15777**). This is accomplished by removing top panel **90** of the calculator housing shown in FIG. **1** and plugging additional program storage modules into the calculator. Added program storage modules are automatically accommodated by the calculator.

As shown in the more detailed memory map of FIG. **6**, the dedicated portion of program storage section **88** includes one word (address **12013**) employed as a "security word" register for storing a code designating a program stored in program storage section **88** as being secure. It also includes one word (address **12000**) employed as an "execution flag " register for storing infor-

mation indicating whether CPU **48** is being controlled by a program stored in the program storage section or by the keyboard input unit; two words (addresses **12001–12002**) employed as "normalize flag" and "print flag" registers for storing information about numeric data being processed by CPU **48**; three words (addresses **12003–12005**) employed as "temporary display" registers for storing housekeeping information about the character position, decimal position, and register location of numeric information being displayed by the output display unit; one word (address **12006**) employed as a "print code buffer" register for storing housekeeping information about alphameric information being printed out by the output printer unit; and four words (addresses **12007–12012**) available for other uses.

As shown in the memory map of FIG. **4**, the basic ROM **80** contains 2,048 16-bit words extending from address **0000** to address **0777** on page **0**, from address **4000** to address **5777** on page **2**, and from address **16000** to address **16777** on page 7. Routines and subroutines of basic instructions for performing the basic functions of the calculator and constants employed by these routines and subroutines are stored in these portions of ROM **80**. An additional 3,072 16-bit words of ROM may also be added on pages **1**, **3**, and **4** in steps of 512 and 1,024 words. This is accomplished by simply inserting plug-in ROM modules **92** into receptacles **94** provided therefor in top panel **90** of the calculator housing as illustrated in FIG. **1** by the partially-inserted plug-in ROM module on the left. As each plug-in ROM module **92** is inserted into one of these receptacles a spring-loaded door **95** at the entrance of the receptacle swings down allowing passage of the plug-in ROM module. Once the plug-in ROM module is fully inserted as illustrated by the plug-in ROM module on the right, a printed circuit terminal board **96** contained within the plug-in ROM module plugs into a mating edge connector mounted inside the calculator. A handle **98** pivotally mounted at the top end of each plug-in ROM module **92** facilitates removal of the plug-in ROM module once it has been fully inserted into one of the receptacles **94**.

Routines and subroutines of basic instructions (and any needed constants) for enabling the calculator to perform many additional functions are stored in each plug-in ROM module **92**. The user himself may therefore quickly and simply adapt the calculator to perform many additional functions oriented toward his specific needs by simply plugging ROM modules of his own choosing into the calculator. Added plug-in ROM modules are automatically accommodated by the calculator and associated with definable section **91** of keyboard input unit **12** or employed to expand the functions performed by this and other sections of the keyboard input unit.

Referring again to FIGS. **3A–B**, M-register **82** of the memory unit comprises a recirculating 16-bit serial shift register into which information may be transferred bit-serially from CPU **48** via T-bus **52** and out of which information may be transferred bit-serially to CPU **48** via $\overline{S}$-bus **50**. Information shifted into M-register **82** may be employed to address any word in RWM **78** or ROM **80** via 15 output lines **106**.

T-register **84** of the memory unit comprises a recirculating 16-bit serial shift register into which information may be transferred either bit-serially from CPU **48** via T-bus **52** or in parallel from any addressed word in RWM **78** and ROM **80** via 16 parallel input lines **108**. Information may be transferred from T-register **104** either bit-serially to CPU **48** via $\overline{S}$-bus **50** or in parallel to any addressed word in RWM **78** via 16 parallel output lines 110. The four least significant parallel of information contained in T-register **104** may comprise binary-coded-decimal information and may be transferred from the T-register in parallel to CPU **48** via three parallel output lines **112** taken with $\overline{S}$-bus **50**.

The control circuitry **85** of the memory unit controls these transfers of information into and out of M-register **82** and T-register **84**, controls the addressing and accessing of RWM **78** and ROM **80**, and refreshes RWM **78**. It performs these functions in response to memory microinstructions, memory clock pulses, and shift clock pulses from CPU **48**.

CPU **48** includes a register unit **114**, an arithmetic-logic unit **116** (hereinafter referred to as the ALU), a programmable clock **118**, and a microprocessor **120**. Register unit **114** comprises four recirculating 16-bit shift registers **122**, **124**, **126**, and **128** and one four-bit shift register **130**. Shift registers **122** and **124** serve as 16-bit serial accumulator registers (hereinafter referred to as the A-register and the B-register, respectively) into which information may be transferred bit-serially from ALU **116** via T-bus **52** and out of which information may be transferred bit-serially to ALU **116** via $\overline{R}$-bus **54**. The four least significant bit positions of A-register **122** also serve as a four-bit parallel accumulator register into which four bits of binary-coded-decimal information may be transferred in parallel from ALU **116** via four parallel input lines **132** and out of which four bits of binary-coded-decimal information may also be transferred in parallel to ALU **116** via three parallel output lines **134** taken with $\overline{R}$-bus **54**.

Shift register **126** serves as a 16-bit system program counter (hereinafter referred to as the P-register) into which information may be transferred bit-serially from ALU **116** via T-bus **52** and out of which information may be transferred bit-serially to ALU **116** via $\overline{R}$-bus **54**. Information contained in the least significant bit position of P-register **126** may also be transferred as a qualifier control signal QPO to microprocessor **120** via output line **135**.

Shift register **128** serves as a sixteen-bit qualifier register (hereinafter referred to as the Q-register) into which information may be transferred bit-serially from ALU **116** via T-bus **52** and out of which information may be transferred bit-serially to ALU **116** via $\overline{R}$-bus **54**. Information contained in the five least-significant bit positions of Q-register **28** is transferred to I/O gating control circuitry **58** as five one-bit I/O qualifier control signals **Q00–Q04** via five parallel output lines **136**, and information contained in the six next-least significant bit positions of the Q-register is transferred to I/O control **60** as six one-bit I/O qualifier control signals **Q05–Q10** via six parallel output lines **138**. Similarly, information contained in the seven least-significant, the ninth and eleventh least-significant, and the most significant bit positions of Q-register **128** and information derived from the thirteenth, fourteenth, and fifteenth bit positions of the Q-register may be transferred to microprocessor **120** as 11 one-bit microprocessor qualifier control signals **Q00–Q06**, **Q08**, **Q10**, **Q15**, and QMR via 11 output lines **140**. Information contained in the twelfth through the fifteenth least-significant bit po-

sitions of Q-register **128** may be transferred to microprocessor **120** as a four-bit primary address code via four parallel output lines **142.**

Shift register **130** serves as a four-bit serial extend register (hereinafter referred to as the E-register) into which information may be transferred bit-serially either from ALU **116** via T-bus **52** or from the least-significant bit position of T-register **84** via input line **144.** Information may also be transferred out of E-register **130** to ALU **116** via $\overline{R}$-bus **54.**

Register unit **114** also includes control circuitry **146** for controlling the transfer of parallel binary-coded-decimal information into and out of A-register **122** and the transfer of serial binary information into and out of A-register **122,** B-register **124,** P-register **126,** Q-register **128,** and E-register **130.** This is accomplished in response to register microinstructions from microprocessor **120,** control signals $\overline{TTX}$ and $\overline{XTR}$ from I/O control **60,** and shift clock control pulses from programmable clock **118.** Control circuitry **146** includes a flip-flop **148** (hereinafter referred to as the A/B flip-flop) for enabling the transfer of information into and out of either the A-register **122** or the B-register **124** as determined by the state of the A/B flip-flop. The state of A/B flip-flop **148** is initially determined by information Q11 transferred to the A/B flip-flop from the twelfth least-significant bit position of Q-register **128** but may be subsequently complemented one or more times by microinstruction CAB from microprocessor **120.**

ALU **116** may perform either one-bit serial binary arithmetic on data received from T-register **84** or M-register **82** via $\overline{S}$-bus **50** and/or from any register of register unit **114** via $\overline{R}$-bus **54** or four-bit parallel binary-coded-decimal arithmetic on data received from T-register **84** via output lines **112** taken with $\overline{S}$-bus **50** and/or from A-register **122** via output lines **134** taken with $\overline{R}$-bus **54.** It may also perform logic operations on data received from memory unit **46** and/or register unit **114** via any of these lines. The arithmetic and logic operations performed are designated by ALU microinstructions from microprocessor **120** and are carried out in response to these microinstructions, shift clock control pulses from programmable clock **118,** and control signal SCB from I/O control **60.** Information is also transferred from ALU **116** to A-register **122** via output lines **132** or to I/O register **56,** M-register **82,** T-register **84,** or any register of register unit **114** via T-bus **52** in response to microinstructions and control signals applied to these registers. If a carry results while ALU **116** is performing either one-bit serial binary arithmetic or four-bit parallel binary-coded-decimal arithmetic, the ALU issues a corresponding qualifier control signal QBC and QDC to microprocessor **120** via one of two output lines **152** and **154.**

Programmable clock **118** includes a crystal-controlled system clock **156,** a clock decoder and generator **158,** and a control gate **160.** System clock **156** issues regularly recurring clock pulses to clock decoder and generator **158** via output line **162.** In response to these regularly recurring clock pulses from system clock **156** and to four-bit clock codes from microprocessor **120,** clock decoder and generator **158** issues trains of n shift clock pulses to ALU **116,** M-register **82,** T-register **82,** and all of the registers of register unit **114** via output line **164.** These trains of *n* shift clock pulses are employed for shifting a corresponding number of bits of serial information into or out of any of these registers or for shifting a carry bit in the ALU. The number *n* of pulses in each of these trains may vary from one to 16 as determined by the number of bits of serial information required during each operation to be performed. In response to a control signal $\overline{CCO}$ from microprocessor **120,** control gate **160** prevents any shift clock pulses from being applied to the ALU or any of these registers. Upon completion of each train of n shift clock pulses, clock decoder and generator **158** issues a ROM clock pulse to microprocessor **120** via output line **166** and an I/O clock pulse to I/O control **60** via output line **168.** In response to the regularly recurring clock signal from system clock **56,** clock decoder and generator **158** also issues correspondingly regularly recurring memory clock pulses to memory unit **46** via output line **170.**

Microprocessor **120** selectively issues two I/O microinstructions to I/O control **60** via two output lines **172,** six memory microinstructions to memory unit **46** via six output lines **174,** 13 register microinstructions to register unit **114** via 13 output lines **176,** and five ALU microinstructions to ALU **116** via five output lines **178.** It also issues a four-bit clock code associated with each of these microinstructions to clock decoder **158** via four output lines **180.** These microinstructions and associated clock codes are issued as determined by the control signal $\overline{POP}$ from the power supply, the 11 microprocessor qualifier control signals from Q-register **128,** the four-bit primary address codes from Q-register **128,** and the five microprocessor qualifier control signals from I/O control **60,** interrupt control **65,** ALU **116,** and P-register **126.**

As shown in the simplified flow chart of FIG. **7,** microprocessor **120** executes a hardware diagnostic routine (stored within the microprocessor itself) in response to the control signal $\overline{POP}$. Upon completion of this diagnostic routine, ALU **116** issues the qualifier control signal QBC indicating whether or not the diagnostic routine was successful, microprocessor **120** thereupon responds to this qualifier control signal by entering the basic machine operating loop and issuing microinstructions causing a 16-bit instruction stored in ROM **80** to be loaded into T-register **84** and transferred from there to Q-register **128.** Microprocessor **120** thereupon sequentially responds to one or more additional qualifier control signals by issuing microinstructions and associated clock codes for executing the instruction then contained in Q-register **128** and causing another 16-bit instruction stored in ROM **80** to be loaded into T-register **84** and transferred from there to the Q-register. When an instruction requiring multiple branching is contained in Q-register **128,** microprocessor **120** issues a pair of microinstructions $\overline{UTR}$ and $\overline{XTR}$ causing the microprocessor to respond to a four-bit primary address code from the Q-register by issuing additional microinstructions and associated clock codes for executing the instruction contained in the Q-register.

As illustrated by the basic machine operating loop shown in the flow chart of FIG. **7,** microprocessor **120** initially responds to the qualifier control signal QNR either by issuing microinstructions and associated clock codes for interrupting the basic machine operating loop and executing an I/O service routine or by issuing microinstructions and associated clock codes for loading A/B flip-flop **148** with the information Q11

contained in Q-register **128.** The manner in which microprocessor **120** responds is determined by the condition of the qualifier control signal QNR, which in turn indicates whether or not the basic machine operating loop should be interrupted.

Assuming the basic machine operating loop is not to be interrupted, microprocessor **120** loads the information Q11 into A/B flip-flop **148** and responds to the qualifier control signal QMR either by issuing microinstructions for transferring an address portion of the instruction contained in Q-register **128** from T-register **84** into M-register **82** or by responding to another qualifier control signal Q15. Again, the manner in which microprocessor **120** responds is determined by the condition of the qualifier control signal QMR, which in turn indicates whether or not the instruction contained in Q-register **128** is a memory reference instruction.

Assuming the instruction contained in Q-register **128** is a memory reference instruction, microprocessor **120** transfers the required address information into the M-register **82** and responds to qualifier control signal Q10 either by issuing microinstructions and associated clock codes to select the base page of the memory (i.e. page **0**) or by issuing microinstructions and associated clock codes to select the current page of the memory (i.e. the page from which the instruction contained in Q-register **128** was obtained). In either case, the microprocessor then issues microinstructions as required to read data from the preset page of the memory at the address designated by the address information last transferred into M-register **82.** Upon completion of this operation, microprocessor **120** responds to qualifier control signal Q15 by issuing additional microinstructions and associated clock codes to execute an indirect memory access operation if the condition of this qualifier control signal indicates that the address information contained in M-register **82** is indirect.

Assuming the address information contained in M-register **82** is direct (or upon completion of the indirect memory access operation), microprocessor **120** issues microinstructions and associated clock codes causing the microprocessor itself to respond to a four-bit primary address code from the Q-register. The microprocessor responds by issuing additional microinstructions and associated clock codes for executing whichever one of 10 possible memory reference instructions is contained in Q-register **128** and designated by the four-bit primary address code. Following execution of the designated memory reference instruction, microprocessor **120** issues microinstructions and associated clock codes causing another 16-bit instruction stored in ROM **80** to be loaded into T-register **84** and transferred from there to Q-register **128** thereby beginning another cycle of the basic machine operating loop.

As illustrated by other possible paths of the basic machine operating loop shown in FIG. **7,** microprocessor **120** sequentially responds to other qualifier control signals when other types of instructions are contained in Q-register **128.** For example, when an I/Oinstruction is contained in Q-register **128,** microprocessor **120** sequentially responds to qualifier control signals QNR, QMR, Q15, Q10, and QRD by issuing microinstructions and associated clock codes to execute the I/O instruction. It should be noted that the microprocessor qualifier control signals not shown in the simplified flow chart of FIG. **7** are variously contained within those flow chart blocks requiring decisions as will hereinafter become apparent.

KEY OPERATIONS

All operations performed by the calculator may be controlled or initiated by the keyboard input unit and/or by keycodes entered into the calculator from the keyboard input unit, the magnetic card reading and recording unit, or peripheral input units such as the marked card reader and stored as program steps in the program storage section of the RWM. The calculator responds to keycodes in basically the same manner whether obtaining them from the keyboard input unit or from the program storage section of the RWM. An operational description of the keyboard input unit is therefore now given with specific reference to FIGS. **1** and **8,** except as otherwise indicated.

Line Switch

An on-off line switch **182,** which may be considered as part of the keyboard input unit, controls the application of power to the calculator and hence initiation of the control signal $\overline{POP}$ from the power supply. The indicator lights **19** serve as a pilot light since at least two of them are always turned on while power is applied to the calculator.

As shown in FIG. **2,** the calculator may be operated at 230, 200, 115, or 100 volts ± 10% as determined by a pair of line voltage selector switches mounted at rear panel **34** of the calculator housing and at a line frequency within the range of 48 to 66 Hertz. The calculator is provided with a 6-amp fuse and either a **1**-amp fuse for operation at a line voltage of 200 or 230 volts ± 10% or a 2-amp fuse for operation at a line voltage of 100 or 115 volts ± 10%. It is also provided with a three-conductor power cable **184** which, when plugged into an appropriate AC power outlet, grounds the calculator housing. The maximum power consumption of the calculator is 150 voltamps. No more than a total of 610 voltamps may be drawn from AC power outlets **42** provided for peripheral units **28.**

Mode Keys (RUN, PRGM, KEY LOG)

When the calculator is first turned on, it is automatically initialized and placed in a manual operating mode. If the calculator is switched to some other operating mode, it may subsequently be placed in the manual operating mode again by simply depressing the RUN mode key. In the manual operating mode, operation of the calculator is manually controlled by the user from the keyboard input unit. During this mode the output display unit displays a decimal numeric representation of the contents of the *x*-, *y*-, and *z*-registers (or of associated memory registers in which the actual or intended contents of the *x*-, *y*-, and *z*-registers are temporarily stored and, for simplicity of description, are then considered to be the contents of the *x*-, *y*-, and *z*-registers). The contents of the *x*-, *y*-, and *z*-registers are displayed adjacent to the corresponding register designators "keyboard *x*", "accumulator *y*", and "temporary *z*", respectively, in the display window. Depression of the RUN mode key also conditions the calculator for operation in an automatic operating mode, a first key-log printing mode, a program-list printing mode, a magnetic card reading mode, and a magnetic card recording mode as determined by other keys hereinafter explained. A run indicator light **19** positioned immediately below the RUN mode key is turned on

when the calculator is operating in any of the RUN modes.

The PRGM mode key is depressed to place the calculator in a program entering mode. In this mode keycodes sequentially entered by the user from the keyboard input unit are stored as program steps in successive program-step registers of the program storage section of the RWM as specified by the user program counter. As described above, 500 program-step registers (user addresses **0000–0499**) are available, and 1536 additional program-step registers (user addresses **(0500–2035**) may be made available, to the user for this purpose. The program step register into which each program step is to be stored and from which each program step is to be obtained in any program-related operation is always specified by the user program counter. Thus, before entering a program or subprogram into the calculator, the user program counter must be set to the address of the program step register into which it is desired to store the initial program step of the program or subprogram to be entered (this address is hereinafter referred to as the desired starting address nnnn of the program). This may be accomplished, when the calculator is in a keyboard-controlled run mode, by depressing the GO TO key followed by the decimal-digit keys **0–9** specifying the desired starting address nnnn of the program. If the desired starting address is **0000,** this may also be accomplished, when the calculator is in the manual operating mode by simply depressing the END key.

Once the user program counter is set to the desired starting address nnnn, the user may proceed to enter the program or subprogram by sequentially performing basically the same key operations that he would normally perform in the manual operating mode. Thus no special language need be learned to program the calculator. During the program entering mode the output display unit displays a decimal numeric representation of the last-entered program step and its associated address and the addresses of the next two program steps to be entered and the present contents of those addresses.

Depression of the PRGM mode key also conditions the calculator for operation in a second key-log printing mode and the program-list printing mode as determined by other keys hereinafter explained. A program mode indicator light **19** positioned immediately below the PRGM mode key is turned on when the calculator is operating in any of the PRGM modes.

The KEY LOG mode key is depressed, when the calculator is in the manual operating mode, to place the calculator in the first key-log printing mode. In this mode, the output printer unit prints out an octal numeric representation of each keyboard operation as it is performed by the user. This provides a permanent record of all keyboard operations (including regular data print-out operations), as illustrated by the following example:

Keyboard entry	Key log
1	01
2	02
↑	27
3	03
4	04
+	33
5	05
÷	35
PRINT	
SPACE	45
	5.00000

If the alpha ROM module enabling the calculator to print out every alphabetic character and many symbols individually or in messages is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each keyboard operation as it is performed by the user. This is illustrated by the following example:

Keyboard entry	Key log	
1	1	01
2	2	02
↑	UP	27
3	3	03
4	4	04
+	+	33
5	5	05
÷	DIV	35
PRINT		
SPACE	PNT	45
		5.00000

In the first key-log printing mode the output display unit displays the same information as during the manual operating mode.

The KEY LOG mode key is depressed, when the calculator is in the program entering mode, to place the calculator in the second key-log printing mode. In this mode the output printer unit prints out an octal numeric representation of each keycode as it is entered into the calculator from the keyboard input unit and a decimal numeric representation of the address at which each such keycode is stored as a program step in the program step in the program storage section of the RWM. This provides a permanent record of all keyboard-entered program steps, as illustrated by the following example:

Keyboard entry	Key log
CLEAR	0000 -------- 20
1	0001 -------- 01
2	0002 -------- 02
↑	0003 -------- 27
3	0004 -------- 03
4	0005 -------- 04
+	0006 -------- 33
5	0007 -------- 05
$x \rightleftarrows y$	0008 -------- 30
PRINT	
SPACE	0009 -------- 45
STOP	0010 -------- 41
END	0011 --------46

If the alpha ROM module is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each keyboard-entered program step. This is illustrated by the following example:

Keyboard entry	Key log	
CLEAR	0000--	CLR --- 20
1	0001--	1 --- 01
2	0002--	2 --- 02

-Continued

Keyboard entry	Key log	
↑	0003-- UP	--- 27
3	0004-- 3	--- 03
4	0005-- 4	--- 04
+	0006-- +	--- 33
5	0007-- 5	--- 05
x⇄y	0008-- XEY	--- 30
PRINT	0009-- PNT	--- 45
STOP	0010-- STP	--- 41
END	0011-- END	--- 46

In the second key-log printing mode the output display unit displays the same information as during the program entering mode.

The KEY LOG mode key is a toggling on-off key (i.e. repeated depressions of the key alternately switch the calculator in and out of either the first or the second key-log printing mode). A key-log indicator light **19** positioned immediately below the KEY LOG mode key is turned on when the calculator is operating in either key-log printing mode.

Program Keys (LIST, LOAD, RECORD)

The LIST program key is depressed, when the calculator is in the manual operating mode, first key-log printing mode, second key-log printing mode, or program entering mode, to place the calculator in the program-list printing mode. In this mode, the output printer unit prints out an octal numeric representation of keycodes then stored as program steps in the program storage section of the RWM and a decimal numeric representation of the addresses of these program steps. These program steps and addresses are printed out in a list beginning with the address initially specified by the user program counter and ending with the address specified by the user program counter when an END program step is encountered or the STOP key is depressed. The user may select the starting address nnnn of the list by depressing to GO TO key followed by the decimal digit keys (**0–9**) designating the desired starting address nnnn, or simply by depressing the END key if the starting address is **0000.** Similarily, the user may terminate the list at any time by depressing the STOP key. When the program-list operation is terminated either by an END program step or by depression of the STOP key, the calculator reverts to its original manual operating, first or second key-log printing, or program entering mode. If the program-list operation is terminated by an END program step, the user program counter specifies the address of the END program step. However, if the program-list operation is terminated by depression of the STOP key, the user program counter specifies the address of the next program step to have been encountered had the STOP key not been depressed.

The list printed out by the output printer unit serves as a permanent record of a sequence of program steps stored as a program, subprogram, or part thereof in the program storage section of the RWM and the address of these program steps. A typical list is illustrated by the right- and left-hand columns of numbers printed out on the strip of thermal-sensitive recording paper 12. If the alpha ROM module is plugged into the calculator, the output printer unit also prints out a mnemonic representation of each of these program steps. This is illustrated by the central column of alphameric characters printed out on the same strip of thermal-sensitive recording paper.

The LOAD program key is depressed, when the calculator is in the manual operating mode or the first key-log printing mode, to place the calculator in the magnetic card reading mode. During this mode, program steps recorded on one or more external magnetic cards **16** are read by the magnetic card reading and recording unit and stored in the program storage section of the RWM. In order to properly accomplish this program loading operation:

1. The user program counter is set to the desired starting address nnnn of the program storage section to be loaded. This may be accomplished by depressing the GO TO key followed by those decimal degit keys designating the desired starting address nnnn, or simply by depressing the END key if the desired starting address is **0000.**

2. A recorded magnetic card **16** inserted into an input receptable **186** of the magnetic card reading and recording unit with the first side **187** to be read placed in the operative reading and recording position as shown in FIG. **1.**

3. The LOAD key is depressed, thereby causing the magnetic card reading and recording unit to begin reading the first recorded side of the magnetic card and turning on an INSERT CARD indicator light **19** positioned immediately below and between the LOAD and RECORD program keys. This program reading operation will terminate and the INSERT CARD indicator light will turn off when an END (i.e. terminating) program step is read. The calculator will thereupon revert to the original manual operating or first key-log printing mode with the user program counter specifying the address of the END program step. In any case, the magnetic card will be partially ejected at an output receptacle **188** of the magnetic card reading and recording unit after each reading pass has been completed.

4. If the INSERT CARD indicator light remains on and the magnetic card reading and recording unit continues to run, the partially ejected magnetic card is retrieved from output receptacle **188,** turned around, and the same magnetic card (or another magnetic card, as appropriate) inserted into input receptacle **186** with the next side **189** to be read placed in the operative reading and recording position. The magnetic card reading and recording unit thereupon begins reading this next recorded side. This program reading operation is repeated, if necessary, until it is terminated by reading an END program step. If desired, program reading operation may also be terminated by depressing the STOP key. The calculator will thereupon also revert to the original manual operating or first key-log printing mode, but with the user program counter specifying the address of the next program step to have been read and loaded into the program storage section of the RWM had the program reading operation not been so terminated.

When the program reading operation is terminated by an END program step, the user program counter is left specifying the address of this END program step so that additional program steps, subprograms, and programs may be chain-loaded into the calculator without extra effort by simply repeating steps 2, 3, and 4 above. The first additional program step will over-write the last encountered END program step thereby leaving only a final END program step at the completion of the composite program. Similarly, when the program reading operation is terminated by depressing the STOP

key, the user program counter is also left specifying the address of the next program step to have been read so that additional program steps, subprograms, and programs may also be chain-loaded by simply repeating steps 2, 3, and 4 above.

The RECORD program key is depressed, when the calculator is in the manual operating mode or the first key-log printing mode, to place the calculator in the magnetic card recording mode. During this mode program steps stored in the program storage section of the RWM are recorded on one or more external magnetic cards by the magnetic card reading and recording unit. In order to properly accomplish this program recording operation:

1. The user program counter is set to the starting address nnnn of the stored program to be recorded. This may be accomplished by depressing the GO TO key followed by the decimal digit keys designating the required address nnnn, or simply by depressing the END key if the starting address is **0000.**

2. A magnetic card **16** is inserted into input receptacle **186** of the magnetic card reading and recording unit with the first side **187** of the card to be recorded placed in the operative reading and recording position as shown.

3. The RECORD program key is depressed, thereby causing the magnetic card reading and recording unit to begin recording the stored program onto the first side of the magnetic card and turning on the INSERT CARD indicator light **19.** This program recording operation will terminate and the INSERT CARD indicator light turn off when an END program step is recorded. The calculator will thereupon revert to the original manual operating or first key-log printing mode with the user program counter specifying the address of the END program step. In any case, the magnetic card will be partially ejected at output receptacle **188** of the magnetic card reading and recording unit after each recording pass has been completed.

4. If the INSERT CARD indicator light remains on, and the magnetic card reading and recording unit continues to run, the partially ejected card is retrieved from output receptacle **188,** turned around, and the same magnetic card (or another magnetic card, as appropriate) inserted into input receptacle **186** with the next side **189** to be recorded placed in the operative reading and recording position. The magnetic card reading and recording unit thereupon begins recording on this next side. This program recording operation is repeated, if necessary, until it is terminated by recording an END program step. If no END program step is encountered, the calculator will continue to request more magnetic card recording passes until 2,036 program steps are recorded, regardless of the actual amount of program storage memory installed in the calculator. If desired, the recording operation can be terminated at any time by depressing the STOP key. The calculator will thereupon also revert to the original manual operating or first key-log printing mode but with the user program counter left specifying the address of the next program step to have been recorded had the recording operation not been so terminated.

Once the program is recorded on one or both sides of one or more magnetic cards **16,** it may be protected against undesired erasures by punching out a perforated portion **190** at the leading edge of each side on which it is recorded. If a protected (notched) magnetic card is inserted into input receptacle **186** of the magnetic card reading and recording unit and the RECORD key depressed, the STATUS (error) indicator light **19** will turn on and will remain on while the magnetic card reading and recording unit drives the magnetic card to output receptacle **188,** whereupon the STATUS light will be extinguished. Nothing will be recorded on the protected magnetic card during this recording pass nor will there be any impairment of the calculator itself or the information previously recorded on the protected magnetic card. The calculator and the magnetic card unit will simply continue to wait for a nonprotected magnetic card to be inserted into input receptacle **186** for the magnetic card reading and recording unit.

Automatic Operating Mode Control Keys (CONTINUE, STOP, END, PAUSE)

The CONTINUE key is depressed, when the calculator is in the manual operating mode or the first key-log printing mode, to start the automatic execution of a program or subprogram stored within the program storage section of the RWM. Automatic execution begins at the address specified by the user program counter. Thus, in order to execute a desired program or subprogram stored within the program storage section of the RWM:

1. The user program counter is set to the starting address nnnn of the desired program or subprogram. This is accomplished by depressing the GO TO key followed by those decimal digit keys designating the starting address nnnn of the desired program. If the starting address is **0000,** this may be accomplished by simply depressing the END key.

2. The CONTINUE key is depressed, when the calculator is in the manual operating mode or the first key-log printing mode, to place the calculator in the automatic operating mode and begin automatic execution of the stored program or subprogram at the address nnnn specified by the user program counter. Automatic execution will continue until a STOP or END program step is encountered or a STOP key is depressed as described below.

The STOP key is depressed, when the calculator is in the automatic operating mode, to halt the automatic execution of a stored program or subprogram immediately after completion of the program step then being executed. Automatic execution of a stored program or subprogram is similarly halted when a STOP program step is encountered. In either case the calculator will thereupon revert to the original manual operating or first key-log printing mode with the program counter specifying the address of the next program step to be encountered. The user may then operate the calculator in the manual operating mode to enter data required by the program being executed or to perform other calculations. So long as the user does not depress the GO TO or END key or perform some other operation altering the last setting of the user program counter, he may resume automatic operation of the calculator at any time by simply depressing the CONTINUE key again.

As described above, the STOP key may also be depressed, when the calculator is in the program-list printing mode, the magnetic card reading mode, or the magnetic card recording mode to halt the program-list printing operation, the magnetic card reading operation or the magnetic card recording operations, respectively. In each of these cases the calculator will revert

to the mode it was in immediately prior to the halted operation with the user program counter specifying the address of the next program step to have been printed, read, or recorded had the STOP key not been depressed.

The END key is depressed, when the calculator is in the manual operating mode, to set the user program counter to address **0000** in the program storage section of the RWM (this is equivalent to depressing the GO TO, **0, 0, 0,** and **0** keys). An END program step terminates the automatic execution of a stored program by the calculator and resets the user program counter to the first available address **0000** in the program storage section of the RWM. The calculator thereupon reverts to the original manual operating or first key-log printing mode, from which automatic operation was initiated. Automatic execution may then be resumed by depressing the CONTINUE key, if the desired starting address is **0000** or by repeating steps 1 and 2 described above in connection with the CONTINUE key if the desired starting address is not **0000.** An END program step also clears any subroutine return-address to which return has not by then been made. As described above, it also terminates the program listing, magnetic card reading, and magnetic card recording operations.

The PAUSE key is typically used only as a program step. Automatic execution of a stored program or subprogram is automatically halted for a ¼ second pause interval whenever a PAUSE program step is encountered or the PAUSE key is depressed. This enables partial results of a calculation to be displayed by the output display unit during automatic execution of the stored program or subprogram. Successive PAUSE program steps may be employed to increase the duration of the pause interval by ¼ second increments. Automatic execution of the stored program or subprogram automatically resumes after the pause interval.

A PAUSE program step may also be used as a conditional step, permitting the user to stop the automatic operation of the calculator immediately after execution of any PAUSE program step. This is accomplished by simply depressing any key (other than STOP) during automatic execution of the program until the PAUSE program step has been executed. In other words, a PAUSE program step immediately followed by depression of any key other than STOP has the same effect as depressing STOP, with the advantage that automatic execution of a stored program or subprogram may thereby be precisely halted at one or more predetermined program steps if the user so desires. Automatic execution of the stored program may then be resumed by depressing the CONTINUE key.

Decimal Display Keys (FLOAT, FIX ( ))

These keys are employed to control the format of numbers displayed by the output display unit when the calculator is in the manual operating and first key-log printing modes. Either a fixed or a floating decimal point format may be used. A fixed point indicator light **19** positioned immediately below the FIX ( ) key is turned on when the fixed decimal point format is being used. Similarly, a floating point indicator light positioned immediately below the FLOAT key is turned on when the floating decimal point format is being used.

In the fixed decimal point format, numbers appear in the form in which they are most commonly written. The decimal point is fixed in its correct position. In the floating decimal point format, numbers appear in a normalized form with the decimal point located immediately after the most significant non-zero digit of the normalized number. Each normalized number is followed by an exponent comprising a positive or negative power of 10 and representing the number of digit places that, and the direction in which, the decimal point must be moved to express the normalized number in the fixed decimal point format. The following examples illustrate the relationship between numbers expressed in both the fixed and floating decimal point formats.

FIXED DECIMAL POINT FORMAT		FLOATING DECIMAL POINT FORMAT
1234.5	=	1.2345×10^{3}
0.0012345	=	1.2345×10^{-3}
−1.2345	=	-1.2345×10^{0}

When the calculator is turned on and automatically initialized, the output display unit displays numbers in the floating decimal point format. If the output display is switched to the fixed decimal point format, it may subsequently be switched back to the floating decimal point format by simply depressing the FLOAT key. Every number displayed in the floating decimal point format includes the sign (if negative) and 10 most significant digits of the normalized number followed by the sign (if negative) of the exponent and two exponent digits. This is illustrated by the following examples:

NUMBERS TO BE DISPLAYED	FLOATING DECIMAL POINT DISPLAY	
1234.5	1.234500000	03
0.0012345	1.234500000	−03
−1.2345	−1.234500000	00

The output display may be changed to the fixed decimal point format by depressing the FIX ( ) key followed by a decimal digit key designating the desired number *n* (**0–9**) of digits to be displayed to the right of the decimal point. Less significant digits are not displayed, and the least significant digit to be displayed is rounded up if the nondisplayed next least significant digit is five or greater. This is illustrated by the following examples for different values of *n*:

NUMBERS TO BE DISPLAYED	VALUES OF *n*	FIXED DECIMAL POINT DISPLAY
123.456784	2	123.46
−6.703256	2	−6.70
123.456784	5	123.45678
−6.703256	5	−6.70326
123.456784	0	123.
−6.703256	0	−7.

If the FIX ( ) key is depressed but not followed by a decimal digit key, the calculator will automatically treat the next key depressed as being the **0** key (i.e., *n* will thereupon equal 0 as in the last example).

Numbers of up to (and including) 10 significant digits and their signs (if negative) may be displayed in the fixed decimal point format. Thus, (10−*n*) digits may be displayed to the left of the decimal point. If a number to be displayed has more than (10−*n*) digits to the left

of the decimal point (i.e. is too large for the selected value of n), the display of that number (not the whole display) overflows and thereupon reverts to the floating decimal point format. This is illustrated by the fixed decimal point display of FIG. **1** wherein the numbers contained in the x and y registers have overflowed and are therefore displayed in the floating decimal point format. It is also illustrated by the following example:

NUMBER TO BE DISPLAYED	VALUE OF n	ACTUAL NUMBER DISPLAYED
1234567.89	5	1.23456789 06

If the first non-zero digit of a number to be displayed is more than n digits to the right of the decimal point (i.e. is too small for the selected value of n), the display of that number (not the whole display) underflows. In this case only zeros will be displayed. This is illustrated by the following example:

NUMBER TO BE DISPLAYED	VALUE OF n	ACTUAL NUMBER DISPLAYED
0.0000012	3	0.000

Regardless of the way in which numbers are displayed, the calculator always stores all numbers and performs all calculations in the floating decimal point format. Furthermore, regardless of the number of digits entered or displayed, each number is stored with 12 significant digits, their associated sign, a two-digit exponent, and its associated sign. Up to (and including) 10 significant digits, their associated sign, the two-digit exponent, and its associated sign can be displayed (however, no sign is displayed for positive numbers or exponents). The remaining two digits (called guard digits) are not displayed. They are employed to maintain greater than 10-place accuracy during calculations and also to automatically round the tenth displayed digit.

The calculator has a dynamic range of from $\pm 10^{-98}$ to $\pm 9.999999999(99) \times 10^{98}$. Whenever this range is exceeded during a calculation the STATUS indicator light **19** turns on.

Numeric Data Entry Keys (**0–9**), ., ENTER EXP, CHG SIGN, π)

The decimal digit keys **0–9** are depressed to enter numbers into the x-register. Numbers are entered serially, the last digit entered becoming the least significant digit. For example, the number 1325 is entered by sequentially depressing the decimal digit keys **1, 3, 2,** and **5.** A number entered into the x-register is terminated as soon as any non-data-entry key is depressed. Another number entered into the x-register will automatically replace a terminated number, but will become a part of any non-terminated number.

The decimal point (.) key is depressed to enter the decimal point into the x-register. For example, the number 1.234 is entered by sequentially depressing the **1, ., 2, 3,** and **4** keys. Regardless of the display format, it is not necessary to use the decimal point key when entering integers. If the decimal point key is not used, the decimal point will be assumed to have followed the last-entered digit. When the fixed decimal point display format is used, the calculator automatically positions the decimal point. Similarly, when the floating decimal point display format is used the calculator automatically corrects the exponent according to the position of the decimal point.

The ENTER EXP key is depressed followed by one or two decimal digit keys to enter a one- or two-digit exponent (power of 10) into the x-register, the last digit entered becoming the least significant digit. If a third digit is entered, it will terminate entry of the exponent and begin a new numeric data entry. A non-terminated number in the x-register may be multiplied directly by successive powers of ten by simply entering successive exponents. For example, the product of $8.3 \times 10^2 \times 10^{14}$ may be obtained by sequentially depressing the **8, ., 3,** ENTER EXP, **2,** ENTER EXP, **1,** and **4** keys. If the ENTER EXP key is depressed as the first key of a numeric data entry (i.e. before any of the decimal digit keys have been depressed or following a terminated number), the number 1 is entered into the x-register. For example, the number 1×10^{16} may be entered by depressing the ENTER EXP, **1,** and **6** keys.

The CHG SIGN key is depressed to change the sign of any terminated or unterminated number in the x-register. If the CHG SIGN key is depressed as the first key of a numeric data entry, it changes the sign of the number then in the x-register (whatever the sign may be) and, once that number is replaced by the first digit of the new data entry, it prefaces the new data entry with a negative sign. This is illustrated by the following example:

KEYS DEPRESSED	CONTENTS OF x-Register
	−123.45
CHG SIGN	123.45
6	−6.
7	−67.

The CHG SIGN key is also depressed immediately following the ENTER EXP key (or the last-entered digit of the exponent) to enter negative exponents into the x-register. For example, the number $8.3 \times 10^{-2} \times 10^4 \times 10^{-12}$ may be entered by sequentially depressing the keys **8, ., 3,** ENTER EXP, CHG SIGN, **2,** ENTER EXP, **4,** ENTER EXP, CHG SIGN, **1,** and **2** keys and the number 1×10^{-16} may be entered by depressing the ENTER EXP, **1, 6,** and CHG SIGN keys.

The π key is depressed to enter the value of π (i.e. 3.14159265360) into the x-register.

CLEAR keys (CLEAR x, CLEAR)

The CLEAR x key clears (i.e. sets to zero) the x-register. It does not affect any other registers.

The CLEAR key clears the x-, y-, and z- (working) registers, clears the a- and b- (data storage) registers, and clears (or resets) the flag, which can be set by the SET FLAG key as hereinafter explained. It does not affect any other registers.

When the calculator is switched on, all of the working, program, and data storage registers of the RWM are automatically cleared. Any terminated numbers subsequently stored in them will automatically be cleared and replaced by any new data entry.

Working Register Control Keys (↑, ↓, ROLL ↑, x⇄y)

The working register control keys are used to reposition the contents of the *x*-, *y*-, and *z*-registers. They do not affect any other registers.

When the ↑ key is depressed, the contents of the *y*-register shift to the *z*-register and the contents of the *x*-register appear in both the *x*- and *y*-registers. The contents of the *z*-register are lost.

When the ↓ key is depressed, the contents of the *y*-register shift to the *x*-register and the contents of the *z*-register appear in both the *y*- and *z*-registers. The contents of the *x*-register are lost.

When the ROLL ↑ key is depressed, the contents of the *x*-register shift to the *y*-register, the contents of the *y*-register shift to the *z*-register, and the contents of the *z*-register shift to the *x*-register. No information is lost.

The $x \rightleftarrows y$ key is depressed to exchange the contents of the *x*- and *y*-registers. The contents of the *z*-register are unaffected by this operation.

Arithmetic Keys (+, −, ×) ÷)

These four keys are used to perform working register arithmetic operations in which the contents of the *x*- and *y*-registers are employed as operands. The results of these arithmetic operations are stored in the *y*-register and the contents of the *x*-register remain unchanged by the arithmetic operations performed. These four keys do not affect any other registers.

The + key is depressed to add the number in the *x*-register to the number in the *y*-register, the sum appearing in the *y*-register.

The − key is depressed to subtract the number in the *x*-register from the number in the *y*-register, the difference appearing in the *y*-register.

The × is depressed to multiply the number in the *y*-register by the number in the *x*-register, the product appearing in the *y*-register.

The ÷ key is depressed to divide the number in the *y*-register by the number in the *x*-register, the quotient appearing in the *y*-register.

The use of these keys and the working register control keys is illustrated by the following method of computing

$$[(3 \times 4) + (8 - 9)]/[(8 \times 2) - 6] = 1.1 :$$

KEYS DEPRESSED	CONTENTS OF *x*-REGISTER	CONTENTS OF *y*-REGISTER	CONTENTS OF *z*-REGISTER
CLEAR	0	0	0
3	3	0	0
↑	3	3	0
4	4	3	0
×	4	12	0
ROLL↑	0	4	12
8	8	4	12
$x \rightleftarrows y$	4	8	12
9	9	8	12
−	9	−1	12
↓	−1	12	12
+	−1	11	12
ROLL↑	12	−1	11
8	8	−1	11
$x \rightleftarrows y$	−1	8	11
2	2	8	11
×	2	16	11
6	6	16	11
−	6	10	11
↓	10	11	11
÷	10	1.1	11

The answer, appearing in the *y*-register, is underlined.

Unary Function Keys ($\sqrt{x}$, x^2, $1/x$, int *x*, CHG SIGN)

These five keys are used to perform unary functions in which the contents of the *x*-register are employed as the argument. The results of the unary functions performed are placed in the *x*-register. These five keys do not affect any other registers.

The $\sqrt{x}$ key is depressed to calculate the square root of the number in the *x*-register. If the number is negative, the square root of its absolute value is calculated and the STATUS indicator light **19** turned on. The STATUS indicator light remains on until the next key is depressed.

The x^2 key is depressed to calculate the square of the number in the *x*-register. If the number is greater than $\sqrt{10} \times 10^{49}$, the number $9.99999999999 \times 10^{98}$ is placed in the *x*-register and the STATUS indicator light **19** turned on. The STATUS indicator light remains on until the next key is depressed.

The $1/x$ key is depressed to calculate the reciprocal of the number in the *x*-register. For example, 1/9.8 may be calculated by sequentially depressing the **9**, **.**, **8**, and $1/x$ keys.

The int *x* key is depressed to truncate the fractional part of the number in the *x*-register. It does not affect the sign of the integer part of the number. For example, if the number −5.9 is contained in the *x*-register when the int *x* key is depressed, the number −5.0 will remain.

The CHG SIGN key has already been described above in connection with the numeric data entry keys.

Data-Storage and Register-Transfer Keys (*a*, *b*, *x* → (), y → (), x ← (), $y \rightleftarrows$(), INDIRECT)

These keys are variously used to perform direct data storage and recall, direct storage-register arithmetic, indirect data storage and recall, and indirect storage-register arithmetic operations. As illustrated below, the *a* and *b* keys are depressed following any of the remaining five keys of this group to specify the *a*- and *b*-registers, respectively. The *a* and *b* keys may also be used to directly recall the contents of the *a*- and *b*-registers, respectively, to the *x*-register without changing the contents of the *a*- and *b*-registers themselves. Either of these functions of the *a* and *b* keys may be performed in response to a single keystroke thereof. For example, the contents of the *b*-register may be directly recalled to the *x*-register by simply depressing the *b* key alone. As noted above the contents of the *b*-register itself will remain unchanged by this recall operation.

The *x* →(), *y* →(), *x* ←(), *and* $y \rightleftarrows$() keys are used to control the transfer of numeric data from the *x*- and *y*-registers to the *a*- and *b*-registers and 49 additional storage registers available to the user (at user addresses **000–048**) in the data storage section of the RWM and from these storage registers to the *x*- and *y*-registers. If the 60 optional storage registers included in the data storage section of the RWM (at user addresses **049–108**) are made available to the user, these same four keys may also be used to control the transfer of numeric data from the *x*- and *y*-registers to these optional storage registers and from them to the *x*- and *y*-registers.

The specific storage register to or from which numeric data is transferred by these four keys is specified by the key or keys depressed immediately following them. Accordingly, the a key is used to specify the *a*-register, the *b* key to specify the *b*-register, and the decimal digit keys **0–9** to selectively specify any of the available storage registers at user addresses **000–108** of the data storage section of the RWM. For simplicity of description, the available storage registers at user ad-

dresses **000–108** will hereinafter be referred to by their addresses (i.e. the storage register at any available address *nnn* will be referred to as register *nnn*). If a nonexistant or nonavailable storage register is designated, the STATUS indicator light is turned on, no data transfer or arithmetic operation is performed, and the operation of the calculator halts. (If such a register is designated while the calculator is automatically executing a stored program, the program-counter specifies the program step immediately following the improper operation.)

The $x \rightarrow ( )$ key is depresssed followed by the *a* key, the *b* key, or decimal digit keys *n*, *n*, *n* to directly store the contents of the *x*-register in the *a*-register, the *b*-register, or register nnn respectively. In any case, the contents of the *x*-register remain unchanged by this operation. For example, π may be stored directly in the *b* register by sequentially depressing the π, $x \rightarrow ( )$, and *b* keys. Similarly, π may be stored directly in register **027,** by sequentially depressing the π, $x \rightarrow ( )$, **0, 2,** and **7** keys. In each of these examples, π will also remain in the *x*-register.

The $y \rightarrow ( )$ key is depressed followed by the *a* key, the *b* key, or decimal digit keys *n*, *n*, *n* to directly store the contents of the *y*-register in the *a*-register, the *b*-register, or register *nnn*, respectively. In any case, the contents of the *y*-register remain unchanged by this operation. For example, a number contained in the *y*-register may be stored directly in register **000,** without changing the contents of the *y*-register, by sequentially depressing the $y \rightarrow ( )$, **0, 0,** and **0** keys.

The $x \leftarrow ( )$ key is depressed followed by the *a* key, the *b* key, or decimal digit keys *n*, *n*, *n*, to directly recall the contents of the *a*-register, the *b*-register, or register *nnn*, respectively, to the *x*-register. In any case, the contents of the recalled register remain unchanged by this operation. For example, the contents of register **012** may be recalled to the *x*-register, without changing the contents of register **012,** by sequentially depressing the $x \leftarrow ( )$, **0, 1,** and **2** keys. Recall from the *a*- or *b*-register to the *x*-register may be accomplished in the same manner as described above, by simply depressing the *a* or *b* key alone.

The $y \rightleftarrows ( )$ key is depressed followed by the *a* key, the *b* key, or decimal digit keys *n*, *n*, *n*, to exchange the contents of the y-register with the contents of the *a*-register, the *b*-register, or register *nnn*, respectively. For example, a number in the *y*-register may be exchanged with a number in register **048** by depressing the $y \rightleftarrows ( )$, **0, 4,** and **8** keys.

The direct storage and recall operations performed by the $x \rightarrow ( )$, $y \rightarrow ( )$, $x \leftarrow ( )$, and $y \rightleftarrows ( )$ keys may be conveniently summarized by employing the following notation:

$$\left\{\begin{matrix} x \rightarrow ( ) \\ y \rightarrow ( ) \\ x \leftarrow ( ) \\ y \rightleftarrows ( ) \end{matrix}\right\} \qquad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

In this notation each pair of braces implies that any one of the enclosed group of key operations or storage registers may be selected. The order of the successive pairs of braces from left to right specifies the key-sequence required to perform the selected key operation with the selected storage register. Thus, any of the key operations enclosed in the left-hand pair of braces may be employed with any of the storage registers enclosed in the righthand pair of braces by first depressing the key performing the selected key operation and then depressing the key or keys designating the selected storage register. The usefullness of these direct storage and recall operations is illustrated by the following method of multiplying a series of numbers n_1, n_2, etc. by a constant K, where $n_1 = 3$, $n_2 = 11.2$, etc. and K = 1.684:

KEYS DEPRESSED	CONTENTS OF *x*-REGISTER	CONTENTS OF *y*-REGISTER	CONTENTS OF *a*-REGISTER
CLEAR	0	0	0
1, 0, 6, 8, 4,	1.684	0	0
$y \rightleftarrows ( ), a$	1.684	0	1.684
3	3	0	1,684
↑	3	3	1,684
a	1.684	3	1.684
×	1.684	5.0520	1.684
1, 1, ., 2	11.2	5.0520	1.684
↑	11.2	11.2	1.684
a	1.684	11.2	1.684
×	1.684	18.8608	1.684

The answers, appearing in the *y*-register, are underlined.

The $x \rightarrow ( )$, $y \rightarrow ( )$, $x \leftarrow ( )$, and $y \rightleftarrows ( )$ keys may also be employed with the +, −, ×, and ÷ arithmetic keys to perform direct register-arithmetic operations in which the contents of the *x*-register or the *y*-register are employed as one operand and the contents of the *a*-register, the *b*-register, or register *nnn* are employed as the other operand. These direct register-arithmetic operations may be summarized as follows by using the above-described notation:

$$\left\{\begin{matrix} x \rightarrow ( ) \\ y \rightarrow ( ) \\ x \leftarrow ( ) \\ y \rightleftarrows ( ) \end{matrix}\right\} \qquad \left\{\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}\right\} \qquad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

Thus, the $x \rightarrow ( )$ key may be employed to directly perform any of the arithmetic operations enclosed by the second pair of braces upon the contents of the *x*-register and the contents of any of the storage registers enclosed by the third pair of braces and store the result in the selected storage register without recalling the contents of the selected storage register and without changing the contents of the *x*-register. For example, the contents of the *x*-register may be subtracted from the contents of the *b*-register by sequentially depressing the $x \rightarrow ( )$, −, and *b* keys. The difference is stored in the *b*-register, and the contents of the *x*-register remain unchanged by the operation. Similarly, the contents of register **042** may be divided by the contents of the *x*-register by sequentially depressing the $x \rightarrow ( )$, ÷, **0, 4,** and **2** keys. The quotient is stored in register **042,** and the contents of the *x*-register itself remain unchanged by the operation.

The $y \rightarrow ( )$ key may be similarly employed to directly perform any of the arithmetic operations enclosed by the second pair of braces upon the contents of the *y*-register and any of the storage registers enclosed by the third pair of braces and store the result in the selected

storage register without recalling the contents of the selected storage register and without changing the contents of the *y*-register. For example, the contents of the *y*-register may be added to the contents of the *a*-register by sequentially depressing the $y \rightarrow$ (), +, and *a* keys. The sum is stored in the *a*-register, and the contents of the *y*-register remain unchanged by the operation. Similarly, the contents of register **038** may be multiplied by the contents of the *y*-register by sequentially depressing the $y \rightarrow$ (), *x*, **0**, **3**, and **8** keys. The product is stored in register **038**, and the contents of the *y*-register remain unchanged by the operation.

The $x \leftarrow$ () key may be employed to directly perform any of the arithmetic operations enclosed by the second pair of braces upon the contents of the *x*-register and the contents of any of the storage registers enclosed by the third pair of braces and store the result in the *x*-register without changing the contents of the selected storage register. For example, the contents of the *a*-register may be added to the contents of the *x*-register by sequentially depressing the $x \leftarrow$ (), +, and *a* keys. The sum is stored in the *x*-register, and the contents of the *a*-register remain unchanged by the operation. Similarly, the contents of the *x*-register may be multiplied by the contents of register **022** by sequentially depressing the $x \leftrightarrows$ (), *x*, **0**, **2**, and **2** keys. The product is stored in the *x*-register, and the contents of register **022** remain unchanged by the operation.

The $y \rightleftarrows$() key may be employed to directly perform any of the arithmetic operations enclosed by the second pair of braces upon the contents of the *y*-register and the contents of any of the storage registers enclosed by the third pair of braces and store the result in the *y*-register without changing the contents of the selected storage register (i.e. the $y \rightleftarrows$() may be used as though it were a $y \leftarrow$ () key in performing register-arithmetic operations).

For example, the contents of the *b*-register may be subtracted from the contents of the *y*-register by sequentially depressing the $y \rightleftarrows$(), −, and *b* keys. The difference is stored in the *y*-register, and the contents of the *b*-register remain unchanged by the operation. Similarly, the contents of the *y*-register may be divided by the contents of regiters **039**, by sequentially depressing the $y \rightleftarrows$(), ÷, **0**, **3**, and **3** keys. The quotient is stored in the *y*-register, and the contents of register **039** remains unchanged by the operation.

The INDIRECT key is used with the above-described data-storage and register-transfer keys to perform indirect data-storage and recall and indirect register-arithmetic operations, in which the contents of a directly-addressed register (e.g. the *a*-register, the *b*-register or any of the registers *nnn*) are employed as the address of an indirectly-addressed storage register (e.g. any of the other storage registers *nnn*) to be used in these operations. When indirectly addressing any of the storage registers **000** through **048** or **108**, care must be taken to insure that the directly-addressed register contains the proper address *nnn* of the indirectly addressed-register nnn. The indirect address used is the absolute value of the integer part of the contents of the directly addressed register. Thus, 1.732 will be treated as the address of register **001**, −6.99 as the address of register **006**, 0.999 as the address of register **000**, and 106.75 as the address of register **106**. Since the storage registers may only contain numeric data, the *a*- and *b*-registers may not be used as the indirectly-addressed registers in these operations.

The indirect data storage and recall operations may be summarized as follows by again using the above-described notation without braces for the INDIRECT key:

$$\left\{\begin{matrix} x \rightarrow () \\ y \rightarrow () \\ x \leftarrow () \\ y \rightleftarrows () \end{matrix}\right\} \text{ INDIRECT } \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

Thus, the $x \rightarrow$ () and $y \rightarrow$ () keys may be employed with the INDIRECT key to indirectly store the contents of the *x*- and *y*-registers, respectively in any storage register nnn designated by the contents of any of the other data storage registers. Moreover, this may be accomplished without changing the contents of the *x*- and *y*-registers or of the directly addressed storage register. For example, the contents of the *x*-register may be indirectly stored in the storage register designated by the contents of the *a*-register by sequentially depressing the $x \rightarrow$ (), INDIRECT, and *a* keys. The contents of the *x*-register and the *a*-register remain unchanged by this operation. Similarly, the contents of the *y*-register may be indirectly stored in the storage register designated by the contents of register **022** by sequentially depressing $y \rightarrow$ (), INDIRECT, **0**, **2**, and **2** keys. The contents of the *y*-register and register **022** remain unchanged by this operation.

The $x \leftarrow$ () key may be similarly employed with the INDIRECT key to indirectly recall to the *x*-register the contents of any storage register nnn designated by the contents of any of the other storage registers. This is accomplished without changing the contents of either the directly- or indirectly-addressed storage register. For example, the contents of the register *nnn* designated by the contents of the *b*-register may be indirectly recalled to the *x*-register by sequentially depressing the $x \leftarrow$ (), INDIRECT, and *b* keys. The contents of the b-register and of the indirectly-addressed register remain unchanged by this operation.

The $y \rightleftarrows$() key may be employed with the INDIRECT key to indirectly exchange the contents of the *y*-register with the contents of any storage register *nnn* designated by the contents of any of the other storage registers. This is accomplished without changing the contents of the directly-addressed storage register. For example, the contents of the *y*-register may be indirectly exchanged with the contents of the register designated by register **041** by sequentially depressing the $y \rightleftarrows$(), INDIRECT, **0**, **4**, and **1** keys. The contents of register **041** remain unchanged by this operation.

The indirect register-arithmetic operations may be summarized as follows:

$$\left\{\begin{matrix} x \rightarrow () \\ y \rightarrow () \\ x \leftarrow () \\ y \rightleftarrows () \end{matrix}\right\} \text{ INDIRECT } \left\{\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}\right\} \quad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

In connection with the indirect register arithmetic operations it should be noted that the INDIRECT key may also be depressed immediately after the selected

arithmetic key. Thus, the indirect register-arithmetic operations may also be summarized as follows:

$$\left\{\begin{matrix} x\rightarrow() \\ y\rightarrow() \\ x\leftarrow() \\ y\rightleftarrows() \end{matrix}\right\} \quad \left\{\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}\right\} \quad \text{INDIRECT} \quad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

Thus, any of the $x\rightarrow()$, $y\rightarrow()$, $x\leftarrow()$, and $y\rightleftarrows()$ keys may be employed with the INDIRECT key and any of the arithmetic keys to indirectly perform register-arithmetic operations employing the contents of either the *x*- or the *y*-register as one operand and the contents of any of the storage registers nnn designated by the contents of any of the other storage registers as the other operand. In the case of the $x\rightarrow()$ and $y\rightarrow()$ keys, the results of these operations are stored in the indirectly addressed storage register nnn, and the contents of the *x*- and *y*-registers and the directly-addressed registers are not changed. Similarly, in the case of the $x\leftarrow()$ and $y\rightleftarrows()$ keys, the results of these operations are stored in the *x*- and *y*-registers, respectively, and the contents of the directly and indirectly addressed registers are not changed.

For example, the contents of the *x*-register may be added to the contents of the register nnn designated by the contents of the *a*-register by sequentially depressing the $x\rightarrow()$, INDIRECT, +, and *a* keys. The sum is stored in the indirectly-addressed register nnn, and the contents of the *x*-register and the *a*-register remain unchanged by this operation. Similarly, the contents of the storage register nnn designated by the contents of register **014** may be multiplied by the contents of the *y*-register by sequentially depressing the $y\rightarrow()$, *x*, INDIRECT, **0**, **1**, and **4** keys. The product is stored in the indirectly-addressed storage register nnn, and the contents of the *y*-register and register **014** remain unchanged by the operation. Similarly, the contents of the register nnn designated by the contents of the *b*-register may be subtracted from the contents of the *x*-register by sequentially depressing the $x\leftarrow()$, INDIRECT, −, and *b* keys. The difference is stored in the *x*-register, and the contents of the *b* -register and the indirectly-addressed register remain unchanged by the operation. As a last example, the contents of the *y*-register may be divided by the contents of the register nnn designated by the contents of register **008** by sequentially depressing the $y\rightleftarrows()$, INDIRECT, ÷, **0**, **0**, and **8** keys. The quotient is stored in the *y*-register, and the contents of register **008** and the indirectly-addressed register *nnn* remain unchanged by this operation.

Multiple-level indirect addressing may be performed to any desired level in the above described indirect data storage and recall and indirect register arithmetic operations by sequentially depressing the INDIRECT key once for each desired level. Multiple-level indirect data storage and recall may be summarized as follows:

$$\left\{\begin{matrix} x\rightarrow() \\ y\rightarrow() \\ x\leftarrow() \\ y\rightleftarrows() \end{matrix}\right\} \quad \text{INDIRECT ....INDIRECT} \quad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

Thus, for example, the contents of the *x*-register may be indirectly stored in the contents of register **017** designated by the contents of register **003** in turn designated by the contents of the *a*-register by sequentially depressing the $x\rightarrow()$, INDIRECT, INDIRECT, and *a* keys. The contents of the *x*-register, the *a*-register, and register **003** remain unchanged by this operation. Similarly, the contents of register **046** designated by the contents of register **031** in turn designated by the contents of register **000** in turn designated by the contents of register **025** may be exchanged with the contents of the *y*-register by sequentially depressing the $y\rightleftarrows()$, INDIRECT, INDIRECT, INDIRECT, **0**, **2**, and **5** keys. The contents of registers **025**, **000** and **031**, remain unchanged by this operation.

Multiple level indirect register-arithmetic may be similarly summarized as follows:

$$\left\{\begin{matrix} x\rightarrow() \\ y\rightarrow() \\ x\leftarrow() \\ y\rightleftarrows() \end{matrix}\right\} \quad \text{INDIRECT...INDIRECT} \quad \left\{\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}\right\} \quad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

OR

$$\left\{\begin{matrix} x\rightarrow() \\ y\rightarrow() \\ x\leftarrow() \\ y\rightleftarrows() \end{matrix}\right\} \left\{\begin{matrix} + \\ - \\ \times \\ \div \end{matrix}\right\} \quad \text{INDIRECT...INDIRECT} \quad \left\{\begin{matrix} a \\ b \\ nnn \end{matrix}\right\}$$

Thus, assuming, for example, that the 60 optional storage registers **049–108** are available to the user, the contents of the *y*-register may be subtracted from the contents of register **108** designated by the contents of register **082**, in turn designated by the contents of register **049** by sequentially depressing the $y\rightarrow()$, INDIRECT, INDIRECT, −, **0**, **4**, and **9** keys. The difference is stored in register **108** and the contents of the *y*-register, register **049**, and register **082** remain unchanged by this operation. Similarly, the contents of the *x*-register may be divided by the contents of register **106** designated by the contents of register **056** in turn designated by the contents of the *b*-register by sequentially depressing the $x\leftarrow()$, ÷, INDIRECT, INDIRECT, INDIRECT, and *b* keys. The results are stored in the *x*-register, and the contents of the *b*-register and registers **003**, **056**, and **106** remain unchanged by this operation.

Numeric Address Termination

As described above, each of the available numerically-addressed storage registers **000** through **048** or **108** employed in the foregoing data-storage and register-transfer operations is properly addressed by selectively depressing the decimal digit keys to specify its three-digit address. Upon entry of the third digit, the numeric address is automatically terminated. Any immediately following digit entries are therefore not intrepreted as part of the numeric address, but rather as the beginning of a new data entry. Thus, for example, if the $x\rightarrow()$, **1**, **0**, **3**, **2**, and **5** keys are sequentially depressed, the contents of the *x*-register are stored in register **103**, and the number **25** is entered into the *x*-register. Similarly, if the $y\rightarrow()$, +, **0**, **0**, **2**, **1**, and + keys are sequentially depressed, the contents of the *y*-register are added to the contents of register **002**, and the number 1 is entered into the *x*-register and thereupon added to the contents of the *y*-register.

A numeric address may also be terminated, without entering leading zeros of the address, by depressing any non-numeric key except the STEP PRGM key or, if the calculator is in the manual operating or first key-log printing mode, the CONTINUE key. However, most of the terminating key entries that may be used will also be executed. For example, if the $x \rightarrow ( )$, **2,** and + keys are sequentially depressed, the contents of the *x*-register are stored in storage register **002** and also added to the contents of the *y*-register. Similarly, if the $y \rightarrow ( )$, +, **3, 8,** and *a* keys are sequentially depressed, the contents of the *y*-register are added to the contents of register **038,** and the contents of the *a*-register are recalled to the *x*-register.

When the calculator is in the manual operating or first key-log printing mode, the STOP key may be used to terminate a numeric address if a no-operation address-terminating key entry is desired. For example, by simply depressing the $x \rightarrow ( )$, **3,** and STOP keys the contents of the *x*-register may be stored in register **003** without performing any other operation. Similarly, when formulating or entering a program by which the calculator is to be controlled in the automatic operating mode, a CONTINUE program step may be employed to terminate a numeric address if a no-operation address-terminating program step is desired. For example, if the program steps $y \rightarrow ( )$, *x*, **2, 9,** CONTINUE, and **3** are sequentially encountered when the calculator is in the automatic operating mode, the contents of register **29** are multiplied by the contents of the *y*-register, and the product is stored in register **29.** No operation is performed by the CONTINUE program step, and the next program step **3** is stored in the *x*-register as the beginning of a new data entry.

Since every numeric storage register except optional registers **100–108** may be uniquely specified by less than three digits, these abbreviated address-termination features permit significant reductions in the number of key-operations and program steps required to perform many calculations. Moreover, these same address termination features may be used in connection with the four-digit numeric addresses of the program storage section of the RWM to achieve still further reductions in the number of key-operations and program-steps required to perform many calculations. Program-Control Keys (GO TO, IF $x<y$, IF $x=y$, IF $x>y$, IF FLAG,

SET FLAG, LABEL, SUB/RETURN)

All of the previously-described keys except the FLOAT, FIX(), RUN, PRGM, KEY LOG, LIST, LOAD, and RECORD keys may be employed both for controlling the operation of the calculator during the manual operating and first key-log printing modes and for entering program steps into the calculator during the program entering and second key-log printing modes. When the calculator is in the manual operating mode, the user may continuously observe the output display of the contents of the *x*-, *y*-, and *z*-registers and, in accordance with his observations, make his own decisions about what to do next at any stage of the calculation. However, this is not possible when the calculator is executing a stored program at high speed in the automatic operating mode. The above eight program-control keys have therefore been provided to permit the calculator itself to test calculated quantities and make decisions based on those tests during the automatic execution of an internally-stored program. These eight keys may be used to permit unconditional branching (or transfers), conditional branching (or transfers), symbolic or labelled branching, storage of "yes-no" information is a flag and conditional branching based on that "yes-no" information at a later stage in the execution of a program, and unconditional or conditional branches to pre-defined program routines with return to the main program sequence upon completion.

Branching instructions in a program cause the user program counter to specify an address other than the next sequential address in the program storage section of the RWM, whereupon execution of the program continues sequentially from the new address. If a branch is conditional, the calculator makes a decision, based upon a specified condition, whether or not to branch. However, if the branch is unconditional, the calculator has no option, and must branch to the address specified in the program (e.g. by a GO TO program step followed by a numeric address nnnn).

The GO TO key is depressed followed by decimal digit keys specifying a selected four-digit address nnnn in the program storage section of the RWM to set the user program counter to the selected address *nnnn.* When this sequence of keycodes is encountered as a sequence of program steps, an unconditional branch is made to the address indicated and the program step stored at that address is executed. Execution of the program then automatically continues to run from that address.

For simplicity of description the terms "keys" and "program steps" will hereinafter be used synonomously since the remaining keys of this group are used almost exclusively to enter keycodes into the calculator as program steps during the program entering mode of the calculator. All of the program steps so entered are automatically executed during the automatic operating mode of the calculator.

The four IF keys are used for conditional branching. The IF $x<y$, IF $x=y$, and IF $x>y$ keys compare the numeric values contained in the *x*- and *y*-registers to determine if the number in *x* is less than the number in *y*, equal to the number in *y*, or greater than the number in *y* respectively. The IF FLAG key tests the condition (yes-no) of the flag, which is controlled by the SET FLAG key hereinafter explained. There can be only two possible results to each test made by each of these four keys. The condition tested is either "met" (YES) or "not met" (NO).

When the condition tested is met (YES), the next program step following the IF is automatically executed. However, when the condition tested is not met (NO), the calculator automatically skips (ignores) the next four program steps and continues execution at the fifth program step following the IF.

If the program steps immediately following the IF constitute a numeric address nnnn and the condition tested is met (YES), an automatic branch is made to that address nnnn. The program step stored at address nnnn is thereupon executed, and automatic execution of the program continued from there. If the program steps immediately following the IF constitute operations (e.g. +, ↑, etc.) then no branch occurs and the operations are executed.

When an IF program step (other than IF FLAG) is encountered, the two numbers in the *x*- and *y*-registers are automatically rounded before the test is made. In

each register, the tenth digit of the number is rounded according to the value of the guard digits; the guard digits are then set equal to zero. Thus the numbers to be tested actually have the same values as would appear in a floating point display with all 10 significant digits displayed. After the test has been made, the numbers in the x- and y-registers retain their rounded values. This means that the actual values of the number in the x- and y-registers may be different after the test than before. If the resulting slight loss in accuracy is undesirable, the numbers in the x- and y-registers can be stored in the data-storage section of the RWM before the IF test is made and can be recalled and substituted for the rounded numbers after the IF test is completed.

The use of the IF $x>y$ and IF $x<y$ keys is illustrated by employing the following sequence of program steps beginning with an IF $x>y$ program step and including a three-step conditional routine for taking the absolute value of the contents of the y-register:

conditional routine for $|y|$

IF $x>y$
$x \rightleftarrows y$
CHG SIGN
$x \rightleftarrows y$
CONTINUE
↑

4-step conditional sequence

If the contents of the x-register are greater in value than the contents of the y-register (i.e. condition met), then every program step in this sequence is executed and the absolute value of the contents of the y-register is calculated. As described above CONTINUE is a no-operation program step. It is used in this and the following examples to fill in the fourth program step to be skipped if the test condition is not met. If the contents of the x-register are equal to or greater in value than than contents of the y-register (i.e. the condition is not met), then the four program steps immediately following the IF $x>y$ program step are skipped and execution continues beginning with the ↑ program step. In this case the absolute value of the contents of the y-register is not calculated.

The use of the IF $x=y$ key is illustrated by employing the following sequence of program steps:

IF $x=y$
2
3
CONTINUE
CONTINUE
+

4-step conditional sequence

If the contents of the x-register equal the contents of the y-register in value (i.e. condition met), then every program step in this sequence is executed. This results in an automatic branch to execute the program step stored at user address **0023.** However, if the contents of the x-register do not equal the contents of the y-register in value (i.e. condition not met), the four program steps immediately following the IF $x=y$ program step are skipped and execution continues beginning with the + program step.

The use of the IF FLAG key is illustrated by the following sequence of program steps:

IF FLAG
4
1
CONTINUE
CONTINUE
SET FLAG

4-step conditional sequence

If the flag controlled by the SET FLAG key has been set (i.e. condition met), then every program step in this sequence is executed. This results in an automatic branch to execute the program step stored at user address **0041.** However, if the flag controlled by the SET FLAG key has not been set (i.e. condition not met), then the four program steps immediately following the IF FLAG program step are skipped and execution continues beginning at the SET FLAG program step by setting the flag.

The SET FLAG key establishes the condition to be tested by the IF FLAG key. The YES condition is established when a SET FLAG keycode is encountered either as a program step or as a keyboard entry. The NO condition is established by clearing the flag. This occurs automatically whenever the calculator is switched on or whenever a CLEAR or IF FLAG keycode is encountered either as a program step or as a keyboard entry. If the flag condition must be retained for use later, the program must include a SET FLAG program step in the sequence of program steps executed following the flag-clearing program step. The flag enables the user to select the conditions which will determine whether a conditional branch (or operation) is to be made.

The LABEL key allows relocatable symbolic addresses to be used within a program. A LABEL key immediately followed by any other programmable key (e.g. LABEL, π) except the END key serves as a symbolic address that may be inserted in a program immediately before any program step a user may wish to relocate independently of its absolute (numeric) address. This symbolic address is relocated by a search command comprising a GO TO key immediately followed by the symbolic address itself (e.g. GO TO, LABEL, π). In response to this search command, the user program counter is reset to the first user available address **0000** in the program storage section of the RWM and is sequentially incremented in a search operation until the symbolic address (e.g. LABEL, π) is found. The user program counter then specifies the next address which is the program step designated by the symbolic address.

If the search command came from the keyboard, the calculator waits for the next key to be depressed. However, if the search command came from the program, then execution automatically continues at the program step designated by the symbolic address. The keys of the symbolic address itself serve as no-operation codes and are ignored during execution of the program. This may be illustrated by the following sequence of program steps.

ADDRESS	PROGRAM STEP
0098	---
0099	GO TO
0100	LABEL

-Continued

ADDRESS	PROGRAM STEP
0101	÷
0102	etc.
.	.
.	.
.	.
.	.
0362	---
0363	LABEL
0364	÷
0365	↑
0366	etc.

The search for the symbolic address "LABEL, ÷" is initiated immediately after address **0101** is designated by the user program counter and starts at address **0000.** When the symbolic address is found, program execution continues with the ↑ program step at address **0365.** The ÷ program, step **0101** and **0364,** serving as part of the symbolic address are not executed.

A specific symbolic address cannot be used to specify more than one location at any one time. If it is, only the first (lowest order numeric address) specified will be valid (i.e. the point of transfer). Any number of different labels can be used at one time, limited to the number of keys available to follow the LABEL key.

If transfer to an undefined symbolic address occurs, the calculator will search all of program memory, and if the symbolic address is not found, will stop with the STATUS indicator light on. The program-counter will specify the next following program steps.

Branching to a symbolic addresss offers considerable advantages over branching to an absolute address. Programs with symbolic addresses can be stored anywhere in the program storage section of the RWM and can be easily moved and relocated because there are no absolute addresses to be changed. Also, any time a program is to be corrected (i.e. program steps changed, added or deleted), any absolute addresses must be checked in case they themselves must now be changed as a result of the corrections. This may entail substantial bookkeeping by the user. If symbolic addresses are used instead of absolute addresses, the corrections will not affect the symbolic addresses.

The main disadvantage of using symbolic addresses is that a search takes considerably more time (depending upon the location of the label in memory) than does a branch to an absolute address. Usually, this will have no significance because, in this case, "time" constitutes only a few thousandths of a second. Even if time is a significant factor, the user may still take advantage of symbolic addresses by writing his original program with symbolic addresses and once the program has been completely debugged, changing the symbolic addresses to the appropriate absolute addresses.

The SUB/RETURN key is used to transfer to (i.e. call) a subroutine (subprogram) and return from the subroutine to the point in the calling program where the transfer was initiated. Subroutines may be nested upto a depth of five. An attempt to nest to a depth of more than five is an error, stopping execution of the program and turning on the STATUS indicator light. Both the automatic initialization occuring at turn-on and the END key (given either as a program step or a keyboard entry) automatically reset the nesting to a depth of zero so that all five depths are then available.

A GO TO key followed by a SUB/RETURN key and either an absolute numeric or symbolic labelled address may be used in the calling program to (unconditionally) call a subroutine. The address used specifies the starting address of the subroutine. If a symbolic address is used, the first two program steps of the subroutine must also be the same symbolic address. An IF key followed by GO TO and SUB/RETURN keys and either an absolute numeric or symbolic labelled starting address may also be used in the calling program to conditionally call a subroutine. Execution of the steps of the subroutine starts automatically as soon as the subroutine is called. A SUB/RETURN key must be included as the last step to be executed in the subroutine. The RETURN causes a branch to the "return-address" in the calling program. The return address is always the address immediately following the last step used to call the subroutine. Execution of the calling program then continues automatically, starting at the return address.

Any number of subroutines may be called individually during a program. However, it is also possible to use more than one subroutine at one time. One subroutine can call a second subroutine which, in turn, can call a third subroutine, and so on. This multiple-calling is known as "nesting". The calculator can remember (store) from one to five return-addresses at a time so that the subroutines may be nested up to a depth of five. Returns are made on a "last-in, first-out" basis, the return always being made to the last return-address stored. As soon as the return is made, that return-address is forgotton (erased from storage) so that the previous address now becomes the "last" one. Thus the returning order is always the opposite of the calling order.

A program written as a subroutine may also be used as a "stand-alone" program. This is accomplished by depressing the END key to erase any return-addresses currently stored in the calculator and by not using the SUB/RETURN key when addressing the memory before the program is run.

Special Function Keys (FMT, PRINT/SPACE, PAPER)

The FMT (format) key is used to initiate special operations not otherwise defined and implemented by the other basic keys of the calculator. It is always used with other keys and, in effect, serves to re-define these other keys to implement the desired special operations. Several of these special operations associated with the FMT key are defined as part of the basic calculator. Others have been defined as part of associated plug-in ROM modules and are available only when the associated ROM modules are plugged into the calculator.

The command-sequences FMT, ↑ and FMT, ↓ are used to operate the X-Y plotter peripheral unit. They are part of the basic calculator. The X-Y plotter-input commands are PEN ↑, PEN↓, and *x* and *y* coordinates to which the pen-carriage is to be moved. The *x* and *y* coordinates are specified by decimal numbers in the range **0000** to **9999.** The corresponding actual pen range of physical movement is determined by adjustments on the plotter itself. The FMT, ↑ command-sequence causes the pen to be raised and the contents of the *x*- and *y*-registers to be fed to the plotter as the *x* and *y* coordinates to which the pen is to be moved. The FMT, ↓ command-sequence lowers the pen and transmits the *x*-*y* coordinates. Note that for both commands the pen is raised or lowered first and the pen

carriage is then moved. The *x-y* coordinates in the *x*- and *y*-registers must be pre-scaled by the user in the range **0000** to **9999.** If these limits are violated, the following events occur:

1. If the contents of the *x*- and/*or* y-registers are less than 0000 (i.e. negative), 0000 is sent to the plotter, the pen is raised, moved, and lowered.
2. If the contents of the *x*- and/or *y*-registers are greater than 9999, 9999 is sent to the plotter, the pen is raised, moved, and lowered.

The result is to plot a series of dots along the boundary of the plot. The raising and lowering of the pen is a visual and audible warning to the user that he is plotting out-of-bounds.

The basic calculator includes provision for recording a user-program in the program storage section of the RWM as a secure program. This provision is completely distinct from a protected recording obtained by physically notching the magnetic card. A secure program is, by definition a program which can be executed only. It cannot be listed, recorded, looked at in the program mode, edited, or changed in any way. It is a program which can only be executed in the automatic operating mode at high speed by using the CONTINUE key (or step-by-step using the STEP PRGM key hereinafter explained).

Any program entered by the user from the keyboard input unit into the program storage section of the RWM, or any non-secure program loaded from a magnetic card, can be recorded as a secure program. This is accomplished in the same manner as described above in connection with the RECORD key except that the keys FMT and RECORD are sequentially depressed in the named order to initiate the recording operation. The program thereupon recorded on the magnetic card or cards is a secure program. The original program also remains in the program storage section of the RWM as a non-secure program, and can be recorded again, either as a secure or nonsecure program. It should be noted that once recording of a secure program has begun, it must be completed by continuing to insert cards until recording is terminated by an END program step. If the recording process is interrupted by depression of the STOP key, the recording will be terminated but the original program stored in the program storage section of the RWM then becomes a secure program and can only be executed — no further recordings can be made.

If a program stored in the program storage section of the RWM is a secure program (either by loading a program recorded as a secure program or by the STOP default outlined above) the following conditions prevail:

1. If the PRGM key is depressed, the entire program memory is cleared.
2. If LIST or RECORD are depressed, they are ignored, so that no listing or recording of the program can be made. These actions do not destroy the program — they are just ignored.
3. Since the program is destroyed by switching to the PRGM mode, the program cannot be changed or edited in any way.

Loading of a secure program is no different from loading a non-secure program — the same procedure holds. Several secure programs may be chain-loaded. If a non-secure program is loaded into the calculator when a secure program is already stored therein, the program storage section of the RWM is cleared of the secure program in all areas not loaded by the non-secure program so that no trace of the secure program remains after loading the non-secure program. Thus, a non-secure program may not be chain-loaded after a secure program. However, the reverse can be done — a secure program can be chain-loaded to a non-secure program but the composite program is then secure.

The command sequence FMT, GO TO is used to implement automatic or program-controlled loading of magnetic program cards. The effect of FMT GO TO is the same as the following sequence: GO TO, **0** LOAD, GO TO, **0,** and CONTINUE. This may be used to load a program from the keyboard, with automatic initiation of execution, or it may be used in a program to "link" programs. If the magnetic card reading and recording unit is not loaded with a magnetic card, the INSERT CARD indicator light will come on indicating to the user that magnetic card should be inserted. The LOAD routine operates in the same manner as described above in connection with the LOAD key and is terminated by an END program step on the magnetic card.

The command sequence FMT, $x \rightarrow$ causes the contents of the available numeric-addressed data-registers in the data storage section of the RWM to be recorded on a magnetic card. The INSERT CARD indicator light comes on, and the recording continues, pass-after-pass, card-after-card until all registers are recorded, at which time the magnetic card reading and recording unit stops and the INSERT CARD indicator light goes out. If the FMT, $x \rightarrow$ command sequence was executed from a stored-program, the program will return to execution at the keycode immediately following the $x \rightarrow$ command. Recording may be terminated after any card by depressing the STOP key if the FMT $x \rightarrow$ command sequence came from a stored program, execution may then be resumed by depressing the CONTINUE key.

The command sequence FMT $x \leftarrow$ causes the loading of the available numeric-addressed datal registers in the data storage section of the RWM from one or more magnetic data cards. These registers are loaded card-after-card until all the registers are loaded. The INSERT CARD light remains on, and the card unit continues to run until all are loaded. The loading may be terminated by a STOP key. If the FMT, x $\leftarrow$ command sequence is executed from a stored program, executive resumes after a completed data load.

The PRINT/SPACE key causes the contents of the *x*-register to be printed in the same format as it is displayed (i.e. FIX (), *n*, or FLOAT). If additional PRINT/SPACE commands follow immediately, they will cause the printer to space (print a blank line). If the PRINT/SPACE command follows immediately after a numeric entry to *x* (from the keyboard or from a sequence of digit-keys in a program) the numeric printout is followed by an * (asterisk) indicating that this was a data entry and not a computed result.

The PAPER key is depressed to space the strip of thermal-sensitive paper used by the output printer unit. It continues to drive the paper upward until it is released.

Program Checking and Editing Keys (STEP/PRGM, BACK/STEP)

The STEP PRGM key is not programmable and is used, from the keyboard only, to single-step programs. When the calculator is in the manual operating mode the STEP PRGM key single-steps program execution. Each time STEP PRGM is depressed the program counter is incremented by one so that one program step

is executed. When the calculator is in the program mode the STEP PRGM key enables the program steps stored in the program storage section of the RWM to be viewed. Each time the STEP PRGM key is depressed, the program counter is incremented by one, so that the address and program step displayed in the *y*-register shifts to the *z*-register, those in the *x*-register shift to the *y*-register and the next higher address and program step appears in the *x*-register.

The BACK STEP key is not programmable and is used to decrement the user program counter by one each time it is pressed. This backs up the output display (i.e. does just the opposite of the STEP PRGM key). It should only be used in the program mode. If the STEP PRGM and BACK STEP keys are depressed alternately, the same program step will be executed repeatedly. This key is extremely useful in editing and checking programs and when used with the PRGM STEP key permits the user to advance either forward or backward through a stored program one step at a time.

Definable and Redefinable Keys

The half keys A–O comprising the group of definable keys **91** enable the calculator to be tailored to the special needs of the user. Operation of these keys is defined by the various plug-in ROM modules **92** that may be used with the calculator. Without these ROM modules the definable keys **91** serve no function and accidently depressing them, or encountering them in the execution of a stored program, will result in a completely non-destructive no-operation.

The plug-in ROM modules **92** include the alpha ROM module mentioned above, a definable functions ROM module, a mathematics ROM module, a statistics ROM module, and a typewriter ROM module. Both the alpha ROM module and the typewriter ROM module redefine nearly all of the keys of the keyboard as well as defining the definable keys **91** themselves. The definable functions ROM module, the mathematics ROM module, and the statistics ROM module each uniquely define the definable keys alone and may each be used at the same time as the alpha ROM module or the typewriter ROM module.

A different overlay **192** is associated with each of the definable functions, mathematics, and statistics ROM modules and is employed with the definable keys **91** to identify the functions performed thereby when its associated ROM module is plugged into the calculator. Each of these overlays **192** comprises a thin metal template that fits over the definable keys **91** and latches into a recess around them. The graphics on these templates visually complete the key shapes and indicate the key function. A small tab positioned just above the nameplate releases each template, which then pops up enough to grasp. Three holes **196** near the top edge of each template allow direct viewing of three light-emitting diode indicator light used to indicate various operating conditions associated with the routines implemented by the ROM module. When the overly and its associated ROM module are not in use they may be secured together by a pair of tabs **198** provided on the ROM module.

A description of the additional key operations that may be provided by the plug-in ROM modules will now be given.

Alpha ROM Module

The Alpha ROM module redefines the keyboard input unit as indicated by the letters printed on the tops of the definable keys **91** and the letters and symbols printed on the front sides of most of the other keys to provide an "alpha keyboard" (see FIG. **8**) containing 54 character-entry keys, five operational keys, and 16 "non-essential" keys (these non-essential keys are either inoperative or duplicate other keys during the alpha mode). During the alpha mode, the key-log feature is deactivated (thus, any keys pressed are not logged).

The 54 character-entry include all of the English alphabetic characters A–Z, all of the decimal numbers 0–9, and all of the following symbols @, $\sqrt{\ }$, / (printed by the ÷ key), x, −, +, π, → , ,,,,, =, $, ?, (,), %, '', and #. Depressing any of these keys during the alpha mode, will cause the alphameric character or symbol indicated thereby to be printed out in line-printer fashion. The output printer unit operates as a line printer in that each character is not immediately printed out, but rather an entire line (16 characters) is first stored and then printed out. The print-out occurs as the 16th character is entered.

The five operational keys include the FMT key, the STOP key, a SPACE key (normally the CONTINUE key), a CLEAR/RETURN key (normally the CLEAR key), and a PAPER key.

During the alpha mode, the following operational keys are depressed to perform various printing operations. Depressing the FMT key twice redefines the keyboard to the alpha mode, after which character keys may be depressed. After the last character is entered, depressing the FMT key causes a line print, a line feed, and returns the keyboard to normal operation. (The output display is blanked during the alpha mode, although the contents of the *x*-, *y*-, and *z*-registers remain unchanged.) For example, the alphabet may be printed by sequentially depressing the RUN, STOP, FMT, FMT, A through Z, and FMT keys.

Depressing the SPACE key inserts a blank space in the printed line (similar in operation to the space bar on the typewriter).

Depressing the CLEAR/RETURN key causes a line print and advances the printer to the next line (i.e. like a typewriter carriage-return and line feed operation). The alpha mode remains set after this instruction. Successive CLEAR instructions will cause the printer to advance, without printing, one line for each instruction.

Depressing the STOP key terminates the alpha mode without a line print or line feed. Any characters entered but not printed will be erased when STOP is pressed. This instruction is not programmable and should not be used while programming alpha messages.

The PAPER key is a manual paper advance control. This operation is not programmable.

The 16 "non-essential" keys include the ↓, $x \rightleftarrows y$, ↑, PRINT/SPACE, SUB/RETURN, END, BACK/STEP, STEP/PRGM, FLOAT, FIX (), RUN, PRGM, KEY LOG, LIST, LOAD, and RECORD keys. These keys are not essential for alpha printing operations. The non-essential keys which are programmable duplicate the SPACE key, while most of the non-programmable keys are "locked out" (i.e. not operational) during alpha printing operations. Pressing BACK STEP or STEP PRGM will cause 1 or 0, respectively, to be printed.

Definable Functions ROM Module

When the definable functions ROM module is plugged into the calculator, the user may employ the definable keys **91** to perform the redefined functions indentified by the associated overlay shown in FIG. **9.** These functions include:

a. Defining special subroutines that can be called by a single key;
b. Protecting such subroutines against accidental erasure;
c. Deleting one, or more, of such subroutines when desired; and
d. Deleting, inserting, or searching for keycodes in any stored program.

a. Defining a Function

A user definable function can be any sequence of program steps, beginning with the key DEFINE F (*i*), where F(*i*) represents one of the keys F1 thru F9. The execution of such sequence of program steps must be terminated with the F-RET key in the same manner as the SUB/RET key in the case of subroutines.

Example:

The following is a function which puts π in *x*, *y*, and *z*.

Program Step	Key
0000	DEFINE
0001	F1
0002	π
0003	↑
0004	↑
0005	F-RET

The above function is identified by the key F1. To execute this function simple press the F1 key in RUN mode or insert F1 in a program at the place you want it executed.

Example:

The following is a call on F1 from another program (the program here happens to be another function F2).

Program Step	Key
0000	DEFINE
0001	F2
0002	CLR
0003	F1
0004	F-RET

Notice that we have assigned the same program step numbers to both the functions F1 and F2. Every function definition is assumed to start at program step **0000.** All GO TO commands when present in a function are coded accordingly. This allows the user to define functions and delete them in a way independent of where they actually reside in the calculator memory. The key DEFINE in both of the above functions serves to mark the start of the definition of a function in the calculator memory. This key is treated as an error if it is pressed during RUN mode. The key DEFINE is also treated as an error if the user attempts to execute it in a program.

Example:

Program Step	Key
0000	DEFINE
0001	F3
0002	CLR
0003	DEFINE←
0004	F2

The DEFINE at program step **0003** will be treated as an error when encountered during the execution of F3. The user here forgot to terminate the execution of F3 with F-RET. This error will cause the calculator to stop execution and turn on the STATUS light.

b. Protecting User Defined Functions

The protection feature allows the user to designate areas of memory to be a part of the calculator's executive system.

i. Protecting A Single Function

Assume that you have just entered the definition of the function F1 in memory (the same F1 discussed earlier). After entering the last keystroke in the definition, namely F-RET, the program counter (*y* display) will point to program step **0006.** Location **0005** which contains F-RET will be displayed in the *z*-register. Switch to the RUN mode. Press PROTECT. The light on the top of the PROTECT key (option **2** light) ..... will come on. The definition of F1 is now protected up to and including the F-RET key. If you switch back to PROGRAM mode, whatever was in program step **0006** before will appear now to be at program step **0000.** All the program steps used in defining F1 became "invisible". For all purposes these program steps are now a part of the calculator system. You can still execute F1 by pressing the corresponding key in RUN mode, or by calling on it from another program or function.

ii. Protecting Several User Defined Functions

Assume now that both the functions F1 and F2, described earlier, are to be protected. After protecting F1 switch to the PROGRAM mode and enter the definition of F2, namely:

Program Step	Key
0000	DEFINE
0001	F2
0002	CLR
0003	F1
0004	F-RET

After entering the definition of F2 switch to the RUN mode. Press DELETE PROTECT. The protect light indicator (option **2**) will go off. Press PROTECT. Both the definitions of F1 and F2 are protected.

iii. Effect of Protection on User Memory

When protecting a certain number of keystrokes the calculator subtracts that number from the total number of program steps available in user memory. The user can compute the number of program steps available at any time as follows. In RUN mode, press END, DELETE, PROTECT, switch to the PROGRAM mode and read the program step number in the *y* display, assume this number is *p*. The number of program steps

available to the user are given by the simple formula:

Available Program Steps = $n - p$

The variable n in the above formula is 500, 1012 or 2036 according to the amount of user memory purchased.

After finding the value of p, as illustrated above, you can switch back to the RUN mode and press PROTECT again to re-establish the protection.

c. Deleting User Defined Functions

To delete a given function simply press the DELETE key followed by the name of the function to be deleted. The DELETE key only applies to the unprotected area of user memory. This prevents accidental erasure of protected functions. Pressing the DELETE key causes the Insert/Delete light to come on (option 1) indicating that a memory modification operation is to be performed. If after pressing the DELETE key you find that you did not want to delete after all, you can press the CLEAR key to abort the delete option. Pressing the CLEAR key at this time will cause the Insert/Delete light to go off. If the function to be deleted does not exist the error light will come on and the delete operation will be aborted. The DELETE key must be pressed while the calculator is in RUN mode. The DELETE key may be followed by a function name (**F1** thru **F9**), CLEAR, or a digit. Other keys following the DELETE key will turn on the error light and abort the delete operation. If encountered during the execution of a stored program the DELETE key will cause the calculator to stop execution and turn on the error indicator.

DELETING THE LAST FUNCTION IN MEMORY

The delete operation is designed to delete a portion of user memory which starts with the DEFINE of the function to be deleted and ending with the keystroke prior to the DEFINE of the next function in memory, or END. To illustrate, consider the following memory map.

Program Step	Key
0000	DEFINE←
0001	F1
0002	CLR
0003	F-RET←
0004	DEFINE
0005	F2
0006	EEX
0007	1
0008	-
0009	IF $X=Y$
0010	F-RET
0011	CONT
0012	CONT
0013	CONT
0014	GO TO
0015	4
0016	END
0017	...

When pressing DELETE **F1** the calculator will delete the memory portion delimited by the two arrows in the memory map. After deletion the DEFINE at program step **0004** will move to program step **0000.** The following keystrokes will move accordingly. Since the last function in a user memory is not likely to be followed by another DEFINE it is advisable to terminate that function with the keystroke END as shown in the memory map (program step **0016**). If the last function in memory is not terminated with END then by deleting that function the user will also delete that portion of memory which follows that function and terminates with an END. When deleting the last function in memory, the delete operation does not delete the END key. Thus deleting **F2** from the memory map shown earlier will result in the following memory map.

Program Step	Key
0000	DEFINE
0001	F1
0002	CLR
0003	F-RET
0004	END

The keystroke END becomes now the terminator to the definition of **F1.**

d. General Editing Capabilities

In addition to deleting functions the user of the definable function block is offered the capability of deleting, inserting, and searching for individual keystrokes in memory. This feature is discussed next.

i. Deleting Individual Keystrokes

If the DELETE key is followed by a **4** digit address* the calculator will delete the keystroke contained in that address and move all the following keystrokes accordingly.

*See next paragraph on Automatic Address Termination for further details on address length.

Example:

Consider the memory map:

Program Step	Key
0000	CLR
0001	GO TO
0002	0←
0003	5
0004	END
0005	1
0006	...

To delete the keystroke at program step **0002** press the DELETE key in RUN mode, (Insert/Delete light will come on), followed by the keys **0 0 0 2.** The resulting memory map will be:

Program Step	Key
0000	CLR
0001	GO TO
0002	5
0003	END
0004	1
0005	...
0006	...

Notice that the keystroke previously at **0003** is now at program step **0002.** All the keystrokes following the deleted key up to the last keystroke in memory have been moved accordingly.

AUTOMATIC ADDRESS TERMINATION AND ABORTING A DELETE:

In the above example we used a 4 digit address to specify the keystroke to be deleted. The user can use a 3, 2, or one digit address if such an address is terminated by a non-numeric key other than CLEAR. The following delete commands are equivalent to that given in the last example:

DELETE	2	STOP
DELETE	02	STOP
DELETE	002	STOP

To abort a delete command before it is executed press the CLEAR key. The Insert/Delete light (option 1) will go off and the delete command will be cancelled. The CLEAR key can be pressed immediately after the DELETE key or during the entry of the address digits. Notice that the fourth digit in an address terminates that address. This digit signals to the calculator that the delete operation is to be performed. A CLEAR key following the fourth digit of an address will have no effect on a delete operation. After performing a delete operation the calculator switches off the Insert/Delete light. If the delete address is bigger than 2035 the status light will come on and the delete operation will be aborted. This number represents the maximum available program steps in the calculator.

ii. Inserting Individual Keystrokes

The definable function block also offers the user the capability of inserting key codes at any arbitrary program step.

Example:

Consider the following memory map:

Program Step	Key
0000	CLR
0001	GO TO
	←
0002	9
0003	END
0004	...

To insert a keystroke before the key at program step **0002,** press INSERT in RUN MODE, (the Insert/Delete light will come on), followed by **0 0 0 2.** Switch to PROGRAM mode. The *y* display will point to program step **0002.** At this location you will find that a CONTINUE (Code **47**) has been inserted. The keystrokes originally at **0002** has now been moved to step **0003.** All the keystrokes which follow this keystroke have been moved accordingly. To put a new keystroke at **0002,** simply press the new key while you are still in PROGRAM mode. The key pressed will replace the CONTINUE inserted earlier by the calculator. Without replacing the CONTINUE at step **0002** the new memory map is as shown below:

Program Step	Key
0000	CLR
0001	GO TO
0002	CONT
0003	9
0004	END
0005	...

The address in an INSERT command may . . . . . be automatically terminated in the same manner described earlier in connection with DELETE. The insert operation can also be aborted in the same fashion.

Notice that the insert operation effects the content of all the program steps following the one designated in that operation.

iii. Searching for a Given Keystroke in Memory:

The ability to search for a given keystroke in memory is provided by the FIND key. By pressing the FIND key, in RUN mode, followed by any other key on the keyboard the user commands the calculator to search for that key in memory starting at the current program counter value. If the key asked for is not found, the status light will come on. However if the key is found: the calculator will automatically switch to PROGRAM mode, and the Z register will contain the program step at which the key was found.

Example:

Consider protecting a function definition area in memory. This area is terminated with the key END. Reset the program counter to **0000** by pressing GO TO **0** or END, in RUN mode. Now press FIND END, the *z*-register will now contain the terminating END in the function area. Press RUN PROTECT. The function area is now protected up to and including that terminating END.

PROGRAMMING HINTS:

The definable functions can be nested five levels deep, irrespective of regular subroutine nesting. A higher nesting value will cause the calculator to stop execution and turn on the status light.

The light on top of the F-RET key (option 3) when lit indicates that the user is executing a sequence of keystrokes in a function definition. If you press STOP while this light is on you may stop the calculator in the middle of a function. To reinitialize program execution press DELETE, F-RET in RUN mode, the F-RET light will go off and the program counter adjusted to the value it had when the function area of memory was called.

Calling on a non-existant function from a stored program will cause the calculator to stop execution and turn on the status light. The program counter at this time will point to the program step which contains the name of the function called.

Mathematics ROM Module

When the mathematics ROM module is plugged into the calculator at the left-hand receptacle **94,** the user may employ the definable keys **91** to perform the additional functions indicated by the mathematics overlay shown in FIG. **10.** All of these additional functions are programmable. Any mathematically illegal functions performed either from the keyboard input unit or a stored program will turn on the STATUS light.

The units to be used in problems involving trigonometric functions or vector arithmetic are selected according to the procedure listed in Table A. The units specified by the appropriate indicator light above the definable key block.

Table A

To Specify:	Specifying Units Press:		Indication:		
DEGREES	TABLE N	1	Deg	Rad	Grad
RADIANS	TABLE N	2	Deg	Rad	Grad
GRADS	TABLE N	3	Deg	Rad	Grad

It should be noted that the setting of degrees, radians, and grads (360 degrees = 400 grads) is programmable.

Trigonometric functions of angles from 0° up to 5760° can be calculated at full accuracy; however, inverse trigonometric functions are calculated only for the principal values of the functions:

$\theta = \sin^{-1} x; -90° \leq \theta \leq +90°$

$\theta = \cos^{-1} x; 0° \leq \theta \leq +180°$

$\theta = \tan^{-1} x; -90° \leq \theta \leq +90°$

For instance: $\cos 150° = \cos 210° = \cos 510° = (\text{etc.}) = -.866$

But: $\cos^{-1} .866 = 150°$.

The sin x key is depressed to calculate the Sine of the contents of the x-register and insert the result in the x-register.

The cos x key is depressed to calculate the Cosine of the contents of the x-register and insert the result in the x-register.

The tan x key is depressed to calculate the Tangent of the contents of the x-register and insert the result in the x-register.

The arc key is depressed followed by a trigonometric key to calculate the inverse trigonometric function of the contents of the x-register and insert the result in the x-register.

For example, the $\sin^{-1} 0.5$ may be calculated by sequentially depressing the TABLE N, **1**, **.**, **5**, arc, and sin x keys. Depression of the TABLE N and **1** keys selected the units (degrees) per Table A above.

The following logarithmic and exponential functions may all be performed by employing one or two keystroke operations.

The TABLE N and **4** keys are sequentially depressed in the order named to calculate the logarithm (to base 10) of the contents of the x-register and display the result in the x-register.

The TABLE N and **5** keys are sequentially depressed in the order named to raise 10 to the power indicated by the contents of the x-register and display the result in the x-register (i.e. 10^x). For example, the number 0.69897 may be raised to the power indicated by the contents of the x-register by sequentially depressing the **.**, **6**, **9**, **8**, **9**, **7**, TABLE N, and **5** keys.

The ln x key is depressed to calculate the logarithm (to base e, i.e. natural logarithm) of the contents of the x-register and display the result in the x-register (i.e. ln x). For example, the $\sqrt[5]{23}$ may be calculated by sequentially depressing the **2**, **3**, ln x, ↑, **5**, ÷, ↓, and e^x keys.

The e^x key is depressed to raise e (i.e. 2.718....) to the power indicated by the contents of the x-register and display the result in the x-register (i.e. e^x).

The x^y key is depressed to raise the contents of the x-register to the power indicated by the contents of the y-register. The result is displayed in the x-register, and the contents of the y-register remain unchanged.

The following keys provide capability for performing complex and vector arithmetic with a single keystroke operation.

The TO POLAR key is depressed to convert rectangular coordinates (consisting of x and y components in the x- and y-registers, respectively) to polar coordinates ($\theta = \tan^{-1}(y/x)$, $R = \sqrt{x^2+y^2}$). When converting from rectangular (cartesian) to polar coordinates, the calculated angle θ will be within the range of $-180° \leq \theta \leq 180°$. The final display is:

temporary z	------
accumulator y	(Angle θ)
keyboard x	(Radius R)

For example, the coordinates 4, 3 (x, y) may be converted to polar form by sequentially depressing the TABLE N, **1** (these keys select the units, i.e. degrees), **3**, ↑, **4**, and TO POLAR keys.

The TO RECTANGULAR key is depressed to convert polar coordinates, when the radius (R) and the angle (θ) are in the x- and y-registers, respectively, to rectangular coordinates ($y = R \sin \theta$, $x = R \cos \theta$). The final display is:

temporary z	------
accumulator y	(y component)
keyboard x	(x component)

For example, the polar coordinates R = 8, $\theta = 120°$ (or $-240°$) may be converted to rectangular form by sequentially depressing the TABLE N, **1**, **1**, **2**, **0**, ↑, **8**, and TO RECTANGULAR keys. The final display is:

temporary z	
accumulator y	6.928
keyboard x	−4.000

The ACCUMULATE +, ACCUMULATE −, and RECALL keys are storage and recall keys associated with the a- and b-data storage registers. These keys provide complete capabilities for vector addition and subtraction.

The ACCUMULATE + key is depressed to simultaneously add the contents of the x- and a-registers together and the contents of the y- and b-registers together. The sums are entered in the a- and b-registers, respectively, while the x- and y-registers remain unchanged.

The ACCUMULATE − key is depressed so simultaneously subtract the contents of the x-register from the contents of the a-register and the contents of the y-register from the contents of the b-register. The remainders are entered into the a- and b-registers, respectively, while the contents of the x- and y-registers remain unchanged.

The TABLE N key permits access to 10 more ROM functions than there are definable keys. A list of these functions is given in Table B below. The TABLE N key may be followed by any key. If this key is different from a numeric or FMT, no operation is performed.

Table B

TABLE N	FUNCTION	
1	SET DEGREES	SETS ARGUMENT
2	SET RADIANS	UNITS FOR TRIGONO-
3	SET GRADS	METRIC FUNCTIONS
4	$\text{Log}_{10}x$	
5	10^x	
6	DEGR, MIN, SEC →	DECIMAL DEGREES
7	DECIMAL DEGREES →	DEGR, MIN, SEC
8	X!	
9	ROUND	
FMT	AUTOMATIC PLOTTER	SCALING

The first five functions of the TABLE N key have already been explained above. However, this key may also be used to perform any of the next five functions

(namely, angle conversion, calculation X!, rounding a number to a specified power of 10, and plotter scaling) not previously described above.

The TABLE N and **6** keys are sequentially depressed in the order named to convert an angle expressed in degrees, minutes and seconds to decimal degrees. The angle must be entered into the calculator as follows:

DISPLAY		
temporary z	←	(Degrees)
accumulator y	←	(Minutes)
keyboard x	←	(Seconds)

The result in decimal degrees, appears in the *x*-register, while the *y*- and *z*-registers are cleared. This is illustrated as follows:

temporary z	— 0
accumulator y	— 0
keyboard x	—DECIMAL DEGREES

The TABLE N and **7** keys are sequentially depressed in the order named to convert an angle expressed in decimal degrees to degrees, minutes and seconds. The angle to be converted must be entered into the *x*-register, and the resultant angle appears as in the previous display. However, the contents of the *y*- and *z*-registers need not be zero for the instruction TABLE N, **7.**

The TABLE N and **8** keys are sequentially depressed in the order named to replace the contents of the *x*-register with X! (where $x\ 0 \leq |x| \leq 69$).

The TABLE N and **9** keys are sequentially depressed in the order named to round the contents of the *y*-register to the power of 10 indicated by the integer value of the contents of the *x*-register. The rounded number appears in the *x*-register, while the *y*-register remains unchanged. For example, the number 5610.0 may be rounded to 10^2 (or nearest 100), by sequentially depressing the **5, 6, 1, 0,** and ↑ followed by the **2,** TABLE N, and **9** keys.

DISPLAY	
accumulator y	5610.000
keyboard x	5600.000 (y rounded)

Similarly, the contents of the *y*-register may be rounded to another power (10^4) by sequentially depressing the **4,** TABLE N, and **9** keys.

DISPLAY	
accumulator y	5610.000
keyboard x	10000.000 (y rounded)

A fractional number may be rounded by inserting a negative number into the *x*-register. For example, the number 0.005 may be rounded to the 10^{-2} or nearest 1/100 by sequentially depressing the ., **0, 0, 5,** ↑, CHG SIGN, **2,** TABLE N, and **9** keys.

DISPLAY	
accumulator y	0.005
keyboard x	0.010 (y rounded)

A problem usually encountered when writing a calculator/plotter program is that of scaling the available problem variables to coordinates which the plotter can use. The plotter scaling feature to be described, simplifies this typical plotting problem. The TABLE N, FMT, and ↑ or ↓ keys are sequentially depressed in the order named to replace the plotter problem variables, which are entered in corresponding *x*- and *y*-registers, with scaled coordinates. The user variable maxima and minima for this scaling operation are stored in user data storage registers **001–004.** The foregoing sequence of keys controls the plotter (using the scaled variables) in the same manner as FMT, ↑, or ↓ described above.

The DEFINABLE *f* ( ) key is used to label and "call" an often used (or favorite) function which is programmed as a subroutine in the calculator. The definable function may be executed at any time from the keyboard by depressing the DEFINABLE key, or it may be called in a program by inserting the DEFINABLE (key) instruction. The definable function is programmed similar to a LABEL subroutine, while the function is executed as a normal subroutine; except it may be called with only one keystroke or program step.

When the DEFINABLE *f* ( ) key is depressed the program counter searches for a subroutine labeled DEFINE *f* ( ), executes, and returns. This key can be used both in keyboard and program control. A user written subroutine to be called by the DEFINABLE *f* ( ) key can be stored anywhere in the program memory, its first program steps must be LABEL, DEFINABLE *f* ( ); its last step must be SUB/RETURN. There are no restrictions on the operations this subroutine may perform. It is illustrated by the following example:

LABEL
DEFINABLE *f* ( )
↑
e^x
$x \leftrightarrows y$
CHG SIGN
e^x
+
2
÷
(SUB/RETURN)

Whenever the key DEFINABLE *f* ( ) is called either in RUN or PROGRAM mode, the hyperbolic cosine of the number in the *x*-register is computed and placed in the *y*-register by this subroutine.

The TABLE N and CLEAR *x* keys are depressed to clear all numerical storage registers without affecting the *a*- and *b*-registers or the *x*-, *y*- and *z*-registers.

A programmed subroutine may be repeated m number of times by inserting the following keys at the end of the subroutine: TABLE N, SUB RETURN, and *n*, where *n* may be any key from **0** to **9** indicating a data storage register that contains *m*, and *m* is equal to the absolute integer value of the contents of the *n*-register. After the subroutine has been repeated *n* times, the program exits the subroutine and resumes normal program operation at the program step following the subroutine "calling instructions". The iterative subroutine feature may be added to a LABEL subroutine, but when the LABEL subroutine is called (during a program) the call instructions must cntain six program

steps. The following partial program shows how to call the LABEL π iterative subroutine.

Iterative "LABEL" Subroutine

STEP	KEY	(KEY/-CODE)	COMMENT
0400	- - -	- -	
0401	CONTINUE	47	Subroutine
0402	CONTINUE	47	CALLS
0403	GO TO	44	Instructions
0404	(SUB/-RETURN)	77	
0405	LABEL	31	(6 keys)
0406	π	56	Return from
0407	÷	35	Subroutine

Statistics ROM Module

The primary function of the statistics ROM module is to carry out the summations of variables, cross-products, and squares needed as fundamental quantities in a variety of statistical analyses. These summations are generated in the general data storage user registers, and the user must be careful to avoid any operations which might destroy or alter the contents of these registers. The number of registers used is dependent on the number of variables treated — as defined by the user.

The registers used are:

n	→	0	
Σx	→	1	
Σx^2	→	2	1-variable; x
Σy	→	3	
Σxy	→	4	
Σy^2	→	5	2-variable; x, y
Σz	→	6	
Σxz	→	7	
Σyz	→	8	
Σz^2	→	9	3-variable; x, y, z
Σa	→	10	
Σxa	→	11	
Σya	→	12	
Σza	→	13	
Σa^2	→	14	4-variable; x, y, z, a
Σb	→	15	
Σxb	→	16	
Σyb	→	17	
Σzb	→	18	
Σab	→	19	
Σb^2	→	20	5-variables; x, y, z, a, b

These are graphically summarized in an easy-to-recall form in this table:

1	x	y	z	a	b
0	1	3	6	10	15
	2	4	7	11	16
	(1)	5	8	12	17
		(2)	9	13	18
			(3)	14	19
				(4)	20
					(5)

In addition to the user-registers **0** → **20** utilized as shown above, registers **21** → **27** are used for collection of maximum/minimum values, for the "seed" of the pseudo-random number generator. This is shown below:

x_{min}	→	21
x_{max}	→	22
y_{min}	→	23
y_{max}	→	24
z_{min}	→	25
z_{max}	→	26
RN	→	27

Any registers not used in a specific sequence are available. For example, if **2** - variable operations are set up, only registers **0** → **5** are in use, and **6** → **20** are free for other purposes.

When the statistis ROM module is plugged into the calculator, the user may employ the definable keys **91** to perform the additional functions indicated by the statistics overlay shown in FIG. **11.** All of these additional functions are programmable and will hereinafter be described key-by-key.

The VARIABLES K key is used to define the number of variables to be treated, 1 to 5. It must be followed immediately by a digit key, **1** to **5.** If any other key is depressed after this key, the STATUS light will be turned on, and the calculator will halt in the display mode. The VARIABLE key and the erroneous key following it are ignored — there is no other action.

If a correct digit key follows, one or more of the indicator lights will come on, to signal the number of variables selected. The pattern is:

1-variable	1-light
2-variable	2-light
3-variable	3-light
4-variable	1 and 3-lights
5-variable	2 and 3-lights

The definition of the number of variables affects subsequent use of the Σ key, MAX/MIN key, and the INITIALIZE and CORRECT keys used in conjunction with it. The number of variables remains unchanged until the VARIABLES key is used to change it, which may be done at any time.

Σ

This key is used to accumulate the data summations of variables, cross-products, and squares as outlined previously. The number of summations is determined by the VARIABLES key described above. That is, the contents of the registers utilized are:

1-variable	x
2-variable	x, y
3-variable	x, y, z
4-variable	x, y, z, a
5-variable	x, y, z, a, b

If the summation key (Σ) is depressed without a previous definition of the number of variables (VARIABALES key followed by digit **1** → **5**), the number of variables is set at three. It will remain at this setting unless changed by use of VARIABLES

The Σ key generates the summations, and leaves the contents of x, y, z, a, and b unchanged.

The INITIALIZE and CORRECT keys work in conjunction with the summation key. They must precede the Σ key.

When the sequence INITIALIZE - Σ is used, all registers (defined by the number of variables set) involved in the summations are cleared to zero. This sequence should always be used before the start of a series of summations on a set of data — otherwise any previous

contents of the registers are included in the summations.

If, after depressing the Σ key, it is discovered that the contents of x, y, z, a, or b were erroneous, the user may remove the erroneous data from the summation by depressing CORRECT - Σ keys in that sequence. This will remove all variables, cross-products, and squares of that data from the summation. The user may then correct the data and reenter it by depressing the Σ key. Since x, y, z, a, and b are unchanged by the use of Σ (or CORRECT - Σ) this is most conveniently done when the erroneous data is still intact - i.e., immediately after Σ . However, if the erroneous data is not discovered until later, the user must reenter the erroneous data in x through b (only x and y if **2** - variable, etc.), use CORRECT - Σ, and then correct the erroneous data and enter it with the Σ key.

The user must be careful not to do any operations during a summation-sequence on data which will alter the contents of any user-registers involved in the summation. However, the contents of any of the summation-registers may be recalled and used at any time, so long as they are not altered.

Once a data-sequence has been entered by use of the Σ key, the summations are available for any desired statistical analysis. For user convenience, four commonly used statistical processes are implemented, to be performed by a single keystroke. The function of these four keys will follow.

The MEAN key computes (from the collected summations) the arithmetic mean of up to three variables; x, y, and z. If four or five variables are set, the MEAN key operates on only three, and does not form the other two means (on a and b).

For various variable settings, the following computations are made, and appear in the x, y, z registers:

1-variable	z	$\phi.\phi$
	y	$\phi.\phi$
	x	$\bar{x}=(\Sigma x/n)=((1)/(0))$
2-variable	z	$\phi.\phi$
	y	$\bar{y}=(\Sigma y/n)=((3)/(0))$
	x	$\bar{x}$
3-variable	z	$\bar{z}=\Sigma z/n=((6)/(0))$
	y	$\bar{y}$
	x	$\bar{x}$

where (**1**) means "contents of register - **1**".

These computations are carried out and the results appear in x, y, z as shown, without changing the contents of any of the summation registers.

The VARIANCE key computes (from the collected summations) the variance of up to three variables, x, y, and z. If four or five variables are set, the VARIANCE key operates on only three, and does not form the vaiance of a or b.

The following computations are performed:

$$x=\frac{\Sigma x^2-\frac{(\Sigma x)^2}{n}}{n-1}=\frac{(2)-\frac{(1)^2}{(0)}}{(0)-1}$$

$$y=\frac{\Sigma y^2-\frac{(\Sigma y)^2}{n}}{n-1}=\frac{(5)-\frac{(3)^2}{(0)}}{(0)-1}$$

$$z=\frac{\Sigma z^2-\frac{(\Sigma z)^2}{n}}{n-1}=\frac{(9)-\frac{(6)^2}{(0)}}{(0)-1}$$

The results appear in the x, y, z registers in the pattern

	1-variable	2-variable	3-variable
z-	$\phi.\phi$	$\phi.\phi$	sz^2
y-	$\phi.\phi$	sy^2	sy^2
x-	sx^2	sx^2	sx^2

The contents of all summation-registers used in these computations remain unchanged.

The REGRESSION key performs linear regression (least-squares curve fitting) using the accumulated summations. The computations and results are controlled by the variable-setting, as outlined below.

Variable - **1**: The regression of a single variable on itself is not performed. If REGRESSION is depressed with VARIABLES - **1** set, the STATUS light is turned on, and the calculator halts in the display mode. There will be no other action - the contents of x, y, z, and all summation — registers will be unchanged.

Variable - **2**: The regression of the dependent variable on one independent variable is performed for the equation:

$$y = a_0 + a_1 x$$

The results are placed in x, y, z in the pattern:

z ——— $\phi.\phi$
y ——— a_0
x ——— a_1

The contents of all summation-registers used in the computation remain unchanged.

Variable - **3**: The regression of the dependent variable on two independent is performed for the equation:

$$z = a_0 + a_1 x + a_2 y$$

The results are placed in x, y, z in the pattern:

z ——— a_0
y ——— a_2
x ——— a_1

The contents of all summation-registers used in the computation remain unchanged.

Variable **4** - and Variable - **5**: This situation is treated as Variable - **3**. Four - and **5** - variable regression may be performed by user-programming. All summations required are generated by the Σ key when variables are set at **4** or **5**.

r^2

The r^2 key generates the correlation coefficient (a measure of goodness-of-fit) for the linear regressions performed by the REGRESSION key. The computations performed are controlled by the variable setting.

Variable - **1**: No computations are performed—the key is ignored.

Variable - 2: The correlation coefficient of the linear regression for:

$$y = a_0 + a_1 x$$

is computed.

The computations performed are best described by introduction of a subsidiary quantity:

$$\mu x_i x^j = \Sigma x_i x_j - [(\Sigma x_i)(\Sigma x_j)/n]$$

The correlation-coefficient for variable - 2 is then defined to be:

$$r^2 = (\mu xy)^2/\mu xx \mu yy$$

This result is placed in register x, and y and z are cleared.

Variable - 3: The correlation-coefficient of the linear regression for:

$$z = a_0 + a_1 x + a_2 y$$

is computed. This is:

$$r^2 = (a_1 \mu_{xz} + a_2 \mu_{yz})/\mu_{zz}$$

The result is placed in register x, and y and z are cleared.

The MAX/MIN key is used to collect the maximum and minimum values of the variables x, y, and z. Since these values are stored in registers **21** through **26**, they do not affect the summation registers (Σ key). Thus, MAX/MIN information may be collected on the same data on which summation information is being collected.

The MAX/MIN storage registers **21** through **26** are initialized by the key-sequence INITIALIZE - MAX/-MIN. This results in loading the registers with:

(21), (23), (25)	$(x, y, z)_{min} = 10^{99}$
(22), (24), (26)	$(x, y, z)_{max} = -10^{99}$

All six registers are initialized without regard to the variable-number setting.

When MAX/MIN is depressed, the contents of x(**1**-variable), $x$and y (**2**-variable) or x, y, and z (**3,4,** or **5**-variable) are compared to the stored contents of the max/min registers. If the new value is less than the contents of the associated "min" register, the new value is substituted— if not, the register is left unchanged. The maximums are handled correspondingly Thus, at any time the max/min registers contain the max/min of the input data since the last initialization. This data is not displayed—the user must recall it, as needed, from the appropriate register.

The CORRECT key does *not* work in association with the MAX/MIN key, since any previous values changed are lost irretrievably by the MAX/MIN operation.

t

The t key collects summations necessary to compute a t-statistic on data in x and y, and computes and displays the statistic. The overall action is quite different from the Σ key, which collects summations only, and leaves the original data unchanged. In contrast, the t key collects needed summations, computes and presents the t-statistic, and destroys the data just entered.

Further, the t-summations are stored in registers **0,1,** and **2** which are the same registers used by the Σ key. Therefore, use of the Σ and$t$ key *canot* be intermixed.

The summations accumulated are:

n	$\rightarrow$	0
$\Sigma(x_2 - y) = \Sigma D$	$\rightarrow$	1
$\Sigma(x - y)^2 = \Sigma D^2$	$\rightarrow$	2

These are accumulated with each depression of the t key. The three registers may be cleared for starting a new data-sequence by the key-sequence INITIALIZE - t.

If an error is made in data entry and the erroneous data is included in the summations by depression of the t key, it is convenient to have a means for removing the erroneous data. However, the data has been destroyed in order to present the t-statistic which is computed after each key depression. When the user discovers the data error, he may reenter the erroneous data in x and y. Then, depression of CORRECT - t will remove the data from the summation. He may then reenter the correct data and include it by depressing the t key.

The computations performed from the summations are:

$$\overline{D} = \Sigma D/n = [\Sigma(x - y)]/n$$

$$\frac{\overline{D}}{s_{\overline{D}}} = \frac{\overline{D}}{\sqrt{\dfrac{\Sigma(x-y)^2 - \dfrac{(\Sigma(x-y))^2}{n}}{n(n-1)}}}$$

$\overline{D}$ is placed in the z - register, n in y, and t in x.

X^2

The X^2 key accumulates the summations and then computes and presents the chi-squared statistic at each depression of the key. It's general operation is the same as the t key in that:

1. Summations are accumulated in **0** and **1** so that use of X^2 and Σ cannot be mixed.
2. The presentation of results after each key depression destroys the data entry in x and y. Correction for erroneous data can be accomplished by reentering the bad data and then depressing CORRECT - X^2.
3. The registers used (**0** and **1**) are cleared by depressing INITIALIZE - X^2.

The summations accumulated are:

$$n \rightarrow 0$$

$$\Sigma(x - y)^2/y \rightarrow 1$$

In the normal context for use of chi-square the "observed"value is in x, the "expected" value in y.

The RANDOM key causes the computation of a sequence of pseudo-random numbers, uniformly distributed in the interval $0 \geqslant RN \geqslant 1$. The method used is congruential products. It is necessary for the user to provide a "seed" for the sequence before using the RANDOM key. A given seed will produce the same sequence of pseudo-random numbers each time it is used.

The seed should be stored by the user into register **27.** After each depression of RANDOM, the newly-generated pseudo-random will be stored in register **27** as a new seed, and the number will also be presented in the x- register. They y and z registers remain unchanged.

The initial seed provided by the user should be selected with certain rules in mind in order to obtain ac-

ceptable pseudo-random number properties. They are:

1. Enter a decimal fraction consisting of 12 digits (i.e., enter a complete number including guarddigits, even though they cannot be seen).
2. The number should be odd.
3. The number should not be evenly divisible by 5.

The $LOG_{10}x$, LOG_ex, and e^x keys provide the specified mathematical function on the argument in x, and the result is left in the x-register. No other registers are changed.

Typewriter ROM Module

When the typewriter ROM module is plugged into the calculator, the keyboard input unit is redefined as shown in FIG. **12** so that the entire keyboard of a properly interfaced tyewriter such as the Facit Model 3841 (hereinafter referred to as the Model 61) may be completely cntrolled by the calculator. The Model 61 is capable of performing three basic operations:

1. Type data contained in the calculator's xregister.
2. Type alphameric messages and control typewriter functions.
3. List programs contained in the calculator memory.

These operations are accomplished by calculator instructions given either from the keyboard or as program steps in a proper sequence. In this section, these instruction sequences will be described. Numerous examples are provided so that the user will be able to manually key the instruction sequences and observe the results. In normal operation, the instruction sequences would be placed in a program.

Table C provides, in brief form, the instruction sequences used to operate the Model 61. The table is not provided to teach typewriter operation; rather, it is provided for quick reference once the instruction sequences are understood.

All of the Model 61 instruction sequences presented in this section can be reduced into component parts consisting of one or more calculator instructions. For example, all of the Model 61 instruction sequences contain the two calculator instructions FMT and 2.

The instructions FMT and **2** service the special purpose of redefining the following calculator instructions so that they will be understood only by the typewriter. The two instructins, when thought of as a component part of an instruction sequence, can be considered as "the typewriter address". Although these two instructions will be shown as part of each of the following instruction sequences, their meaning will not be explained again.

TABLE C

COMMAND SET Instruction Sequence: Typing a Number	
1. FMT 2 PRINT 2. FMT 2 w.d. 3. FMT 2 w.d. PRINT	Types the number in the x-register in a notation and location specified by the last w.d. instruction. w specifies field width (location). The number will be right justified in the field. At turn-on, the calculator assumes a w of 20. d specifies the number of digits to the right of the decimal point or specifies floating point.

Alphanumeric Typing

FMT **2** FMT Message FMT

Allows the user to type labels and headings. Also allows control of all keyboard functions except margin setting. Messages may include print instructions. Calculator keyboard has two modes, shifted and unshifted (see keyboard diagram on the back page).

Listing A Program

FMT **2** List

Lists the contents of the calculator's memory beginning with the present location of the program counter and continuing until an end statement is encountered in the memory or stop on the calculator keyboard is pressed. The listing includes address location, key code, and key mnemonic.

All commands are available on program or manual request.

The instruction sequence FMT, **2**, w.d, PRINT allows the user to type the data contained in the calculator's x-register. In this sequence, the notation (fixed or floating point) of the typed data can be specified. In addition, the location where the data will be typed on the typewriter platen may also be specified. This instruction sequence consists of the following component parts:

FMT **2** — Typewriter address.

w.d. — Specifies the location and notation of the typed data.

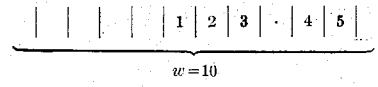
PRINT — Causes the data point to be typed.

The instruction sequence is initiated with the typewriter address and is terminated by the PRINT instruction. The component parts of the instruction sequence have the following meaning for the typewriter.

w.d. — This component of the instruction sequence allows the location and notation of the typed data to be specified. The w part of the component specifies the location; the d part specifies the notation. The decimal point in the component delimits the two parts. Although a w cannot be given without also giving a d, they are best explained separately.

w specifies the field width (w) in which the data is to be typed. The left side of the field is defined by the location of the typewriter carriage when the data is typed. The right side of the field is defined by the value of w. For example, if the typewriter carriage were sitting 10 spaces from the left margin setting and w was specified as twenty, then after the data has been typed the carriage will be located 30 spaces from the left margin setting.

Any data typed in a field will automatically be typed in the right most spaces (right justified) of the field. The following drawing illustrates the number 123.45 right justified in a field of ten.



The field must be large enough to contain the typed data, including the decimal point and any signs that may be present in the data point. If, for one example, the data point were the digit one, a w of one would be large enough to contain the data point; however, if the data point were a negative one, a field width of two would be required. If the field is not large enough to contain the data point, the data will not be typed; instead, the entire field would be filled with asterisks to notify you that the data point could not be fitted into the specified field width.

W may be specified as any number between 1 and 63 (inclusive). Since the data point is always right justified in the field, you will use *w* to place the data point at the desired location on the typewriter platen. Do not, however, set *w* so large that the data, when typed, exceeds the right margin setting. If the right margin is exceeded, the program will be stopped at the memory location following the PRINT instruction (explained later) that exceeded the right margin setting and the STATUS lamp will be lit.

When a *w* instruction is given, the value is automatically stored in the calculator. A *w* instruction, once given, need not be given again in a program and may be used any number of times when succeeding FMT, **2,** PRINT (explained later) instruction sequences.

The value selected for *w* is lost when the calculator is switched OFF. When the calculator is switched ON a *w* of 20 is automatically stored.

The *d* part of the w.d. component allows the user to control the mode of the calculator display from a program and, therefore, the notation (fixed or floating point) of the typed data.

In this instruction sequence, when the data in the calculator's *x*-register is typed, it will be typed as it is displayed in the *x*-register. In order to provide the capability of specifying the notation (fixed or floating point) of the typed data, this instruction sequence is capable of controlling the format of the calculator display. This is provided by the *d* part of the w.d component.

The following table shows the relationship between *d* and the notation of the typed data.

d	Typed Notation
ϕ	decimal part and decimal point are suppressed
1	one digit to the right of the decimal point is typed.
2	two digits to the right of the decimal point are typed.
.	
.	
.	
9	nine digits to the right of the decimal point are typed.
.	←(decimal point) the data will be typed in floating point notation.

A data point in the calculator's x-register will always contain a decimal point; however, if the data point is displayed with no digits to the right of the decimal point (e.g., 10.), the data, when typed, will be typed without the decimal point.

The *d* part of the w.d component is most easily thought of as a programmable FIX ( ) - FLOAT instruction. It has the same effect as those keys on the calculator display. Also, all of the rules of operation concerning overflow and underflow applicable to the FIX ( ) and FLOAT keys apply equally to d.

In the case of overflow, if the calculator display contains nine digits to the right of the decimal point and a data point containing two digits to the left of the decimal point is entered in the *x*-register, the register will overflow and display the data point in floating point notation. Exactly the same situation can occur with *d*. If you specify a *d* of nine and the data point in the *x*-register overflows, then the data point will be typed in floating point notation.

There is a potential problem in the interaction between *w* and *d* that the operator should be aware of. For example, assume that *w* has been set for two and *d* has been specified as zero. If the data point in the *x*-register overflows and an attempt is made to type the data point, the field width of two will not be large enough to contain the data point in floating notation and two asterisks will be typed.

Whereas an overflow condition which exceeds the field width will cause asterisks to be typed, an underflow condition will not. In an underflow condition, the data point in the *x*-register has become so small that the *d* specification does not allow the data point to be displayed with enough accuracy. Like the overflow condition, the data point is, in essence, not typed. For example, assume the value of the data point in the *x*-register were 1.23×10^{-11} and *d*were specified as zero. When the data point is typed it will be typed as zero.

The operator should be aware of the range in which the typed data will lie and set the w.d specification so that neither underflow or overflow occur. If the range is likely to be very great, the field width and *d* should be set so that the data point is typed in floating point notation (**16 . .**).

Because *d*is essentially stored by changing the format of the calculator display, the value for *d* remains unchanged until the display format is changed. This can be done by giving another *d* instruction or by manually pressing the FLOAT or FIX ( ) keys. If, for example, you specify a *d* of two, stop the program and press FLOAT, then, when the data is typed it will be typed in floating point notation.

The PRINT component of this instruction sequence causes the typewriter to type the data contained in the calculator's *x*-register. The data will be typed in the location and notation specified by the last w.d component given.

When a FMT, **2,** w.d, PRINT instruction sequence has been executed, the instruction sequence will automatically be terminated and the typewriter address reset. This means that if you wished to type the same data twice, the entire instruction sequence (less w.d, explained later) must be given again.

In order to provide maximum programming flexibility, certain component parts of the FMT, **2,** w.d, PRINT instruction sequence may, optionally, be omitted from the instruction sequence. The following variations of the sequence are allowable:

FMT, **2,** w.d, PRINT — Type the data in the *x*-register in a specified location and notation.

FMT, **2,** w.d — Establish a field width (*w*) and notation (*d*) which will be used later in the program by one or more FMT, **2,** PRINT instruction sequences.

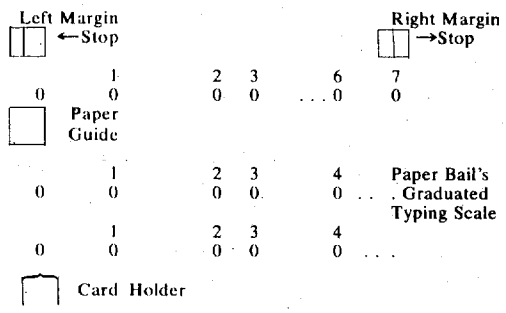
FMT, **2,** PRINT — Type the data in the *x*-register in a location and notation stored earlier in the program.

The FMT, **2,** w.d instruction sequence allows you to save program steps by specifying *w* and *d* only once in a program and reusing them any number of times with succeeding FMT, **2,** PRINT instruction sequences.

The PRINT instruction in these sequences will automatically terminate the instruction sequence and reset the typewriter address. In the case of the FMT, **2,** w.d instruction sequence, the instruction sequence is not terminated; however, the next calculator instruction, in addition to performing its normal function will terminate the sequence.

The use of the FMT, 2, w.d, PRINT instruction sequence is illustrated by the following example:
A piece of paper 8½ inches or wider is placed in the typewriter and the margin stops, paper guide and carriage set as shown below:

```
Left Margin                                  Right Margin
 ←Stop                                        →Stop

        1        2  3        6    7
 0      0        0  0    . . . 0    0
 Paper
 Guide

        1        2  3        4    Paper Bail's
 0      0        0  0        0 . . . Graduated
                                    Typing Scale
        1        2  3        4
 0      0        0  0        0 . . .

 Card Holder
```

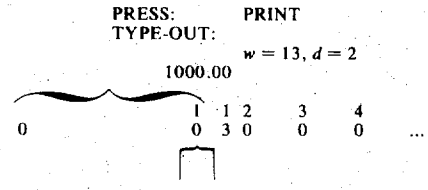


The number 1,000 is keyed into the calculator's x-register, and floating point display is selected.

```
DISPLAY:          1.000000000 03 → X
PRESS:     FMT,   2          1   3   .   2
DISPLAY:          1000.00→ X
```

The w (13) was stored and the d(2) changed the display. The instruction sequence is not terminated, however, and a PRINT instruction may be added:

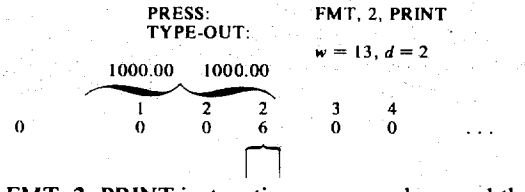
```
PRESS:        PRINT
TYPE-OUT:
                w = 13, d = 2
        1000.00
        1  1 2    3    4
 0      0  3 0    0    0    ...
```



The red pointer mark on the CARD HOLDER points to 13 on the PAPER BAIL'S graduated typing scale, not 14 as one might expect. That is because the scale starts at zero, not one.

The PRINT instruction has terminated the instruction sequence. To type the data again will require another instruction sequence:

```
PRESS:        FMT, 2, PRINT
TYPE-OUT:
                w = 13, d = 2
    1000.00  1000.00
        1    2    2    3    4
 0      0    0    6    0    0    . . .
```



The FMT, 2, PRINT instruction sequence has used the w and d stored in the preceding FMT, 2, w.d instruction sequence. The type-out is right-justified on the twenty-sixth space as the left side of the field is defined by the location of the carriage when the data is typed.

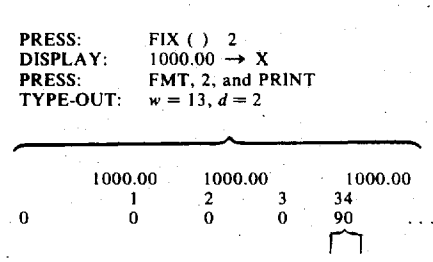
It should be noted that a FMT, 2, w.d PRINT instruction sequence does not contain the capability of a carriage return/line feed instruction. This capability does exist, however, as explained below.

A FMT, 2, w.d instruction sequence is terminated by the next calculator instruction given. In addition to terminating the FMT, 2, w.d instruction sequence, the instruction will be performed in its normal manner.

```
PRESS:      FMT,  2
              1   3   .   .
                  ↑
DISPLAY:          1.000000000   03 → X, Y
```

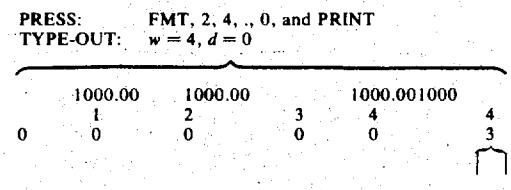
This instruction sequence has specified floating point notation; however, this display can be manually overridden with the FIX ( ) - FLOAT keys:

```
PRESS:        FIX ( )  2
DISPLAY:      1000.00 → X
PRESS:        FMT, 2, and PRINT
TYPE-OUT:     w = 13, d = 2

        1000.00    1000.00      1000.00
            1      2      3     34
 0          0      0      0     90    . . .
```



Data in the calculator's x-register can be typed without a decimal point. THis is done by specifying a $d$of zero:

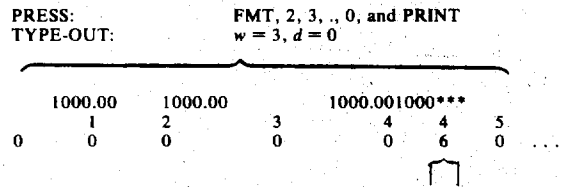
```
PRESS:        FMT, 2, 4, ., 0, and PRINT
TYPE-OUT:     w = 4, d = 0

        1000.00    1000.00      1000.001000
            1      2      3      4      4
 0          0      0      0      0      3
```



The data point just typed was typed in a field width of four, just large enough to contain the data. Spaces between this data type-out and the one previously typed could have been provided by specifying a larger field width, w.

Asterisks will be typed across a field that is not large enough to contain the data.

```
PRESS:        FMT, 2, 3, ., 0, and PRINT
TYPE-OUT:     w = 3, d = 0

        1000.00    1000.00      1000.001000***
            1      2      3      4      4      5
 0          0      0      0      0      6      0    . . . .
```



The FMT, 2, FMT, MESSAGE, FMT instruction sequence allows labels and headings to be typed for data. With this instruction sequence, the user can control

both characters on each typewriter key and control all typewriter functions directly from the calculator, either by stored program step or manually, from the calculator keyboard.

The instruction sequence consists of the following component parts:

FMT	2	— Typewriter address
FMT		— Established the TYPING MESSAGES instruction sequence.
MESSAGE		— Your label or heading
FMT		— Terminates the instruction sequence.

Other calculator instructions which terminate this instruction sequence are:

LOAD	STOP *	STEP PRGM	FLOAT
RUN	KEYLOG	RECORD	END *
BACK STEP	FIX ( )	PRGM	LIST

Programmable instructions.

In the MESSAGE component of this instruction sequence, one programmable calculator instruction will control one typewriter key. Which character on the typewriter key will be typed or which typewriter function will be performed, will depend on the mode of the calculator keyboard when the instruction is given. In the MESSAGE component of this sequence, the calculator can be in one of two modes, unshifted or shifted.

When the instructions FMT, 2, FMT are given, the calculator keyboard will automatically be in the unshifted mode. In this mode of operation, the calculator instructions and the typewriter character typed or function performed have the correspondence shown by the characters and symbols included within the keys of FIG. 12.

The unshifted calculator keyboard can control the typewriter's numbers and capital alphabet. In addition, the functions TAB CLEAR, TAB, CR/LF (Carriage return/Line feed), SPACE, BLACK RIBBON may also be performed. The FMT instruction and the blank keys shown in the diagram will terminate the instruction sequence. Three instructions (PRINT, TAB and SHIFT) available on the unshifted calculator keyboard require special mention.

A PRINT instruction placed in a MESSAGE will cause the data in the calculator's x-register to be typed. The data will be typed in the location and notation specified by the last w.d given. A w.d may not be placed in a MESSAGE; however, they are stored when they are given and automatically recalled and used with a PRINT instruction placed in a MESSAGE.

A TYPING MESSAGES instruction sequence will not affect the contents of the calculator's display registers. Therefore, data can be placed in the x-register and typed with a PRINT instruction placed in this instruction sequence.

Giving a TAB instruction will cause the typewriter carriage to move either to the next tab set or to the right margin setting. If the carriage is less than two spaces from a tab setting and a TAB instruction is given, then that tab setting will be ignored by the typewriter and the carriage will move to the next tab setting or right margin setting.

The SHIFT instruction allows one to change the mode of the calculator's keyboard. If the calculator keyboard were unshifted and a SHIFT instruction was given, then the calculator keyboard would be in the shifted mode. In the shifted mode of operation, the calculator instructions and the typewriter character typed or the function performed have the correspondence shown by the characters and symbols printed below the keys of FIG. 12.

The shifted calculator keyboard can control the typewriter's small letter alphabet and the symbols above the numbers and in addition, the functions TAB CLEAR ALL (CLEAR ALL TABS), TAB SET, CR/LF, SPACE, BACK SPACE. The FMT instruction and the blank keys shown in the diagram will terminate the instruction sequence. The operation of the PRINT and SHIFT instructions on the shifted calculator keyboard are the same as on the unshifted keyboard. The two instructions that require special mention on the shifted calculator keyboard are the CR/LF instruction and the RED RIBBON instruction.

The CR/LF instruction will cause the typewriter to perform a carriage return and line feed. In addition, this instruction will cause the calculator keyboard to become unshifted.

A RED RIBBON instruction will cause the typewriter to type in red. Once red ribbon is selected, only two things can reset the red ribbon, they are, giving a BLACK RIBBON instruction (available on the calculator's unshifted keyboard) or switching the calculator interface OFF. Terminating the TYPING MESSAGES instruction sequence wil not reset a RED RIBBON instruction.

The MESSAGE component of a TYPING MESSAGES instruction sequence may be any number of program steps (limited only by the size of the calculator memory). However, a typewritten line must not be allowed to exceed the right margin setting. If the right margin setting is exceeded, the program will stop at the second memory location following the instruction that exceeded the right margin setting and the STATUS lamp will light.

(It should be noted that if the right margin setting is exceeded in a typing data instruction sequence, the program will stop at the memory location immediately following the PRINT instruction that exceeded the right margin setting.)

Because of the two things an operator must keep track of, an example of TYPING MESSAGES is potentially confusing. For example, if the instruction A were placed in a MESSAGE, the operator could not easily predict whether the character typed would be A or a, further, the color would be unknown. Which character and which color would, of course, depend on whether the calculator keyboard were shifted or unshifted and the status of the ribbon color. The following format makes keeping track of these two things easier and, therefore, helps clarify the examples.

STEP	B	R	S	U	PROG. INST.	KEY	CODE	M	Y	Z
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										
0										
1										
2										
3										
4										
5										

The format is essentially the standard programming pad format with five additional columns: B, R, S, U and PROG INST. Mark the B and R column so that you will know the status of the ribbon color, red or black. In the S and U columns, you can mark the mode of the calculator keyboard when the instruction is given — S for shifted, U for unshifted. In the PROG INST column you can record the typewriter instruction that, considering the mode of the calculator keyboard, corresponds to the instruction in the KEY column. In the following examples, the PROG INST column will not be used except for typewriter instruction sequences.

To type messages the typewriter's LEFT MARGIN STOP and PAPER GUIDE are set to zero. The typewriter carriage is positioned against the LEFT MARGIN STOP. A piece of paper 8½ inches or wider is placed in the typewriter and the RIGHT MARGIN STOP set to 70. The number 123.45 is keyed into the calculator's *x*-register and a display mode of FIX ( ), 3 selected. In this example, the MESSAGE "The value is" will be typed followed by the data just entered into the calculator's *x*-register. This is accomplished by keying in the following instructions:

STEP	B	R	S	U	PROG. INST.	KEY	CODE	X	Y	Z
0000					FMT	FMT	42	123.450		
1					2	2	02	"		
2					7	7	07	"		
3					.	.	21	"		
4					2	2	02	123.45		
5					FMT	FMT	42	"		
6					2	2	02	"		
7					FMT	FMT	42	"		
8	■			■	BLK RIB	S/R	77	"		
9	■			■	CR/LF	CLR	20	"		
0010	■			■	T	XTO	23	"		
1	■			■	SHIFT	UP	27	"		
2	■			■	h	H	74	"		
3	■			■	e	E	60	"		
4	■			■	SPACE	CNT	47	"		
5	■			■	v	INT	64	"		
6	■			■	a	A	62	"		
7	■			■	l	L	72	"		
8	■			■	u	1/X	17	"		
9	■			■	e	E	60	"		
0020	■			■	SPACE	CNT	47	"		
1	■			■	i	I	66	"		
2	■			■	s	YTO	40	"		
3	■			■	PRINT	PNT	46	"		
4	■			■	FMT	FMT	42	"		
5										

TYPE-OUT: The value is 123.45

Steps **0000-0004**

The FMT **2** w.d instruction sequence established a field of seven with two digits to the right of the decimal point. The data point (123.45) requires a field width of six; setting the width to seven will ensure that there will be a space between the data and the last character typed.

Steps **0008-0009**

The BLK RIBBON instruction in step **0008** will ensure that the MESSAGE in this example is typed in black. Prior to that time the status of the ribbon color was unknown; therefore, neither the B or R column was filled in.

The CR/LF instruction in step **0009** will ensure the MESSAGE is typed against the LEFT MARGIN STOP.

The next example performs the same type of operation with the instruction sequences used in a slightly different manner. In this example the TYPING MESSAGES instruction sequence will be used to type the message "123.45 squared =" in black and then set the red ribbon. Next a TYPING DATA instruction sequence is used to set a field width of ten with four digits

:o the right of the decimal point, and type the data con-:ained in the *x*-register.

If the number 123.45 is not still in the *x*-register from the last example, it is entered into the *x*-register and the following calculator keys are depressed in the order given:

STEP	BRSU	PROG. INST.	KEY	CODE	x	y	z
0000			UP	27	123.45	123.45	
1			X	36	123.45	15239.9025	
2			DN	25	15239.9025	"	
3		FMT	FMT	42	"	"	
4		2	2	02	"	"	
5		FMT	FMT	42	"	"	
6		CR/LF	CLR	20	"	"	
7		1	1	01	"	"	
8		2	2	02	"	"	
9		3	3	03	"	"	
0010		.	.	21	"	"	
1		4	4	04	"	"	
2		5	5	05	"	"	
3		SPACE	CNT	47	"	"	
4		SHIFT	UP	27	"	"	
5		s	YTO	40	"	"	
6		q	b	14	"	"	
7		u	1/X	17	"	"	
8		e	A	82	"	"	
9		r	e	13	"	"	
0020		e	E	60	"	"	
1		d	D	63	"	"	
2		SPACE	CNT	47	"	"	
3		=	–	34	"	"	
4		RED RIB	S/R	77	"	"	
5		FMT	FMT	42	"	"	
6		FMT	FMT	42	"	"	
7		2	2	02	"	"	
8		1	1	01	"	"	
9		0	0	00	"	"	
0030		.	.	21	"	"	
1		4	4	04	"	"	
2		PRINT	PNT	45	"	"	
3							

In the last example, a BLK RIBBON instruction was given. Since then, the ribbon color has not been changed. Therefore, in step **0003**, the B column can be filled in even though a BLK RIBBON instruction was not given.

At step **0024** a RED RIBBON instruction is given so that the data in the *x*-register will be typed in red. Remember, the red ribbon can be reset to black ribbon by two things only, giving a BLK RIBBON instruction or turning the typewriter interface OFF.

TYPE-OUT: 123.45 Squared = 15239.9025

t may be noted that there is no space in the type-out between the "=" and the data. That can be corrected n two ways:

1. Setting the *w* specification to 11.
2. Inserting a SPACE instruction immediately following the "=" instruction.

The instruction sequence FMT, 2, LIST allows the user to list a program contained in the calculator memory. The listing will be typed in a column format with each line consisting of the memory location, the mnemonic of the instruction stored in the location and the key code of the instruction. The listing will begin at the location of the program counter when this instruction sequence is given and will continue until either an END instruction is given or STOP on the calculator keyboard is pressed.

(It should be noted that if the ribbon color status is red, the listing will be typed in red.)

This may be illustrated by entering a typewriter exerciser program into the memory of the calculator and then:

PRESS: END FMT 2 LIST
TYPE-OUT:

```
0000--UP ---27
0001-- * ---36
0002--DN ---25
0003--FMT---42
0004-- 2 ---02
0028--SFL---54
0029--S/R---77
0030--PNT---45
0031--CLR---20
0032--S/R---77
0033--END---46
```

KEY CODES AND MNEMONICS

All of the keys of the keyboard input unit and their associated octal keycodes and mnemonics are listed in Table D below. Every key has only one keycode and only one mnemonic (if it has a mnemonic), regardless of how many different functions it may be used to perform. The keycodes for the FLOAT, FIX (), RUN, PRGM, KEY LOG, LIST, LOAD, RECORD, BACK STEP, AND STEP PRGM keys are applied to the CPU in seven-bit binary form. Since these keycodes are used only to control the mode of the calculator, are not programmable, and are never printed out by the output printer unit, they have no associated mnemonic. All of the remaining keycodes are applied to the CPU in six-bit binary form, are programmable, and have a mnemonic that may be printed out by the output printer unit.

TABLE D

KEY	KEY CODE	MNE-MONIC	KEY	KEY CODE	MNE-MONIC
0	00	0	GO TO	44	GTO
1	01	1	PRINT	45	PNT
			SPACE		
2	02	2	END	46	END
3	03	3	CON-TINUE	47	CNT
4	04	4	IF $x=y$	50	X=Y
5	05	5	LABEL	51	LBL
6	06	6	IF $x<y$	52	X<Y
7	07	7	IF $x>y$	53	X>Y
8	10	8	SET FLAG 54	SFL	
9	11	9	K	55	K
x^2	12	XSG	π	56	π
$\vec{a}$	13	a	PAUSE	57	PSE
$\vec{b}$	14	b	E	60	E
G	15	G	C	61	C
F	16	F	A	62	A
$1/x$	17	1/X	D	63	D
CLEAR	20	CLR	int x	64	INT
.	21	.	I	65	I
ROLL↑	22	RUP	B	66	B
$x\rightarrow()$	23	XTO	$x\leftarrow()$	67	XFR
$y\rightleftarrows()$	24	UE	M	70	M
↓	25	DN	0	71	O
ENTER EXP	26	EEX	L	72	L
↑	27	UP	N	73	N
$x\div y$	30	XEY	H	74	H
INDIRECT	31	IND	J	75	J
CHG SIGN	32	CHS	$\sqrt{x}$	76	$\sqrt{}$
+	33	+	SUB	77	S/R
			RETURN		
−	34	−	RECORD	102	
÷	35	DIV	LOAD	103	
×	36	x	LIST	104	
CLEAR x	37	CLX	KEY LOG	105	
$y\rightarrow()$	40	YTO	PRGM	106	
STOP	41	STP	RUN	107	
FMT	42	FMT	FIX ()	110	
IF FLAG	43	IFG	FLOAT	110	

PROCESSING KEYCODES

The manner in which keycodes entered into the CPU from the keyboard input unit or from the program storage section of the memory unit are processed is shown and described in the keycode processing flow chart of FIG. **13.** Once the calculator is turned on, it operates in the display routine until a key is depressed. If a FLOAT, FIX (), RUN, PRGM, KEY LOG, LIST, LOAD, RECORD, BACK STEP, or STEP PRGM key (each having a seven-bit keycode) is depressed, the calculator operates in a director routine to determine which of these keys was depressed and thereupon selects the routine for performing the function required by that key. Upon completion of the selected routine, the calculator reverts to operation in the display routine.

If any other key (each having a six-bit keycode), except the CONTINUE key, is depressed and the PRGM key has not been depressed or has been followed by the RUN key, the calculator operates in the interpreter routine to determine which six-bit keycode it has received and to select the appropriate routine for performing the function required by that six-bit keycode. Unpon completion of the selected routine the calculator reverts to operation in the display routine.

If the CONTINUE key is depressed and the PRGM key has not been depressed or has been followed by the RUN key, the calculator operates in a fetch routine to sequentially fetch keycodes, as designated by the user program counter, from the program storage section of the RWM. Each fetched keycode is interpreted and the routine for performing the function required thereby selected in the same manner as if the fetched keycode had been received from the keyboard input unit. However, upon completion of the selected routine the calculator reverts to the fetch routine unless the fetched keycode was a STOP, END, or PAUSE (if the execution of the PAUSE is immediately followed by depression of a key), in which cases the calculator reverts to operation in the display routine.

If any key having a six-bit keycode is depressed and the PRGM key has been depressed and has not been followed by the RUN key, the calculator operates in a store routine to store the six-bit keycode received from the keyboard input unit as a program step in the program storage section of the RWM. Upon completion of this storage operation the calculator reverts to operation in the display routine.

If the KEY LOG key has been depressed and has not been followed by the RUN key or the PRGM key, the calculator also prints out each six-bit keycode received from the keyboard input unit in any of the abovementioned cases.

The display routine, the director routine, the interpreter routine, the routines for performing the various functions required by the six-bit and seven-bit keycodes, and other routines used by the calculator are shown and described in FIGS. **14–135.**

BASIC INSTRUCTION SET

Every routine and subroutine of the calculator comprises a sequence of one or more of 71 basic 16 -bit instructions listed below these 71are all implemented serially by the micro-processor in a time period which varies according to the specific instruction, to whether or not it is indirect, and to whether or not the skip condition has been met.

Upon completion of the execution of each instruction, the program counter (P register) has been incremented by one except for instructions JMP, JSM, and the skip instructions in which the skip condition has been met. The M-register is left with contents identical to the P-register. The contents of the addressed memory location and the A and B registers are left unchanged unless specified otherwise.

Memory Reference Group

The 14 memory reference instructions refer to a specific address in memory determined by the address field $-m>$, by the ZERO/CURRENT page bit, and by the DIRECT/INDIRECT bit. Page addressing and indirect addressing are both described in detail in the reference manuals for the Hewlett-Packard Model 2116 computer (hereinafter referred to as the HP 2116).

The address field $<m>$ is a 10 bit field consisting of bits **0** through **9.** The ZERO/CURRENT page bit is bit **10** and the DIRECT/INDIRECT bit is bit **15,** except for reference to the A or B register in which case bit **8** becomes the DIRECT/INDIRECT bit. An indirect reference is denoted by a <, I> following the address $<m>$.

REGISTER REFERENCE OF A OR B REGISTER: If the location <A> or <B> is used in place of $<m>$ for any memory reference instruction, the instruction will treat the contents of A or B exactly as it would the contents of location $<m>$. See the note below on the special restriction for direct register reference of A or B.

ADA	*m*, I Add to A. The contents of the addressed memory location m are added (binary add) to contents of the A register, and the sum remains in the A register. If carry occurs from bit 15, the E register is loaded with 0001, otherwise E is left unchanged.
ADB	*m*, I Add to B. Otherwise identical to ADA.
CPA	*m*, I Compare to A and skip if unequal. The contents of the addressed memory location are compared with the contents of the A register. If the two 16-bit words are different, the next instruction is skipped; that is, the P and M registers are advanced by two instead of one. Otherwise, the next instruction will be executed in normal sequence.
CPB	*m*, I Compare to B and skip is unequal. Otherwise identical to CPA.
LDA	*m*, I Load into A. The A register is loaded with the contents of the addressed memory location.
LDB	*m*, I Load into B. The B register is loaded with the contents of the addressed memory location.
STA	*m*, I Store A. The contents of the A register are stored into the addressed memory location. The previous contents of the addressed memory location are lost.
STB	*m*, I Store B. Otherwise identical to STA.
IOR	*m*, I "Inclusive OR" to A. The contents of the addressed location are combined with the contents of the A register as an "INCLUSIVE OR" logic operation.
ISZ	*m*, I Increment and Skip if Zero. The ISZ instruction adds ONE to the contents of the addressed memory location. If the result of this operation is ZERO, the next instruction is skipped; that is, the P and M registers are advanced by TWO instead of ONE. The incremental value is written back into the addressed memory location. Use of ISZ with the A or B register is limited to indirect reference; see footnote on restrictions.
AND	*m*, Logical "AND" to A. The contents of the addressed location are combined with the contents of the A register as an "AND" logic operation.
DSZ	*m*, I Decrement and Skip if Zero. The DSZ instruction subtracts ONE from the contents of the addressed memory location. If the result of this operation is zero, the next instruction is skipped. The decremented value is written back into the addressed memory location. Use of DSZ with the A or B register is limited to indirect reference; see footnote on restrictions.
JSM	*m*, I Jump to Subroutine. The JSM instruction permits jumping to a subroutine in either ROM or R/W memory. The contents of the P register is stored at the address contained in location 1777 (stack pointer). The contents of the stack pointer is incremented by one, and both M and P are loaded with the referenced memory location.
JMP	*m*, I Jump. This instruction transfers control to the contents of the addressed location. That is, the referenced memory location is loaded into both M and P registers, effecting a jump to that location.

Shift-Rotate Group

The eight shift-rotate instructions all contain a 4 bit variable shift field <*n*> which permits a shift of one through 16 bits; that is $1 \leq n \leq 16$. If <*n*> is omitted, the shift will be treated as a one bit shift. The shift code appearing in bits 8,7,6,5 is the binary code for $n-1$, except for SAL and SBL, in which cases the complementary code for $n-1$ is used.

AAR	*n* Arithmetic right shift of A. The A register is shifted right *n* places with the sign bit (bit 15) filling all vacated bit positions. That is, the *n*+1 most significant bits become equal to the sign bit.
ABR	*n* Arithmetic right shift of B. Otherwise identical to AAR.
SAR	*n* Shift A right. The A register is shifted right *n* places with all vacated bit positions cleared. That is, the *n* most significant bits become equal to zero.
SBR	*n* Shift B right. Otherwise identical to SAR.
SAL	*n* Shift A left. The A register is shifted left *n* places with the *n* least significant bits equal to zero.
SBL	*n* Shift B left. Otherwise identical to SAL.
RAR	*n* Rotate A right. The A register is rotated right *n* places, with bit 0 rotated around to bit 15.
RBR	*n* Rotate B right. Otherwise identical to RAR.

Alter-Skip Group

The 16 alter-skip instructions all contain a 5-bit variable skip field <*n*> which, upon meeting the skip condition, permits a relative branch to any one of 32 locations. Bits **9, 8, 7, 6, 5** are coded for positive or negative relative branching in which the number <*n*> is the number to be added to the current address, (skip in *forward* direction), and the number <-*n*> is the number to be subtracted from the current address, (skip in negative direction). If <*n*> is omitted, it will be interpreted as a ONE.

<*n*>=0	CODE=00000	REPEAT SAME INSTRUCTION
<*n*>=1	CODE=00001	DO NEXT INSTRUCTION
<*n*>=2	CODE=00010	SKIP ONE INSTRUCTION
<*n*>=15	CODE=01111	ADD 15 TO ADDRESS
<*n*>=-1	CODE=11111	DO PREVIOUS INSTRUCTION
<*n*>=-16	CODE=10000	SUBTRACT 16 FROM ADDRESS
<*n*>=nothing	CODE=00001	DO NEXT INSTRUCTION

The alter bits consist of bits **10** and bits **4.** The letter <*s*> following the instruction places a ONE in bit **10** which causes the tested bit to be *set* after the test. Similarly the letter <*c*> will place a ONE in bit **4** to *clear* the test bit. If both a set and clear bit are given, the set will take precedence. Alter bits do not apply to SZA, SZB, SIA, and SIB.

SZA	*n* Skip if A zero. If all 16 bits of the A register are zero, skip to location defined by *n*.
SZB	*n* Skip if B zero. Otherwise identical to SZA.
RZA	*n* Skip if A not zero. This is a "Reverse Sense" skip of SZA.
RZB	*n* Skip if B not zero. Otherwise identical to RZA.
SIA	*n* Skip if A zero; then increment A. The A register is tested for zero, then incremented by one. If all 16 bits of A were zero before incrementing, skip to location defined by *n*.
SIB	*n* Skip if B zero; then increment B. Otherwise identical to SIA.
RIA	*n* Skip if A not zero; then increment A. This is a "Reverse Sense" skip of SIA.
RIB	*n* Skip if B not zero; then increment B. Otherwise identical to RIA.
SLA	*n*, S/C Skip if Least Significant bit of A is zero. If the least significant bit (bit 0) of the A register is zero, skip to location defined by *n*. If either S or C is present, the test bit is altered accordingly after test.
SLB	*n*, S/C Skip if Least Significant bit of B is zero. Otherwise identical to SLA.
SAM	*n*, S/C Skip if A is Minus. If the sign bit (bit 15) of the A register is a ONE, skip to location defined by *n*. If either S or C is present, bit 15 is altered after the test.
SBM	*n*, S/C Skip if B is Minus. Otherwise identical to SAM.
SAP	*n*, S/C Skip if A is Positive. If the sign bit (bit 15) of the A register is a ZERO, skip to location defined by *n*. If either S or C is present, bit 15 is altered after the test.
SBP	*n*, S/C Skip if B is Positive. Otherwise identical to SAP.
SES	*n*, S/C Skip if Least Significant bit of E is Set. If bit 0 of the E register is a ONE, skip to location defined by *n*. If either S or C is present, the entire E register is set or cleared respectively.
SEC	*n*, S/C Skip if Least Significant bit of E is Clear. If bit 0 of the E register is a ZERO, skip to location defined by *n*. If either S or C is present, the entire E register is set or cleared respectively.

Complement Execute-DMA Group.

These seven instructions include complement operations and several special-purpose instructions chosen to speed up printing and extended memory operations.

CMA	Complement A. The A register is replaced by its One's complement.
CMB	Complement B. The B register is replaced by its One's complement.
TCA	Two's Complement A. The A register is replaced by its One's Complement and incremented by one.
TCB	Two's complement B. The B register is replaced by its One's Complement and incremented by one.
EXA	Execute A. The contents of the A register are treated as the current instruction, and executed in the normal manner. The A register is left unchanged unless the instruction code causes A to be altered.
EXB	Execute B. Otherwise identical to EXA.
DMA	Direct Memory Access. The DMA control in Extended Memory is enabled by setting the indirect bit in M and giving a WTM instruction. The next ROM clock transfers A→M and the following two cycles transfer B→M. ROM clock then remains inhibited until released by DMA control.

Note: Special Restriction for Direct Register Reference of A or B

For the five register reference instructions which involve a write operation during execution, a register reference to A or B must be restricted to an INDIRECT reference. These instructions are STA, STB, ISZ, DSZ, and JSM. A DIRECT register reference to A or B with these instructions may result in program modification. (This is different from the hp 2116 in which a memory reference to the A or B register is treated as a reference to locations O or 1 respectively.) A reference to location 0 or 1 will actually refer to locations 0 or 1 in Read Only Memory.

Input/Output Group (IOG)

The 11 IOG instructions, when given with a select code, are used for the purpose of checking flags, setting or clearing flag and control flip-flops, and transferring data between the A/B registers and the I/O register.

STF <SC>	Set the flag. Set the flag flip-flop of the channel indicated by select code <SC>.
CLF <SC>	Clear the flag flip-flop of the channel indicated by select code <SC>.
SFC <SC>	Skip if flag clear. If the flag flip-flop is clear in the channel indicated by <SC>, skip the next instruction.
SFS <SC> H/C	Skip if flag set. If the flag flip-flop is set in the channel indicated by <SC>, skip the next instruction. H/C indicates if the flag flip-flop should be held or cleared after executing SFS.
CLC <SC> H/C	Clear control. Clear the control flip-flop in the channel indicated by <SC>. H/C indicates if the flag flip-flop should be held or cleared after executing CLC.
STC <SC> H/C	Set Control. Set the control flip-flop in the channel indicated by <SC>. H/C indicates if the flag flip-flop should be held or cleared after executing STC.
OT* <SC> H/C	Output A or B. Sixteen bits from the A/B register are output to the I/O register. H/C allows holding or clearing the flag flop after execution of OT*. The different select codes allow different functions to take place after loading the I/O register.
SC=00	Data from the A or B register is output eight bits at a time for each OT* instruction given. The A or B register is rotated right eight bits.
SC=01	The I/O register is loaded with 16 bits from the A/B registers.
SC=02	Data from the A/B register is output one bit at a time for each OT* instruction for the purpose of giving data to the Magnetic Card Reader. The I/O register is unchanged.
SC=04	The I/O register is loaded with 16 bits from the A/B register and the control flip flop for the printer is then set.
SC=08	The I/O register is loaded with 16 bits from the A/B register and the control flip flop for the display is then set.
SC=16	The I/O register is loaded with 16 bits from the A/B register and then data in the I/O register is transferred to the switch latches.
LI* <01> H/C	Load into A or B. Load 16 bits of data into the A/B register from the I/O register. H/C allows holding or clearing the flag flop after LI* has been executed.
LI* <00>	The least significant 8 bits of the I/O register are loaded into the most significant locations in the A or B register.
MI* <01> H/C	Merge into A or B. Merge 16 bits of data into the A/B register from the I/O register by "inclusive or". H/C allows holding or clearing the flag flop after MI* has been executed.
MI* <00>	The least significant 8 bits of the I/O register are combined by inclusive OR with the least significant 8 bits of the A or B register, and rotated to the most significant bit locations of the A or B register.

MAC Instruction Group

A total of 16 MAC instructions are available for operation

a. with the whole floating-point data (like transfer, shifts, etc), or
b. with two floating-point data words to speed up digit and word loops in arithmetic routines.

NOTE:

$<A_{0-3}>$	means: contents of A-register bit 0 to 3
AR 1	is a mnemonix for arithmetic pseudo-register located in R/W memory on addresses 1744 to 1747 (octal)
AR 2	is a mnemonix for arithmetic pseudo-register located in R/W memory on addresses 1754 to 1757 (octal)
D_i	means: mantissas *i*-th decimal digit; most significant digit is D1 least significant digit is D12 decimal point is located between D1 and D2

Every operation with mantissa means BCD-coded decimal operation.

RET	Return 16-bit-number stored at highest occupied address in stack is transferred to P- and M-registers. Stack pointer (=next free address in stack) is decremented by one. <A>, <B>, <E> unchanged.
MOV	Move overflow The Contents of E-register is transferred to A_{0-3}. Rest of A-register and E-register are filled by zeros. <B> unchanged.
CLR	Clear a floating-point data register in R/W memory on location <A> ZERO→<A>, <A>+1, <A>+2, <A>+3 <A>, >B>, <E> unchanged
XFR	Floating-point data transfer in R/W memory from location <A> to location <B> Routine starts with exponent word transfer. Data on location <A> is unchanged. <E> unchanged.
MRX	AR1 mantissa is shifted to right *n*-times. Exponent word remains unchanged. $<B_{0-3}> = n$ (binary coded) 1st shift: $<A_{0-3}> \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; D_{12} is lost *j*th shift: $\theta \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; D_{12} is lost *n*th shift: $\theta \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; $D_{12} \rightarrow A_{3-0}$ $\theta \rightarrow E$, A_{4-15} each shift: $<B_{0-3}> - 1 \rightarrow B_{0-3}$ $<B_{4-15}>$ unchanged
MRY	AR2 mantissa is shifted to right *n*-times. Otherwise identical to MRX
MLS	AR2 mantissa is shifted to left once. Exponent word remains unchanged. $\theta \rightarrow D_{12}$; $D_i \rightarrow D_{i-1}$; $D_1 \rightarrow A_{0-3}$ <B> unchanged
DRS	AR1 mantissa is shifted to right once Exponent word remains unchanged $\theta \rightarrow D_1$; $D_i \rightarrow D_{i+1}$; $D_{12} \rightarrow A_{0-3}$ ZERO → E and A_{4-15} <B> unchanged
DLS	AR1 mantissa is shifted to left once. Exponent word remains unchanged. $<A_{0-3}> \rightarrow D_{12}$; $D_i \rightarrow D_{i-1}$; $D_i \rightarrow A_{0-3}$

—Continued

$\theta \rightarrow E, A_{4-15}$
<B> unchanged

FXA Fixed-point addition
Mantissas in pseudo-registers AR2 and AR1 are added together and result is placed into AR2. Both exponent words remain unchanged. When overflow occurs "0001" is set into E-reg., in opposite case <E> will be zero.
<AR2> + <AR1> + DC → AR2
DC = θ if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
<B>, <AR1> unchanged

FMP Fast multiply
Mantissas in pseudo-registers AR2 and AR1 are added together $<B_{0-3}>$-times and result is placed into AR2. Total decimal overflow is placed to A_{0-3}. Both ex- ponent words remain unchanged.
<AR2> + <AR1> * $<B_{0-3}>$+DC → AR2
DC = 0 if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
ZERO → E, A_{4-15}
<AR1> unchanged

FDV Fast divide
Mantissas in pseudo-registers AR2 and AR1 are added together so many times until first decimal overflow occurs. Result is placed into AR2. Both exponent words remain unchanged. Each addition without overflow causes +1 increment of <B>.
1st addition: <AR2> + <AR1> = DC → AR2
DC = 0 if <E> was 0000 before routine execution
DC = 1 if <E> was 1111 before routine execution
next additions: <AR2>+ <AR1> → AR2
ZERO → E
<AR1> unchanged

CMX 10's complement of AR1 mantissa is placed back to AR1, and ZERO is set into E-register. Exponent word remains unchanged
<B> unchanged

CMY 10's complement of AR2 mantissa.
Otherwise identical to CMX

MDI Mantissa decimal increment.
Mantissa on location <A> is incremented by decimal ONE on D_{12} level, result is placed back into the same location, and zero is set into E-reg.
Exponent word is unchanged.
When overflow occurs, result mantissa will be
1,000 0000 0000 (dec)
and 0001 (bin) will be set into E-reg.
<B> unchanged.

NRM Normalization
Mantissa in pseudo-register AR2 is rotated to the left to get $D_1 \neq 0$. Number of these 4-bit left shifts is stored in B_{0-3} in binary form ($<B_{4-15}>=0$)
when $<B_{0-3}>=0,1,2, \ldots, 11$ (dec) ➡, <E>=0000
When $<B_{0-3}>=12$ (dec) ➡ mantissa is zero, and <E>=
0001
Exponent word remains unchanged
<A>unchanged.

The binary codes of all of the above instructions are listed in the following coding table, where * implies the A or B register, D/I means direct/indirect, A/B means A register/B register, Z/C means zero page (base page) (current page, H/S means hold test bit/set test bit, and H/C means hold test bit/clear test bit. D/I, A/B, Z/C, H/S, and H/C are all coded as 0/1.

CODING TABLE

GROUP	OCTAL	INSTR	15	14 13 12	11 10	9 8 7 6 5 4 3 2 1 0
MEMORY	-0----	AD*	D/I	0 0 0	A/B Z/C	← MEMORY ADDRESS →
REFERENCE GROUP	-1----	CP*	D/I	0 0 1	A/B Z/C	
	-2----	LD*	D/I	0 1 0	A/B Z/C	
	-3----	ST*	D/I	0 1 1	A/B Z/C	
	-4----	IOR	D/I	1 0 0	0 Z/C	
	-4----	ISZ	D/I	1 0 0	1 Z/C	
	-5----	AND	D/I	1 0 1	0 Z/C	
	-5----	DSZ	D/I	1 0 1	1 Z/C	
	-6----	JSM	D/I	1 1 0	0 Z/C	
	-6----	JMP	D/I	1 1 0	1 Z/C	
SHIFT-	07---0	A*R	0	1 1 1	A/B --	SHIFT ← → CODE — 0 0 0 0
ROTATE	07---2	S*R	0	1 1 1	A/B --	0 0 1 0
GROUP	07---4	S*L	0	1 1 1	A/B --	0 1 0 0
	07---6	R*R	0	1 1 1	A/B --	0 1 1 0
ALTER-	07---0	SZ*	0	1 1 1	A/B O	0 1 0 0
SKIP	07---0	RZ*	0	1 1 1	A/B 1	0 1 0 0
GROUP	07---0	SI*	0	1 1 1	A/B 0	1 1 0 0 0
	07---0	RI*	0	1 1 1	A/B 1	1 1 0 0 0
	07---1	SL*	0	1 1 1	A/B H/S	H/C 1 0 0
	07---2	S*M	0	1 1 1	A/B H/S	H/C 1 0 1
	07---3	S*P	0	1 1 1	A/B H/S	H/C 1 0 1
	07---4	SES	0	1 1 1	A/B H/S	H/C 1 1 0
	07---5	SEC	0	1 1 1	A/B H/S	H/C 1 1 0
REGISTER	07--17	ADA	0	1 1 1	A/B --	D/I 0 0 0 1 1 1
REFERENCE	07--37	ADB	0	1 1 1	A/B --	D/I 0 0 1 1 1 1
GROUP	07--57	CPA	0	1 1 1	A/B --	D/I 0 1 0 1 1 1
	07--77	CPB	0	1 1 1	A/B --	D/I 0 1 1 1 1 1
	07--17	LDA	0	1 1 1	A/B --	D/I 1 0 0 1 1 1
	07--37	LDB	0	1 1 1	A/B --	D/I 1 0 1 1 1 1
	07-557	STA	0	1 1 1	A/B --	1 0 1 1 0 1 1 1
	07-577	STB	0	1 1 1	AB--	1 0 1 1 1 1 1 1
	0--17	IOR	0	1 1 1	A/B --	D/I 1 0 0 0 1 1 1
	07-637	ISZ	0	1 1 1	A/B --	1 1 0 0 1 1 1 1

CODING TABLE – Continued

GROUP	OCTAL	INSTR	15	14 13 12	11 10	9 8 7 6 5 4 3 2 1 0
	07--57	AND	0	1 1 1	A/B --	D/I 1 0 1 0 1 1 1 1 1
	07-677	DSZ	0	1 1 1	A/B --	1 1 0 1 1 1 1 1 1 1
	07-717	JSM	0	1 1 1	A/B --	1 1 1 0 0 1 1 1 1
	07--37	JMP	0	1 1 1	A/B --	D/I 1 1 0 1 1 1 1 1
COMP	07-016	EX*	0	1 1 1	A/B -	---- 0 0 1 1 1 1 0
EXECUTE	070036	DMA	0	1 1 1	0 -	---- 0 1 1 1 1 0
DMA	07-056	CM*	0	1 1 1	A/B -	---- 1 0 1 1 1 0
	07-076	TC*	0	1 1 1	A/B -	---- 1 1 1 1 1 0
INPUT	1727--	STF	1	1 1 1	- 1	0 1 1 1 1 SELECT CODE
OUTPUT GROUP	1737--	CLF	1	1 1 1	- 1	1 1 1 1 1
	17-7--	SFC	1	1 1 1	- 1	H/C 1 1 1 0
	17-5--	SFS	1	1 1 1	- 1	H/C 1 0 1 0
	17-5--	CLC	1	1 1 1	- 1	H/C 1 0 1 1
	17-6--	STC	1	1 1 1	- 1	H/C 1 1 0 0
	17-1--	OT*	1	1 1 1	A/B 1	H/C 0 0 1 1
	17-2--	LI*	1	1 1 1	A/B 1	H/C 0 1 0 1
	17-0--	MI*	1	1 1 1	A/B 1	H/C 0 0 0 1
MAC	170402	RET	1	1 1 1	0 0	0 1 0 0 0 0 0 0 0 1 0
GROUP	170002	MOV	1	1 1 1	0 0	0 0 0 0 0 0 0 0 0 1 0
	170000	CLR	1	1 1 1	0 0	0 0 0 0 0 0 0 0 0 0 0
	170004	XFR	1	1 1 1	0 0	0 0 0 0 0 0 0 0 1 0 0
	174430	MRX	1	1 1 1	1 0	0 1 0 0 0 1 1 0 0 0
	174470	MRY	1	1 1 1	1 0	0 1 0 0 1 1 1 0 0 0
	171400	MLS	1	1 1 1	0 0	1 1 0 0 0 0 0 0 0 0 0
	170410	DRS	1	1 1 1	0 0	0 1 0 0 0 0 0 1 0 0 0
	175400	DLS	1	1 1 1	1 0	1 1 0 0 0 0 0 0 0 0 0
	170560	FXA	1	1 1 1	0 0	1 1 0 0 1 1 0 0 0 0 0
	171460	FMP	1	1 1 1	0 0	1 1 0 0 1 1 0 0 0 0 0
	170420	FDV	1	1 1 1	0 0	0 1 0 0 0 1 0 0 0 0 0
	174400	CMX	1	1 1 1	1 0	0 1 0 0 0 0 0 0 0 0 0
	170400	CMY	1	1 1 1	0 0	0 1 0 0 0 0 0 0 0 0 0
	170540	MDI	1	1 1 1	0 0	0 1 0 1 1 0 0 0 0 0 0
	171450	NRM	1	1 1 1	0 0	1 1 0 0 1 0 1 0 0 0 0

DETAILED LISTING OF ROUTINES AND SUBROUTINES OF BASIC INSTRUCTIONS

A complete listing of all of the routines and subroutines of basic instructions employed by the calculator and of all of the constants employed by these routines and subroutines is given below. All of these routines, subroutines, and constants are stored either in the basic ROM or in the plug-in ROM modules employed therewith. Each page of each different group of routines, subroutines, and constants is numbered at the top left-hand corner of the page, and each line of each page is separately numbered in the first column from the left-hand side of the page. This facilitates reference to different parts of the listing. Descriptive headings are also provided throughout the listing to identify routines, subroutines, groups of constants, different portions of the ROM, the plug-in ROM modules, etc. Each instruction of each routine or subroutine and each constant stored in the ROM or plug-in ROM modules is represented in octal form by six digits in the third column from the left-hand side of the page, and the address of the ROM location in which each such instruction or constant is stored is represented in octal form by five digits in the second column from the left-hand side of the page.

Mnemonic labels serving as symbolic addresses or names are given in the fourth column from the left-hand side of the page for most of the constants and many of the instructions to facilitate references to these constants and instructions and associated instructions. The mnemonic code of each basic instruction and of each pseudo instruction is given in the fifth column from the left-hand side of the page. As noted above, each basic instruction is employed as a step in a routine or subroutine of one or more basic instructions and therefore has an address in the ROM. Psuedo instructions such as ORG, EQU, etc. which appear (and are recognizable as not being one of the 71 basic machine instructions listed above) are used for control of the Assembler, which translates the symbolic/mnemonic coding of the fourth, fifth, and sixth columns into the address and contents of ROM registers which appear in the second and third columns. (See chapter 4 of the Hewlett-Packard "Assembler Programmer's Reference Manual" of April, 1970.) They are not employed as steps in the routines and subroutines performed by the

calculator and therefore have no addresses in the ROM. Mnemonic operand codes are given in the sixth column from the left-hand side of the page, and descriptive comments are given to the right of the sixth column. The format, assembly, and use of the listing is explained in greater detail in the above-mentioned Hewlett-Packard "Assembler Programmer's Reference Manual".

CONSTANT AREA INITIATOR CIN

```
0001                    ASMB,A,B,L
0003  00003                        ORG 3B
0004  00003 000004 XADDP           OCT 4
0005  00004 001750 XADD            OCT 1750
0006  00005 001740 YADD            OCT 1740
0007  00006 001760 ZADD            OCT 1760
0008  00007 001764 AADD            OCT 1764
0009  00010 001770 BADD            OCT 1770
0010  00011 001720 T1AD            OCT 1720
0011  00012 001724 T2AD            OCT 1724
0012  00013 001730 T3AD            OCT 1730
0013  00014 001734 T4AD            OCT 1734
0014  00015 001744 AR1AD           OCT 1744
0015  00016 001754 AR2D            OCT 1754
0016  00017 000000 DISP            BSS 1
0017  00020 000021 DPNT            OCT 21
0018  00021 000041 STOP            OCT 41
0019  00022 000047 CONTK           OCT 47
0020  00023 177773 M5              DEC -5
0021  00024 177766 M10             DEC -10
0022  00025 177577 CMSK            OCT 177577
0023  00026 176777 PMSK            OCT 176777
0024  00027 001000 RMSK            OCT 1000
0025  00030 077760 FMSK            OCT 77760
0026  00031 177777 M1              DEC -1
0027  00032 177000 MMSK            OCT 177000
0028  00033 001660 RBAD            OCT 1660
0029  00034 000003 .3              DEC 3
0030  00035 000000 NEXE            BSS 1
0031  00036 000000 NUMBE           BSS 1
0032  00037 001710 RBAD1           OCT 1710
0033  00040 000000 RINC1           BSS 1
0034  00041 001716 INTA            OCT 1716
0035  00042 000000 OPER1           BSS 1
0036  00043 000177 KMSK            OCT 177
0037  00044 001712 SSP             OCT 1712
0038  00045 111000 INM             OCT 111000
0039  00046 001702 USP             OCT 1702
0040  00047 012014 UPC             OCT 12014
0041  00050 000000 RLA             BSS 1

0043  00001                        ORG 1B
0044  00001 043060                 OCT 43060
0045  12000                        ORG 12000B
0046                               REP 12
0047  12000 000000                 OCT 0
0047  12001 000000                 OCT 0
0047  12002 000000                 OCT 0
0047  12003 000000                 OCT 0
0047  12004 000000                 OCT 0
0047  12005 000000                 OCT 0
0047  12006 000000                 OCT 0
0047  12007 000000                 OCT 0
0047  12010 000000                 OCT 0
0047  12011 000000                 OCT 0
0047  12012 000000                 OCT 0

0001                    ASMB,A,B,L
0003  00003                        ORG 3B
0004  00003 000004 XADDP           OCT 4
0005  00004 001750 XADD            OCT 1750
0006  00005 001740 YADD            OCT 1740
0007  00006 001760 ZADD            OCT 1760
0008  00007 001764 AADD            OCT 1764
```

CONSTANT AREA INITIATOR CIN
—Continued

```
0009  00010 001770 BADD            OCT 1770
0010  00011 001720 T1AD            OCT 1720
0011  00012 001724 T2AD            OCT 1724
0012  00013 001730 T3AD            OCT 1730
0013  00014 001734 T4AD            OCT 1734
0014  00015 001744 AR1AD           OCT 1744
0015  00016 001754 AR2D            OCT 1754
0016  00017 000000 DISP            BSS 1
0017  00020 000021 DPNT            OCT 21
0018  00021 000041 STOP            OCT 41
0019  00022 000047 CONTK           OCT 47
0020  00023 177773 M5              DEC -5
0021  00024 177766 M10             DEC -10
0022  00025 177577 CMSK            OCT 177577
0023  00026 176777 PMSK            OCT 176777
0024  00027 001000 RMSK            OCT 1000
0025  00030 077760 FMSK            OCT 77760
0026  00031 177777 M1              DEC -1
0027  00032 177000 MMSK            OCT 177000
0028  00033 001660 RBAD            OCT 1660
0029  00034 000003 .3              DEC 3
0030  00035 000000 NEXE            BSS 1
0031  00036 000000 NUMBE           BSS 1
0032  00037 001710 RBAD1           OCT 1710
0033  00040 000000 RINC1           BSS 1
0034  00041 001716 INTA            OCT 1716
0035  00042 000000 OPER1           BSS 1
0036  00043 000177 KMSK            OCT 177
0037  00044 001712 SSP             OCT 1712
0038  00045 111000 INM             OCT 111000
0039  00046 001702 USP             OCT 1702
0040  00047 012014 UPC             OCT 12014
0041  00050 000000 RLA             BSS 1

0043  00001                        ORG 1B
0044  00001 043060                 OCT 43060
0045  12000                        ORG 12000B
0046                               REP 12
0047  12000 000000                 OCT 0
0047  12001 000000                 OCT 0
0047  12002 000000                 OCT 0
0047  12003 000000                 OCT 0
0047  12004 000000                 OCT 0
0047  12005 000000                 OCT 0
0047  12006 000000                 OCT 0
0047  12007 000000                 OCT 0
0047  12010 000000                 OCT 0
0047  12011 000000                 OCT 0
0047  12012 000000                 OCT 0
0047  12013 000000                 OCT 0
0049  00051                        ORR
0050  00051 000000 NINCR           BSS 1
0051  00052 000000 CLR1A           BSS 1
0052  00053 000200 TBLE            OCT 200
0053  00054 177774 M4              DEC -4
0054*
0055  00055 162000 MAX             OCT 162000
0056*
0057  00056 000065 ONAD            OCT 65
0058  00057 114631 NINE            OCT 114631
0059  00060 174013 MPWD            DEC -2037
0060  00061 000044 GTOK            OCT 44
0061  00062 000051 LBLK            OCT 51
0062  00063 000001 .1              OCT 1
0063  00064 000070 PIAD            OCT 70
0064  00065 000000 ONE             OCT 0
0065  00066 010000                 OCT 10000
0066  00067 000000                 OCT 0
0067  00070 000000                 OCT 0
0068  00071 030501 PI              OCT 30501
0069  00072 054446                 OCT 54446
0070  00073 051540                 OCT 51540
0071  00074 000026 EEXK            OCT 26
```

—Continued

```
CONSTANT AREA INITIATOR   CIN
0072  00075 000032 CNSK    OCT 32
0073  00076 000000 EEXA    BSS 1
0074  00077 177775 M3      DEC -3
0075  00100 000077 GSBK    OCT 77
0076  00101 177400 MASK    OCT 177400
0077  00102 012001 NORM    OCT 12001
0078  00103 012000 EXCF    OCT 12000
0079  00104 000012 .10     DEC 10
0080  00105 000400 S1      OCT 400
0081  00106 000014 .12     DEC 12
0082  00107 012002 PRTF    OCT 12002
0083  00110 000045 PRTK    OCT 45
0084  00111 012013 SECW    OCT 12013
0085  00112 001707 USTKL   OCT 1707
0086  00113 002000 KLOG    OCT 2000
0087  00114 002000 CLRM    OCT 2000
0088                       END
*  NO ERRORS*

0001                ASMB,A,B,L,T
[illegible]   004775
IFXY   005015
IFCR   005055
IFLG   005062
GOTO   005070
GTOP   005071
GTO2   005102
GTO3   005110
SBT    005115
GSUB   005135
LABEL  005144
SRCH   005157
SRCH1  005161
LBLF   005165
LBLF2  005173
LBLF1  005175
GSBP   005204
SUBR   005210
SUBRP  005213
GSB    005222
SFLG   005225
GINC1  005232
RFLG   001723
ATMP   001717
MDEC   004176
EMST   004110
KBAS   001707
SUBF   001721
USTK   001710
USTKL  000112
USTKB  000046
ERROR  004750
MINUS  004263
PCNT   001701
INIT   000167
M1     000031
SPTR   001777
M5     000023
YADD   000005
FTMP   000014
XADD   000004
Y      001740
FT     001734
D3     000034
D4     000003
.2     005233
.3     000034
M3     000077
M4     000054
PMODE  001776
GSBK   000100
MSK5   005234
S1     000105
.1     000063
AR1AD  000015
AR2AD  000016
AR1    001744
AR2    001754
MPWD   000060
KBUF   001725
TEMP1  001726
TEMP2  001727
GTOK   000061
LBLK   000062
ADDRC  001720
GETK   000115
GSBA   005235
FLMS   005236
CLFM   000025
MASK   000101
IBUF   001775
EXCF   000103
DISP   000017
NUMF   001722
X      001750
CLN    000101
MAX    000055

*  NO ERRORS*
```

PROGRAMMABLE OPTION PROP

```
0001                ASMB,A,B,L,T
0003*THE FOLLWING ROUTINES PERFORM THE
0004*PROGRAM FUNCTIONS
0005*
0006*
0007  04775                ORG 4775B
0008*
0009*THE FOLLWING SUBROUTINES EXECUTE THE IF COMMANDS
0010*


0012*ROUND X AND Y
0013  04775 024016 RND     LDB AR2AD
0014  04776 170004         XFR                TRANSFER VALUE TO AR2
0015  04777 070075         SEC *+1,C     CLEAR E
0016  05000 170560         FXA           ADD 50 TO AR2
0017  05001 070355         SEC *+7        SKIP ON NO OVERFLOW
0018  05002 020105         LDA S1
0019  05003 001754         ADA AR2       INCREMENT THE EXP
0020  05004 031754         STA AR2
```

—Continued

```
                PROGRAMMABLE OPTION      PROP
0021  05005 020063         LDA .1
0022  05006 024063         LDB .1
0023  05007 174470         MRY            PUSH IN THE OVERFLOE
0024  05010 021757         LDA AR2+3      CLEAN THE GARD DIGITS
0025  05011 050101         AND CLN
0026  05012 031757         STA AR2+3
0027  05013 020016         LDA AR2AD
0028  05014 170402         RET

0030  05015 020015 IFXY    LDA AR1AD
0031  05016 170000         CLR                 CLEAR AR1
0032  05017 023234         LDA MSK5
0033  05020 031747         STA AR1+3           PUT IN 5 MASK
0034  05021 020005         LDA YADD
0035  05022 062775         JSM RND    ROUND Y
0036  05023 024005         LDB YADD
0037  05024 170004         XFR        TRANSFER THE NEW VALUE OF Y
0038  05025 020016         LDA AR2AD
0039  05026 024014         LDB FTMP
0040  05027 170004         XFR        PUT THE VALUE IN FLOATING TEMPORARY
0041  05030 020004         LDA XADD
0042  05031 062775         JSM RND    ROUND X
0043  05032 024004         LDB XADD
0044  05033 170004         XFR        PUT ROUNDED VALUE BACK IN X
0045  05034 020004         LDA XADD      LOAD THE ADDRESS OF X
0046  05035 024014         LDB FTMP
0047  05036 062263         JSM MINUS     SUBTRACT X FROM Y
0048  05037 021734         LDA FT        LOAD THR RESULT'S EXPONENT
0049  05040 025725         LDB KBUF      RETREIVE THE INPUT KEY
0050  05041 074151         SLB *+3       X>Y CONDITION?
0051  05042 070551         SLA *+11      YES,SKIP IF FALSE
0052  05043 067070         JMP GOTO      PROCESS TRUE CONDITION
0053  05044 074002         SBR 1         X<Y CONDITION?
0054  05045 074311         SLB *+6       NO,X=Y
0055  05046 070131         SLA *+2,C     YES,POSSIBLE TRUE
0056  05047 067055         JMP INCR      FALSE,ADJUST PROG COUNTER
0057  05050 021735         LDA FT+1      LOAD THE 1ST MAG WORD
0058  05051 072750         SZA GOTO      TRUE IF NON-ZERO
0059  05052 067055         JMP INCR      PROCESS FALSE CONDITION
0060  05053 021735         LDA FT+1      LOAD THE 1ST MAG WORD
0061  05054 070614         SZA GOTO      TRUE IF MAG=0
0062  05055 025701 INCR    LDB PCNT   LOAD THE PROG COUNTER
0063  05056 004003         ADB D4     ADD 4 TO IT
0064  05057 035701         STB PCNT
0065  05060 035720          STB 1720B  CLEAR THE CHANGE SIGN FLAG
0066  05061 067115         JMP SGT


0068*IF FLAG
0069*
0070  05062 021776 IFLG    LDA PMODE     LOAD THE STATUS WORD
0071  05063 053236         AND FLMS      ISOLATE THE FLAG BIT
0072  05064 071450         SZA INCR      SKIP IF NOT SET
0073  05065 021776         LDA PMODE     RE-LOAD THE STATUS WORD
0074  05066 050025         AND CLFM      CLEAR THE FLAG
0075  05067 031776         STA PMODE     STORE THE NEW STATUS


0077*
0078* GO TO PROCESSING
0079*
0080*THE GO TO IS PROCESSED USING
0081*TWO ENTRIES. THE 1ST ENTRY
0082*ACCEPTS THE GOTO KEYSTROKE.
0083*THE 2ND ENTRY ACCEPTS THE 1ST,2ND, AND 3RD
0084*DESTINATION DIGIT
0085*
0086*
0087* FIRST ENTRY
0088*
0089  05070 027232 GOTO    LDB GINC1   SET THE NEXT
0090  05071 035723 GTOR    STB RFLG    ENTRY'S ADDRESS
0091  05072 020054         LDA M4    LOAD -4
```

—Continued

```
                PROGRAMMABLE OPTION      PROP
0092  05073 031720          STA ADDRC      INITIALIZE THE DIGIT COUNTER
0093  05074 074742          SBR 16       CLEAR THE RESULT REGISTER
0094  05075 035717          STB ATMP
0095  05076 024031          LDB M1         PUT -1 IN SUB/RET FLAG
0096  05077 035721          STB SUBF
0097  05100 170402          RET            RETURN
0098*
0099*NON-FIRST ENTRY
0100  05101 067110          JMP GTO3       TERMINATION EXIT
0101*
0102  05102 021725 GTO2     LDA KBUF       LOAD THE INPUT KEY
0103  05103 062176          JSM MDEC       FIND THE BINARY EQUIVALENT
0104  05104 045720          ISZ ADDRC        3 DIGITS IN?
0105  05105 170402          RET            NO,GO BACK FOR MORE
0106*
0107*
0108*EXECUTION ENTRY
0109*
0110  05106 031722          STA NUMF        RESET THE NUMBER FLAG
0111  05107 045701          ISZ PCNT        INCREMENT THE PROG COUNTER
0112  05110 055701 GTO3     DSZ PCNT        DECREMENT THE PROG COUNTER
0113  05111 025717          LDB ATMP          LOAD THE ADDRESS VALUE
0114  05112 025707          ADB KBAS        FIND ABSOLUTE ADDRESS
0115  05113 035721          ISZ SUBF      SUBROUTINE JUMP?
0116  05114 165721          JMP SUBF,I     YES,CHHANNEL TO GOSUB
0117  05115 074117 SGT      LDA B        TAKE A COPY OF THE ADDRESS
0118  05116 031720           STA ADDRC       RESET THE ADDRESS COUNTER
0119  05117 000055          ADA MAX      CHECK IF OUT OF RANGE
0120  05120 070352           SAM *+7      SKIP IF WITHIN RANGE
0121  05121 120103          LDA EXCF,I   LOAD THE EXECUTE FLAG
0122  05122 070210           SZA *+4      SKIP IF IN CALCULATOR MODE
0123  05123 025701           LDB PCNT     LOAD THE PROF COUNTER
0124  05124 004023          ADB M5       BACK UP THE PROG COUNTER
0125  05125 035701          STB PCNT
0126  05126 066110           JMP EMST     LIGHT THE ERROR LIGHT AND STOP
0127  05127 035701          STB PCNT
0128  05130 124103           LDB EXCF,I    LOAD THE EXECUTION FLAG
0129  05131 076110           RZB *+2       SKIP IF EXECUTING A PROG
0130  05132 066122           JMP 41228    TERMINATE
0131  05133 035722           STB NUMF       RESET THE NUMBER FLAG
0132  05134 064167           JMP INIT       RETURN

0134*GOSUB PROCESSING
0135  05135 021710 GSUB     LDA USTK      LOAD THE U STACK POINTER
0136  05136 010112           CPA USTKL     UPPER LIMIT REACHED
0137  05137 066110           JMP EMST    LIGHT THE ERROR LIGHT   AND STOP
0138  05140 021701           LDA PCNT     LOAD VALUE OF PROG COUNT
0139  05141 131710           STA USTK,I   REMEMBER THE RETURN ADDR
0140  05142 045710           ISZ USTK     INCREMENT THE U STACK
0141  05143 067115           JMP SGT       PERFORM A SIMPLE GOTO
0142*
0143*
0144*GO TO LABEL ,AND LABEL KEY PROCESSING
0145*
0146*
0147  05144 023232 LABEL    LDA GINC1
0148  05145 011723           CPA RFLG      GOTO PRECEEDED?
0149  05146 067151           JMP *+3       YES
0150  05147 045701           ISZ PCNT      NO,SKIP THE NEXT KEY
0151  05150 067230           JMP SFLG+3   RE-INITIALIZE
0152  05151 060115           JSM GETK      GET THE NEXT KEY
0153  05152 024060           LDB MPWD      LOAD THE VALUE OF MAX MEMORY
0154  05153 035726           STB TEMP1   STORE IN TEMPORARY
0155  05154 025707           LDB KBAS    LOAD START OF USER AREA
0156  05155 035727           STB TEMP2   RESET A LOCAL PROG COUNT
0157  05156 024062           LDB LBLK    LOAD CODE FOR LBL KEY
0158*
0159*SEARCH LOOP
0160*
0161  05157 115727 SRCH     CPB TEMP2,I IS CURRENT KEY A LABEL?
0162  05160 067165           JMP LBLF      YES,STOP THE SEARCH
0163  05161 045727 SRCH1    ISZ TEMP2       POINT TO THE NEXT KEY
0164  05162 045726           ISZ TEMP1    MEMORY EXHAUSTED?
0165  05163 067157           JMP SRCH
0166  05164 066110           JMP EMST    LIGHT THE ERROR LIGHT A ND STOP
```

—Continued

```
                PROGRAMMABLE OPTION      PROP

*
*LABEL FOUND
*
 05165 055727 LBLF    DSZ TEMP2      POINT TO THE PRECEEDING KEY
 05166 125727         LDB TEMP2,I    LOAD THE PRECEEDING KEY
 05167 045727         ISZ TEMP2
 05170 045727         ISZ TEMP2     POINT TO THE LABEL'S NAME
 05171 014100         CPB GSBK      GOSUB/RET PRECEEDED?
 05172 067204         JMP GSBP      YES,MAY BE ALRIGHT
 05173 014061 LBLF2   CPB GTOK      GOTO PRECEEDED?
 05174 067156         JMP SRCH-1    YES,IGNORE LABEL KEY
 05175 125727 LBLF1   LDB TEMP2,I   LOAD THE LABEL'S NAME
 05176 045727         ISZ TEMP2
 05177 070077         CPB A        IS IT THE ONE SEARCHED FOR?
 05200 067202         JMP *+2      YES,TERMINATE
 05201 067156         JMP SRCH-1   RETURN TO THE SEARCH LOOP
 05202 025727         LDB TEMP2     LOAD VALUE OF LOCAL PROG COUNT
 05203 067113         JMP SGT-2     RETURN TO THE GOTO
*
 05204 025727 GSBP    LDB TEMP2     LOAD THE PROG POINTER
 05205 004077         ADB M3        POINT 2 KEYS BACK
 05206 074537         LDB B,I       LOAD KEY
 05207 067173         JMP LBLF2

                SUB/RET PROCESSING

*THE FOLLOWING CODE DEALS WITH THE
* SUB/RET KEY
*
*
 05210 027232 SUBR    LDB GINC1
 05211 015723         CPB RFLG        GOTO PRECEEDED?
 05212 067222         JMP GSB         YES,DO A GOSUB
 05213 021710 SUBRR   LDA USTK
 05214 010046         CPA USTKB       USER STACK EMPTY?
 05215 066110         JMP EMST    LIGHT THE ERROR LIGHT AND STOP
 05216 055710         DSZ USTK        DECREMENT THE STACK POINTER
 05217 121710         LDA USTK,I      LOAD THE RETURN ADDRESS
 05220 031701         STA PCNT        STORE IN THE PROG COUNTER
 05221 067230         JMP SFLG+3   RETURN
*
 05222 023235 GSB     LDA GSBA         GOSUB
 05223 031721         STA SUBF        SET GOSUB TRANSFER IN INDF
 05224 170402         RET             RETURN
*
*
*INTERPRETER TABLE LINKAGE
 00250                ORG 250B
 00250 005015         DEF IFXY       IF X=Y
 00252                ORG 252B
 00252 005015         DEF IFXY       IF X<Y
 00253                ORG 253B
 00253 005015         DEF IFXY       IF X>Y
 00254 005225         DEF SFLG       I SET FLAG
 05225                ORR

                IF, AND SET FLG PROCESSING

*
*SET FLAG
*
 05225 021776 SFLG    LDA PMODF      LOAD THE STATUS WORD
 05226 043236         IOR FLMS       OR WITH FLAG MASK
 05227 031776         STA PMODF
 05230 031720         STA 1720B    SET THE CHANGE SIGN FLAG
 05231 064167         JMP INIT      RE-INITIALIZE
*
*LINKAGE AREA
*
```

—Continued

```
                    IF, AND SET FLG PROCESSING
0234  05232 005102 GINC1  DEF GT02
0235  01723        RFLG   EQU 1723B
0236  01717        ATMP   EQU 1717B

0238  04176        MDEC   EQU 4176B
0239  04110        EMST   EQU 4110B  EMERGENCY STOP
0240  01707        KBAS   EQU 1707B
0241  01721        SUBF   EQU 1721B
0242  01710        USTK   EQU 1710B
0243  00112        USTKL  EQU 112B
0244  00046        USTKB  EQU 46B

0246  04750        ERROR  EQU 4750B
0247  04263        MINUS  EQU 4263B

0249  01701        PCNT   EQU 1701B
0250  00167        INIT   EQU 167B
0251  00031        M1     EQU 31B
0252  01777        SPTR   EQU 1777B
0253  00023        M5     EQU 23B
0254  00005        YADD   EQU 5B
0255  00014        FTMP   EQU 14B
0256  00004        XADD   EQU 4B
0257  01740        Y      EQU 1740B
0258  01734        FT     EQU 1734B
0259  00034        D3     EQU 34B
0260  00003        D4     EQU 3B
0261  05233 000002 .2     DEC 2
0262  00034        .3     EQU 34B
0263  00077        M3     EQU 77B
0264  00054        M4     EQU 54B
0265  01776        PMODE  EQU 1776B
0266  00100        GSHK   EQU 100B
0267  05234 000120 MSKS   OCT 120
0268  00105        S1     EQU 105B
0269  00063        .1     EQU 63B
0270  00015        AR1AD  EQU 15B
0271  00016        AR2AD  EQU 16B
0272  01744        AR1    EQU 1744B
0273  01754        AR2    EQU 1754B
0274  00060        MPWD   EQU 60B
0275  01725        KBUF   EQU 1725B     INPUT KEY BUFFER
0276  01726        TEMP1  EQU 1726B
0277  01727        TEMP2  EQU 1727B
0278  00061        GTOK   EQU 61B
0279  00062        LBLK   EQU 62B
0280  01720        ADDRC  EQU 1720B
0281  00115        GETK   EQU 115B
0282  05235 005134 GSBA   DEF GSUB-1
0283  05236 000200 FLMS   OCT 200
0284  00025        CLFM   EQU 25B
0285  00101        MASK   EQU 101B
0286  01775        IBUF   EQU 1775B
0287  00103        EXCF   EQU 103B
0288  00017        DISP   EQU 17B
0289  01722        NUMF   EQU 1722B
0290  01750        X      EQU 1750B
0291  00101        CLN    EQU 101B
0292  00055               ORG 55B
0293  00055 162000 MAX    OCT 162000


0295*INTERPRETER TABLE LINKAGE
0296  00243               ORG 243B
0297  00243 005062        DEF IFLG        IF FLAG
0298  00244 005070        DEF GOTO
0299  00277               ORG 277B
0300  00277 005210        DEF SUBR
0301  00251               ORG 251B
0302  00251 005144        DEF LABEL
0303  05237               ORR
0304                      END
*  NO ERRORS*
```

```
0001                ASMB,A,B,L,T
UP      004001                          CTMP    001720
ABETC   004011                          CLR1A   000052
RDLUP   004022                          CNGS    001720
AB1     004027                          RBAD1   000037
DOWN    004033                          RBAD    000033
NIND    004043                          INDF    001721
XINT    004050                          AR1AD   000015
XINT1   004051                          AR1     001744
XINT2   004062                          ATMP    001715
YINTO   004064                          OPER    001720
XFRM    004066                          XADD    000004
INTOY   004067                          RFLG    001723
YEXC    004071                          YADD    000005
ACTR1   004074                          TADD1   000013
ACTR3   004101                          ZADD    000006
ACTR4   004103                          INIT    000167
DREC    004115                          ADDRC   001716
ETRA    004120                          .1      000063
ETRA2   004130                          .3      000034
ETRA1   004133                          M1      000031
EXCH    004136                          M3      000077
COMPI   004147                          NUMF    001722
COMP    004166                          KBUF    001725
XEXY    004173                          ERROR   004750
MDEC    004176                          NLIM    000375
DPNT    000020                          INTR2   000357
BTMP    001717                          OPERH   000042
                                        RINC1   000040

                                        *  NO ERRORS*
          #01  TRANSFER BLOCK        TRB
0001                ASMB,A,B,L,T
0003*THIS PART PERFORMS ALL THE REGISTER TRANSFERS
0004*EVERY ROUTINE IS LABELLED ACCORDINGLY
0005*
0006*
0007*
0008* UP ROUTINE
0009  04000                 ORG 40008

0011  04000 170402          RET          TRAP WORD,RETURN

0013  04001 020005 UP       LDA YADD
0014  04002 024006          LDB ZADD
0015  04003 170004          XFR          TRANSFER Y TO Z
0016  04004 020004          LDA XADD
0017  04005 024005          LDB YADD
0018  04006 170004          XFR          TRANSFER X TO Y
0019  04007 035720          STB CNGS       RESET THE CHANGE SIGN FLAG
0020  04010 064167          JMP INIT       RE-INITIALIZE
0021*
0022*INTERPRETER TABLE ENTRY FOR UP
0023*
0024  04227                 ORG 2278
0025  04227 004001          DEF UP
0026  04011                 ORR            RESTORE THE ORIGIN
0027*


0029*EXECUTION OF REFERENCES TO A AND B
0030*
0031*
0032  04011 021725 ABETC    LDA KBUF       LOAD THE INPUT KEY
0033  04012 070704          SAL 2          MULTIPLY CODE BY 4
0034  04013 000037          ADA RBAD1    ADD TO BEGINING OF A&B AREA
0035  04014 024040          LDB RINC1      LOAD THE ROUITNG VALUE
0036  04015 015723          CPB RFLG       TRANSFER FUNCTION PRECEEDED?
0037  04016 066020          JMP *+2        YES
0038  04017 066027          JMP AB1        NO,TRANSFER A OR B INTO X
0039  04020 070137          LDB A        COPY A INTO B
0040  04021 066105          JMP ACTR4+2 PERFORM THE TRANSFER
```

—Continued

```
                TRANSFER BLOCK          TRB
0042*
0043*ROLL UP ROUTINE
0044  04022 020006 ROLUP  LDA ZADD
0045  04023 024015        LDB AR1AD
0046  04024 170004        XFR         TRANSFER Z TO A TEMPORARY
0047  04025 062001        JSM UP      TRANS. Y TO Z, AND X TO Y
0048  04026 020015        LDA AR1AD
0049  04027 024004 A81    LDB XADD
0050  04030 170004        XFR         TRANSFER Z TO X
0051  04031 035720        STB CNGS      RESET THE CHANGE SIGN FLAG
0052  04032 064167        JMP INIT      RE-INITIALIZE
0053*
0054*
0055*INTERPRETER TABLE ENTRY FOR ROLL UP
0056*
0057  00222               ORG 222B
0058  00222 004022        DEF ROLUP
0059  04033               ORR             RESTORE THE ORIGIN
0060*
0061*DOWN ROUTINE
0062  04033 020005 DOWN   LDA YADD
0063  04034 024004        LDB XADD
0064  04035 170004        XFR         TRANSFER Y TO X
0065  04036 020006        LDA ZADD
0066  04037 024005        LDB YADD
0067  04040 170004        XFR         TRANFER Z TO Y
0068  04041 035720        STB CNGS      RESET THE CHANGE SIGN FLAG
0069  04042 064167        JMP INIT      RE-INITIALIZE
0070*
0071*
0072*INTERPRETER TABLE ENTRY FOR DOWN
0073*
0074  00225               ORG 225B
0075  00225 004033        DEF DOWN
0076  04043               ORR             RESTORE THE ORIGIN
0077*
0078*INDIRECT KEY PROCESSING
0079*
0080  04043 021723 NIND   LDA RFLG      LOAD THE ROUTING REGISTER
0081  04044 010040        CPA RINC1     KEY FOLLOWED A TRANSFER KEY?
0082  04045 055721        OSZ INDF      DECREMENT THE INDIRECT FLAG
0083  04046 170402        RET
0084*
0085*INTERPRETER TABLE ENTRY FOR INDIRECT
0086  04047 004043        DEF NIND
0087*
0088  00231               ORG 231B
0089  00231 004043        DEF NIND
0090  04050               ORR             RESTORE THE ORIGIN
0091*
0092*
0093*
0094*****************************************
0095*
0096*
0097*
0098*THE FOLLOWING ROUTINE PERFORMS THE OPERATIONS
0099*NEEDED FOR X+(), X TO (), Y TO (), X EXCH Y,
0100*AND Y EXCH(). THE ROUTINE IS CONSTRUCTED OF
0101*THREE PHASES, THE FIRST PHASE ACCEPTS THE
0102*KEY CODE FOR THE COMMANDS>. THE SECOND PHASE
0103*ACCEPTS THE FIRST DIGIT IN THE REGISTER
0104*REFERENCE>. THE THIRD PHASE ACCEPTS THE 2ND
0105*DIGIT IN THE REGISTER REFERENCE,AND PERFORMS THE
0106*TRANSFER
0107*
0108* X TO (), PHASE1
0109  04050 020004 XINT   LDA XADD    STORE THE ADDRESS OF THE
0110  04051 031715 XINT1  STA ATMP    X REGISTER IN TEMPORARY
0111  04052 070742        SAR 16        INITIALIZE THE RESULT LOCATION
```

—Continued

```
                    TRANSFER BLOCK          TRB
0112  04053 031717          STA BTMP
0113  04054 020034          LDA .3          LOAD DECIMAL 3
0114  04055 031716          STA ADDRC       START REGISTER DIGIT COUNT
0115  04056 020031          LDA M1
0116  04057 031720          STA OPER        PUT -1 IN OPER
0117  04060 031721          STA INDF        INDICATE ZERO INDIRECT
0118  04061 024040          LDB RINC1       SET THE INCREMENT TO BE
0119  04062 035723 XINT2    STB RFLG        USED IN THE NUMBER RTN
0120  04063 170402          RET
0121*
0122*
0123* Y TO (), PHASE1
0124*
0125  04064 020005 YINTO    LDA YADD        STORE THE ADDRESS OF Y
0126  04065 066051          JMP XINT1       AND THE INCREMENT
0127*
0128*
0129* X←() , PHASE1
0130*
0131  04066 020004 XFRM     LDA XADD        STORE THE X ADDR, AND FLAG
0132  04067 072053 INTOY    SAP *+1,S       AS NEGATIVE FOR LATER USE
0133  04070 066051          JMP XINT1
0134*
0135*
0136* Y EXCH (), PHASE1
0137  04071 020005 YEXC     LDA YADD        STORE THE Y ADDR, AND FLAG
0138  04072 066067          JMP INTOY       AS NEGATIVE FOR LATER USE
0139*

0141*INTERPRETER TABLE ENTRY FOR TRANSFER BLOCK
0142*
0143  00223                 ORG 223B
0144  00223 004050          DEF XINT          X INTO ()
0145  00224 004071          DEF YEXC          Y EXCH ()

0147  00267                 ORG 267B
0148  00267 004066          DEF XFRM          X←()

0150  00240                 ORG 240B
0151  00240 004064          DEF YINTO         Y INTO ()
0152  04073                 ORR               RESET THE ORIGIN


0154*
0155* PHASE2 OF THE TRANSFER ROUTINE
0156*
0157  04073 066101          JMP ACTR3       TERMINATION ENTRY
0158  04074 021725 ACTR1    LDA KBUF        LOAD THE INPUT KEY
0159  04075 062176          JSM MDEC        CONVERT REFERENCE TO BUNARY
0160  04076 055716          DSZ ADDRC       DECREMENT THE ADDRESS COUNTER
0161  04077 170402          RET             RETURN IF NOT 3 DIGITS
0162  04100 045722          ISZ NUMF        TERMINATE DIGIT SEQUENCE
0163  04101 025717 ACTR3    LDB BTMP        LOAD THE LAST RESULT
0164  04102 074076          TCB             MAKE NEGATIVE
0165  04103 074704 ACTR4    SBL 2    MULTIPLY BY 4
0166  04104 004033          ADB RBAD        FIND THE ACTUAL ADDRESS
0167  04105 074117          LDA B       PUT THE ADDRESS IN A
0168  04106 000375          ADA NLIM    CHECK WHETHER OUT OF RANGE
0169  04107 070213          SAP *+4     SKIP IF OK
0170  04110 062750          JSM ERROR   LIGHT THE ERROR LIGHT
0171  04111 031722          STA NUMF    RESET THE NUMBER FLAG
0172  04112 066161          JMP 161B    STOP EXECUTION
0173  04113 045721          ISZ INDF        INDIRECT FLAG SET?
0174  04114 066147          JMP COMP1   NO, PROCESS AN INDIRECT
0175  04115 021715 DREC     LDA ATMP    DIRECT,LOAD THE SOURCE ADDRESS
0176  04116 045720          ISZ OPER        SKIP IF SIMPLE TRANSFER
0177  04117 165720          JMP OPER,I      PERFORM A TRANSFER FUCTION
0178*
0179*
0180*AT THIS POINT THE A REGISTER CONTAINS THE
0181*SOURCE ADDRESS,(X OR Y), AND THE B REGISTER
0182*CONTAINS THE ACTUAL DESTINATION
0183*
```

—Continued

```
                TRANSFER BLOCK          TRB
0184  04120 070432 ETRA    SAM ETRA2,C       SKIP IF YEXCH() OR X<>()
0185  04121 170004          XFR               PERFORM A TRANSFER
0186  04122 045722          ISZ NUMF        NUMBER FLAG SET?
0187  04123 064167          JMP INIT    RETURN
0188  04124 020052          LDA CLR1A     RESET THE ROUTING REGISTER
0189  04125 031723          STA RFLG      TO NUMBER ENTRY
0190  04126 025725          LDB KBUF      LOAD THE INPUT KEY
0191  04127 064357          JMP INTR2   PICK THE NEXT KEY
0192  04130 035717 ETRA2    STB BTMP        STORE DESTINATION INTO TEMP
0193  04131 010005          CPA YADD    SOURCE ADDRESS IS Y?
0194  04132 066136          JMP EXCH    YES, PERFORM A Y EXCH ()
0195  04133 070137 ETRA1    LDB A       NO, SWAP THE SOURCE AND DESTINATION
0196  04134 021717          LDA BTMP    ADDRESSES
0197  04135 066121          JMP ETRA+1
0198  04136 031720 EXCH     STA CTMP    STORE THE SOURCE ADDRESS
0199  04137 024013          LDB TADD1         PICK A FLOATING TEMPORARY
0200  04140 170004          XFR         INTO A DATA TEMPORARY
0201  04141 021717          LDA BTMP    LOAD DESTINATION INTO A
0202  04142 025720          LDB CTMP    TRANSFER THE DESTINATION'S CONTENT
0203  04143 170004          XFR         TO THE Y REGISTER
0204  04144 020013          LDA TADD1       LOAD ADDRESS OF TEMPORARY
0205  04145 025717          LDB BTMP      LOAD DESTINATION ADDRESS
0206  04146 066121          JMP ETRA+1    PUT TEMPORARY INTO DESTINATION
0207*
0208*INDIRECT ADDRESS PROCESSING
0209*
0210  04147 074117 COMPI    LDA B         LOAD THE REGISTER'S ADDRESS
0211  04150 024015          LDB AR1AD     TRANSFER ITS CONTENT
0212  04151 170004          XFR           TO AR1
0213  04152 074742          SBR 16        CLEAR B
0214  04153 035717          STB BTMP      CLEAR THE RESULT'S LOCATION
0215  04154 021744          LDA AR1       LOAD THE EXPONENT WORD
0216  04155 070113          SAP *+2       GO ON IF POSITIVE
0217  04156 066104          JMP ACTR4+1     ASSUME REFERENCE TO 000
0218  04157 070342          SAR 8         SHIFT DOWN THE EXPONENT
0219  04160 024063          LDB .1        PUT 1 IN B
0220  04161 070037          ADB A         ADD THE EXPONENT TO IT
0221  04162 000077          ADA M3        CHECK IF EXP > 2
0222  04163 070112          SAM *+2       SKIP IF NOT BIGGER
0223  04164 024034          LDB .3        RESTRICT ADDRESS TO 3 DIGITS
0224  04165 035716          STB ADDRC     STORE IN THE ADDRESS COUNTER
0225  04166 175400 COMP     DLS           PICK A DIGIT FROM THE REGISTER
0226  04167 062176          JSM MDEC      FIND THE BINARY EQUIVALENT
0227  04170 055716          DSZ ADDRC     ALL DIGITS IN?
0228  04171 066166          JMP COMP      NO,REPEAT
0229  04172 066102          JMP ACTR3+1 YES,COMPUTED ADDRESS IS IN B
0231*
0232* X EXCH Y
0233*
0234  04173 024004 XEXY     LDB XADD    STORE THE ADDRESS OF X
0235  04174 020005          LDA YADD    PUT THE Y ADDRESS INTO A
0236  04175 066130          JMP ETRA2     PERFORM AN EXCHANGE
0237*
0238*LINKAGE TABLE ENTRY FOR Y EXCH X
0239  00230                 ORG 2308
0240  00230 004173          DEF XEXY
0241  04176                 ORR
0242*
0243*
0244* DECIMAL TO BINARY CONVERTER
0245*
0246*THIS ROUTINE ACCEPTS A NUMBER IN A , AND
0247*A NUMBER IN B. THE ROUTINE MULTIPLIES THE
0248*NUMBER IN A BY 10 AND ADDS B TO THE RESULT
0249*
0250  04176 025717 MDEC     LDB BTMP    LOAD PREVIOUS RESULT
0251  04177 074704          SBL 2       4*B
0252  04200 005717          ADB BTMP    4*B+B
0253  04201 074037          ADB B         TEN TIMES B
0254  04202 070037          ADB A     (B)=10*B+A
0255  04203 035717          STB BTMP    STORE THE NEW RESULT
0256  04204 170402          RET
0257*
```

—Continued

```
                    TRANSFER BLOCK          TRB
0258*
0259*INTERPRETER TABLE ENTRY FOR A AND B KEYS
0260  00213                 ORG 213B
0261  00213 004011          DEF ABETC
0262  00214 004011          DEF ABETC
0263  04205                 ORR             RESTORE THE ORIGIN
0264*
0265*
0266  00020       DPNT      EQU 20B       OCTAL CODE FOR.        ROM
0267  01717       BTMP      EQU 1717B
0268  01720       CTMP      EQU 1720B
0269  00052       CLR1A     EQU 52B
0270  01720       CNGS      EQU 1720B      CHANGE SIGN FLAG      R/W.S
0271  00037       RBAD1     EQU 37B        BEGINING ADDR OF A     ROM
0272  00033       RBAD      EQU 33B       BASE ADDRESS OF USER REG ROM
0273  01721       INDF      EQU 1721B
0274  00015       AR1AD     EQU 15B       ADDRESS OF AR1
0275  01744       AR1       EQU 1744B     LOCATION OF AR1
0276  01715       ATMP      EQU 1715B      TEMPORARY STORAGE A   R/W.N *
0277  01720       OPER      EQU 1720B
0278  00004       XADD      EQU 4B         ADDRESS OF X           ROM
0279  01723       RFLG      EQU 1723B     SHIFT INDICATOR         R/W.N
0280  00005       YADD      EQU 5B      ADDRESS OF Y             ROM
0281  00013       TADD1     EQU 13B        ADDRESS OF DECIMAL TEMPORARY
0282  00006       ZADD      EQU 6B         ADDRESS OF THE Z REF    ROM

0284  00167       INIT      EQU 167B      INITIALIZATION ROUTINE

0286  01716       ADDRC     EQU 1716B   ADDRESS COUNTER        R/W.N
0287  00063       .1        EQU 63B      DECIMAL 1              ROM
0288  00034       .3        EQU 34B        DECIMAL 3
0289  00031       M1        EQU 31B      CONSTANT -1              ROM
0290  00077       M3        EQU 77B     DECIMAL -3             ROM
0291  01722       NUMF      EQU 1722B    NUMBER FLAG              R/W.N
0292  01725       KBUF      EQU 1725B    KEYCODE BUFFER          R/W.N
0293  24750       ERROR     EQU 4750B
0294  00375                 ORG 375B
0295  00375 177000 NLIM     OCT 177000


0297  00357       INTR2     EQU 357B    RECYCLING EMTRY TO INTERPRETER


0299  00042                 ORG 42B
0300  00042 004120 OPERH    DEF ETRA

0302*DEFINITION OF 2ND ENTRY IN TRANSFER BLOCK
0303*
0304  00040                 ORG 40B
0305  00040 004074 RINC1    DEF ACTR1
0306  04205                 ORR             RESTORE THE ORIGIN
0307                        END
*  NO ERRORS*


0001                ASMB,A,B,L,T

CHGS    000377                          NUMBT   000555
CHS1    000416                          NUMT2   000567
MAGN1   000426                          EEXA    000076
TWS     000442                          INTR2   000357
TWO     000444                          KBUF    001725
ONE     000450                          MMSK    000032
NONS    000451                          S1      000105
DPROC   000456                          RFLG    001723
DPR1    000465                          DPNT    000020
CLR     000466                          AR1     001744
CLR1    000471                          AR1AD   000015
CLR2    000501                          AR2AD   000016
EEXL    000510                          XADD    000004
EEXL1   000530                          INIT    000167
EEX     000532                          SGF     001721
EEX1    000543                          NUMF    001722
                                        OPTF    001717
```

—Continued

```
                TRANSFER BLOCK          TRB
CNGS    001720                                  NEXR    001715
INTA    000041                                  EXR     001716
BTMP    001725                                  NORM    000102
M1      000031                                  AR2     001754
.1      000063                                  PRTF    000107
.3      000034                                  PRTK    000110
X       001750                                  NEXE    000035
EXPC    001721                                  NUMBE   000036
EEXK    000074                                  CLR1A   000052
CNSK    000075                                  *  NO ERRORS*


                CHANGE SIGN ROUTINE     NMR
0001                    ASMB,A,B,L,T
0003  00377                     ORG 377B
0004*CHANGE SIGN
0005*
0006  00377 025723 CHGS         LDB RFLG        LOAD THE ROUTING REGISTER
0007  00400 020063              LDA .1        PUT 1 IN A
0008  00401 014076              CPB EEXA      IN ENTER EXPONENT LOOP?
0009  00402 064416              JMP CHS1      YES,CHANNEL TO THAT PHASE
0010  00403 025750              LDB X         LOAD THE EXPONENT OF X
0011  00404 110102              CPA NORM,I    DURING NUMBER ENTRY?
0012  00405 025744              LDB AR1         YES,LOAD EXPONENT OF AR1
0013  00406 076111              SLB *+2,S       SET MAGNITUDE SIGN IF CLEAR
0014  00407 074071              SLB *+1,C       CLEAR MAG SIGN IF SET
0015  00410 035750              STB X           STORE NEW EXP IN X
0016  00411 035744              STB AR1       STORE NEW EXPONENT IN AR1
0017  00412 031720              STA CNGS      SET THE CHANGE SIGN FLAG
0018  00413 110102              CPA NORM,I      DURING NUMBER ENTRY?
0019  00414 170402              RET
0020  00415 064167              JMP INIT         RE-INITIALIZE


0022*CHANGE SIGN DURING EXPONENT ENTRY
0023*
0024  00416 025744 CHS1         LDB AR1       LOAD EXPONENT OF AR1
0025  00417 005715              ADB NEXR      SUBTRACT THE RESIDUE
0026  00420 074076              TCB           COMPLIMENT THE EXPONENT
0027  00421 005716              ADB EXR       ADD BACK THE RESIDUE
0028  00422 055720              DSZ CNGS     EXPONENT NEGATIVE?
0029  00423 031720              STA CNGS        NO,MAKE IT NEGATIVE
0030  00424 064530              JMP EEXL1     TERMINATE CHANGE SIGN


0032*INTERPRETER TABLE LINKAGE
0033  00232                     ORG 232B
0034  00232 000377              DEF CHGS
0035  00425                     ORR

                NUMBER ENTRY ROUTINE     NMR


0038****** NUMBER ENTRY ROUTINE
0039* THIS ROUTINE BUILDS A FLOATING POINT NUMBER
0040*OUT OF INDIVIDUAL DIGIT ENTRIES
0041*THE ROUTINE ALSO TAKES CARE OF THE
0042*ENTER EXPONENT OPERATIONS
0043*
0044*
0045*
0046**CONTINUATOR ENTRY
0047  00425 064555              JMP NUMBT      JUMP TO TERMINATE ENTRY
0048  00426 021725 MAGN1         LDA KBUF      LOAD THE INPUT KEY
0049  00427 045721              ISZ SGF      SKIP IF THE SIG FLAG IS SET
0050  00430 064451              JMP NONS       PROCESS NON-SIGNIFICANT DIGIT
0051  00431 025744              LDB AR1        LOAD THE EXPONENT WORD
0052  00432 004105              ADB S1          INCREMENT THE EXPONENT
0053  00433 035744              STB AR1       STORE THE NEW EXPONENT
0054  00434 045717              ISZ DPTF      SKIP IF DECIMAL POINT IS IN
0055  00435 064442              JMP TWS
```

—Continued

```
                NUMBER ENTRY ROUTINE     NMR
0056  00436 055717          DSZ DPTF     SET THE DECIMAL POINT FLAG
0057  00437 004032          ADB MMSK      DECREMENT THE EXPONENT
0058  00440 035744          STB AR1       STORE THE NEW EXPONENT
0059  00441 072150          RZA TWO     SKIP IF CURRENT DIGIT IS NOT ZERO
0060  00442 024031 TWS      LDB M1      SET THE SIG DIG
0061  00443 035721          STB SGF       FLAG
0062  00444 175400 TWO      DLS           PUSH THE CURRENT DIGIT INTO AR1
0063  00445 070150          SZA **3     SKIP IF NO OVERFLOW
0064  00446 024063          LDB .1      PUT ONE IN B
0065  00447 174430          MRX         PUSH OUT THE NEW DIGIT
0066  00450 170402 ONE      RET       RETURN
0067  00451 071550 NONS     SZA TWO     SKIP IF DIGIT IS 0
0068  00452 045717          ISZ DPTF     SKIP IF DECIMAL POINT IS IN
0069  00453 064442          JMP TWS       SET THE SIGNIFICANT FLAG
0070  00454 055717          DSZ DPTF    SET THE DECIMAL POINT FLAG
0071  00455 064444          JMP TWO      PROCESS THE CURRENT DIGIT


0073*DECIMAL POINT PROCESSING
0074  00456 021717 DPROC    LDA DPTF     LOAD THE DECIMAL POINT FLAG
0075  00457 070312          SAM DPR1     SKIP IF ALREDY SET
0076  00460 020031          LDA M1       SET THE FLAG
0077  00461 031717          STA DPTF
0078  00462 031722          STA NUMF    SET THE NUMBER FLAG
0079  00463 045721          ISZ SGF     SIGNIFICANT FLAG SET?
0080  00464 031721          STA SGF      NO,SET IT
0081  00465 170402 DPR1     RET
0082**FIRST ENTRY IN NUMBER ROUTINE
0083*
0084  00466 024031 CLR      LDB M1
0085  00467 035722          STB NUMF    SET THE DIGIT SEQUENCE FLAG
0086  00470 165723          JMP RFLG,I    ROUTE ACCORDING TO RFLG
0087  00471 020015 CLR1     LDA AR1AD     LOAD ADDRESS OF AR1
0088  00472 170000          CLR
0089  00473 144102          ISZ NORM,I      INDICATE INPUT IS NOT NORMALIZED
0090  00474 055720          DSZ CNGS    SKIP IF CHANGE SIGN DID NOT PRECEED
0091  00475 064477          JMP **2      DO NOT CHANGE SIGN
0092  00476 045744          ISZ AR1      MAKE MAGNITUDE NEGATIVE
0093  00477 020041          LDA INTA        LOAD ADDRESS OF WORK FLAGS
0094  00500 170000          CLR           CLEAR THE WORK FLAGS
0095  00501 020035 CLR2     LDA NEXE     LOAD ADDRESS OF CONTINUATION ENTRY
0096  00502 031723          STA RFLG     STORE IN THE ROUTING REGISTER
0097  00503 021725          LDA KBUF     LOAD INPUT KEY
0098  00504 010020          CPA DPNT     DECIMAL POINT?
0099  00505 064456          JMP DPROC    YE,PROCESS DECIMAL POINT
0100  00506 064427          JMP MAGN1+1    NO,PROCESS MAGNITUDE
0101*
0102*ENTER EXPONENT PROCESSING
0103*
0104*

0106*ENTER EXPONENT LOOP
0107  00507 064555          JMP NUMBT     TERMINATION ENTRY
0108  00510 021725 EEXL     LDA KBUF      LOAD THHE INPUT KEY
0109  00511 055721          DSZ EXPC      DECREMENT THE EXPONENT DIGIT COUNTER
0110  00512 064514          JMP **2      PROCEED
0111  00513 064567          JMP NUMT2    2 EXP DIGITS ARE ALREADY IN
0112  00514 025720          LDB CNGS     LOAD THE CHANGE SIGN FLAG
0113  00515 014063          CPB .1      SKIP IF IT INDICATES POSIT EXPONENT
0114  00516 070076          TCA          COMPLIMENT THE INPUT DIGIT
0115  00517 070404          SAL 8       POSITION THE NEW INPUT DIGIT
0116  00520 025744          LDB AR1      LOAD THE EXPONENT WORD
0117  00521 005715          ADB NEXR     SUBTRACT THE RESIDUE
0118  00522 035725          STB BTMP    STORE THE OLD EXP
0119  00523 074704          SBL 2       4*EXP
0120  00524 005725          ADB BTMP    5*EXP
0121  00525 074037          ADB B       10*EXP
0122  00526 070037          ADB A       ADD IN THE NEW DIGIT
0123  00527 005716          ADB EXR      ADD THE EXPONENT RESIDUE BACK
0124  00530 035744 EEXL1    STB AR1     STORE THE NEW EXPONENT
0125  00531 170402          RET
```

—Continued

```
                NUMBER ENTRY ROUTINE     NMR
0127*FIRST ENTRY
0128  00532 020015 EEX     LDA AR1AD    NOT IN NUMBERENTRY
0129  00533 170000         CLR          CLEAR AR1
0130  00534 045747         ISZ AR1+3    PUT 1 IN AR1
0131  00535 055720         DSZ CNGS     SKIP IF CHANGE SIGN PRECEEDED
0132  00536 064540         JMP *+2
0133  00537 045744         ISZ AR1      MAKE MAGNITUDE NEGATIVE
0134  00540 144102         ISZ NORM,I     INDICATE NUMBER IS NOT NORMALIZED
0135  00541 020031         LDA M1
0136  00542 031722         STA NUMF      SET THE NUMBER FLAG
0137  00543 020076 EEX1    LDA EEXA     ENTRY DURING NUMBER BUILDING
0138  00544 031723         STA RFLG     SET ROUTING FLAG TO EEXL
0139  00545 020034         LDA .3
0140  00546 031721         STA EXPC     INIT THE EXP DIGIT COUNTER
0141  00547 031720         STA CNGS       FORCE EXPONENT SIGN +
0142  00550 021744         LDA AR1      LOADTHE EXPONENT
0143  00551 031716         STA EXR      STORE EXPONENT RESIDUE
0144  00552 070076         TCA          COMPLIMENT
0145  00553 031715         STA NEXR     STORE NEGATIVE OF ABOVE
0146  00554 170402         RET
0147*
0148*NUMBER ENTRY TERMINATION LOOP
0149*
0150  00555 045723 NUMBT    ISZ RFLG     CORRECT THE ROUTING REGISTER
0151  00556 021725          LDA KBUF     LOAD THE IINPUT KEY
0152  00557 010020          CPA DPNT     DECIMAL POINT?
0153  00560 064456          JMP DPROC    YES,DO NOT TERMINATE
0154  00561 010074          CPA EEXK     ENTER EXPONENT?
0155  00562 064543          JMP EEX1     YES, PROCEED TO ENTER EXP
0156  00563 010075          CPA CNSK     CHANGESIGN?
0157  00564 064377          JMP CHGS     YES,PROCESS CHANGE SIGN
0158  00565 010110          CPA PRTK     DOES PRINT FOLLOW?
0159  00566 144107          ISZ PRTF,I   SET PRINT ENTRY FLAG
0160*
0161  00567 020015 NUMT2    LDA AR1AD
0162  00570 024016          LDB AR2AD    NUMBER TO AR2
0163  00571 170004          XFR
0164  00572 020016          LDA AR2AD
0165  00573 171450          NRM          NORMALIZE
0166  00574 074742          SBR 16         CLEAR B
0167  00575 070135          SEC *+2,C    SKIP IF MAGNITUDE IS NOT ZERO
0168  00576 035754          STB AR2        CLEAR THE EXPONENT WORD
0169  00577 134102          STB NORM,I     CLEAR THE NON-NORMALIZED FLAG
0170  00600 035722          STB NUMF     RESET THE NUMBER FLAG
0171  00601 024004          LDB XADD     TRANSFER NORMALIZED NUMBER
0172  00602 170004          XFR          TO X
0173  00603 060167          JSM INIT     RE-INITIALIZE
0174  00604 025725          LDB KBUF     LOAD THE FOLLOWING KEY
0175  00605 064357          JMP INTR2    PICK THE NEXT KEY


0177*INTERPRETER LINKAGE ENTRY FOR ENTER EXPONENT
0178  00226                 ORG 226B
0179  00226 000532          DEF EEX      DEFINE ENTRY
0180  00606                 ORR          RESET THE ORIGIN
0181*
0182  00076                 ORG 76B
0183  00076 000510 EEXA     DEF EEXL
0184  00606                 ORR
0185*
0186*INTERPRETER LINKAGE FOR NUMBER
0187  00200                 ORG 200B
0188  00200 000466          DEF CLR
0189  00201 000466          DEF CLR
0190  00202 000466          DEF CLR
0191  00203 000466          DEF CLR
0192  00204 000466          DEF CLR
0193  00205 000466          DEF CLR
0194  00206 000466          DEF CLR
0195  00207 000466          DEF CLR
0196  00210 000466          DEF CLR
0197  00211 000466          DEF CLR
0198*
0199  00221                 ORG 221B
```

NUMBER ENTRY ROUTINE NMR—Continued

```
0200  00221 000471          DEF CLR1
0201  00606                 ORR            RESET THE ORIGIN
0202*
0203  01722                 ORG 1722B
0204  01722 000000          OCT 0
0205  00606                 ORR
0206*LINKAGE AREA
0207*

0209  00357          INTR2   EQU 357B     RECYCLING ENTRY TO THE INTERPRETER

0211  01725          KBUF    EQU 1725B
0212  00032          MMSK    EQU 32B       DECREMENT FOR EXPONENT
0213  00105          S1     EQU 105B
0214  01723          RFLG    EQU 1723B   INCREMENT INDICATOR     R/W,N
0215  00020          DPNT    EQU 20B      OCTAL CODE FOR .
0216  01744          AR1     EQU 1744B     LOCATION OF AR1
0217  00015          AR1AD   EQU 15B       ADDRESS OF AR1                ROM
0218  00016          AR2AD   EQU 16B
0219  00004          XADD    EQU 4B

0221  00167          INIT    EQU 167B      INITIALIZATION ENTRY

0223  01721          SGF     EQU 1721B   SIGNIFICANT FLAG    R/W,N
0224  01722          NUMF    EQU 1722B    NUMBER ENTRY FLAG     R/W,N
0225  01717          DPTF    EQU 1717B    DECIMAL POINT FLAG  R/W,N
0226  01720          CNGS    EQU 1720B
0227  00041          INTA    EQU 41B
0228  01725          BTMP    EQU 1725B     TEMPORARY LOACTION B          R/W,S
0229  00031          M1      EQU 31B       DECIMAL -1                    ROM
0230  00063          .1      EQU 63B
0231  00034          .3      EQU 34B      DECIMAL 3              ROM
0232  01750          X       EQU 1750B
0233  01721          EXPC    EQU 1721B    EXP DIGIT COUNTER      R/W,N
0234  00074          EEXK    EQU 74B      OCTAL CODE FOR ENT EXP   ROM
0235  00075          CNSK    EQU 75B      OCTAL CODE FOR CNG SGN   ROM
0236  01715          NEXR    EQU 1715B
0237  01716          EXR     EQU 1716B     EXPONENT RESIDUE              R/W,N
0238  00102          NORM    EQU 102B      NORMALIZED FLAG        R/W,N
0239  01754          AR2      EQU 1754B
0240  00107          PRTF    EQU 107B
0241  00110          PRTK    EQU 110B



0244  00035                  ORG 35B
0245  00035 000426 NEXE      DEF MAGN1
0246  00036 000466 NUMBE     DEF CLR      DEFINE 1ST ENTRY IN NUMBER


0248  00052                  ORG 52B
0249  00052 000471 CLR1A     DEF CLR1
0250  00606                  ORR

0252                         END
*  NO ERRORS*

0001                 ASMB,A,L,T,B
INTRP   000622
INBUF   001774
INTSR   001775
MSK     000043
STOP    000021
MAXC    000376
*  NO ERRORS*

0001                 ASMB,A,L,T,B
0002  00002                ORG 2B
0003  00002 060622         JSM INTRP
0004  00622                ORG 622B
0005  00622 031774 INTRP   STA INBUF    SAVES A IN INPUT BUFFER.
0006  00623 020021         LDA STOP    LOAD THE STOP CODE
```

NUMBER ENTRY ROUTINE NMR—Continued

```
0007  00624 172500        SFS 00      SKIP IF STOP IS INPUT
0008  00625 172241        LIA 01      LOAD A WITH INPUT KEY
0009  00626 050043        AND MSK       CLEAN UP KEYSTROKE.
0010  00627 031775        STA INTSR   STORE KEYSTROKE IN LOC. KNOWN TO MONITOR.
0011  00630 000376         ADA MAXC     SUBTRACT MAX KEYCODE FROM A
0012  00631 073713         SAP *-2,S    SKIP IF KEYCODE >111
0013  00632 021774        LDA INBUF   RESTORE A.
0014  00633 170402        RET
0015  01774          INBUF EQU 1774B
0016  01775          INTSR EQU 1775B
0017  00043          MSK   EQU 43B
0018  00021          STOP  EQU 21B       KEYCODE FOR STOP
0019  00376                ORG 376B
0020  00376 177666 MAXC    OCT 177666
0021  00003                ORR
0022                       END
*  NO ERRORS*

0001                ASMB,A,B,L,T
DTO     000172
CONT    000173
RCRD    000174
DFRM    000175
DNC     000176
UPC     000177
FMTE    000634
LOOK    000662
FOUND   000671
ENDS    000103
INCR    000105
PHTR    000673
PLT1    000703
PLT2    000705
PDAT    000712
PDAT0   000721
PDAT1   000724
PDAT2   000736
OUT     000737
OUTPT   000754
DONE    000766
SECD    000372
SECD1   000373
MSK     000374
SYNC    000105
GETK    000115
TBLE    000053
NINE    000057
XADD    000004
YADD    000005
AR2AD   000016
AR2     001754
.3      000034
FINT1   000161
FINT0   000155
.1      000063
M1      000031
USP     000046
AADD    000007
INM     000045
*  NO ERRORS*
```

BASIC FORMAT BLOCK

```
0001                ASMB,A,B,L,T
0003  00634                ORG 634B
0004  00172                ORG 172B
0005  00172 116006 DTO     DEF 160006B,I
0006  00173 116003 CONT    DEF 160003B,I
0007  00174 116004 RCRD    DEF 160004B,I
0008  00175 116005 DFRM    DEF 160005B,I
0009  00176 006000 DNC     OCT 6000
0010  00177 004000 UPC     OCT 4000
0011  00634                ORR      RESET THE ORIGIN
0012*
```

BASIC FORMAT BLOCK —Continued

```
0013*
0014  00634 060115 FMTE   JSM GETK       GET THE NEXT KEY
0015  00635 024053         LDB TBLE       LOAD THE START OF THE INTERPRETER TABEL
0016  00636 070037         ADB A
0017  00637 074537         LDB B,I        LOAD THE CORRESPONDING ENRTY
0018  00640 014246         CPB 246B      NEXT KEY IS END?
0019  00641 064155         JMP FINT0     YES, EXECUTE END
0020  00642 014244         CPB 244B      GOTO?
0021  00643 164173         JMP CONT,I     YES
0022  00644 014302         CPB 302B       RECORD?
0023  00645 164174         JMP RCRD,I     YES
0024  00646 014227         CPB 227B       UP KEY?
0025  00647 064703         JMP PLT1       JUMP TO PLOTTER ROUTINE
0026  00650 014225         CPB 225B       DOWN KEY?
0027  00651 064705         JMP PLT2       YES, JUMP TO THE PLOTTER ROUTINE
0028  00652 014223         CPB 223B     XTO KEY?
0029  00653 164172         JMP DTO,I    YES,TRANSFER DATA TO MAG CARD
0030  00654 014267         CPB 267B     X←() KEY?
0031  00655 164175         JMP DFRM,I   YES,PICK DATA FROM MAG CARD
0032  00656 014241         CPB 241B       STOP KEY?
0033  00657 064161         JMP FINT1      YES,RETURN TO CALC MODE
0034  00660 014247         CPB 247B      CONTINUE?
0035  00661 064673         JMP PHTR       YES,START THE PHOTO READER

0037*SEARCH ROUTINE
0038  00662 024176 LOOK   LDB DNC        LOAD START OF SEARCH AREA
0039  00663 014103         CPB ENDS     ALL BLOCKS TESTED?
0040  00664 170402         RET          YES,RETURN
0041  00665 074457         CPA B,I       DOES BLOCK START WITH SAME KEY?
0042  00666 064671         JMP FOUND     YES,THIS IS THE BLOCK SEARCHED FOR
0043  00667 004105         ADB INCR      NO,POINT TO THE NEXT BLOCK
0044  00670 064663         JMP LOOK+1    COMPARE KEY WORDS
0045  00671 074070 FOUND   SIB *+1       POINT TO 2ND WORD IN BLOCK
0046  00672 074737         JMP B,I       JUMP TO OPTIONAL BLOCK
0047  00103        ENDS    EQU 103B
0048  00105        INCR    EQU 105B

0050*PHOTOREADER START UP
0051  00673 020373 PHTR   LDA SECD1
0052  00674 173741         CLF 01       ENABLE THE INTERRUPT
0053  00675 025775         LDB 1775B         LOAD INPUT BUFFER. IF POS, IMPLIES
0054  00676 074112         SBM *+2           SINGLE STEPPING OR KEYBOARD, SUPPRESS
0055  00677 072051         SLA *+1,S         CONTINUE FROM PHOTOREADER.
0056  00700 172141         OTA 01       OUTPUT THE SELECT CODE
0057  00701 172601         STC 01
0058  00702 064161         JMP 161B     STOP THE EXECUTION IF ANY


                PLOTTER ROUTINES

0062  00703 020177 PLT1   LDA UPC      LOAD THE CODE FOR PEN UP
0063  00704 064706         JMP *+2
0064  00705 020176 PLT2   LDA DNC      LOAD THW CODE FOR PEN DOWN
0065  00706 074742         SBR 16       CLEAR B
0066  00707 035720         STB 1720B    INITIALIZE THE REG DELECT
0067  00710 035722         STB 1722B    INITIALIZE THE SYNC BIT
0068  00711 060754         JSM OUTPT     OUTPUT
0069  00712 024004 PDAT   LDA XADD
0070  00713 024016         LDB AR2AD    TRANSFER Y OR
0071  00714 170004         XFR              X TO AR2
0072  00715 070742         SAR 16
0073  00716 025754         LDB AR2        LOAD THE EXPONENT WORD
0074  00717 074251         SLB PDAT1      SKIP IF MAG IS POSITIVE
0075  00720 031755         STA AR2+1      ASSUME  0 COORDINATES
0076  00721 020177 PDAT0   LDA UPC       PULL THE PEN UP
0077  00722 060754         JSM OUTPT
0078  00723 064737         JMP OUT        OUTPUT THIS COORDINATE
0079  00724 074153 PDAT1   SBP *+3       SKIP IF EXP IS POSITIVE
0080  00725 031755         STA AR2+1     ASSUME 0000
0081  00726 064740         JMP OUT+1     OUTPUT 0000
0082  00727 074340         ABR 8          SHIFT THE EXPONENT DOWN
0083  00730 074076         TCB            COMPLIMENT THE EXPONENT
0084  00731 004034         ADB .3
0085  00732 074213         SBP PDAT2      SKIP IF EXP <4
0086  00733 020057         LDA NINE       STORE 9999 AS COORDINATE
```

PLOTTER ROUTINES —Continued

```
0087  00734 031755          STA AR2+1
0088  00735 064721          JMP PDAT0
0089  00736 174470 PDAT2    MRY              POSITION THE COORDINATE
0090  00737 021755 OUT      LDA AR2+1        LOAD THE 4 MOST SIG DIGITS
0091  00740 070342          SAR 8            PICK UP THE TOP TWO
0092  00741 060754          JSM OUTPT      OUTPUT THIS COORDINATE
0093  00742 024105          LDB SYNC
0094  00743 035722          STB 17228        MAKE THE SYNC BIT 1
0095  00744 021755          LDA AR2+1        LOAD THE 4 MOST SIG DIGITS AGAIN
0096  00745 050374          AND MSK          ISOLATE THE BOTTOM TWO
0097  00746 060754          JSM OUTPT        OUTPUT THIS COORDINATE
0098  00747 021720          LDA 17208        LOAD THE REG SELECT FLAG
0099  00750 072710          RZA DONE         SKIP IF INDICATES DONE
0100  00751 020005          LDA YADD         LOAD THE ADDRESS OF Y
0101  00752 031720          STA 17208        SET THE REG SELECT FLAG TO NON-ZERO
0102  00753 064713          JMP PDAT+1     GET OOUT THE NEXT Y COORDINATE
0103*
0104*
0105  00754 040372 OUTPT    IOR SECD      PUT IN THE SELECT CODE
0106  00755 172141          OTA 01       OUTPUT THE SELECT CODE
0107  00756 172741          STF 01      PUT STATUS ON I/O REG
0108  00757 176241          LIB 01       INPUT THE STATUS
0109  00760 074504          SBL 6        POSITION THE STANDBY BIT
0110  00761 074253          SBP DONE     SKIP IF PLOTTER IS NOT AVAILABLE
0111  00762 041722          IOR 17228      PUT IN THE SYNC BIT
0112  00763 172141          OTA 01        OUTPUT THE COMMAND
0113  00764 172601          STC 01        SET THE CONTROL
0114  00765 172701          SFC 01         SKIP IF DEVICE ISS NOT DONE
0115  00766 170402 DONE     RET              RETURN
0116  00767 064765          JMP *-2      WAIT
0117*

0119*
0120*LINKAGE AREA
0121  00372                 ORG 372B
0122  00372 160000 SECD     OCT 160000     SELECT CODE
0123  00373 170000 SECD1    OCT 170000
0124  00374 000377 MSK      OCT 377       LOW 8 BIT NASK
0125  00105         SYNC    EQU 105B      SYNC BIT NON -ZERO  400 OCTAL
0126  00115         GETK    EQU 115B
0127  00053         TBLE    EQU 53B
0128  00057                 ORG 57B
0129  00057 114631  NINE    OCT 114631
0130  00004         XADD    EQU 4B
0131  00005         YADD    EQU 5B
0132  00016         AR2AD   EQU 16B
0133  01754         AR2     EQU 1754B
0134  00034         .3      EQU 34B
0135  00161         FINT1   EQU 161B
0136  00155         FINT0   EQU 155B
0137  00063         .1      EQU 63B
0138  00031         M1      EQU 31B
0139  00046         USP     EQU 46B
0140  00007         AADD    EQU 7B
0141  00045         INM     EQU 45B
0142*LINKAGE IN INTERPRETER TABLE
0143  00242                 ORG 242B
0144  00242 000634          DEF FMTE
0145  00770                 ORR
0146                        END
*  NO ERRORS*
```

—Continued

```
0001                ASMB,A,B,L,T
CHECK   005274
SKP1    005313
YZERT   005314
UNNRM   005317
FLTCK   005333
DSET    005340
DSET2   005344
FIXM    005354
```

```
60 TEST    005367
   INCM    005403
   ZOUT    005405
   SINOT   005411
   CKFLT   005424
65 ZERT    005426
   DIGOT   005434
   PCHK    005436
   D1DEC   005441
```

—Continued

```
FROUT   005456
QUIT    005475
QUIT1   005507
RESTR   005514
CONTU   005522
OUTPT   005543
BASE    005551
ONE     000063
M1      000031
M15     005567
M13     005570
TWLVE   000106
PTR     005571
AR2AD   000016
AR1AD   000015
AR2     001754
ELEVN   005572
M4      000054
M12     005573
XADD    000004
FLOAT   001776
FIVE    005574
D1      001737
DEC     005575
EXPNT   001754
CPOS    001735
P       005576
IPTR    005577
TWO     005600
AR21    001755
THREE   000034
BLANK   005572
DOT     000106
TEN     000104
M3      000077
NORM    001723
M10     000024
MINUS   000104
ALLON   005601
FIFTN   005601
SSP     001777
ENTER   005602
```

—Continued

```
CATCH   005626
DPASS   005617
CLRM    005623
RSTRT   005625
FETCH   005627
GO1     005644
GO      005650
CHECM   005653
REGPT   000104
POSCT   001754
NPOS    001755
DREG    001756
DPTR    001757
OTHRP   005661
BUFFR   001775
FLGMS   005601
AR2ND   001757
AR1ND   001747
FOUR    000052
REG     001734
POSNR   001736
MASK    005601
THFV    000035
M5      000023
PRGM    005665
NEWPT   005713
SUBEM   005717
OUT     005734
BOUT    005736
DECB    005752
MAXP    005757
ABLNK   005760
PC      001701
PCTMP   001737
DECMT   001735
DECPT   001754
M8      005761
PAUSE   005762
CONST   005601
MAX     000055
CERR    005776
BITMK   000770
*  NO ERRORS*
```

```
0001                    ASMB,A,B,L,T
0002  0017              ORG 17B         TELLS MONITOR WHERE DISPLAY IS.
0003  0017 005602       DEF ENTER       ENTRY PT FOR DISPLAY.
0004  0257              ORG 257B
0005  0257 005762       DEF PAUSE
0006  5274              ORG 5274B
0007  5274 020003 CHECK LDA 3B         LOAD THE POINTER TO THE REG ADDRESSES
0008  5275 133571       STA PTR,I     REGISTER ADDRESSING.
0009  5276 120102       LDA 102B,I    LOAD THE NORM FLAG
0010  5277 070610       SZA SKP1      SKIP IF X NORM.
0011  5300 020015       LDA AR1AD     TRANSFER AR1 TO
0012  5301 024004       LDB XADD      X
0013  5302 170004       XFR
0014  5303 123577       LDA IPTR,I     DOUBLE INDIRECT.
0015  5304 024016       LDB AR2AD
0016  5305 170004       XFR
0017  5306 171450       NRM
0018  5307 074076       TCB
0019  5310 004106       ADB TWLVE
0020  5311 035737       STB D1
0021  5312 067317       JMP UNNRM
0022  5313 123577 SKP1  LDA IPTR,I     LOAD ADDRESS OF CURRENT REGISTER.
0023  5314 024016 YZENT LDB AR2AD     TRANSFER THE CURRENT REG
0024  5315 170004       XFR           TO AR2
0025  5316 035737       STB D1        SET D1 TO ANY LARGE NUMBER.
0026  5317 024015 UNNRM LDB AR1AD     INIT THE REGISTER PTR.
0027  5320 020015       LDA AR1AD     LOAD THE ADDRESS OF AR1
```

—Continued

```
0028  05321 170000        CLR         INSURE NO NON BCD CODES FOR ROUNDING.
0029  05322 035734        STB REG
0030  05323 023601        LDA FIFTN     INITIALIZE THE DISPLAY
0031  05324 133576        STA P,I      CHARACTER COUNTER.
0032  05325 020023        LDA M5        INITIALIZE THE POSITION COUNTER
0033  05326 031736        STA POSNR
0034  05327 021776        LDA FLOAT     LOAD THE STATUS WORD
0035  05330 050027        AND 27B       ISOLATE THE MAIN MODE
0036  05331 072110        RZA *+2       SKIP IF IN RUN
0037  05332 067665        JMP PRGM      JUMP TO PROG DISPLAY
0038  05333 025754 FLTCK  LDB AR2       LOAD THE EXPONENT WORD
0039  05334 074340        ABR 8         SHIFT DOWN
0040  05335 035754        STB EXPNT     STORE FOR NOW
0041  05336 021776        LDA FLOAT
0042  05337 070253        SAP DSET2       SKIP IF IN FLOATING PT.
0043  05340 020106 DSET   LDA TWLVE
0044  05341 133575        STA DEC,I
0045  05342 027572        LDB ELEVN
0046  05343 067354        JMP FIXM
0047  05344 053601 DSET2  AND ALLON
0048  05345 133575        STA DEC,I
0049  05346 107575        ADB DEC,I
0050  05347 007600        ADB TWO         B←EXP+DEC+2.
0051  05350 023573        LDA M12
0052  05351 074017        ADA B
0053  05352 035735        STB CPOS
0054  05353 071253        SAP DSET      SKIP FI NO OVERFLOW.
0055  05354 023574 FIXM   LDA FIVE       ROUNDING COUNT.
0056  05355 174430        MRX


0057  05356 170560        FXA
0058  05357 070415        SEC TEST       OVERFLOW?
0059  05360 045754        ISZ EXPNT     INCREMENT THE EXPONENT
0060  05361 067362        JMP *+1       DUMMY JUMP FOR -1 EXPNT
0061  05362 020063        LDA ONE
0062  05363 024063        LDB ONE
0063  05364 174470        MRY
0064  05365 045735        ISZ CPOS
0065  05366 067367        JMP *+1
0066  05367 123575 TEST   LDA DEC,I
0067  05370 010106        CPA TWLVE       BYPASS BLANKS IF IN FLOATING PT.
0068  05371 067411        JMP SINOT
0069  05372 023567        LDA M15
0070  05373 001735        ADA CPOS          A←-(15-CPOS)
0071  05374 127575        LDB DEC,I
0072  05375 007570        ADB M13
0073  05376 074076        TCB               B←(13-DEC)
0074  05377 070037        ADB A           B←(13-DEC)-(15-CPOS)
0075  05400 074153        SBP *+3       (15-CPOS)>(13-DEC). OUTPUT (13-DEC)-1 BLMKS
0076  05401 074076        TCB         B←(15-CPOS)-(13-DEC)
0077  05402 074017        ADA B       A HAS -(13-DEC)
0078  05403 070070 INCM   SIA *+1       (13-DEC)>=(15-CPOS). OUTPUT(15-CPOS)-1 BLKS
0079  05404 031735        STA CPOS
0080  05405 023572 ZOUT   LDA BLANK
0081  05406 063543        JSM OUTPT
0082  05407 045735        ISZ CPOS
0083  05410 067405        JMP ZOUT
0084  05411 127577 SINOT  LDB IPTR,I
0085  05412 074537        LDB B,I
0086  05413 023572        LDA BLANK
0087  05414 074111        SLB *+2
0088  05415 020104        LDA MINUS
0089  05416 063543        JSM OUTPT
0090  05417 021737        LDA D1
0091  05420 072110        RZA *+2
0092  05421 045737        ISZ D1
0093  05422 021754        LDA EXPNT
0094  05423 070453        SAP DIGOT
0095  05424 021776 CKFLT  LDA FLOAT
```

—Continued

```
0096  05425 070352        SAM DIGOT
0097  05426 070742 ZERT   SAR 16
0098  05427 063543        JSM OUTPT
0099  05430 045754        ISZ EXPNT
0100  05431 067426        JMP ZERT
0101  05432 045737        ISZ D1
0102  05433 067441        JMP D1DEC       TO DSZ D1.
0103  05434 171400 DIGOT  MLS
0104  05435 063543        JSM OUTPT
0105  05436 123576 PCHK   LDA P,I
0106  05437 010034        CPA THREE
0107  05440 067456        JMP FROUT       TO FLOAT PT ROUTINE.
0108  05441 055737 D1DEC  DSZ D1
0109  05442 067434        JMP DIGOT
0110  05443 123575        LDA DEC,I       IF DEC NOT OUT, PUT OUT A ZERO.
0111  05444 000063        ADA ONE
0112  05445 070056        CMA


0113  05446 103576        ADA P,I
0114  05447 070152        SAM *+3
0115  05450 045737        ISZ D1
0116  05451 067434        JMP DIGOT
0117  05452 023572        LDA BLANK       OUTPUT ZEROS ON NUMB ENTRY.
0118  05453 063543        JSM OUTPT
0119  05454 045737        ISZ D1          KEEP D1 AT ZERO WHILE OUTPUT BLANKS.
0120  05455 067436        JMP PCHK
0121  05456 123575 FROUT  LDA DEC,I
0122  05457 003573        ADA M12         ZERO IF OVERFLOW OR FT. PT.
0123  05460 073050        RZA D1DEC
0124  05461 025754        LDB EXPNT
0125  05462 023572        LDA BLANK
0126  05463 074113        SBP *+2
0127  05464 020104        LDA MINUS
0128  05465 063543        JSM OUTPT
0129  05466 021754        LDA EXPNT
0130  05467 070113        SAP *+2
0131  05470 070076        TCA
0132  05471 027752        LDB DECB
0133  05472 007600        ADB TWO
0134  05473 035754        STB DECPT
0135  05474 067713        JMP NEWPT
0136  05475 021747 QUIT   LDA AR1ND       EVEN DISPLAY FILE.
0137  05476 070604        SAL 4
0138  05477 043572        IOR BLANK
0139  05500 031747        STA AR1ND
0140  05501 024104        LDB REGPT
0141  05502 107571        ADB PTR,I
0142  05503 004454        ADB M4
0143  05504 074537        LDB B,I
0144  05505 020015        LDA AR1AD
0145  05506 170004        XFR
0146  05507 147571 QUIT1  ISZ PTR,I
0147  05510 123577        LDA IPTR,I      LD THE ADDRESS OF NEXT REGISTER.
0148  05511 010007        CPA 7B          CHECK IF DONE
0149  05512 067514        JMP RESTR       YES,TREMINATE
0150  05513 067314        JMP YZENT       NO,CONSIDER THE NEXT REGISTER
0151  05514 120102 RESTR  LDA 102B,I
0152  05515 070210        SZA *+4
0153  05516 020004        LDA XADD
0154  05517 024015        LDB AR1AD
0155  05520 170004        XFR
0156  05521 170402        RET
0157  05522 125734 CONTU  LDB REG,I
0158  05523 074604        SBL 4
0159  05524 074217        IOR B
0160  05525 131734        STA REG,I
0161  05526 021776        LDA FLOAT
0162  05527 050027        AND 27B
0163  05530 070250        SZA *+5         SKIP IF IN PROG MODE
0164  05531 123575        LDA DEC,I       CHECK DECIMAL POSITION.
```

—Continued

```
0165  05532 033600          ADA TWO
0166  05533 113576          CPA P,I
0167  05534 067541          JMP *+5
0168  05535 157576          DSZ P,I



0169  05536 170402          RET
0170  05537 055777          DSZ SSP        DONE, DECREMENT THE STACK POINTER
0171  05540 067475          JMP QUIT
0172  05541 157576          DSZ P,I
0173  05542 020100          LDA DOT
0174  05543 045736 OUTPT    ISZ POSNR
0175  05544 067522          JMP CONTU
0176  05545 024054          LDB M4
0177  05546 035736          STB POSNR
0178  05547 045734          ISZ REG
0179  05550 067522          JMP CONTU
0180  05551 005552 BASE     DEF *+1
0181  05552 073400          OCT 73400      0
0182  05553 010400          OCT 10400      1
0183  05554 037000          OCT 37000      2
0184  05555 035400          OCT 35400      3
0185  05556 054400          OCT 54400      4
0186  05557 065400          OCT 65400      5
0187  05560 067400          OCT 67400      6
0188  05561 030400          OCT 30400      7
0189  05562 077400          OCT 77400      8
0190  05563 075400          OCT 75400      9
0191  05564 004000          OCT 4000       MINUS
0192  05565 000000          OCT 0          BLANK
0193  05566 000200          OCT 200        DECIMAL
0194  00063       ONE       EQU 63B
0195  00031       M1        EQU 31B
0196  05567 177761 M15       DEC -15
0197  05570 177763 M13       DEC -13
0198  00106       TWLVE     EQU 106B
0199  05571 012005 PTR       OCT 12005
0200  00016       AR2AD     EQU 16B
0201  00015       AR1AD     EQU 15B
0202  01754       AR2       EQU 1754B
0203  05572 000013 ELEVN     DEC 11
0204  00054       M4        EQU 54B
0205  05573 177764 M12       DEC -12
0206  00004       XADD      EQU 4B
0207  01776       FLOAT     EQU 1776B
0208  05574 000005 FIVE DEC 5
0209  01737       D1        EQU 1737B
0210  05575 012004 DEC       OCT 12004
0211  01754       EXPNT     EQU 1754B
0212  01735       CPOS      EQU 1735B
0213  05576 012003 P         OCT 12003
0214  05577 112005 IPTR      OCT 112005
0215  05600 000002 TWO       OCT 2
0216  01755       AR21      EQU 1755B
0217  00034       THREE     EQU 34B
0218  05572       BLANK     EQU ELEVN
0219  00106       DOT       EQU TWLVE
0220  00104       TEN       EQU 104B
0221  00077       M3        EQU 77B
0222  01723       NORM      EQU 1723B
0223  00024       M10       EQU 24B
0224  00104       MINUS     EQU TEN



0225  05601 000017 ALLON     DEC 15
0226  05601       FIFTN     EQU ALLON
0227  01777       SSP       EQU 1777B
0228*
0229  05602 076453 ENTER     SPP *+1,S
0230  05603 035775           STB 1775B
```

—Continued

```
0231  05604 173741          CLF 01
0232  05605 063274          JSM CHECK
0233  05606 021775 CATCH LDA BUFFR        CHECK BUFFER TO SPEED UP MARKED CARD
0234  05607 070153          SAP *+3         PROCESSING.
0235  05610 063617          JSM DPASS
0236  05611 067606          JMP CATCH
0237  05612 021776          LDA FLOAT     CLEAR ERROR CONDITION.
0238  05613 053776          AND CERR
0239  05614 031776          STA FLOAT     STORE ERRORLESS STATUS WORD.
0240  05615 021775          LDA BUFFR
0241  05616 170402          RET
0242  05617 023661 DPASS LDA OTHRP     ENTRY POINT DOES NOT CAAL DISPLAY BUILDER
0243  05620 031757          STA DPTR
0244  05621 020012          LDA 12B
0245  05622 031756          STA DREG
0246  05623 020054 CLRM   LDA M4
0247  05624 031755          STA NPOS
0248  05625 070742 RSTRT SAR 16
0249  05626 031754          STA POSCT
0250  05627 121756 FETCH LDA DREG,I
0251  05630 070546          RAR 12
0252  05631 131756          STA DREG,I
0253  05632 053601          AND ALLON
0254  05633 000551          ADA BASE
0255  05634 070517          LDA A,I
0256  05635 141757          IOR DPTR,I
0257  05636 041754          IOR POSCT
0258  05637 045755          ISZ NPOS
0259  05640 067644          JMP GO1
0260  05641 045756          ISZ DREG
0261  05642 024054          LDB M4
0262  05643 035755          STB NPOS
0263  05644 025754 GO1    LDB POSCT        THIS IS A FILLER TO EVEN DISP. TIMES.
0264  05645 017601          CPB ALLON
0265  05646 067653          JMP CHECM
0266  05647 173750          CLF 8          DISABLE DISPLAY.
0267  05650 172150 GO     OTA 8          OUTPUT DATA TO DISPLAY.
0268  05651 045754          ISZ POSCT
0269  05652 067627          JMP FETCH
0270  05653 173750 CHECM CLF 8          KEEP DISPLAY TIMES REL. CONSTANT.
0271  05654 045757          ISZ DPTR
0272  05655 125757          LDB DPTR,I
0273  05656 017665          CPB OTHRP+4
0274  05657 170402          RET              TERMINATE DISPLAY MANIP.
0275  05660 067625          JMP RSTRT
0276  00104          REGPT EQU TEN     USED FOR DISPLAY FILE ADDRESSING.
0277  01754          POSCT EQU 1754B
0278  01755          NPOS  EQU 1755B
0279  01756          DREG  EQU 1756B
0280  01757          DPTR  EQU 1757B


0281  05661 005662 OTHRP DEF *+1
0282  05662 000140          OCT 140        X REG SELECT.
0283  05663 000120          OCT 120
0284  05664 000060          OCT 60         Z REG SELECT.
0285  01775          BUFFR EQU 1775B
0286  05601          FLGMS EQU ALLON     MASK IS 15 FOR FIX/FLOAT.
0287  01757          AR2ND EQU 1757B
0288  01747          AR1ND EQU 1747B
0289  00052          FOUR  EQU 52B
0290  01734          REG   EQU 1734B
0291  01736          POSNR EQU 1736B
0292  05601          MASK  EQU ALLON
0293  00035          THFV  EQU 35B
0294  00023          M5    EQU 23B
0295  05665 023752 PRGM   LDA DECB
0296  05666 031754          STA DECPT        SHOWS WHAT POWER OF TEN TO USE.
0297  05667 025701          LDB PC        LOAD THE PROG COUNTER
0298  05670 004055          ADB MAX       CHECK FOR OVERFLOW
0299  05671 074152          SBM *+3       SKIP IF IT DID NOT OVERFLOW
0300  05672 027757          LDB MAXP      LOAD VALUE OF MAX PERMIS PC
```

—Continued

```
0301  05673 035701        STB PC        STORE IT
0302  05674 123571        LDA PTR,I
0303  05675 000023        ADA M5         TRICKY FORMULA TO COMPUTE WHAT
0304  05676 070076        TCA               VALUE OF PC TO USE.
0305  05677 025707        LDR 1707B      CONVERT ABS PC TO RELATIVE.
0306  05700 074076        TCB
0307  05701 001701        ADA PC
0308  05702 031737        STA PCTMP        TEMP PC.
0309  05703 074017        ADA B
0310  05704 070353        SAP NEWPT
0311  05705 023760        LDA ABLNK
0312  05706 031734        STA 1734B
0313  05707 031735        STA 1735B
0314  05710 031736        STA 1736B
0315  05711 031737        STA 1737B
0316  05712 067514        JMP RESTR
0317  05713 125754 NEWPT  LDB DECPT,I
0318  05714 035735        STB DECMT
0319  05715 074753        SBP OUT
0320  05716 074742        SBR 16          SKIP TO OUT IF LOOP IS ENDED.
0321  05717 001735 SUBEM  ADA DECMT
0322  05720 070152        SAM *+3
0323  05721 074070        STB *+1
0324  05722 067717        JMP SUBEM
0325  05723 031735        STA DECMT
0326  05724 074117        LDA B
0327  05725 063543        JSM OUTPT
0328  05726 021735        LDA DECMT
0329  05727 070076        TCA
0330  05730 101754        ADA DECPT,I
0331  05731 070076        TCA
0332  05732 045754        ISZ DECPT
0333  05733 067713        JMP NEWPT
0334  05734 027761 OUT    LDB M8
0335  05735 035735        STB DECMT
0336  05736 020104 BOUT   LDA MINUS


0337  05737 063543        JSM OUTPT
0338  05740 045735        ISZ DECMT
0339  05741 067736        JMP BOUT
0340  05742 121737        LDA PCTMP,I      HAS OUTPUTTED ALL BLANKS.
0341  05743 070102        SAR 3             NOW WORK ON OCTAL VALUES.
0342  05744 063543        JSM OUTPT
0343  05745 121737        LDA PCTMP,I
0344  05746 050770        AND BITMK
0345  05747 063543        JSM OUTPT
0346  05750 023572        LDA BLANK
0347  05751 063543        JSM OUTPT       NEATEN DESPLAY.
0348  05752 005753 DECB   DEF *+1
0349  05753 176030        DEC -1000       4 DIGIT ADDRESSING.
0350  05754 177634        DEC -100
0351  05755 177766        DEC -10
0352  05756 177777        DEC -1
0353  05757 015777 MAXP   OCT 15777
0354  05760 135673 ABLNK  OCT 135673
0355  01701        PC     EQU 1701B
0356  01737        PCTMP  EQU D1
0357  01735        DECMT  EQU CPOS
0358  01754        DECPT  EQU EXPNT
0359  05761 177770 M8     DEC -8
0360  05762 173741 PAUSE  CLF 01      ENABLE INTERRUPT.
0361  05763 063274        JSM CHECK     BUILD DISPLAY LIST.
0362  05764 021775        LDA 1775B     LOAD INPUT BUFFER.
0363  05765 070112        SAM *+2
0364  05766 064161        JMP 161B        JMP STOP ROUTINE.
0365  05767 023601        LDA CONST
0366  05770 031722        STA 1722B     TEMP LOC.
0367  05771 063617        JSM DPASS
0368  05772 055722        DSZ 1722B
0369  05773 067771        JMP *-2
0370  05774 173741        STF 01        TURN OFF INTERRUPT.
```

—Continued

```
0371   05775 170402          RET
0372   05601          CONST  EQU FIFTN
0373   00055          MAX    EQU 55B
0374   05776 167777 CERR     OCT 167777      ERROR MASKING.
0375   00770                 ORG 770B
0376   00770 000007 BITMK    OCT 7
0377                         END
*  NO ERRORS*
```

```
0001                  ASMB,A,L,T,B
CLEAN   016374
BLNKM   016375
THRTN   016376
PRINT   016377
PRINN   016401
KILEM   016403
KILWD   016405
MAJFC   016407
GETIT   016447
SERCH   000662
HERE    016451
RETUR   016470
PRTEM   016472
SPACE   016501
SPC1    016502
ICHCK   016511
OUTK    016516
OK      016525
HOLD    001754
STOP    000161
OUTWD   001737
GROUP   016532
POS     016533
CHAR    001736
ADDRS   001735
TEMP    001734
I       016534
BUF     016535
OCT42   016536
YDIS    000013
MSK     016537
ROTWD   016540
SIX     016541
TBL     016542
CODBL   016552
```

—Continued

```
MANPT   016616
NORML   016627
NMBR    016633
FCH     016635
SET     016646
PRNT    000110
XDIS    000012
INTR2   000357
INIT    000167
XLSB    001727
LISTM   016650
CKM1    016656
STRT1   016660
EOUT    016667
SPICE   016670
ENDM    016002
KEYLG   016674
WORK    016712
FRCM    016726
FRCM1   016731
```

```
MSKA    000770
ROT     016733
UNROT   016736
PUT     016740
KEYCG   016743
THREE   000034
M3      000077
M1      000031
M10     000024
FLOAT   001776
PC      001701
ABLNK   016751
CHECK   016752
SECUR   000111
*  NO ERRORS*
```

```
0001                  ASMB,A,L,T,B
0002*       BASIC PRINT ROUTINE.
0003  00113                 ORG 113B
0004  00113 016674          DEF KEYLG       KEYLOG ENTRY.
0005  00304                 ORG 304B
0006  00304 016650          DEF LISTM       LIST ENTRY.
0007  00305 016743          DEF KEYCG        BIT DIDDLER ENTRY FOR DEYLOG.
0008  00245                 ORG 245B
0009  00245 016616          DEF MANPT        PRINTER ENTRY.
0010  10007                 ORG 10007B
0011  10007 016447          DEF GETIT               FOR ALPHA PRINT ROUTINES.
0012  10023                 ORG 10023B
0013  10023 016633          DEF NMBR                FOR ALPHA PRINT ROUTINES.
0014  16374                 ORG 16374B
0015  16374 177760 CLEAN    OCT 177760        MOVE THESE TO MAG CARD AREA.
0016  16375 000273 BLNKM    OCT 273
0017  16376 000015 THRTN    DEC 13
0018  16377                 ORG 16377B
0019  16377 070742 PRINT    SAR 16
```

—Continued

```
0020  16400 132534        STA I,I   I=VARIABLE SETTING SEVEN LINES FOR PRINTER.
0021  16401 070742 PRINN  SAR 16        DIFFERENT ENTRY POINTS DEP. ON FUNCTION .
0022  16402 132532        STA GROUP,I     1 OF 4 CHRA GROUPS.
0023  16403 070742 KILEM  SAR 16        CHAR POS WITHIN
0024  16404 132533        STA POS,I       A GROUP.
0025  16405 070742 KILWD  SAR 16
0026  16406 031737        STA OUTWD      WORD TO OUTPUT TO PRINTER.
0027  16407 024013 MAJFC  LDB YDIS       ADDRESS OF Y DISPLAY LIST.
0028  16410 106532        ADB GROUP,I
0029  16411 074517        LDA B,I         GETS CHARACTER FROM DISP FILE.
0030  16412 070546        RAR 12       POSITION IT.
0031  16413 074557        STA B,I       PUT IT BACK REPOSITIONED.
0032  16414 052537        AND MSK      ISOLATE CHAR.
0033  16415 031736        STA CHAR      SAVE TEMPORARILY.
0034  16416 070137        LDB A         COMPUTE BASE ADDRESS OF CHAR. CODE.
0035  16417 070704        SAL 2         4*CHAR
0036  16420 074071        SLB *+1,C     FORMS CHAR -1 FOR ODD CHARS.
0037  16421 074017        ADA B
0038  16422 070002        SAR 1         NOW HAVE OFFSET
0039  16423 002552        ADA CODBL      START OF CODE TABLE.
0040  16424 031735        STA ADDRS      ADDRESS OF BYTE OF CODE FOR THIS CHAR.
0041  16425 025736        LDB CHAR
0042  16426 022542        LDA TBL       OFFSET TABLE.
0043  16427 102534        ADA I,I
0044  16430 070517        LDA A,I        GETS EVEN ODD INSTRUCTIONS.
0045  16431 074111        SLB *+2
0046  16432 070404        SAL 8        EFFECTIVE MALSK.
0047  16433 070342        SAR 8
0048  16434 031734        STA TEMP       SAVE FOR EXTRACTION.
0049  16435 052537        AND MSK        MSK=17, GETS
0050  16436 001735        ADA ADDRS      ACTUAL ADDRESS OF OF PTR CODE.
0051  16437 070537        LDB A,I             "     "   " ACTUAL PTR CODE.
0052  16440 021734        LDA TEMP       RESTORE ROTATION WORD.
0053  16441 070142        SAR 4
0054  16442 070544        SAL 5       SETS ROTATION CODE.
0055  16443 042540        IOR ROTWD      FORMS RBR N.
0056  16444 070016        EXA            POSITIONS B.



0057  16445 136535        STB BUF,I        SAVES PRINT CODE
0058  16446 022536        LDA OCT42        FMT SEARCH
0059  16447 060662 GETIT  JSM SERCH        FINDS ROUTINE
0060  16450 126535        LDB BUF,I        RESTORE B
0061  00662         SERCH EQU 662B          ROUTINE THAT CHECKS FOR OPTIONAL BLOCKS.
0062  16451 074244 HERE   SBL 11
0063  16452 074202        SBR 5          MASK AND SET UP FOR POSITIONING.
0064  16453 122533        LDA POS,I
0065  16454 070111        SLA *+2
0066  16455 074544        SBL 5
0067  16456 021737        LDA OUTWD       GETS OUTPT WORD
0068  16457 074217        IOR B
0069  16460 031737        STA OUTWD
0070  16461 126533        LDB POS,I      CHECK IF TIME TO OUTPUT.
0071  16462 074311        SLB RETUR      IF POS = 0 OR 2 DO NOT OUTPUT.
0072  16463 014034        CPB THREE
0073  16464 066472        JMP PRTEM      POS =3, GO OUTPUT WORD WITH PEN.
0074  16465 031754        STA HOLD          SAVE FOR LATER OUTPUT.
0075  16466 146533        ISZ POS,I
0076  16467 066405        JMP KILWD
0077  16470 146533 RETUR  ISZ POS,I       INCREMENT POS, FROM 0 OR 2. GROUP = CONST.
0078  16471 066407        JMP MAJFC
0079  16472 142532 PRTEM  IOR GROUP,I
0080  16473 062516        JSM OUTK          OUTPUT AND WAIT FOR FLAG.
0081  16474 122532        LDA GROUP,I
0082  16475 000077        ADA M3
0083  16476 070553        SAP ICHCK        SOFTWARE PRINTER PROTECTION.
0084  16477 146532        ISZ GROUP,I
0085  16500 066403        JMP KILEM
0086  16501 026365 SPACE  LDB 163658      SPACE THREE LINES.
0087  16502 070742 SPC1   SAR 16
0088  16503 031754        STA HOLD         SAVE FOR LATER OUTPUT.
```

—Continued

```
0089  16504 142532        IOR GROUP,I
0090  16505 062516        JSM OUTK          OUTPUT AND WAIT FOR FLAGS.
0091  16506 074130        SIB *+2
0092  16507 066502        JMP SPC1
0093  16510 170402        RET
0094  16511 122534 ICHCK  LDA I,I
0095  16512 012541        CPA SIX
0096  16513 066501        JMP SPACE
0097  16514 146534        ISZ I,I
0098  16515 066401        JMP PRINN
0099  16516 035737 OUTK   STB OUTWD         SAVE CONTENTS OF B REG, HAS SPACING INFO
0100  16517 024060        LDB 60B           GET DELAY.
0101  16520 172504        SFS 4             SKIP IF PRINTER NOT READY.
0102  16521 066525        JMP OK            PRINTER READY, OUTPUT STUFF.
0103  16522 077730        RIB *-2           TIME NOT UP, CHECK PTF AGAIN.
0104  16523 062736        JSM UNROT         TIME UP, RESTORE STATUS.
0105  16524 064161        JMP STOP          AND STOP EVERYTHING.
0106  16525 025754 OK     LDB HOLD
0107  16526 176141        OTB 1             OUTPUT FIRST GROUP.
0108  16527 172144        OTA 4             OUTPUT 2ND GROUP, WITH PEN.
0109  16530 025737        LDB OUTWD         RESTORE B REGISTER.
0110  16531 170402        RET
0111  01754         HOLD  EQU 1754B         TEMP LOCATION FOR PRINTER CODE.
0112  00161         STOP  EQU 161B


0113  01737         OUTWD EQU 1737B
0114  16532 012003 GROUP  OCT 12003
0115  16533 012004 POS    OCT 12004
0116  01736         CHAR  EQU 1736B
0117  01735         ADDRS EQU 1735B
0118  01734         TEMP  EQU 1734B
0119  16534 012005 I      OCT 12005
0120  16535 012006 BUF    OCT 12006
0121  16536 000042 OCT42  OCT 42
0122  00013         YDIS  EQU 13B       ADDRS OF Y ISPLAY LIST.
0123  16537 000017 MSK    OCT 17
0124  16540 074006 ROTWD  OCT 74006
0125  16541 000006 SIX    DEC 6
0126  16542 016543 TBL    DEF *+1
0127  16543 110360        OCT 110360,040221,170101,110761
      16544 040221
      16545 170101
      16546 110761
0128  16547 040622        OCT 40622,170502,111362
      16550 170502
      16551 111362
0129  16552 016553 CODBL  DEF *+1
0130  16553 035063        OCT 35063,53461,34004,30204
      16554 053461
      16555 034004
      16556 030204
0131  16557 010216        OCT 10216,35041,14420,76016
      16560 035041
      16561 014420
      16562 076016
0132  16563 042046        OCT 42046,3056,4312,45742
      16564 003056
      16565 004312
      16566 045742
0133  16567 004037        OCT 4037,41701,3056,14420
      16570 041701
      16571 003056
      16572 014420
0134  16573 075061        OCT 75061,34037,2104,20410
      16574 034037
      16575 002104
      16576 020410
0135  16577 035061        OCT 35061,35061,34016,43057
      16600 035061
      16601 034016
      16602 043057
```

—Continued

```
0136  16603 002114          OCT 2114,0,76000,0
      16604 000000
      16605 076000
      16606 000000
0137  16607 000000          OCT 0,0,0,14
      16610 000000
      16611 000000
      16612 000014
0138  16613 030000          OCT 30000,24237,10500
      16614 024237



      16615 010500
0139  16616 162752 MANPT  JSM CHECK,I
0140  16617 062733        JSM ROT
0141  16620 120107        LDA 1078,I
0142  16621 070331        SLA NORML,C       LSB=1 IF UNNORMALIZED.
0143  16622 130107        STA 1078,I
0144  16623 021727        LDA XLSB
0145  16624 052374        AND CLEAN         GET SET TO INSET E IS DIS LIST.
0146  16625 042376        IOR THRTN
0147  16626 031727        STA XLSB
0148  16627 020012 NORML  LDA XDIS
0149  16630 024013        LDB YDIS
0150  16631 031720        STA 1720B      RESET THE CHANGE SIGN FLAG
0151  16632 170004        XFR
0152  16633 062377 NMBR   JSM PRINT
0153  16634 062736        JSM UNROT
0154  16635 060115 FCH    JSM 115B       PICK THE NEXT KEY
0155  16636 010110        CPA PRNT       IS IT A PRINT KEY?
0156  16637 066646        JMP SET        YES
0157  16640 124103        LDB 103B,I        LOAD THE EXECUTE FLAG.
0158  16641 076150        RZB *+3        SKIP IF EXECUTING A PROG
0159  16642 055777        DSZ 1777B      DECREMENT THE SYSTEM STACK
0160  16643 064340        JMP 340B       JUMP TO THE EXEC WITH KEY
0161  16644 055701        DSZ 1701B        DEC STACK PTR, RET. AVIODS EXECUTING NEXT
0162  16645 170402        RET              KEY AFTER SEQUENCES OF PRINTS ON STEP PRGM
0163  16646 062670 SET    JSM SPICE      SPACE THE PRINTER
0164  16647 066635        JMP FCH
0165  00110         PRNT  EQU 110B
0166  00012         XDIS  EQU 12B
0167  00357         INTR2 EQU 357B
0168  00167         INIT  EQU 167B
0169  01727         XLSB  EQU 1727B
0170*                   TO CHANGE PTR ORIGN, DEFINE THRTN DEC 13,
0171*                    THREE DEC 3, AND M3 DEC -3.
0172*                 LIST ROUTINE.
0173  16650 120111 LISTM LDA SECUR,I     GET SECURITY WORD.
0174  16651 073610        RZA SET-1        SECURE PROGRAM, DO NOT LIST.
0175  16652 062733        JSM ROT
0176  16653 072053        SAP *+1,S        BUFFER TO NEGATIVE.
0177  16654 031775        STA 1775B       LOOK FOR STOP KEY.
0178  16655 062670        JSM SPICE      SPACE THE PRINTER
0179  16656 021775 CKM1   LDA 1775B
0180  16657 070413        SAP EOUT
0181  16660 162752 STRT1  JSM CHECK,I
0182  16661 062377        JSM PRINT
0183  16662 121701        LDA PC,I
0184  16663 012002        CPA ENDM
0185  16664 066667        JMP EOUT
0186  16665 045701        ISZ PC
0187  16666 066656        JMP CKM1
0188  16667 062736 EOUT   JSM UNROT
0189  16670 024024 SPICE  LDB M10
0190  16671 020034        LDA THREE
0191  16672 132532        STA GROUP,I
0192  16673 066502        JMP SPC1
0193  16002         ENDM  EQU 16002B
```

—Continued

```
0194*
0195*
0196*                KEYLOG ROUTINES.
0197   16674 074344 KEYLG SBL 9
0198   16675 074113       SBP *+2
0199   16676 066731       JMP FRCM1
0200   16677 025775       LDB 1775B          INPUT BUFFER.
0201   16700 021776       LDA FLOAT
0202   16701 070544       SAL 5         POSITION KEYLOG BIT. ZERO IF OFF.
0203   16702 070112       SAM *+2
0204   16703 170402       RET           NO KEYLOG, RETURN.
0205   16704 070744       SAL 1         PICK THR RUN BIT
0206   16705 070252       SAM WORK      POSITIVE IF IN PROG MODE
0207   16706 055701       DSZ PC           FOOL LIST BUILDER.
0208   16707 162752       JSM CHECK,I
0209   16710 045701       ISZ PC          UNFOOL LIST BUILDER.
0210   16711 066726       JMP FRCM        FORCE TO PRGM MODE.
0211   16712 074117 WORK  LDA B
0212   16713 074102       SBR 3
0213   16714 074604       SBL 4
0214   16715 050770       AND MSKA
0215   16716 074217       IOR B         KEYCODE IN A.
0216   16717 070404       SAL 8
0217   16720 042375       IOR BLNKM        BLANKS FILLING IN.
0218   16721 031733       STA 1733B       PUT IN Y DISPLAY LIST.
0219   16722 022751       LDA ABLNK
0220   16723 031732       STA 1732B
0221   16724 031731       STA 1731B
0222   16725 031730       STA 1730B       BLANKS THE DISPLAY LIST.
0223   16726 062733 FRCM  JSM ROT
0224   16727 062377       JSM PRINT
0225   16730 062736       JSM UNROT
0226   16731 025775 FRCM1 LDB 1775B
0227   16732 170402       RET
0228   00770        MSKA  EQU 770B
0229   16733 021776 ROT   LDA FLOAT
0230   16734 070606       RAR 13
0231   16735 064331       JMP 331B            STORE AS STAUS WORD.
0232   16736 021776 UNROT LDA FLOAT
0233   16737 070106       RAR 3
0234   16740 026526 PUT   LDB OK+1
0235   16741 076030       RIB *
0236   16742 064330       JMP 330B         OUTPUT STATUS LITES.
0237*
0238*              KEYLOG BIT DIDDLER.
0239   16743 021776 KEYCG LDA FLOAT
0240   16744 070446       RAR 10
0241   16745 072111       SLA *+2,S
0242   16746 070071       SLA *+1,C
0243   16747 070246       RAR 6
0244   16750 064330       JMP 330B        STORES AND OUTPUTS A STATUS.
0245   00034        THREE EQU 34B
0246   00077        M3    EQU 77B
0247   00031        M1    EQU 31B
0248   00024        M10   EQU 24B
0249   01776        FLOAT EQU 1776B
0250   01701        PC    EQU 1701B
0251   16751 135673 ABLNK OCT 135673
0252   16752 025274 CHECK OCT 5274
0253   00111        SECUR EQU 111B
0254               END
*   NO ERRORS*

0001               ASMB,A,B,L,T
SRCH   010000
OPTON  010010
NOOK   010022
NMBR   010023
ZMPOS  010024
```

—Continued

```
TORTH   010027
OK      010032
SNU     010053
BUF     010112
MSK.1   010113
MSK.2   010114
MSK.3   010115
WDPTR   001755
OFFST   001755
RWORD   001727
M2      010116
YLEST   001733
ENTRY   010117
GETKY   010124
NEXT    010136
MSKEM   010143
OUT1    010145
KEPPT   010160
PNTM    010161
BLNK1   010163
STPEM   010167
FILL    010170
ALLF    010211
APRNT   010220
APRNN   010222
AKILM   010225
AKILW   010227
MAJFC   010231
ARET    010270
APRTM   010272
ASPAC   010301
ASPC1   010302
AICHK   010311
GOTEM   010316
OUTK    010321
```

—Continued

```
AGRUP   001756
APOS    001757
CNT     001744
TEMP    001745
AOUTW   001746
KEY     001747
ROTWD   010327
STOP    000021
MASK    010330
FLOAT   001776
GROUP   010331
POS     010332
I       010333
OFTBL   010334
BLN     010335
PCNT    001701
MASK0   000100
INBUF   001775
SIXTN   010336
SIX     010337
M3      000077
THREE   000034
ONE     000063
TWO     010340
LBL     010000
RETRN   010341
TABLE   000013
DTBL    010342
CTBL    010543
*  NO ERRORS*
```

```
0001                    ASMB,A,B,L,T
0002*
0003*                     ALPHA BLOCK.
0004*
0005  10000                       ORG 10000B        START OF ALPHA BLOCK.
0006  10000 000042 SRCH    OCT 42      THE SEARCH CODE FOR THIS BLOCK.
0007  10001 021777         LDA 1777B          STACK PTR, USE TO DETERMINE WHETHER
0008  10002 000031         ADA 31B         (-1)  WANT ALPHA OR MNEM.
0009  10003 070517         LDA A,I            GETE LOCATION OF RETURN PT.
0010  10004 012007         CPA 10007B         IS IS THE PRINTER AREA.
0011  10005 066010         JMP OPTON          YES, MUST WANT MNEM.
0012  10006 066117         JMP ENTRY          NO , INPLIES,ALPHA.
0013  10010                ORG 10010B
0014  10010 021777 OPTON   LDA 1777B        EXAMINE CONTENTS OF STACK TO
0015  10011 002116         ADA M2           DETERMINE IF ARE PRINTING A NUMBER
0016  10012 070517         LDA A,I          OR ARE DOING ALPHA OPERATION AS
0017  10013 012023         CPA NMBR         LIST, KEYLOG, ETC.
0018  10014 170402         RET              DOING NUMBER PRINT, DO NOT ENTER ALPHA.
0019  10015 122331         LDA GROUP,I      ONE OF THE OFF PAGE VARIABLES.
0020  10016 012340         CPA TWO
0021  10017 066024         JMP ZRPOS       CHECK FOR POS=0.
0022  10020 010063         CPA ONE          ALSO LEGAL.
0023  10021 066027         JMP TORTH        IS POS EQ 2 OR 3.
0024  10022 170402 NOOK    RET
0025  10023        NMBR    EQU *
0026  10024                ORG 10024B          NMBR SHOWS PRINT NUMBER.
0027  10024 122332 ZRPOS   LDA POS,I
0028  10025 073650         RZA NOOK
0029  10026 066032         JMP OK
0030  10027 122332 TORTH   LDA POS,I
0031  10030 070002         SAR 1           CHECK FOR TWO OR THREE.
0032  10031 071451         SLA NOOK        NO, RET.
0033  10032 122332 OK      LDA POS,I
0034  10033 002116         ADA M2
```

ALPHA PRINT BLOCK—Continued

```
0035  10034 070113        SAP *+2
0036  10035 070076        TCA
0037  10036 031755        STA WDPTR       NOW WORD PTR IS IN A(0,1,2).
0038  10037 021733        LDA YLEST
0039  10040 052113        AND MSK.1      MS HALF.
0040  10041 070137        LDB A
0041  10042 021733        LDA YLEST
0042  10043 052114        AND MSK.2       LS HALF
0043  10044 070744        SAL 1
0044  10045 074217        IOR B
0045  10046 070402        SAR 9           KEY NOW IN A.
0046  10047 070137        LDB A
0047  10050 070071        SLA *+1,C
0048  10051 070744        SAL 1           2*KEY.
0049  10052 001755        ADA WDPTR
0050  10053 002342 SNU    ADA DTBL      DIRECTIVE TABLE.
0051  10054 070517        LDA A,I        ENTRY OT CODE TABLE.
0052  10055 074111        SLB *+2
0053  10056 070404        SAL 8
0054  10057 070342        SAR 8         HAS ENTRY OT CODE TABLE.
0055  10060 031727        STA RWORD
0056  10061 070137        LDB A           FOR COMPUTING CORRECT ENTRY.



0057  10062 070704        SAL 2
0058  10063 074071        SLB *+1,C
0059  10064 074017        ADA B
0060  10065 070002        SAR 1
0061  10066 025727        LDB RWORD
0062  10067 031755        STA OFFST
0063  10070 122334        LDA OFTBL,I
0064  10071 102333        ADA I,I
0065  10072 070517        LDA A,I
0066  10073 074111        SLB *+2
0067  10074 070404        SAL 8
0068  10075 070342        SAR 8
0069  10076 031727        STA RWORD
0070  10077 052115        AND MSK.3       EQU MSK 0CT 17.
0071  10100 001755        ADA OFFST      HAS OFFSET FOR LINE.
0072  10101 002543        ADA CTBL
0073  10102 070537        LDB A,I         GETS PRITNER CODE.
0074  10103 021727        LDA RWORD
0075  10104 070142        SAR 4
0076  10105 070544        SAL 5       POSITIONS ROTATION.
0077  10106 042327        IOR ROTWD
0078  10107 070016        EXA
0079  10110 136112        STB BUF,I           PUT PRINTER CODE IN SAFE PLACE.
0080  10111 170402        RET
0081  10112 012006 BUF    OCT 12006
0082  10113 170000 MSK.1  OCT 170000
0083  10114 007400 MSK.2  OCT 7400
0084  10115 000017 MSK.3   OCT 17
0085  01755        WDPTR  EQU 1755B
0086  01755        OFFST  EQU WDPTR
0087  01727        RWORD  EQU 1727B
0088*   I,GROUP,AND POS WILL BE INDIRECT OT THE VARIABLES.
0089  10116 177776 M2     DEC -2
0090  01733        YLEST  EQU 1733B



0092*
0093*          ALPHA PRINT BLOCK
0094*
0095  10117 021775 ENTRY LDA INBUF
0096  10120 010300       CPA 300B          STEP PROGRAM.
0097  10121 066124       JMP GETKY
0098  10122 072052       SAM *+1,S
0099  10123 031775       STA INBUF
0100  10124 070742 GETKY SAR 16
```

ALPHA PRINT BLOCK—Continued

```
0101  10125 031744        STA CNT
0102  10126 120103        LDA 103B,I          A PROGRAM WITH NO ENDING FORMAT.
0103  10127 070350        SZA NEXT
0104  10130 121701        LDA PCNT,I      GET KEY FROM MEMORY.
0105  10131 045701        ISZ PCNT        SET UP PC FOR NEXT USER.
0106  10132 025775        LDB INBUF           CHECK FOR STOP.
0107  10133 014021        CPB STOP
0108  10134 066145        JMP OUT1
0109  10135 066143        JMP MSKEM
0110  10136 173741 NEXT    CLF 01          ENABLE INTERRUPT.
0111  10137 021775        LDA INBUF
0112  10140 071752        SAM *-1         STOP KEY YET?
0113  10141 076053        SBP *+1,S         AFTER KEY IS IN, RESET
0114  10142 035775        STB INBUF         INPUT BUFFER.
0115  10143 050100 MSKEM   AND MASK0      YES, CLOAN UP KEY IF 7 BIT.
0116  10144 010021        CPA STOP        IF STOP KEY,
0117  10145 064161 OUT1    JMP 1618            GO TO STOP ROUTINE.
0118  10146 012341        CPA RETRN       IS IT A RETURN COMMAND(CARRAIGE RETURN)?
0119  10147 066160        JMP KEPPT       KEEP PRINTING ANOTHER LINE AFTER THIS.
0120  10150 012000        CPA LBL
0121  10151 066163        JMP BLNK1     CLEAR AND PRINT REST OF LINE.
0122  10152 062170        JSM FILL       VALID KEY, PUT IN BUFFER AREA.
0123  10153 045744        ISZ CNT
0124  10154 021744        LDA CNT
0125  10155 012336        CPA SIXTN       IS BUFFER FULL?
0126  10156 066161        JMP PNTM        PRINT BUFFER AS IAS.
0127  10157 066126        JMP GETKY+2         BYPASS SETTING INPUT BUF NEG.
0128  10160 062211 KEPPT   JSM ALLF
0129  10161 062220 PNTM    JSM APRNT
0130  10162 066124        JMP GETKY
0131  10163 021744 BLNK1   LDA CNT
0132  10164 070150        SZA STPEM     TAKES CARE OF LBL-LBL AND 17TH CHAR = LBL.
0133  10165 062211        JSM ALLF
0134  10166 062220        JSM APRNT
0135  10167 170402 STPEM   RET           MAIN RETURN.
0136*
0137  10170 031747 FILL    STA KEY
0138  10171 021744        LDA CNT
0139  10172 025744        LDB CNT
0140  10173 070002        SAR 1
0141  10174 000013        ADA TABLE
0142  10175 031745        STA TEMP        ADD OF OCATION.
0143  10176 070517        LDA A,I
0144  10177 074111        SLB *+2
0145  10200 070346        RAR 8
0146  10201 070346        RAR 8
0147  10202 052330        AND MASK



0148  10203 041747        IOR KEY
0149  10204 074111        SLB *+2
0150  10205 070346        RAR 8
0151  10206 070346        RAR 8
0152  10207 131745        STA TEMP,I
0153  10210 170402        RET
0154*
0155  10211 022335 ALLF    LDA BLN          FILLS REMAINDER OF BUFFER WITN BLANKS.
0156  10212 062170        JSM FILL
0157  10213 045744        ISZ CNT
0158  10214 021744        LDA CNT
0159  10215 012336        CPA SIXTN
0160  10216 170402        RET
0161  10217 066211        JMP ALLF
0162*
0163  10220 070742 APRNT   SAR 16          THIS ROUTINE PRINTS THE CONTENTS OF THE
0164  10221 132333        STA I,I
0165  10222 070742 APRNN   SAR 16
0166  10223 031744        STA CNT       RESTART THE LINE COUNT.
0167  10224 031756        STA AGRUP
0168  10225 070742 AKILM   SAR 16
0169  10226 031757        STA APOS
0170  10227 070742 AKILW   SAR 16
```

ALPHA PRINT BLOCK—Continued

```
0171  10230 031746        STA AOUTW        OUTPUT WORD.
0172  10231 021744 MAJFC  LDA CNT        FETCH WD FROM BUFFER.
0173  10232 012336        CPA SIXTN
0174  10233 066316        JMP GOTEM
0175  10234 045744        ISZ CNT
0176  10235 070137        LDB A
0177  10236 074002        SBR 1
0178  10237 004013        ADB TABLE
0179  10240 074537        LDB B,I
0180  10241 070111        SLA *+2
0181  10242 074404        SBL 8
0182  10243 074342        SBR 8
0183  10244 074117        LDA B
0184  10245 070071        SLA *+1,C
0185  10246 070744        SAL 1
0186  10247 000034        ADA THREE
0187  10250 062053        JSM SNU
0188  10251 074244        SBL 11
0189  10252 074202        SBR 5
0190  10253 021757        LDA APOS
0191  10254 070111        SLA *+2
0192  10255 074544        SBL 5
0193  10256 021746        LDA AOUTW
0194  10257 074217        IOR B          B IS PRINTER CODE.
0195  10260 031746        STA AOUTW       JUST AS IN BASIC PRINT ROUTINE.
0196  10261 025757        LDB APOS
0197  10262 074311        SLB ARET
0198  10263 014034        CPB THREE
0199  10264 066272        JMP APRTM      POS=3, OUTPUT WITH PEN.
0200  10265 172141        OTA 01         POS=1, OUTPUT WITHOUT PEN.
0201  10266 045757        ISZ APOS
0202  10267 066227        JMP AKILW
0203  10270 045757 ARET   ISZ APOS

0204  10271 066231        JMP MAJFC
0205  10272 041756 APRTM  IOR AGRUP
0206  10273 062321        JSM OUTK
0207  10274 021756        LDA AGRUP
0208  10275 014034        CPA THREE
0209  10276 066311        JMP AICHK
0210  10277 045756        ISZ AGRUP
0211  10300 066225        JMP AKILM
0212  10301 024077 ASPAC  LDB M3
0213  10302 070742 ASPC1  SAR 16
0214  10303 172141        OTA 1
0215  10304 041756        IOR AGRUP
0216  10305 062321        JSM OUTK
0217  10306 074130        SIB *+2
0218  10307 066302        JMP ASPC1
0219  10310 170402        RET
0220  10311 122333 AICHK  LDA I,I
0221  10312 012337        CPA SIX
0222  10313 066301        JMP ASPAC
0223  10314 146333        ISZ I,I
0224  10315 066222        JMP APRNN
0225  10316 020034 GOTEM  LDA THREE
0226  10317 031756        STA AGRUP
0227  10320 066311        JMP AICHK
0228  10321 172144 OUTK   OTA 4
0229  10322 020060        LDA 60B          TIME DELAY.
0230  10323 172504        SFS 4
0231  10324 170402        RET
0232  10325 073730        RIA *-2
0233  10326 064161        JMP 161B
0234  01756         AGRUP EQU 1756B
0235  01757         APOS  EQU 1757B
0236  01744         CNT   EQU 1744B
0237  01745         TEMP  EQU 1745B
0238  01746         AOUTW EQU 1746B
0239  01747         KEY   EQU 1747B
```

ALPHA PRINT BLOCK—Continued

```
0240  10327 074006 ROTWD OCT 74006
0241  00021        STOP  EQU 21B
0242  10330 177400 MASK  OCT 177400
0243  01776        FLOAT EQU 1776B
0244  10331 012003 GROUP OCT 12003
0245  10332 012004 POS   OCT 12004
0246  10333 012005 I     OCT 12005
0247  10334 016542 OFTBL OCT 16542
0248  10335 000047 BLN   OCT 47
0249  01701        PCNT  EQU 1701B
0250  00100        MASK0 EQU 100B
0251  01775        INBUF EQU 1775B
0252  10336 000020 SIXTN DEC 16
0253  10337 000006 SIX   DEC 6
0254  00077        M3    EQU 77B
0255  00034        THREE EQU 34B
0256  00063        ONE   EQU 63B
0257  10340 000002 TWO   DEC 2
0258  10000        LBL   EQU 10000B
0259  10341 000020 RETRN OCT 20


0260  00013        TABLE EQU 13B
0261*       CODE TABLES FOR PRINT ROUTINES.
0262  10342 010343 OTBL  DEF *+1
0263  10343 027056       OCT 27056,1,27056,1
      10344 000001
      10345 027056
      10346 000001
0264  10347 027056       OCT 27056,1003,27056,1003
      10350 001003
      10351 027056
      10352 001003
0265  10353 027056       OCT 27056,2005,27056,2005
      10354 002005
      10355 027056
      10356 002005
0266  10357 027056       OCT 27056,3007,27056,3007
      10360 003007
      10361 027056
      10362 003007
0267  10363 027056       OCT 27056,4011,27056,4011
      10364 004011
      10365 027056
      10366 004011
0268  10367 021456       OCT 21456,17072,16056,22435
      10370 017072
      10371 016056
      10372 022435
0269  10373 027056       OCT 27056,35422,27056,16022
      10374 035422
      10375 027056
      10376 016022
0270  10377 027001       OCT 27001,10464,27043,10440
      10400 010464
      10401 027043
      10402 010440
0271  10403 007056       OCT 7056,13413,16456,30013
      10404 013413
      10405 016456
      10406 030013
0272  10407 016443       OCT 16443,20037,15432,34037
      10410 020037
      10411 015432
      10412 034037
0273  10413 027056       OCT 27056,22017,10031,21467
      10414 022017
      10415 010031
      10416 021467
0274  10417 010056       OCT 10056,10040,21433,33067
      10420 010040
```

ALPHA PRINT BLOCK—Continued

```
        10421 021433
        10422 033067
0275    10423 021424        OCT 21424,10031,22017,33442
        10424 010031
        10425 022017
        10426 033442
0276    10427 007056        OCT 7056,11446,17056,26046

        10430 011446
        10431 017056
        10432 026046
0277    10433 027017        OCT 27017,5024,27041,5064
        10434 005024
        10435 027041
        10436 005064
0278    10437 027016        OCT 27016,21427,27043,34474
        10440 021427
        10441 027043
        10442 034474
0279    10443 022036        OCT 22036,17437,15033,17000
        10444 017437
        10445 015033
        10446 017000
0280    10447 010424        OCT 10424,14021,17422,32465
        10450 014021
        10451 017422
        10452 032465
0281    10453 011033        OCT 11033,17431,15037,24467
        10454 017431
        10455 015037
        10456 024467
0282    10457 010016        OCT 10016,14431,7437,33456
        10460 014431
        10461 007437
        10462 033456
0283    10463 021427        OCT 21427,24015,22027,30460
        10464 024015
        10465 022027
        10466 030460
0284    10467 021443        OCT 21443,36457,22044,31052
        10470 036457
        10471 022044
        10472 031052
0285    10473 017056        OCT 17056,10426,13456,24026
        10474 010426
        10475 013456
        10476 024026
0286    10477 027033        OCT 27033,26036,27020,15463
        10500 026036
        10501 027020
        10502 015463
0287    10503 027056        OCT 27056,10016,27056,10016
        10504 010016
        10505 027056
        10506 010016
0288    10507 027056        OCT 27056,6017,27056,6017
        10510 006017
        10511 027056
        10512 006017
0289    10513 012056        OCT 12056,14424,17456,20424
        10514 014424
        10515 017456
        10516 020424
0290    10517 027043        OCT 27043,6421,27035,6444
        10520 006421
        10521 027035
        10522 006444
```

ALPHA PRINT BLOCK—Continued

```
0291  10523 027056        OCT 27056,14032,27056,14032
      10524 014032
      10525 027056
      10526 014032
0292  10527 027056        OCT 27056,13431,27056,13431
      10530 013431
      10531 027056
      10532 013431
0293  10533 027056        OCT 27056,11425,27056,11425
      10534 011425
      10535 027056
      10536 011425
0294  10537 027036        OCT 27036,23464,27035,23467
      10540 023464
      10541 027035
      10542 023467
0295*     END OF DIRECTIVE TABLE.
0296  10543 010544 CTBL   DEF *+1
0297  10544 035063        OCT 35063,53461,34004,30204,10216
      10545 053461
      10546 034004
      10547 030204
      10550 010216
0298  10551 035041        OCT 35041,14420,76016,42046,3056
      10552 014420
      10553 076016
      10554 042046
      10555 003056
0299  10556 004312        OCT 4312,45742,4037,41701,3056
      10557 045742
      10560 004037
      10561 041701
      10562 003056
0300  10563 014420        OCT 14420,75061,34037,2104,20410
      10564 075061
      10565 034037
      10566 002104
      10567 020410
0301  10570 035061        OCT 35061,35061,34016,43057,2114
      10571 035061
      10572 034016
      10573 043057
      10574 002114
0302  10575 000000        OCT 0,76000,0,0,614
      10576 076000
      10577 000000
      10600 000000
      10601 000614
0303  10602 035061        OCT 35061,77061,42036,43076,43076
      10603 077061
      10604 042036
      10605 043076
      10606 043076

0304  10607 035060        OCT 35060,41021,34036,22451,22476
      10610 041021
      10611 034036
      10612 022451
      10613 022476
0305  10614 077020        OCT 077020,75020,76037,41036,41020
      10615 075020
      10616 076037
      10617 041036
      10620 041020
0306  10621 037020        OCT 37020,41161,36021,43077,43061
      10622 041161
      10623 036021
      10624 043077
      10625 043061
0307  10626 034204        OCT 34204,10204,34001,2041,3056
```

ALPHA PRINT BLOCK—Continued

```
        10627 010204
        10630 034001
        10631 002041
        10632 003056
0308    10633 043124        OCT 43124,61222,42020,41020,41037
        10634 061222
        10635 042020
        10636 041020
        10637 041037
0309    10640 043565        OCT 43565,53061,42021,43465,47061
        10641 053061
        10642 042021
        10643 043465
        10644 047061
0310    10645 035061        OCT 35061,43061,34036,43076,41020
        10646 043061
        10647 034036
        10650 043076
        10651 041020
0311    10652 035061        OCT 35061,43262,32036,43076,51121
        10653 043262
        10654 032036
        10655 043076
        10656 051121
0312    10657 035060        OCT 35060,34061,34037,10204,10204
        10660 034061
        10661 034037
        10662 010204
        10663 010204
0313    10664 043061        OCT 43061,43061,34021,43052,24204
        10665 043061
        10666 034021
        10667 043052
        10670 024204
0314    10671 043061        OCT 43061,53265,24021,42504,25061
        10672 053265
        10673 024021
        10674 042504
        10675 025061
0315    10676 043052        OCT 43052,10204,10037,2104,21037


        10677 010204
        10700 010037
        10701 002104
        10702 021037
0316    10703 000204        OCT 204,76204,0,36410,21410
        10704 076204
        10705 000000
        10706 036410
        10707 021410
0317    10710 000037        OCT 37,1740,12,25752,76512
        10711 001740
        10712 000012
        10713 025752
        10714 076512
0318    10715 024500        OCT 24500,0,4,10204,10204
        10716 000000
        10717 000004
        10720 010204
        10721 010204
0319    10722 000056        OCT 56,64512,24004,10204,10004
        10723 064512
        10724 024004
        10725 010204
        10726 010004
0320    10727 000000        OCT 0,0,20,20202,10420
        10730 000000
        10731 000020
        10732 020202
        10733 010420
```

ALPHA PRINT BLOCK—Continued

```
0321   10734 010764        OCT 10764,34276,10030,62104,21143
       10735 034276
       10736 010030
       10737 062104
       10740 021143
0322   10741 004204        OCT 4204,10204,4010,10204,10210
       10742 010204
       10743 004010
       10744 010204
       10745 010210
0323   10746 000042        OCT 42,10420,16,43042,10004
       10747 010420
       10750 000016
       10751 043042
       10752 010004
0324   10753 000202        OCT 202,76104,0,0,0
       10754 076104
       10755 000000
       10756 000000
       10757 000000
0325   10760 035041        OCT 35041,33265,24000,24237,10500
       10761 033265
       10762 024000
       10763 024237
       10764 010500
0326   10765 000014        OCT 14,45122,32020,41034,45114
       10766 045122

       10767 032020
       10770 041034
       10771 045114
0327   10772 000000        OCT 0,30604,20002,10420,20202
       10773 030604
       10774 020002
       10775 010420
       10776 020202
0328                       END
*  NO ERRORS*
```

—Continued

```
0001                ASMB,A,B,L,T
ABORT  007015
INTR2  000362
TENX   007030
GETD   007033
FIXD   007037
ENDIT  007050
FLOAT  007051
PNTKY  000110
FMTKY  007054
LISTK  007055
PRINT  007056
PNT    007063
SMAK   007075
OUTSP  007102
FIX0   007106
OK     007124
CKM    007130
OUTAL  007136
DONE   007144
STAR   007145
TEN    000104
M10    000024
ELEVN  007152
TIMES  007153
W      007154
DOT    000020
MSK    007155

TEMP   001737
CHECK  007156
INIT   007157
M15    007160
M16    007161
FIFTN  007162
M1     000031
FORTN  007163
ROUT   001736
ASTER  007164
SPACE  007165
DBLNK  007166
SETGT  007167
GET    007173
#DPNT  001735
WDCNT  001734
M4     007204
OUTPT  007205
NMBR   007216
REALO  007217
DLY    007223
EOL    007233
STP    007237
DLY1   007242
ERR    007243
DELAY  000063
TIME   007244
CR     007245
```

—Continued

```
LF      000104
SCLFD   007246
TWO     007247
ONE     000063
OCT56   007250
OCT40   007251
OCT55   007252
DEC58   007253
LISTM   007254
KLU     007263
KEYOT   007277
NXTL    007321
FINE    007341
CRLF    007342
RUN     007347
YADD    000005
PC      001701
M5      000023
SECUR   007355
END     007356
SIX     007357
THREE   000034
OUTA    007360
FMT     007365
```

—Continued

```
KEYIN   007367
KOUT    007414
SDIDL   007433
CRTAB   007440
TBM     007445
PUTIT   007461
RETLF   007466
DOPRN   007470
UPARO   007472
CLR     007473
DNARO   007474
SHIFT   007475
TAB     007476
CLRAL   007477
GETKY   007500
FLGCK   007510
MEMGT   007512
DISP    007517
CONTU   007532
CNTKY   000022
INBUF   001775
DISPY   000017
BASE    007535
*  NO ERRORS*
```

9810 TYPEWRITER BLOCK.

```
0001                ASMB,A,B,L,T
0003  07000           ORG 7000B
0004  07000 000002     OCT 2          SEARCH POINT ENTRY.
0005  07001 020031     LDA M1           SET THE STEP PROGRAM FLAG.
0006  07002 031720     STA 1720B
0007  07003 063500     JSM GETKY    GET NEXT KEY.
0008  07004 013054     CPA FMTKY      IS THIS A FORMAT KEY?
0009  07005 067365     JMP FMT       YES, START ALPHA PRINT.
0010  07006 010110     CPA PNTKY     PRINT OPERATION?
0011  07007 067063     JMP PNT        YES, CALL LIST BUILDER, PRINT, ETC.
0012  07010 013055     CPA LISTK      LIST KEY?
0013  07011 067254     JMP LISTM        YES, LIST PROGRAM ON TYPEWRITER.
0014  07012 000024     ADA M10         CHECK FOR A NUMBER.
0015  07013 070212     SAM *+4           YESM, IT'S A NUMBER. USE FOR FIELD WIDTH
0016  07014 000104     ADA TEN          NO NUMBER, GEVE KEY TO INTERPRETER.
0017  07015 070137 ABORT LDB A         BABY THE INTERPRETER.
0018  07016 064362     JMP INTR2
0019  00362         INTR2 EQU 362B     ADDS TABLE OFFSET TO B AND JUMPS TO ROUTINE
0020  07017 000104     ADA TEN          RESTORE THE NUMBER.
0021  07020 133154     STA W,I          PUT IN WIDTH ALLOCATED STORE.
0022  07021 031717     STA 17178      PAMPER TIMES TEN SUBR.
0023  07022 063500     JSM GETKY      GET NEXT KEY, COULD BE WIDTH OR TERMINATOR.
0024  07023 000024     ADA M10         DIGIT?
0025  07024 070212     SAM TENX       YES, USE AS PART OF WIDTH SPEC.
0026  07025 013152     CPA ELEVN       CHECK FOR DECIMAL.
0027  07026 067037     JMP FIXD         USE DECIMAL AS TERMINATOR FOR WIDTH.
0028  07027 067014     JMP ABORT-1      THIS KEY IS NOT PART OF SEQUENCE, ABORT
0029*                    BY RETURNING KEY TO MONITOR.
0030  07030 000104 TENX  ADA TEN       RESTORE VALUE.
0031  07031 163153       JSM TIMES,I    TAKE 10 TEMES PREVIOUS DIGIT PLUS CURRENT D
0032  07032 137154       STB W,I       PUT RESULT IN WIDTH ALLOCATED STORE.
0033  07033 063500 GETD  JSM GETKY       GET DECIMAL SETTING KEY.
0034  07034 010020       CPA DOT       WAS IT A DECIMAL*?
0035  07035 067037       JMP *+2         YES, GET NEXT KEY.
0036  07036 067015       JMP ABORT     CAN'T RECOGNIZE, GIVE TO INTERPERETER.
0037  07037 063500 FIXD  JSM GETKY      WILL NOW SET DECIMAL FIELD.
0038  07040 000024       ADA M10         DIGIT?
0039  07041 070413       SAP FLOAT      NO, SET TO FLOATING PT.
0040  07042 000104       ADA TEN         RESTORE.
0041  07043 031737       STA TEMP
0042  07044 021776       LDA 1776B      CALCULATOR STATUS WORD.
0043  07045 053155       AND MSK        CLEAN LAST FOUR BITS.
0044  07046 041737       IOR TEMP        SET NEW DECIMAL FIELD.
0045  07047 031776       STA 1776B      AND SAVE RESULT.
0046  07050 067056 ENDIT JMP PRINT       CHECK NEXT KEY FOR PRINT?
```

9810 TYPEWRITER BLOCK—Continued

```
0047  07051 021776 FLOAT LDA 17768      SET MACHINE TO FLOATING PT.
0048  07052 072053       SAP *+1,S      SET FT. PT.
0049  07053 067047       JMP ENDIT-1      SAVE RESULT, CHECK NEXT KEY FOR PRINT.
0050  00110        PNTKY EQU 110B
0051  07054 000042 FMTKY OCT 42
0052  07055 000104 LISTK OCT 104
0053  07056 172160 PRINT OTA 16          OUTPUT STATUS.
0054  07057 063500       JSM GETKY
0055  07060 010110       CPA PNTKY        WAS IT A PRINT?
0056  07061 067063       JMP PNT          YES, TYPE IT.
0057  07062 067015       JMP ABORT


              TYPEWRITER PRINT ROUTINE.

0060  07063 163156 PNT    JSM CHECK,I      BUILD DISPLAY LIST.
0061  07064 123154        LDA W,I       CHECK WIDTH.
0062  07065 072150        RZA *+3         IF ZERO, SET TO INITIAL VALUE.
0063  07066 023157        LDA INIT
0064  07067 133154        STA W,I         INIT IS PRESET VALUE(DEFAULT).
0065  07070 003161        ADA M16         IF WIDTH LARGE, OUTPUT BLANKS
0066  07071 070213        SAP SMAK       OUTPUT BLANKS TO RIGHT JUSTIFY.
0067  07072 123154        LDA W,I
0068  07073 031736        STA ROUT       SET ACTUAL NUMBER OF DIGITS OUT.
0069  07074 067106        JMP FIX0       CHECK FOR FIXED 0.
0070  07075 023162 SMAK   LDA FIFTN      SET ROUT TO FIFTEEN DEGITS OUT.
0071  07076 031736        STA ROUT
0072  07077 070076        TCA
0073  07100 103154        ADA W,I        OUTPUT WW-15 BLANKS.
0074  07101 031737        STA TEMP
0075  07102 023165 OUTSP  LDA SPACE
0076  07103 063217        JSM REALO
0077  07104 055737        DSZ TEMP        DONE?
0078  07105 067102        JMP OUTSP         NOPE.

0080  07106 020012 FIX0   LDA 12B           SET UP GET OPERATION ON X DISPLAY REG.
0081  07107 063167        JSM SETGT
0082  07110 027162        LDB FIFTN       B+15, FOR OVERFLOW CHECK.
0083  07111 021776        LDA 17768       IF FIXED ZERO, SUPPRESS DECIMAL OUT.
0084  07112 070512        SAM OK            FLOATING PT, CAN'T BE FIXED.
0085  07113 053162        AND FIFTN         CHECK LEAST 4 BITS FOR 0.
0086  07114 072410        RZA OK          NON ZERO, CANT BE FIXED ZERO.
0087  07115 123154        LDA W,I          IF W>15, WANT TO OUTPUT SPACE, OTHERWISE
0088  07116 003160        ADA M15          DON'T.
0089  07117 070212        SAM OK-1         W < 15, DON'T OUTPUT SPACE.
0090  07120 023165        LDA SPACE          FIXED ZERO, SPACE AND FOOL OVERFLOW.
0091  07121 063217        JSM REALO       OUTPUT WITHOUT ASCII CONVERSION.
0092  07122 055736        DSZ ROUT          KEEP DECIMAL FROM GOING OUT.
0093  07123 027163        LDB FORTN
0094  07124 074076 OK     TCB
0095  07125 005736        ADB ROUT
0096  07126 074413        SBP OUTAL      DO NOT CHECK ANY DIGITS FOR BLANKS.
0097  07127 035737        STB TEMP
0098  07130 063173 CKM    JSM GET        GET DIGIT FROM DISPLAY LIST.
0099  07131 013166        CPA DBLNK          IS THIS A BLANK IN THE DISPLAY LIST?
0100  07132 067134        JMP *+2            YES, LOOK FOR MORE , CHECK ING FOR
0101  07133 067145        JMP STAR           OVERFLOW. STAR OUTPUTS ASTERISKS.
0102  07134 045737        ISZ TEMP
0103  07135 067130        JMP CKM
0104  07136 021736 OUTAL  LDA ROUT       OUTPUT THE DIGITS FROM DIL LIST.
0105  07137 031737        STA TEMP       NOW OUTPUT FIRST TEMP THINGS.
0106  07140 063173        JSM GET
0107  07141 063205        JSM OUTPT
0108  07142 055737        DSZ TEMP
0109  07143 067140        JMP *-3
0110  07144 170402 DONE   RET


0112  07145 023164 STAR   LDA ASTER         OUTPUT *'S ON OVERFLOW.
0113  07146 063217        JSM REALO         OUTPUT WITHOUT ASCII CONVERSION.
```

TYPEWRITER PRINT ROUTINE—Continued

```
0114  07147 055736        DSZ ROUT
0115  07150 067145        JMP STAR
0116  07151 170402        RET


0118  00104         TEN   EQU 104B
0119  00024         M10   EQU 24B
0120  07152 000007 ELEVN DEC 7          A PURE KLUDGE, DETECTS DEC. AFTER SUB 10.
0121  07153 004176 TIMES OCT 4176
0122  07154 012007 W      OCT 12007      WIDTH LOCATION.
0123  00020         DOT   EQU 20B
0124  07155 077760 MSK    OCT 77760
0125  01737         TEMP  EQU 1737B
0126  07156 005274 CHECK  OCT 5274
0127  07157 000024 INIT   DEC 20
0128  07160 177761 M15    DEC -15
0129  07161 177760 M16    DEC -16
0130  07162 000017 FIFTN  DEC 15
0131  00031         M1    EQU 31B
0132  07163 000016 FORTN  DEC 14
0133  01736         ROUT  EQU 1736B
0134  07164 000052 ASTER  OCT 52
0135  07165 000040 SPACE  OCT 40
0136  07166 000013 DBLNK  DEC 11


              INPUT OUTPUT SUBPROGRAMS.


0138  07167 031735 SETGT  STA WDPNT    SET TO ANY REGISTER ADDRESS.
0139  07170 027204        LDB M4
0140  07171 035734        STB WDCNT
0141  07172 170402        RET             INITIALIZED VALUES FOR GET ROUTINE.
0142  07173 121735 GET    LDA WDPNT,I
0143  07174 070546        RAR 12           STANDARD ACCESS METHOD.
0144  07175 131735        STA WDPNT,I
0145  07176 053162        AND FIFTN        ISOLATE THE DISPLAY LIST MEMBER.
0146  07177 045734        ISZ WDCNT
0147  07200 170402        RET
0148  07201 063170        JSM SETGT+1      RESTORE WDCNT.
0149  07202 045735        ISZ WDPNT
0150  07203 170402        RET
0151  01735         WDPNT EQU 1735B
0152  01734         WDCNT EQU 1734B
0153  07204 177774 M4     DEC -4


0155  07205 000024 OUTPT  ADA M10         IS THIS A NUMBER? CONVERT TO ASCII.
0156  07206 070412        SAM NMBR       YES IT IS.
0157  07207 013247        CPA TWO          NO, IS IT A DECIMAL.
0158  07210 023250        LDA OCT56        YES, SET 56.
0159  07211 010063        CPA ONE        IS IT A SPACE?
0160  07212 023251        LDA OCT40          YES SET 40.
0161  07213 072210        RZA *+4
0162  07214 023252        LDA OCT55       MUST BE -, SET 55.
0163  07215 067217        JMP *+2
0164  07216 003253 NMBR   ADA DEC58          CONVERT NUMBER TO ASCII.
0165  07217 043246 REALO  IOR SCLED         CODE NOW IN A.
0166  07220 172141        OTA 01
0167  07221 172601        STC 01
0168  07222 074742        SBR 16          BEGIN TIME DELAY FOR TYPEWRITER UNCONNECTED
0169  07223 020063 DLY    LDA DELAY
0170  07224 172701        SFC 01         SKIP IF DEVICE NOT READY.
0171  07225 067233        JMP EOL        CHECK FOR END OF LINE.
0172  07226 073730        RIA *-2
0173  07227 074070        SIB *+1          INCREMENT B FOR MULTIPLE DELAY.
0174  07230 017244        CPB TIME       TIME UP?
0175  07231 064161        JMP 1618         YES, HALT PROGRAM.
0176  07232 067223        JMP DLY          NO, WAIT MORE.
0177  07233 172241 EOL    LIA 01
0178  07234 070604        SAL 4            STATUS LINE FROM TYPEWRITER INTERFACE.
0179  07235 070112        SAM STP          END OF LINE, SKIP TO ERROR CONDITION.
0180  07236 170402        RET          NO END OF LINE, RETURN.
```

INPUT OUTPUT SUBPROGRAMS.—Continued

```
0181  07237 163243 STP   JSM ERR,I        END OF ILINE INDICATION.
0182  07240 063347       JSM RUN          RESET TO RUN MODE.
0183  07241 064161       JMP 161B         JUMP TO STOP ROUTINE.
0184  07242 166170 DLY1  DEC -5000        DELAY FOR END OF LINE.
0185  07243 004750 ERR   OCT 4750
0186  00063        DELAY EQU 63B
0187  07244 000001 TIME  OCT 1            SET WAIT TIME FOR FLAG.
0188  07245 000015 CR    OCT 15
0189  00104        LF    EQU 104B
0190  07246 170000 SCLED OCT 170000



0191  07247 000002 TWO   OCT 2
0192  00063        ONE   EQU 63B
0193  07250 000056 OCT56 OCT 56
0194  07251 000040 OCT40 OCT 40
0195  07252 000055 OCT55 OCT 55
0196  07253 000072 DEC58 DEC 58

          TYPEWRITER LIST ROUTINE.

0198  07254 123355 LISTM LDA SECUR,I          GET THE SECURITY WORD.
0199  07255 073050       RZA STP-1           RETURN IF NON-ZERO => SECURE.
0200  07256 021776       LDA 1776B           GET STATUS WORD.
0201  07257 070406       RAR 9          LOOK AT RUN/PROGRAM BIT.
0202  07260 070071       SLA *+1,C        FORCE TO PROGRAM MODE.
0203  07261 070306       RAR 7          RESTORE.
0204  07262 031776       STA 1776B
0205  07263 163156 KLU   JSM CHECK,I      BUILD DISPLAY LIST.
0206  07264 020013       LDA 13B          Y DISPLAY LIST ADDRESS.
0207  07265 063167       JSM SETGT       INITIALIZE GET ROUTINE.
0208  07266 063342       JSM CRLF         OUTPUT CR/LF TO TYPEWRITER.
0209  07267 023357       LDA SIX          OUTPUT FIRST 6 ELEMENTS OF DISPLAY LIST.
0210  07270 031737       STA TEMP
0211  07271 063173       JSM GET        EXTRACTS FIRST ELEMENT OF DISPLAY LIST.
0212  07272 063205       JSM OUTPT      AND OUTPUTS IT.
0213  07273 055737       DSZ TEMP       ARE 6 DONE YET?
0214  07274 067271       JMP *-3        NOPE.
0215  07275 070742       SAR 16
0216  07276 031737       STA TEMP        YES, NOW PUT OUT 3 NMEMONIC CHARACTERS.
0217  07277 121701 KEYOT LDA PC,I        GET CURRENT KEYCIDE.
0218  07300 070002       SAR 1           COMPUTE OFFSET ENTRY INTO CONVERSION TABLE.
0219  07301 070137       LDB A
0220  07302 070704       SAL 2
0221  07303 074017       ADA B            BASIC OFFSET FOR CHARACTER NOW IN B.
0222  07304 025737       LDB TEMP
0223  07305 014034       CPB THREE          HAVE THREE CHARACTERS BEEN OUTPUTTED?
0224  07306 067321       JMP NXTL       NEXT LIST (REST OF DISPLAY LIST).
0225  07307 045737       ISZ TEMP
0226  07310 074017       ADA B          GET POINTER TO ACTUAL CODE FOR THAT CHARACTER.
0227  07311 003535       ADA BASE        BASE OF CODE TABLE.
0228  07312 070016       EXA       EXECUTE LOAD A INSTRUCTION. FETCH CODE.
0229  07313 125701       LDB PC,I       WANT TO POSITION CHODE.
0230  07314 074111       SLB *+2
0231  07315 070404       SAL 8
0232  07316 070342       SAR 8           NOW CODE IS UNPACKED IN LEAST 8 BITS.
0233  07317 063217       JSM REALO       OUTPUT EXACT CODE.
0234  07320 067277       JMP KEYOT        REPEAT FOR NECT CHARACTER.



0236  07321 020023 NXTL  LDA M5         OUTPUT REST OF DISPLAY LIST.
0237  07322 031737       STA TEMP
0238  07323 063173       JSM GET        BUPASS THE THREE CHARACTERS NOT OUTPUTTED.
0239  07324 063173       JSM GET
0240  07325 063173       JSM GET
0241  07326 063173       JSM GET        NOW WANT THIS CHARACTER.
0242  07327 063205       JSM OUTPT        OUTPUT IT.
0243  07330 045737       ISZ TEMP       DONE?
0244  07331 067326       JMP *-3        YOU BET NOT.
```

TYPEWRITER LIST ROUTINE.—Continued

```
0245  07332 121701        LDA PC,I
0246  07333 013356        CPA END       END OF LISTING?
0247  07334 067341        JMP FINE      YES, QUIT.
0248  07335 045701        ISZ PC         NO, GET NEXT PROGRAM STEP.
0249  07336 172700        SFC 00          SKIP IF STOP KEY NOT PRESSED.
0250  07337 067360        JMP OUTA          GO HANDLE STOP OPERATION.
0251  07340 067263        JMP KLU



0252  07341 063347 FINE   JSM RUN       RESTORE RUN MODE.
0253  07342 023245 CRLF   LDA CR        OUTPUT CR/LF.
0254  07343 063217        JSM REALO
0255  07344 020104        LDA LF
0256  07345 063217        JSM REALO
0257  07346 170402        RET           THIS IS REDUNDANT.
0258  07347 021776 RUN    LDA 1776B       GET STATUS WORD.
0259  07350 070406        RAR 9          POSITION RUN/PROG BIT.
0260  07351 072051        SLA *+1,S        FORCE TO PROGRAM MODE.
0261  07352 070306        RAR 7          RESTORE.
0262  07353 031776        STA 1776B
0263  07354 170402        RET
0264  00005         YADD  EQU 5B
0265  01701         PC    EQU 1701B
0266  00023         M5    EQU 23B
0267  07355 012013 SECUR  OCT 12013           LOCATION OF SECURITY WORD.
0268  07356 000046 END    OCT 46
0269  07357 000006 SIX    DEC 6
0270  00034         THREE EQU 34B
0271  07360 063341 OUTA   JSM FINE       RESTORE RUN MODE, OUTPUT CR/LF.
0272  07361 160017        JSM DISPY,I     AND GO TO DISPLAY ROUTINE, TO GOET
0273  07362 010022        CPA CNTKY      NEXT KEY, IF CONTINUE, RETURN TO
0274  07363 067254        JMP LISTM       TYPEWRITER LIST ROUTINE, ESSE
0275  07364 064340        JMP 340B       ABORT AND EXECUTE KEY.

              TYPEWRITER ALPHA PRINT BLOCK.

0277  07365 070742 FMT    SAR 16         CLEAR THE SHIFT CONDITION.
0278  07366 133475        STA SHIFT,I
0279  07367 063500 KEYIN  JSM GETKY       GET A KEY FOR PROCESSING.
0280  07370 070137        LDB A          CHECK FOR 7 BIT CODE.
0281  07371 074344        SBL 9
0282  07372 074213        SBP *+4         SKIP IF NOT 7 BIT CODE.
0283  07373 055777        DSZ 1777B        DSZ STACK PTR TWICE TO
0284  07374 055777        DSZ 1777B        GET TO BASE LEVEL.
0285  07375 064342        JMP 342B          GIVE TO SYSTEM, TAKING KEY FROM
0286*       INPUT BUFFER.
0287  07376 013472        CPA UPARO      UP ARROW IS SHIFT KEY.
0288  07377 067433        JMP SDIDL      DIDDLE THE SHIFT WORD.
0289  07400 013054        CPA FMTKY      CLOSING FORMAT STRING?
0290  07401 170402        RET            YES, RETURN TO KEYBD OR PRGM.
0291  07402 013473        CPA CLR        CLEAR KEY IS CR/LF,
0292  07403 067466        JMP RETLF      GIVE CR/LF, RESET SHIFT.
0293  07404 013474        CPA DNARO      CLEAR ALL TABS? YES IF
0294  07405 067440        JMP CRTAB      IN UPPER CASE (SHIFTED CONDITION).
0295  07406 010021        CPA 21B       IS THIS A STOP?
0296  07407 064161        JMP 161B         YES, QUIT IMMEDIATELY.
0297  07410 013356        CPA END        IS THIS AN END KEY.
0298  07411 064161        JMP 161B         YES, STOP AND POINT OUT THE TROUBLE.
0299  07412 010110        CPA PNTKY        IF THE PRINT KEY, PRINT THE X REGISTER
0300  07413 067470        JMP DOPRN        AND CONTINUE IN STRING.
0301  07414 031737 KOUT   STA TEMP       BEGIN NORMAL PROCESSING OF ALPHA KEY.
0302  07415 070002        SAR 1
0303  07416 070137        LDB A          COMPUTE OFFSET FOR LOOKUP.
0304  07417 070704        SAL 2
0305  07420 074017        ADA B
0306  07421 000034        ADA THREE      GETTING TO UNSHIFTED OFFSET.
0307  07422 103475        ADA SHIFT,I     SHIFT?  1IF YES, 0 IF NO.
0308  07423 003535        ADA BASE        GET TABLE ADDRESS FOR TYPE CODE.
0309  07424 025737        LDB TEMP
0310  07425 070016        EXA       LOAD A WITH CODE; B IS ORIGINAL KEY.
```

TYPEWRITER ALPHA PRINT BLOCK.—Continued

```
0311  07426 074111        SLB *+2
0312  07427 070404        SAL 8
0313  07430 070342        SAR 8
0314  07431 063217        JSM REALO        OUTPUT CODE,
0315  07432 067367        JMP KEYIN        AND GET ANOTHER KEY.
0316  07433 123475 SDIDL  LDA SHIFT,I      IF SHIFT=1, MAKE IT 0.
0317  07434 000031        ADA M1
0318  07435 070076        TCA               IF SHIFT=0, MAKE IF 1.
0319  07436 133475        STA SHIFT,I
0320  07437 067367        JMP KEYIN
0321  07440 127475 CRTAB  LDB SHIFT,I      IF UPPER SHIFT, THEN CLEAR ALL
0322  07441 076110        RZB *+2
0323  07442 067414        JMP KOUT         TABS, ELSE CLEAR FROM CURRENT POSITION.
0324  07443 070742        SAR 16
0325  07444 063217        JSM REALO        OUTPUT NOP TO TEST IF TYPE CONNECTED.
0326  07445 023476 TBM    LDA TAB          TAB TO RIGHT MARGIN.
0327  07446 063461        JSM PUTIT          OUTPUTS AND WAITS FOR FLAG.
0328  07447 172241        LIA 01          AT RIGHT MARGIN?
0329  07450 070604        SAL 4
0330*
0331*       IF A TAB HAS GONE PAST THE RIGHT MARGIN SETTING,
0332*       THEN NO END OF LINE SENSOR WILL COME UP, SO

0333  07451 172700        SFC 00          IT'S PASSIBLE TO LOCK UP. THUS,
0334  07452 064161        JMP 161B          CHECK HERE FOR STOP KEY TO END LOOP.
0335*
0336*
0337  07453 071513        SAP TBM          NO, TAB AGAIN.
0338  07454 023477        LDA CLRAL
0339  07455 063461        JSM PUTIT         OUTPUT NOT CHECKING FOR END OF LINE.
0340  07456 070742        SAR 16             OUTPUT NOP TO CLEAR LINE SENSOR.
0341  07457 063217        JSM REALO        STANDARD OUTPUT.
0342  07460 067365        JMP FMT          CONTINUE CHECKING FOR MORE KEYS.
0343  07461 172141 PUTIT  OTA 1            OUTPUT TO TYPEWRITER.
0344  07462 172601        STC 01
0345  07463 172501        SFS 1
0346  07464 067463        JMP *-1          WAIT.
0347  07465 170402        RET
0348  07466 063342 RETLF  JSM CRLF         CR, LF OUTPUT.
0349  07467 067365        JMP FMT          RESET SHIFT.

0351  07470 063063 DOPRN  JSM PNT          PRINT X REGISTER.
0352  07471 067365        JMP FMT
0353  07472 000027 UPARO  OCT 27
0354  07473 000020 CLR    OCT 20
0355  07474 000025 DNARO  OCT 25
0356  07475 012010 SHIFT  OCT 12010        HOLDS SHIFT KEY.
0357  07476 170040 TAB    OCT 170040           REALLU A SPACE BECAUSE TYPEWRITER
0358*CANNOT GIVE END OF LINE ON A TAB.
0359  07477 170013 CLRAL  OCT 170013       CLEAR ALL TABS.
```

KEY FETCH ALLOWING STEP PROGRAM.

```
0361  07500 172700 GETKY  SFC 00       SKIP IF STOP NOT PRESSED.
0362  07501 067517        JMP DISP       STOP KEY PRESSED, TREAT AS STEP PROGRAM.
0363  07502 025775        LDB INBUF      CHECK FOR STEP PROGRAM.
0364  07503 014300        CPB 300B        STOP PROGRAM?
0365  07504 067510        JMP FLGCK       YES, CHECK THE FLAG.
0366  07505 074742        SBR 16
0367  07506 035720        STB 1720B      CLEAR THE STEP PROGRAM FLAG,
0368  07507 064115        JMP 115B       AND GO TO THE REAL GETKY ROUTINE.
0369  07510 025720 FLGCK  LDB 1720B
0370  07511 076310        RZB DISP      IF FLAG SET, WAIT FOR NEXT KEY.
0371  07512 121701 MEMGT  LDA PC,I      GET KEY FROM MEMORY.
0372  07513 045701        ISZ PC
0373  07514 024031        LDB M1        SET STEP PROGRAM FLAG.
0374  07515 035720        STB 1720B
0375  07516 170402        RET
```

KEY FETCH ALLOWING STEP PROGRAM—Continued

```
0376  07517 070742 DISP  SAR 16        CLEAR FLAG.
0377  07520 031720       STA 1720B
0378  07521 160017       JSM DISPY,I        JMP TO DISPLAY ROUTNE.
0379  07522 010300       CPA 300B      STEP PROGRAM?
0380  07523 067512       JMP MEMGT
0381  07524 010022       CPA CNTKY      IS THIS A CONTINUE?
0382  07525 067532       JMP CONTU       YES, TREAT AS CONTINUE.
0383  07526 010021       CPA 21B        IS THIS A STOP KEY?
0384  07527 067521       JMP DISP+2       YES, IGNORE IT.
0385  07530 055777       DSZ 1777B
0386  07531 064340       JMP 340B        GIVE TO MONITOR.
0387  07532 072053 CONTU  SAP *+1,S      SET INPUT BUFFER TO NEGATIVE, IN
0388  07533 031775       STA 1775B      PREPARATION FOR PROGRAM EXECUTION.
0389  07534 064115       JMP 115B        GET THE NEXT KEY.
0390  00022        CNTKY EQU 22B         OCT 47, CODE FOR CONTINUE.
0391  01775        INBUF EQU 1775B
0392  00017        DISPY EQU 17B
```

TYPEWRITER CONVERSION TABLE.

```
0394*TYPEWRITER CODE CONVERSION TABLE.
0395*
0396*THERE ARE TWO KEYS ASSOCIATED WITH EACH FIVE WORDS OF TABLE.
0397*ODD NUMBERED KEYCODES USE THE RIGHT 8 BITS OF EACH WORD,
0398*AND EVEN NUMBERED KEYCODES USE THE LEFT 8 BITS. THE FIRST
0399*THREE ENTRIES GIVE THE NMEMONIC CHARACTERS TO USE IN A LISTING.
0400*THE NEXT ENTRY IS THE UNSHIFTED TYPEWRITER CHARACTER DURING
0401*AN ALPHA TYPE OPERATION, AND THE 5TH ENTRY IS THE SHIFTED
0402*TYPEWRITER CHARACTER.
0403*
0404  07535 023536 BASE  LDA *+1          ADDRESS INDEPENDENT TABLE FETCH.
0405  07536 020040       ASC 4,  01  01
      07537 030061
      07540 020040
      07541 030061
0406  07542 057441       OCT 57441
0407  07543 020040       ASC 5,  23  23"#
      07544 031063
      07545 020040
      07546 031063
      07547 021043
0408  07550 020040       ASC 5,  45  45$%
      07551 032065
      07552 020040
      07553 032065
      07554 022045
0409  07555 020040       ASC 5,  67  67&'
      07556 033067
      07557 020040
      07560 033067
      07561 023047
0410  07562 020040       ASC 5,  89  89()
      07563 034071
      07564 020040
      07565 034071
      07566 024051
0411  07567 054040       ASC 1,X
0412  07570 051541       OCT 51541
0413  07571 050440       ASC 1,Q
0414  07572 055122       ASC 1,ZR
0415  07573 075162       OCT 75162          END OF X**2, LOWER CASE A.
0416  07574 020040       ASC 1,
0417  07575 061107       OCT 61107,20040,50507,70547      END OF LOWER B, G ENTRY.
      07576 020040
      07577 050507
      07600 070547
0418  07601 020061       ASC 4, 1F/ XFU
      07602 043057
      07603 020130
      07604 043125
0419  07605 063165       OCT 63165
0420  07606 041440       ASC 5,C L.R  . >            CLR,. ENTRY.
```

TYPEWRITER CONVERSION TABLE—Continued

```
      07607 046056
      07610 051040
      07611 020056
      07612 020076


0421  07613 051130        ASC 4,RXUTPO@T
      07614 052524
      07615 050117
      07616 040124
0422  07617 060164        OCT 60164
0423  07620 054504        ASC 3,YDEN
      07621 042516
      07622 020040
0424  07623 054014        OCT 54014,74001      TAB CLEAR, SET,ETC.
      07624 074001
0425  07625 042525        ASC 3,EUEPX
      07626 042520
      07627 054040
0426  07630 077040        OCT 77040,57040       GETS UP AND RIGHT ARROW.
      07631 057040
0427  07632 054111        ASC 3,XIENYD
      07633 042516
      07634 054504
0428  07635 004527        OCT 4527,567
      07636 000567
0429  07637 041440        ASC 3,C H+S
      07640 044053
      07641 051440
0430  07642 076453        OCT 76453,56473
      07643 056473
0431  07644 020104        ASC 5, D-I V-/=?
      07645 026511
      07646 020126
      07647 026457
      07650 036477
0432  07651 020103        ASC 5, C*L X*,:<
      07652 025114
      07653 020130
      07654 025054
      07655 035074
0433  07656 054523        ASC 4,YSTTOPS
      07657 052124
      07660 047520
      07661 051440
0434  07662 071400        OCT 71400
0435  07663 043111        ASC 5,FIMFTG ? ?
      07664 046506
      07665 052107
      07666 020077
      07667 020077
0436  07670 043520        ASC 4,GPTNOT#              GOTO, PRINT.
      07671 052116
      07672 047524
      07673 021440
0437  07674 004040        OCT 4040              BACKSPACE ON UPPER CASE GOTO.
0438  07675 042503        ASC 5,ECNNDT
      07676 047116
      07677 042124
      07700 020040
      07701 020040
0439  07702 054114        ASC 5,XL=RYL%$$$


      07703 036502
      07704 054514
      07705 022444
      07706 022444
      07707 054130        ASC 5,XX<>YY("("
```

TYPEWRITER CONVERSION TABLE—Continued

```
      07710 036076
      07711 054531
      07712 024042
      07713 024042
0441  07714 051440        ASC 4,S FKL =K                SET FLAG ,K
      07715 043113
      07716 046040
      07717 036513
0442  07720 036553        OCT 36553                 END OF SET FLAG, K.
0443  07721 050120        ASC 3,PPIS E
      07722 044523
      07723 020105
0444  07724 050051        OCT 50051,70051
      07725 070051
0445*END OF PI, PAUSE.
0446  07726 020040        ASC 4,  EC  EC
      07727 042503
      07730 020040
      07731 042503
0447  07732 062543        OCT 62543
0448  07733 020040        ASC 4,  AD  AD
      07734 040504
      07735 020040
      07736 040504
0449  07737 060544        OCT 60544          END OF A,D.
0450  07740 044440        ASC 4,I NIT VI
      07741 047111
      07742 052040
      07743 053111
0451  07744 073151        OCT 73151
0452  07745 020130        ASC 4, XBF RBY
      07746 041106
      07747 020122
      07750 041131
0453  07751 061171        OCT 61171              END OF B,XFR.
0454  07752 020040        ASC 4,  MO  MO
      07753 046517
      07754 020040
      07755 046517
0455  07756 066557        OCT 66557
0456  07757 020040        ASC 4,  LN  LN
      07760 046116
      07761 020040
      07762 046116
0457  07763 066156        OCT 66156
0458  07764 020040        ASC 4,  HJ  HJ
      07765 044112
      07766 020040
      07767 044112
0459  07770 064152        OCT 64152
0460  07771 051523        ASC 3,SSQ/TR           SQT X, SUB/RETURN.



      07772 050457
      07773 052122
0461  07774 075407        OCT 75407,55406
      07775 055406
0462                      END
*  NO ERRORS*
```

—Continued

ASMB,A,B,L,T

```
MOVE   002005
DONE   002014
SRCH   002017
PRTCR  002025
FINIS  002041
FI     002076
FI1    002100
FI2    002103
NOTF   002113
FWND   002116
FIE    002145
FRET   002146
FRET1  002150
FRET0  002155
FRET2  002161
FRET3  002170
FRET4  002176
STCHK  002203
CLEAN  002211
MLOW   002214
MHI    002215
```

—Continued

```
RST4    002216
RST1    002217
RST2    002224
RST3    002227
RSTC    002230
FIND    002242
GETK1   002256
UGLY    002267
DLET    002270
DELT8   002276
DELT0   002321
DELT1   002327
DELT2   002332
DELT3   002334
DELT4   002355
BAD     002360
GOOD    002366
DELT5   002400
DELT7   002417
DELT6   002421
INSRT   002424
INSL    002451
LAST    002462
ACUM    002466
ACUM1   002470
ACUM2   002503
ACUM3   002515
ACUM4   002516
ACUM5   002523
DLMSK   002525
M60     002526
ERR1    002527
M76     002530
DLMOF   002531

MDEC    002532
DLTK    002533
CANCK   002534
INSTK   002535
INTK    002536
FINDK   002537
```

—Continued

```
ENDK    002540
XFRK    002541
CONTK   000022
.3      000034
SSP     001777
ENM     002064
DTBL1   002067
ERROR   002072
PROF    002042
FSL2    002043
PRMSK   002044
FSOR    002045
FSLL    002045
FSHL    002046
FNCB    002047
DEFK    002050
PRTCT   002051
FRETK   002052
FOFM    002053
M2      002054
AUSO    000047
KBUF    001725
TEMP1   001724
MAX     000055
PCNT    001701
PMODE   001776
INIT    000167
FSTP    001665
EXCF    000103
USLO    001707
CONT    000132
GETK    000115
KLOG    000113
P2      000772
M1      000031
M10     000024
TEMP2   001664
ADDRC   001721
INTR2   000357
SECW    000111
*  NO ERRORS*
```

USER DEFINABLE FUNCTION BLOCK

```
0001                    ASMB,A,B,L,T
0003  02000                     ORG 2000B
0004*LOCAL MONITOR
0005*
0006  02000 025725          LDB KBUF        LOAD THE INPUT KEY
0007  02001 006067          ADB DTBL1       FIND THE TABLE ENTRY
0008  02002 035720          STB 1720B       RESET THE CHANGE SIGN FLAG
0009  02003 074537          LDB B,I         FIND THE ROUTINE'S ADDRESS
0010  02004 074737          JMP B,I         JUMP TO THE ROUTINE


0012*MOVE ROUTINE
0013* AT ENTRY TO THIS SUBROUTINE, A&B REG MUST CONTAIN THE
0014*STARTING AND ENDING ADDRESS OF THE AREA TO BE MOVED.
0015*
0016  02005 170004 MOVE    XFR             MOVE 4 WORDS
0017  02006 035724         STB TEMP1       STORE B TEMPORARILY
0018  02007 004055         ADB MAX         ADD MAX ADDRESS TO B
0019  02010 074213         SBP DONE        SKIP ON ADDRESS OVERFLOW
0020  02011 025724         LDB TEMP1       RESTORE B
0021  02012 070070         SIA *+1         INCREMENT A
0022  02013 077530         RIB MOVE        INCREMENT B,AND PERFORM NEXT TRANSFER
0023  02014 170402 DONE    RET


0025*LINKAGE AREA
0026  02015 000000         BSS 2           ENTRIES FOR KEYCODES 15 AND 16
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0028*SEARCH ROUTINE
0029* AT THE ENTRY TO THIS SUBROUTINE AREG MUST CONTAIN THE
0030* KEYCODE SEARCHED FOR. WHEN RETURNING, THE SUBROUTINE WILL
0031* PUT IN B THE ADDRESS OF THE KEYCODE IF FOUND, OTHERWISE
0032* B WILL BE SET TO A NEGATIVE VALUE.
0033*
0034  02017 074457 SRCH   CPA B,I       KEY MATCHES CURRENT ONE IN MEMORY
0035  02020 170402        RET             YES, RETURN
0036  02021 016064        CPB ENM         MEMORY EXHAUSTED?
0037  02022 074742        SBR 16          YES,CLEAR B
0038  02023 077630        RIB SRCH     INCREMENT B AND TEST NEXT LOC IF POSSIBLE
0039  02024 077613        SBP SRCH+1,S   MAKE B NEGATIVE AND RESTURN


0041*PROTECT KEY PROCESSING
0042*
0043  02025 120103 PRTCP  LDA EXCF,I     LOAD THE EXECUTE FLAG
0044  02026 070110        SZA *+2         SKIP IF NOT EXECUTING A PROG
0045  02027 166072        JMP ERROR,I     ERROR,STOP EXECUTION
0046  02030 120111        LDA SECW,I      LOAD THE SECURITY WORD
0047  02031 072410        RZA FINIS        RETURN IF SECURE PROGRAM
0048  02032 021776        LDA PMODE        LOAD THE STATUS WORD
0049  02033 052044        AND PRMSK        ISOLATE THE PROTECT BIT
0050  02034 072250        RZA FINIS        RETURN IF ALREADY PROTECTED
0051  02035 022044        LDA PRMSK       LOAD THE PROTECT MASK
0052  02036 060130        JSM 130B        TURN ON THE PROTECT BIT AND LIGHT
0053  02037 025701        LDB PCNT         LOAD THE NEW USER ORIGIN



0054  02040 035707        STB USLO         STORE IN POINTER
0055  02041 064167 FINIS  JMP INIT         RETURN


0057*LINKAGE AREA
0058  02055               ORG 2055B
0059  02055 000000        BSS 1            LINKAGE FOR KEYCODE 55
0060*
0061  02060               ORG 2060B        LINKAGE FOR KEYCODES 60 THRU 75
0062  02060 000000        BSS 14           LINKAGES FOR KEYCODES 60 THRU 75


0064*PROCESSING OF FUNCTION CALL KEYS (FI)
0065*
0066  02076 062203 FI     JSM STCHK        CHECK THE FUNCTION STACK
0067  02077 021665        LDA FSTP         LOAD THE FUNCTION STACK POINTER
0068  02100 012046 FI1    CPA FSHL         IS STACK FULL?
0069  02101 166072        JMP ERROR,I   ERROR,STIP EXECUTION
0070  02102 024047        LDB AUSO         LOAD THE ABSOLUTE USER ORIGIN
0071  02103 022050 FI2    LDA DEFK      LOAD CODE FOR DEF KEY
0072  02104 062017        JSM SRCH      SEARCH FOR A DEF
0073  02105 074312        SBM NOTF         SKIP IF NON FOUND
0074  02106 074070        SIB *+1          POINT TO KEY FOLLOWING DEF
0075  02107 074517        LDA B,I          LOAD THE KEY
0076  02110 011725        CPA KBUF         IS IT THIS FI?
0077  02111 066116        JMP FWND         YES, SEARCH SUCCEEDED
0078  02112 066103        JMP FI2          SERCH SOME MORE
0079  02113 055701 NOTF   DSZ PCNT      DECREMENT THE PROG COUNTER
0080  02114 060167        JSM INIT      RE-INITIALIZE
0081  02115 166072        JMP ERROR,I   STOP EXECUTION

0083  02116 035724 FWND   STB TEMP1      STORE B TEMPORARILTY
0084  02117 021701        LDA PCNT       LOAD THE VALUE OF THE PROG COUNTER
0085  02120 124103        LDB EXCF,I      LOAD THE EXECUTE FLAG
0086  02121 076110        RZB *+2         SKIP IF EXECUTING A PROGRAM
0087  02122 072052        SAM *+1,S       INDICATE FI CAME FROM KEYBOARD
0088  02123 131665        STA FSTP,I      STORE THE RETURN ADDRESS
0089  02124 045665        ISZ FSTP        INCREMENT THE FUNCTION STACK POINTER
0090  02125 021776        LDA PMODE       LOAD THE STATUS WORD
0091  02126 042047        IOR FNCB        TURN ON THE FUNCTION BIT
0092  02127 172160        OTA 16          OUTPUT TO THE INDICATORS
0093  02130 031776        STA PMODE       STORE THE NEW STATUS
0094  02131 021707        LDA USLO      LOAD THE CURRENT LOCAL ORIGIN
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0095  02132 131665         STA FSTP,I     STORE IN FUNCTION STACK
0096  02133 045665         ISZ FSTP
0097  02134 025724         LDB TEMP1     RESTORE B
0098  02135 004031         ADB M1        POINT TO DEF
0099  02136 035707         STB USLO      CHANGE THE LOCAL ORIGIN
0100  02137 004772         ADB P2        POINT TO FIRST EXECUTABLE KEY
0101  02140 035701         STB PCNT      CHANGE THE PROG COUNTER
0102  02141 120103         LDA EXCF,I    LOAD THE EXECUTION FLAG
0103  02142 072150         RZA FIE       SKIP IF EXECUTING A PROG
0104  02143 060167         JSM INIT      RE-INITIALIZE
0105  02144 064132         JMP CONT      JUMP TO CONTINUE ENTRY
0106  02145 064167 FIE     JMP INIT      RETUTN



0108*F-RET PROCESSING
0109*
0110  02146 062203 FRET    JSM STCHK     CHECK THE FUNCTION STACK
0111  02147 021665         LDA FSTP      LOAD THE FUNCTION STACK POINTER
0112  02150 012045 FRET1   CPA FSLL      IS FUNCTION STACK EMPTY?
0113  02151 166072         JMP ERROR,I   ERROR,STOP EXECUTION
0114  02152 124103         LDB EXCF,I    LOAD THE EXECUTION FLAG
0115  02153 076310         RZB FRET2     SKIP IF EXECUTING A PROG
0116  02154 025665         LDB FSTP      LOAD THE FUNCTION STACK POINTER
0117  02155 006054 FRET0   ADB M2        SUBTRACT 2 FROM IT
0118  02156 074517         LDA B,I       LOAD THE RETURN ADDRESS
0119  02157 072052         SAM *+1,S     MAKE IT NEGATIVE
0120  02160 074557         STA B,I       STORE IT BACK
0121  02161 055665 FRET2   DSZ FSTP      DECREMENT THE FUNCTION STACK POINTER
0122  02162 121665         LDA FSTP,I    LOAD THE LOCAL ORIGIN VALUE
0123  02163 031707         STA USLO      STORE AS USER ORIGIN
0124  02164 055665         DSZ FSTP      POINT TO THE RETURN ADDRESS
0125  02165 021665         LDA FSTP      LOAD THE POINTER VALUE
0126  02166 012045         CPA FSLL      IS STACK EMPTY?
0127  02167 062176         JSM FRET4   YES,TURN OFF THE FUNCTION BIT
0128  02170 121665 FRET3   LDA FSTP,I    LOAD THE RETURN ADDRESS
0129  02171 070172         SAM *+3,C     SKIP IF RETURN TO CALC MODE
0130  02172 031701         STA PCNT      RESET THE PROG COUNTER
0131  02173 064167         JMP INIT      RETUTN
0132  02174 031701         STA PCNT      RESTORE THE PROG COUNTER
0133  02175 064161         JMP 161B      STOP EXECUTION

0135  02176 021776 FRET4   LDA PMODE     LOAD THE STATUS WORD
0136  02177 052053         AND FOFM      TURN OFF THE FUNCTION BIT
0137  02200 031776         STA PMODE     STORE THE NEW STATUS
0138  02201 172160         OTA 16        OUTPUT TO THE INDICATORS
0139  02202 170402         RET         RETURN
0140*FUNCTION STACK CHECKER
0141*
0142  02203 021665 STCHK   LDA FSTP      LOAD THE FUNCTION STACK POINTER
0143  02204 002214         ADA MLOW      CHECK IF LOWER THAN LEGAL
0144  02205 070212         SAM CLEAN     RE-INITIALIZE IT IF LOWER
0145  02206 021665         LDA FSTP      LOAD THE POINTER VALUE AGAIN
0146  02207 002215         ADA MHI       CHECK IF HIGHER THAN LEGAL
0147  02210 070152         SAM CLEAN+2   SKIP IF WITHIN RANGE
0148  02211 022045 CLEAN   LDA FSOR      LOAD THE FUNCTION STACK ORIGIN
0149  02212 031665         STA FSTP      INITIALIZE THE POINTER
0150  02213 170402         RET

0152  02214 176112 MLOW    OCT 176112
0153  02215 176077 MHI     OCT 176077



0156  02216 066516 RST4    JMP ACUM4      TURN OFF THE DELETE LIGHT AND RETURN
0157  02217 120111 RST1    LDA SECW,I     LOAD THE SECURITY WORD
0158  02220 073710         RZA RST4        RETURN IF A SECURE PROG
0159  02221 062230         JSM RSTC      EMPTY THE FUNCTION STACK IF NEEDED
0160  02222 024047         LDB AUSO      LOAD THE ABSOLUTE ORIGIN
0161  02223 035707         STB USLO      MAKE IT CURRENT ORIGIN
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0152  02224 021776 RST2   LDA PMODE       LOAD THE STATUS WORD
0153  02225 052042        AND PROF      TURN OFF THE PROTECT BIT
0154  02226 066517        JMP ACUM4+1     TURN OFF THE DELETE LIGHT

0156  02227 062516 RST3   JSM ACUM4       TURN OFF THE DELETE LIGHT
0157  02230 062203 RSTC   JSM STCHK      CHECK THE FUNCTION STACK
0158  02231 021665        LDA FSTP      LOAD THE FUNCTION STACK POINTER
0159  02232 012045        CPA FSLL      IS STACK EMPTY?
0160  02233 170402        RET             RETURN
0161  02234 026043        LDB FSL2      REDUCE THE STACK LEVEL TO ONE
0162  02235 035665        STB FSTP      STORE IN STACK POINTER
0163  02236 122045        LDA FSLL,I    LOAD THE 1ST RETURN ADDRESS
0164  02237 070072        SAM *+1,C     CLEAR CALC FLAG IF ANY
0165  02240 132045        STA FSLL,I
0166  02241 066161        JMP FRET2


0168*FIND KEY PROCESSING
0169*
0170  02242 062256 FIND   JSM GETK1       PICK THE NEXT KEY
0171  02243 025701        LDB PCNT      LOAD THE PROG COUNTER VALUE
0172  02244 062017        JSM SRCH      SEARCH FOR THE KEYCODE
0173  02245 074113        SBP *+2       SKIP IF FOUND
0174  02246 166072        JMP ERROR,I   INDICATE ERROR, NOT FOUNF
0175  02247 074070        SIB *+1       PUT KEY IN Z DISPLAY
0176  02250 035701        STB PCNT      ADJUST THE PROF COUNTER
0177  02251 124111        LDB SECW,I    LOAD THE SECURITY WORD
0178  02252 074110        SZB *+2       SSKIP IF NON SECURE PROG
0179  02253 064167        JMP INIT      RETURN IF SECURE
0180  02254 060167        JSM INIT      RE-INITIALIZE
0181  02255 064326        JMP 3268      SWITCH TO PROG MODE


0183*THE FOLLOWING SUBROUTINE FETCHES THE NEXT KEY. THIS
0184* ROUITNE ABORTS THE OPERATION IF KEY CAME FROM PROGRAM
0185* STORAGE.
0186*
0187  02256 124103 GETK1  LDB EXCF,I    LOAD THE EXECUTION FLAG
0188  02257 076410        RZB UGLY      SKIP IF EXECUTING A PROG
0189  02260 060115        JSM GETK      CALL ON THE KEY FETCHER
0190  02261 124103        LDB EXCF,I     LOAD THE EXECUTION FLAG
0191  02262 076250        RZB UGLY       SKIP IF KEY CAME FROM MEMORY
0192  02263 070137        LDB A          PUT KEY IN B ALSO
0193  02264 160113        JSM KLOG,I     PRINT IT IF NEEDED
0194  02265 074117        LDA B          LOAD KEY BACK IN A
0195  02266 170402        RET            RETURN
0196  02267 166072 UGLY   JMP ERROR,I    STOP THE OPERATION


0198*DELETE KEY PROCESSING
0199*
0200  02270 120103 DLET   LDA EXCF,I    LOAD THE EXECUTION FLAG
0201  02271 073710        RZA UGLY      SKIP IF EXECUTING A PROG
0202  02272 021776        LDA PMODE     LOAD THE STATUS WORD
0203  02273 042525        IOR DLMSK      TURN ON THE DELETE LIGHT
0204  02274 031776        STA PMODE      STORE THE NEW STATUS




0205  02275 172160        OTA 16          OUTPUT TO THE INDICATORS
0206  02276 062256 DELT8  JSM GETK1     GET THE NEXT KEY
0207  02277 012534        CPA CANCK     CANCEL KEY FOLLOWS?
0208  02300 066516        JMP ACUM4      YES,RETURN
0209  02301 012051        CPA PRTCT       DELETE-PROTECT?
0210  02302 066217        JMP RST1        YES
0211  02303 012052        CPA FRETK       DELETE-FRET?
0212  02304 066227        JMP RST3          YES
0213  02305 012533        CPA DLTK        DELETE-DELETE?
0214  02306 066276        JMP DELT8       GET THE NEXT KEY
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0225  02307 004024        ADB M10      CHECK IF KEY IS NUMERIC
0226  02310 074753        SBP DELT1    SKIP IF NOT
0227  02311 031664        STA TEMP2    STORE THE GIVEN DIGIT
0228  02312 062230        JSM RSTC     RESET THE FUNCTION STACK IF NEEDED
0229  02313 062466        JSM ACUM     CALCULATE THE REFERENCED ADDRESS
0230  02314 025664        LDB TEMP2
0231  02315 035701        STB PCNT     POINT TO THE EMPTIED AREA
0232  02316 045664        ISZ TEMP2    INCREMENT IT
0233  02317 021664        LDA TEMP2
0234  02320 062005        JSM MOVE     MOVE MEMORY
0235  02321 062516 DELT0  JSM ACUM4    TURN OFF THE DELETE LIGHT
0236  02322 021721        LDA ADDRC    LOAD THE ADDRESS COUNTER
0237  02323 072110        RZA *+2      SKIP IF ADDRESS WAS TERMINATED AUTOMATICA
0238  02324 170402        RET          RETURN
0239  02325 025725        LDB KBUF     LOAD THE TERMIAATING KEY
0240  02326 064357        JMP INTR2    GIVE IT TO THE INTERPRETER
0241  02327 070137 DELT1  LDB A        PUT KEY IN B
0242  02330 006526        ADB M60      KEY LESS THAN 60?
0243  02331 074153        SBP DELT3    SKIP IF >=60, LEGAL
0244  02332 162527 DELT2  JSM ERR1,I   TURN ON THE ERROR LIGHT
0245  02333 066516        JMP ACUM4    ABORT THE DELETE OPERATIONS
0246*
0247  02334 070137 DELT3  LDB A        PUT KEY BAK IN B
0248  02335 006530        ADB M76      CHECK IF KEYCODE >76
0249  02336 075613        SBP DELT2    ILLEGAL IF >= 76
0250  02337 012533        CPA DLTK     KEY IS DELETE?
0251  02340 066270        JMP DLET     RE-INITIALIZE THE DELETE OPERATION
0252  02341 012536        CPA INTK     INTX KEY FOLLOWS?
0253  02342 066332        JMP DELT2    ERROR,ILLEGAL
0254  02343 012050        CPA DEFK     KEY IS DEFINE?
0255  02344 066332        JMP DELT2    ERROR,ILLEGAL
0256  02345 012537        CPA FINDK    KEY IS DEFINE?
0257  02346 066332        JMP DELT2    ERROR, ILLEGAL
0258  02347 012541        CPA XFRK     X FROM KEY?
0259  02350 066332        JMP DELT2    ERROR,ILLEGAL
0260  02351 031725        STA KBUF     STORE KEY IN BUFFFER
0261  02352 062230        JSM RSTC     RESET THE FUNCTION STACK IF NEEDED
0262  02353 021725        LDA KBUF     LOAD THE INPUT KEY
0263  02354 025707        LDB USL0     LOAD THE LOCAL PROG ORIGIN
0264  02355 022050 DELT4  LDA DEFK     LOAD CODE FOR DEFINE KEY
0265  02356 062017        JSM SRCH     SEARCH FOR DEF
0266  02357 074113        SBP *+2      SKIP IF SEARCH SUCCEEDED
0267  02360 066332 BAD    JMP DELT2    ERROR,RETURN
0268  02361 074070        SIB *+1      POINT TO KEY FOLLOWING DEF
0269  02362 021725        LDA KBUF     LOAD THE F KEY SEARCHED FOR
0270  02363 074457        CPA B,I      IS THIS THE FUNCTION TO BE DELETED?



0271  02364 066366        JMP GOOD     YES, GO AHEAD
0272  02365 066355        JMP DELT4    NO, SEARCH SOME MORE
0273  02366 004031 GOOD   ADB M1       POINT TO THE FUNCTION BEGINNING
0274  02367 035724        STB TEMP1    STORE ADDRESS TEMPORARILY
0275  02370 022050        LDA DEFK     LOAD CODE FOR DEF
0276  02371 074070        SIB *+1      POINT PASS THE DEF
0277  02372 062017        JSM SRCH     SEARCH FOR NEXT DEF
0278  02373 074253        SBP DELT5    SKIP IF FOUND
0279  02374 022540        LDA ENDK     LOAD CODE FOR END
0280  02375 025724        LDB TEMP1    LOAD SEARCH ADDRESS
0281  02376 062017        JSM SRCH     SEARCH FOR END INSTEAD
0282  02377 075052        SBM BAD      ERROR IF NOT FOUND
0283  02400 074117 DELT5  LDA B
0284  02401 025701        LDB PCNT     LOAD THE PROG COUNTER
0285  02402 074076        TCB          MAKE IT NEGATIVE
0286  02403 005724        ADB TEMP1    SUBTRACT FROM START OF DELETE ARESA
0287  02404 074653        SBP DELT6    SKIP IF COUNTER IS LESS
0288  02405 025701        LDB PCNT
0289  02406 074076        TCB
0290  02407 070037        ADB A        SUBTRACT FROM END OF DELETE AREA
0291  02410 074353        SBP DELT7    SKIP IF COUNTER IS SITTING IN BETWWEEN
0292  02411 025724        LDB TEMP1    FIND THE LENGHT OF AREA TO BE MOVED
0293  02412 074076        TCB
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0294  02413 070037         ADB A
0295  02414 074076         TCB              NEG OF LENGTH IS AVAILABLE HERE
0296  02415 005701         ADB PCNT         ADJUST THE PROG COUNTER
0297  02416 066420         JMP *+2
0298  02417 025724 DELT7   LDB TEMP1        LOAD START OF AREA TO BE DELETED
0299  02420 035701         STB PCNT         ADJUST THE PROG COUNTER
0300  02421 025724 DELT6   LDB TEMP1
0301  02422 062005         JSM MOVE          MOVE IT
0302  02423 066516         JMP ACUM4         TERMINATE


0304*INSERT PROCESSING
0305*
0306  02424 120103 INSRT   LDA EXCF,I        LOAD THE EXECUTE FLAG
0307  02425 070110         SZA *+2           SKIP IF IN CALCULATOR MODE
0308  02426 166072         JMP ERROR,I       ERROR,STOP EXECUTION
0309  02427 021776         LDA PMODE        LOAD THE STATUS WORD
0310  02430 042525         IOR DLMSK        TURN ON THE INSERT/DELETE BIT
0311  02431 031776         STA PMODE        STORE THE NEW STATUS
0312  02432 172160         OTA 16           OUTPUT TO THE INDICATORS
0313  02433 062256         JSM GETK1        GET THE NEXT KEY
0314  02434 004024         ADB M10           CHECK IF NUMERIC
0315  02435 074352         SBM *+7          SKIP IF IT IS
0316  02436 012534         CPA CANCK        CANCEL KEY FOLLOWS?
0317  02437 066516         JMP ACUM4        YES,RETURN
0318  02440 012535         CPA INSTK        INSERT KEY?
0319  02441 066424         JMP INSRT        REPEAT THE INSERT LOOP
0320  02442 062516         JSM ACUM4        TURN OFF THE DELETE BIT
0321  02443 166072         JMP ERROR,I       ERROR
0322  02444 031664         STA TEMP2         STORE THE INPUT DIGIT
0323  02445 062230         JSM RSTC          RESET THE FUNCTION STACK IF NEEDED
0324  02446 062466         JSM ACUM          CALCULATE THE REFERENCED ADDRESS
0325  02447 022064         LDA ENM           POINT TO END OF MEMORY



0326  02450 031724         STA TEMP1
0327  02451 011664 INSL    CPA TEMP2
0328  02452 066516         JMP ACUM4        RETURN
0329  02453 000031         ADA M1
0330  02454 170537         LDB A,I           LOAD KEY FROM THAT ADDRESS
0331  02455 135724         STB TEMP1,I       MOVE IT
0332  02456 011664         CPA TEMP2         DONE?
0333  02457 066462         JMP LAST          YES,PERFORM LAST LOOP
0334  02460 055724         DSZ TEMP1         DECREMENT MEMORY POINTER
0335  02461 066451         JMP INSL          GO BACK TO MOVE LOOP
0336  02462 024022 LAST    LDB CONTK
0337  02463 070577         STB A,I           PUT IN VACATED LOCATION
0338  02464 031701         STA PCNT         POINT TO THE INSERTED KEY
0339  02465 066321         JMP DELT0        RETURN


0341*ADDRESS COMPUTING SUBROUTINE
0342*
0343  02466 024034 ACUM    LDB .3            INITIALIZE AN ADDRESS COUNTER
0344  02467 035721         STB ADDRC
0345  02470 062256 ACUM1   JSM GETK1         GET THE NEXT KEY
0346  02471 004024         ADB M10           NUMERIC?
0347  02472 074453         SBP ACUM2         SKIP IF NOT
0348*
0349  02473 025664         LDB TEMP2        LOAD THE PREVIOUS RESULT
0350  02474 074704         SBL 2            4*B
0351  02475 005664         ADB TEMP2        5*B
0352  02476 074037         ADB B            10*B
0353  02477 070037         ADB A            10*B+A
0354  02500 035664         STB TEMP2
0355*
0356  02501 055721         DSZ ADDRC         4 DIGITS ADDRESS IN?
0357  02502 066470         JMP ACUM1         NO,GET THE NEXT DIGIT
0358  02503 031725 ACUM2   STA KBUF          STORE THE TERMINATOR KEY
0359  02504 012534         CPA CANCK         KEY IS CANCEL?
0360  02505 066523         JMP ACUM5        YES,ABORT THE INSERT OR DELETE OPERATION
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0361  02506 025664          LDB TEMP2     LOAD THE COMPUTED RESULT
0362  02507 005707          ADB USLO        FIND THE ACTUAL ADDRESS
0363  02510 035664          STB TEMP2       STORE IT
0364  02511 004055          ADB MAX         CHECK IF ADDRESS OUT OF RANGE
0365  02512 074152          SBM ACUM3       SKIP IF OK
0366  02513 062516          JSM ACUM4       TURN OFF THE DELETE LIGHT
0367  02514 166072          JMP ERROR,I     TURN ON THE ERROR LIGHT
0368  02515 170402 ACUM3    RET             NORMAL RETURN
0369*
0370  02516 021776 ACUM4    LDA PMODE       LOAD THE STATUS WORD
0371  02517 052531          AND DLMOF       TURN OFF THE DELETE BIT
0372  02520 031776          STA PMODE       STORE THE NEW STATUS
0373  02521 172160          OTA 16          OUTPUT TO THE INDICATORS
0374  02522 064167          JMP INIT        RETURN
0375  02523 055777 ACUM5    DSZ SSP       DECREMENT THE SYSTEM STACK
0376  02524 066516          JMP ACUM4     RETURN


0378*LINKAGE AREA
0379  02525 020000 DLMSK    OCT 20000



0380  02526 177720 M60      OCT 177720
0381  02527 004750 ERR1     OCT 4750
0382  02530 177702 M76      OCT 177702
0383  02531 157777 DLMOF    OCT 157777
0384  02532 004176 MDEC     OCT 4176
0385  02533 000062 DLTK     OCT 62
0386  02534 000020 CANCK    OCT 20          CANCEL INSERT OR DELETE (CLEAR)
0387  02535 000066 INSTK    OCT 66          CODE FOR INSERT
0388  02536 000064 INTK     OCT 64
0389  02537 000015 FINDK    OCT 15
0390  02540 000046 ENDK     OCT 46
0391  02541 000067 XFRK     OCT 67
0392  00022        CONTK    EQU 22B
0393  00034        .3       EQU 34B
0394  01777        SSP      EQU 1777B


0396  02060                 ORG 2060B
0397  02060 002076          DEF FI        F5 ENTRY
0398  02061 002076          DEF FI        F3 ENTRY
0399  02062 002270          DEF DLET       DELETE ENTRY
0400  02063 002076          DEF FI        F4 ENTRY
0401  02064 015777 ENM      OCT 15777     END MEMORY ADDRESS
0402  02065 002076          DEF FI
0403  02066 002424          DEF INSRT      INSERT ENTRY
0404  02067 002000 DTBL1    OCT 2000      ADDRESS OF LOCAL DIRECTIVE TABLE
0405  02070 002076          DEF FI        F7 ENTRY
0406  02071 002076          DEF FI        F9 ENTRY
0407  02072 004110 ERROR    OCT 4110
0408  02073 002076          DEF FI        F8 ENTRY
0409  02074 002076          DEF FI
0410  02075 002076          DEF FI      F5  NEW ENTRY
0411*
0412  02015                 ORG 2015B
0413  02015 002242          DEF FIND      FIND ENTRY
0414  02016 002025          DEF PRTCP       PROTECT ENTRY
0415*
0416  02055                 ORG 2055B
0417  02055 002146          DEF FRET      F-RET ENTRY
0418  02042                 ORR


0420  02042 137777 PROF     OCT 137777
0421  02043 001670 FSL2     OCT 1670
0422  02044 040000 PRMSK    OCT 40000
0423  02045 001666 FSOR     OCT 1666
0424  02045        FSLL     EQU FSOR
0425  02046 001700 FSHL     OCT 1700
0426  02047 004000 FNCB     OCT 4000
```

USER DEFINABLE FUNCTION BLOCK—Continued

```
0427  02050 000072 DEFK   OCT 72        CODE FOR DEFINE
0428  02051 000016 PRTCT  OCT 16
0429  02052 000055 FRETK  OCT 55
0430  02053 173777 FOFM   OCT 173777
0431  02054 177776 M2     DEC -2
0432  00047        AUSO   EQU 47B
0433  01725        KBUF   EQU 1725B
0434  01724        TEMP1  EQU 1724B
0435  00055        MAX    EQU 55B
0436  01701        PCNT   EQU 1701B
0437  01776        PMODE  EQU 1776B
0438  00167        INIT   EQU 167B
0439  21665        FSTP   EQU 1665B
0440  00103        EXCF   EQU 103B
0441  01707        USLO   EQU 1707B
0442  00132        CONT   EQU 132B
0443  00115        GETK   EQU 115B
0444  00113        KLOG   EQU 113B
0445  00772        P2     EQU 772B
0446  00031        M1     EQU 31B
0447  00024        M10    EQU 24B
0448  01664        TEMP2  EQU 1664B
0449  01721        ADDRC  EQU 1721B
0450  00357        INTR2  EQU 357B
0451  00111        SECW   EQU 111B


0453  21665               ORG 1665B
0454  01665 000000        OCT 0
0455  02055               ORR
0456                      END
*  NO ERRORS*



                  ASMB,B,L,A,C
*  NO ERRORS*

0001              ASMB,B,L,A,C
0002  02000             ORG 2000B
0003  01776       FLGWD EQU 1776B
0004  00113       KELOG EQU 113B      KEYLOG ENTRY POINT
0005  00103       XEOFL EQU 103B       KEYBOARD EXECUSION FLAG
0006  00115       GETKE EQU 115B       GET-ANOTHER-KEY ENTRY
0007  01725       KYBUF EQU 1725B      KEYCODE BUFFER
0008  01701       PGCNT EQU 1701B
0009  00167       MON   EQU 167B       MAIN CALC MONITOR ENTRY POINT
0010  00015       ARIP  EQU 15B
0011  01665       COUT  EQU 1665B      SUMMATION OUTER LOOP COUNTER
0012  01666       RPNTR EQU 1666B      SUMMATION REGISTER POINTER
0013  01667       CIN   EQU 1667B      SUMMATION INNER LOOP COUNTER
0014  01664       CLFLG EQU 1664B      CLEAR-CORRECT FLAG WORD
0015  00064       PIP   EQU 64B       POINTER TO PI
0016  00003       FOUR  EQU 3B        BINARY CONSTANT FOUR
0017  00276       SQRT  EQU 276B      ENTRY (INDIRECT) TO SQUARE-ROOT
0018  00264       INTX  EQU 264B      ENTRY (INDIRECT) TO INTEGER-PART
0019  00212       XSQR  EQU 212B      ENTRY (INDIRECT) TO X-SQUARE
0020  00330       STLIT EQU 330B      ENTRY TO SET-LIGHT ROUTINE(OTA,STA)
0021  00063       ONE   EQU 63B       BINARY CONSTANT ONE
0022*

0024*
0025  02000 021725 STAT  LDA KYBUF       GET KEYCODE
0026  02001 042121        ADA BASE1      ADD OFFSET
0027  02002 031720        STA 1720B
0028  02003 074737        JMP A,I        JUMP TO KEYCODE ROUTINE
0029  02004 004147        DEF RAND      KEYCODE 15
0030  02005 002103        DEF CORR           "    16
0031  02006 060115 KEY2  JSM GETKE      GET-SECOND-KEY ROUTINE
```

—Continued

```
0032  02007 070137        LDB A           MOVE KEYCODE
0033  02010 120103        LDA XEQFL,I     GET EXECUTION FLAG
0034  02011 072110        RZA *+2         SKIP IF NOT FROM KEYBOARD
0035  02012 160113        JSM KELOG,I        ITS FROM KEYBOARD, KEYLOG IT
0036  02013 170402        RET
0037*
0038  02014 062006 VARBL  JSM KEY2        GO GET DIGIT KEY
0039  02015 016102        CPB FIVE        IS IT FIVE?
0040  02016 066030        JMP FVE
0041  02017 014003        CPB FOUR        IS IT FOUR?
0042  02020 066032        JMP FUR
0043  02021 016101        CPB THREE       IS IT THREE?
0044  02022 066034        JMP TREE
0045  02023 016053        CPB TWO         IS IT TWO?
0046  02024 066075        JMP TOO
0047  02025 014063        CPB ONE         IS IT ONE?
0048  02026 066077        JMP WON
0049  02027 166071        JMP ERSTP,I     NOT CORRECT DIGIT-ERROR
0050  02030 022046 FVE     LDA FLG2M       GET DIGIT-2 FLAG IN A
0051  02031 066035         JMP INCL3
0052  02032 022045 FUR     LDA FLG1M       GET DIGIT-1 FLAG IN A
0053  02033 066035         JMP INCL3
0054  02034 070742 TREE    SAR 16          CLEAR DIGIT FLAGS IN A
0055  02035 042072 INCL3   IOR FLG3M       PUT IN DIGIT-3 FLAG
0056  02036 070137 PUTIT   LDB A           SAVE FLAGS
0057  02037 022056         LDA FLGMS       GET FLAG MASK
0058  02040 070056         CMA             MAKE IT FLAG-REMOVAL MASK
0059  02041 051776         AND FLGWD       GET FLAG-WORD,MASKING OUT DIGIT FLAGS
0060  02042 074217         IOR B           PUT NEW DIGIT FLAGS IN
0061  02043 066073         JMP REST
0062  02044 002014         DEF VARBL             KEYCODE 55
0063  02045 020000 FLG1M   OCT 20000         OPTION 1 G FLAG
0064  02046 040000 FLG2M   OCT 40000         OTION 2 FLAG
0065  02047 002526         DEF REGR        KEYCODE 60
0066  02050 002471         DEF MEAN        KEYCODE 61
0067  02051 002127         DEF CLEAR       KEYCODE 62
0068  02052 002434         DEF VRNCE       KEYCODE 63
0069  02053 000002 TWO     OCT 2           BINARY CONSTANT TWO
0070  02054 003560         DEF LNX         KEYCODE 65
0071  02055 003177         DEF RSQR        KEYCODE 66
0072  02056 064000 FLGMS   OCT 64000       DIGIT FLAG MASK
0073  02057 002717         DEF TPAIR             KEYCODE 70
0074  02060 002207         DEF SUM               KEYCODE 71
0075  02061 003023         DEF CHI               KEYCODE 72
0076  02062 003063         DEF MAXM              KEYCODE 73
0077  02063 003771         DEF LOG         KEYCODE 74
0078  02064 003421         DEF EXP         KEYCODE 75
0079  02065 004400 FMP     DEF 4400B       ENTRY (INDIRECT) TO F.P. MULTIPLY

0080  02066 004263 FSUB    DEF 4263B       ENTRY (INDIRECT) TO F.P. SUBTRACT
0081  02067 004275 FAD     DEF 4275B       ENTRY (INDIRECT) TO F.P. ADD
0082  02070 004453 FDV     DEF 4453B       ENTRY (INDIRECT) TO F.P. DIVIDE
0083  02071 004110 ERSTP   DEF 4110B       ENTRY (INDIRECT) TO ERROR-STOP
0084  02072 044000 FLG3M   OCT 44000       DIGIT-3 FLAG (MASK)
0085*
0086  02073 060330 REST    JSM STLIT       OUTPUT AND SAVE FLAG (INDICATOR) WORD
0087  02074 064167         JMP MON
0088*
0089  02075 022046 TOO     LDA FLG2M       DIGIT WAS 2 - GET DIGIT-2 FLAG
0090  02076 066036         JMP PUTIT
0091  02077 022045 WON     LDA FLG1M       DIGIT WAS 1 - GET DIGIT-1 FLAG
0092  02100 066036         JMP PUTIT
0093  02101 000003 THREE   OCT 3           BINARY CONSTANT THREE
0094  02102 000005 FIVE    OCT 5           BINARY CONSTANT FIVE
0095*
0096  02103 021664 CORR    LDA CLFLG       CORRECT KEY - GET CLEAR/CORRECT FLAGWORD
0097  02104 072051         SLA *+1,S       SET CORRECT FLAG
0098  02105 031664         STA CLFLG       PUT CLEAR/CORRECT FLAGWORD BACK
0099  02106 062046         JSM KEY2        GET NEXT KEY
0100  02107 016122         CPB K70         IS IT T?
0101  02110 066717         JMP TPAIR
0102  02111 016123         CPB K71         IS IT SUM?
0103  02112 066207         JMP SUM
0104  02113 016124         CPB K72         IS IT CHI?
0105  02114 067023         JMP CHI
0106  02115 021664 CANCL   LDA CLFLG          GET CLEAR-CORRECT FLAG WORD
0107  02116 070071         SLA *+1,C           TURN OFF CORRECT FLAG, NEXT KEY WAS WRONG
0108  02117 031664         STA CLFLG       SAVE FLAG WORD
0109  02120 064167         JMP MON              GO DO NEXT KEY
```

—Continued

```
0110*
0111  02121 101767 BASE1 OCT 101767     BASE FOR INDIRECT JUMP INTO KEY-TABLE
0112  02122 000070 K70   OCT 70         KEYCODE 70 CONSTANT
0113  02123 000071 K71   OCT 71         KEYCODE 71 CONSTANT
0114  02124 000072 K72   OCT 72         KEYCODE 72 CONSTANT
0115  02125 000073 K73   OCT 73         KEYCODE 73 CONSTANT
0116  02126 000000 CKSM1 OCT 0          CHECK-SUM WORD FOR 2000-2777 BLOCK

0118*
0119*
0120  02127 062006 CLEAR JSM KEY2       CLEAR KEY - GET NEXT KEY
0121  02130 016122       CPB K70        IS IT T?
0122  02131 066141       JMP TCLR
0123  02132 016123       CPB K71        IS IT SUM?
0124  02133 066150       JMP SUMCL
0125  02134 016124       CPB K72        IS IT CHI?
0126  02135 066143       JMP CHICL
0127  02136 016125       CPB K73        IS IT MAX/MIN?
0128  02137 066154       JMP MAXCL
0129  02140 166071       JMP ERSTP,I    WRONG KEY - ERROR
0130  02141 022345 TCLR  LDA R2         IT IS T - CLEAR R2,R1,R0
0131  02142 170000       CLR
0132  02143 022344 CHICL LDA R1         IT IS CHI - CLEAR R1, R0
0133  02144 170000       CLR
0134  02145 020033       LDA R0
0135  02146 170000       CLR
0136  02147 064167       JMP MON
0137  02150 021664 SUMCL LDA CLFLG      IT IS SUM - GET CLEAR/CORRECT FLAGWORD
0138  02151 072053       SAP *+1,S      SET CLEAR FLAG
0139  02152 031664       STA CLFLG      SAVE CLEAR/CORRECT FLAGWORD
0140  02153 066207       JMP SUM        GO TO SUM ROUTINE TO CLEAR SUM REGISTER
0141*
0142  02154 020015 MAXCL LDA AR1P         LOAD AR1 PNTR. TO MAKE MAX NO. FOR INIT.
0143  02155 170000       CLR             CLEAR AR1
0144  02156 170540       MDI             INCR. BY 1
0145  02157 174400       CMX             10S COMPLEMENT TO GET ALL 9S
0146  02160 026206       LDB EX98        GET +E+98 EXPONENT WORD
0147  02161 134015       STB AR1P,I
0148*
0149  02162 020015       LDA AR1P
0150  02163 027106       LDB R21
0151  02164 170004       XFR             JMOVE +MAX TO XMIN
0152  02165 020015       LDA AR1P
0153  02166 027110       LDB R23
0154  02167 170004       XFR             MOVE +MAX TO YMIN
0155  02170 020015       LDA AR1P
0156  02171 027112       LDB R25
0157  02172 170004       XFR             MOVE +MAX TO ZMIN
0158  02173 144015       ISZ AR1P,I
0159  02174 020015       LDA AR1P
0160  02175 027107       LDB R22
0161  02176 170004       XFR             MOVE -MAX TO XMAX
0162  02177 020015       LDA AR1P
0163  02200 027111       LDB R24
0164  02201 170004       XFR             MOVE -MAX TO YMAX
0165  02202 020015       LDA AR1P
0166  02203 027113       LDB R26
0167  02204 170004       XFR             MOVE -MAX TO ZMAX
0168  02205 064167       JMP MON            GO TO DO SOME THING ELSE
0169  02206 061000 EX98  OCT 61000


0171*
0172*
0173  02207 062337 SUM   JSM GTFLG      GET THE DIGIT FLAGS
0174  02210 072110       RZA SUM1           SKIP IF ANY ON
0175  02211 062326       JSM SET3       NO FLAGS - SET FOR 3-VARIABLE
0176  02212 062334 SUM1  JSM FLAG3      GET DIGIT-3 FLAG OUT OF FLAGWORD
0177  02213 051776       AND FLGWD          A SEE IF FLAG 3 SET
0178  02214 070710       SZA SUM2           IT IS NOT, SO SKIP
0179  02215 062331       JSM FLAG2            GET FLAG 2
0180  02216 070210       SZA SUM12             IS IT 2 AND 3?  SKIP IF NOT
0181  02217 022310       LDA S4         IT IS 5-SET UP SUM-LOOP ENTRY
0182  02220 026342       LDB R20        TO R20 AND 8
0183  02221 066241       JMP SUMR
```

—Continued

```
0184  02222 062466 SUM12  JSM  FLAG1      IS IT 1 OR 3?
0185  02223 070210        SZA  SUM13      IT IS 3 WITHOUT 1 - SKIP
0186  02224 022311        LDA  S3         IT IS 4 - SET UP SUM-LOOP ENTRY
0187  02225 027114        LDB  R14        TO R14 AND A
0188  02226 066241        JMP  SUMR
0189  02227 022312 SUM13  LDA  S2         IT IS 3-SET SUM-LOOP ENTRY TO R9 AND Z
0190  02230 026353        LDB  R9           SET MAXIMUM SUMMATION REGISTER
0191  02231 066241        JMP  SUMR         GO TO DO SUM
0192  02232 062331 SUM2   JSM  FLAG2
0193  02233 070210        SZA  SUM3         IT IS NOT, SO SKIP
0194  02234 022313        LDA  S1           SET FOR SUM TO Y
0195  02235 026350        LDB  R5           SET MAXIMUM SUMMATION REGISTER
0196  02236 066241        JMP  SUMR          GO TO DO SUM
0197  02237 022314 SUM3   LDA  S0           SET FOR SUM TO X
0198  02240 026345        LDB  R2           SET MAXIMUM SUMMATION REGISTER
0199  02241 031665 SUMR   STA  COUT         SAVE OUTER LOOP COUNTER
0200  02242 035666        STB  RPNTR        SAVE REGISTER POINTER
0201  02243 021665 OLOOP  LDA  COUT          GET OUTER LOOP COUNTER
0202  02244 031667        STA  CIN          MAKE INNER LOOP COUNTER THE SAME
0203  02245 021664 ILOOP  LDA  CLFLG         GET CLEAR-CORRECT FLAG WORD
0204  02246 070213        SAP  ILOP1        SEE IF CLEAR FLAG ON
0205  02247 021666        LDA  RPNTR        IT IS, GET POINTER
0206  02250 170000        CLR               CLEAR THE SUMMATION REGISTER
0207  02251 066271        JMP  TESTI        GO TO TEST FOR INNER LOOP TERMINATION
0208  02252 121667 ILOP1  LDA  CIN,I        NOT CLEAR - LOAD FIRST FACTOR POINTER
0209  02253 026324        LDB  T2P          SET TO
0210  02254 170004        XFR               MOVE FIRST FACTOR TO TEMP
0211  02255 121665        LDA  COUT,I       SET TO SECOND FACTOR
0212  02256 026324        LDB  T2P          SET TO
0213  02257 162065        JSM  FMP,I        PRODUCT OF FACTORS IN TEMP
0214  02260 021664        LDA  CLFLG        GET CLEAR-CORRECT FLAG WORD
0215  02261 070251        SLA  ILOP2        IF NO CORRECT FLAG, SKIP
0216  02262 022324        LDA  T2P          CORRECT ON - LOAD FROM
0217  02263 025666        LDB  RPNTR        SET TO
0218  02264 162066        JSM  FSUB,I       SUBTRACT OUT FACTOR AS CORRECTION
0219  02265 066271        JMP  TESTI        GO TO TEST FOR INNER LOOP TERMINATION
0220  02266 022324 ILOP2  LDA  T2P          NO CORRECT, SET TO
0221  02267 025666        LDB  RPNTR        SET TO
0222  02270 162067        JSM  FAD,I        ADD NEW FACTOR IN
0223  02271 021666 TESTI  LDA  RPNTR        GET REGISTER POINTER
0224  02272 000003        ADA  FOUR         MOVE TO NEXT REGISTER
0225  02273 031666        STA  RPNTR        SAVE THE NEW POINTER
0226  02274 022315        LDA  S            GET INNER LOOP LIMIT

0227  02275 011667        CPA  CIN          SEE IF INNER LOOP DONE- SKIP IF NOT
0228  02276 066301        JMP  TESTO        GO TO SEE IF OUTER LOOP IS DONE
0229  02277 045667        ISZ  CIN          IT IS  NOT DONE - MOVER FIRST FACTOR PNTR
0230  02300 066245        JMP  ILOOP         GO DO THE INNER LOOP AGAIN
0231  02301 011665 TESTO  CPA  COUT         SEE IF OUTER LOOP IS DONE
0232  02302 066305        JMP  FINS         OUTER LOOP DONE
0233  02303 045665        ISZ  COUT         MOVE OUTER LOOP COUNTER
0234  02304 066243        JMP  OLOOP         GO DO THE OUTER LOOP AGAIN
0235  02305 074742 FINS   SAR  16           CLEAR A REGISTER
0236  02306 031664        STA  CLFLG         CLEAR OUT THE CLEAR-CORRECT FLAG REISTER
0237  02307 064167        JMP  MON               GO TO DO ANOTHER KEY
0238  02310 002316 S4     DEF  BR         S-VARIABLE START-POINT
0239  02311 002317 S3     DEF  AR         4-VARIABLE START-POINT
0240  02312 002320 S2     DEF  ZR              3-VARIABLE START POINT
0241  02313 002321 S1     DEF  YR              2-VARIABLE START POINT
0242  02314 002322 S0     DEF  XR              1-VARIABLE START POINT
0243  02315 002323 S      DEF  WONE            CONSTANT 1.0 POINTER
0244  02316 001770 BR     DEF  1770B      ADDRESS OF B-REGISTER
0245  02317 001764 AR     DEF  1764B      ADDRESS OF A-REGISTER
0246  02320 001760 ZR     DEF  1760B      ADDRESS OF Z-REGISTER
0247  02321 001740 YR     DEF  1740B      ADDRESS OF Y-REGISTER
0248  02322 001750 XR     DEF  1750B      ADDRESS OF X-REGISTER
0249  02323 000065 WONE   DEF  65B            CONSTANT 1.0 POINTER
0250  02324 001670 T2P    DEF  1670B      ADDRESS OF DECIMAL TEMP-2
0251  02325 001674 T3P    DEF  1674B      ADDRESS OF DECIMAL TEMP-3

0253*
0254*
0255*
0256  02326 022072 SET3   LDA  FLG3M      GET DIGIT-3 MASK
0257  02327 041776        IOR  FLGWD      PUT 3-FLAG IN FLAGWORD
0258  02330 064330        JMP  STLIT      JUMP TO LIGHT SETTING ROUTINE (OTA,STA)
0259*
0260*
```

—Continued

```
0261  02331 022046 FLAG2 LDA FLG2M     GET DIGIT-2 MASK
0262  02332 051776       AND FLGWD     SEE IF 2-FLAG ON
0263  02333 170402       RET
0264*
0265*
0266  02334 022072 FLAG3 LDA FLG3M     GET DIGIT-3 MASK
0267  02335 051776       AND FLGWD     SEE IF 3-FLAG ON
0268  02336 170402       RET
0269*
0270*
0271  02337 022056 GTFLG LDA FLGMS     GET FLAG WORD
0272  02340 051776       AND FLGWD     MASK TO GET DIGIT FLAGS
0273  02341 170402       RET


0275*
0276*
0277*       MU-TABLE
0278*
0279  02342 001540 R20    DEF 1540B
0280  00033        R0     EQU 33B
0281  02343 001654 XX     DEF 1654B       R1  MU-XX TRIPLE
0282  02344 001654 R1     DEF 1654B       R1
0283  02345 001650 R2     DEF 1650B       R2
0284  02346 001644 YY     DEF 1644B       R3  MU-YY TRIPLE
0285  02347 001644 R3     DEF 1644B       R3
0286  02350 001634 R5     DEF 1634B       R5
0287  02351 001630 ZZ     DEF 1630B       R6  MU-ZZ TRIPLE
0288  02352 001630 R6     DEF 1630B       R6
0289  02353 001614 R9     DEF 1614B       R9
0290  02354 001654 XY     DEF 1654B       R1  MU-XY TRIPLE
0291  02355 001644        DEF 1644B       R3
0292  02356 001640 R4     DEF 1640B       R4
0293  02357 001654 XZ     DEF 1654B       R1  MU-XZ TRIPLE
0294  02360 001630        DEF 1630B       R6
0295  02361 001624 R7     DEF 1624B       R7
0296  02362 001644 YZ     DEF 1644B       R3  MU-YZ TRIPLE
0297  02363 001630        DEF 1630B       R6
0298  02364 001620 R8     DEF 1620B       R8
0299*
0300  02365 002343 XXP    DEF XX        POINTER FOR MU-XX
0301  02366 002346 YYP    DEF YY        POINTER FOR MU-YY
0302  02367 002351 ZZP    DEF ZZ        POINTER FOR MU-ZZ
0303  02370 002354 XYP    DEF XY        POINTER FOR MU-XY
0304  02371 002357 XZP    DEF XZ        POINTER FOR MU-XZ
0305  02372 002362 YZP    DEF YZ        POINTER FOR MU-YZ
0306*
0307*
0308*     MU SUBROUTINE
0309*
0310  02373 031665 MU     STA COUT        SAVE MAX-VARIABLE POINTER
0311  02374 121665        LDA COUT,I      GET FIRST FACTOR ADDRESS
0312  02375 026324        LDB T2P         SET TO
0313  02376 170004        XFR              FIRST FACTOR TO TEMP
0314  02377 045665        ISZ COUT        MOVE POINTER TO SECOND FACTOR ADDRESS
0315  02400 121665        LDA COUT,I      GET SECOND FACTOR ADDRESS
0316  02401 026324        LDB T2P         SET TO
0317  02402 162065        JSM FMP,I       PRODUCT IN TEMP
0318  02403 045665        ISZ COUT        MOVER POINTER TO THIRD FACTOR ADDRESS
0319  02404 020033        LDA R0          SET TO GET N
0320  02405 026324        LDB T2P         SET TO
0321  02406 162070        JSM FDV,I       DIVIDE PRODUCT BY N
0322  02407 121665        LDA COUT,I      GET THIRD FACTOR ADDRESS
0323  02410 026324        LDB T2P         SET TO
0324  02411 162066        JSM FSUB,I      DIFFERENCE IN TEMP
0325  02412 126324        LDB T2P,I       GET EXPONENT WORD (SIGN) IN B
0326  02413 074151        SLB PLUS        SKIP IF SIGN +
0327  02414 074071        SLB *+1,C          ITS MINUS - CHANGE TO PLUS
0328  02415 056417        JMP PLUS+1
0329  02416 076051 PLUS   SLB *+1,S          ITS PLUS - CHANGE TO MINUS
0330  02417 136324        STB T2P,I       PUT ALTERED EXPONENT WORD BACK

0331  02420 170402        RET
```

—Continued

```
0333*
0334*
0335*     MU-TO-VARIANCE SUBROUTINE
0336*
0337  02421 020030 MUTVR LDA R0          SET PNTR TO GET N
0338  02422 026325       LDB T3P         SET TO TO ANOTHER TEMP
0339  02423 170004       XFR             MOVE N TO ANOTHER TEMP
0340  02424 022323       LDA WONE         SET PNTR TO GET 1.0
0341  02425 026325       LDB T3P         SET TO
0342  02426 162066       JSM FSUB,I      (N-1) IN SECOND TEMP
0343  02427 022325       LDA T3P         SET FROM
0344  02430 026324       LDB T2P         SET TO
0345  02431 162070       JSM FDV,I       DIVIDE MU BY (N-1) TO GET VARIANCE
0346  02432 022324       LDA T2P         SET FROM PNTR TO GET RESULT
0347  02433 170402       RET
0349*
0350*
0351*   VRNCE - SUBROUTINE TO COMPUTE VARIANCE OF X, (Y,Z)
0352*
0353  02434 063252 VRNCE JSM CLER3       CLEAR X,Y,Z REGISTERS
0354  02435 062337       JSM GTFLG       GET DIGIT FLAGS
0355  02436 072110       RZA VRN1        IF VARIABLES DEFINED (ANY FLAG) - SKIP
0356  02437 166071       JMP ERSTP,I     VARIABLES NOT DEFINED  - ERROR
0357  02440 062334 VRN1  JSM FLAG3       SEE IF DIGIT-3 (3,4,5 VARIABLES) ON
0358  02441 070110       SZA VRN2        IT ISN'T - SKIP
0359  02442 066446       JMP VRNC3       IT IS - DO 3-VARIABLE VARIANCE
0360  02443 062331 VRN2  JSM FLAG2       SEE IF DIGIT-2 ON
0361  02444 070610       SZA VRNC1       IT ISN'T - DO 1-VARIABLE VARIANCE
0362  02445 066453       JMP VRNC2       IT IS - DO 2-VARIABLE VARIANCE
0363  02446 022367 VRNC3 LDA ZZP         DO VARIANCE OF Z
0364  02447 062373       JSM MU
0365  02450 062421       JSM MUTVR
0366  02451 026320       LDB ZR
0367  02452 170004       XFR
0368  02453 022366 VRNC2 LDA YYP         DO VARIANCE OF Y
0369  02454 062373       JSM MU
0370  02455 062421       JSM MUTVR
0371  02456 026321       LDB YR
0372  02457 170004       XFR
0373  02460 022365 VRNC1 LDA XXP         DO VARIANCE OF X
0374  02461 062373       JSM MU
0375  02462 062421       JSM MUTVR
0376  02463 026322       LDB XR
0377  02464 170004       XFR
0378  02465 064167       JMP MON

0380  02466 022045 FLAG1 LDA FLG1M
0381  02467 051776       AND FLGWD
0382  02470 170402       RET
0383*
0384*
0385*     MEAN - SUBROUTINE TO COMPUTE MEAN X (Y,Z)
0386*
0387  02471 063252 MEAN  JSM CLER3       CLEAR X,Y,Z REGISTERS
0388  02472 062337       JSM GTFLG       GET DIGIT FLAGS
0389  02473 072110       RZA MEN1        IF VARIABLES DEFINED (ANY FLAG) - SKIP
0390  02474 166071       JMP ERSTP,I     VARIABLES NO DEFINED - ERROR
0391  02475 062334 MEN1  JSM FLAG3       SEE IF DIGIT-3 (3,4,5 VARIABLES) ON
0392  02476 070110       SZA MEN2        IT ISN'T - SKIP
0393  02477 066503       JMP MEAN3       IT IS - DO 3-VARIABLE MEAN
0394  02500 062331 MEN2  JSM FLAG2       SEE IF DIGIT-2 ON
0395  02501 070710       SZA MEAN1       IT ISN'T - DO 1-VARIABLE MEAN
0396  02502 066511       JMP MEAN2       IT IS - DO 2-VARIABLE MEAN
0397  02503 022352 MEAN3 LDA R6          DO MEAN OF Z
0398  02504 026320       LDB ZR
0399  02505 170004       XFR
0400  02506 020033       LDA R0
0401  02507 026320       LDB ZR
0402  02510 162070       JSM FDV,I
0403  02511 022347 MEAN2 LDA R3          DO MEAN OF Y
0404  02512 026321       LDB YR
0405  02513 170004       XFR
```

—Continued

```
0406  02514 020033        LDA R0
0407  02515 026321        LDB YR
0408  02516 162070        JSM FDV,I
0409  02517 022344 MEAN1  LDA R1          DO MEAN OF X
0410  02520 026322        LDB XR
0411  02521 170004        XFR
0412  02522 020033        LDA R0
0413  02523 026322        LDB XR
0414  02524 162070        JSM FDV,I
0415  02525 064167        JMP MON
0417*
0418*
0419*      REGR - SUBROUTINE TO DO REGRESSION COEFFICIENT COMPUTATION
0420*             FOR 1 OR 2 INDEPENDENT F VARIABLE
0421*
0422  02526 062337 REGR   JSM GTFLG      GET DIGIT FLAGS
0423  02527 072110        RZA REG1       IF VARIABLES DEFINED (ANY FLAG) - SKIP
0424  02530 166071        JMP ERSTP,I    VARIABLES NO DEFINED - ERROR
0425  02531 062334 REG1   JSM FLAG3      SEE IF DIGIT 3 (3,4,5 VARIABLES) ON
0426  02532 070110        SZA REG2       IT ISN'T - SKIP
0427  02533 066567        JMP REGR3      IT IS - DO 3-VARIABLE REGRESSION
0428  02534 062331 REG2   JSM FLAG2      SEE IF DIGIT-2 ON
0429  02535 070110        SZA REGR1      IT ISN'T - SKIP
0430  02536 066540        JMP REGR2      IT IS - DO 2-VARIABLE REGRESSION
0431  02537 166071 REGR1  JMP ERSTP,I    VARIABLE 1 - ERROR
0432  02540 022370 REGR2  LDA XYP        DO 2-VARIABLE REGRESSION
0433  02541 062373        JSM MU           MAKE MU-XY
0434  02542 062707        JSM TXFR
0435  02543 022365        LDA XXP
0436  02544 062373        JSM MU           MAKE MU-XX
0437  02545 062713        JSM TDVX
0438  02546 022344        LDA R1           SET TO GET SUM-X
0439  02547 026324        LDB T2P
0440  02550 170004        XFR              SUM-X TO TEMP
0441  02551 022322        LDA XR
0442  02552 026324        LDB T2P
0443  02553 162065        JSM FMP,I        A1 * SUM-X IN TEMP
0444  02554 022347        LDA R3
0445  02555 026321        LDB YR
0446  02556 170004        XFR              MOVE SUM-Y TO Y
0447  02557 022324        LDA T2P
0448  02560 026321        LDB YR
0449  02561 162066        JSM FSUB,I       SUM-Y - A1*SUM-X IN Y
0450  02562 020033        LDA R0
0451  02563 026321        LDB YR
0452  02564 162070        JSM FDV,I        MAKE A-0 IN Y, BY DIVIDING BY N
0453  02565 063256        JSM CLER1      CLEAR Z REGISTER
0454  02566 064167        JMP MON        GO DO NEXT THING
0455  02567 062571 REGR3  JSM REGR4      3-VARIABLE REGRESSION - CALL AS SUBR
0456  02570 064167        JMP MON        DO GO NEXT THING
0457  02571 022372 REGR4  LDA YZP        SUBROUTINE FOR 3-VARIABLE REGRESSION
0458  02572 062373        JSM MU           MAKE MU-YZ
0459  02573 022324        LDA T2P
0460  02574 026321        LDB YR
0461  02575 170004        XFR              MOVE MU-YZ TO X
0462  02576 022365        LDA XXP
0463  02577 062373        JSM MU           MAKE MU-XX
0464  02600 022324        LDA T2P
0465  02601 026321        LDB YR
0466  02602 162065        JSM FMP,I        MU-YX * MU-XX IN X
0467  02603 022371        LDA XZP
0468  02604 062373        JSM MU           MAKE MU-XZ
0469  02605 062707        JSM TXFR
0470  02606 022370        LDA XYP
0471  02607 062373        JSM MU
0472  02610 063246        JSM TMPX


0473  02611 022322        LDA XR
0474  02612 026321        LDB YR
0475  02613 162066        JSM FSUB,I         NUMERATOR IN X
0476  02614 022365        LDA XXP
0477  02615 062373        JSM MU           MAKE MU-XX
```

—Continued

```
0478  02616 062707        JSM TXFR
0479  02617 022366        LDA YYP
0480  02620 062373        JSM MU              MAKE MU-YY
0481  02621 063246        JSM TMPX
0482  02622 022370        LDA XYP
0483  02623 062373        JSM MU              MAKE MU-XY
0484  02624 022324        LDA T2P
0485  02625 026320        LDB ZR
0486  02626 170004        XFR                 MOVE MU-XY TO Z
0487  02627 022324        LDA T2P
0488  02630 026320        LDB ZR
0489  02631 162065        JSM FMP,I           MU-XY SQUARED IN Z
0490  02632 022320        LDA ZR
0491  02633 026322        LDB XR
0492  02634 162066        JSM FSUB,I          DENOM IN Z
0493  02635 022322        LDA XR
0494  02636 026321        LDB YR
0495  02637 162070        JSM FDV,I           A-2 IN X, ALL DONE
0496  02640 022371        LDA XZP
0497  02641 062373        JSM MU              MAKE MU-XZ
0498  02642 062707        JSM TXFR
0499  02643 022370        LDA XYP
0500  02644 062373        JSM MU              MAKE MU-XY
0501  02645 022321        LDA YR
0502  02646 026324        LDB T2P
0503  02647 162065        JSM FMP,I           A2 * MUXY IN TEMP
0504  02650 022324        LDA T2P
0505  02651 026322        LDB XR
0506  02652 162066        JSM FSUB,I          MUXZ - A1*MUXY IN Y
0507  02653 022365        LDA XXP
0508  02654 062373        JSM MU              MAKE MU-XX
0509  02655 062713        JSM TDVX
0510  02656 022352        LDA R6
0511  02657 026320        LDB ZR
0512  02660 170004        XFR                 SUM-Z IN Z
0513  02661 022344        LDA R1
0514  02662 026324        LDB T2P
0515  02663 170004        XFR                 SUM-X IN TEMP
0516  02664 022322        LDA XR
0517  02665 026324        LDB T2P
0518  02666 162065        JSM FMP,I           A1*SUM-X IN TEMP
0519  02667 022324        LDA T2P
0520  02670 026320        LDB ZR
0521  02671 162066        JSM FSUB,I          SUM-Z - A1*SUM-X IN Z
0522  02672 022347        LDA R3
0523  02673 026324        LDB T2P
0524  02674 170004        XFR                 SUM-Y IN TEMP
0525  02675 022321        LDA YR
0526  02676 026324        LDB T2P
0527  02677 162065        JSM FMP,I           A2*SUM-Y IN TEMP
0528  02700 022324        LDA T2P

0529  02701 026320        LDB ZR
0530  02702 162066        JSM FSUB,I
0531  02703 022343        LDA R0
0532  02704 026320        LDB ZR
0533  02705 162070        JSM FDV,I           A0 IN Z --- ALL DONE
0534  02706 170402        RET


0536*
0537*
0538  02707 022324 TXFR   LDA T2P         TRANSFER DECIMAL TEMP TO X
0539  02710 026322        LDB XR
0540  02711 170004        XFR
0541  02712 170402        RET
0542*
0543*
0544  02713 022324 TDVX   LDA T2P         DIVIDE X BY DEC. TEMP - RESULT IN X
0545  02714 026322        LDB XR
0546  02715 162070        JSM FDV,I
0547  02716 170402        RET
```

```
0549*
0550*
0551*
0552*     TPAIR - SUBROUTINE TO COMPUTE A RUNNING PAIRED-T STATISTIC
0553*
0554*
0555  02717 022321 TPAIR LDA YR
0556  02720 026322       LDB XR
0557  02721 162066       JSM FSUB,I        (X - Y) IN X
0558  02722 022322       LDA XR
0559  02723 026324       LDB T2P
0560  02724 170004       XFR               SAVE (X-Y) IN TEMP
0561  02725 160212       JSM XSQR,I
0562  02726 021664       LDA CLFLG         GET CLEAR-CORRECT FLAG WORD
0563  02727 070631       SLA TP2,C
0564  02730 031664       STA CLFLG         SAVE THE FLAG
0565  02731 022322       LDA XR
0566  02732 026345       LDB R2
0567  02733 162066       JSM FSUB,I        MAKE CORRECTION BY SUBTRACTING FACTOR
0568  02734 022324       LDA T2P
0569  02735 026344       LDB R1
0570  02736 162066       JSM FSUB,I        MAKE CORRECTION ON D-SQUARE
0571  02737 022323       LDA WONE
0572  02740 024033       LDB R0
0573  02741 162066       JSM FSUB,I
0574  02742 066755       JMP TP3
0575  02743 031664 TP2    STA CLFLG          SAVE THE FLAG WORD
0576  02744 022322       LDA XR          ACCUMULATE (X-Y) SQUARED IN R2
0577  02745 026345       LDB R2
0578  02746 162067       JSM FAD,I
0579  02747 022324       LDA T2P         ACCUMULATE (X-Y) IN R1
0580  02750 026344       LDB R1
0581  02751 162067       JSM FAD,I
0582  02752 022323       LDA WONE
0583  02753 024033       LDB R0
0584  02754 162067       JSM FAD,I          INCREMENT COUNTER
0585  02755 021660 TP3    LDA 1660B      CHECK IF N (R0) IS 1.0
0586  02756 001661       ADA 1661B
0587  02757 001662       ADA 1662B
0588  02760 001663       ADA 1663B
0589  02761 010066       CPA 66B        COMPARE TO FIRST MAG.WORD OF DEC 1.0
0590  02762 067015       JMP DEFAL      IT IS 1.0 - DON7T COMPUTE T
0591  02763 022365       LDA XXP        COMPUTE VARIANCE OF D-BAR
0592  02764 062373       JSM MU
0593  02765 062421       JSM MUTVR
0594  02766 026322       LDB XR
0595  02767 170004       XFR
0596  02770 020033       LDA R0
0597  02771 026322       LDB XR
0598  02772 162070       JSM FDV,I      DIVIDE VARIANCE BY N
0599  02773 160276       JSM SQRT,I         VARIANCE OF D IN X
0600  02774 022344       LDA R1
0601  02775 026324       LDB T2P
0602  02776 170004       XFR                MOVE SUM-D TI TEMP
0603  02777 020033       LDA R0
0604  03000 026324       LDB T2P
0605  03001 162070       JSM FDV,I          D-BAR IN TEMP
0606  03002 022324       LDA T2P
0607  03003 026320       LDB ZR
0608  03004 170004       XFR                D-BAR TO Z
0609  03005 022322       LDA XR
0610  03006 026324       LDB T2P
0611  03007 162070       JSM FDV,I
0612  03010 062707       JSM TXFR
0613  03011 020033 TP4    LDA R0
0614  03012 026321       LDB YR
0615  03013 170004       XFR                 N IN Y
0616  03014 064167       JMP MON              GO TO DO SOMETHING ELSE
0617  03015 022322 DEFAL  LDA XR         CLEAR X
0618  03016 170000       CLR
0619  03017 022320       LDA ZR         CLEAR Z
0620  03020 170000       CLR
0621  03021 067011       JMP TP4
0622  03022 000000 CKSM2  OCT 0
```

—Continued

```
0624*
0625*
0626*       CHI - SUBROUTINE TO COMPUTE AND DISPLAY CHI-SQUARE STATISTIC
0627*
0628*
0629  03023 022321 CHI    LDA YR
0630  03024 026322        LDB XR
0631  03025 162066        JSM FSUB,I          (X-Y) IN X
0632  03026 160212        JSM XSQR,I
0633  03027 022321        LDA YR
0634  03030 026322        LDB XR
0635  03031 162070        JSM FDV,I           (X-Y)**2/Y IN TEMP
0636  03032 021664        LDA CLFLG           GET THE FLAG WORD
0637  03033 070471        SLA CHI2,C          SKIP IF NOT CORRECT
0638  03034 031664        STA CLFLG           SAVE THE FLAG WORD
0639  03035 022322        LDA XR
0640  03036 026344        LDB R1
0641  03037 162066        JSM FSUB,I          SUBTRACT OUT FACTOR TO DO CORRECTION
0642  03040 022323        LDA WONE
0643  03041 024033        LDB R0
0644  03042 162066        JSM FSUB,I
0645  03043 067053        JMP CHI3
0646  03044 031664 CHI2   STA CLFLG            SAVE THE FLAG WORD
0647  03045 022322        LDA XR
0648  03046 026344        LDB R1
0649  03047 162067        JSM FAD,I           ADD IN THE NEW FACTOR
0650  03050 022323        LDA WONE
0651  03051 024033        LDB R0
0652  03052 162067        JSM FAD,I           INCREMENT N
0653  03053 022344 CHI3   LDA R1
0654  03054 026322        LDB XR
0655  03055 170004        XFR             MOVE CHI TO X
0656  03056 020033        LDA R0
0657  03057 026321        LDB YR
0658  03060 170004        XFR                 MOVE N TO Y
0659  03061 063256        JSM CLER1
0660  03062 064167        JMP MON


0662*
0663*
0664*       MAXM - SUBROUTINE TO KEEP RUNNING MIN AND MAX ON X (Y,Z)
0665*
0666*
0667  03063 062337 MAXM   JSM GTFLG       GET DIGIT FLAGS
0668  03064 072110        RZA MXA1        IF VARIABLES DEFINED (ANY FLAG) - SKIP
0669  03065 062326        JSM SET3        VARIABLES NOT DEFINED - SET FOR 3
0670  03066 062334 MXA1   JSM FLAG3       SEE IF DIGIT-3 (3,4,5 VARIABLES) ON
0671  03067 070110        SZA MXA2        IT ISN'T - SKIP
0672  03070 067074        JMP MAX3        IT IS - DO 3-VARIABLE MAX/MIN
0673  03071 062331 MXA2   JSM FLAG2       SEE IF DIGIT-2 ON
0674  03072 070410        SZA MAX1        IT ISN'T - DO 1-VARIABLE MAX/MIN
0675  03073 067077        JMP MAX2        IT IS - DO 2-VARIABLE MAX/MIN
0676  03074 022320 MAX3   LDA ZR
0677  03075 027112        LDB R25
0678  03076 063115        JSM CHKMX       MAX/MIN OF Z
0679  03077 022321 MAX2   LDA YR
0680  03100 027110        LDB R23
0681  03101 063115        JSM CHKMX       MAX/MIN OF Y
0682  03102 022322 MAX1   LDA XR
0683  03103 027106        LDB R21
0684  03104 063115        JSM CHKMX       MAX/MIN OF X
0685  03105 064167        JMP MON         GO DO ANOTHER THING
0686  03106 001534 R21    OCT 1534        ADDRESS OF XMIN
0687  03107 001530 R22    OCT 1530        ADDRESS OF XMAX
0688  03110 001524 R23    OCT 1524        ADDRESS OF YMIN
0689  03111 001520 R24    OCT 1520        ADDRESS OF YMAX
0690  03112 001514 R25    OCT 1514        ADDRESS OF ZMIN
0691  03113 001510 R26    OCT 1510        ADDRESS OF ZMAX
0692  03114 001570 R14    OCT 1570        ADDRESS OF UPPER REG. FOR 4-VARIABLE SUM
```

—Continued

```
0694*
0695*
0696*     CHKMX - SUBROUTINE TO CHECK AND SET MIN AND/OR MAX OF 1 VARIABLE
0697*
0698  03115 031665 CHKMX STA COUT          SAVE VARIABLE POINTER
0699  03116 035667       STB CIN           SAVE MINIMUM-REGISTER POINTER
0700  03117 026324       LDB T2P
0701  03120 170004       XFR               MOVE X (Y,Z) TO TEMP
0702  03121 021667       LDA CIN
0703  03122 026324       LDB T2P
0704  03123 162066       JSM FSUB,I        (X - XMIN) IN TEMP
0705  03124 122324       LDA T2P,I         GET SIGN OF DIFFERENCE
0706  03125 070211       SLA MX1           SKIP IF POSITIVE - SAME OLD MIN
0707  03126 021665       LDA COUT
0708  03127 025667       LDB CIN
0709  03130 170004       XFR               SIGN WAS NEG - SAVE X AS NEW XMIN
0710  03131 020054 MX1    LDA MN4
0711  03132 001667       ADA CIN           MOVE REGISTER POINTER TO MAX
0712  03133 031667       STA CIN           SAVE IT
0713  03134 026324       LDB T2P
0714  03135 170004       XFR               MOVE XMAX TO TEMP
0715  03136 021665       LDA COUT
0716  03137 026324       LDB T2P
0717  03140 162066       JSM FSUB,I        (XMAX - X) IN TEMP
0718  03141 122324       LDA T2P,I         GET SIGN
0719  03142 070211       SLA MX2           SKIP IF POSITIVE
0720  03143 021665       LDA COUT
0721  03144 025667       LDB CIN
0722  03145 170004       XFR               SIGN WAS NEG - SAVE X AS NEW XMAX
0723  03146 170402 MX2    RET                 BOTH DONE - GO BACK
0724  00054        MN4    EQU 54B

0726*
0727*
0728*      RAND - SUBROUTINE TO MAKE A RANDOM NUMBER
0729*
0730  03147 023176 RAND   LDA CPNTR      LOAD CONSTANT (MULTIPLIER) POINTER
0731  03150 027165        LDB R27
0732  03151 162065        JSM FMP,I      PRODUCT IN R27
0733  03152 023165        LDA R27        MOVE COPY TO X
0734  03153 026322        LDB XR
0735  03154 170004        XFR
0736  03155 160264        JSM INTX,I     TRUNCATE COPY IN X TO INTEGER
0737  03156 022322        LDA XR
0738  03157 027165        LDB R27
0739  03160 162066        JSM FSUB,I     SUBT. INTEGER-PART TO GET 0-1 R.N. IN R27
0740  03161 023165        LDA R27        MOVE NEW RANDOM NUMBER TO X
0741  03162 026322        LDB XR
0742  03163 170004        XFR
0743  03164 064167        JMP MON        GO DO ANOTHER THING
0744  03165 001504 R27    OCT 1504       ADDRESS OF RANDOM NUMBER - R27
0745  03166 000400 CNSTN  OCT 400        DECIMAL 29.0 CONSTANT (MULTIPLIER) EXP.
0746  03167 024400        OCT 24400      DECIMAL 29.0 CONSTANT (MULTIPLIER) MAG1
0747  03170 000000        OCT 0          DECIMAL 29.0 CONSTANT (MULTIPLIER) MAG2
0748  03171 000000        OCT 0          DECIMAL 29.0 CONSTANT (MULTIPLIER) MAG3
0749  03172 000000        OCT 0          SPARE WORD
0750  03173 000000        OCT 0          SPARE WORD
0751  03174 000000        OCT 0          SPARE WORD
0752  03175 000000        OCT 0          SPARE WORD
0753  03176 003166 CPNTR  DEF CNSTN      POINTER TO MULTIPLIER


0755  03177 062337 RSQR   JSM GTFLG      GET DIGIT FLAGS
0756  03200 072110        RZA RSQ1       IF VARIABLES DEFINED (ANY FLAG) - SKIP
0757  03201 166071        JMP ERSTP,I    VARIABLES NOT DEFINED - ERROR
0758  03202 062334 RSQ1   JSM FLAG3      SEE IF DIGIT-3 (3.4.5 VARIABLES) ON
0759  03203 070110        SZA RSQ2       IT ISN'T - SKIP
0760  03204 067225        JMP RSQR3      IT IS - DO 3-VARIABLE CORR. COEFF.
0761  03205 062331 RSQ2   JSM FLAG2      SEE IF DIGIT -2 ON
0762  03206 070110        SZA RSQ3       IT ISN'T - SKIP
0763  03207 067211        JMP RSQR2      IT IS - DO 2-VARIABLE CORR. COEFF.
0764  03210 064167 RSQ3   JMP MON        1-VARIABLE - GO DO SOMETHING ELSE
0765  03211 022370 RSQR2  LDA XYP        2-VARIABLE CORR.COEFF.
0766  03212 062373        JSM MU                MAKE MU-XY
```

—Continued

```
0767  03213 062707        JSM TXFR
0768  03214 160212        JSM XSQR,I
0769  03215 022365        LDA XXP
0770  03216 062373        JSM MU                  MAKE MU-XX
0771  03217 062713        JSM TDVX
0772  03220 022366        LDA YYP
0773  03221 062373        JSM MU                  MAKE MU-YY
0774  03222 062713        JSM TDVX
0775  03223 063254        JSM CLER2
0776  03224 064167        JMP MON
0777  03225 062571 RSQR3  JSM REGR4      3-VARIABLE CORR.COEFF.
0778  03226 022371        LDA XZP
0779  03227 062373        JSM MU                  MAKE MU-XY
0780  03230 063246        JSM TMPX
0781  03231 022372        LDA YZP
0782  03232 062373        JSM MU                  MAKE MU-YX
0783  03233 022324        LDA T2P
0784  03234 026321        LDB YR
0785  03235 162065        JSM FMP,I               A2*MUYZ IN Y
0786  03236 022321        LDA YR
0787  03237 026322        LDB XR
0788  03240 162067        JSM FAD,I               A1*MUXY + A2*MUYZ IN X
0789  03241 022367        LDA ZZP
0790  03242 062373        JSM MU                  MAKE MU-ZZ
0791  03243 062713        JSM TDVX
0792  03244 063254        JSM CLER2
0793  03245 064167        JMP MON
0794***
0795*
0796*
0797  03246 022324 TMPX   LDA T2P         X MULTIPLIED BY TEMP, RESULT IN X
0798  03247 026322        LDB XR
0799  03250 162065        JSM FMP,I
0800  03251 170402        RET
0801*
0802*
0803  03252 022322 CLER3  LDA XR          CLEAR X
0804  03253 170000        CLR
0805  03254 022321 CLER2  LDA YR          CLEAR Y
0806  03255 170000        CLR
0807  03256 022320 CLER1  LDA ZR          CLEAR Z
0808  03257 170000        CLR
0809  03260 170402        RET
```

—Continued

```
0811*
0812* R/W MEMORY AREA
0813*
0814  01664         TEMP   EQU 1664B
0815  01665         TABLØ  EQU TEMP+1
0816  01666         DIGIT  EQU TEMP+2
0817  01667         SHIFT  EQU TEMP+3
0818  01670         COUNT  EQU TEMP+4
0819  01671         RCPFG  EQU TEMP+5
0820  01672         FNCFG  EQU TEMP+6
0821  01673         SIGN   EQU TEMP+7
0822  01744         AR1E   EQU 1744B
0823  01754         AR2E   EQU 1754B
0824  01730         ADRTE  EQU 1730B
0825*
0826*
0827*************************************
0828*
0829***     LABELS
0830*
0831  00015         AR1    EQU 15B
0832  00016         AR2    EQU 16B
0833  00063         P1     EQU 63B
0834  00034         P3     EQU 34B
0835  00003         P4     EQU 3B
0836  00104         P10    EQU 104B
0837  00106         P12    EQU 106B
0838  00105         P256   EQU 105B
0839  00031         N1     EQU 31B
0840  00054         N4     EQU 54B
0841  00023         N5     EQU 23B
0842  00024         N10    EQU 24B
0843  00101         N256   EQU 101B
0844  00056         ONEE   EQU 56B
0845  00012         AWRDN  EQU 12B
0846  01725         KBUF   EQU 1725B
0847*
0848  01724         ADR1   EQU 1724B
0849  01720         ADR2   EQU 1720B
0850  00004         ADRX   EQU 4B
0851  00011         ADRØ   EQU 11B
0852  00013         ADRT   EQU 13B
0853  00167         SETUP  EQU 167B
0854*
0855*
0856*************************************
0857*
0858*
0859*
0860* LINKAGE TO MATH SUBROUTINES
0861*
0862  03261 004577 STMAX  OCT 4577
0863  03262 004741 ROUND  OCT 4741
0864  03263 004366 TERM   OCT 4366
0865*
0866*
```

```
0867**********************************
0868*
0869***
0870* LINKAGE TO ARITHMETIC
0871*
0872  03264 004447 FD    OCT 4447
0873*
0874*
0875**********************************
0876*
0877* LINK WORDS TO THE KEYBOARD
0878*
0879  00217        ONVX  EQU 2178
0880*
0881*
0883*
0884*
0885**********************************
0886*
0887* ROM AREA
0888*
0889*
0890  03265 000000 C1    OCT 0000000     LN(2)
0891  03266 003223       OCT 003223
0892  03267 012161       OCT 012161
0893  03270 100126       OCT 100126
0894*
0895  03271 000000       DEC 0000        LN(1.1)
0896  03272 004523       OCT 004523      0953
0897  03273 010027       OCT 010027      1017
0898  03274 114004       OCT 114004      9804
0899*
0900  03275 000000       DEC 0000        LN(1.01)
0901  03276 004625       OCT 004625      0995
0902  03277 001460       OCT 001460      0330
0903  03300 102462       OCT 102462      8532
0904*
0905  03301 000000       DEC 0000        LN(1.001)
0906  03302 004631       OCT 004631      0999
0907  03303 050003       OCT 050003      5003
0908  03304 031410       OCT 031410      3308
0909*
0910  03305 000000       DEC 0000        LN(1.0001)
0911  03306 004631       OCT 004631      0999
0912  03307 112400       OCT 112400      9500
0913  03310 001463       OCT 001463      0333
0914*
0915  03311 000000       OCT 000000      LN(1,00001)
0916  03312 004631       OCT 004631      0999
0917  03313 114520       OCT 114520      9950
0918  03314 000003       OCT 000003      0003
0919*
0920  03315 000000 C2    OCT 0000000      LN(10)
0921  03316 021402       OCT 021402      2302
0922  03317 054120       OCT 054120      5850
0923  03320 111400       OCT 111400     9300
0924*
0925***
0926*
0927*
0928*
0929*
0930*
0931  03321 000144 P100  OCT 000144
0932  03322 177634 N100  OCT 177634
0933  03323 001400 EP3   OCT 001400
0934  03324 176400 EN3   OCT 176400
0935  03325 003261 THETL DEF C1-4
0936  03326 000006 P6    DEC 6
0937  03327 000015 P13   DEC 13
0938  03330 177764 N12   DEC -12

0939  03331 000020 D0010 OCT 000020
0940  03332 003315 LN10  DEF C2
0941*
0942*
0943**********************************
0944*
0945***      PROGRAM SUBROUTINES
```

—Continued

```
0946*
0947  03333 035672 INIT  STB FNCFG
0948*
0949  03334 120004       LDA ADRX,I
0950  03335 031673       STA SIGN      SAVE ORIGINAL SIGN
0951*
0952  03336 020004 XTT   LDA ADRX
0953  03337 031720       STA ADR2
0954  03340 067417       JMP OTT+1
0955*
0956*
0957  03341 063417 NORM  JSM OTT+1
0958  03342 171450       NRM
0959  03343 074404       SBL 8          ALLIGN SHIFTS
0960  03344 074076       TCB
0961  03345 005754       ADB AR2E
0962  03346 035754       STB AR2E      STORE EXPONENT
0963  03347 020016       LDA AR2
0964  03350 170402       RET
0965*
0966*
0967  03351 025754 UNRM  LDB AR2E
0968  03352 074413       SBP ZERO
0969  03353 074340       ABR 8
0970  03354 074076       TCB
0971*
0972  03355 074117       LDA B
0973  03356 003330       ADA N12
0974  03357 070112       SAM *+2
0975  03360 027327       LDB P13
0976*
0977  03361 163262       JSM ROUND,I
0978*
0979  03362 070075 ZERO  SEC *+1,C     CLEAR E REG
0980  03363 021755       LDA AR2E+1
0981  03364 001756       ADA AR2E+2
0982  03365 001757       ADA AR2E+3
0983  03366 072154       SES *+3,S     SKIP IF NOT ZERO
0984  03367 072110       RZA *+2       SKIP IF NOT ZERO
0985  03370 070075       SEC *+1,C     CLEAR E IF ZERO
0986  03371 025754       LDB AR2E
0987  03372 170402       RET
0988*
0989*
0990*
0991*
0992*
0993*
0994  03373 020012 CLER  LDA AWRDN
```

—Continued

```
0995  03374 170000       CLR
0996  03375 000003       ADA P4
0997  03376 170000       CLR
0998  03377 000003       ADA P4
0999  03400 170000       CLR
1000  03401 000034       ADA P3
1001  03402 031666       STA DIGIT
1002  03403 170402 CK    RET
1003*
1004*
1005*
1006*
1007*
1008  03404 001665 UPTAB ADA TABL0
1009  03405 031665       STA TABL0
1010  03406 067414       JMP OTO+1
1011*
1012*
```

—Continued

```
1014  03407 020016 TTX   LDA AR2
1015  03410 024004       LDB ADRX
1016  03411 170004       XFR
1017  03412 170402       RET
1018*
1019  03413 020056 OTO   LDA ONEE
1020  03414 024015       LDB AR1
1021  03415 067411       JMP TTX+2
1022*
1023*
1024*
1025*
1026*
1027  03416 020056 OTT   LDA ONEE
1028  03417 024016       LDB AR2
1029  03420 067411       JMP TTX+2
1030*
```

—Continued

```
1032*
1033*
1034*************************************
1035*
1036***      MAIN PROGRAMS
1037*
1038*
1039*
1040*************************************
1041*
1042  03421 024063 EXP   LDB P1
1043  03422 063333       JSM INIT
1044*
1045  03423 074742       SBR 16
1046  03424 035670       STB COUNT
1047*
1048  03425 025754       LDB AR2E
1049  03426 074212       SBM *+4
1050  03427 007324       ADB EN3
1051  03430 074112       SBM *+2
1052  03431 067461       JMP STM1-1
1053*
1054  03432 023332       LDA LN10
1055  03433 024004       LDB ADRX
1056  03434 031724       STA ADR1
1057  03435 163264       JSM FD,I
1058  03436 025754       LDB AR2E
1059  03437 070742       SAR 16
```

—Continued

```
1060  03440 074252       SBM *+5
1061  03441 004101       ADB N256
1062  03442 171400       MLS
1063  03443 001670       ADA COUNT
1064  03444 070604       SAL 4
1065  03445 031670       STA COUNT
1066  03446 075553       SBP *-5
1067  03447 035754       STB AR2E
1068  03450 063342       JSM NORM+1
1069  03451 063407       JSM TTX
1070  03452 021724       LDA ADR1
1071  03453 024004       LDB ADRX
1072  03454 162065       JSM FMP,I
1073  03455 063351       JSM UNRM
1074*
1075  03456 021670       LDA COUNT
1076  03457 070542       SAR 12
1077  03460 070150       SZA LT3
1078  03461 074044       SBL 15
1079  03462 167261 STM1  JMP STMAX,I
1080*
1081  03463 025670 LT3   LDB COUNT
1082  03464 074404       SBL 8
1083  03465 074142       SBR 4
1084  03466 021670       LDA COUNT
1085  03467 070342       SAR 8
1086  03470 070344       SAL 9
1087  03471 070037       ADB A
1088  03472 070704       SAL 2
1089  03473 070037       ADB A
1090  03474 035671 NOEX  STB RCPFG
1091*
1092  03475 063373       JSM CLER
1093*
1094  03476 023325       LDA THETL
1095  03477 024012       LDB AWRDN
1096  03500 004031       ADB N1
1097  03501 035666       STB DIGIT
1098  03502 027326       LDB P6
1099  03503 031665       STA TABL0
1100  03504 035667       STB SHIFT
1101  03505 063362 PTA   JSM ZERO
1102  03506 070154       SES *+3
1103  03507 145666       ISZ DIGIT,I
1104  03510 067525       JMP PTA1
1105  03511 170400       CMY
1106  03512 045666       ISZ DIGIT
1107  03513 020003       LDA P4
1108  03514 063404       JSM UPTAB
1109  03515 074004       SBL 16
1110  03516 170420       FDV
1111  03517 135666       STB DIGIT,I
1112  03520 170400       CMY
1113  03521 170560       FXA
1114  03522 171400       MLS
1115  03523 055667       DSZ SHIFT
1116  03524 067505       JMP PTA
1117  03525 024023 PTA1  LDB N5
1118  03526 045666       ISZ DIGIT
1119  03527 171400       MLS
1120  03530 131666       STA DIGIT,I
1121  03531 077670       RIB *-3        STORE LAST 6 DIGITS
1122*
1123  03532 171400       MLS
1124  03533 000023       ADA N5
1125  03534 070135       SEC *+2,C
1126  03535 145666       ISZ DIGIT,I
1127*
1128  03536 063413       JSM OTO
1129*
1130  03537 063716       JSM PTJ
1131*
1132  03540 021671 BACK  LDA RCPFG
1133  03541 031754       STA AR2E       STORE EXPONENT
```

—Continued

```
1134*
1135  03542 171450        NRM             NORMALIZE ANSWER
1136*
1137  03543 021673        LDA SIGN
1138  03544 070151        SLA *+3
1139  03545 063407        JSM TTX
1140  03546 160217        JSM ONVX,I
1141*
1142  03547 074742        SBR 16
1143  03550 021754        LDA AR2E
1144  03551 031744        STA AR1E
1145  03552 020004        LDA ADRX
1146  03553 031720        STA ADR2
1147  03554 163263        JSM TERM,I
1148  03555 020011 RESET  LDA ADR0
1149  03556 170000        CLR
1150  03557 064167        JMP SETUP
1152*
1153*
1154**********************************
1155*
1156*
1157  03560 024031 LNX    LDB N1
1158  03561 063333        JSM INIT
1159*
1160*
1161  03562 021754        LDA AR2E
1162  03563 070171        SLA *+3,C
1163  03564 024063        LDB P1
1164  03565 167261        JMP STMAX,I
1165  03566 063362        JSM ZERO
1166  03567 071675        SEC *-3,C
1167*
1168  03570 063373        JSM CLER
1169  03571 063716        JSM PTJ
1170  03572 055666        DSZ DIGIT
1171*
1172  03573 023326        LDA P6
1173  03574 027332        LDB LN10
1174  03575 035665        STB TABL0
1175  03576 031664        STA TEMP
1176  03577 031667        STA SHIFT
1177  03600 020016        LDA AR2
1178  03601 170000        CLR
1179  03602 121666 PTP    LDA DIGIT,I
1180  03603 070076        TCA
1181  03604 024063        LDB P1
1182  03605 174470        MRY
1183  03606 055666        DSZ DIGIT
1184  03607 055667        DSZ SHIFT
1185  03610 067602        JMP PTP
1186  03611 070742        SAR 16
1187  03612 025664        LDB TEMP
1188  03613 074076        TCB
1189  03614 004106        ADB P12
1190  03615 035667        STB SHIFT
1191  03616 174470        MRY
1192  03617 020054 PTQ    LDA N4
1193  03620 063404        JSM UPTAB
1194  03621 070742        SAR 16
1195  03622 025667        LDB SHIFT
1196  03623 074650        SZB NAW
1197  03624 004031        ADB N1
1198  03625 174430        MRX
1199  03626 000023        ADA N5
1200  03627 070175        SEC *+3,C
1201  03630 020015        LDA AR1
1202  03631 170540        MDI
1203  03632 125666        LDB DIGIT,I
1204  03633 074076        TCB
1205  03634 171460        FMP
1206  03635 055666        DSZ DIGIT
1207  03636 055667        DSZ SHIFT
```

—Continued

```
1208  03637 067617        JMP PTQ
1209*
1210  03640 063407 NAW    JSM TTX
1211*
1212*
1213  03641 020016        LDA AR2
1214  03642 170000        CLR
1215*
1216  03643 021673        LDA SIGN        GET EXPONENT WORD
1217  03644 000105        ADA P256
1218  03645 070153        SAP *+3
1219  03646 070071        SLA *+1,C
1220  03647 070076        TCA             COMPLEMENT IF NEGATIVE
1221  03650 070342        SAR 8
1222*
1223  03651 074742        SBR 16
1224  03652 003322        ADA N100
1225  03653 070152        SAM *+3
1226  03654 004105        ADB P256
1227  03655 067657        JMP *+2
1228  03656 003321        ADA P100
1229*
1230  03657 000024        ADA N10
1231  03660 070152        SAM *+3
1232  03661 007331        ADB D0010
1233  03662 067657        JMP *-3
1234  03663 000104        ADA P10
1235*
1236  03664 070037        ADB A
1237  03665 035755        STB AR2E+1
1238*
1239  03666 023323        LDA EP3
1240  03667 031754        STA AR2E
1241  03670 063342        JSM NORM+1
1242*
1243  03671 027332        LDB LN10
1244  03672 162065        JSM FMP,I
1245*
1246  03673 020016        LDA AR2
1247*
1248  03674 024004        LDB ADRX
1249  03675 035724        STB ADR1
1250  03676 063341        JSM NORM
1251  03677 024013        LDB ADRT
1252  03700 170004        XFR
1253  03701 021724        LDA ADR1
1254  03702 063341        JSM NORM
1255  03703 063410        JSM TTX+1
1256*
1257  03704 145724        ISZ ADR1,I
1258*
1259  03705 021673        LDA SIGN
1260  03706 070113        SAP *+2
1261  03707 045730        ISZ ADRTE
1262  03710 020004        LDA ADRX
1263  03711 024013        LDB ADRT
1264  03712 162067        JSM FAD,I
1265  03713 020013        LDA ADRT
1266  03714 063410 OUTS   JSM TTX+1
1267  03715 067555        JMP RESET
1268*
1269*
1270**********
1271*
1272*
1273  03716 020012 PTJ    LDA AWRDN
1274  03717 031666        STA DIGIT       INITIALIZE DIGIT LOCATION
1275*
1276  03720 020106        LDA P12
1277  03721 031667        STA SHIFT       INITIALIZE COUNT
1278  03722 025672        LDB FNCFG
1279  03723 074252        SBM *+5
1280  03724 063416        JSM OTT
1281  03725 067733        JMP PTI
```

—Continued

```
1282*
1283  03726 174400 PTR   CMX
1284  03727 170560       FXA
1285  03730 020016       LDA AR2
1286  03731 063414       JSM OTO+1
1287  03732 067736       JMP PTH-2
1288*
1289  03733 121666 PTI   LDA DIGIT,I
1290  03734 072110       RZA *+2          SKIP IF DIGIT NOT ZERO
1291  03735 067761       JMP PTG
1292*
1293  03736 020104       LDA P10
1294  03737 031665       STA TABL0
1295  03740 025667 PTH   LDB SHIFT
1296  03741 074076       TCB
1297  03742 004106       ADB P12          CALC SHIFT COUNT
1298  03743 070742       SAR 16
1299  03744 174430       MRX              SHIFT AR1
1300*
1301  03745 025672       LDB FNCFG
1302  03746 014063       CPB P1           SKIP IF NOT EXP
1303  03747 000023       ADA N5
1304*
1305  03750 170560       FXA              CALC X+10↑(-J)X
1306*
1307  03751 070414       SES PTG
1308  03752 055665       DSZ TABL0
1309  03753 067755       JMP GOON
1310  03754 067761       JMP PTG
1311*
1312  03755 020016 GOON  LDA AR2
1313  03756 063414       JSM OTO+1
1314  03757 155666       DSZ DIGIT,I
1315  03760 067740       JMP PTH
1316*
1317  03761 045666 PTG   ISZ DIGIT
1318  03762 055667       DSZ SHIFT        DECREMENT COUNTER
1319  03763 067765       JMP *+2
1320  03764 170402       RET
1321*
1322*
1323  03765 025672       LDB FNCFG
1324  03766 014031       CPB N1
1325  03767 067726       JMP PTR
1326  03770 067733       JMP PTI

1328**********************************
1329*
1330  03771 063560 LOG   JSM LNX
1331  03772 023332       LDA LN10
1332  03773 024004       LDB ADRX
1333  03774 167264       JMP FD,I
1334*
1335*
1336*
1337*
1338                     END
*  NO ERRORS*
```

CROSS-REFERENCE SYMBOL TABLE

A	0000/00	0028	0032	0056	1087	1089	1236			
ADR0	0851	1148								
ADR1	0848	1056	1070	1249	1253	1257				
ADR2	0849	0953	1146							
ADRT	0852	1251	1263	1265						
ADRTE	0824	1261								
ADRX	0850	0949	0952	1015	1055	1071	1145	1248	1262	1332

CROSS-REFERENCE SYMBOL TABLE —Continued

AR	0245	0239									
AR1	0831	1020	1201								
AR1E	0822	1144									
AR1P	0010	0142	0147	0149	0152	0155	0158	0159	0162	0165	
AR2	0832	0963	1014	1028	1177	1213	1246	1285	1312		
AR2E	0823 1133	0961 1143	0962 1161	0967 1237	0980 1240	0981	0982	0986	1048	1058	1067
AWRDN	0845	0994	1095	1273							
B	0000/00	0060	0972								
BACK	1132										
BASE1	0111	0026									
BR	0244	0238									
C1	0890	0935									
C2	0920	0940									
CANCL	0106										
CHI	0629	0075	0105								
CHI2	0646	0637									
CHI3	0653	0645									
CHICL	0132	0126									
CHKMX	0698	0678	0681	0684							
CIN	0013	0202	0208	0227	0229	0699	0702	0708	0711	0712	0721
CK	1002										
CKSM1	0116										
CKSM2	0622										
CLEAR	0120	0067									
CLER	0994	1092	1168								
CLER1	0807	0453	0659								
CLER2	0805	0775	0792								
CLER3	0803	0353	0387								
CLFLG	0014 0564	0096 0575	0098 0636	0106 0638	0108 0646	0137	0139	0203	0214	0236	0562
CNSTN	0745	0753									
CORR	0096	0030									
COUNT	0818	1046	1063	1065	1075	1081	1084				
COUT	0011 0322	0199 0698	0201 0707	0211 0715	0231 0720	0233	0310	0311	0314	0315	0318
CPNTR	0753	0730									
D0010	0939	1232									
DEFAL	0617	0590									
DIGIT	0816 1183	1001 1203	1097 1206	1103 1274	1106 1289	1111 1314	1118 1317	1120	1126	1170	1179
EN3	0934	1050									
EP3	0933	1239									
ERSTP	0083	0049	0129	0356	0390	0424	0431	0757			
EX9R	0169	0146									
EXP	1042	0078									
FAD	0081	0222	0578	0581	0584	0649	0652	0788	1264		

CROSS-REFERENCE SYMBOL TABLE —Continued

FD	0872	1057	1333								
FDV	0082	0321	0345	0402	0408	0414	0452	0495	0533	0546	0598
	0605	0611	0635								
FINS	0235	0232									
FIVE	0094	0039									
FLAG1	0380	0184									
FLAG2	0261	0179	0192	0360	0394	0428	0673	0761			
FLAG3	0266	0176	0357	0391	0425	0670	0758				
FLG1M	0063	0052	0091	0380							
FLG2M	0064	0050	0089	0261							
FLG3M	0084	0055	0256	0266							
FLGMS	0072	0057	0271								
FLGWD	0003	0059	0177	0257	0262	0267	0272	0381			
FMP	0079	0213	0317	0443	0466	0489	0503	0518	0527	0732	0785
	0799	1072	1244								
FNCFG	0820	0947	1278	1301	1323						
FOUR	0016	0041	0224								
FSUB	0080	0218	0324	0342	0449	0475	0492	0506	0521	0530	0557
	0567	0570	0573	0631	0641	0644	0704	0717	0739		
FUR	0052	0042									
FVF	0050	0040									
GETKF	0006	0031									
GOON	1312	1309									
GTFLG	0271	0173	0354	0388	0422	0667	0755				
ILOOP	0203	0230									
ILOP1	0208	0204									
ILOP2	0220	0215									
INCL3	0055	0051	0053								
INIT	0947	1043	1158								
INTX	0018	0736									
K70	0112	0100	0121								
K71	0113	0102	0123								
K72	0114	0104	0125								
K73	0115	0127									
KBUF	0A46										
KELOG	0004	0035									
KEY2	0031	0038	0099	0120							
KYBUF	0007	0025									
LN10	0940	1054	1173	1243	1331						
LNX	1157	0070	1330								
LOG	1330	0077									
LT3	1081	1077									
MAX1	0682	0674									
MAX2	0679	0675									
MAX3	0676	0672									
MAXCL	0142	0128									
MAXM	0667	0076									

CROSS-REFERENCE SYMBOL TABLE—Continued

MEAN	0387	0066								
MEAN1	0409	0395								
MEAN2	0403	0396								
MEAN3	0397	0393								
MEN1	0391	0389								
MEN2	0394	0392								
MN4	0724	0710								
MON	0009	0087	0109	0136	0168	0237	0378	0415	0454	0456	0616
	0660	0685	0743	0764	0776	0793					
MU	0310	0364	0369	0374	0433	0436	0458	0463	0468	0471	0477
	0480	0483	0497	0500	0508	0592	0766	0770	0773	0779	0782
	0790										
MUTVR	0337	0365	0370	0375	0593						
MX1	0710	0706									
MX2	0723	0719									
MXA1	0670	0668									
MXA2	0673	0671									
N1	0839	1096	1157	1197	1324						
N10	0842	1230									
N100	0932	1224									
N12	0938	0973									
N256	0843	1061									
N4	0840	1192									
N5	0841	1117	1124	1199	1303						
NAW	1210	1196									
NOEX	1090										
NORM	0957	1068	1241	1250	1254						
OLOOP	0201	0234									
ONE	0021	0047									
ONEE	0844	1019	1027								
ONVX	0879	1140									
OTO	1019	1010	1128	1286	1313						
OTT	1027	0954	0957	1280							
OUTS	1266										
P1	0833	1042	1163	1181	1302						
P10	0836	1234	1293								
P100	0931	1228									
P12	0837	1189	1276	1297							
P13	0937	0975									
P256	0838	1217	1226								
P3	0834	1000									

CROSS-REFERENCE SYMBOL TABLE —Continued

P4 0835 0996 0998 1107

P6 0936 1098 1172

PGCNT 0008

PIP 0015

PLUS 0329 0326 0328

PTA 1101 1116

PTA1 1117 1104

PTG 1317 1291 1307 1310

PTH 1295 1287 1315

PTI 1289 1281 1326

PTJ 1273 1130 1169

PTP 1179 1185

PTQ 1192 1208

PTR 1283 1325

PUTIT 0456 0090 0092

R0 0280 0134 0319 0337 0400 0406 0412 0450 0531 0572 0583
0596 0603 0613 0643 0651 0656

R1 0282 0132 0409 0438 0513 0569 0580 0600 0640 0648 0653

R14 0692 0187

R2 0283 0130 0198 0566 0577

R20 0279 0182

R21 0686 0150 0683

R22 0687 0160

R23 0688 0153 0680

R24 0689 0163

R25 0690 0156 0677

R26 0691 0166

R27 0744 0731 0733 0738 0740

R3 0285 0403 0444 0522

R4 0292

R5 0286 0195

R6 0288 0397 0510

R7 0295

R8 0298

R9 0289 0190

RAND 0730 0029

RCPFG 0819 1090 1132

REG1 0425 0423

REG2 0428 0426

REGR 0422 0065

REGR1 0431 0429

REGR2 0432 0430

CROSS-REFERENCE SYMBOL TABLE —Continued

REGR3	0455	0427									
REGR4	0457	0455	0777								
RESET	1148	1267									
REST	0086	0061									
ROUND	0863	0977									
RPNTR	0012	0200	0205	0217	0221	0223	0225				
RSQ1	0758	0756									
RSQ2	0761	0759									
RSQ3	0764	0762									
RSQR	0755	0071									
RSQR2	0765	0763									
RSQR3	0777	0760									
S	0243	0226									
S0	0242	0197									
S1	0241	0194									
S2	0240	0189									
S3	0239	0186									
S4	0238	0181									
SET3	0256	0175	0669								
SETUP	0853	1150									
SHIFT	0817	1100	1115	1176	1184	1190	1195	1207	1277	1295	1318
SIGN	0821	0950	1137	1216	1259						
SQRT	0017	0599									
STAT	0025										
STLIT	0020	0086	0258								
STM1	1079	1052									
STMAX	0862	1079	1164								
SUM	0173	0074	0103	0140							
SUM1	0176	0174									
SUM12	0184	0180									
SUM13	0189	0185									
SUM2	0192	0178									
SUM3	0197	0193									
SUMCL	0137	0124									
SUMR	0199	0183	0188	0191	0196						
T2P	0250	0209	0212	0216	0220	0312	0316	0320	0323	0325	0330
	0344	0346	0439	0442	0447	0459	0464	0484	0487	0502	0504
	0514	0517	0519	0523	0526	0528	0538	0544	0559	0568	0579
	0601	0604	0606	0610	0700	0703	0705	0713	0716	0718	0783
	0797										
T3P	0251	0338	0341	0343							
TABLØ	0815	1008	1009	1099	1174	1294	1308				
TCLR	0130	0122									

CROSS-REFERENCE SYMBOL TABLE — Continued

TOVX	0544	0437	0509	0771	0774	0791				
TEMP	0814	0815	0816	0817	0818	0819	0820	0821	1175	1187
TERM	0864	1147								
TESTI	0223	0207	0219							
TESTO	0231	0228								
THETL	0935	1094								
THREE	0093	0043								
TMPX	0797	0472	0481	0780						
TOO	0089	0046								
TP2	0575	0563								
TP3	0585	0574								
TP4	0613	0621								
TPAIR	0555	0073	0101							
TREE	0054	0044								
TTX	1014	1021	1029	1069	1139	1210	1255	1266		
TWO	0069	0045								
TXFR	0538	0434	0469	0478	0498	0612	0767			
UNRM	0967	1073								
UPTAB	1008	1108	1193							
VARBL	0038	0062								
VRN1	0357	0355								
VRN2	0360	0358								
VRNC1	0373	0361								
VRNC2	0368	0362								
VRNC3	0363	0359								
VRNCE	0353	0068								
WON	0091	0048								
WONE	0249	0243	0340	0571	0582	0642	0650			
XEQFL	0005	0033								
XR	0248	0242	0376	0410	0413	0441	0473	0491	0493	0505
	0539	0545	0556	0558	0565	0576	0594	0597	0609	0617
	0634	0639	0647	0654	0682	0734	0737	0741	0787	0798
XSQR	0019	0561	0632	0768						
XTT	0952									
XX	0281	0300								
XXP	0300	0373	0435	0462	0476	0507	0591	0769		
XY	0290	0303								
XYP	0303	0432	0470	0482	0499	0765				
XZ	0293	0304								
XZP	0304	0467	0496	0778						
YR	0247	0241	0371	0404	0407	0445	0448	0451	0460	0465
	0494	0501	0525	0555	0614	0629	0633	0657	0679	0784
	0805									
YY	0284	0301								
YYP	0301	0368	0479	0772						
YZ	0296	0305								
YZP	0305	0457	0781							

CROSS-REFERENCE SYMBOL TABLE — Continued

ZERO	0979	1101	1165								
ZR	0246	0240	0366	0398	0401	0485	0488	0490	0511	0520	0529
	0532	0607	0619	0676	0807						
ZZ	0287	0302									
ZZP	0302	0363	0789								

ASMB,A,B,C,L,T — Continued

TEMP	001664		RESET	002004
TABLØ	001665		STMAX	002007
LIMIT	001666		FALSE	002010
SHIFT	001667		ROUND	002011
COUNT	001670		TERM	002012
RCPFG	001671		FD	002013
FNCFG	001672		FMP	002014
SIGN	001673		FAD	002015
SAVE	001674		FSB	002016
RY2	001675		FSR	000276
RY1	001676		ONVX	000217
GFG	001677		XSQ	000212
AR1E	001744		UP	000227
AR2E	001754		XEY	000230
ADRXE	001750		INTX	000264
ADRTE	001730		YMX	000234
ADRPE	001672		FMT	000242
ADRYE	001740		MULT	000236
AR1	000015		FSN	000232
AR2	000016		CLX	000237
P1	000063		C8	002031
P2	000772		C2	002065
P3	000034		EP3	002071
P4	000003		C4	002124
P7	000770		C9	002130
P10	000104		C5	002133
P12	000106		C7	002137
P22	000074		N108	002143
P256	000105		END	002144
N1	000031		TWOPI	002145
N4	000054		PI/2	002146
N5	000023		DMERR	002147
N10	000024		GPERR	002152
N256	000101		CONV	002153
EN2	000032		P100	002154
ONE	000056		N100	002155
AWRDN	000012		FNS	002156
PI	000064		EN3	002157
RETC	000100		EN12	002160
ASNC	000064		ENDA	002161
ADR1	001724		THETT	002162
ADR2	001720		P6	002163
ADRX	000004		P11	002164
ADRY	000005		P13	002165
ADRØ	000011		CLXC	002166
ADRT	000013		P34	002167
ADRA	000007		N2	002170
ADRB	000010		N12	002171
ADRF	000033		N13	002172
ADRZ	000006		N6	002173
UNITS	001776		CKSM1	002174
KBUF	001725		DMA1Ø	002175
ERR	000066		ERROR	002176
EXCF	000103		XMIN	002177
USP	001710		YMIN	002200
EXEC	000132		RADF	002201
SUBF	001721		GRAF	002202
NXCD	000115		UNITM	002203
INTP	000362		FINAL	002204
SETUP	000167		ACSC	002205
ZZAD	001707		ATNC	002206
KYLOG	000113		LN1Ø	002207
JTA	002002		ADRP	002211

—Continued

SRCH	002212
GSBA	002217
USERC	002222
INIT	002231
XTT	002234
DIV	002237
NORM	002246
FRAC	002256
UNRM	002276
ZERO	002307
SCRT	002320
SCRT1	002336
XAT	002343
CK	002345
STN	002346
UPTAB	002353
NXCD1	002356
OTO	002363
P2TO	002366
TTR1	002370
XTP	002373
RITT	002376
YTP	002401
OTT	002403
PTX	002405
TTO	002407
SIN	002411
CS	002426
COS	002440
ARC	002445
ACS	002455
ACSS	002463
ASN	002464
TAN	002466
GET1	002473
FAL1	002514
GET	002531
OUT	002614
ASNS	002620
ATNS	002633
OK	002645
POOT	002670
GONO	002702
SOUT	002723
LOG	002705
RETA	002711
UPARO	002712
REXP	002717
EXP	002722
STM1	002737
LT3	002740
BACK1	002776
LNX	003000
OUTS	003062
XFACT	003063
FAC3	003072
IN	003101
PTA	003103
IN2	003124
PTL1	003135
PTB	003136
PTD	003140
PTE	003151
PTN	003152
PTF1	003157
AWRS	003164
PTL	003166
PTC	003173

—Continued

PTM	003174
PTK	003225
PTF	003231
PTJ	003234
PTR	003243
PTS	003247
PTI	003251
PTH	003255
GOON	003272
PTG	003275
IN3	003301
PTP	003306
PTQ	003323
NAW	003344
IOO	003345
RET2	003352
ATN	003353
EATN	003354
MODE	003355
RTOD	003371
PXX	003374
DTOR	003376
RTOG	003400
GTOR	003402
PRE	003405
ABSX	003406
TENX	003416
ABS	003420
SCALE	003422
SCL1	003431
FDV	003450
DMSTD	003452
DTOMS	003465
DSUB	003466
XAM	003473
RCL	003475
ACM	003502
ACP	003510
TP	003516
NOMO	003554
TR	003561
TCS	003576
NZ	003577
CLX1	003601
RNDY	003602
TTX	003611
MET	003631
RND2	003654
F	003660
CLRR	003674
DEG	003677
RET3	003707
ADRL	003710
RAD	003677
GRA	003677
XAA	003712
USER	003714
DULUP	003733
DL1	003760
DL2	003761
THETL	002152
FTABL	002152
LINK	002146
CKSM2	003767
CLER	003770
KK	003776

* NO ERRORS*

```
            MATH BLOCK FOR THE 9810, CALLED 'CORD3'

0001                ASMB,A,B,C,L,T
0002*
0004*
0005*       MIKE WINGERT
0006*
0007*************************************
0008*
0009  02000             ORG 2000B      OPTION BLOCK ORIGIN
0010*
0011*************************************
0012*
0013* R/W MEMORY AREA
0014*
0015  01664         TEMP  EQU 1664B      TEMPORARY
0016  01665         TABL0 EQU TEMP+1     CORDIC CONSTANT ADDRESSES
0017  01666         DIGIT EQU TEMP+2     DIGIT STACK POINTER
0018  01667         SHIFT EQU TEMP+3     SHIFT FACTOR
0019  01670         COUNT EQU TEMP+4     BCD STORAGE
0020  01671         RCPFG EQU TEMP+5     RECIPROCAL FLAG
0021  01672         FNCFG EQU TEMP+6     FUNCTION FLAG
0022  01673         SIGN  EQU TEMP+7     EXPONENT WORD STORAGE
0023  01674         SAVE  EQU TEMP+8     TEMPORARY
0024  01675         RY2   EQU TEMP+9     REGISTER ADDRESS STORAGE
0025  01676         RY1   EQU TEMP+10    REGISTER ADDRESS STORAGE
0026  01677         GFG   EQU TEMP+11    G FLAG
0027  01744         AR1E  EQU 1744B      EXPONENT WORD OF AR1
0028  01754         AR2E  EQU 1754B      EXPONENT WORD OF AR2
0029  01750         ADRXE EQU 1750B      EXPONENT WORD OF X
0030  01730         ADRTE EQU 1730B      EXPONENT WORD OF T
0031  01672         ADRPE EQU 1672B      EXPONENT WORD OF P
0032  01740         ADRYE EQU 1740B      EXPONENT WORD OF Y

0034*
0035*
0036*************************************
0037*
0038***     LABELS
0039*
0040  00015         AR1   EQU 15B        ADDRESS OF AR1
0041  00016         AR2   EQU 16B        ADDRESS OF AR2
0042  00063         P1    EQU 63B        POSITIVE ONE
0043  00772         P2    EQU 772B       POSITIVE TWO
0044  00034         P3    EQU 34B        POSITIVE THREE
0045  00003         P4    EQU 3B         POSITIVE FOUR
0046  00770         P7    EQU 770B       POSITIVE SEVEN
0047  00104         P10   EQU 104B       POSITIVE TEN
0048  00106         P12   EQU 106B       POSITIVE TWELVE
0049  00074         P22   EQU 74B        POSITIVE 22
0050  00105         P256  EQU 105B       POSITIVE 256
0051  00031         N1    EQU 31B        NEGATIVE ONE
0052  00054         N4    EQU 54B        NEGATIVE FOUR
0053  00023         N5    EQU 23B        NEGATIVE FIVE
0054  00024         N10   EQU 24B        NEGATIVE TEN
0055  00101         N256  EQU 101B       NEGATIVE 256
0056  00032         EN2   EQU 32B        NEGATIVE EXPONENT TWO
0057  00056         ONE   EQU 56B        ADDRESS OF FLOATING ONE
0058  00012         AWRDN EQU 12B        ADDRESS OF DIGIT STACK
0059  00064         PI    EQU 64B        ADDRESS OF PI
0060  00100         RETC  EQU 100B       RETURN CODE (OCT 77)
0061  00064         ASNC  EQU 64B        SINE CODE (OCT 70)
0062  01724         ADR1  EQU 1724B      ADDRESS OF SECOND OPERAND
0063  01720         ADR2  EQU 1720B      ADDRESS OF FIRST OPERAND
0064  00004         ADRX  EQU 4B         ADDRESS OF X
0065  00005         ADRY  EQU 5B         ADDRESS OF Y
0066  00011         ADR0  EQU 11B        ADDRESS OF THETA
0067  00013         ADRT  EQU 13B        ADDRESS OF T
0068  00007         ADRA  EQU 7B         ADDRESS OF A
0069  00010         ADRB  EQU 10B        ADDRESS OF B
0070  00033         ADRF  EQU 33B        ADDRESS OF REGISTER 0
0071  00006         ADRZ  EQU 6B         ADDRESS OF Z
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0072  01776          UNITS EQU 17768      STATUS WORD
0073  01725          KBUF  EQU 17258      KEYCODE BUFFER
0074  00066          ERR   EQU 66B        EEROR BIT MASK
0075  00103          EXCF  EQU 103B       EXECUTION FLAG
0076  01710          USP   EQU 17108      USER STACK POINTER
0077  00132          EXEC  EQU 132B       EXECUTER
0078  01721          SUBF  EQU 1721B      SUBROUTINE FLAG
0079  00115          NXCD  EQU 115B       NEXT CODE ROUTINE
0080  00362          INTP  EQU 362B       INTERPRETER
0081  00167          SETUP EQU 167B       SETUP ROUTINE
0082  01707          ZZAD  EQU 1707B      BEGINNING OF PROGRAM MEMORY
0083  00113          KYLOG EQU 113B       KEYLOG ROUTINE

0085*
0086*
0087*************************************
0088*
0089* LINKAGE TO THE MAINLINE ROUTINES
0090*
0091  02000 021725         LDA KBUF       LOAD THE KEYCODE
0092  02001 002146         ADA LINK       ADD THE LINK WORD TO IT
0093  02002 070517 JTA     LDA A,I        LOAD THE ADDRESS OF THE ROUTINE
0094  02003 070717         JSM A,I        EXECUTE THE KEYCODE ROUTINE
0095  02004 020011 RESET   LDA ADR0       CLEAR THE SYSTEM
0096  02005 170000         CLR              FLAG AREA
0097  02006 064167         JMP SETUP      RETURN THRU THE SETUP ROUTINE
0098*
0099*
0100*************************************
0101*
0102* LINKAGE TO MATH SUBROUTINES
0103*
0104  02007 004577 STMAX OCT 4577         LINK TO THE STORE MAXIMUM NO. ROUTINE
0105  02010 004371 FALSE OCT 4371         LINK TO THE STORE ZERO ROUTINE
0106  02011 004741 ROUND OCT 4741         LINK TO THE ROUND ROUTINE
0107  02012 004366 TERM  OCT 4366         LINK TO THE CALCULATE EXPONENT ROUTINE
0108*
0109*
0110*************************************
0111*
0112* LINKAGE TO ARITHMETIC
0113*
0114  02013 004447 FD    OCT 4447         LINK TO THE FLOATING DIVIDE ROUTINE
0115  02014 004374 FMP   OCT 4374         LINK TO THE FLOATING MULTIPLY ROUTINE
0116  02015 004271 FAD   OCT 4271         LINK TO THE FLOATING ADD ROUTINE
0117  02016 004257 FSB   OCT 4257         LINK TO THE FLOATING SUBTRACT ROUTINE

0119*
0120*
0121*************************************
0122*
0123* LINK WORDS TO THE KEYBOARD
0124*
0125  00276          FSR   EQU 276B       SQUARE ROOT
0126  00217          ONVX  EQU 217B       ONE OVER X
0127  00212          XSQ   EQU 212B       X SQUARED
0128  00227          UP    EQU 227B       UP
0129  00230          XEY   EQU 230B       X EXCHANGE Y
0130  00264          INTX  EQU 264B       INTEGER X
0131  00234          YMX   EQU 234B       Y MINUS X
0132  00242          FMT   EQU 242B       FORMAT
0133  00236          MULT  EQU 236B       Y TIMES X
0134  00232          FSN   EQU 232B       CHANGE SIGN
0135  00237          CLX   EQU 237B       CLEAR X

0137*
0138*
0139*************************************
0140*
0141* F() TABLE
0142*
0143  02017 003707         DEF RET3       RETURN (TABLE 0)
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0144  02020 003677        DEF DEG        DEGREES
0145  02021 003677        DEF RAD        RADIANS
0146  02022 003677        DEF GRA        GRADS
0147  02023 002705        DEF LOG        LOGRITHMN BASE 10
0148  02024 003416        DEF TENX       TEN TO THE POWER
0149  02025 003452        DEF DMSTD      DEGREES, MINUTES, SECONDS TO DEGREES
0150  02026 003465        DEF DTDMS      DEGREES TO DEGREES, MINUTES, SECONDS
0151  02027 003063        DEF XFACT      X FACTORIAL
0152  02030 003602        DEF RNDY       ROUND Y TO THE POWER IN X

0154*
0155*
0156**********************************
0157*
0158* CONSTANT AREA
0159*
0160  02031 000400 C8     OCT 000400     200/PI
0161  02032 061546        OCT 061546     6366
0162  02033 014567        OCT 014567     1977
0163  02034 021545        OCT 021545     2365
0164*
0165  02035 000000        OCT 000000     LN(2)
0166  02036 003223        OCT 003223     6931
0167  02037 012161        OCT 012161     4718
0168  02040 100126        OCT 100126     8056
0169*
0170  02041 000000        DEC 0000       LN(1.1)
0171  02042 004523        OCT 004523     0953
0172  02043 010027        OCT 010027     1017
0173  02044 114004        OCT 114004     9804
0174*
0175  02045 000000        DEC 0000       LN(1.01)
0176  02046 004625        OCT 004625     0995
0177  02047 001460        OCT 001460     0330
0178  02050 102462        OCT 102462     8532
0179*
0180  02051 000030        DEC 0000       LN(1.001)
0181  02052 004631        OCT 004631     0999
0182  02053 050003        OCT 050003     5003
0183  02054 031410        OCT 031410     3308
0184*
0185  02055 000000        DEC 0000       LN(1.0001)
0186  02056 004631        OCT 004631     0999
0187  02057 112400        OCT 112400     9500
0188  02060 001463        OCT 001463     0333
0189*
0190  02061 000000        OCT 000000     LN(1,00001)
0191  02062 004631        OCT 004631     0999
0192  02063 114520        OCT 114520     9950
0193  02064 000003        OCT 000003     0003
0194*
0195  02065 000000 C2     OCT 0000000     LN(10)
0196  02066 021402        OCT 021402     2302
0197  02067 054120        OCT 054120     5850
0198  02070 111400        OCT 111400     9300
0199*
0200***
0201*
0202  02071 001400 EP3    OCT 001400     POSITIVE 9999
0203  02072 114631        OCT 114631     9999
0204  02073 000000        OCT 0          0000
0205*
0206  02074 000000        DEC 0000       ARCTAN(1)
0207  02075 003605        OCT 003605     0785
0208  02076 034601        OCT 034601     3981
0209  02077 061500        OCT 061500     6340
0210*
0211  02100 000030        DEC 0000       ARCTAN(.1)
0212  02101 004626        OCT 004626     0996
0213  02102 056145        OCT 056145     6865
0214  02103 022221        OCT 022221     2491
0215*
0216  02104 000000        DEC 0000       ARC TAN (.01)
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3' —Continued

```
0217  02105 004631          OCT 004631     09999
0218  02106 113146          OCT 113146     9666
0219  02107 064147          OCT 064147     6867
0220*
0221  02110 000000          DEC 0000       ARCTAN(.001)
0222  02111 004631          OCT 004631     0999
0223  02112 114626          OCT 114626     9996
0224  02113 063147          OCT 063147     6667
0225*
0226  02114 000000          OCT 000000     ARCTAN(.0001)
0227  02115 004631          OCT 004631     0999
0228  02116 114631          OCT 114631     9999
0229  02117 113147          OCT 113147     9667
0230*
0231  02120 000000          OCT 000000     ARCTAN(.00001)
0232  02121 004631          OCT 004631     0999
0233  02122 114631          OCT 114631     9999
0234  02123 114627          OCT 114627     9997
0235*
0236  02124 000000 C4       OCT 0000000      2PI
0237  02125 061203          OCT 061203      6283
0238  02126 014123          OCT 014123      1853
0239  02127 003440          OCT 003440      0720
0240*
0241  02130 000400 C9       OCT 000400     SIXTY
0242  02131 060000          OCT 060000     6000
0243  02132 000000          OCT 0          0000
0244*
0245  02133 000000 C5       OCT 0000000    PI/2
0246  02134 012560          OCT 12560      1570
0247  02135 074543          OCT 074543     7963
0248  02136 023200          OCT 023200     2680
0249*
0250  02137 000400 C7       OCT 000400     180/PI
0251  02140 053451          OCT 053451     5729
0252  02141 053571          OCT 053571     5779
0253  02142 050450          OCT 050450     5128
0254*
0255*
0256  02143 177624 N108     DEC -108       NEGATIVE 108
0257  02144 000046 END      OCT 46         END CODE
0258  02145 002124 TWOPI    DEF C4         ADDRESS OF 2PI
0259  02146 002133 PI/2     DEF C5         ADDRESS OF PI/2
0260  02147 002137 DPERR    DEF C7         DEGREES TO RADIANS CONSTANT ADDRESS
0261  02150 003420          DEF ABS        ABSOLUTE VALUE...KC=15
0262  02151 003714          DEF USER       USER DEFINED FUNCTION...KC=16
0263  02152 002031 GPERR    DEF C8         GRADS TO RADIANS CONSTNAT ADDRESS
0264  02153 002130 CONV     DEF C9         ADDRESS OF FLOATING 60
0265  02154 000144 P100     OCT 000144     POSITIVE 144
0266  02155 177634 N100     OCT 177634     NEGATIVE 100
0267  02156 002071 FNS      DEF EP3        ADDRESS OF FLOATING 9999
0268  02157 176400 EN3      OCT 176400     EXPONENT NEGATIVE 3
0269  02160 172000 EN12     OCT 172000     EXPONENT NEGATIVE 12
0270  02161 002144 ENDA     DEF END        END CODE ADDRESS
0271  02162 002064 THETT    DEF C2-1       TRIG CONSTANT TABLE
0272  02163 000006 P6       DEC 6          POSITIVE SIX
0273  02164 000013 P11      DEC 11         POSITIVE 11
0274  02165 000015 P13      DEC 13         POSITIVE 13
0275  02166 000037 CLXC     OCT 37         CLEAR X CODE
0276  02167 000042 P34      DEC 34         POSITIVE 34
0277  02170 177776 N2       DEC -2         NEGATIVE 2
0278  02171 177764 N12      DEC -12        NEGATIVE 12
0279  02172 177763 N13      DEC -13        NEGATIVE 13
0280  02173 177772 N6       DEC -6         NEGATIVE 6
0281*
0282  02174 000000 CKSM1    NOP        CHECKSUM WORD FOR THE FIRST 512 WORDS
0283*
0284  02175 000020 D0010    OCT 000020     BCD 0010
0285  02176 004750 ERROR    OCT 4750       ERROR ROUTINE ADDRESS
0286  02177 001654 XMIN     OCT 1654       XMIN ADDRESS (R1)
0287  02200 001644 YMIN     OCT 1644       YMIN ADDRESS (R3)
0288  02201 040000 RADF     OCT 040000     RADIAN FLAG BIT
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0289   02202 004000 GRAF   OCT 004000     GRAD FLAG BIT
0290   02203 113777 UNITM  OCT 113777     OPTION LIGHT MASK
0291   02204 001664 FINAL  OCT 1664       R REGISTER END
0292   02205 000073 ACSC   OCT 73         COSINE KEYCODE
0293   02206 000071 ATNC   OCT 71         TANGENT KEYCODE
0294   02207 002065 LN10   DEF C2         ADDRESS OF LN(10)
0295   02210 003660        DEF F          FUNCTION TABLE...KC=55
0296   02211 001672 ADRP   DEF ADRPE      ADDRESS OF P
0297   02212 005152 SRCH   OCT 5152       SEARCH LABEL ROUTINE
0298   02213 003510        DEF ACP        ACCUMULATE PLUS...KC=60
0299   02214 003475        DEF RCL        RECALL...KC=61
0300   02215 003516        DEF TP         TO POLAR...KC=62
0301   02216 003502        DEF ACM        ACCUMULATE MINUS...KC*63
0302   02217 005235 GSBA   OCT 5235       GO SUB ROUTINE
0303   02220 003000        DEF LNX        NATURAL LOG...KC=65
0304   02221 003561        DEF TR         TO RECTANGULAR...KC=66
0305   02222 000016 USERC  OCT 16         DEFINABLE F(X) KEYCODE
0306   02223 002411        DEF SIN        SINE...KC=70
0307   02224 002466        DEF TAN        TNAGENT...KC=71
0308   02225 002445        DEF ARC        ARC...KC=72
0309   02226 002440        DEF COS        COSINE...KC=73
0310   02227 002712        DEF UPAR0      X TO THE POWER Y...KC=74
0311   02230 002722        DEF EXP        E TO THE POWER X...KC=75

0313*
0314*
0315*************************************
0316*
0317***      PROGRAM SUBROUTINES
0318*
0319*
0320***      INITIALIZE
0321*
0322   02231 035672 INIT   STB FNCFG      STORE THE FUNCTION FLAG
0323   02232 120004        LDA ADRX,I     SAVE THE EXPONENT
0324   02233 031673        STA SIGN         WORD OF THE ARGUMENT
0325*
0326   02234 020004 XTT    LDA ADRX       SAVE THE ADDRESS OF
0327   02235 031720        STA ADR2         THE ARGUMENT
0328   02236 066377        JMP R1TT+1     LOAD THE ARGUMENT INTO AR2
0329*
0330*************************************
0331*
0332***      NORMALIZE TWO FLOATING POINT NUMBERS
0333*
0334   02237 035724 DIV    STB ADR1       STORE THE ADDRESS OF THE SECOND NUMBER
0335   02240 062246        JSM NORM       NORMALIZE THE FIRST NO. AND
0336   02241 024013        LDB ADRT         TRANSFER IT
0337   02242 170004        XFR              INTO RESERVED LOCATION T
0338   02243 021724        LDA ADR1       LOAD THE ADDRESS OF THE SECOND NO.
0339   02244 062246        JSM NORM       NORMALIZE IT AND
0340   02245 067612        JMP TTX+1        TRANSFER IT INTO X
0341*
0342*************************************
0343*
0344***      NORMALIZE AND CORRECT THE EXPONENT
0345*
0346   02246 062377 NORM   JSM R1TT+1     TRANSFER THE SPECIFIED NO. INTO AR2
0347   02247 171450        NRM            NORMALIZE IT
0348   02250 074404        SBL 8          ADD THE NEGATIVE
0349   02251 074076        TCB              NO. OF SHIFTS
0350   02252 035754        ADB AR2E         OT THE EXPONENT WORD
0351   02253 035754        STB AR2E         AND RESTORE IT
0352   02254 020016        LDA AR2        LOAD THE ADDRESS OF AR2
0353   02255 170402        RET            RETURN

0355*
0356*************************************
0357*
0358***      FIND THE FRACTIONAL PART OF ONE NO. OF A SPECIFIED QUANTITY
0359*
0360   02256 063403 FRAC   JSM GTOR+1     DIVIDE THE NO. SPECIFIED IN B BY THAT IN A
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3' —Continued

```
0361*
0362  02257 025754        LDB AR2E      LOAD THE EXPONENT WORD
0363  02260 070742        SAR 16        CLEAR THE A REGISTER
0364  02261 031670        STA COUNT     CLEAR THE INTEGER STORAGE WORD
0365  02262 074252        SBM *+5       SKIP IF THE EXPONENT IS NEGATIVE
0366*
0367  02263 004101        ADB N256      SUBTRACT 1 FROM THE EXPONENT
0368  02264 171400        MLS           LEFT SHIFT THE MANTISSA
0369  02265 001670        ADA COUNT     SAVE THE SHIFTED DIGIT AND
0370  02266 070604        SAL 4           ALLIGN IT AND
0371  02267 031670        STA COUNT       STORE IT
0372  02270 075553        SBP *-5       SKIP IF THE EXPONENT IS STILL NEGATIVE
0373  02271 035754        STB AR2E      STORE THE NEW EXPONENT
0374*
0375  02272 062247        JSM NORM+1    NORMALIZE THE FRACTION JUST CALCULATED
0376*
0377  02273 063611        JSM TTX       TRANSFER IT INTO X
0378  02274 021724        LDA ADR1      MULTIPLY IT BY THE CONSTANT
0379  02275 067473        JMP XAM         ORIGINALLY GIVEN IN A
0380*
0381************************************
0382*
0383***     UNNORMALIZE A NUMBER AND ROUND IT UP
0384*
0385  02276 025754 UNRM   LDB AR2E      SKIP IF THE EXPONENT
0386  02277 074413        SBP ZERO        IS NON-NEGATIVE
0387  02300 074340        ABR 8         ALLIGN AND
0388  02301 074076        TCB             COMPLEMENT THE EXPONENT
0389*
0390  02302 074117        LDA B         IF THE EXPONENT IS
0391  02303 002171        ADA N12         GREATER THAN 12 IN
0392  02304 070112        SAM *+2         MAGNITUDE REPLACE IT
0393  02305 026165        LDB P13         WITH 13
0394*
0395  02306 162011        JSM ROUND,I   SHIFT AND ROUND AR2 THHIS MANY TIMES
0397*
0398************************************
0399*
0400***     DETERMINE WHETHER AR2 IS ZERO
0401*
0402  02307 070075 ZERO   SEC *+1,C     CLEAR THE OVERFLOW BIT
0403  02310 021755        LDA AR2E+1    ADD THE THREE MANTISSA
0404  02311 001756        ADA AR2E+2      WORDS TOGETHER INTO THE
0405  02312 001757        ADA AR2E+3      A REGISTER
0406  02313 072154        SFS *+3,S     SET THE OVERFLOW AND JUMP IF THE MANTISSA IS NONZERO
0407  02314 072110        RZA *+2       JUMP IF THE MANTISSA IS NONZERO (SECOND CHECK)
0408  02315 070075        SEC *+1,C     THE MANTISSA IS ZERO - CLEAR THE OVERFLOW BIT
0409  02316 025754        LDB AR2E      LOAD THE EXPONENT WORD
0410  02317 170402        RET           RETURN
0411*
0412************************************
0413*
0414***     CALCULATE SQR(1+OR-XX)
0415*
0416  02320 031670 SCRT   STA COUNT     STORE THE CODE WORD
0417*
0418  02321 062343        JSM XAT       STORE THE NO.
0419  02322 170004        XFR             IN T
0420*
0421  02323 021750        LDA ADRXE     LOAD THE EXPONENT WORD
0422  02324 002202        ADA GRAF      REPLACE THE NO. WITH
0423  02325 070113        SAP *+2         ZERO IT ITS EXPONENT
0424  02326 162010        JSM FALSE,I     IS LESS THAN -8
0425  02327 160212        JSM XSQ,I     SQUARE THE NUMBER
0426*
0427  02330 021670        LDA COUNT     NEGATE THE NUMBER IF
0428  02331 070111        SLA *+2         THE LSB OF THE CODE
0429  02332 144004        ISZ ADRX,I      WORD IS 1
0430  02333 020056        LDA ONE       ADD THE NUMBER
0431  02334 063712        JSM XAA         TO ONE
0432*
0433  02335 160276        JSM FSR,I     TAKE THE SQUARE ROOT OF THIS QUANTITY
0434*
0435  02336 021670 SCRT1  LDA COUNT     LOAD THE CODE WORD
0436  02337 070213        SAP *+4       SKIP IF THE MSB OF THE CODE WORD IS ZERO
0437  02340 020013        LDA ADRT      LET A=ADDRESS OF T
0438  02341 024004        LDB ADRX      LET B=ADDRESS OF X
0439  02342 170402        RET           RETURN
0440  02343 020004 XAT.   LDA ADRX      LET A=ADDRESS X
0441  02344 024013        LDB ADRT      LET B=ADDRESS T
0442  02345 170402 CK     RET           RETURN
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3' —Continued

```
0444*
0445**********************************
0446*
0447***     TRIG PREFIX TO THE INITIALIZE ROUTINE
0448*
0449  02346 074742 STN   SBR 16        CLEAR THE
0450  02347 035664       STB TEMP        TEMP FLAG
0451  02350 035677       STB GFG         AND THE GFG FLAG
0452  02351 062231       JSM INIT      LOAD THE ARGUMENT AND SAVE ITS EXPONENT WORD
0453  02352 067355       JMP MODE      CONVERT THE ARGUMENT TO THE PROPER UNITS AND RETURN
0454*
0455**********************************
0456*
0457***     FIND AND FETCH THE NEXT CORDIC CONSTANT
0458*
0459  02353 001665 UPTAH ADA TABL0      CALCULATE THE NEXT CONSTANT ADDRESS
0460  02354 031665       STA TABL0      STORE IT AWAY
0461  02355 066364       JMP OTO+1      TRANSFER IT INTO AR1 AND RETURN
0462*
0463**********************************
0464*
0465***     FETCH THE NEXT KEYCODE
0466*
0467  02356 060115 NXCD1 JSM NXCD       GET THE NEXT KEYCODE
0468  02357 070137       LDB A          DUPLICATE IT
0469  02360 120103       LDA EXCF,I     RETURN IF NOT IN THE
0470  02361 073210       RZA CK           KEYLOG MODE
0471  02362 164113       JMP KYLOG,I    PRINT THE CODE AND RETURN

0473*
0474**********************************
0475*
0476* FLOATING POINT NUMBER TRANSFER AREA
0477*
0478  02363 020056 OTO   LDA ONE        TRANSFER ONE
0479  02364 024015       LDB AR1          INTO AR1
0480  02365 067613       JMP TTX+2        AND RETURN
0481*
0482  02366 022146 P2TO  LDA PI/2       TRANSFER PI/2
0483  02367 066364       JMP OTO+1        INTO AR1 AND RETURN
0484*
0485*
0486  02370 020016 TTR1  LDA AR2        TRANSFER AR2
0487  02371 025676       LDB RY1          INTO RY1
0488  02372 067613       JMP TTX+2        AND RETURN
0489*
0490  02373 020004 XTP   LDA ADRX       TRANSFER X
0491  02374 026211       LDB ADRP         INTO P
0492  02375 067613       JMP TTX+2        AND RETURN
0493*
0494  02376 021676 R1TT  LDA RY1        TRANSFER RY1
0495  02377 024016       LDB AR2          INTO AR2
0496  02400 067613       JMP TTX+2        AND RETURN
0497*
0498  02401 020005 YTP   LDA ADRY       TRANSFER Y
0499  02402 066374       JMP XTP+1        INTO P AND RETURN
0500*
0501  02403 020056 OTT   LDA ONE        TRANSFER ONE
0502  02404 066377       JMP R1TT+1       INTO AR2 AND RETURN
0503*
0504  02405 022211 PTX   LDA ADRP       TRANSFER P INTO
0505  02406 067612       JMP TTX+1        X AND RETURN
0506*
0507  02407 020016 TTO   LDA AR2        TRANSFER AR2
0508  02410 066364       JMP OTO+1        INTO AR1

0510*
0511*
0512**********************************
0513*
0514***     MAIN PROGRAMS
0515*
0516*
0517***     CALCULATE THE SINE OF X
0518*
0519  02411 062346 SIN   JSM STN        INITIALIZE FLAGS AND FETCH THE ARGUMENT IN RADIANS
0520  02412 045664       ISZ TEMP       SET THE TEMP FLAG = 1
0521  02413 062467       JSM TAN+1      CALCULATE THE TANGENT
0522*
0523  02414 031674       STA SAVE       SAVE THE FLAG IN A
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0524  02415 072150        RZA *+3       JUMP IF THE ARGUMENT WAS NOT A MULTIPLE OF PI/2
0525  02416 062403        JSM OTT       LOAD ONE INTO AR2
0526  02417 066426        JMP CS        JUMP TO DETERMINE ITS SIGN
0527  02420 070742        SAR 16        LET THE CODE WORD BE ZERO
0528  02421 062320        JSM SCRT      JUMP TO CALCULATE SRT(XX+1)
0529*
0530  02422 063450        JSM FDV       CALCULATE X/SRT(XX+1)
0531*
0532  02423 020004        LDA ADRX      SAVE THE SQUARE ROOT
0533  02424 026204        LDB FINAL       QUANTITY TO BE USED
0534  02425 170004        XFR             IN THE TO RECT ROUTINE
0535*
0536  02426 021754 CS     LDA AR2E      LOAD THE EXPONENT WORD
0537  02427 072051        SLA *+1,S     SET THE MANTISSA SIGN
0538  02430 025671        LDB RCPFG     LOAD THE RECIPROCAL FLAG
0539  02431 006170        ADB N2        DETERMINE WHETHER IT
0540  02432 074702        SHR 15          WAS 0, 1 OR 2,3 AND SET B=0 OR 1
0541  02433 005673        ADB SIGN      ADD THE ORIGINAL ARGUMENT SIGN TO THIS
0542  02434 074111        SLB *+2       IF THE RESULT IS ONE
0543  02435 070071        SLA *+1,C       THE FINAL ANSWER WILL BE ZERO, ELSE ONE
0544  02436 074742        SBR 16        CLEAR THE B REGISTER
0545  02437 066614        JMP OUT       STORE THE ANSWER
0546*
0547**********************************
0548*
0549***     CALCULATE THE COSINE OF X
0550*
0551  02440 062346 COS    JSM STN       INITIALIZE THE FLAGS AND FETCH THE ARGUMENT IN RADIA
0552  02441 022146        LDA PI/2      ADD PI/2 TO
0553  02442 063712        JSM XAA       THE ARGUMENT
0554  02443 062232        JSM XTT-2     SET UP THE ARGUMENT + ADDRESSES
0555  02444 066412        JMP SIN+1     CALCULATE THE SINE OF THIS QUANTITY
0557*
0558**********************************
0559*
0560***     PROCESS THE ARC KEYCODE
0561*
0562  02445 062356 ARC    JSM NXCD1     FETCH THE NEXT KEYCODE
0563*
0564  02446 014064        CPB ASNC      IF IT IS SIN
0565  02447 066464        JMP ASN         JUMP TO THE ARCSINE ROUTINE
0566  02450 016206        CPB ATNC      IF IT IS TAN
0567  02451 067353        JMP ATN         JUMP TO THE ARCTANGENT ROUTINE
0568  02452 016205        CPB ACSC      IF IT IS COS
0569  02453 066455        JMP ACS         JUMP TO THE ARCCOSINE ROUTINE
0570  02454 170402        RET           OTHERWISE IGNORE IT
0571*
0572**********************************
0573*
0574***     CALCULATE THE ARCCOSINE OF X
0575*
0576  02455 020031 ACS    LDA N1        LET THE CODEWORD BE -1
0577  02456 062621        JSM ASNS+1    CALCULATE THE ARCCOSINE OF X
0578  02457 021677        LDA GFG       LOAD THE ORIGINAL EXPONENT WORD
0579  02460 070151        SLA ACSS      SKIP IF THE MANTISSA SIGN BIT IS 0
0580  02461 020064        LDA PI        OTHERWISE ADD PI
0581  02462 063712        JSM XAA         TO THE RESULT
0582  02463 067354 ACSS   JMP EATN      CONVERT THE ANSWER TO THE PROPER UNITS
0583*
0584**********************************
0585*
0586***     CALCULATE THE ARCSINE OF X
0587*
0588  02464 062620 ASN    JSM ASNS      CALCULATE THE ARCSINE OF X
0589  02465 067354        JMP EATN      CONVERT THE ANSWER TO THE PROPER UNITS
0591*
0592**********************************
0593*
0594***     CALCULATE THE TANGENT OF X
0595*
0596  02466 062346 TAN    JSM STN       FETCH THE ARGUMENT AND INITIALIZE FLAGS
0597*
0598  02467 024063        LDB P1        SET THE RECIPROCAL
0599  02470 035671        STB RCPFG       FLAG TO ONE
0600  02471 025754        LDB AR2E      LOAD THE EXPONENT WORD
0601  02472 074153        SHP *+3       SKIP IF IT IS POSITIVE
0602  02473 045677 GET1   ISZ GFG       OTHERWISE SET THE GFG
0603  02474 066531        JMP GET         FLAG TO 1 AND DO NOT SCALE THE NO.
0604*
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0605  02475 022145        LDA TWOPI      REDUCE THE ARGUMENT TO
0606  02476 062256        JSM FRAC         A FRACTIONAL NO. OF CIRCLES
0607  02477 062307        JSM ZERO       DETERMINE IF THE RESULT IS ZERO
0608  02500 070635        SEC FAL1,C     SKIP IF IT IS
0609  02501 075512        SBM GET1       IT IS NON-ZERO;SKIP IF THE EXPONENT IS NEGATIVE
0610*
0611  02502 062366        JSM P2TO       OTHERWISE LOAD PI/2 INTO AR1
0612  02503 170400        CMY              AND SUBTRACT IT FROM THE
0613  02504 074742        SBR I6           ARGUMENT UNTIL AN
0614  02505 170420        FDV              OVERFLOW OCCURS
0615  02506 035671        STB RCPFG      SET RCPFG=THE NO. OF SUBTRACTS MINUS ONE
0616*
0617  02507 062307        JSM ZERO       DETERMINE IF THE RESULT IS ZERO
0618  02510 070474        SES *+9,C      SKIP IF IT IS NOT ZERO
0619*
0620  02511 025671        LDB RCPFG      LOAD THE RECIPROCAL FLAG
0621  02512 074151        SLB *+3        SKIP IF IT IS EVEN
0622  02513 045671        ISZ RCPFG      IF ITS ODD INCREMENT IT
0623  02514 166010 FAL1   JMP FALSE,I      AND STORE ZERO AND RETURN
0624  02515 025754        LDB AR2E       LOAD THE EXPONENT WORD
0625  02516 055664        DSZ TEMP       SKIP IF SIN OR COS ARE BEING CALCULATED
0626  02517 166007        JMP STMAX,I    OTHERWISE STORE MAX.NO. AND RETURN
0627  02520 170402        RET            RETURN
0628*
0629  02521 062247        JSM NORM+1     NORMALIZE THE RESULT AND CORRECT ITS EXPONENT
0630  02522 074352        SBM GET        SKIP IF THE EXPONENT IS NEGATIVE
0631  02523 021673        LDA SIGN       LOAD THE ORIGINAL EXPONENT WORD
0632  02524 050063        AND P1         SAVE THE MANTISSA SIGN
0633  02525 100101        ADA N256       SET THE EXPONENT TO -1
0634  02526 031754        STA AR2E       ESTABLISH THIS AS THE EXPONENT
0635  02527 031673        STA SIGN         WORD FOR THE REST OF THE ALGORITHMN
0636*
0637  02530 066534        JMP GET+3      JUMP TO AVOID THE OFFSET;THE NO. IS>1
0638  02531 035673 GET    STB SIGN       STORE THE NEW EXPONENT WORD
0639  02532 024063        LDB P1         LOAD 1; THE OFFSET
0640  02533 162011        JSM ROUND,I    OFFSET AND ROUND THE ARGUMENT
0641*
0642  02534 063164        JSM AWRS       INITIALIZE THE DIGIT STACK POINTER
0643*
0644  02535 021754        LDA AR2E       LOAD THE EXPONENT WORD
0645  02536 070240        AAR 6          OFFSET IT AND MULTIPLY IT BY 4
0646  02537 070076        TCA            COMPLEMENT IT
0647  02540 002162        ADA THETT      CALCULATE THE ADDRESS OF THE PROPER CORDIC CONSTANT
0648  02541 025754        LDB AR2E       LOAD THE EXPONENT WORD
0649  02542 074340        ABR 8          ALLIGN IT
0650  02543 004770        ADB P7         ADD 7 TO IT (THIS IS THE NO. OF CORDIC CONSTANTS TO
0651  02544 074453        SBP *+9        SKIP IF THE RESULT IS POSITIVE
0652  02545 171450        NRM            NORMALIZE THE NO.- ITS TOO SMALL
0653  02546 063611        JSM TTX        STORE IT IN X
0654  02547 025671        LDB RCPFG      LOAD THE RECIPROCAL FLAG
0655  02550 074111        SLB *+2          AND JUMP TO GET THE EXPONENT WORD
0656  02551 066612        JMP OUT-2        IF IT IS ODD
0657  02552 160217        JSM ONVX,I     OTHERWISE RECIPROCATE X AND
0658  02553 025671        LDB RCPFG        JUMP TO GET THE
0659  02554 066612        JMP OUT-2        EXPONENT WORD
0660  02555 075410        SZB *-8        IF THE EXPONENT WAS -7 DO THE ABOVE
0661*
0662  02556 063101        JSM IN         EXECUTE PHASE I OF THE CORDIC ROUTINE
0663*
0664  02557 021754        LDA AR2E       LOAD THE EXPONENT WORD
0665  02560 000105        ADA P256       INCREMENT THE EXPONENT BY 1
0666  02561 070340        AAR 8          ALLIGN IT (THIS IS AN OFFSET FACTOR IN PHASE II)
0667  02562 074742        SBR I6         STORE ZERO IN THE
0668  02563 035754        STB AR2E         EXPONENT OF AR2
0669  02564 070137        LDB A          DUPLICATE THE ABOVE CALCULATION
0670  02565 070076        TCA            COMPLEMENT IT
0671  02566 006163        ADB P6         ADD SIX TO IT (THIS IS THE NO. OF LOOPS OF PHASE II)
0672  02567 063124        JSM IN2        EXECUTE PHASE II OF THE CORDIC ROUTINE
0673*
0674  02570 021671        LDA RCPFG      LOAD THE RECIPROCAL FLAG
0675  02571 070211        SLA *+4        SKIP IF IT IS EVEN
0676*
0677  02572 021676        LDA RY1        LOAD THE FACTORS FOR THE
0678  02573 024015        LDB AR1          FINAL RECIPROCAL DIVISION (X/Y)
0679  02574 066577        JMP *+3        JUMP TO NORMALIZE THEM
0680  02575 020015        LDA AR1        LOAD THE ADDRESSES FOR THE
0681  02576 025676        LDB RY1          FINAL DIVISION (Y/X)
0682*
0683  02577 062237        JSM DIV        NORMALIZE THE FACTORS AND ESTABLISH THEIR EXPONENTS
0684  02600 062343        JSM XAT        FETCH THE ADDRESSES
0685  02601 063450        JSM FDV        DO THE FINAL DIVISION
0686*
0687  02602 025673        LDB SIGN       LOAD THE ORIGINAL EXPONENT WORD
0688  02603 005671        ADB RCPFG      ADD THE RCPFG TO THE MANTISSA SIGN
0689  02604 004105        ADB P256       INCREMENT THE EXPONENT BY 1
0690*
0691  02605 021671        LDA RCPFG      LOAD THE RECIPROCAL FLAG
0692  02606 070111        SLA *+2        IF IT IS ODD THE EXPONENT
0693  02607 066612        JMP OUT-2        IS FINISHED SO JUMP
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0694  02610 074076        TCB           OTHERWISE COMPLEMENT IT AND
0695  02611 006175        ADB D0010     PROPAGATE ANY LOW ORDER BITS INTO EXPONENT
0696*
0697  02612 005677        ADB GFG       ADD GFG TO THE MANTISSA SIGN (FINAL SIGN)
0698  02613 021754        LDA AR2E      THE FINAL EXPONENT WORD WILL BE
0699  02614 031744 OUT    STA AR1E        THE QUOTIENT EXPONENT OF THE QUOTIENT AND THIS COM
0700*
0701  02615 020004        LDA ADRX      SET X AS THE
0702  02616 031720        STA ADR2        RETURN ADDRESS
0703  02617 166012            JMP TERM,I    CLEAN UP THE ANSWER AND RETURN


0705*
0706************************************
0707*
0708***      SUBROUTINE FOR ARCSINE AND ARC COSINE
0709*
0710  02620 020063 ASNS  LDA P1        LOAD THE CODE WORD FOR SCRT
0711  02621 025750       LDB ADRXE     SAVE THE ORIGINAL EXPONENT
0712  02622 035673       STB SIGN        FOR FUTURE
0713  02623 035677       STB GFG         REFERENCES
0714  02624 062320       JSM SCRT      CALCULATE SQR (1+OR-XX)
0715*
0716  02625 000063       ADA P1        FETCH THE FIRST FOUR DIGITS
0717  02626 070517       LDA A,I         OF THE 'TO BE' DIVISOR
0718  02627 070610       SZA OK-2      SKIP IF IT IS ZERO
0719*
0720  02630 062336       JSM SCRT1     FETCH THE ADDRESSES FOR THE IDENTITY
0721  02631 063450       JSM FDV       DIVIDE THE TWO QUANTITITES
0722  02632 063611       JSM TTX       TRANSFER THE RESULT INTO X
0723*
0724************************************
0725*
0726***      CALCULATE THE ARCTANGENT OF X
0727*
0728  02633 026157 ATNS  LDB EN3       FETCH THE ARGUMENT AND
0729  02634 062231       JSM INIT        INITIALIZE THE FLAGS
0730*
0731  02635 062307       JSM ZERO      DETERMINE WHETHER THE ARGUMENT IS ZERO
0732  02636 070114       SES *+2       IF THE ARGUMENT IS
0733  02637 170402       RET             ZERO RETURN
0734*
0735  02640 074312       SBM OK+1      SKIP IF THE EXPONENT IS NEGATIVE
0736  02641 006160       ADB EN12      ADD -12 TO THE EXPONENT AND
0737  02642 074152       SBM OK          SKIP IF THE RESULT IS NEGATIVE
0738  02643 162010       JSM FALSE,I   OTHERWISE STORE ZERO AND
0739  02644 066670       JMP POOT        JUMP TO ADD PI/2 TO IT
0740  02645 160217 OK    JSM ONVX,I    RECIPROCATE THE ARGUMENT IF IT IS >1
0741*
0742  02646 063770       JSM CLER      CLEAR OUT THE DIGIT STACK AREA
0743*
0744  02647 021754       LDA AR2E      LOAD THE EXPONENT WORD
0745  02650 070300       AAR 7         DIVIDE IT BY 2
0746  02651 070076       TCA           COMPLEMENT IT
0747  02652 000074       ADA P22       CALCULATE 22-2(EXPONENT) ;A SHIFT FACTOR
0748  02653 024106       LDB P12       LET B=12
0749*
0750  02654 063124       JSM IN2       EXECUTE PHASE II OF THE CORDIC ROUTINE
0751  02655 021754       LDA AR2E      SAVE THE PRESENT
0752  02656 031674       STA SAVE        EXPONENT WORD
0753  02657 070340       AAR 8         ALLIGN IT
0754  02660 070076       TCA           COMPLEMENT IT
0755  02661 002163       ADA P6        LET A = 6-EXP
0756*
0757  02662 002172       ADA N13       IF THIS QUANTITY IS
0758  02663 070112       SAM *+2         GREATER THAN 12
0759  02664 020031       LDA N1          REPLACE IT WITH 12
0760  02665 002165       ADA P13         OTHERWISE LEAVE IT
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3' —Continued

```
0761*
0762  02666 026145       LDB TWOPI     LET B=THE ADDRESS PRIOR TO THE FIRST CORDIC ADDRESS
0763  02667 063301       JSM IN3       EXECUTE PHASE III OF THE CORDIC ALGORITHM
0764*
0765  02670 021673 POUT  LDA SIGN      LOAD THE ORGINAL EXPONENT WORD
0766  02671 031754       STA AR2E      STORE IT IN AR2
0767  02672 070412       SAM GONO      SKIP IF THE EXPONENT IS NEGATIVE
0768*
0769  02673 050063       AND P1        SAVE THE ORIGINAL
0770  02674 031754       STA AR2E        SIGN OF THE MATISSA
0771*
0772  02675 025674       LDB SAVE      LOAD THE INTERMEDIATE EXPONENT WORD
0773  02676 062277       JSM UNRM+1    UNNORMALIZE THE ANSWER
0774*
0775  02677 062366       JSM P2TO      SUBTRACT THE
0776*
0777  02700 170400       CMY             ANSWER FROM
0778  02701 170560       FXA             PI/2
0779*
0780  02702 062247 GONO  JSM NORM+1    NORMALIZE THE RESULT
0781*
0782  02703 074742 SOUT  SBR 16        CLEAR THE B REGISTER
0783  02704 066613       JMP OUT-1       AND EXIT

0785*
0786************************************
0787*
0788***      CALCULATE THE LOG OF X
0789*
0790  02705 063000 LOG   JSM LNX       CALC THE NATURAL LOG OF X
0791  02706 022207       LDA LN10      LOAD THE ADDRESS OF LN(10)
0792  02707 055672       DSZ FNCFG     SKIP IF THE MAX NO. WAS STORED
0793  02710 067403       JMP GTOR+1    DIVIDE THE RESULT BY LN(10)
0794  02711 170402 RET6  RET           RETURN
0795*
0796************************************
0797*
0798***      RAISE X TO THE POWER IN Y
0799*
0800  02712 021751 UPARO LDA ADRXE+1   LOAD THE FIRST FOUR DIGITS OF X
0801  02713 072210       RZA REXP      SKIP IF THEY ARE NONZERO
0802*
0803  02714 021740       LDA ADRYE     LOAD THE EXPONENT WORD OF Y
0804  02715 071631       SLA RET6,C    SKIP IF THE MANTISSA IS NEGATIVE
0805  02716 066776       JMP BACK1     OTHERWISE JUMP TO STORE THE MAX. NO.
0806*
0807  02717 063000 REXP  JSM LNX       CALCULATE THE NATURAL LOG OF X
0808  02720 020005       LDA ADRY      MULTIPLY THE RESULT BY X
0809  02721 063473       JSM XAM         AND PASS INTO THE EXP ROUTINE

0811*
0812************************************
0813*
0814***      CALCULATE THE EXPONENTIAL OF X
0815*
0816  02722 024063 EXP   LDB P1        LOAD THE FUNCTION FLAG
0817  02723 062231       JSM INIT      FETCH THE ARGUMENT
0818*
0819  02724 025754       LDB AR2E      LOAD THE EXPONENT WORD AND
0820  02725 074152       SBM *+3         SKIP IF THE EXP IS NEGATIVE
0821  02726 006157       ADB EN3       OTHERWISE SUBTRACT 3 FROM IT
0822  02727 074353       SBP STM1-1      AND STORE MAX IF IT IS STILL POSITIVE
0823*
0824  02730 022207       LDA LN10      REDUCE THE ARGUMENT TO A
0825  02731 062256       JSM FRAC        FRACTION OF LN(10)
0826  02732 062276       JSM UNRM      UNNORMALIZE THIS QUANTITY
0827*
0828  02733 021670       LDA COUNT     LOAD THE INTEGER PART (NO. OF LN10'S)
0829  02734 070542       SAR 12        CHECK TO SEE IF THERE
0830  02735 070150       SZA LT3         ARE THREE DIGITS
0831  02736 074044       SBL 15        IF THERE ARE JUMP TO
0832  02737 166007 STM1  JMP STMAX,I     THE STORE MAX ROUTINE
0833*
0834  02740 025670 LT3   LDB COUNT     LOAD THE NO. OF LN10'S
0835  02741 074404       SBL 8         ISOLATE THE
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0836  02742 074142        SBR 4           SECOND DIGIT IN B
0837  02743 021670        LDA COUNT       RELOAD THE NO. OF LN10'S
0838  02744 070342        SAR 8           ISOLATE THE FIRST
0839  02745 070344        SAL 9             DIGIT AND ADD
0840  02746 070037        ADB A             TEN TIMES IT
0841  02747 070704        SAL 2             INTO THE
0842  02750 070037        ADB A             B REGISTER
0843  02751 035671        STB RCPFG       STORE THIS QUANTITY IN RCPFG
0844*
0845  02752 063770        JSM CLER        CLEAR OUT THE DIGIT STACK AREA
0846*
0847  02753 022152        LDA THETL       LET A= THE ADDRESS AFTER THE LAST CONSTANT
0848  02754 024012        LDB AWRDN       INTIIALIZE THE
0849  02755 004031        ADB N1            DIGIT STACK
0850  02756 035666        STB DIGIT         POINTER
0851  02757 026163        LDB P6          LET B= THE NO. OF LOOPS TO BE CYCLED
0852  02760 063101        JSM IN          EXECUTE PHASE I OF THE ROUTINE
0853*
0854  02761 024023        LDB N5          STORE THE NEXT FIVE
0855  02762 045666        ISZ DIGIT         DIGITS OF THE RESULT
0856  02763 171400        MLS               INTO THE DIGIT STACK
0857  02764 131666        STA DIGIT,I       BY LEFT SHIFTING
0858  02765 077670        RIB *-3           THE RESULT
0859*
0860  02766 062363        JSM OTO         SET AR1 =1
0861*
0862  02767 063234        JSM PTJ         EXECUTE PHASE IV OF THE ALGORITHM
0863*
0864  02770 021671        LDA RCPFG       STORE THE CALCULATED BINARY NO.
0865  02771 031754        STA AR2E          OF LN10'S IN THE EXPONENT BITS
0866*
0867  02772 171450        NRM             NORMALIZE THE ANSWER
0868*
0869  02773 021673        LDA SIGN        LOAD THE ORIGINAL EXPONENT WORD
0870  02774 070151        SLA *+3         IF THE ORIGIANL MANTISSA WAS
0871  02775 063611        JSM TTX           NEGATIVE TRANSFER THE ARGUMENT
0872  02776 160217 BACK1  JSM ONVX,I      RECIPROCATE X
0873*
0874  02777 066703        JMP SOUT        RETURN THRU THE RESET ROUTINE

0876*
0877**********************************
0878*
0879***     CALCULATE THE NATURAL LOG OF X
0880*
0881*
0882  03000 024031 LNX    LDB N1          LET THE FUNCTION FLAG BE -1
0883  03001 062231        JSM INIT        FETCH THE ARGUMENT AND INITIALIZE THE FLAGS
0884*
0885  03002 062307        JSM ZERO        DETERMINE IF THE ARGUMENT IS NEGATIVE
0886  03003 070234        SES *+4,C       SKIP IF IT IS NOT
0887  03004 024063        LDB P1            JUMP TO STORE
0888  03005 035672        STB FNCFG       NEGATIVE INFINITY AND
0889  03006 166007        JMP STMAX,I     SET A FLAG FOR LOG BASE 10
0890  03007 074111        SLB *+2         SKIP IF THE ARGUMENT IS NEGATIVE
0891  03010 162176        JSM ERROR,I     SET THE STATUS LIGHT IF IT IS
0892*
0893  03011 063770        JSM CLER        OTHERWISE CLEAR THE DIGIT STACK AND CONTINUE
0894  03012 063234        JSM PTJ         EXECUTE PHASE IV OF THE ALGORITHM
0895  03013 055666        DSZ DIGIT       DECREMENT THE DIGIT STACK POINTER
0896*
0897  03014 022163        LDA P6          LET A=6 , THE NO. OF CONSTANTS USED
0898  03015 026207        LDB LN10        SET B= THE ADDRESS PRIOR TO THE FIRST CONSTANT
0899  03016 063301        JSM IN3         EXECUTE PHASE III OF THE ALGORITHM
0900*
0901  03017 063611        JSM TTX         TRANSFER THE RESULT INTO X
0902*
0903*
0904  03020 020016        LDA AR2         CLEAR THE AR2
0905  03021 170000        CLR               WORKING REGISTER
0906*
0907  03022 021673        LDA SIGN        LOAD THE ORIGINAL EXPONENT WORD
0908  03023 000105        ADA P256        INCREMENT THE EXPONENT BY 1
0909  03024 070340        AAR 8           ALLIGN THE EXPONENT
0910  03025 070113        SAP *+2         SKIP IF IT IS POSITIVE
0911  03026 070076        TCA             OTHERWISE COMPLEMENT IT
0912*
0913  03027 002155        ADA N100        ADD -100 TO THE EXPONENT
0914  03030 024105        LDB P256        SET B = BCD 0100
0915  03031 070153        SAP *+3         SKIP IF THE EXPONENT -100 IS POSITIVE
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
0916  03032 002154        ADA P100       RESTORE THE EXPONENT
0917  03033 074742        SBR 16         CLEAR B
0918*
0919  03034 000024        ADA N10        SUBTRACT 10 FROM THE EXPONENT
0920  03035 070152        SAM *+3        IF THE ANSWER IS POSITIVE ADD
0921  03036 006175        ADB D0010        A BCD 0010 INTO B
0922  03037 067034        JMP *-3        JUMP BACK TO SUBTRACT ANOTHER 10
0923  03040 000104        ADA P10        RESTORE THE EXPONENT AND CONTINUE
0924*
0925  03041 070037        ADB A          ADD THE REMAINING NO. OF COUNTS
0926  03042 035755        STB AR2E+1       INTO B AND STORE THE WORD IN AR2
0927*
0928  03043 022071        LDA EP3        SET THE EXPONENT OF AR2
0929  03044 031754        STA AR2E         EQUAL TO 3
0930  03045 062247        JSM NORM+1     NORMALIZE AR2 AND CORRECT THE EXPONENT
0931*
0932  03046 026207        LDB LN10       MULTIPLY AR2 BY THE
0933  03047 162014        JSM FMP,I        NATURAL LOG OF TEN
0934*
0935  03050 020016        LDA AR2        NORMALIZE THIS RESULT AND
0936*
0937  03051 024004        LDB ADRX         THE MANTISSA DERIVED
0938  03052 062237        JSM DIV          BY THE ALGORITHM
0939*
0940  03053 145724        ISZ ADR1,I     CHANGE THE SIGN OF THE ALGORITHM MANTISSS
0941*
0942  03054 021673        LDA SIGN       LOAD THE ORIGINAL EXPONENT WORD
0943  03055 070113        SAP *+2        IF THE EXPONENT IS NEGATIVE CHANGE
0944  03056 045730        ISZ ADRTE        THE SIGN OF THE MULTIPLE OF LN10
0945  03057 062343        JSM XAT        ADD THESE TWO
0946  03060 162015        JSM FAD,I        PARTIAL RESULTS
0947  03061 020013        LDA ADRT       TRANSFER THE RESULT
0948  03062 067612 OUTS   JMP TTX+1        INTO X

0950*
0951************************************
0952*
0953***      CALCULATE X FACTORIAL
0954*
0955  03063 063405 XFACT  JSM PRE        TAKE THE INTEGER ABSOLUTE VALUE OF X
0956  03064 075710        SZB OUTS       STORE ZERO IF X IS ZERO
0957  03065 021750        LDA ADRXE      LOAD THE EXPONENT WORD
0958  03066 000032        ADA EN2        IF THE EXPONENT IS GREATER THAN
0959  03067 070112        SAM *+2          OR EQUAL TO TWO JUMP TO
0960  03070 166007        JMP STMAX,I      THE STORE MAX ROUTINE
0961*
0962  03071 062373        JSM XTP        TRANSFER X INTO P
0963*
0964  03072 020056 FAC3   LDA ONE        SUBTRACT FLOATING
0965  03073 026211        LDB ADRP         ONE FROM THE
0966  03074 162016        JSM FSB,I        P TEMPORARY
0967*
0968  03075 021673        LDA ADRPE+1    LOAD THE FIRST FOUR DIGITS OF P
0969  03076 070410        SZA PTA+3      RETURN IF THEY ARE ZERO
0970*
0971  03077 063374        JSM PXX        OTHERWISE TRANSFER P TO X
0972  03100 067072        JMP FAC3       JUMP BACK TO CONTINUE

0974*
0975*
0976************************************
0977*
0978* CORDIC LOOP
0979*
0980*
0981************************************
0982*
0983***      PHASE I
0984*
0985  03101 031665 IN     STA TABL0      STORE THE CONSTANT ADDRESS
0986  03102 035667        STB SHIFT      STORE THE NO. OF LOOPS TO BE EXECUTED
0987*
0988  03103 062307 PTA    JSM ZERO       DETERMINE IF THE NUMBER IS ZERO
0989  03104 070154        SES *+3        SKIP IF IT NOT ZERO
0990  03105 145666        ISZ DIGIT,I    INCREMENT THE LAST DIGIT COUNT
0991  03106 170402        RET              AND RETURN
0992*
0993  03107 170400        CMY            COMPLEMENT THE ARGUMENT
0994*
0995  03110 045666        ISZ DIGIT      INCREMENT THE POINTER TO THE DIGIT BEING CALCULATER
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
2996*
2997  03111 020003        LDA P4          FETCH THE NEXT CONSTANT
2998  03112 062353        JSM UPTAB         INTO AR1
2999*
1000  03113 074004        SHL 16          CLEAR B AND SUBTRACT THE CONSTANT
1001  03114 170420        FDV               FROM THE ARGUMENT UNTIL OVERFLOW
1002*
1003  03115 135666        STB DIGIT,I     STORE THE NUMBER OF SUBTRACTS BEFORE OVERFLOW
1004*
1005  03116 170400        CMY             RESTORE THE ARGUMENT TO ITS LAST
1006  03117 170560        FXA               POSITIVE VALUE
1007*
1008  03120 171400        MLS             LEFT SHIFT ALLIGN IT FOR THE NEXT LOOP
1009*
1010  03121 055667        DSZ SHIFT       DECREMENT THE LOOP COUNTER AND
1011  03122 067103        JMP PTA           CONTINUE EXECUTING IF NONZERO
1012  03123 170402        RET             OTHERWISE RETURN

1014*
1015**********************************
1016*
1017***     PHASE II
1018*
1019  03124 031674 IN2    STA SAVE        STORE THE SHIFT FACTOR
1020  03125 035667        STB SHIFT       STORE THE NO. OF LOOPS TO BE EXECUTED
1021*
1022  03126 062363        JSM OTO         SET AR1=1
1023*
1024  03127 020011        LDA ADR0        ESTABLISH RY1 AS A
1025  03130 031676        STA RY1           TEMPORARY LOCATION POINTER
1026  03131 020004        LDA ADRX        ESTABLISH RY2 AS A
1027  03132 031675        STA RY2           TEMPORARY LOCATION POINTER
1028*
1029  03133 025672        LDB FNCFG       JUMP OVER THE NEXT GROUP
1030  03134 076650        RZB PTE           OF INSTRUCTIONS IF THE
1031  03135 067166 PTL1   JMP PTL           ATN IS BEING EXECUTED
1032*
1033*
1034  03136 076610 PTB    RZB PTN         JUMP TO POINT N IF ARCTAN
1035  03137 067173        JMP PTC         OTHERWISE JUMP TO POINT C
1036*
1037  03140 055666 PTD    DSZ DIGIT       DECREMENT THE DIGIT STACK POINTER
1038  03141 025672        LDB FNCFG       LOAD THE FUNCTION FLAG
1039  03142 075550        SZB PTL1        SKIP IF IT IS ZERO (TAN)
1040*
1041  03143 021755        LDA AR2E+1      LOAD THE FIRST FOUR DIGITS OF THE ARGUMENT
1042  03144 070542        SAR 12          IF THE FIRST DIGIT IS NOT ZERO
1043  03145 070150        SZA *+3           THEN DO A RIGHT SHIFT
1044  03146 170410        DRS               ON AR1 AND JUMP
1045  03147 067152        JMP PTN           TO CONTINUE
1046  03150 171400        MLS             OTHERWISE DO A LEFT SHIFT ON AR2
1047  03151 062370 PTE    JSM TTR1          AND UPDATE RY1 WITH THIS VALUE
1048*
1049  03152 174400 PTN    CMX             COMPLEMENT AR1 (X) AND
1050  03153 170560        FXA               ADD IT TO AR2 (Y)
1051  03154 070214        SES *+4         SKIP IF AN OVERFLOW OCCURRED
1052  03155 062376        JSM R1TT        OTHERWISE THIS DIGIT IS FINISHED SO
1053  03156 174400        CMX               RESTOR E AR2 WITH RY1 AND RECOMPLEMENT AR1
1054  03157 067231 PTF1   JMP PTF         JUMP TO POINT F
1055*
1056  03160 174400        CMX             THIS DIGIT IS NOT FINISHED, RECOMPLEMENT AR1
1057*
1058  03161 062307        JSM ZERO        DETERMINE IF THE RESULT IS ZERO
1059  03162 070534        SES PTM,C       SKIP TO POINT M IF IT IS NOT
1060  03163 155666        DSZ DIGIT,I     IF IT IS,DECREMENT THE LAST CALCULATED DIGIT
1061  03164 020012 AWRS   LDA AWRDN       TERMINATE THE DIGIT
1062  03165 067776        JMP KK            STACK POINTER
1063  03166 024063 PTL    LDB P1          RIGHT SHIFT AR2 1 DIGIT
1064*
1065  03167 162011        JSM ROUND,I       AND ROUND UP THE LAST DIGIT
1066*
1067  03170 062370        JSM TTR1        TRANSFER THIS NEW AR2 INTO RY1
1068*
1069  03171 121666        LDA DIGIT,I     IF THE PRESENT DIGIT
1070  03172 071250        SZA PTF1          IS ZERO, SKIP
1071*
1072  03173 170560 PTC    FXA               ADD AR1 TO AR2 (X TO Y)
1073*
1074  03174 021676 PTM    LDA RY1           INTERCHANGE RY1
1075  03175 025675        LDB RY2             AND RY2 (THE OLDY
1076  03176 031675        STA RY2             WITH THE
1077  03177 035676        STB RY1             NEW ONE)
1078*
1079  03200 062370        JSM TTR1        STORE RY1 IN RY2
1080*
1081  03201 021675        LDA RY2         FETCH THE LAST
```

```
1082  03202 062377        JSM R1TT+1      Y
1083*
1084  03203 021672        LDA FNCFG       LOAD THE FUNCTION FLAG
1085*
1086  03204 025667        LDB SHIFT       TAKE THE LOOP COUNTER
1087  03205 004031        ADB N1            SUBTRACT ONE FORM IT
1088  03206 074744        SBL 1             AND MULTIPLY IF BY 2
1089*
1090  03207 070150        SZA *+3         IF THE TANGENT IS BEING CALCULATED
1091*
1092  03210 074076        TCB               ADD "SAVE",THE SHIFT FACTOR
1093  03211 067213        JMP *+2           TO IT TWICE
1094*
1095  03212 005674        ADB SAVE          OTHERWISE COMPLEMENT IT
1096  03213 005674        ADB SAVE          AND ADD SAVE TO IT ONCE
1097*
1098  03214 074117        LDA B           IF THIS QUANTITY IS
1099  03215 002171        ADA N12           GREATER THAN 12 SKIP
1100  03216 070353        SAP PTK           TO POINT K
1101*
1102  03217 162011        JSM ROUND,I     OTHERWISE SHIFT RIGHT AR2 THIS MANY TIMES
1103*
1104  03220 025672        LDB FNCFG       IF THE TANGENT IS BEING
1105  03221 076110        RZB *+2           CALCULATED SUBTRACT AR1 FROM
1106  03222 170400        CMY               AR2 ; OTHERWISE ADD THE TWO
1107  03223 170560        FXA               TOGETHER INTO AR2
1108*
1109  03224 062407        JSM TTO         DUPLICATE AR2 INAR1
1110*
1111  03225 062376 PTK    JSM R1TT        TRANSFER RY1 INTO AR2 (NEW Y)
1112*
1113  03226 025672        LDB FNCFG       DECREMENT THE DIGIT STACK
1114  03227 155666        DSZ DIGIT,I       DIGIT AND JUMP
1115  03230 067136        JMP PTB           TO POINT B IF NOT ZERO; CONTINUE IF IT IS
1116*
1117  03231 055667 PTF    DSZ SHIFT       DECREMENT THE MAIN LOOP COUNTER
1118  03232 067140        JMP PTD         JUMP TO POINT D IF IT IS NONZERO
1119  03233 170402        RET             OTHERWISE RETURN

121*
122**********************************
123*
124***     PHASE III
125*
126  03234 020012 PTJ    LDA AWRDN       INITIALIZE THE DIGIT
127  03235 031666        STA DIGIT         STACK POINTER
128*
129  03236 020106        LDA P12         INITIALIZE THE LOOP
130  03237 031667        STA SHIFT         COUNT TO 12
131  03240 025672        LDB FNCFG       LOAD THE FUNCTION FLAG
132  03241 074312        SBM PTS         SKIP IF LN IS BEING CALCULATED
133  03242 062403        JSM OTT         IF EXP SET AR2=1 AND
134*
135  03243 025672 PTR    LDB FNCFG       LOAD THE FUNCTION FLAG
136  03244 074253        SBP PTI         SKIP IF IT IS POSITIVE (EXP)
137  03245 174400        CMX             OTHERWISE SUBTRACT
138  03246 170560        FXA               AR1 FROM AR2 AND
139  03247 062407 PTS    JSM TTO           DUPLICATE THE RESULT IN AR1
140  03250 067253        JMP PTH-2         PREPARE FOR THE NEXT SUBTRACTION
141*
142  03251 121666 PTI    LDA DIGIT,I     LOAD THE PRESENT DIGIT FROM THE
143  03252 070750        SZA GOON-1        STACK AND SKIP IF IT IS ZERO
144*
145  03253 020104        LDA P10         INTIALIZE THE INNER
146  03254 031665        STA TABL0         LOOP COUNT TO 10
147  03255 025667 PTH    LDB SHIFT       CALCULATE THE SHIFT FACTOR
148  03256 074076        TCB               FOR AR1 OF 12-SHIFT AND
149  03257 004106        ADB P12           SET THIS IN B
150  03260 070742        SAR 16          CLEAR THE A REGISTER AND
151  03261 174430        MRX               RIGHT SHIFT AR1 B TIMES
152*
153  03262 025672        LDB FNCFG       IF THE EXP IS BEING CALCULATED
154  03263 014063        CPB P1            AND THE LAST SHIFTED DIGIT IS
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1155  03264 000023        ADA N5          >=5, THEN INCREMENT THE NEXT SUM
1156*
1157  03265 170560        FXA             ADD AR1 TO AR2  (CALC X+104(-J)X)
1158*
1159  03266 070354        SES PTG         EXIT FROM THE INNER LOOP ON OVERFLOW
1160  03267 055665        DSZ TABL0       DECREMENT THE INNER LOOP COUNTER
1161  03270 067272        JMP GOON        CONTINUE IF IT IS NON ZERO
1162  03271 067275        JMP PTG         OTHERWISE JUMP TO POINT G
1163*
1164  03272 062407 GOON   JSM TTO         DUPLICATE AR2 IN AR1
1165  03273 155666        DSZ DIGIT,I     DECREMENT THE PRESENT STACK DIGIT AND
1166  03274 067255        JMP PTH           JUMP TO POINT H IF NON-ZERO
1167*
1168  03275 045666 PTG    ISZ DIGIT       INCREMENT THE DIGIT STACK POINTER
1169  03276 055667        DSZ SHIFT       DECREMENT THE MAIN LOOP COUNTER
1170  03277 067243        JMP PTR         JUMP BACK TO CONTINUE
1171  03300 170402        RET             OTHERWISE RETURN

1173*
1174**********************************
1175*
1176***     PHASE IV
1177*
1178  03301 035665 IN3    STB TABL0       STORE THE TABLE CONSTANT ADDRESS
1179  03302 031664        STA TEMP        STORE THE LOOP COUNTER TO
1180  03303 031667        STA SHIFT         BE USED IN THE FIRST LOOP
1181*
1182  03304 020016        LDA AR2         CLEAR OUT THE AR2
1183  03305 170020        CLR               TEMPORARY REGISTER
1184*
1185  03306 121666 PTP    LDA DIGIT,I     LOAD THE PRESENT DIGIT FROM THE
1186  03307 070076        TCA               STACK, COMPLEMENT IT AND SHIFT
1187  03310 024063        LDB P1            IT INTO THE MOST SIGNIFICANT
1188  03311 174470        MRY               LOCATION OF AR2
1189  03312 063345        JSM IOD         POINT THE DIGIT STACK POINTER TO THE NEXT WORD
1190  03313 055667        DSZ SHIFT       DECREMENT THE DIGIT STACK POINTER
1191  03314 067306        JMP PTP         IF IT IS NON-ZERO JUMP BACK TO GET THE NEXT DIGIT
1192*
1193*
1194  03315 070742        SAR 16          CALCULATE 12 MINUS THE NUMBER OF
1195  03316 025664        LDB TEMP          DIGITS SHIFTED INTO AR2 ABOVE
1196  03317 074076        TCB               AND LET THIS BE THE LOOP COUNT
1197  03320 004106        ADB P12           FOR THE NEXT SECTION
1198  03321 035667        STB SHIFT         THEN SHIFT INTO AR2 THIS SAME
1199  03322 174470        MRY               NUMBER OF ZEROS
1200*
1201  03323 020054 PTQ    LDA N4          SUBTRACT 4 FROM THE PRESENT CONSTANT ADDRESS
1202  03324 062353        JSM UPTAB         AND LOAD THIS CONSTANT INTO AR1
1203  03325 070742        SAR 16          RIGHT SHIFT THIS CONSTANT THE NUMBER
1204  03326 025667        LDB SHIFT         OF COUNTS SPECIFIED BY THE LOOP COUNTER
1205  03327 074650        SZB NAW           MINUS ONE EXCEPT WHEN THIS NO.
1206  03330 004031        ADB N1            IS NEGATIVE IN WHICH CASE JUST EXIT
1207  03331 174430        MRX               FROM THE LOOP AND RETURN
1208*
1209  03332 000023        ADA N5          IF THE LAST SHIFTED DIGIT
1210  03333 070175        SEC *+3,C         IS =>5 THEN INCREMENT THE
1211  03334 020015        LDA AR1           CONSTANT LOCATED IN AR1
1212  03335 170540        MDI               BY ONE
1213*
1214  03336 125666        LDB DIGIT,I     LOAD AND COMPLEMENT THE PRESENT
1215  03337 074076        TCB               DIGIT IN THE DIGIT STACK
1216  03340 171460        FMP             ADD AR1 TO AR2 THIS MANY TIMES
1217*
1218  03341 063345        JSM IOD         POINT THE DIGIT STACK POINTER TO THE NEXT WORD
1219  03342 055667        DSZ SHIFT       DECREMENT THE LOOP COUNTER AND
1220  03343 067323        JMP PTQ           JUMP TO POINT Q IF IT IS NONZERO
1221  03344 170402 NAW    RET             RETRUN IF IT IS ZERO
1222*
1223  03345 025672 IOD    LDB FNCFG         LOAD THE FUNCTION FLAG
1224  03346 074151        SLB *+3         IF THE LN IS BEING CALCULATED
1225  03347 055666        DSZ DIGIT         DECREMENT THE DIGIT STACK POINTER
1226  03350 170402        RET               AND RETURN
1227  03351 045666        ISZ DIGIT       OTHERWISE INCREMENT THE DIGIT STACK
1228  03352 170402 RET2   RET               POINTER AND RETURN

1230*
1231**********************************
1232*
1233***     CONTROL OF THE ARCTANGENT ROUTINE
1234*
1235  03353 062633 ATN    JSM ATNS        CALCULATE THE ARCTAN OF X
1236*
1237  03354 020031 EATN   LDA N1          SET THE ATN FLAG FOR THE MODE CONVERSION
1238*
1239**********************************
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1240*
1241***     CONVERT X TO A DIFFERENT UNITS
1242*
1243  03355 031665 MODE  STA TABL0     STORE THE FUNCTION FLAG
1244  03356 025776       LDB UNITS     LOAD THE STATUS WORD
1245  03357 074502       SBR 11        SKIP IF THE GRADS BIT
1246  03360 074151       SLB *+3         IS NOT SET
1247  03361 070752       SAM RTOG      CONVERT X TO GRADS IF THE ATN CALKED THE ROUTINE
1248  03362 067402       JMP GTOR      OTHERWISE CONVERT X TO RADIANS FROM GRADS
1249  03363 074204       SBL 12        IF THE RADIANS BIT IS SET
1250  03364 075312       SRM RET2        EXECUTE A RETURN
1251  03365 024063       LDB P1        OTHERWISE SET THE DEGREE BIT AND
1252  03366 063677       JSM DEG         TURN ON THAT LIGHT
1253  03367 045665       ISZ TABL0     CONVERT X TO DEGREES IF ATN CALLED THE ROUTINE
1254  03370 067376       JMP DTOR      OTHERWISE CONVERT X TO RADIANS FROM DEGREES
1255*
1256  03371 022147 RTOD  LDA DPERR     TRANSFER THE CONVERSION (DEGREES)
1257  03372 062374       JSM XTP+1       CONSTANT INTO A TEMPORARY P
1258  03373 045675       ISZ ADRPE+3   INCREMENT THE LAST DIGIT OF IT
1259  03374 022211 PXX   LDA ADRP      MULTIPLY THE X REGISTER
1260  03375 067473       JMP XAM         BY IT AND RETURN
1261*
1262  03376 022147 DTOR  LDA DPERR     CONVERT THE CONTENTS OF
1263  03377 067403       JMP GTOR+1      X TO RADIANS FROM DEGREES
1264*
1265  03400 022152 RTOG  LDA GPERR     CONVERT THE CONTENTS OF
1266  03401 067372       JMP RTOD+1      X TO GRADS
1267*
1268  03402 022152 GTOR  LDA GPERR     CONVERT THE CONTENTS OF
1269  03403 024004       LDB ADRX        X TO RADIANS FROM
1270  03404 067450       JMP FOV         GRADS

1272*
1273************************************
1274*
1275***     CALCULATE THE INTEGER ABSOLUTE VALUE OF X
1276*
1277  03405 160264 PRE   JSM INTX,I    TAKE THE INTEGER VALUE OF X
1278  03406 020004 ABSX  LDA ADRX      SAVE THE ADDRESS OF THE QUANTITY
1279  03407 031720       STA ADR2        BEING USED
1280*
1281  03410 070537       LDB A,I       LOAD THE EXPONENT WORD
1282  03411 074071       SLB *+1,C     CLEAR THE MANTISSA SIGN BIT
1283  03412 070577       STB A,I       RESTORE IT
1284*
1285  03413 025751       LDB ADRXE+1   LOAD THE FIRST FOUR DIGITS OF X
1286  03414 020056       LDA ONE       LOAD THE ADDRESS OF FLOATING ONE
1287  03415 170402       RET           RETURN
1288*
1289************************************
1290*
1291***     RAISE TEN TO THE POWER IN X
1292*
1293  03416 022207 TENX  LDA LN10      MULTIPLY X BY LN(10) AND RAISE
1294  03417 066721       JMP REXP+2      E TO THIS POWER
1295*
1296************************************
1297*
1298***     TAKE THE ABSOLUTE VALUE OF Y
1299*
1300  03420 020005 ABS   LDA ADRY      LOAD THE ADDRESS OF THE Y REGISTER
1301  03421 067410       JMP ABSX+2    TAKE THE ABSOLUTE VALUE OF Y

1303*
1304************************************
1305*
1306***     SCALE THE X AND Y REGISTERS
1307*
1308  03422 022177 SCALE LDA XMIN      LOAD THE ADDRESS OF XMIN
1309  03423 024004       LDB ADRX      LOAD THE ADDRESS OF X
1310  03424 063431       JSM SCL1      REPLACE X WITH 9999(X-XMIN)/(XMAX-XMIN)
1311*
1312  03425 022200       LDA YMIN      LOAD THE ADDRESS OF YMIN
1313  03426 024005       LDB ADRY      LOAD THE ADDRESS OF Y
1314  03427 063431       JSM SCL1      REPLACE Y WITH 9999(Y-YMIN)/(YMAX-YMIN)
1315  03430 164242       JMP FMT,I     JUMP TO THE FORMAT ROUTINE
1316*
1317*
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1318  03431 031664 SCL1  STA TEMP      STORE THE ADDRESS OF THE MINMUM VARIABLE
1319  03432 035667       STB SHIFT       AND THE INDEPENDENT VARIABLE
1320*
1321  03433 162016       JSM FSB,I     SUBTRACT THE MIN FROM THE VARIABLE
1322*
1323  03434 021664       LDA TEMP      CALCULATE THE ADDRESS OF THE MAX VALUE
1324  03435 000054       ADA N4          AND TRANSFER THE MAX VALUE
1325  03436 026211       LDB ADRP        INTO A TEMPORARY STORAGE
1326  03437 170004       XFR             REGISTER P
1327*
1328  03440 021664       LDA TEMP      SUBTRACT THE MIN FACTOR
1329  03441 026211       LDB ADRP        FROM THE MAX FACTOR
1330  03442 162016       JSM FSB,I       IN P
1331*
1332  03443 022156       LDA FNS       DIVIDE THIS QUANTITY
1333  03444 026211       LDB ADRP        BY THE MAXIMUM VALUE
1334  03445 162013       JSM FD,I        PLOT VARIABLE
1335*
1336  03446 022211       LDA ADRP      LOAD THE ADDRESS OF THE RESULT
1337  03447 025667       LDB SHIFT     LOAD THE ADDRESS OF SHIFTED VARIABLE
1338  03450 031724 FDV   STA ADR1      STORE THE ADDRESS OF THE DIVIDEND
1339  03451 166013       JMP FD,I      DIVIDE (A) BY (B)

1341*
1342**********************************
1343*
1344***     CONVERT DEGREES, MINUTES, AND SECONDS TO DECIMAL DEGREES
1345*
1346  03452 022153 DMSTD LDA CONV      DIVIDE THE SECONDS QUANTITY
1347  03453 063403       JSM GTOR+1      BY 60 AND REPLACE IT
1348*
1349  03454 063711       JSM XAA-1     ADD Y TO X
1350*
1351  03455 022153       LDA CONV      DIVIDE THIS SUM IN REGISTER
1352  03456 063403       JSM GTOR+1      Y BY SIXTY AND PUT IT IN X
1353*
1354  03457 020006       LDA ADRZ      ADD THE DEGREES QUANTITY TO
1355  03460 063712       JSM XAA         THIS QUANTITY IN X
1356*
1357  03461 020006       LDA ADRZ      CLEAR OUT REGISTERS
1358  03462 170000       CLR             Z AND Y AND RETURN
1359  03463 020005       LDA ADRY      LOAD THE ADDRESS OF Y
1360  03464 066005       JMP RESET+1   RETURN
1361*
1362**********************************
1363*
1364***     CONVERT DECIMAL DEGREES TO DEGREES, MINUTES , AND SECONDS
1365*
1366  03465 063466 DTDMS JSM DSUB      EXECUTE THE SUBROUTINE TWICE AND RETURN
1367*
1368  03466 160227 DSUB  JSM UP,I      DO AN UP
1369  03467 160264       JSM INTX,I    TAKE THE INTEGER VALUE OF X
1370  03470 160234       JSM YMX,I     SUBTRACT X FROM Y
1371  03471 160230       JSM XEY,I     EXCHANGE X AND Y
1372  03472 022153       LDA CONV      MULTIPLY X BY
1373  03473 024004 XAM   LDB ADRX        FLOATING 60
1374  03474 166014       JMP FMP,I       AND RETURN

1376*
1377**********************************
1378*
1379***     RECALL A AND B INTO X AND Y
1380*
1381  03475 020010 RCL   LDA ADRB      TRANSFER THE
1382  03476 024005       LDB ADRY        B REGISTER INTO
1383  03477 170004       XFR             THE Y REGISTER
1384*
1385  03500 020007       LDA ADRA      TRANSFER THE A REGISTER
1386  03501 067612       JMP TTX+1       INTO THE X REGISTER
1387*
1388**********************************
1389*
1390***     SUBTRACT X AND Y FROM A AND B
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1391*
1392  03502 020004 ACM   LDA ADRX      SUBTRACT THE X REGISTER
1393  03503 024007       LDB ADRA        FROM THE QUANTITY
1394  03504 162016       JSM FSB,I       IN THE A REGISTER
1395*
1396  03505 020005       LDA ADRY      SUBTRACT THE Y REGISTER
1397  03506 024010       LDB ADRB        FROM THE QUANTITY
1398  03507 166016       JMP FSB,I       IN THE B REGISTER
1399*
1400**********************************
1401*
1402***     ADD X AND Y TO A AND B
1403*
1404  03510 020005 ACP   LDA ADRY      ADD THE Y REGISTER
1405  03511 024010       LDB ADRB        TO THE QUANTITY IN
1406  03512 162015       JSM FAD,I       THE B REGISTER
1407*
1408  03513 020004       LDA ADRX      ADD THE X REGISTER
1409  03514 024007       LDB ADRA        TO THE QUANTITY IN
1410  03515 166015       JMP FAD,I       THE A REGISTER

1412*
1413**********************************
1414*
1415***     CONVERT RECTILLINEAR COORDINATES TO POLAR COORDINATES
1416*
1417  03516 062401 TP    JSM YTP       SAVE Y IN REGISTER P
1418*
1419  03517 025740       LDB ADRYE     LOAD THE EXPONENT WORD OF Y
1420  03520 074044       SBL 15        LOCATE THE MANITSSA SIGN IN THE MSB OF B
1421  03521 021750       LDA ADRXE     LOAD THE EXPONENT WORD OF X
1422  03522 050063       AND P1        ISOLATE THE MANTISSA SIGN
1423  03523 074017       ADA B         COMBINE THIS WITH B AND
1424  03524 031670       STA COUNT       STORE THESE MANTISSA SIGNS IN COUNT
1425*
1426  03525 021751       LDA ADRXE+1   SKIP IF THE X REGISTER
1427  03526 072250       RZA *+5         IS NOT ZERO
1428*
1429  03527 063420       JSM ABS       MAKE Y POSITIVE
1430  03530 022146       LDA PI/2      SUBTRACT PI/2
1431  03531 063612       JSM TTX+1       FROM X AND JUMP TO
1432  03532 067554       JMP NOMO        FINALIZE IT
1433*
1434  03533 020004       LDA ADRX      DIVIDE THE P REGISTER (Y)
1435  03534 026211       LDB ADRP        BY X AND PUT THE
1436  03535 063450       JSM FDV         ANSWER IN P
1437*
1438  03536 160212       JSM XSQ,I     CALCULATE X SQUARED
1439  03537 160230       JSM XEY,I     EXCHANGE IT WITH Y
1440  03540 160212       JSM XSQ,I     CALCULATE Y SQUARED
1441*
1442  03541 063711       JSM XAA-1     ADD Y TO X
1443*
1444  03542 160276       JSM FSR,I       TAKE THE SQUARE ROOT OF IT
1445  03543 160230       JSM XEY,I     STORE THE NASWER IN Y
1446*
1447  03544 062405       JSM PTX       RECALL THE QUOTIENT Y/X INTO X
1448*
1449  03545 062633       JSM ATNS      CALCULATE THE ARCTAN IN RADIANS OF IT
1450*
1451  03546 063406       JSM ABSX      FORCE THERESULT OF IT IN X POSITIVE
1452  03547 021670       LDA COUNT     LOAD THE ORIGINAL MANTISSA SIGNS
1453  03550 070211       SLA NOMO      SKIP IF THE SIGN OF X WAS POSITIVE
1454*
1455  03551 045750       ISZ ADRXE     OTHERWISE SUBTRACT THE
1456  03552 020064       LDA PI          X REGISTER FROM PI AND
1457  03553 063712       JSM XAA         LEAVE THE ANSWER IN X
1458*
1459  03554 021670 NOMO  LDA COUNT     LOAD THE ORIGINAL MANTISSA SIGNS
1460  03555 070113       SAP *+2       IF THE ORIGINAL MANTISSA SIGNS
1461  03556 160232       JSM FSN,I       WAS NEGATIVE CHANGE THE SIGN OF X
1462*
1463  03557 063354       JSM EATN      CONVERT X TO THE PROPER UNITS
1464  03560 164230       JMP XEY,I     EXCHANGE REGISTERS X AND Y AND RETURN
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1466*
1467*************************************
1468*
1469***      CONVERT POLAR COORDINATES TO RECTILLINEAR COORDINATES
1470*
1471  03561 160230 TR     JSM XEY,I      EXCHANGE X AND Y
1472  03562 062411        JSM SIN        CALCULATE THE SINE OF X
1473*
1474  03563 160230        JSM XEY,I      EXCHANGE X AND Y
1475  03564 160236        JSM MULT,I     CALCULATE XSIN Y IN Y
1476*
1477  03565 025674        LDB SAVE       SKIP IF THE COSINE IS NOT ZERO
1478  03566 074550        SZB CLX1       SET X=0 IF IT IS
1479*
1480  03567 021671        LDA RCPFG      LOAD THE RECIPROCAL FLAG FROM
1481  03570 010772        CPA P2         THE SINE CALCULATION AND SKIP
1482  03571 067576        JMP TCS        IF IT IS 2
1483*
1484  03572 025741        LDB ADRYE+1    SKIP IF THE Y REGISTER
1485  03573 074210        SZB NZ           IS ZERO
1486*
1487  03574 001677        ADA GFG        ADD THE G FLAG TO THE RECIP FLAG
1488  03575 010063        CPA P1         CHANGE THE SIGN OF X  IF
1489  03576 160232 TCS    JSM FSN,I        THIS RESULT IS 1
1490  03577 022204 NZ     LDA FINAL      LOAD THE ADDRESS OF SQR(1-TANY+2)
1491  03600 067403        JMP GTOR+1     AND DIVIDE X BY THIS AND RETURN
1493*
1494*************************************
1495*
1496***      ROUND Y TO THE EXPONENT IN X
1497*
1498  03601 164237 CLX1   JMP CLX,I      STORE ZERO IN X AND RETURN
1499*
1500  03602 160264 RNDY   JSM INTX,I     CALCULATE X=INTX
1501*
1502  03603 020005        LDA ADRY       TRANSFER Y INTO A
1503  03604 062377        JSM RITT+1       WORKING TEMPORARY
1504*
1505  03605 021750        LDA ADRXE      CHECK TO SEE IF X IS
1506  03606 000032        ADA EN2          <100
1507  03607 070312        SAM *+6        SKIP IF IT IS
1508  03610 071451        SLA CLX1       STORE ZERO IF X IS POSITIVE
1509*
1510  03611 020016 TTX    LDA AR2        OTHERWISE SET THE X
1511  03612 024004        LDB ADRX         REGISTER EQUAL TO THE
1512  03613 170004        XFR              Y REGISTER
1513  03614 170402        RET              AND RETURN
1514*
1515*
1516  03615 021750        LDA ADRXE      LOAD THE EXPONENT WORD OF X
1517  03616 025751        LDB ADRXE+1    LOAD THE FIRST 4 DIGITS  OF X
1518  03617 074542        SBR 12         ISOLATE THE FIRST DIGIT OF X
1519  03620 070342        SAR 8          ALLIGN THE EXPONENT OF X
1520  03621 070431        SLA MET,C      SKIP IF THE EXPONENT OF X IS 1
1521*
1522  03622 074744        SHL 1          OTHERWISE ADD THE
1523  03623 074017        ADA B            FIRST DIGIT OF X
1524  03624 074744        SHL 2          INTO B TEN TIMES
1525  03625 074017        ADA B            TO CONVERT IT TO BINARY
1526  03626 025751        LDB ADRXE+1    THEN ISOLATE THE
1527  03627 074604        SHL 4            SECOND DIGIT
1528  03630 074542        SBR 12           OF X
1529*
1530  03631 074017 MET    ADA B          ADD THIS UNITS DIGIT TO THE SUM (POSSIBLY 0)
1531  03632 025750        LDB ADRXE      LOAD THE EXPONENT WORD
1532  03633 074111        SLB *+2          OF X AND SKIP IF
1533  03634 067636        JMP *+2          THE MANTISSA SIGN BIT IS 0
1534  03635 070076        TCA            COMPLEMENT THE SUM IF IT IS 0
1535*
1536  03636 025754        LDB AR2E       LOAD THE EXPONENT WORD OF Y
1537  03637 074340        ABR 8          ALLIGN IT
1538  03640 070037        ADB A          ADD IT TO THE SUM
1539  03641 074076        TCB              AND SUBTRACT THIS
1540  03642 006164        ADB P11          QUANTITY FROM 11
1541  03643 074452        SHM RND2       SKIP IF THE RESULT IS NEGATIVE
1542*
1543  03644 074117        LDA B          DUPLICATE THIS QUANTITY
1544  03645 002172        ADA N13          AND CHECK TO SEE THAT
1545  03646 070112        SAM *+2          IT IS LESS THAN 12
1546  03647 164237        JMP CLX,I      STORE ZERO IN X IF IT IS NOT AND RETURN
1547  03650 035667        STB SHIFT      SAVE THIS SHIFT COUNT
1548  03651 162011        JSM ROUND,I    SHIFT AND ROUND Y
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1549*
1550  03652 171450        NRM             RENORMALIZE IT
1551  03653 015667        CPB SHIFT       STORE THIS RESULT IF THE NORMALIZE
1552  03654 066703 RND2   JMP SOUT          COUNT = THE SHIFT COUNT
1553  03655 024105        LDB P256        OTHERWISE INCREMENT THE
1554  03656 062252        JSM NORM+4        EXPONENT OF THE ANSWER BY 1
1555  03657 066703        JMP SOUT          AND RETURN THE ANSWER IN X
1557*
1558**********************************
1559*
1560***     PROCESS THE TABLE FUNCTION
1561*
1562  03660 062356 F      JSM NXCD1       FETCH THE NEXT KEYCODE
1563*
1564  03661 016167        CPB P34         JUMP TO THE SCALE ROUTINE
1565  03662 067422        JMP SCALE         IF IT IS FORMAT
1566*
1567  03663 014100        CPB RETC        JUMP TO THE DULUP ROUTINE
1568  03664 067733        JMP DULUP         IF IT IS RETURN
1569*
1570  03665 074117        LDA B           DUPLICATE THE CODE
1571  03666 016166        CPB CLXC        JUMP TO THE CLEAR-ALL ROUTINE
1572  03667 067674        JMP CLRR          IF IT IS CLX
1573*
1574  03670 000024        ADA N10         SUBTRACT 10 FROM THE CODE
1575  03671 070713        SAP RET3        RETURN IF THE RESULT IS POSITIVE
1576*
1577  03672 002152        ADA FTABL       OTHERWISE CALCULATE THE PROPER
1578  03673 066002        JMP JTA           FUNCTION AND JUMP TO IT
1579*
1580**********************************
1581*
1582***     CLEAR ALL OF USER AREA
1583*
1584  03674 023710 CLRR   LDA ADRL        LOAD THE ADDRESS OF THE END OF USER MEMORY
1585  03675 026143        LDB N108        LOAD THE NEGATIVE OF THE NO. OF REGISTERS+1
1586  03676 067772        JMP CLER+2      JUMP TO CLEAR ALL DIGITAL REGISTERS
1587*
1588**********************************
1589*
1590***     TURN ON THE PROPER OPTION LIGHT
1591*
1592  03677 021776 DEG    LDA UNITS       LOAD THE STATUS WORD
1593  03700 052203        AND UNITM       CLEAR THE OPTION FLAGS
1594*
1595  03701 014034        CPB P3          SET THE DEGREE, RADIAN OR
1596  03702 026201        LDB RADF          GRAD FLAG DEPENDING ON THE
1597  03703 074106        RBR 3             COUNT (1,2 OR 3) IN THE
1598  03704 074017        ADA B             B REGISTER
1599*
1600  03705 172160        OTA 16          TURN ON / OFF THE PROPER LIGHTS
1601  03706 031776        STA UNITS       STORE THE NEW STATUS WORD
1602  03707 170402 RET3   RET               AND RETURN
1603*
1604*
1605  03710 001000 ADRL   OCT 1000        LAST WORD OF USER REGISTERS
1606  03677        RAD    EQU DEG         THE RADIAN, GRAD, AND DEGREE
1607  03677        GRA    EQU DEG           ROUTINES ARE ONE
1609*
1610**********************************
1611*
1612***     ADD Y TO X
1613*
1614  03711 020005        LDA ADRY        LOAD THE ADDRESS OF Y
1615*
1616  03712 024004 XAA    LDB ADRX        LOAD THE ADDRESS OF X
1617  03713 166015        JMP FAD,I       JUMP TO THE FLOATING ADD ROUTINE
1618*
1619**********************************
1620*
1621***     EXECUTE THE USER DEFINED FUNCTION
1622*
1623  03714 122217 USER   LDA GSBA,I      SET UP THE GOTO SUB
1624  03715 031721        STA SUBF          LINKAGE
1625  03716 022222        LDA USERC       LOAD THE USER KEY CODE
1626  03717 124103        LDB EXCF,I      LOAD THE EXECUTE FLAG AND
1627  03720 074110        SZB *+2           SKIP IF IT IS ZERO
1628  03721 166212        JMP SRCH,I      IF IN PROGRAM CONTROL JUMP TO THE ROUTINE
```

MATH BLOCK FOR THE 9810, CALLED 'CORD3'—Continued

```
1629*
1630  03722 162212        JSM SRCH,I    OTHERWISE FIND THE ADDRESS OF THE ROUTINE AND RETURN
1631  03723 021776        LDA UNITS     LOAD THE STATUS WORD
1632  03724 050066        AND ERR       IF THE ERROR BIT IS SET
1633  03725 073110        RZA RET3        DO A RETURN
1634  03726 021710        LDA USP       OTHERWISE DECREMENT THE
1635  03727 000031        ADA N1          STACK POINTER IN A AND
1636  03730 026161        LDB ENDA        STORE AN END STATEMENT IN THE
1637  03731 070577        STB A,I         NEXT LOCATION TO BE EXECUTED
1638  03732 064132        JMP EXEC      JUMP TO THE EXECUTER

1640*
1641************************************
1642*
1643***      EXECUTE THE DO-LOOP FUNCTION
1644*
1645  03733 062373 DULUP  JSM XTP         STORE X IN P
1646  03734 060115        JSM NXCD        JUMP TO GET THE NEXT CODE
1647*
1648  03735 070704        SAL 2           MULTIPLY IT BY 4
1649  03736 070076        TCA             COMPLEMENT IF AND
1650  03737 000033        ADA ADRF          SUBTRACT IT FROM THE ADDRESS OF R0
1651  03740 031664        STA TEMP        STORE THIS REGISTER ADDRESS AWAY
1652*
1653  03741 063612        JSM TTX+1       TRANSFER THE COUNTER INTO X
1654*
1655  03742 063405        JSM PRE         TAKE THE INTEGER, ABSOLUTE VALUE OF IT
1656  03743 074710        SZB DL2         EXECUTE A RETURN IF IT IS ZERO
1657  03744 024004        LDB ADRX        SUBTRACT ONE FROM THE
1658  03745 162016        JSM FSB,I         COUNTER IN X
1659*
1660  03746 021751        LDA ADRXE+1     IF IT IS ZERO WE ARE DONE SO
1661  03747 070510        SZA DL2           JUMP TO EXIT
1662*
1663  03750 055710        DSZ USP         DECREMENT THE USER STACK POINTER
1664  03751 121710        LDA USP,I       LOAD THE LAST RETURN ADDRESS
1665  03752 002173        ADA N6          SUBTRACT SIX FROM IT
1666  03753 025707        LDB ZZAD        LOAD THE ADDRESS OF STEP 00
1667  03754 074076        TCB             COMPLEMENT IT AND
1668  03755 070037        ADB A           ADD IT TO THE ABOVE ADDRESS
1669  03756 074112        SBM DL1         SKIP IF THE RETURN ADDRESS IS ILLEGAL
1670*
1671  03757 131710        STA USP,I       OTHERWISE STORE THE NEW RETURN ADDRESS
1672*
1673  03760 045710 DL1    ISZ USP         RESTORE THE USER STACK POINTER
1674*
1675  03761 020004 DL2    LDA ADRX        TRANSFER THE NEW VALUE OF THE
1676  03762 025664        LDB TEMP          COUNTER BACK INTO
1677  03763 170004        XFR               ITS REGISTER
1678  03764 062405        JSM PTX         RESTORE THE X REGISTER
1679*
1680  03765 024100        LDB RETC        LOAD A RETURN KEYCODE
1681  03766 064362        JMP INTP        JUMP TO THE INTERPRETER
1682*
1683*
1684  02152          THETL EQU GPERR      CORDIC LOG CONSTANT TABLE POINTER
1685  02152          FTABL EQU GPERR      TABLE FUNCTION LINK WORD
1686  02146          LINK  EQU PI/2       KEYCODE ROUTINE LINK WORD

1688*
1689************************************
1690*
1691* CHECKSUM WORD FOR THE SECOND 512 WORDS (3000-3777)
1692*
1693  03767 000000 CKSM2  NOP
1694*
1695************************************
1696*
1697***      CLEAR OUT A SPECIFIED NO. OF FLOATING POINT REGISTERS
1698*
1699  03770 020012 CLER   LDA AWRDN       LOAD THE ADDRESS OF THE DIGIT STACK
1700  03771 026170        LDB N2          LOAD THE NO. OF CLR'S MINUS ONE
1701  03772 170000        CLR             CLEAR 4 WORDS
1702  03773 000003        ADA P4          ADD 4 TO THE ADDRESS
1703  03774 077730        RIB *-2         SKIP IF THE COUNT IS NOT ZERO AND ADD 1 TO IT
1704  03775 000031        ADA N1          OTHERWISE SUBTRACT 1 FROM IT
1705  03776 031666 KK     STA DIGIT         AND STORE THIS ADDRESS IN DIGIT
1706  03777 170402        RET             THIS WORD IS NOT NECESSARY
1707*
1708                      END
*  NO ERRORS*
```

CROSS-REFERENCE SYMBOL TABLE

A	0000/00	0093	0094	0468	0669	0717	0840	0842	0925	1281	1283
	1538	1637	1668								
ABS	1300	0261	1429								
ABSX	1278	1301	1451								
ACM	1392	0301									
ACP	1404	0298									
ACS	0576	0569									
ACSC	0292	0568									
ACSS	0582	0579									
ADR0	0066	0095	1024								
ADR1	0062	0334	0338	0378	0940	1338					
ADR2	0063	0327	0702	1279							
ADRA	0068	1385	1393	1409							
ADRB	0069	1381	1397	1405							
ADRF	0070	1650									
ADRL	1605	1584									
ADRP	0296	0491	0504	0965	1259	1325	1329	1333	1336	1435	
ADRPE	0031	0296	0968	1258							
ADRT	0067	0336	0437	0441	0947						
ADRTE	0030	0944									
ADRX	0064	0323	0326	0429	0438	0440	0490	0532	0701	0937	1026
	1269	1278	1309	1373	1392	1408	1434	1511	1616	1657	1675
ADRXE	0029	0421	0711	0800	0957	1285	1421	1426	1455	1505	1516
	1517	1526	1531	1660							
ADRY	0065	0498	0808	1300	1313	1359	1382	1396	1404	1502	1614
ADRYE	0032	0803	1419	1484							
ADRZ	0071	1354	1357								
AR1	0040	0479	0678	0680	1211						
AR1E	0027	0699									
AR2	0041	0352	0486	0495	0507	0904	0935	1182	1510		
AR2E	0028	0350	0351	0362	0373	0385	0403	0404	0405	0409	0536
	0600	0624	0634	0644	0648	0664	0668	0698	0744	0751	0766
	0770	0819	0865	0926	0929	1041	1536				
ARC	0562	0308									
ASN	0588	0565									
ASNC	0061	0564									
ASNS	0710	0577	0588								
ATN	1235	0567									
ATNC	0293	0566									
ATNS	0728	1235	1449								
AWRDN	0058	0848	1061	1126	1699						
AWPS	1461	0642									
	0000/00	0390	1098	1423	1523	1525	1530	1543	1570	1598	
BACK1	0872	0805									
B	0195	0271	0294								
C	0216	0258									
D	0245	0259									

CROSS-REFERENCE SYMBOL TABLE — Continued

C7	0250	0260									
C8	0160	0263									
C9	0241	0264									
CK	0442	0470									
CKSM1	0282										
CKSM2	1693										
CLER	1699	0742	0845	0893	1586						
CLRR	1584	1572									
CLX	0135	1498	1546								
CLX1	1498	1478	1508								
CLXC	0275	1571									
CONV	0264	1346	1351	1372							
COS	0551	0309									
COUNT	0019 1452	0364 1459	0369	0371	0416	0427	0435	0828	0834	0837	1424
CS	0536	0526									
D0010	0284	0695	0921								
DEG	1592	0144	1252	1606	1607						
DIGIT	0017 1114	0850 1127	0855 1142	0857 1165	0895 1168	0990 1185	0995 1214	1003 1225	1037 1227	1060 1705	1069
DIV	0334	0683	0938								
DL1	1673	1669									
DL2	1675	1656	1661								
DMSTD	1346	0149									
DPERR	0260	1256	1262								
DSUB	1368	1366									
DTDMS	1366	0150									
DTOR	1262	1254									
DULUP	1645	1568									
EATN	1237	0582	0589	1463							
EN12	0269	0736									
EN2	0056	0958	1506								
EN3	0268	0728	0821								
END	0257	0270									
ENDA	0270	1636									
EP3	0202	0267	0928								
ERR	0074	1632									
ERROR	0285	0891									
EXCF	0075	0469	1626								
EXEC	0077	1638									
EXP	0816	0311									
F	1562	0295									
FAC3	0964	0972									
FAD	0116	0946	1406	1410	1617						
FAL1	0623	0608									
FALSE	0105	0424	0623	0738							
FD	0114	1334	1339								

CROSS-REFERENCE SYMBOL TABLE — Continued

FDV	1338	0530	0685	0721	1270	1436					
FINAL	0291	0533	1490								
FMP	0115	0933	1374								
FMT	0132	1315									
FNCFG	0021	0322	0792	0888	1029	1038	1084	1104	1113	1131	1135
	1153	1223									
FNS	0267	1332									
FRAC	0360	0606	0825								
FSR	0117	0966	1321	1330	1394	1398	1658				
FSN	0134	1461	1489								
FSR	0125	0433	1444								
FTABL	1685	1577									
GET	0638	0603	0630	0637							
GET1	0602	0609									
GFG	0026	0451	0578	0602	0697	0713	1487				
GONO	0780	0767									
GOON	1164	1143	1161								
GPERR	0263	1265	1268	1684	1685						
GRA	1607	0146									
GRAF	0289	0422									
GSBA	0302	1623									
GTOR	1268	0360	0793	1248	1263	1347	1352	1491			
IN	0985	0662	0852								
IN2	1019	0672	0750								
IN3	1178	0763	0899								
INIT	0322	0452	0729	0817	0883						
INTP	0080	1681									
INTX	0130	1277	1369	1500							
IOD	1223	1189	1218								
JTA	0093	1578									
KBUF	2073	0091									
KK	1705	1062									
KYLOG	0083	0471									
LINK	1686	0092									
LN10	0294	0791	0824	0898	0932	1293					
LNX	0882	0303	0790	0807							
LOG	0790	0147									
LT3	0834	0830									
MET	1530	1520									
MODE	1243	0453									
MULT	0133	1475									
N1	0051	0576	0759	0849	0882	1087	1206	1237	1635	1704	
N10	0054	0919	1574								
N120	0266	0913									
N1W8	0256	1585									
N12	0278	0391	1099								

CROSS-REFERENCE SYMBOL TABLE—Continued

N13	0279	0757	1544								
N2	0277	0539	1700								
N256	0055	0367	0633								
N4	0052	1201	1324								
N5	0053	0854	1155	1209							
N6	0280	1665									
NAN	1221	1205									
NOMO	1459	1432	1453								
NORM	0346	0335	0339	0375	0629	0780	0930	1554			
NXCD	0079	0467	1646								
NXCD1	0467	0562	1562								
NZ	1490	1485									
OK	0740	0718	0735	0737							
ONE	0057	0430	0478	0501	0964	1286					
ONVX	0126	0657	0740	0872							
OTO	0478	0461	0483	0508	0860	1022					
OTT	0501	0525	1133								
OUT	0699	0545	0656	0659	0693	0783					
OUTS	0948	0956									
P1	0042 1187	0598 1251	0632 1422	0639 1488	0710	0716	0769	0816	0887	1063	1154
P10	0047	0923	1145								
P100	0265	0916									
P11	0273	1540									
P12	0048	0748	1129	1149	1197						
P13	0274	0393	0760								
P2	0043	1481									
P22	0049	0747									
P256	0050	0665	0689	0908	0914	1553					
P2T0	0482	0611	0775								
P3	0044	1595									
P34	0276	1564									
P4	0045	0997	1702								
P6	0272	0671	0755	0851	0897						
P7	0046	0650									
PI	0059	0580	1456								
PI/2	0259	0482	0552	1430	1686						
POOT	0765	0739									
PRE	1277	0955	1655								
PTA	0988	0969	1011								
PTB	1034	1115									
PTC	1072	1035									
PTD	1037	1118									

CROSS-REFERENCE SYMBOL TABLE—Continued

PTE	1047	1030									
PTF	1117	1054									
PTF1	1054	1070									
PTG	1168	1159	1162								
PTH	1147	1140	1166								
PTI	1142										
PTJ	1126	0862	0894								
PTK	1111	1100									
PTL	1063	1031									
PTL1	1031	1039									
PTM	1074	1059									
PTN	1049	1034	1045								
PTP	1185	1191									
PTQ	1201	1220									
PTR	1135	1170									
PTS	1139	1132									
PTX	0504	1447	1678								
PXX	1259	0971									
RITT	0494	0328	0346	0502	1052	1082	1111	1503			
RAD	1636	0145									
RADF	0286	1596									
RCL	1381	0299									
RCPFG	0220 0843	0538 0864	0599 1480	0615	0620	0622	0654	0658	0674	0688	0691
RESET	0095	1360									
RET2	1228	1250									
RET3	1602	0143	1575	1633							
RET6	0794	0804									
RETC	0060	1567	1680								
REXP	0807	0801	1294								
RND2	1552	1541									
RNDY	1540	0152									
ROUND	0106	0395	0640	1065	1102	1548					
RTOD	1256	1266									
RTOG	1265	1247									
RY1	0025	0487	0494	0677	0681	1025	1074	1077			
RY2	0024	1027	1075	1076	1081						
SAVE	0023	0523	0752	0772	1019	1095	1096	1477			
SCALE	1308	1565									
SCL1	1318	1310	1314								
SCRT	0416	0528	0714								
SCRT1	0435	0720									

CROSS-REFERENCE SYMBOL TABLE — Continued

SETUP	0081	0097									
SHIFT	0018	0986	1010	1020	1086	1117	1130	1147	1169	1180	1190
	1198	1204	1219	1319	1337	1547	1551				
SIGN	0022	0324	0541	0631	0635	0638	0687	0712	0765	0869	0907
	0942										
SIN	0519	0306	0555	1472							
SOUT	0782	0874	1552	1555							
SWCH	0297	1628	1630								
STM1	0832										
STMAX	0104	0626	0832	0889	0960						
STN	0449	0519	0551	0596							
SUBF	0078	1624									
TABLE	0016	0459	0460	0985	1146	1160	1178	1243	1253		
TAN	0596	0307	0521								
TCS	1489	1482									
TEMP	0015	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025
	0026	0450	0520	0625	1179	1195	1318	1323	1328	1651	1676
TEMX	1293	0148									
TERM	0137	0703									
THETL	1684	0847									
THETT	0271	0647									
TP	1417	0300									
TM	1471	0304									
TTO	0507	1109	1139	1164							
TTP1	0486	1047	1067	1079							
TTX	1510	0340	0377	0480	0488	0492	0496	0505	0653	0722	0871
	0941	0948	1386	1431	1653						
TWOPI	0258	0605	0762								
UNITM	0290	1593									
UNITS	0072	1244	1592	1601	1631						
UNRM	0385	0773	0826								
UP	0128	1368									
UPARO	0800	0310									
UPTAH	0459	0998	1202								
USER	1623	0262									
USERC	0305	1625									
USP	0076	1634	1663	1664	1671	1673					
XAA	1616	0431	0553	0581	1349	1355	1442	1457			
XAM	1373	0379	0809	1260							
XAT	0440	0418	0684	0945							
XEY	0129	1371	1439	1445	1464	1471	1474				
XFACT	0955	0151									
XMIN	0286	1308									
XSQ	0127	0425	1438	1440							
XTP	0490	0499	0962	1257	1645						

CROSS-REFERENCE SYMBOL TABLE—Continued

```
XTT     0326    0554

YMIN    0287    1312

YMX     0131    1370

YTP     0498    1417

ZERO    0402    0607    0617    0731    0885    0988    1058

ZZAD    0082    1666

END JOB        RUN=0030 MIN. 02.5 SEC.  EXEC=0000 MIN. 00.0 SEC.

JOB       6/1/71      TIME=0064 MIN. 27.2 SECS.
```

—Continued

```
0001                ASMB,A,L,T,B
OFF     016000
END     016002
FMTGT   016004
WRITM   016012
FMTRC   016014
WRIT1   016024
PRTED   016027
READM   016040
FCHM    016052
NTERM   016053
QUIT    016067
LDATA   016072
INITM   016077
MORGT   016104
ENTR    016123
LDDTA   016124
LCALL   016127
ALLOK   016133
RDATA   016134
BYTE    016144
FDATA   016146
.REC.   016164
DOIT    016170
CONT    016172
ENDIT   016176
MORE    016177
RECRD   016201
RCALL   016204
OUH     016210
OKGO    016212
DON     016221
.LOD.   016222
WDGET   016226
GO.ON   016227
OK      016232
CRDIN   016240
LOOK    016241
STOPK   016247
JMP10   016254
STOP    016256
GET     016267
WDOUT   016274
BTOUT   016300
JMP14   016317
OUT1    016322
OUT2    016325
DATIN   016330
WDBLD   016331
FLAG    016332
JMP4    016341
EOCL    016345
.END1   016352
.END.   016354
DATST   016363
DP1     000063
SETUP   000031
SAFE    001737
PRGAD   001701
INBUF   001775
MASK    016364
M2      016365
BTCNT   001736
FMTOT   016366
HOLD    001735
SECUR   000111
PCSAV   001734
DP7     000770
LLON    000105
ELON    000066
STATS   001776
PC      001701
ZPC     001707
STKPT   001777
M1      000031
CRDFG   001733
MAXAD   016367
CNT     001732
PTR     001731
BASE    000375
MAXWD   016370
M3      000077
OCT77   000100
KLUDG   016371
DELAY   016372
RC1     016373
ALLCR   016760
CLRM    016764
CKIT    016766
TEMP    001730
DATA    001730
*  NO ERRORS*
```

MAGNETIC CARD BLOCK.

```
0001                    ASMB,A,L,T,B
0003  00302                ORG 302B
0004  00302 016012         DEF WRITM
0005  00303 016040         DEF READM
0006  00114                ORG 114B
0007  00114 016760         DEF ALLCR          CLEAR MEMORY ROUTINE.
0008  00172                ORG 172B
0009  00172 016134         DEF RDATA
0010  00173 016004         DEF FMTGT
0011  00174 016014         DEF FMTRC
0012  00175 016072         DEF LDATA
0013  16000                ORG 16000B
0014  16000 000037 OFF     OCT 37
0015  16001 000035         OCT 35          THIS IS THE CLX, DIV, END TO CATCH PC.
0016  16002 000046 END     OCT 46
0017  16003 064161         JMP 161B
0018*           MAG CARD ROUTINES.
0019*
0020  16004 021707 FMTGT LDA ZPC          FORMAT GOTO ROUTINE.
0021  16005 031701       STA PC           SET USER PROGRAM CON. TO 0.
0022  16006 062040       JSM READM        LOAD CARD.
0023  16007 021707       LDA ZPC
0024  16010 031701       STA PC           SET PC TO ZERO FOR EXECUTION.
0025  16011 064134       JMP 134B        JUMP TO THE COTINUE ENTRY
0026*
0027  16012 120111 WRITM LDA SECUR,I        GET AND CHECK SECURITY WORD.
0028  16013 070450       SZA WRIT1          IF ZERO, OK TO RECORD.
0029  16014 120111 FMTRC LDA SECUR,I    FORMAT RECORD ROUTINE.
0030  16015 070110       SZA *+2
0031  16016 170402       RET              DO NOT RESPOND IF SEC. WD.
0032  16017 022002       LDA END           SET SECURITY E TO NON ZERO AND NOT 1'S.
0033  16020 130111       STA SECUR,I      SET SW TO NON ZERO.
0034  16021 062024       JSM WRIT1         DO NORMAL RECORD,
0035  16022 070742       SAR 16            CLEAR SECURITY WORD,
0036  16023 066070       JMP QUIT+1        AND RETURN.


0038  16024 072052 WRIT1 SAM *+1,S          NEGATIVE FOR STOP PROCESSING.
0039  16025 031775       STA INBUF
0040  16026 066164       JMP .REC.
0041  16027 024031 PRTED LDB SETUP        SET HEADS TO SAFE CONDITION.
0042  16030 062325       JSM OUT2
0043  16031 070137       LDB A             SAVE A TO USE A FOR IOR OPERATION.
0044  16032 020066       LDA ELON         TURN ON ERROR LIGHT.
0045  16033 041776       IOR STATS
0046  16034 172160       OTA 16          OUTPUT TO LIGHT LATCHES.
0047  16035 074117       LDA B          RESTORE A, WHICH IS CODE TO RECORD.
0048  16036 062354       JSM .END.         EXPELL CARD, AND WAIT FOR ANOTHER.
0049  16037 066203       JMP RECRD+2


0051*                LOAD ROUTINE.
0052  16040 021776 READM LDA 17768          GET STATUS WORD.
0053  16041 070504       SAL 6              RETURN WITHOUT LOADING IF
0054  16042 070112       SAM *+2           IN PROGRAM MODE, ELSE CAR LOOK AT
0055  16043 170402       RET              SECURE PROGRAM.
0056  16044 021701       LDA PC            PREPARE TO LSAVE USER PC.
0057  16045 031734       STA PCSAV        USE IN LATER CLEAR MEMORY.
0058  16046 072052       SAM *+1,S        NEGATIVE FOR STOP PROCESSING.
0059  16047 031775       STA INBUF
0060  16050 070742       SAR 16             CLEAR OUT TEMP SW LOCATION
0061  16051 031737       STA SAFE           FOR SUBSEQUENT ORING OPERATIONS.
0062  16052 062222 FCHM  JSM .LOD.       LOAD THE CARD.
0063  16053 021737 NTERM LDA SAFE        RET SW FROM CARD.
0064  16054 070110       SZA *+2          CARD JUST READ.
0065  16055 066067       JMP QUIT         WAS NONZERO, NOT MUCH TROUBLE.
0066  16056 120111       LDA SECUR,I      GET CURRENT SECURITY WORD.
0067  16057 070410       SZA QUIT         ALL ZERO, QUIT.
0068  16060 021707       LDA ZPC           GET RELATIVE ORIGIN OF USER AREA.
0069  16061 025734       LDB PCSAV        GET START OF LOADED PROGRAM.
0070  16062 062764       JSM CLRM          CLEAR MEMORY FROM A TO B-1.
```

MAGNETIC CARD BLOCK.—Continued

```
0071  16063 021701        LDA PC          GET CURRENT END OF USER PROGRAM.
0072  16064 070070        STA *+1         INCREMENT TO AVOID CLEARING LAST LOCATION.
0073  16065 026367        LDB MAXAD       B IS MAXIMUM USER PROGRAM ADDRESS.
0074  16066 062764        JSM CLRM        CLEAR FROM END OF USER PROG TO UPPER LIMIT.
0075  16067 021737 QUIT   LDA SAFE
0076  16070 134111        STA SECUR,I     SET UP THE NEW SECURITY WORD.
0077  16071 170442        RET             END ROUTINE.


0078*              CARD READER UTILITY ROUTINES.
0079*
0080*
0081  16072 022370 LDATA  LDA MAXWD       SET UP UPPER BOUND ON DATA(REGISTER 0).
0082  16073 031731        STA PTR
0083  16074 024031        LDA M1          SET CARD IN FLAG TO UNCHECKED.
0084  16075 031733        STA CRDFG
0085  16076 031730        STA DATA        TELLS ROUTINES THAT DATA IS BEING RECORDED.
0086  16077 072742 INITM  SAR 16          INITIALIZE CONTENTS OF PTR.
0087  16100 131731        STA PTR,I
0088  16101 020477        LDA MB          SET UP BYTE COUNTER AND STOP PROCESSING.
0089  16102 031775        STA INBUF       STOP PROCESSING.
0090  16103 031732        STA CNT         BYTE CONTER.
0091  16104 062124 MORGT  JSM LDDTA       GET 6 BITS OF DATA IN A REGISTER.
0092  16105 125731        LDB PTR,I
0093  16106 074246        RAR 6           POSITION TO FORM WORD.
0094  16107 074217        IOR B           MERGE DATA AND PREVIOUS DATA.
0095  16110 131731        STA PTR,I
0096  16111 045732        ISZ CNT         END OT THIS WORD?.
0097  16112 066104        JMP MORGT       NO, GET MORE DATA.
0098  16113 125731        LDB PTR,I         YES, POSITION WORD PROPERLY.
0099  16114 074646        RAR 14
0100  16115 135731        STA PTR,I
0101  16116 025731        LDB PTR         GET VALUE OF POINTER, FOR
0102  16117 044375        ADB BASE           CHECK IF ALL DATA REGISTERS HAVE BEEN RECORDED.
0103  16120 074152        SHM ENTR           YES, SKIP TO DRIVE CARD OUT.
0104  16121 055731        DSZ PTR         IT IS NOT ", CONTINUE.
0105  16122 066077        JMP INITM       REPEAT FOR NEXT WORD.
0106  16123 066352 ENTR   JMP .END1
0107  16124 045733 LDDTA  ISZ CRDFG       HAS THE CARD BEEN CHECKED?
0108  16125 066133        JMP ALLOK       YES, GET DATA.
0109  16126 024031        LDB SETUP       NO, CHECK IT.
0110  16127 062240 LCALL  JSM CRDIN          TEST FOR CARD IN.
0111  16130 062330        JSM DATIN          GET FIRST DATA.
0112  16131 041737        IOR SAFE           OR WITH ACCUMULATED CARD SECURITY WORD.
0113  16132 031737        STA SAFE           SAVE ACCUMULATED SECURITY WORD.
0114  16133 066330 ALLOK  JMP DATIN       JUMP TO DATA AND EXECUTE RETURN.


0115  16134 024031 RDATA  LDA M1          BEGIN RECORD DATA ROUTINE.
0116  16135 031733        STA CRDFG       SET CARD FLAG TO UNCHECKED. CAUSES CRDIN
0117  16136 031730        STA DATA        LET ROUTINES KNOW DATA IS BEING RECORDED.
0118  16137 031775        STA INBUF       STOP PROCESSING.
0119  16140 022370        LDA MAXWD       REGISTER 0 O F DATA.
0120  16141 031731        STA PTR         INDEXED VARIABLE.
0121  16142 120111        LDA SECUR,I        GET AND RECORD SECURITY WORD TO
0122  16143 062201        JSM RECRD          KEEP FROM MESSING THINGS UP ON LOAD.
0123  16144 020077 BYTE   LDA M3
0124  16145 031732        STA CNT         SET CNTR FOR 3 6 BIT BYTES.
0125  16146 121731 FDATA  LDA PTR,I        GET 1ST 16 BIT WORD.
0126  16147 074246        RAR 6             GET AND POSITION PROPER 6 BITS.
0127  16150 131731        STA PTR,I       STORE FOR NEXT BYTE OUT.
0128  16151 062201        JSM RECRD       RECORD 6 BITS OF DATA.
0129  16152 045732        ISZ CNT         IND BYTE COUNT.
0130  16153 066146        JMP FDATA         PUT OUT NEXT BYTE.
0131  16154 121731        LDA PTR,I       POSTTION DATA SORD AFTER MESSING IT UP.
0132  16155 074646        RAR 14
0133  16156 131731        STA PTR,I
0134  16157 021731        LDA PTR      CHECK IF ALL
0135  16160 044375        ADA BASE
0136  16161 070652        SAM ENDIT          AREA IN MACHINE IS RECORDED.
0137  16162 055731        DSZ PTR         NO, GET MORE.
0138  16163 066144        JMP BYTE      16 BIT WORD.
0139*
0140  16164 024031 .REC.  LDA M1          BASIC PROGRAM RECORD ROUTINE.
0141  16165 031733        STA CRDFG
0142  16166 070056        CMA             SET A TO ANYTHING BUT -1,
0143  16167 031730        STA DATA        AND STORE TO SHOW THAT ARE "NOT" REC DATA.
0144  16170 120111 DOIT   LDA SECUR,I     RECORD SECURITY WORD.
0145  16171 062201        JSM RECRD
0146  16172 121741 CONT   LDA PRGAD,I     GET PROGRAM WORD.
0147  16173 062201        JSM RECRD
0148  16174 121701        LDA PRGAD,I
0149  16175 012002        CPA END         IS PROGRAM DONE?
0150  16176 066352 ENDIT  JMP .END1         DRIVE CARD OUT THEN EXECUTE RETURN.
0151  16177 045701 MORE   ISZ PRGAD       NO, CONTINUE.
```

MAGNETIC CARD BLOCK.—Continued

```
0155  16200 066172        JMP CONT
0156  16201 045733 RECRD  ISZ CRDFG      TEST IF CARD IN HAS BEEN CHECKED.
0157  16202 066212         JMP OKGO      YES, CONTINUE.
0158  16203 026363        LDB DATST     NO, SET UP PROPER HEAD CONDITIONS.
0159  16204 062240 RCALL  JSM CRDIN     TEST FOR CARDIN.
0160  16205 074046        RBR 2          TEST FOR PROTECTED CARD.
0161  16206 074112        SBM OOH       CARD NOT PROTECTED, CONTINUE.
0162  16207 066027        JMP PRTED     CARD PROTECTED, GET ANOTHER.
0163  16210 024060 OOH    LDB 60B       GET A DELAY ON RECORD TO AVOID
0164  16211 076030        RIB *         MISSING FIRST STROBE ON LOAD.
0165  16212 062274 OKGO   JSM WDOUT      OUTPUT 6 BITS.
0166  16213 062267        JSM GET        GET SENSOR DATA ON END OF CARD.
0167  16214 074146        RBR 4
0168  16215 074212        SBM DON       NOT ENDED, NORMAL RETURN.
0169  16216 062354        JSM .END.     CARD ENDED, EXPELL CARD.
0170  16217 124111        LDA SECUR,I    RECORD SECURITY WORD,
0171  16220 066203        JMP RECRD+2
0172  16221 174402 DON    RET            NORMAL RETURN POINT.
0173*
0174*          LOAD ROUTINE.
0175  16222 024031 .LOD.     LDA M1
0176  16223 031733           STA CRDFG
0177  16224 070456        CMA
0178  16225 031730        STA DATA      SHOW DATA NOT BEING LOADED.
0179  16226 062124 WDGET  JSM LODTA      INPUT 6 BITS.
0180  16227 131701 GO.ON  STA PRGAD,I    STORE PROGRAM WORD.
0181  16230 012002        CPA END        IS PROGRAM DONE?
0182  16231 066352        JMP .END1      EXPELL CARD AND EXECUTE RETURN.
0183  16232 022367 OK     LDA MAXAD     NO, CONTINUE.
0184  16233 070456        CMA                 -15777
0185  16234 001701        ADA PRGAD        IF POSITIVE, HAS OVERFLOWED.
0186  16235 045701        ISZ PRGAD
0187  16236 071412        SAM WDGET
0188  16237 066352        JMP .END1
0189*
0190*
0191  16240 062325 CRDIN  JSM OUT2
0192  16241 062267 LOOK   JSM GET       LOAD SENSOR DATA.
0193  16242 074306        RBR 7        CHECK TOP SENSOR OF CARD.
0194  16243 074213        SBP STOPK      SKIPS IF UNBLOCKED.
0195  16244 074606        RBR 13         CONTINUE THE TEST FOR CARD IN.
0196  16245 075611        SLB LOOK     NOT IN IF SKIP, TY AGAIN.
0197  16246 174402        RET
0198  16247 070137 STOPK  LDB A         SAVE A SO CAN TURN ON LOAD LIGHT.
0199  16250 021776        LDA STATS
0200  16251 240105        IOR LLON       TURN ON LOAD LIGHT.
0201  16252 172160        OTA 16
0202  16253 074117        LDA H        RESTORE A.
0203  16254 025775 JMP1W  LDB INHUF      NO, TEST FRO STOP.
0204  16255 075212        SBM LOOK
0205  16256 062352 STOP   JSM .END1         TURN OFF CRRD READER.
0206  16257 021730        LDA DATA
0207  16260 070212        SAM *+4
0208  16261 055777        DSZ STKPT
0209  16262 121777        LDA STKPT,I
0210  16263 012373        CPA RC1        WAS THIS CALLED FROM RECORD MODE?
0211  16264 064161        JMP 161B
0212  16265 062053        JSM NTERM
0213  16266 064161        JMP 161B



0215  16267 026364 GET    LDB MASK      SET MSB OF IO REG
0216  16270 176141         OTB 01       TO 1.
0217  16271 172741         STF 01
0218  16272 176241         LIB 01
0219  16273 174402         RET
0220  16274 026365 WDOUT  LDB M2         A REG HAS WD TO B RECORDED.
0221  16275 035736         STB BTCNT      SET UP BYTE COUNTER.
0222  16276 026366         LDB FMTOT      STROBE FORMAT.
0223  16277 070456         CMA
0224  16300 031735 BTOUT   STA HOLD     COMP. FOR PROPER POLARITY REC/LOAD.
0225  16301 050770         AND DP7       MASK IN CONTROL BITS.
0226  16302 042364         IOR MASK
0227  16303 172141         OTA 01       A REGISTER OUTPUT.
0228  16304 172642         STC 02
0229  16305 022372         LDA DELAY          THIS DELAY GIVES PACKING DENSITY.
0230  16306 072030         RIA *
0231  16307 176141         OTB 01         OUTPUT STROBE.
0232  16310 172742         STF 02
0233  16311 074006         RBR 1          GET NEXT STROBE.
0234  16312 022372         LDA DELAY         PACKING DENSITY DELAY.
0235  16313 072030         RIA *          DELAY UNTIL 3 MSB ARE READY TO OUTPT.
0236  16314 045736         ISZ BTCNT      TEST FOR END OF WORD.
0237  16315 066317         JMP JMP14
0238  16316 174402         RET            WORD IS FINISHED.
0239  16317 021735 JMP14   LDA HOLD      GET WORD.
```

MAGNETIC CARD BLOCK.—Continued

```
0240  16320 070106        RAR 3
0241  16321 066300        JMP BTOUT     RET OT OUTPUT 3 MSB.
0242  16322 176141 OUT1   OTB 01
0243  16323 172602        STC 02        OUTPUTS BITS 0 1 2 4 5 OF A REG
0244  16324 170402        RET           TO CARD READER.
0245  16325 062322 OUT2   JSM OUT1
0246  16326 172742        STF 02        OUTPUTS 6 LSB OF A TO CR.
0247  16327 170402        RET



0249  16330 070742 DATIN  SAR 16
0250  16331 173742 WDBLD  CLF 02
0251  16332 172502 FLAG   SFS 02        TEST FOR CR STROBE FLAG.
0252  16333 066345        JMP EOCL      NO FLAG. TEST FOR END OF CARD.
0253  16334 062267        JSM GET       YES LOAD DATA.
0254  16335 072212        SAM JMP4,S      SKIP IF THIS IS MS THREE BITS.
0255  16336 074217        IOR B           PUT IN THE CURRENT LS BITS.
0256  16337 052371        AND KLUDG       GET THE EXTRA BITS FROM GET OUT, BUT KEEP MSB.
0257  16340 066331        JMP WDBLD     GET THE MS THREE BITS.
0258  16341 074444 JMP4   SHL 3         POSITION THE MS 3 BITS.
0259  16342 074217          IOR B         MERGE WITH OTHER 3 BITS
0260  16343 050100        AND OCT77     GETS RID O F REST OF BITS FROM GET.
0261  16344 170402          RET           RETURN
0262  16345 062267 EOCL   JSM GET
0263  16346 074346        RBR 7       LOAD SENSOR DATA AND ROTATE END.
0264  16347 075152        SHM FLAG    0 CARD SENSTOR TO MSB.
0265  16350 062354        JSM .END.   CARD ENDED, DRIVE CARD OUT,
0266  16351 066126        JMP LCALL-1       GET CARD, SECURITY WD, ETC.
0267  16352 025776 .END1   LDB STATS       TURN OFF LOAD LIGHT.
0268  16353 176160        OTB 16
0269  16354 062267 .END.  JSM GET
0270  16355 074204        SBL 12
0271  16356 075712        SBM .END.       MAKE SURE CARD PASSED DRIVE ROLLES.
0272  16357 026217        LDB OKGO+5        DELAY TILL CARD HAS PASSED ROLLERS.
0273  16360 076030        RIB *
0274  16361 026000        LDB OFF
0275  16362 066325        JMP OUT2        TURN OUT CARD READER.
0276*
0277  16363 000057 DATSI  OCT 57
0278  20063       DP1   EQU 63B     DEC 1.
0279  00031       SETUP EQU 31B     M1.
0280  01737       SAFE  EQU 1737B
0281  01701       PRGAD EQU 1701B
0282  01775       INBUF EQU 1775B
0283  16364 177740 MASK  OCT 177740
0284  16365 177776 M2    DEC -2
0285  01736       HTCNT EQU 1736B
0286  16366 052525 FMTOT OCT 52525
0287  01735       HOLD  EQU 1735B
0288  00111       SECUR EQU 111B      SECURITY WORD.
0289  01734       PCSAV EQU 1734B
0290  00770       OP7   EQU 770B
0291  00105       LLON  EQU 105B
0292  00066       ELON  EQU 66B       OCT 10000.
0293  01776       STATS EQU 1776H
0294  01701       PC    EQU PRGAD
0295  01707       ZPC   EQU 1707B       FLOATING USER ORIGIN.
0296  01777       STKPT EQU 1777H



0298*******************************
0299*    THINGS INSERTED DURING DEVELOPMENT*****
0300  00031       M1     EQU 31B
0301  01733       CRDFG EQU 1733B
0302  16367 015777 MAXAD OCT 15777       UPPER USER AREA.,
0303  01732       CNT   EQU 1732B
0304  01731       PTR   EQU 1731B
0305  00375       BASE  EQU 375B
0306  16370 001663 MAXWD OCT 1663        UPPER ROUND TO DATA AREA.
0307  00077       M3    EQU 77B
0308  00100       OCT77 EQU 100B
0309  16371 100007 KLUDG OCT 100007      USED TO GET MSB AND THREE LSB.
0310  16372 177754 DELAY OCT 177754
0311  16373 016204 RC1   DEF RCALL       USED TO DETERMINE CALLER ON STOP KEY.



0313  16760             ORG 16760B
0314  16760 070742 ALLCR SAR 16          CLEAR SECURITY WORD.
0315  16761 133111        STA SECUR,I
0316  16762 021747        LDA ZPC         SET TO CLEAR ALL USER MEMORY.
0317  16763 026367        LDB MAXAD
0318  16764 035730 CLRM   STB TEMP          CLEARS FROM A TO B-1. BOMBS IF B<A.
0319  16765 074742        SHR 16          WANT TO DEPOSIT B INTO A, INDIRECT.
0320  16766 011730 CKIT   CPA TEMP        DONE YET?
0321  16767 170402        RET             YES, RETURN.
```

MAGNETIC CARD BLOCK. —Continued

```
0322  16770 070577        STB A,I        NO, CLEAR LOCATION A.
0323  16771 073670        RIA CKIT       A ALWAYS NON ZERO. INCREMENT A.
0324  01730         TEMP  EQU 1730B      TEMP R/W.
0325  01730         DATA  EQU TEMP


0327**********************************
0328                        END
*  NO ERRORS*
```

MICROPROCESSOR

All of the above-listed routines and subroutines of basic instructions are implemented by the basic computing system shown in FIGS. 3A–B. Central control of this system is achieved by microprocessor **120.** As shown in the block diagram of FIG. **136** and in the detailed schematic diagram of FIGS. **137**A–D, the microprocessor comprises a bipolar ROM **200** including seven ROM chips organized into 256 words of 28 bits. Eight J-K flip-flops contain the ROM address; (i.e. a 4-bit primary address and a 4-bit secondary address). A single chip 16-bit data selector permits any one of 16 different qualifier lines to be tested with a 4-bit qualifier code. This 4-bit qualifier code ROM chip serves a dual function in that it provides a complementing code to the 4 primary address flip-flops as well as selecting the proper qualifier to be tested. If branching in any ROM state is desired, the microinstruction BRC must also be given. BRC occurring with a QN (qualifier not met) signal from the data selector will cause the least significant bit of the address code to be inhibited to the secondary address flip-flop, thus causing the address to "branch" according to the state of the qualifier.

An additional feature of this ROM organization is the IQN microinstruction (inhibit if qualifier not met). When the IQN is given and the qualifier selected by the qualifier code is not met, the signal CCO (clock code zero) goes low. This inhibits all shift clock pulses from the clock decoder which in effect prevents execution of microinstructions in that ROM state.

To minimize the ROM word length, two three-to-eight-line decoders are used to expand 3 R-code outputs and 3 X-code outputs into a total of 14 microinstructions. Also the SCO and SCI outputs from ROM No. **5** are decoded in the Memory. The ALU code outputs, **AC0, AC1,** and **AC2** are treated as address inputs to the ALU ROM and therefore need no decoding.

The microprocessor is responsible for the following:

1. Issuing a four-bit clock code to the clock decoder during each ROM state.
2. Issuing microinstructions to the memory, including the read and write microinstructions.
3. Issuing microinstructions to the shift registers for gating serial data into or out of the proper registers.
4. Issuing a four-bit ALU code to the Arith Logic Unit to select the proper binary or BCD arithmetic function.
5. Performing logical decisions (branching) based on the states of 16 qualifier inputs to the microprocessor.
6. Issuing next address information to the ROM address flip-flops in the microprocessor.
7. Transferring control to the input/output controller via the I/O strobe for execution of input or output instructions.

The full set of 28 ROM outputs with their associated microinstructions, the list of 16 qualifiers and assigned codes, and the microprocessor mnemonics are contained in the following tables:

MICRO-INSTRUCTION SET TABLE

	←POSITIVE TRUE→	←NEGATIVE TRUE→ OUTPUTS			
CONTROL FIELD	ROM OUTPUT	DECODED μ-INSTRUCTION OUTPUT			FUNCTION
GENERAL	1. IQN				INHIBIT SHIFT CLOCK IF QUALIFIER NOT MET.
	2. BRC				BRANCH: INHIBITS SOO IF QUALIFIER NOT MET.
	3. $\overline{TTT}$				T BUS → T REG
	4. $\overline{TTM}$				T BUS → M 5.
	5. $\overline{XTR}$				A/B REG→ R BUS
		Decoded In Memory			
			SC1	SC0	
S-CODE	6. SC1	ZTS	0	0	ZERO → S BUS
	7. SC0	MTS	0	1	M REG→ S BUS
		TTS	1	0	T REG → S BUS
		UTS	1	1	ONE → S BUS
		Decoded In Microprocesser			
			RC2	RC1	RC0
R-CODE	8. RC2	UTR	0	0	0	ONE → R BUS
	9. RC1	PTR	0	0	1	P REG → R BUS
	10. RC0	TRE	0	1	0	T REG → E REG → R BUS
		WTM·ZTR	0	1	1	STORE CONTENTS T REG → MEMORY
		TQ6·ZTR	1	0	0	T BUS → Q REG (BIT 6)
		QTR	1	0	1	Q REG → R BUS
		RDM·ZTR	1	1	0	READ MEMORY <M> → T REG
		ZTR	1	1	1	ZERO → R BUS
		Decoded In Microprocessor			
			XC2	XC1	XC0	

MICRO-INSTRUCTION SET TABLE – Continued

CONTROL FIELD	POSITIVE TRUE ROM OUTPUT	DECODED μ-INSTRUCTION OUTPUT	NEGATIVE TRUE OUTPUTS			FUNCTION
X-CODE	11. XC2	TTQ	0	0	0	T BUS → Q REG
	12. XC1	QAB	0	0	1	Q REG. (BIT 11) → AB FLIP-FLOP
	13. XC0	BCD	0	1	0	BCD ARITHMETIC MODE OF ALU
		TBE	0	1	1	T BUS → E REG → R BUS
		CAB	1	0	0	COMPLEMENT THE AB FLIP-FLOP
		TTP	1	0	1	T BUS → P REG
		TTX	1	1	0	T BUS → A/B REG
		NOP	1	1	1	NONE OF THE ABOVE
		Decoded In ALU	AC2	AC1	AC0	
ALU	14. AC2	XOR	0	0	0	EXCLUSIVE OR ..... R + →T BUS
	15. AC1	AND	0	0	1	LOGICAL AND ..... R · S → T BUS
	16. AC0	IOR	0	1	0	INCLUSIVE OR .... R + S → T BUS
		ZTT	0	1	1	ZERO → T BUS
		ZTT·CBC	1	0	0	ZERO → T BUS, CLEAR BINARY CARRY
		IOR·CBC	1	0	1	INCLUSIVE OR, CLEAR BINARY CARRY
		IOR·SBC	1	1	0	INCLUSIVE OR, SET BINARY CARRY
		ADD	1	1	1	BINARY ADD

CONTROL FIELD	ROM OUTPUT	FUNCTION
CLOCK	17. CC1 18. CC2 19. CC4 20. CC8	This 4-bit code initializes a presettable down counter to generate any number of shift clocks from 1 through 16. Shift is inhibited by IQN if QUALIFIER not met.
QUALIFIER	21. QC3 22. QC2 23. QC1 24. QC0	This 4-bit code performs two functions: (1.) Addressing the data selector to select one of sixteen QUALIFIER inputs, (2.) Provides complement code to PRIMARY Flip-Flops.
SECONDARY ADDRESS	25. SO3 26. SO2 27. SO1	This 4-bit code provides complement code to the SECONDARY Flip-Flops. If BRC is given and QUALIFIER is not
	28. SO0	met, the SO0 bit is inhibited.

SPECIAL MICRO-INSTRUCTIONS

TQR = UTR · XTR ..... TRANSFERS Q-REGISTER TO PRIMARY ADDRESS as shown

- Q14·Q11 → PO4
- Q12 → PO5
- Q13 → PO6
- Q14 → PO7

IOS = PTR · XTR .....INITIATES
a) TRANSFER OF CONTROL TO I/O IF Q10 = 1
b) SETS "SINGLE SERVICE" FF IN I/O VIA SRA IF Q10 = 0

SPECIAL OPERATIONS

BCD SUM → $A_{<0-3>}$ = BCD · UTR · ROM CLOCK
CLEAR DECIMAL CARRY = QAB · ROM CLOCK
SET DECIMAL CARRY = UTR · BCD · ROM CLOCK
DECIMAL ADD = BCD · ZTT ..... $T_{<0-3>} + A_{<0-3>} \rightarrow Q_{<0-3>}$
10's COMPLEMENT W/DECIMAL ADD = BCD · ADD .....
..... $\overline{T}_{<0-3>} + A_{<0-3>} \rightarrow Q_{<0-3>}$

QUALIFIER SET TABLE

QUALIFIER CODE QC3	QC2	QC1	QC0	MNE-MONIC	FUNCTION
0	0	0	0	Q00	SHIFT/SKIP ONE BIT
0	0	0	1	Q01	SHIFT/SKIP TWO BITS
0	0	1	0	Q02	SHIFT/SKIP FOUR BITS
0	0	1	1	Q03	SHIFT/SKIP EIGHT BITS
0	1	0	0	Q04	FAST SQUARE ROOT QUALIFIER
0	1	0	1	Q05	SET BIT IN A/S GROUP; FDV QUALIFIER
0	1	1	0	Q06	T-BUS QUALIFIER VIA TQ6
0	1	1	1	QBC	BINARY CARRY FROM ALU
1	0	0	0	QPO	P-REGISTER BIT 0, FOR BCD COUNTING
1	0	0	1	Q15	INDIRECT ADDRESS, CLEAR BIT IN A/S GROUP
1	0	1	0	QMR	MEMORY REFERENCE QUALIFIER
1	0	1	1	Q10	CURRENT PAGE QUALIFIER FXA QUALIFIER
1	1	0	0	QNR	NON-SERVICE REQUEST QUALIFIER
1	1	0	1	Q08	FMP QUALIFIER
1	1	1	0	QDC	DECIMAL CARRY FROM ALU
1	1	1	1	* QRD	ROM DISABLE (NORMALLY ZERO)

POP WILL PRESET ROM ADDRESS FLIP-FLOPS AT TURN-ON
* QRD MAY BE USED WITH IQN TO INSURE ZERO SHIFT EXCEPT WHEN IN I/O LOOP

MICROPROCESSOR MNEMONICS

Clock Signals	
MCK	Memory Clock
SCK	Shift Clock
XTC	External Clock
RCF	ROM Clock for Flip Flops
RCA	ROM Clock for Address Flip Flops
IIC	Inhibit Internal Clock
INH	Inhibit Clock
IPS	Inhibit ROM Clock (Also primary and secondary Flip Flop)
CC8, CC4, CC2, CC1	Clock Code: Binary Code that programs the number of shift clocks
CCϕ	Inhibits Shift Clocks
Address Mnemonics	
POP	Power on Preset
IQN	Inhibit if Qualifier not met
BRC	Branch
Q-Register	
TQR	Transfer Q11, Q12, Q13, Q14 to primary addr. Flip Flops
TTQ	T-Bus to Q-Register
QTR	Q-Register to R-Bus
Q10, Q9, Q8, Q7, Q6, Q5, Q4, Q3, Q2, Q1, Qϕ	Bits 1ϕ- ϕ of Q-Register
Data Qualifiers	
QPϕ	Bit ϕ of P-Register

MICROPROCESSOR MNEMONICS — Continued

Clock Signals	
QRD	Qualifier ROM Disable (I/O Interupt)
QNR	Qualifier No Request (keyboard Interupt)
QDC	Decimal Carry
QBC	Binary Carry
Memory	
SCϕ, SCI	S-Bus Code
TTT	T-Bus to T-Register
TTM	T-Bus to M-Register
RDM	Read Memory
WTM	Write Memory
A, B, P, E-Registers	
QAB	Q-Register to AB Flip Flop
AB = ϕ	A-Register Operation
AB = ϕ	B-Register Operation
TTX (ROM)	T-Bus to A or B Register (Originates at ROM Decoder)
XTR	A or B Register to R-Bus
TTP	T-Bus to P-Register
PTR	P-Registers to R-Bus
TBE	T-Bus to E-Register to R-Bus
TRE	T-Register to E-Register
UTR	Units to R-Bus
AC2, AC1, ACϕ	Arithmetic Codes for Arithmetic Logic Unit
BCD	Decimal Arithmetic
SDR	Disables ROMs For Single Step Tester Operation

Each of the ROM chips of FIGS. **136** and **137A–D** is organized into 256 words of 4 bits each constructed in accordance with the following table, where each L represents a low (or 0) state and each H represents a high or (1) state:

ROM CHIP 3

```
0012  2053,09,1,4,01024
0013  ROM3
0014  C2,11,71
0015  414
0016  000-007  LLLL LLHL HHHH LHHL HHHH LHHH HHLH HHLH
0017  008-015  LHLL LHLL HLHH HLLH HLHL LHLH LLLL LLLL
0018  016-023  LHHH LHHH LLLL LHHH LLLL LLLL LHLL HHLL
0019  024-031  LLHL LLHL LHLH HLHL LHLL HLHL LLLL LLLL
0020  032-039  LHLL LLLL HHLL LLLL HLLL HLLL LLHL HLLH
0021  040-047  LLHH LLHH LLLL LLLL LHLH HLLL HHLH LHHH
0022  048-055  HLLL HHLL LLLH LLLH LHHL LLLH HLHH HLHH
0023  056-063  LLLH LLLH HHLL LHLH LLLH HHLH HLHH LHLL
0024  064-071  LHHL LHHH HLHH HHHL LLLL HLLL LHHL HLHH
0025  072-079  LLLH LLLH LHLL LHLL HLLL LLHL LLLL HLLH
0026  080-087  LHHH LHLH HLLL LHHL HLHH HHLL LLLL LLLL
0027  088-095  LLHH LLLL HHHH LHHH LLLL LLHH LLLL LHLL
0028  096-103  HHHL HLLL LHHL LHHH LHHH LHHH LHHH HHHH
0029  104-111  LLHH LLHL LHLL LLLL LLLL LHLH HHLL HHLL
0030  112-119  LLLH LLLH LLLL LHLH LHLH LHLH HLLH LHHH
0031  120-127  LLLL LHLH HHHH LHHL LLLH LHLL LLLL LHLH
0032  128-135  HLHL LHLL HHLL LHLH LHHH HLHH LLLL LLLL
0033  136-143  LHLH LLHL LLHH HHLH LLHL HLHL HLLH LHLL
0034  144-151  LLLL LLHL HLLH HLLL HLLH LLLH HLLH HLLH
0035  152-159  HHHL LHHH HLHH HLLH LLLH LLLL LHHH HHLH
0036  160-167  HLLL HHHL LLLH HHHL LHHH LLHL LLHL HHHH
0037  168-175  LHHL HLLL LHLL HLHL LLLL LLLH LHLL HLLL
0038  176-183  LLLL LHHL LLHL HLLH LHHH LLHH HLHH HLHH
0039  184-191  LHHH HLLL LLHL LLHL LHLH LLLL LLHL LLLL
0040  192-199  LLLH LLLL LLLH LLLH HHLH HHLH LLHH HLHH
0041  200-207  HHHH LHLH HLLH LHLL HHLH LHLH LLLL LLLL
0042  208-215  HLHH HLLL LHHL LLLH HHHL LLHL HLHL HLHL
0043  216-223  HHHH HHLH LLLL LLLL LLLL LHLL LLLH HLLH
0044  224-231  HLLH HLLH LLLL LLLL HHLH HLLL LHHL HHHL
0045  232-239  LLLL HHHL HLLL HLLH HHLL HLHL LLLL LHHH
0046  240-247  LLHL LLHL LLHH LLLL LLLH HHLH LLHL LHLL
0047  248-255  LHLL LHLH HLLL LHHH LLLL LLLL LHLH LLLL
0048
```

SN23703

SN23703

ROM CHIP 4

```
0049  2054,09,1,4,01024
0050  ROM4
0051  02,11,71
0052  441
0053  000-007  HLHL HLLH LLLH HLLL LHHL HLLL HLHH LLHH
0054  008-015  LHHH HHLH HLLL HLLL HLLH LHHL HLLL LHHH
0055  016-023  HLLL LLHH LLHL LLLL HLHH HLLH LLLL LLLH
0056  024-031  LLLL LLLL HLHL LLHL LLLL LLLH LHLL LHHH
0057  032-039  LHLH HLLH HLLL HLHH LHLH HHLH HHLL HLLL
0058  040-047  LLLL LLLL HHLH HHLL HHLH HHLL HLLH HLHL
0059  048-055  HHHH LLHH LLLL HLLL LLLL HLHH HHLL LHLL
0060  056-063  LLLH LLLH HHHL HHHL LLLL HLLL HLHL LLLL
0061  064-071  LLLH HHLH HHLH HLLL HLHL LLLL HHHH HLHH
0062  072-079  LLLL LLLL HLHL LLHL LLHH HLLL LHLH LLHH
0063  080-087  HHHL HLHH HLHL HLLL LLLL HHLH LHLL LLLH
0064  088-095  HLLL LHLH HHHL HHLL LHHH LHLL HLLL LLLL
0065  096-103  HHHL HLHH HLLL HLHL HHHH HHLL HHHL LLHL
0066  104-111  LLLL LLLL LHLL LLHL HLLL LLLL HLHL LLHH
0067  112-119  LHHH LHLH HLLL HLLL LLHH HLHH HHHH LHLL
0068  120-127  LLHL HHHH HLLL HLHH LHLL HLLH HLLH HHHL
0069  128-135  LLLH HLHH LHHL HLHH HLHH HLLH HLHH LLHH
0070  136-143  HLLH LHLL HLHL LLHH LLLL HHHH LLLL HLLH
0071  144-151  HLHL HLHL HLLL HLLH HLHH LHLL HHLL HHLL
0072  152-159  HLLL HLLH HLHL HHHH LLLL HLLL HLHL LHLL
0073  160-167  LLLH HHHL HHHL LLHH HHHL HLHH HHHH HHHL
0074  168-175  HLLL HHLH LHLL HLLL HLLL LLLL LLHL LLLH
0075  176-183  HHLL HLLH HLLH HLLL HHHH HLLL HHLL LHLL
0076  184-191  HLLL LLHH LHLL HHLL LLHH HLLL LHLH LHHH
0077  192-199  LHHL HLLH LLLL HLLL HLHH HLLL HHHL HHHL
0078  200-207  LLHL HHHH HHHL HHHL LLHH LLLH HLLL LHHH
0079  208-215  LHHH HLLL HLLL LLLH HLHL LLLH LHLL LHHL
0080  216-223  LLHL HLHL HLHL LLHL HLLL LLLL LLLL HLHL
0081  224-231  HLLL HLLH LHHL HHLL HLLL LHLL LHLL LHHH
0082  232-239  HHLL LLHH LLHL LHLH LHHH LLLL HLLL LLHH
0083  240-247  HHLL LHLL LLHH HLLL HLLL HLHH LHLL LHLL
0084  248-255  LLHH HHLH LLLH LLLL HHLL HHLH LHLL HLLL
0085
```

SN23704

SN23704

ROM CHIP 5

```
0086  2055,09,1,01024
0087  ROM5
0088  02,11,71
0089  515
0090  000-007  HHHH LLLL LLLL HHHH LLLL LLLL LLLH LLLH
0091  008-015  HHHH LLLL HLLH LLLL HHHH HHHH LLLL LLHH
0092  016-023  LLLL LLLL LLLL LLHH LLLL HHHH HLHH LLLL
0093  024-031  LLHH LLHH LLLL LHLH LLHH LHLH HLHH HLHH
0094  032-039  HHHH LLLL LLLL LLLL HLHH HLHH HHHL LLLL
0095  040-047  LHHH LHHH LLLL LLLL LLHH HLHH LLLL LLLH
0096  048-055  LHLH HLHH LLLL HHHH LLLL LLLL LLLL LLLL
0097  056-063  LLLH LLLH LLHH LLHH LLLL HLHH HHHH HHHH
0098  064-071  HHHH HLHH LLHH LLLL LLLH LLLL LLLL LHLH
0099  072-079  HHHH LLLH HHHH LLLL HHHH HHHH LLLH HHHL
0100  080-087  LLLL LLLL HLHH LLLL HHHH HHHH HHHH HHHH
0101  088-095  LHHH LLLL LLLH LLLL LHLL LLHH LLHL HHLL
0102  096-103  HHHH HLHH HLHH HLHH LLHH HLHH LLLL LLLH
```

ROM CHIP 5—Continued

0103	104-111	LLLL	LLHH	HHHH	HHHH	HLHH	HHHH	HHHH	LLLL
0104	112-119	HHHH	HLHH	HHHL	LLLL	LLHL	LLLL	LLHH	LLLL
0105	120-127	HLHH	LLLL	LLHH	HLHH	LLHH	HHLH	LLLL	HHHH
0106	128-135	HHHH	LHLH	HHHH	LLLL	LLLH	LLHH	LLLL	LLLL
0107	136-143	HHHH	HHHH	LHLH	HLHH	HLHH	HLHH	LLLL	LLLH
0108	144-151	LLLL	LLHH	HLHH	LLHH	LLLL	HHHL	LLLL	HHHL
0109	152-159	LLLL	LLLL	LLLL	LHLL	HHHH	HLHH	LLHH	LHLL
0110	160-167	HHHH	HHHH	LHLH	LLHL	HHHH	LLLL	HHHH	LLLH
0111	168-175	HHHH	HHHH	HHHH	LLLL	HHHH	HHHH	HHHH	LHLL
0112	176-183	HHHH	LHLH	HHHH	HHHH	LLLL	LLHL	HHHL	LLLL
0113	184-191	HHHH	HLHH	LHLH	LHLH	HHHH	HLHH	LLHH	LLHH
0114	192-199	HHHH	HLHH	HHHH	HHHH	LHLH	HLHH	LHLH	LHLH
0115	200-207	LLLL	HHHL	HHHH	LLLH	HHHH	LLLL	HLHH	HHHH
0116	208-215	HHHH	LLHH	HLHH	HHHH	LLHH	HHHH	LLLL	HHHH
0117	216-223	LLLL	HLHH	LLLL	LLLL	HLLH	LHLH	LHLH	LLLL
0118	224-231	LLLL	LLLL	LLLL	HHHL	HLHH	HLHH	LLHL	LLLL
0119	232-239	HHHH	LLLL	HLHH	HHHH	LLLL	LLLL	HHHL	HHHL
0120	240-247	LLHH	LLLL	LHLL	LLLH	HHHH	HHHH	HLLH	LHLH
0121	248-255	HHHH	LLLL	HLHH	LLHH	HHHH	LLLL	HHHH	HHHH
0122									

ROM CHIP 6

0123	?056,09,2,4,01024								
0124	ROM6								
0125	02,11,71								
0126	595								
0127	000-007	LHHL	LHHH	HHHL	LLHH	HHHL	HLHH	HHHL	HHHL
0128	008-015	LLHL	LLHH	LLHH	LHLH	LHLH	LHLL	LHHH	LHLH
0129	016-023	LHHH	LHHH	LHHH	LHHL	LLLH	LHHL	LHHH	LHHH
0130	024-031	HHHH	HLHH	LLHH	LLLH	LHLH	LLLH	LHHH	LHHH
0131	032-039	LLLL	LHHH	LHHH	LHHH	LHLH	LHHH	LHHL	HHHH
0132	040-047	HHHH	HLHH	HHHH	HLHH	HLHH	LHLH	LHHH	HHHL
0133	048-055	LLLH	LHLH	LHHH	LHLL	LHHH	LLLH	LLHH	LLHH
0134	056-063	HHHH	HLHH	LHHH	LHHH	LHHH	LLHH	LHLH	LLLH
0135	064-071	LLHH	LHHL	LHHH	LHHH	LHHL	LHLH	HLHH	LLLH
0136	072-079	LHHH	HLHL	LHHL	LHHH	LHLH	LHHL	LLHL	LHHH
0137	080-087	LHHH	LHHH	LLHH	HHHL	LHHH	LHHL	LHLH	LHHH
0138	088-095	HLHL	HLHL	LLHH	LHHH	LHHH	LHHL	LLHH	LHHH
0139	096-103	LLHL	LHHH	LHHH	LHHL	LHLH	LHHL	LHHH	HHHL
0140	104-111	LLHH	HLHL	LHHH	LLHL	LLHH	LHHL	LHLH	LHHH
0141	112-119	LHLL	LHHH	LHHL	LHLH	LLLH	LHHH	HHLH	LHHH
0142	120-127	LHLH	HLHH	LHLL	LHHH	LHHH	LHHH	LHLH	LHHH
0143	128-135	LLHL	LHHH	LHHL	LHHH	LHLH	LHHH	LHLH	LHLH
0144	136-143	LHHL	LLHH	LLLH	LHLH	LHLH	LHHH	LHHH	LLLH
0145	144-151	LHHH	LHHH	LLLH	LHLL	LHHH	LHHH	LLHH	HHHL
0146	152-159	HLHL	HHHL	LHHH	LHHH	LHHH	LLHH	LHHH	LLHH
0147	160-167	LHLL	LLLL	LLLH	LLHH	LHHH	LLHH	LHLH	LLHH
0148	168-175	LLLL	LLHL	LHHH	LHHH	LHHH	LHLH	LHHH	LHHH
0149	176-183	LHHH	LHHH	LHLH	LHLH	LHLH	LLHH	LHHL	LHHH
0150	184-191	LLHL	LLHH	LHHH	LHHH	LHLH	LHHH	LHHH	LLHH
0151	192-199	LHLH	LLHH	LHLH	LHHH	LHLH	LHHH	LLLH	LLLH
0152	200-207	HLLH	HHHL	LHHL	LHHH	LHHL	LHHH	LHLH	LHHH
0153	208-215	LHLH	LHLL	LHLH	LHLH	HHHH	LHLH	LHHH	LHHH
0154	216-223	HLHH	LHHH	LHHH	LHHH	LLHH	LHHH	LHLH	LLHH
0155	224-231	HLHH	HHHL	LHHH	LHHH	LLHH	LHLH	LHHH	HLHH
0156	232-239	HLLL	LHHH	LLLH	LLHH	LLLH	LHHH	LLHH	LHHH
0157	240-247	LHLL	LHHH	LHLH	LLLH	LHLL	LHLH	LLHH	LLLH
0158	248-255	LLHL	LHHH	LHHH	LHLL	LHLH	LHLH	LHHH	LHHH
0159									

SN23706

SN23706

ROM CHIP 7

```
0160  2057,09,1,4,01024
0161  ROM7
0162  02,11,71
0163  596
0164  000-007  LHHH HHHH LLLL LLLH LLLL HHHH LHHH LHHH
0165  008-015  LLLH HLLH HHHH HHHH LLHL LHHH LHHH LHLH
0166  016-023  LHLL LHLL LHHH LHHH LHHH LHHH LLHH LHHH
0167  024-031  LHHH LLLL LHHH LHHH LLHL LHHH LHHH LLHH
0168  032-039  LHHH LHHH LHLL HHHH LLLL LHHH LHHH HHHH
0169  040-047  LHHH LLLL LHHH LLLL HHHH HHHL HHLH LHLL
0170  048-055  LHHH LHHL LHHH LHHH LHLH LHHH LHHH LLLL
0171  056-063  LHHH LLLL LHHH LLLL LHHH HHHL HLHL LLLL
0172  064-071  LHHH HHHH LHHH HHHH LHHH LHLL HLLL HHHH
0173  072-079  LLLH LHHH LHHH LHHH LHLL LHHH LHHH LHHH
0174  080-087  LHLL HHLL HHHL HHHH LHHH LHLL LLLL LLLL
0175  088-095  LHHH LHHH LHHH LHHH LHLL LHHH LHHH LHHH
0176  096-103  LHHH HHHL HLHH HHHH HHHH HHHH HHLL LHHH
0177  104-111  LLLH LHHH LLLL LHHH LLHH LHHH HLLL LHHH
0178  112-119  LHHH LHHL LHHH LHHH LHHH LHHH LHHH LHHH
0179  120-127  LHHH HLLL LHHH LLHH LLLL LHHH LHLL LHHH
0180  128-135  LHHH HLLL LHHH HHHH HHHH HHHH LLLL LHHH
0181  136-143  HHHH LHHH LHHH LLHH LHHL LHHL LHHH HLLL
0182  144-151  LLLL HHHH HLHH HHHH HHHH LHHH LLLL HHHH
0183  152-159  LLLH LHLL HHHH LHLH LHHH LLHH HHHH HLLL
0184  160-167  LHHH LHHH LHHH LLLL LLLL HLLH LHLL HHHH
0185  168-175  LLLH LHHH LHLL HHHH LLLL LHLL LHLL LHLL
0186  176-183  LLLL HHHH LLLL LLLL LLLH HLLH LHHH LHHH
0187  184-191  LLLH LHHL LHHH LHHH LHLH LHHL LHLH LLLL
0188  192-199  LLLL HHHL LLLL LHHH HHHH HHHH HHHH HHHH
0189  200-207  LLLH HHHH LHLH HLLL LHHH LHHH LLHH LLLL
0190  208-215  LHHH HHHH HLHH LHLL HHHH LLHL LHHH LHHH
0191  216-223  LLLH HLHH LLLH LLLH LLLL LHHH LHHH HHHH
0192  224-231  HLLL HHHH LHHH LHLL HHHL LLHH LLLL LHHH
0193  232-239  HHHH LHHH LLHH LLLL LHHH LHLH LHHH LHHH
0194  240-247  LHHH LHHH LLLL LLLL HHHH LLHL LLLH LHHH
0195  248-255  LHHH HHHH LLHH LHHH HHHH HLLH LHHH LHHH
0196
```

SN23707

SN23707

ROM CHIP 8

```
0197  2058,09,1,4,01024
0198  ROM8
0199  02,11,71
0200  733
0201  000-007  LHHH HHHL HHHL HHHL HHHL HLHL LLHL LLHH
0202  008-015  HHHL HHHL HHHL HHHH HHHL LHHL LHHL HHHH
0203  016-023  HHHL HHHL HHHL LHHH HHHH LHHH LHLL LHLL
0204  024-031  LHHL LHHL HHHH HHHH HHHL HHHH HHHL HHHL
0205  032-039  HHHL LHHL HHLH HHHL HHHH HHHL HHHL LHHL
0206  040-047  LHHL LHHL LHHL LHHL HLHL HHHH HHHL HHHL
0207  048-055  HHHH HHHH HHHL LHHL HHHL HHHH HHHH HHHH
0208  056-063  LHHL LHHL HLHH HHHH HHHL LHLH HLHL HHHL
0209  064-071  LHHL HHHL HLHH HHHL HHHH HHHL LHHL HHHH
0210  072-079  HHHL HHHL LHHH HHHL LLLH LHHH HHHH HHHH
0211  080-087  HHHL HHHL HHHH HHHL HHHH HHHH LHHL LHLH
0212  088-095  HHHL HHHL HHHL HHHL HHHL LHHH HHHH HHHH
```

ROM CHIP 8—Continued

0213	096-103	HHHL	LHLL	HHHL	HHHL	LHHL	HHHL	HHHL	LLHL
0214	104-111	HHHL	HHHL	HHHH	HHHL	LHLH	LHHH	LLLL	LHLL
0215	112-119	HHHH	HHHL	LHHH	HHHH	HLHH	HHHL	HLHL	HHHL
0216	120-127	LHHL	LHHL	HLHL	LHLL	HHHH	HHHL	HHHL	HHHH
0217	128-135	LHHL	HHHH	LHHH	HHHL	HHHL	HLHL	HHHH	HHHH
0218	136-143	HHHH	LHHL	HHHH	LHLH	HHHH	HHHL	HHHH	LLHL
0219	144-151	HHHH	HLHH	HHHH	HHHL	HLLL	HHHH	LHHL	HHHL
0220	152-159	HHHL	HHHL	HHHL	HHHL	HHHH	HHHH	HLHL	HHHL
0221	160-167	LHHL	HHLL	HHHH	HHHH	HHHH	HHHH	LHLL	HHHL
0222	168-175	HHHL	HHHL	HHLH	HLLL	HHHH	HHHH	HHLH	HHHL
0223	176-183	HHHH	LHLL	HHLL	HHLL	HHHH	HHHL	HHHL	HHHL
0224	184-191	HHHL	HHHH	HHHL	HHHL	HHLL	HHHL	HHHL	HLLL
0225	192-199	LLLL	HHHH	HHHL	HHHH	HLHL	HLLL	HLHH	HLHH
0226	200-207	HLHL	HHHL	HHHH	HHHH	HHHH	HHHL	HHHH	HHHH
0227	208-215	HHLH	HHHL	HHHH	HHHL	HLHH	HLHL	LHLL	LHLH
0228	216-223	HLHL	HHHL	HHHH	HHHH	HHHL	HHHL	HHHL	HHHH
0229	224-231	HHHL	LHHH	LLLL	HHLH	HHHH	LHLH	HHHH	HHHH
0230	232-239	LHHL	HHHL	HHHH	LLLL	HHHH	HHHL	LLLH	HHHH
0231	240-247	HLHL	HHHL	LHLH	LLHL	HHHL	HLHL	HHHL	HHHH
0232	248-255	HHHL	HHHL	LHLL	HLHL	HHHL	HHHL	LHHL	LHHL
0233										

ROM CHIP 9

0234	2059,09,1,4,01024									
0235	ROM9									
0236	02,11,71									
0237	586									
0238	000-007	HHLH	HHLL	HHLL	LHLH	HHLL	HHHH	HHLH	HHLH
0239	008-015	HHLL	HHHH	LHLH	HHLL	HLHH	HHLL	HHLL	LHLL
0240	016-023	HHLL	HHLL	HHLL	HHHH	HHLL	HHLL	HHLL	HHLL
0241	024-031	HHLL	HHHH	HHHL	LHLL	HLHH	LHLL	HHHL	HHLL
0242	032-039	HLLH	HHLL	LHHH	HHLL	LHHH	LHLL	HHLL	HHLL
0243	040-047	HHLL	HHHH	HHLL	HHHH	HHHH	HHLL	HHLL	HHLL
0244	048-055	LHLL	HHLL	HHLL	HHLH	HHLL	HHLL	LHLL	LHHH
0245	056-063	HHLL	HHHH	HHLL	HHLH	HHLL	HHLL	HHHH	HLLH
0246	064-071	HHLH	HHLH	HHLL	HHLL	HHLL	LHLL	HHHH	LHLL
0247	072-079	HLLL	HLHH	HHLL	HHLL	HHLH	LHLL	HHLL	LHHL
0248	080-087	HHLL	HHLL	HHLL	HHLL	LHHL	HHHH	HHLH	HHLH
0249	088-095	HLHH	HLHH	HHLL	HHLL	HHLL	HHHH	HHHL	LHHL
0250	096-103	HLLL	HHLL	HHLL	HHLL	HHLL	HHLL	HHLL	HHLH
0251	104-111	HHHH	HLHH	HHLH	LHLL	HHLL	LHLL	HHLH	HHLL
0252	112-119	HLLL	HHLL	HHLL	HHLL	HHLL	HHLL	HHLL	HHLL
0253	120-127	HHLH	HHHH	HHHH	HHLL	HHLH	HHHL	LHLL	HLLH
0254	128-135	HHLH	LHHH	HHLL	HHLL	LHLL	HHLL	LHHH	LHLL
0255	136-143	HHHH	HHHL	LHLL	HHLL	HHLL	HHLL	LHHL	HHLL
0256	144-151	HLLH	HHHH	HHLL	HLHH	HHLL	HLLH	HHHH	HHLL
0257	152-159	HHLL	HHLL	HHLL	HHLL	HLLH	HHLL	HHLL	LHHH
0258	160-167	HHLH	HHHL	LHLL	LHHH	LHHL	HHHH	LLLL	HHLL
0259	168-175	HHHH	LHLL	HHHH	HHLL	LHLH	HLLL	LHHL	HHLL
0260	176-183	HLLH	LHLL	HHLH	HHLH	LHHH	LHHH	HHLL	HHLL
0261	184-191	HHLL	HHLL	LHLL	HHHL	LHHH	HHLL	HHLL	LHHH
0262	192-199	HHHL	HHLL	LHLH	LHLL	LHLL	HHHL	LHLL	LHLL
0263	200-207	HHHH	HHLL	HLHL	LHHH	LHLL	LHLL	HHLL	HLLH
0264	208-215	HHHL	HLHH	HHLL	HLHH	HLLL	HHHH	HHHL	LHHL
0265	216-223	HHLL	HHLL	HHLL	HHHH	LHHH	LHLL	LHLL	HHLL
0266	224-231	LHHH	HHHH	LHLL	LHLL	HHLL	HHLL	LHHL	HHLL
0267	232-239	HHHH	HHLL	HHLL	LLHH	HHLL	HHLL	HHLL	LHLL
0268	240-247	HHHH	HHLL	LHHH	HHLL	LHHH	HHHH	LHHH	LHLL
0269	248-255	HLLL	HHLL	HHLL	HHHH	HLLL	HHLL	HHLL	HHLL
0270										

Each of the 71 basic instructions employed by the calculator is implemented by one or more of the above-described microinstructions and associated control signals issued by the microprocessor. The manner in which this is accomplished is shown and described in detail in the flow charts of FIGS. **138**A–H. Each rectangular box of these flow charts represents a state of ROM **200** of the microprocessor and includes the mnemonic of the microinstructions and control signals stored in that ROM state. The number at the upper right-hand corner of each of these rectangular boxes represents the number of shift clock pulses required by the microinstructions of that ROM state. A simplified overview of these detailed flow charts is shown in FIG. **7**.

PROGRAMABLE CLOCK

Given a computing system organized to process binary data serially and under control of microinstructions stored in ROM **200** as shown in FIGS. **3**A–B and **136**, the implementation of a general purpose instruction set requires that some number of bits be shifted into or out of the storage registers. Depending on the operation being performed, the number of bits may vary from zero to *n*, where *n* is the number of bits in a single machine word.

If each clock period of the ROM clock corresponds to a one bit shift, a count loop must be employed to provide the desired number of shifts. A rather large number of such count loops would exist in order to implement an entire instruction set. An alternative method is to provide additional hardware which permits assignment of the desired number of shifts in a single state of ROM **200**. Such an arrangement requires a variable cycle time for each state of ROM **200**, but results in a very substantive saving in total number of ROM states.

To implement a variable number of shift clocks in a single state of the microprocessor, two separate clocks are required. The shift clock is applied to the data storage registers in the memory, the shift register block, the arithmetic logic unit and the input/output block. The ROM clock is applied to the ROM address flip-flops in the microprocessor, and occurs once for each state in the microprogram. The number of shift clock pulses that occur in any given ROM state is determined by a 4-bit clock code sent to the clock decoder from the microprocessor.

If no shift clocks are desired, a separated signal CC0 from the microprocessor inhibits the shift clock output, independent of the clock code issued in that state. In this way, any number of shifts between and including zero and 16 may be implemented with a 4-bit clock code and an inhibit signal.

This inhibit signal offers an additional powerful feature when gated by the qualifier test logic in the microprocessor as shown in FIG. **136**. The qualifier test logic including a 4-bit qualifier code from ROM **3** that selects one of 16 qualifier inputs to the data selector. The data selector output QN (qualifier not met) will be high if the selected qualifier input was low. By using the QN signal to gate the inhibit microinstruction, IQN, the shift clock will be inhibited only when the qualifier is not met. Thus all microinstructions requiring shift clocks that are issued in a given ROM state may be either executed or inhibited, depending on the logical state of the qualifier under test.

The ROM clock is applied to the eight J-K flip-flops which address the 256 word microprocessor ROM. During any given state, the complementing (J-K) inputs to the four primary address flip-flops are set up by the qualifier code or *q*-register code. the four secondary address flip-flop inputs are determined by the ROM **4** outputs, the BRC microinstruction, and the data selector output QN. Where ROM clock goes low, the negative edge-triggered flip-flops will cause transition of the ROM address to the next ROM state.

As shown in the block diagram of FIG. **139** and the detailed schematic diagram of FIGS. **140**A–C, a crystal controlled system clock output is inverted to generate memory clock, MCK. This signal is again inverted to clock a D flip-flop having an output (control clock), which will go low if the end-of-count signal (borrow) from the down counter has occurred at the D input. The ROM clock will also go low at this time, initiating a new ROM state in the microprocessor. Control clock will normally remain low for one system clock period, and in turn generates a load signal which is delayed a half period from control clock by means of a second D flip-flop. The 4-bit clock code from the microprocessor is preset into the counter while the load signal is low.

As the load signal goes high, ROM clock also goes high, completing the fixed interval portion of ROM clock and shift clock as shown in FIG. **141**. A series of clock pulses are now gated onto shift clock, SCK, until the preset counter has counted down to zero, causing control clock to again go low, completing the ROM cycle.

The inhibit signal, $\overline{\text{INH}}$, from memory may lengthen the normal fixed interval of ROM clock by clearing the D flip-flop and holding control clock low. This may occur during memory refresh or external test operations. In this situation, the counter remains preset and the correct number of shifts will be generated when the inhibit goes away.

SHIFT REGISTER UNIT

As shown in the detailed schematic diagrams of FIGS. **137**A–D and **142**A–D, A-register **122**, B-register **124**, P-register **126**, Q-register **128**, and E-register **130** comprise bipolar status registers, the contents of which are recirculated when data is output to the R-bus or the S-bus. Full control of these registers in use and type of operations performed is maintained by the microinstructions from the from the microprocessor. The number of bits to be shifted in any one ROM state of the microprocessor is determined by the number of shift clocks from the clock decoder. This shift clock appears at the shift clock input of each shift register that is enabled by the microprocessor during that ROM cycle.

ARITHMETIC LOGIC UNIT

The development of complex and read-only memory arrays on a single chip have made possible a hardware implementation of central processing units (CPUs) and arithmetic logic units (ALUs) with far fewer components than were previously possible. In this application, two bipolar read-only memory chips are combined with carry flip-flops and adapted to perform 1-bit binary logic and arithmetic operations as well as 4-bit binary coded decimal (BCD) arithmetic operations. The two bipolar read-only memory chips may comprise, for example. Hewlett-Packard 16-pin dual-in-line packaged bipolar ROMs organized into 256 words by 4-bits and of the same type as shown and described in U.S. Pat. App. Ser. No. 12,262 filed Feb. 18, 1970 by John C. Barrett et al. and assigned to the same assignee as this patent application.

The binary/BCD Arithmetic Logic Unit consists of five integrated circuits connected as shown in the block diagram of FIG. **143** and the detailed schematic diagram of FIGS. **142A–D**. Specifically, the packages consist of two 1024-bit ROMs, a dual D-type flip-flop and two quad two-input NAND gates. The function code input BCD selects between the binary mode and BCD mode of operation.

In the binary mode, the function code inputs **AC0, AC1,** and **AC2** select the desired logical function or arithmetic operation as given in FIG. **144.** The binary input data enters ROM No. **1** on the carry, S-bus and R-bus input lines, and the binary result appears on the T-bus and binary carry output lines. ROM No. **2** is not used in the binary mode.

In the BCD mode of operation, the two function code lines **AC0** and **AC1** are disabled from the Microprocessor and these two lines carry the **TO2** and **TO3** bits of BCD data from the T-Register. The ALU function code line **AC2** is used to select the desired BCD operation. If **AC2** is low, the 4-bit output $\Sigma_0, \Sigma_1, \Sigma_1, \Sigma_3$ will be the BCD sum of the two BCD data inputs. If **AC2** is high and decimal carry has been set, the 4-bit output $\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3$ will be the BCD Tens Complement of the BCD data from the T-Register. In the BCD mode, the binary carry output will be disabled and the decimal carry output will be enabled to ROM No. **1.**

Although only one-fourth of the available registers in ROM No. **1** are required for the eight binary operations, the concept of adding a second 1024-bit ROM to perform the BCD operations grew from several basic concepts:

1. The least significant BCD sum bit, Σ_0, is always identical to the binary sum bit; therefore, only three additional outputs, Σ_1, Σ_2, and Σ_3 need be generated. For BCD complement operations, the decimal carry flip-flop defines whether or not the least significant bit should be complemented.
2. In forming the "nine's complement" of the T-Register BCD data in ROM No. **1,** it can be seen that for 8421 code the second least significant bit **TO1** is the same before and after forming the complement. Thus only two bits, **TO2** and **TO3** need be complemented prior to input into ROM No. **2.** The ten's complement with add is then found by presetting decimal carry and performing a BCD sum of the three most significant digits in ROM No. **2.**
3. With only eight ROM inputs available, some sharing of inputs is required for ROM No. **1.** During binary operations, all four function codes and only one bit of T-Register data is required. During BCD operations, all four bits of T-Register data and only two function codes are required. Use of two NAND gates in wire-OR connection with the open collector function codes **AC0** and **AC1** permits sharing of the two inputs.

This arrangement left one input still available to ROM No. **2.** By programming this input to always make output DC1 true, the micro-instruction $\overline{\text{UTR}}$ can serve two purposes—placing units on the R-bus and also set decimal carry if BCD is true. When BCD is false, clock is inhibited to decimal carry. This feature permits saving decimal carry information during all binary operations. Similarly, binary carry is saved during the four binary operations AND, IOR, XOR, and ZTT by connecting **AC2** such that when **AC2** is false the shift clock is inhibited to the binary carry flip-flop.

In summary, the mode select input BCD performs the following functions:

1. Addresses the proper 128 word set of word lines in ROM No. **1.**
2. Enables the **TO2** and **TO3** data lines to ROM No. **1** only in BCD mode.
3. Enables clock to decimal carry flip-flop only in BCD mode.
4. Selects binary carry or decimal carry into ROM No. **1** as appropriate.
5. Transfers outputs $\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3$ to A-Register only in BCD mode.

The remaining three ALU function codes select the proper set of word lines in ROM No. **1** to perform the eight binary functions listed in FIG. **144.** In addition, the **AC2** input performs the following functions.

1. Enables clock to binary carry flip-flop only during the four carry-related binary functions and the BCD comp/add function.
2. In the BCD mode, **AC2** causes BCD data bit **TO0, TO2** and **TO3** to convert the nine's complement form.

The ALU has a total of 15 inputs which include eight data inputs, two clock inputs and five microinstructions. Four data output lines are required, and two additional output lines from carry flip-flops are available as qualifier inputs to the microprocessor. The ALU and shift register mnemonics are listed in the following table:

SHIFT REGISTERS & ALU BOARD MNEMONICS

$\overline{\text{TRE}}$	T-Register to E-Register to $\overline{\text{R}}$-Bus
TOϕ	Bit ϕ of T-Register
$\overline{\text{TBE}}$	T-Bus to E-Register to $\overline{\text{R}}$-Bus
$\overline{\text{TTX}}$ - TEST	T-Bus to A/B-Register from Tester
$\overline{\text{TTX}}$ - I/O	T-Bus to A/B-Register from I/O (Board No. 12)
$\overline{\text{TTX}}$ - ROM	T-Bus to A/B-Register from μ-Processor (Board No. 13)
$\overline{\text{TTX}}$	Logical "OR" of Three $\overline{\text{TTX}}$ Signals
AB	Status of AB-Flip-Flop AB = O A-Reg. Operation AB = 1 B-Reg. Operation
$\overline{\text{XTR}}$	A/B Register to $\overline{\text{R}}$-Bus
$\overline{\text{UTR}}$	Logical "1" to $\overline{\text{R}}$-Bus
$\overline{\text{TQR}}$	Q-Register to Primary Address Flip-Flops
$\overline{\text{AB}}$	Complement of AB
$\overline{\text{TTP}}$	T-Bus to P-Register
SCK	Shift Clock
$\overline{\text{QP}\phi}$	Qualifier, Bit ϕ of P-Register
$\overline{\text{PTR}}$	P-Register to $\overline{\text{R}}$-Bus
$\overline{\text{QO}\phi}$	Q-Register Bit ϕ
$\overline{\text{QTR}}$	Q-Register to $\overline{\text{R}}$-Bus
RCK	ROM Clock
QAB	Q-Register to AB-Flip-Flop, also clears decimal carry.
SCB	Set Binary Carry
$\overline{\text{QDC}}$	Qualifier, Decimal Carry
$\overline{\text{BCD}}$	Decimal Arithmetic
AC2	ALU Operation Code
QBC	Qualifier, Binary Carry
$\overline{\text{S}}$-BUS	Data Bus
AC1	ALU Operation Code
ACϕ	ALU Operation Code
TO2	Bit 2 of T-Register
TO3	Bit 3 of T-Register
SDR	Signal to Disable ROMs
TO1	Bit 1 of T-Register
T-BUS	Data Bus
ALU	Arithmetic Logic Unit

(—) Indicates Negative True Signal

FIG. **145** lists five dual-in-line integrated circuits that may be employed in the ALU. The following table gives an example of how the two ALU ROM chips shown in FIGS. **142A–D** and **143** can be constructed to implement the above described ALU functions (in this table each 1 represents a low state and each 0 represents a high state):

ROM #1

I3391-0006/

/	1000;	1/	0000;	2/	0000;	3/	1000
4/	1000;	5/	0000;	6/	0000;	7/	1000
8/	0000;	9/	0000;	10/	0000;	11/	1000
12/	0000;	13/	0000;	14/	0000;	15/	1000
16/	0000;	17/	1000;	18/	1000;	19/	1000
20/	0000;	21/	1000;	22/	1000;	23/	1000
24/	1000;	25/	1000;	26/	1000;	27/	1000
28/	1000;	29/	1000;	30/	1000;	31/	1000
32/	1000;	33/	0000;	34/	0000;	35/	1000
36/	1000;	37/	0000;	38/	0000;	39/	1000
40/	0000;	41/	0000;	42/	0000;	43/	1000
44/	0000;	45/	0000;	46/	0000;	47/	1000
48/	0000;	49/	1000;	50/	1000;	51/	1000
52/	0000;	53/	1000;	54/	1000;	55/	1000
56/	1000;	57/	1000;	58/	1000;	59/	1000
60/	1000;	61/	1000;	62/	1000;	63/	1000
64/	1100;	65/	1100;	66/	1100;	67/	1100
68/	1100;	69/	1100;	70/	1100;	71/	1100
72/	0000;	73/	0000;	74/	0000;	75/	1000
76/	0000;	77/	0000;	78/	0000;	79/	1000
80/	0100;	81/	0100;	82/	0100;	83/	1100
84/	0100;	85/	0100;	86/	0100;	87/	1100
88/	0000;	89/	1000;	90/	1000;	91/	0100
92/	1000;	93/	0100;	94/	0100;	95/	1100
96/	1100;	97/	1100;	98/	1100;	99/	1100
100/	1100;	101/	1100;	102/	1100;	103/	1100
104/	0000;	105/	0000;	106/	0000;	107/	1000
108/	0000;	109/	0000;	110/	0000;	111/	1000
112/	0100;	113/	0100;	114/	0100;	115/	1100
116/	0100;	117/	0100;	118/	0100;	119/	1100
120/	0000;	121/	1000;	122/	1000;	123/	0100
124/	1000;	125/	0100;	126/	0100;	127/	1100
128/	0000;	129/	0000;	130/	0000;	131/	0000
132/	0000;	133/	0000;	134/	0000;	135/	0000
136/	0010;	137/	1010;	138/	1010;	139/	0110
140/	1010;	141/	0110;	142/	0110;	143/	1110
144/	0001;	145/	1001;	146/	1001;	147/	0101
148/	1001;	149/	0101;	150/	0101;	151/	1101
152/	0011;	153/	1011;	154/	1011;	155/	0111
156/	1011;	157/	0111;	158/	0111;	159/	1111
160/	0000;	161/	0000;	162/	0000;	163/	0000
164/	0000;	165/	0000;	166/	0000;	167/	0000
168/	0010;	169/	1010;	170/	1010;	171/	0110
172/	1010;	173/	0110;	174/	0110;	175/	1110
176/	0000;	177/	0000;	178/	0000;	179/	0000
180/	0000;	181/	0000;	182/	0000;	183/	0000
184/	0011;	185/	1011;	186/	1011;	187/	0111
188/	1011;	189/	0111;	190/	0111;	191/	1111
192/	0000;	193/	0000;	194/	0000;	195/	0000
196/	0000;	197/	0000;	198/	0000;	199/	0000
200/	1010;	201/	0010;	202/	0110;	203/	1010
204/	0110;	205/	1010;	206/	1110;	207/	0110
208/	1011;	209/	0010;	210/	0111;	211/	1011
212/	0111;	213/	1011;	214/	1111;	215/	0111
216/	1001;	217/	0001;	218/	0101;	219/	1001
220/	1010;	221/	1001;	222/	1101;	223/	0101
224/	0000;	225/	0000;	226/	0000;	227/	0000
228/	0000;	229/	0000;	230/	0000;	231/	0000
232/	1011;	233/	0011;	234/	0111;	235/	1011
236/	0111;	237/	1011;	238/	1111;	239/	0111
240/	0000;	241/	0000;	242/	0000;	243/	0000
244/	0000;	245/	0000;	246/	0000;	247/	0000
248/	1010;	249/	0010;	250/	0110;	251/	1010
252/	0110;	253/	1010;	254/	1110;	255/	0110

ROM #2

I3301-0007/

/	0000;	1/	0000;	2/	0000;	3/	0000
4/	0000;	5/	0000;	6/	0000;	7/	0000
8/	0000;	9/	0000;	10/	0000;	11/	0000
12/	0000;	13/	0000;	14/	0000;	15/	0000
16/	0000;	17/	0000;	18/	0000;	19/	0000
20/	0000;	21/	0000;	22/	0000;	23/	0000
24/	0000;	25/	0000;	26/	0000;	27/	0000
28/	0000;	29/	0000;	30/	0000;	31/	0000
32/	0000;	33/	0000;	34/	0000;	35/	0000
36/	0000;	37/	0000;	38/	0000;	39/	0000
40/	0000;	41/	0000;	42/	0000;	43/	0000
44/	0000;	45/	0000;	46/	0000;	47/	0000
48/	0000;	49/	0000;	50/	0000;	51/	0000
52/	0000;	53/	0000;	54/	0000;	55/	0000
56/	0000;	57/	0000;	58/	0000;	59/	0000
60/	0000;	61/	0000;	62/	0000;	63/	0000
64/	0000;	65/	0000;	66/	0000;	67/	0000
68/	0000;	69/	0000;	70/	0000;	71/	0000
72/	0000;	73/	0000;	74/	0000;	75/	0000
76/	0000;	77/	0000;	78/	0000;	79/	0000
80/	0000;	81/	0000;	82/	0000;	83/	0000
84/	0000;	85/	0000;	86/	0000;	87/	0000
88/	0000;	89/	0000;	90/	0000;	91/	0000
92/	0000;	93/	0000;	94/	0000;	95/	0000
96/	0000;	97/	0000;	98/	0000;	99/	0000
100/	0000;	101/	0000;	102/	0000;	103/	0000
104/	0000;	105/	0000;	106/	0000;	107/	0000
108/	0000;	109/	0000;	110/	0000;	111/	0000
112/	0000;	113/	0000;	114/	0000;	115/	0000
116/	0000;	117/	0000;	118/	0000;	119/	0000
120/	0000;	121/	0000;	122/	0000;	123/	0000
124/	0000;	125/	0000;	126/	0000;	127/	0000
128/	1111;	129/	1101;	130/	1011;	131/	1001
132/	0111;	133/	0000;	134/	0000;	135/	0000
136/	1101;	137/	1011;	138/	1001;	139/	0111
140/	1110;	141/	0000;	142/	0000;	143/	0000
144/	1011;	145/	1001;	146/	0111;	147/	1110
148/	1100;	149/	0000;	150/	0000;	151/	0000
152/	1001;	153/	0111;	154/	1110;	155/	1100
156/	1010;	157/	0000;	158/	0000;	159/	0000
160/	0111;	161/	1110;	162/	1100;	163/	1010
164/	1000;	165/	0000;	166/	0000;	167/	0000
168/	0000;	169/	0000;	170/	0000;	171/	0000
172/	0000;	173/	0000;	174/	0000;	175/	0000
176/	0000;	177/	0000;	178/	0000;	179/	0000
180/	0000;	181/	0000;	182/	0000;	183/	0000
184/	0000;	185/	0000;	186/	0000;	187/	0000
188/	0000;	189/	0000;	190/	0000;	191/	0000
192/	1101;	193/	1011;	194/	1001;	195/	0111
196/	1110;	197/	0000;	198/	0000;	199/	0000
200/	1011;	201/	1001;	202/	0111;	203/	1110
204/	1100;	205/	0000;	206/	0000;	207/	0000
208/	1001;	209/	0111;	210/	1110;	211/	1100
212/	1010;	213/	0000;	214/	0000;	215/	0000
216/	0111;	217/	1110;	218/	1100;	219/	1010
220/	1000;	221/	0000;	222/	0000;	223/	0000
224/	1110;	225/	1100;	226/	1010;	227/	1000
228/	0110;	229/	0000;	230/	0000;	231/	0000
232/	0000;	233/	0000;	234/	0000;	235/	0000
236/	0000;	237/	0000;	238/	0000;	239/	0000
240/	0000;	241/	0000;	242/	0000;	243/	0000
244/	0000;	245/	0000;	246/	0000;	247/	0000
248/	0000;	249/	0000;	250/	0000;	251/	0000
252/	0000;	253/	0000;	254/	0000;	255/	0000

MEMORY UNIT

The calculator uses an all semiconductor memory system. Peripheral circuitry is bipolar and the memory consists of *n*-channel MOS read only memory (ROM) and *p*-channel MOS read/write memory (RWM).

Addressing and physical layout of the memory module is done so that the number of words can be increased from 3K in the basic machine to 7.5K in the largest machine. The smallest increment of memory that can be added in 512 words. Provisions have been made to contain all add-on RWM within the memory module. Add-on ROM is external to the memory module, behind the display.

The basic machine contains 3K words of memory, organized into 2K × 16 ROM, 512 × 16 and 512 × 6 RWM. The 16 bit RWM words are divided into 109 user registers (four words) and 76 words for processor use. The 6 bit RWM words are program steps. The largest machine contains 5K words of ROM and 2.5K words of RWM, of which 512 words are 16 bit.

Read/Write Memory

As shown in FIGS. **146–150** memory is made up of 1024 × 1, dynamic, Read/write memory chips (Intel **1103**). These devices are P-channel, MOS using silicon gate technology. To maintain the contents of memory, the device must be refreshed every 2 ms. This is accomplished by performing a read cycle at a given address. On each chip are 32 refresh amplifiers so that each read cycle, 32 cells get refreshed. The entire chip is then refreshed by cycling through the lower 5 address bits and reading each distinct address. The refresh period is 20 μs at least every 2 ms.

Logic levels on all input lines to the RWM chips are 0 to + 16v. This includes the three clock lines (chip select, Y-enable or write, and precharge), 10 address lines, and input data. The output data, however, is a current of 600 μa or more into 1K ohms or less. This low level output is "wire-or able" with other chips to build larger systems.

Read Only Memory

As shown in FIG. **146, 147, 151,** and **152** ROM chips are 4096 bit, *n*-channel MOS arranged 512 × 8. The devices are static and consume no power when not enabled. Data is retrieved from the ROMs by pulling the chip enable line from 0 to + 12v (turning the chip on), addressing the desired cells (0 or 4v levels) and selecting which output devices are to be enabled (4v or 0v). The output levels are sufficient to drive one TTL gate directly, and can be "wire-or/ed" for large systems.

As further shown in FIGS. **153** and **154** A–D, each ROM chip comprises six input buffers. These input buffers generate both the input and its complement. On the basis of the 64 possible combinations of the 6-inputs I_0–I_5, one of the 64 lines in the decoder is selected. The selected line enables one of the vertical lines in the 64 × 64 bit storage array. For example, let $I_0 - I_5 = 0$ and $I_6 - I_8$ be "don't cares". This means line **X00** (octal) is selected.

The two 8 out of 32 select decoders must choose 16 lines from the 64 horizontal lines selected by the vertical line **X00-** (The 8 out of 32 select decoder is actually a 2 out of 8 decoder repeated 4 times in each of the section A – B.) The output from four MOS FET's *a*, *b*, *c*, and *d* are wire or/ed. MOS devices *a'*, *b'*, *c'*, and *d'* are also connected similarly. If I_6 and $I_7 = 0$, horizontal lines **1XX, 2XX, 3XX, 5XX, 6XX, 7XX** are grounded in each of the four sections A–B. This insures that MOS FET's *b*, *c*, *d*, *b'*, *c'*, and *d'* are non-conductive. This allows signals on lines **0XX** and **4XX** to pass into the output sections through transistors *a* and *a'*.

The output section contains the output buffer, 1 out of 2 decoder, and the output drivers *s*. The output butter provides a stage of gain and "wire or's" four lines from the storage array. The 1 our of 2 decoder clamps the gates of 2 of the 4 output drivers in each section A–B by enabling either line I_8 or its complement ($\overline{I_8}$). This disables 1 of 2 signals coming from the output buffer. The output drivers then can be tied together with line (*e*) for a 512 × 8 organization.

Each of the above-listed constants and routines and subroutines of basic instructions employed by the calculator is stored in these ROM chips. This is accomplished by partitioning the 0 to 16,777 octal addresses of the memory map of FIG. **4** into $512_{10} \times 8_{10}$ blocks which is the capacity of each 4096-bit ROM chip. These blocks are represented in the memory map of FIG. **155.** The next step is the inversion of the 8 bits (if the input was 1, it is now a 0) and complimenting of the address (i.e. once the octal addresses **0 – 16,777** are partitioned into 512 × 8 blocks, only the last 3 octal digits are important). For this reason, address 777_8 goes to $000{,}776_8$ goes to **001, ...,** and **000** goes to 777_8.

The 16 bits of each constant and basic instruction are stored in the $512_{10} \times 8_{10}$ ROM chips by organizing the ROM chips into 64 × 64 bit matrices and computing the row and column numbers of each bit of each matrix by operating on each address and the particular bit (**15** through **8,** or **7** through **0**). The column number is computed by subtracting the last two digits of the address from 100_8. For example, the column number of address **000** $= 100_8 - 00_8 = 100 = 64_{10}$ and the column number of address **777** $= 100_8 - 77_8 = 1$. The computation of the row number (referred to as IR in the flowchart of FIG. **156**) can best be described by referring to the flowchart of FIG. **156** and the associated table of FIG. **157.** Once the row and column numbers are found it is a simple matter of storing in that location of the matrix that particular bit (i.e., a 1 or a 0). A 0 is stored at a designated location by forming a metal gate to complete a MOS FET device at that location, and a 1 is stored at a designated location by leaving off the metal gate so that a MOS FET device is not formed at that location.

M-Register

As shown in FIGS. **146, 147,** and **158** A–D included on the M-Register board is the 16 bit Address or M-Register, all chip enable decoding and buffering, and address buffers for both ROM and RWM. The register uses four, four bit, serial in and out, parallel in and out shift Registers. Upon receipt of a $\overline{TTT}$ instruction from the microprocessor, serial data from the T-Bus is accepted into the M-register. Nothing is done with this data until either a read or write instruction is received, then one of two decoders are enabled. These chip Enable decoders uniquely decode which block of 512 words, either ROM or RMW, is being addressed. If ROM is being addressed, the signal is inverted and amplified to +12v. For RWM the Chip Enable enables a gate, which allows a 16 Volt clock signal to reach the enabled RWM chips. The clock wave-form is generated on the control card.

The dynamic characteristic of the RWM chips, requires that all chips be enabled simultaneously during a refresh cycle, to refresh the entire read/write memory. The buffer circuits in the output of the Chip Enable decoders allow the chip select clock to reach all of the RWM chips during refresh but only those being accessed, during a read or write cycle.

Open collector nand gates with resistor pull-ups are used as buffers for the ROM address lines. Using the open collector gates, the address lines can be pulled above the required 4v level. The nand gates are enabled during a memory cycle so that the ROM address lines are inhibited at a 5v level. The RWM address lines must pull from 0v to + 16v. High voltage, open collector, inverters with discrete transistor pull-ups are used as buffers for the 5 most significant bits. The 5 least significant address bits are bussed to the control card where they are used in part of the refresh circuitry.

Control

A memory cycle consists of a read or write instruction from the processor accompanied by 12 clock pulses from the shift clock. As shown in FIGS. **146, 147, 159** A–D, and **160** control uses these pulses and instructions to generate the clocks required by the RWM chips. A synchronous 4 bit counter (SN74193) is used to count clock pulses and the 4 outputs are decoded by a 1 of 16 decoder (SN74154) to generate J and K input to flip-flops. The outputs from the flip-flops are then buffered to become the required clock signals (Precharge, Y-enable, chip select).

Refreshing the read/write memory is also taken care of by the control card. An astable multivibrator with a repetition rate of 500 hz. minimum generates a signal which allows a refresh cycle to occur. A flip-flop generates the actual signal (REF), but only if the astable multivibrator signal is high, there is no read or write cycle in progress and the processor signal, $\overline{CCT}$, is high. $\overline{CCT}$ goes high between processor instructions, thus it is known that nothing is going to be interrupted when REF is generated. REF is then buffered by an open collector inverter and given to the processor as $\overline{INH}$. $\overline{INH}$ halts the machine and the refresh cycle begins.

The same counter used for a memory cycle, is used during refresh to again generate the necessary clocks (Precharge and chip select). When the counter returns to state 0 and REF is present, a second counter is advanced one count. This second counter provides the refresh addresses which go to the RWM only if REF is present. When this counter returns to state 0, it causes REF and $\overline{INH}$ to return to preset conditions and the machine continues normal operation.

A further function of the control card is to make the 1024X1 memory chips appear to the processor like 512 × 2 chips. This is done by accessing the RWM twice during each memory cycle. Hence, half-way through each refresh or memory cycle a flip-flop changes state to generate one address bit. The M-register thus provides only 9 address bits and the control card 1 bit, independent of the M-register.

Other signals generated on the control card, direct the flow of data in the T-register.

T-Register

Data to and from the memory is temporarily stored in the T-register. As shown in FIGS. **146, 147,** and **161** A–D four 4-bit, serial in and out, parallel in and out shift registers make up the actual T-register. The registers have a mode control (TMC) which when low, allows serial data flow and when high, allows parallel data flow.

Serial data enters the T-register in the presence of a $\overline{TTT}$ instruction. Data is serially transmitted to the $\overline{S}$-bus during a TTS instruction, and simultaneously recirculated into the T-register to prevent loss of data.

Parallel data is accepted from either ROM or RWM during a read cycle. The ROM data is buffered by nand gates and the RWM by sense amplifiers followed by the same nand gates. Multiplexing the RWM means that only one-half the data bits are transmitted between RWM and the R-register during each half of the memory cycle. The first half of the read cycle, the odd bits are loaded into the T-register. To complete the transfer these bits must be shifted right one place and the odd bits again loaded from the RWM. This is done by allowing the odd bits to appear as data at the input of the even bits. When new data is clocked into the T-register during the second half of the read cycle, the even inputs are also clocked in, filling the T-register. This isn't done with ROM since all ROM is transferred 16 bits parallel.

Data is written into memory in a manner similar to the way data is read from RWM. The bits are 16v levels, after buffering, and are written by the odd bits followed by the even bits. The nand gates between the T-register and the 16v buffers are gates from the control card to handle the write multiplexing.

INPUT-OUTPUT CONTROL UNIT

The input-output control unit allows the calculator to communicate with the internal input, input-output, and output units and with external peripheral devices. As shown in FIGS. **140** A–C and **162** A–D, the input-output control unit is contained on two printed circuit boards; the "Control and System Clock" board and the "I/O Register and Gate Interface" board. A third board, shown in FIG. **163,** is an I/O motherboard providing room for connecting four external interface cards to the calculator.

The internal input, input-output, and ouput units are distinguished from peripheral devices by the fact that the I/O language set addresses them directly. Hence each I/O instruction contains an internal peripheral address as part of its makeup. The five internal directly-addressable input, input-output, and output units are the I/O register, the magnetic card reader, the output printer, the *x*-, *y*-, and *z*- register display, and the keyboard mode lights.

The external peripheral devices are indirectly addressable and are connected via cable to an interface card which is plugged into the I/O motherboard at the rear of the calculator. The term indirectly addressable is defined here to mean the external peripheral devices are addressed by lines leading from the four most significant bits in the I/O register, thereby requiring an address word to be loaded into the directly addressable I/O register.

I/O CONTROL AND SYSTEM CLOCK SECTION

The function of the I/O Control and System Clock Section is to provide control to the I/O Register and Gate Interface Section. This is accomplished by use of an I/O instruction set stored in the main memory of the calculator.

The microprocessor causes instructions from the memory unit to be loaded into the T-Register and then to be transferred to the Q-Register. The microprocessor determines the type of instruction and causes the proper execution of the instruction. If the instruction is an I/O type, control is transferred by the microprocessor to the I/O Control and System Clock Section.

The microprocessor remains in a two state waiting loop while the I/O Control section is active. Time in the wait loop is between 0.72 μ seconds and 6.5 μ seconds.

Bits **5** through **10** from the Q-Register are connected to the I/O Control section and remain constant during an I/O instruction execution time. Bits **5** through **8** representing the I/O instruction code are gated to the I/O

address flip flops and entered on each clock time while the I/O is inactive. The four outputs of the address flip flops are connected to the address input of a 1 of 16 decoder and represent the starting state address of the I/O instruction to be executed. When the I/O Control Section is enabled, the input gates passing bits **5** through **8** to the I/O address flip flops are closed and the 1 of 16 decoder enabled. This allows the starting state I/O micro instructions to come from the 1 of 16 decoder. The next state address coming from the closed input gates will be the exit state ($1111=17_8$) unless modified by reopening the gates to let the original starting state code through or by modifying the output of one or more of the input gates using a "wire or" connection coming from the 1 of 16 decoder output. This address is sent to the I/O address flip flops inputs and clocked in on the leading edge of the first half clock cycle. The first half clock cycle turns off the 1 of 16 decoder and the address changes. The second half clock cycle enables the 1 of 16 decoder, allowing the next state micro instruction to appear. (See FIG. **164** for the timing described above.) The process continues until the exit state is encountered. On the exit state, the I/O Control is disabled and control is returned to the microprocessor.

The I/O instructions involving the transfer of data between the I/O and the CPU (OT, LI, MI), require 16 passes through the same state (1 pass for each of 16 bits). This is achieved by checking the output of a 16 bit down counter and then decrementing after each pass through the state. If the counter indicates 0 has not been reached, it causes the starting state address to be reloaded into the address flip flops by opening the input gates. When 16 passes have been indicated by the counter, the input gates are not allowed to open; however, the next state (1111) is modified by the output of the 1 of 16 decoder through a wire-or connection on the 2cd bit to give state **1101.** This address is input to the I/O address flip flops as in the preceeding paragraph.

The above-described operation of the I/O control section is also illustrated and further described in the flow chart of FIG. **165.**

Bit **9** is called a hold/clear bit. It allows a clear flag (CLF) to take place or not to take place after execution of the other I/O instructions. (STF excepted)

Bit **10** is used in conjunction with the micro instructions $\overline{PTR}$ and $\overline{XTR}$ to give control to the I/O.

The I/O control and programmable clock mnemonics are given in the following table:

I/O CONTROL BOARD MNEMONICS

$\overline{CC\phi}$	Clock Code	Zero
CC1	do.	One
CC2	do.	Two
CC4	do.	Four
CC8	do.	Eight
CCT	Control Clock to Tester	
$\underline{CEM}$	Call Extended Memory	
$\overline{CLC}$	Clear Control	
CLF	Clear Flag	
DRC	Data Register Clock	
EBT	Eight Bit Transfer	
EOW	End of Word	
$\overline{IIC}$	Inhibit Internal OSC	
$\overline{INH}$	Inhibit Clock	
$\overline{IPS}$	Inhibit Primary/Secondary	
ITS	Input to S-Bus	
MCK	Memory Clock	
$\overline{POP}$	Power On Pulse	
$\overline{PTR}$	P-Reg to R-Bus	
QFG	Qualifier	Flag
Q5	do.	Five
Q6	do.	Six
Q7	do.	Seven
Q8	do.	Eight
Q9	do.	Nine
Q10	do.	Ten
QRD	do.	ROM Disable
RCA	ROM Clock Address	
RCF	ROM Clock Flip Flop	
SCB	Set Carry Bit	
SCK	Shift Clock	
$\underline{SCT}$	Shift Clock to Tester	
$\overline{SRA}$	Service Request Acknowledge	
STC	Set Control	
STF	Set Flag	
TCK	Tester Clock	
$\underline{TTO}$	T-Bus to Output	
$\overline{TTX}$	T-Bus to A/B Reg.	
$\underline{XTO}$	External OSC	
$\overline{XTR}$	A/B Reg. to R-Bus	

Note: (—) indicates negative true signal

I/O REGISTER AND GATE INTERFACE SECTION

As shown in FIGS. **162** A–D, the directly addressable I/O Register (address **01**) is a 16 bit universal (Parallel in/out, Serial in/out) register that is connected to the calculator processor by the serial-in $\overline{S}$-Bus and the serial-out T-Bus. Information is passed non-inverted from the A or B registers bit serial to the I/O Register with the I/O instruction OTX **01.** Sixteen lines connected to the parallel outputs of the I/O Register provide data out to the internal input, imput-output, and output units and to the external output interfaces. (NOTE: each I/O unit or interface may place only 1 TTL load on the output lines.)

Parallel entry to the I/O Register is through 12 party lines connected to the 12 least significant parallel inputs. The input lines are negative true with all input interfaces tying to the lines through open collectors. Care must be taken to insure there is no disturbance to the lines while an interface is inactive. Input information is passed inverted to the A or B Register Bit serial with the I/O instructions LIX **01** or MIX **01.** (The inversion puts positive true information into the A or B register.)

Input information is entered into the I/O Register in three ways:

a. Service Request.

Entry by the service request method is controlled by a service inhibit flip flop. When the service inhibit flip flop has been cleared with the I/O instruction CLF **01,** a service request may be initiated by returning the $\overline{SSI}$ (Service Strobe Input) party line to ground through an open collector on the interface. This signal causes the parallel inputs to be strobed into the I/O Register and sends a request for service (QNR) to the microprocessor. The microprocessor prior to receiving a request for service would have been cycling through various instruction paths and checking for a service request after execution of each instruction. Upon receipt of a request for service, the processor interrupts the sequence of instructions it was doing and loads an address into the M-Register which contains the starting address of the service routine. At the same time a signal, $\overline{SRA}$ (Service Request Acknowledge), turns off the service inhibit flip flop and also sets the single service flip flop which permits only one service interrupt to the processor per service strobe input. The single service flip flop is reset when the service strobe is removed. All lines from an interface using the service request method for entering information are inhibited when the Service Inhibit flip flop is set.

b. Return of Channel Flag After Command was Given to an External Peripheral Device.

This method implies the calculator must control

the peripheral. That is to say the calculator transmits the indirect address and control enable ($\overline{CEO}$) from the "I/O Register and gate interface" section to the interface with the expectation of information being returned by the peripheral through the interface to the I/O Register. Because of this expectation, only limited instructions may be performed by the calculator while waiting. The Service Request method must be inhibited during this wait so that input information is not destroyed by another peripheral using Service Request.

When a controlled peripheral responds, its flag and data are processed at the interface. The signal $\overline{CFI}$ (Channel Flag In) causes the loading of parallel data from the interface into the I/O register and clears the control enable flip flop so that the $\overline{CEO}$ signal is removed from the interface. The calculator can interrogate the Control Enable flip flop with the instructions SFS **01** or SFC **01** to determine when data has been loaded in.

c. Giving the I/O Instruction STF **01.**

The instruction STF **01** as described in *a* sets the service inhibit flip flop inhibiting the service request mode of entry. The STF **01** instruction also causes a parallel load of the input lines into the I/O Register.

The I/O Register is used to transfer data and control between the calculator and the directly addressable magnetic card reader (address **02**). To record information on a card, the control word and data is transferred from the A Register to the I/O Register. The I/O instruction STC**02** clocks this information via MLS into a latch located at the card reader. The strobe bit for the recorded data is output to the I/O Register from the B Register. The I/O instruction STF**02** clocks the strobe latch located at the card reader via MCR. The I/O instruction STF**01** loads status from the card reader into the I/O Register (see 1-C). This status is transferred to the A or B Register where it is processed.

To enter information from the magnetic card reader a control word is transferred from the A-Register to the card reader latch as above. When a strobe is encountered from the card, the card reader sends a signal, $\overline{MFL}$, to the I/O Register and gate interface section, which sets the magnetic card flag flip flop. The I/O instruction SFS**02** is used to determine the state of the magnetic card flag flip flop. When the flip flop is set, data is loaded into the I/O Register with the I/O instruction STF**01.**

The directly-addressable output printer (address **04**) requires 26 bits of parallel information from the calculator. Sixteen bits come from the I/O Register and 10 bits come from a register at the printer. A 16 bit word with "don't cares" in the least 6 significant bits is transferred to the I/O Register with the I/O instruction OTX **01.** A second 16 bit word is transferred to the I/O Register with the instruction OTX **04.** The 10 valid printer bits already in the I/O Register overflow into the 10 bit printer register. The significance of the address **04** in the OTX instruction is that it allows the microinstruction EOW (End of Word) to set the printer enable flip flop after the 16th bit has been transferred. At the end of the printers response it returns a signal ($\overline{PTF}$) to the printer enable flip flop clearing it. The printer enable flip flop can also be cleared with the I/O instruction CLF **04.** The state of the printer enable flip flop is checked with the I/O instructions SFC **04** or SFS **04.**

The directly-addressable output display (address **08**) recieves information from the I/O Register. A 16 bit word is transferred to the I/O Register with the instruction OTX **08.** The address **08** allows the display enable flip flop to be set with the micro-instruction EOW after the 16th bit has been transferred. The display enable flip flop sends a signal $\overline{DEN}$ to the display indicating information is ready in the I/O Register. The display enable flip flop is cleared with the I/O instruction CLF **08.**

The directly-addressable keyboard indicator lights (address **16**) are used to indicate the mode for various keys. The lights are driven from a latch or series of latches which get information from the I/O Register. Information is transferred to the I/O Register with the I/O instruction OTX **16.** After the 16th bit has been transferred, the microinstruction EOW and address **16** send a signal KLS which strobes the information from the I/O Register into the keyboard lights latch.

The keyboard does not have an address. It transfers information to the CPU through the I/O Register using the Service Request method described in *a* above. The keyboard is disabled while the service inhibit flip flop is set except for the stop key which when depressed sends the signal $\overline{STP}$ which is processed independent of the service inhibit flip flop.

All external peripheral interfaces are indirectly addressed from the four most significant bits in the I/O Register. Thus to communicate with an external peripheral, an address (**0000** excluded) must be loaded into the I/O Register. Data and status will be loaded at the same time if the peripheral is to act as a receiver. If the peripheral is to act as a transmitter, only the address and status need be loaded. Next, the I/O instruction STC **01** sets the Control Enable Out flip flop. This flip flop sends a signal $\overline{CEO}$ to all external interface slots. The $\overline{CEO}$ signal and the decoded (from the 4 bit address) address allow the interface to command the peripheral. After the peripheral has responded, information given back to the interface by the peripheral is processed to the I/O Register in the manner described under *b*, "Return of Channel Flag After Command was Given to an External Peripheral Device".

The I/O Register and gating control circuit mnemonics are given in the following table:

I/O REGISTER AND GATE BOARD

$\overline{CEC}$	Control Enable Out	
$\overline{CFI}$	Channel Flag In	
CLF	Clear Flag	
COϕ, 1,2,3	Code Out	
$\overline{DEN}$	Display Enable	
DIϕ, 1,2,3,4,5,6,7	Data In	
DOϕ, 1,2,3,4,5,6,7	Data Out	
DRC	Data Register Clock	
EBT	Eight Bit Transfer	
EOW	End of Word	
IOD	I/O Data	
KLS	Key Lights Strobe	
$\overline{MCR}$	Mag Card Reset	
$\overline{MFL}$	Mag Flag	
MLS	Mag Latch Strobe	
$\overline{PEN}$	Printer Enable	
$\overline{POP}$	Power On Pulse	
$\overline{PTF}$	Printer Flag	
Qϕ	Qualifier Bit	ϕ
Q1	do.	1
Q2	do.	2
Q3	do.	3
Q4	do.	4
QFG	do.	Flag
$\overline{QNR}$	do.	Not Request
$\overline{SIH}$	Service Inhibit	
SIϕ, 1,2,3	Status In	
SOϕ, 1,2,3	Status Out	

-Continued

I/O REGISTER AND GATE BOARD

$\overline{SRA}$	Service Request Acknowledge
$\overline{SSI}$	Service Strobe In
STC	Set Control
STP	Stop
STF	Set Flag
T-Bus	T-Bus
TTO	T Bus to Output

NOTE: (⁻) indicates negative true signal

As shown in FIG. **166,** when addressing a peripheral device, bits loaded into the four most significant locations in the I/O register from the CPU constitute the peripheral address code. As part of the output party line system the address code is routed to all I/O interface slots. Each I/O interface card decodes the four line address code to a unique single line for use on that particular I/O card. The binary codes **10** through **15** have been reserved for dedicated peripheral addresses which are used by dedicated keys (from the keyboard) and dedicated I/O drivers. Binary codes **1** through **9** are for general use. Code **0** is a non-addressing code and is used in operations that do not involve addressing a specific peripheral. The following table summarizes the address code assignments:

ADDRESS CODE ASSIGNMENTS

ADDRESS	4-BIT CODE	ASSIGNED PERIPHERAL
15	HHHH	TYPEWRITER
14	HHHL	PLOTTER
13	HHLH	
12	HHLL	KEYBOARD & KEYBOARD-LIKE PERIPHERALS
11	HLHH	
10	HLHL	
9	HLLH	GENERAL USE; ONE OF NINE SELECTABLE
8	HLLL	do.
7	LHHH	do.
6	LHHL	do.
5	LHLH	do.
4	LHLL	do.
3	LLHH	do.
2	LLHL	do.
1	LLLH	do.
ϕ	LLLL	USED ON INTERRUPT I/O INTERFACE CARDS WHEN THE INTERRUPT BECOMES ENABLED

The general usage codes (**1–9**) are decoded outputs from a four line to 1 of 10 decoder (SN 7442 for example). It is intended that the codes **1** through **9** be jumper selectable. This would allow the user to select a code for his system peripherals or allow him to use more than one of the same peripheral by selecting different address codes.

Since the I/O register is used to communicate with the internal input, input-output, and output units as well as peripheral devices, a given peripheral's address code will appear randomly in the I/O register address field with there being no intention of expecting the peripheral to respond: Therefore, a second piece of information is necessary for the I/O interface card to form a unique signal which will indicate to the peripheral to respond. This second piece of information is control information and is described hereinafter.

The I/O interface cards contain TTL compatible logic for manipulating control and data from the calculator and/or the peripheral. All I/O interface cards which are intended to be used with the calculator must provide storage either on the I/O interface card or in the peripheral. Thus data being transferred from the calculator to the I/O card must be stored at the instant the peripheral is requested to respond. Likewise data coming from a peripheral must be stored until the calculator accepts it. This requirement is important and must be considered on all compatible interface cards.

The calculator can supply up to 100 ma. maximum at +5 volts to each I/O interface card. Power exceeding this absolute maximum must be supplied by the peripheral.

The following table lists the pin assignments for all I/O lines at the plug-in slots on the calculator back plane, as viewed from the rear of the calculator, left to right.

EXTERNAL I/O INTERFACE PIN ASSIGNMENTS

	1	A
	2 +5	B +5
	3 USED	C USED
	4 USED	D 10/20
	5 USED	E USED
	6 DI ϕ	F DO ϕ
	7 DO 1	H DO 2
	8 DI 3	J DO 3
	9 DI 2	K DI 1
	10 DO 4	L DI 4
	11 DO 5	M DI 5
	12 DO 6	N DI 6
	13 DO 7	P DI 7
	14 SO ϕ	R SI ϕ
	15 SO 1	S SI 1
	16 SO 2	T SI 2
	17 SO 3	U SI 3
	18 CO ϕ	V CO 1
	19 CO 2	W CO 3
	20 $\overline{SSI}$	X $\overline{SIH}$
	21 $\overline{CEO}$	Y $\overline{CFI}$
	22	Z $\overline{STP}$

FIG. **167** lists all I/O lines with brief definitions and specifications and FIG. **163** shows the source and relative relationship of the I/O lines. The output address data lines (Co O–3) transmit the address code along the party lines to all interface slots. These lines will go high and low according to information being shifted in or out of the I/O Register. At anytime a peripheral is addressed the lines will become steady 1 instruction time (8 μs) before control information is passed to the I/O interface card or before data or status is taken from the I/O interface card or before data or status is taken from the I/O interface card and will remain constant until the control information is removed. After the control information is removed, the state of the I/O lines become unpredictable until the next addressing takes place. Address data coming to the I/O interface card is positive true and each interface may place 1 TTL load on each address line.

The output data lines (DO **0–7**) output data from the A or B accumulator in 8 bit bytes from the eight least significant locations in the I/O register to all interface card slots. The logic state is positive true (Data = 1 = High). Each interface card may place 1 standard TTL load on each data line.

The output data status lines (SO **0–3**) output status data from the A or B accumulator and are driven from the next four locations above the data out positions in the I/O register. (DO positions = **0** thru **7**; SO positions = **8** thru **11.**) These lines are used for sending additional information to a peripheral. The logic state is positive true. One standard TTL load may be placed on each output data status line. (Special drivers, fast data transfer, and interrupt do not make use of SO **3.**)

The input data lines (DI 0–7) transmit input data in 8 bit bytes to the eight least significant bit positions of the I/O register (locations 0 thru 7) from the I/O interface card. Each "Data In" line has a 1K pull up resistor to +5 volts and under the party line system must be driven low for a logical 1 from open collector gates on each addressed I/O interface card. The logic state is negative true.

The input data status lines (SI 0–3) receive information from the I/O interface cards and transmit it to locations 8 through 11 in the I/O register. Each line has a 1K pull us resistor to +5 volts. These lines are used to provide additional information to the calculator about the state of a peripheral. The logic state is negative true.

The negative true "Device Ready" output line ($\overline{CEO}$) transmits a control signal, which when combined with an address code will initiate a peripheral response on the addressed I/O interface card. Device Ready is controlled by the I/O interface driver and therefore may look different depending upon the driver. For example, when the calculator wishes to transmit data to the I/O interface card or to initiate a peripheral response prior to receiving data from the peripheral, the calculator causes the Device Ready output lines to go low and stay low until the peripheral response is over and the calculator receives the signal "Device Request" ($\overline{CFI}$) from the I/O interface card. The Device Ready flip-flop always receives a clear signal whenever the I/O reister completes a parallel load.

The Device Request party line $\overline{CFI}$ when driven low from an open collector gate on the I/O interface card will cause the loading to parallel input information into the 12 least significant locations of the I/O register. The active state of the line is low (negative true).

The peripheral flag, indicating to the I/O interface card the peripheral has received data/control or is ready to input data, is gated through an open collector nand gate onto the Device Request ($\overline{CFI}$) party line. The open collector gate is enabled by the I/O interface card's address and Device Ready ($\overline{CEO}$). The Device Request line is pulled up inside the calculator by a 1K resistor to +5 volts.

The Device Request ($\overline{CFI}$) signal must stay low until Device Ready ($\overline{CEO}$) has been cleared (goes high). At this time data transfer has terminated and peripheral's flag and control must be cleared in preparation for the next pass. Since a parallel load in the I/O register causes the Device Ready flip-flop to receive a clear signal, when a Device Request ($\overline{CFI}$) is entered, a parallel load takes place and afterward Device Ready ($\overline{CEO}$) is cleared. The calculator uses Device Request in its general mode of data transfer.

The Half Status output line ($\overline{STP}$) is a line that goes low when the STOP key on the calculator is depressed. It will stay low for the duration of the key depression. One standard TTL load may be placed on this line by each I/O interface card.

The prevent Interrupt output line ($\overline{SIH}$), when low indicates to the I/O interface card that a request for service must not be given to the calculator. One standard TTL load may be placed on this line by each I/O interface card.

The Service Request (Lo) lines ($\overline{SSI}$), when driven low, causes the loading of parallel input information into the 12 least significant locations of the I/O register and causes a CPU interrupt for service. The peripheral's request for service is gated with the Prevent Interrupt ($\overline{SIH}$) line onto the Service Request party line through an open collector nand gate. A 1K pull-up resistor to +5 volts is connected to the line inside the calculator.

The line 10/20 is used on the I/O interface card to differentiate between the present calculator and other calculators, thereby permitting the use of compatible I/O interface cards. It is grounded in the case of the present calculator and open in the case of others.

The general format for all data transfer consists of 8 bit parallel bytes (this calculator, like the Hewlett-Packard model 9100 calculator uses a 7 bit field). Other data formats are handled by specially developed drivers, such as the ROM plug-in module employed for driving the typewriter.

The state of a peripheral is generally checked before attempting an output. This is done by first inhibiting the interrupt system. The address of the I/O interface card is shifted into the I/O register. The decoded address code enables the open collector gates on the I/O interface card. The status of the peripheral is passed to the Status In lines and loaded into the I/O register with an I/O instruction issued by the calculator. The I/O register information is transferred to the A or B accumulator and processed. If the peripheral is ready, the output data word consisting of the address code, output status (if necessary) and the eight bit data byte is formed in the A or B accumulator. The output data word is transferred to the I/O register after which the Device Ready ($\overline{CEO}$) flip-flop is set. The I/O interface card receives the data, address code and Device Ready and a peripheral response is initiated. The calculator interrogates the state of the Device Ready flip-flop to determine when the I/O interface card has received the information and the peripheral response is done. The peripheral I/O interface card signals the calculator it is done by transmitting the Device Request ($\overline{CIF}$) signal to the calculator. The output waveforms are shown in FIG. 168.

Before inputing data from an I/O interface card it is necessary to determine if the peripheral has responded and is ready to input data. After a peripheral response has been initiated, as described previously, the calculator waits for the Device Request ($\overline{CFI}$) which loads the data into the I/O Register and clears the Device Ready ($\overline{CEO}$). The calculator checks the state of Device Ready and when it goes false ($\overline{CEO}$ = HIGH), the calculator knows data is present in the I/O Register and proceeds to shift it into the A or B accumulators for processing. The input waveforms are shown in FIG. 169.

When blocks of data are to be transferred between a peripheral and the calculator, the interrupt is turned off, and transfer rates as high as 100,000 bits/sec may be possible. Before either input or output of a block of data can start, it is necessary for the calculator to check the status of the peripheral to see if it is turned on and ready. Register will remain unchanged during the block transfer. A single I/O instruction shifts the 8 bit byte of data from the eight least significant locations in A or B to the eight data locations in the I/O Register; gives Device Ready ($\overline{CEO}$) goes low) 120 ns after the shift is completed; and shifts the 8 most significant bits in A or B to the eight least significant locations in A or B in preparation for the next transfer. (Note the Address and status field in the I/O Register are not disturbed in the shifting). Device Ready stays true (low) until the peripheral has received the data and is ready for more. The I/O interface card then returns Device Request ($\overline{CFI}$) to the calculator. The receiving of Device Re-

quest by the calculator causes loading of the parallel input party lines into the input status and input data locations of the I/O Register, and clears the Device Ready signal ($\overline{CEO}$) goes high). The logic sense of Device Ready is observed by the calculator and when it goes false ($\overline{CEO}$ = High) the CPU proceeds to output the next eight bit byte of data.

If the output I/O interface card is not returning information on the input lines all input lines will be high when the loading, described in the preceding paragraph, takes place. Therefore, if at the beginning the code in the output status field is being used by the I/O interface card and must remain something other than all high it will be necessary for the I/O interface card to receive the output status from the calculator and return it back to the status inputs so that when Device Request occurs the status field does not get changed in the I/O Register.

Input: After determining if the peripheral is ready to start transferring a block of data the calculator turns off the interrupt and shifts the address code into the I/O Register. (The Address code remains unchanged during the block transfer.) The Device Ready is given ($\overline{CEO}$ = Low) to the calculator when the 8 bit data byte is ready for input. The Device Request signal causes the input data and status to be loaded into the I/O Register and causes Device Ready to go false ($\overline{CEO}$ = High). The calculator by checking when Device Ready goes false knows the data has been loaded. A single I/O instruction shifts the 8 bit data byte from the I/O Register into the eight most significant locations in the A or B accumulators (shifting the previous information in A or B **8** places to the right) and causes Device Ready to go true ($\overline{CEO}$ = Low) 120 ns after the last bit has been shifted into A or B. As before if output status is to be retained on the I/O interface card it must be returned to the I/O Register upon each input data transfer. Wave forms illustrating high speed operations resemble the wave forms for the calculator in FIGS. **8** and **9.**

The interrupt system is controlled by the Service Inhibit flip-flop. An interrupting peripheral is allowed to request service for input from the processor anytime the Prevent Interrupt line is disabled ($\overline{SIH}$ = High). The calculator allows only those peripherals which use the calculator key-codes to interrupt, and these must be user controlled such that only one interrupt at a time is taking place. Data (keycodes) is loaded into the I/O Register at the instant the Service Request is given by the I/O interface card ($\overline{SSI}$ = Low) if Prevent Interrupt is disabled ($\overline{SIH}$ = High) or if Prevent Interrupt having been enabled ($\overline{SIH}$ = Low) goes high while Service Request is being given. At the same time a qualifier is routed to the CPU indicating a request for service is active. Recognizing the request for service the CPU branches to the service routine which enables Prevent Interrupt ($\overline{SIH}$ = Low) and loads the starting address for the software service routine into the M-Register. After servicing the interrupt entry the Prevent Interrupt is disabled ($\overline{SIH}$ = High) and the next interrupt allowed to take place. The interrupt waveforms are shown in FIG. **170.**

The following table lists the general I/O instruction set and the associated codes:

I/O INSTRUCTION SET

NAME	INSTRUCTION EXECUTION TIME	INSTRUCTION CODE 15	14	13	12	11	10	9	8	7	6	5	4 3 2 1 0
STF	9 μs	H	H	H	H	—	H	L	H	H	H	H	SELECT CD
CLF	9 μs	H	H	H	H	—	H	H	H	H	H	H	do.
SFC	9 μs	H	H	H	H	—	H	H/C	H	H	H	L	do.
SFS	9 μs	H	H	H	H	—H	H/C	H	L	H	L	do.	
CLC	9 μs	H	H	H	H	—	H	H/C	H	L	H	H	do.
STC	9 μs	H	H	H	H	—	H	H/C	H	H	L	L	do.
OT*	15 μs	H	H	H	H	A/B	H	H/C	L	L	H	H	do.
LI*	15 μs	H	H	H	H	A/B	H	H/C	L	H	L	H	do.
MI*	15 μs	H	H	H	H	A/B	H	H/C	L	L	L	H	do.

The following describes the function of each I/O instruction with the five allowable select codes.

STF	<SC>		Set the flag. STF is a 240 *ns* positive true pulse which accomplishes the following with the various select codes.
	STF	$\phi\phi$	Not used by the calculator.
	STF	ϕ1	a. Sets the Service Inhibit flip-flop to the true state ($\overline{SIH}$=Low; interrupt not allowed). b. Causes parallel input data and status to be loaded into the I/O Register.
	STF	ϕ2	Generates a 240 *ns* positive true MCR pulse for the magnetic card reader.
	STF	ϕ4, ϕ8, 16	Not used by the calculator.
CLF	<CF>		Clear the flag. CLF is a 240 *ns* positive true pulse which accomplishes the following with the various select codes.
	CLF	$\phi\phi$	Not used by the calculator
	CLF	ϕ1	a. Clears the Service Inhibit flip-flop to the false state. ($\overline{SIH}$=High; interrupt allowed.) b. Loads address locations in I/O Register with ϕ's. (ϕ=Low) c. Clears Device Ready flip-flop ($\overline{CEO}$= High).
	CLF	ϕ2	Clears magnetic card reader flag flip-flop.
	CLF	ϕ4	Clears Printer Enable flip-flop (PEN= Low).
	CLF	ϕ8	Clears Display Enable flip-flop ($\overline{DEN}$= High).
	CLF	16	Generates a 240 *ns* positive true KLS pulse.

SFC	<SC> *ns*	H/C	Skip if flag clear. SFC is a 240 *ns* positive true pulse which accomplishes the following with the various select codes. If C is given a 240 *ns* CLF pulse is given after SFC.
	SFC	$\phi\phi$	Causes the next instruction to be skipped if the STOP key has not been depressed.
	SFC	$\phi 1$	Causes the next instruction to be skipped if Device Ready is true ($\overline{CEO}$=Low).
	SFC	$\phi 2$	Causes the next instruction to be skipped if the magnetic card reader flag flip-flop is clear.
	SFC	$\phi 4$	Causes the next instruction to be skipped if the printer enable flip-flop is clear. ($\overline{PEN}$=Low).
SFS	<SC>	H/C	Skip is flag set. SFS is a 240 *ns* positive true pulse which accomplishes the following with the various codes. If C is given then a 240 *ns* CLF Pulse is issued after SFS.
	SFS	$\phi\phi$	Causes the next instruction to be skipped if the STOP key is depressed.
	SFS	$\phi 1$	Causes the next instruction to be skipped if Device Ready is false ($\overline{CEO}$=High).
	SFS	$\phi 2$	Causes the next instruction to be skipped if the magnetic card reader flag flip-flop is set.
	SFS	$\phi 4$	Causes the next instruction to be skipped if the printer enable flip-flop is set (PEN=High).
$\overline{CLC}$	<SC>	H/C	Clear Control. $\overline{CLC}$ is a 240 *ns* negative true pulse and is not used by the calculator. If C is given then a 240 *ns* positive true CLF pulse is given after $\overline{CLC}$.
STC	<SC>	H/C	Set the Control. STC is a 240 *ns* positive true pulse which accomplishes the following with the various select codes. If C is given a 240 *ns* CLF pulse is issued after STC.
	STC	$\phi\phi$	Not used by the calculator.
	STC	$\phi 1$	Sets the Device Ready flip-flop ($\overline{CEO}$= Low).
	STC	$\phi 2$	Generates a 240 *ns* positive true MLS pulse for the magnetic card reader.
	STC	$\phi 4, \phi 8, 16$	Not used by the calculator.
OTX	<SC>	H/C	Output A or B causes data bits from A or B to be shifted to the I/O Register and accomplishes the following with the various select codes. If C if given, a 240 *ns* CLF pulse is given after OTX is executed.
	OTX	$\phi\phi$	The 8 least significant bits in the A or B register are shifted non-inverted to the 8 least significant locations in the I/O Register, and 120 *ns* after the 8th shift the Device Ready flip-flop is set ($\overline{CEO}$=Low). The 8 most significant bits are shifted right 8 places and the least 8 significant bits are recirculated to the 8 most significant locations in the A or B registers. The 8 most significant bits in the I/O Register are untouched.
	OTX	$\phi 1$	Sixteen bits from the A or B register are shifted non-inverted to the I/O Register. The data in A or B recirculates.
	OTX	$\phi 2$	Not used by the calculator
	OTX	$\phi 4$	Same as OTX $\phi 1$ and in addition, 120 *ns* after the 16th bit has been shifted the printer enable flip-flop is set.
	OTX	08	Same as OTX $\phi 1$ and in addition, 120 *ns* after the 16th bit has been shifted the display enable flip-flop is set.
	OTX	16	Same as OTX $\phi 1$ and in addition, 120 *ns* after the 16th bit has been shifted the 240 *ns* KLS signal is generated.
LIX	<SC>	H/C	Load into A or B. Loads data bits from the I/O Register into the A or B Registers and accomplishes the following with the various select codes. If C is given, a 240 *ns* CLF pulse is given after LIX is executed.
	LIX	$\phi\phi$	The eight least significant bits in the I/O Register are shifted inverted to the eight most significant locations of A or B, and 120 *ns* after the 8th shift the Device Ready flip-flop is set ($\overline{CEO}$=Low). A or B is shifted right eight places as the I/O Register data comes in. The 8 most significant bits in the I/O Register are untouched.
	LIX	$\phi 1$	The 16 bits of the I/O register are transferred inverted to the A or B register. Data in the I/O Register is lost.
	LIX	$\phi 2, \phi 4, \phi 8, 16$	Not used by the calculator.
MIX	<SC>	H/C	Merge into A or B. Merges data from the

		I/O Register into A or B registers and accomplishes the following with various select codes. If C is given, a 240 *ns* CLF pulse is given after MIX is executed.
MIX	$\phi\phi$	The eight least significant bits in the I/O register are merged with the eight least significant bits of the A or B register and shifted to the 8 most significant locations of A or B; 120 *ns* after the merge takes place the Device Ready flip-flop is set ($\overline{CEO}$=Low). A or B shifts right 8 places as the data is merged and shifted to the most significant locations. The 8 most significant bits of the I/O Register are untouched.
MIX	ϕ1	The 16 bits of the I/O Register are merged with the 16 bits of the A or B register and contained in the A or B register.

MIX 02, 04, 08, 16 Not used by the calculator.

Examples of various drivers which transfer data are given below:

Example 1: Typical Subroutine to Get Status of I/O Device.

Calling Sequence:

```
    LDB Select Code
    JSM Stat
```

```
Stat  STF 1     Turn off the interrupt system.
      OTB 1     Load I/O Register with select code.
      STF 1     Load I/O Register with status of I/O
                device.
      LIA 1     Loan A Register with status information.
      CLF 1     Turn on interrupt.
      RET       Return.
```

Example 2: Typical Subroutine to Output an 8 bit character.

Calling Sequence:

```
      OTA 1     Output 16 bits to the I/O Register.
      STC 1
      SFS 1     Loop until I/O flag is set by the
      JMP *-1   output device.
      CLF 1
```

Example 3: High speed output where the calculator is faster than
output device

Calling Sequence:

```
      ST* 1     -(Number of 16 bit words to be output)+1
      ST* J     Address of first word in the array.
      LDB SC    Select Code
      JSM OUT2
OUT2  JSM STAT  Get status of output device
      RAR 9     and position it.
```

Example 4A: Typical subroutine to input an 8 character.

Calling sequence is:

```
    LDB Select code
    JSM In
    ... Return is made with the data in the A Register.
```

```
In  STF   1       Turn off interrupt system
    OTB   1       Load I/O Register with the select code
    STC   1, C    Pulse the flag & turn interrupt system
                  on
    JSM   STAT    Get status off the input device
    RAR   9       and position it.
    SAP   *-2, C  If device is busy then continue to
    SAR   7       loop else position data bits
    RET           Return.
    SAP   OUT2    If device is busy, continue to loop
    STF   1       Turn off interrupt system.
    OTB   1       Output select code
    LDB   1       B  Counter for number of words to be
                  output
    LDA   J, I    Load next data word
    SEC   *+1, C  E← $\phi$
```

-Continued

```
    LDB Select code
    JSM In
    ... Return is made with the data in the A Register.
```

```
    OTA   $\phi$     Output 8 bits from A
    SFS   1       Loop until device sets
    JMP   *-1     flag.
    SEC   *-3, S  If E=$\phi$ and E← J then loop to output
                  last 8 bits
    ISZ   J       Increment array address pointer
    RIB   *-7     Increment count and loop if not
                  finished.
    CLF   1       Turn on interrupt system
    RET           Return
```

Example 3B: If the output device is faster than the calculator then
fewer instructions can be used.

```
    OTA$\phi$     Output first 8 bits
    OTA$\phi$     Output second 8 bits.
```

Example 5A: High speed input where the calculator is faster than
the input device.
Calling sequence:

```
    ST* 1     -(Number of 16 bit words to be input)+1
    ST* J     Address
    LDB SC    Select code
    JSM In2
    In 2 JSM  Get status of input device
    STAT
    RAR 9       and position it.
    SAP In2   If device is busy, continue to loop
    STF 1     Turn off interrupt system
    OTB 1     Output select code
    STC 1     Command device to read
    LDB 1     R←Counter for number of words to be input
    SEC *+1, C  E← $\phi$
    SFS 1     Loop until input
    JMP *-1     device sets flag
    LIA $\phi$   Load 8 bits from I/O Register
    SEC *-3, S  If E=$\phi$ and E→1 then loop to input last 8 bits
    STA J, I  Save data word in array
    ISZ J     Increment array address pointer
    RIB *-7   Increment count and loop if not finished.
    CLF 1     Turn on interrupt system
    RET       Return
```

Example 5B: If the input device is faster than the calculator then
the number of instructions can be reduced.

```
    LIA $\phi$   Input first 8 bits
    LIA $\phi$   Input second 8 bits
```

All output I/O interface cards which are to be fully interchangeable with both the present and other calculators must have storage either on the I/O interface card or in the peripheral to which information is being transmitted. FIG. **171** illustrates the logic used to interface an X-Y plotter which has storage on the I/O interface card.

Blocks A and B are the storage latches which store information coming from the I/O register. When the output of gate C goes high, data is latched; when low, the outputs of the latch track the inputs. Gates D decode the Address code (**14** = **1110**) and pass it positive true to gate E. Device Ready ($\overline{CEO}$) is also passed positive true to gate (E). Gates (H) are open collector and pass status and Device Request ($\overline{CFI}$) onto the input party lines. An example of a **9810A** output would be: Output the address **14** which enables status gates H and see if the power is on. If on, output address, status, and data to gates A, B, and D. The output of C is low allowing data and status to pass. Next give Device Ready ($\overline{CEO}$ = Low) this enables flip-flop G, clocks flip-flop F which causes A and B to latch, and sends control to the peripheral. The peripheral acknowledges receipt of control by returning FLAG (FLAG = High) in a busy state this continues to keep A and B latched and clears control flip-flop F. When the peripheral is done acting, the FLAG is returned to the not busy state (FLAG = Low) which clocks flip-flop G and causes output at C to go low enabling A and B. The output of G drives the $\overline{CFI}$ gate which has been enabled from E and $\overline{CFI}$ goes low. $\overline{CFI}$ is received by the calculator which responds by returning $\overline{CEO}$ high. This causes the output of E to go low, clearing flip-flop G and returning $\overline{CFI}$ high. This completes one output cycle.

All input I/O interface cards which are to be fully interchangeable with both the present and other calculators must have storage either on the I/O interface card or in the peripheral from which information is being received. FIG. **172** illustrates the logic required on a general purpose interface card with storage.

Block A is used to store information coming from the peripheral. B stores status coming from the I/O register which may be needed by the peripheral. The output tracks the input whenever the enable on the latch is low. Block C decodes the address code into one of 10 addresses which are jumper selectable. An example of a calculator input would be as follows. The address code would be decoded by C. The calculator would load status through the open collector input status gates D. If the peripheral is on and ready, the address code and output status (if necessary) would be sent to B and C. The decoded address is passed, positive true, to gate E. The enable at B is low so that status is passed to the peripheral. The Device Ready is given ($\overline{CEO}$ = Low) and comes to E positive true. The output of E clocks flip-flop F through gate H. The output of F gives control to the peripheral and also enables A to receive data. The peripheral responds in a busy state (FLAG = High). When data is ready to be input the FLAG is driven low. Data is latched when the FLAG goes low in A. Also when FLAG goes low, G, having been enabled by the output of H, is clocked driving J from its Q output. I is enabled by the output of H and so $\overline{CFI}$ is driven low. Data is loaded into the I/O register from open collector gates I and $\overline{CEO}$ driven high as a result of the calculator receiving $\overline{CFI}$. This clears flip-flop G and disables the input gates I completing an input cycle.

FIG. **173** illustrates the logic required, on an I/O interface board, to input using the interrupt.

A power preset circuit, block A, will be necessary on this card to prevent an interrupt when the peripheral power is turned off or on. This can usually be done by sensing the peripherals' +5 volts and presetting when the voltage drops below 3 to 4 volts.

The calculator must be in a display routine such that the Prevent Interrupt is false ($\overline{SIH}$ = High) before an interrupt can take place. This enables the flip-flop B to be clocked from the peripheral FLAG. The clock inputs to E and F have been high enabling the storage latches prior to the receipt of FLAG. FLAG clocks B causing E and F to latch, enabling open collector gates H through gates I and J, and drives gate G which sends Service Interrupt ($\overline{SSI}$ = Low) to the calculator. The calculator responds by loading the data into the I/O register and returning Prevent Interrupt true ($\overline{SIH}$ = Low). This clears (B). Prevent Interrupt is returned false after the entry has been processed and the cycle is complete.

KEYBOARD INPUT UNIT

The keyboard input unit is shown in FIGS. **174A–D** and **175.** It includes a contactless keyboard of the type shown and described in U.S. Patent application Ser. No. 74,949 entitled NONCONTACTING KEYBOARD, filed on Sep. 24, 1970, issued as U.S. Pat. No. 3,668,697 on June 6, 1972, and assigned to the same assignee as this patent application. The contactless keyboard is made up of an array of printed circuit transformers. Each transformer has its secondary and primary interlaced in a spiral coil as shown in FIGS. **176–177.** The secondaries of all the coils are tied in series to form the sense line. The primaries of the coils are arranged in separate pairs. Each coil is connected in series, with opposite polarity, to its pair as shown in FIG. **178.** Every pair has a drive and sink line, which is being selected and driven by the scanner.

Centered above each coil is a metal disc at the end of the key shaft. When a key is depressed the disc proximates the coil. The disc acts like a shorted turn and reduces the coupling of the coil, and unbalances the pair. This unbalance is amplified by the comparator, when it is greater than the on bias. The comparator triggers the one shot, which turns off the scanner and lowers the on bias. The scanner remains at its present state, which corresponds to the drive and sink line of the key depressed. This state is the keycode of the key pressed. When the key is released a spring retracks the key and disc. When the unbalance is less than the new bias, the comparator turns off and the scanner starts again ready for a new key. The two bias levels give the key mechanical hysteresis.

When two keys are depressed the first one down will be entered, and as long as a key is down no other key can be entered. An exception is when the other key is its pair. In this case the two keys will cancel each other. When the first key is released, the second one will be entered. When the first key is released, while more than one other key is down, the next key to be entered will be the next in the scan sequence, not necessarily the second key down.

For the keyboard to work each pair of primary coils must be balanced. To balance a pair of coils the following rules should be used when laying out the printed circuit board:

1. Sense lines must run in pairs. The closer the better. They should be thin traces.
2. The sense windings of a pair of coils can be anywhere on the sense line. For best results they should be close.

3. Drive lines should be in pairs when possible. Drive clamp and source lines should be grouped together well away from the sense lines. When a drive line crosses a sense line it should be at right angles.
4. Connect to spiral so to add a turn (or part of a turn) not to subtract. Try to duplicate additional turns on a spiral pair. Connect to spiral at a right angle, from a distance.
5. For a pair of spirals separated by some distance, run the common connection away from the sense line and in the drive grouping.
6. Check each pair of spirals for errors in drive or sense polarity. This can cause either an incorrect code (least digit), or a constant full output. One method to check for proper polarity is to assign current direction for both drive and sense. Then at each spiral check for proper polarity. This is illustrated in FIG. **179.**

MAGNETIC CARD READING AND RECORDING UNIT

The magnetic card reading and recording unit is shown in the block diagram of FIG. **180** and in the detailed schematic diagram of FIGS. **181**A–G. The manner in which it interacts with the calculator and operates to record and load secure and unsecure programs and to separately record and load data is shown and described in the block diagram of FIG. **182,** the flow charts of FIGS. **60–61** and **63–65,** and in the memory map of FIG. **62.**

Operation of the card reader is largely automatic. It is only necessary to specify the type of operation and the limits desired. These commands are entered via the calculator keyboard. The calculator then determines the necessary commands required to cause the magnetic card reader to perform the desired operation.

Several modes of operation are possible. Programs can be recorded on magnetic cards and loaded back into the calculator. Similarly, program and data information can be recorded and loaded, or data alone. Very long programs or blocks of data can be stored on several cards. The information is loaded back into the calculator by inserting the cards into the reader in the same sequence as they were recorded. The proper linking of the information stored on the cards is automatically performed by the calculator.

Information is stored on the magnetic card in 3 bit bytes. Three tracks record the information and a fourth track provides a timing mark. Two bytes form six bit words in the calculator. The card reader automatically begins and terminates the recording, irrespective of the length of card used. Different card lengths can be mixed together without affecting the operation of the reader. Cards may be interchanged from one calculator to another.

No mechanical switches are used in the card reader. The only moving part is the card drive motor and capstan. The mechanical assembly and electronics assembly are modular and can be replaced as separate and independent units in the calculator.

OUTPUT DISPLAY UNIT

As shown in FIGS. **183** and **184**A–B, the output display unit includes three 15-character rows of seven-segment light-emitting diode arrays. Each array requires an anode driver for each of its seven segments and one for the decimal point. The cathodes are common to all the diodes in one character. Respective anodes for the characters of a register are tied together to one driver. A series PNP transistor enables one of three sets of eight anode drivers each. The cathodes of a column of three characters are tied together to one of 15 cathode drivers.

The data required to operate the display is as follows:

1. Four bits to determine the column to be enabled.
2. One of three bits to determine the register.
3. Eight bits to determine the diodes of a character to be lighted.
4. A line to enable the display when the data is present.

The duty cycle is less than 1/45 requiring pulses of more current than a character could stand continuously. A retriggerable one shot is fired at every display enable pulse and will disable the column drivers after 1/2 ms. if the machine hangs up.

OUTPUT PRINTER UNIT

Several methods have been described for producing printed characters by thermal means (see particularly U.S. Pat. No. 3,161,457 issued to H. Schroeder et al.) but they typically employ a rectangular matrix of resistors to form an entire character at once. Commercial versions of this sort of printer are marketed by National Cash Register and Texas Instruments. As described in Schroeder's patent, a matrix five elements wide and seven elements high is typically employed.

The output printer unit employed in this calculator is constructed as shown in FIGS. **185, 186, 187**A–B, **188, 189**A–D, and **191**A–B. It includes a row of print elements distributed linearly across a printing head, as shown in FIG. **188,** to print a 16-character line. Each print element is an electrical resistor, of a size and shape intended to produce a dot on thermally-sensitive paper moved at right angles to the line of print elements. Dots are formed in the conventional manner by pulsing the resistor element with a pulse of electrical current, which raises its temperature by joule heating.

Each of the 16 characters of each line is formed in a 5 × 7 dot matrix. For example, as illustrated in FIG. **192** the letter A is produced by printing the darkened dots in the top row and then stepping down to the next row, etc.

Each line of print contains 16 5 × 7 matrices. The matrices are made of seven rows of 80 dots spaced in five dot groups to produce 16 characters. The printer produces each line of print by printing the top row of all 16 characters and then stepping down to print row **2** until all seven rows are printed. Three blank steps are then added to produce the space between lines.

Each of the seven rows of printing contains 80 dots (five for each of the 16 characters) which may or may not be printed. This requires that 80 information bits be supplied for each row printed. To accomplsih this, each row is split into four groups of 20 dots (four characters). (Since the I/O Register of the calculator is only 16 bits long an extra 10 bit shift register is contained in the printer hardware.) Each group of 20 bits is transmitted to the printer along with the group number by the I/O Register and is printed when the printer enable signal is given. The printer then prints that group of dots and returns a printer flag signal to the calculator. The next group of information is then supplied until all 28 groups have been printed. The three step commands are then given to provide the space between lines.

The printer requires the following information to print any group of dots:

1. Dots to be printed,
2. group number, and

3. Printer enable.

As shown and described in the flow chart of FIG. **193**, this information is transmitted to the printer through the calculator I/O register. Since the total number of information bits needed is greater than the I/O registers length, two 16 bit words are transmitted to the printer. The first 16 bit word contains the dot patterns for characters **1** and **2** as shown in the following table:

CONTENT OF I/O REGISTER AFTER FIRST LOADING

	Character 2								Character 1							
L H Dot					R H Dot	L H Dot				R H Dot	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
C03	C02	C01	C0ϕ	S03	S02	S01	S0ϕ	D07	D06	D05	D04	D03	D02	D01	D0ϕ

ϕ—Don't Care

Character **1** is contained in bits SO2–SO0 and DO7, DO6 with the left dot in bit SO2 and the right dot in DO6. Character **2** is contained in bit CO3-CO0 and SO3 with the left dot in CO3 and the right dot in SO3. When the I/O Register is loaded with the second 16 bit word these bits will appear in the internal 10 bit shift register. The second 16 bit word contains the dot pattern for characters **3** and **4** and the group number as shown in the following table:

I/O REGISTER AFTER SECOND LOADING

	Character 4					Character 3								Group	
L H Dot					R H Dot	L H Dot				R H Dot	ϕ	ϕ	ϕ	0 ϕ	0 1 1 1	0 1 0 1
C03	C02	C01	C0ϕ	S03	S02	S01	S0ϕ	D07	D06	D05	D04	D03	D02	D01	D0ϕ

ϕ—Don't Care

Character **3** is contained in bits SO2–SO0 and DO7, DO6 with the left dot in SO2 and the right dot in DO6. Character **4** is contained in bits CO3–CO0 and SO3 with the left dot in CO3 and the right dot in SO3. The group number is contained in bits DO0 and DO2. Groups are numbered as follows:

GROUP	PRINTED CHARACTERS From Left to Right	D01	D00
1	1, 2, 3, 4	0	0
2	5, 6, 7, 8	0	1
3	9, 10, 11, 12	1	0
4	13, 14, 15, 16	1	1

When group **4** is detected the printer automatically steps to the next line. The time interval between printer enable and the return of printer flag is extended to allow the system to physically move. The printing speed is given in the following table:

Group 1	8 ms
Group 2	8 ms
Group 3	8 ms
Group 4	18 ms
Row 1	42 ms
Row 2	42 ms
Row 3	42 ms
Row 4	42 ms
Row 5	42 ms
Row 6	42 ms
Row 7	42 ms
Space	18 ms
Space	18 ms
Space	18 ms
Total Time to Print 1 line	348 ms
Lines per second	2.87 ms

As shown in FIGS. **185–186**, paper is loaded into the ooutput printer unit by lifting the wire bucket cover **220** and placing a roll of paper **222** with the free end into the paper bucket formed by the front and rear bucket halves, **224** and **226** respectively. The only care needed by the operator is to be sure that the paper uprolls forward from the bottom. The wire bucket cover performs a dual function of keeping the free end of the paper in the bucket while loading and after the paper is loaded prevents the free end from reloading itself through the mechanism.

The weight of the paper then stretches the rubber belts **228** and the roll of paper rolls forward until it rests against the paper guide **230**. The paper rolls forward due to the "downhill" slope of the belt from the top of the rear idler pulley **232** to the bottom of the paper guide. With the roll of paper in the position described above the belts moving forward the free end of the paper is constrained by the belts, paper guide and roll to move below the paper guide and between it and the belts.

The belts are driven by the drive pulley **234** which is in turn driven by a gear set from the platen **236**. The platen is driven by the motor **238** via a belt and gear set. The diameter and speed of the platen is such that its surface speed is approximately 5% faster than that of the belts to insure that the paper is always under tension after loading is completed. The drive and rear idler pulleys are crowned so that the belts will be self-centering. The front idler pulley **270** is flat and serves to keep the lower portion of the belt out of the bucket area.

The print head **242** is pressed against the paper and platen by means of a spring, hence it is necessary to remove the print head from the platen while loading paper. This is accomplished by the head lifter and paper deflector **244** such that when it is rotated on its axis it cams the print head off the platen and positions a small plate in the path of the paper which guides the paper up and between the platen and print head.

The paper is guided through the mechanism while loading and while the printer is working by edge guiding the paper. Ordinarily paper does not lend itself well to edge guiding due to its very low compressive strength. To overcome this the paper is bent around the convex bottom surface of the paper guide and the belt, which is under tension, is very near the edge thereby preventing the paper from buckling. The relationship of the paper, paper guide and belts can be seen by looking at Section AA of Figure **186**.

When selecting materials for the various parts of the loading mechanism it is important to be sure that the coefficients of friction between the various parts are compatible. The C.F. between the paper and paper guide should be low relative to the C.F. between the paper and belts in order that the belts can drive the

paper through the mechanism. Similarly, the C.F. between the paper and platen should be high relative to the C.F. between print head and paper in order that the paper can be driven through while printing. In addition, the drag introduced by the belt and paper guide due to the paper moving faster than the belt while printing must not be so great as to tear the paper or impose an impossible load on the motor.

The right half of top panel **90** of the calculator housing is hinged at the back and provided with a handle **246** at the front so that it may readily be raised by the user and stopped at an oblique upright position to expose and facilitate replenishment of the supply of thermal-sensitive paper for the output printer unit and also to serve as a music stand for holding operating or program-running instructions or any other material the user desired. A transparent plastic retainer is mounted on the underside of the hinged right half of the top panel **90** to hold such material.

POWER SUPPLY

The power supply system employed in the caluclator is constructed as shown in FIGS. **194–196** and **197A–B**. As shown in FIG. **194**, a centertapped transformer secondary is connected to terminals **2-3, 2-1, 2-2,** and **2-4**. The AC voltage from the transformer is rectified by diodes **CR1** and **CR2** and filtered by capacitor **C1**. The output of this rectifier/filter circuit is nominally 18 volts DC at 2.7 amps with a 2 volt peak to peak ripple.

Q1 and **Q2** serve as a switch to connect the 5 volt output bus to the 18 volt unregulated supply through inductors **L1** and **L2**. **CR3** serves to clamp the input of L1 to ground when **Q1** and **Q2** are switched off. Current flow in **L1** and **L2** are substantially constant and equal to the load current.

Loss in high current transistor **Q2** is minimized because **Q2** can be completely saturated. Loss in driver transistor **Q1** is minimized because **Q1** can also be saturated. Resistor **R7** limits the maximum drive current to Q2. Losses in **R7** can be minimized by proper positioning of the tap on **L1** consistent with transistor parameters and circuit requirements.

IC1 is a linear differential amplifier integrated circuit to drive **Q1** and **Q2**. Any differential amplifier with sufficient voltage capability and bandwidth will work. Since the amplifier employed is linear, **R7** and **R4** have been included in the circuit to provide sufficient hysteresis for reliable switching. This hysteresis stabilizes the switching frequency and thus stabilizes the switching losses.

Because hysteresis has been added to the circuit, a significant ripple signal (at the switching frequency) must be present on the feedback signal to the amplifier. This need for a ripple signal limits the amount of capacity that can appear between the output of **L1** and ground. **L2** serves to isolate this point from the rest of the system. The amount of capacity that can appear between the output of **L2** and ground is essentially unlimited and significantly reduces power supply ripple, and greatly improves response to load transients.

The second winding of **L2** is a path for the feedback from the remote sensing. The required ripple signal is added to the feedback signal by transformer action in **2**. Another possible configuration is shown in FIG. **98**.

The power supply also includes an overvoltage crowbar circuit (Qr, **CR4,** and **R9**) and a short circuit shutdown circuit (using **Q5**). In the event that the +5 volt bus is grounded, or the crowbar is triggered, **Q5** saturates and locks **IC1** off.

The resistor **R8** makes a current generator of **IC1**. Resistors **R5** and **R6** discharge the bases of **Q1** and **Q2** respectively. **IC2** and its associated components generate a "power on pulse" to initialize the instrument. **IC1** is referenced and powered from an external +12 volt supply. Powering the IC from +12 rather than the unregulated +18 reduces power dissipation in **IC1**.

The +24 volt supply of FIG. **196** is referenced by the +16 volt supply with the amplifier common returning to +12 volts to minimize power loss and voltage stress in **IC1** of FIG. **196**. The +12 volt supply of FIG. **197A** references the +12, +5, and +16 supplies directly. The +12 amplifier **IC1** of FIG. **197A** may be biased either from the unregulated supply for the +12 volt supply or from the operating +16 volt supply. Diodes **CR5** and **CR6** determine the appropriate source. This provides a greater power supply margin for the +12 volt supply. Similarly the +16 volt amplifier is biased from the +20 to give that supply greater margin.

All supplies except the +20 volt supply are current limited. The +24 volt supply is current limited at a value greater than the rating of its series fuse. If a short circuit occurs in the +24 volt supply, it will current limit until the fuse opens. The average current from this supply is 1.1 amps with transients to 2 amps. The current limit is set to 2.5 amps. There is not sufficient thermal capacity available to allow **Q1** of FIG. **196** to carry sustained short circuit current so the fuse has been included to protect the various power supply components. All supplies except the +20 volt supply are crowbar protected against over-voltage.

TYPEWRITER INTERFACE

This interface couples the Facit-Odhner model 3841 output typewriter to the calculator.

The unit mounts directly on the back of the typewriter. Communications with the calculator are made through about 5 feet of cable which is terminated by the I/O plug containing a board for buffering and some logic.

Referring to FIGS. **199A–B** and **200A–C**, characters from the calculator appear on the data lines as ASCII codes. These codes are recoded by a ROM into the six bit Facit typewriter code for the 46 type bars, and one bit for upper case shift. Functions such as space, tab, line feed, etc. are recoded for easy recognition in the interface since each function must be driven by a separate line. A data latch after the ROM holds codes for processing. If new data arrives during this processing, the two codes are compared to determine if they both drive the same type bar and if they are both numbers. Non-repeating numbers can be typed at 14.5 characters per second, otherwise typing speed is 12 characters per second (reduce these speeds 17% for 50HZ operation). Codes in the latch are gated to the program solenoids or the function solenoids by the control logic.

To understand the coding, notice that two blocks of codes on the Facit typewriter code map are empty. If all function codes are put in these blocks, they can be identified by control logic by testing for $(\overline{6 \cdot 4})$. Each function code puts a 1 on one of five lines and this line opens the correct solenoid gate. Bit **8** is used to discriminate between two sets of function gates. In the case of a program solenoid code, bit **8** identifies numerals.

The control clock is provided by a sync. pulse which is generated in the typewriter by a vaned wheel attached to the end of the main drive shaft. The vanes interrupt a light beam. When a type cycle is initiated, a

modulo 8 counter counts sync. pulses and the count is decoded by a 1 or 8 decoder. At each of the eight states, combinational logic can enable solenoid gates, set or clear flag flip-flops or change the counter to state zero, or state 6, or inhibit the counter.

We claim:

1. An electronic desk type calculator, portable calculator, or the like including an input unit for entering information into the calculator, a basic first memory unit into which information may be written and from which information may be read, a basic second memory unit for storing routines and/or subroutines to be executed by the calculator in performing selected functions, plug-in adaptor means for enabling the user to removably plug one or more additional second memory units for storing routines and/or subroutines to be executed by the calculator in performing additional selected functions into the calculator to increase the number of functions that may be performed by the calculator, processing means responsive to information from the input unit or the basic first memory unit and to operating states within the calculator itself for selectively executing one or more of the routine and/or subroutines stored in the basic second memory unit to perform one or more of the selected functions employing information from one or both of the input and basic first memory units, said processing means being further responsive to information from the input unit or the basic first memory unit and to operating states within the calculator itself when an additional second memory unit is plugged into the calculator for selectively executing one or more of the routines and/or subroutines stored in that additional second memory unit to perform one or more of the additional selected functions employing information from one or both of the input and basic first memory units, and an output unit for providing an output indication of the results of the selected and additional selected functions performed by the calculator.

2. An electronic calculator as in claim 1 wherein said basic first memory unit comprises a first section for storing data and a separate second section for storing program steps, and said first and second sections of the basic first memory unit may be separately internally expanded to increase the data and program-step storage capacity of the basic first memory unit.

3. An electronic calculator as in claim 1 wherein said basic first memory unit is a read-write memory, and each of said basic and additional second memory units is a read-only memory.

4. An electronic calculator as in claim 1 wherein said calculator includes one or more receptacles into said calculator and additional second memory units include electrical connection means for electrically connecting the calculator and each of the additional second memory units plugged into one of the receptacles in the calculator, said plug-in adaptor means including the receptacles in the calculator and the electrical connection means, and said processing means includes means for detecting either the presence or the absence of each additional second memory unit plugged into one of the receptacles in the calculator.

5. An electronic calculator as in claim 4 wherein said processing means includes addressing means coupled to the basic first and second memory units and to the electrical connection means for addressing the basic first and second memory units and each of the additional second memory units plugged into one of the receptacles in the calculator, and accessing means coupled to the basic first and second memory units and to the electrical connection means for accessing the basic first and second memory units and each of the additional second memory units plugged into one of the receptacles in the calculator.

6. An electronic calculator as in claim 5 wherein said electrical connection means comprises an electrical connector on each of the additional second memory units and a mating electrical connector in each of the receptacles in the calculator, said addressing means comprises a memory address register and a first bus system for coupling the memory address register to the basic first and second memory units and to the electrical connector in each of the receptacles in the calculator, and said accessing means comprises a memory access register and a second bus system for coupling the memory access register to the basic first and second memory units and to the electrical connector in each of the receptacles in the calculator.

7. An electronic calculator as in claim 6 wherein said basic first memory unit is a read-write memory, and each of said basic and additional second memory units is a read-only memory.

8. An electronic calculator as in claim 6 wherein said input unit is a keyboard input unit including a first plurality of keys for enabling the user to manually initiate the basic functions that may be performed by the calculator without an additional second memory unit, and a second plurality of keys employed alone or with one or more of the first plurality of keys for enabling the user to manually initiate the added functions that may be performed by the calculator when each of the additional second memory units is plugged into one of the receptacles in the calculator.

9. An electronic calculator as in claim 8 wherein said basic first memory unit is a read-write memory, and each of said basic and additional second memory units is a read-only memory.

10. An electronic calculator as in claim 8 including a keyboard overlay for designating the added functions that may be initiated by the second plurality of keys when one of the additional second memory units is plugged into one of the receptacles in the calculator.

11. An electronic calculator as in claim 1 wherein said input unit is a keyboard input unit including a first plurality of keys for enabling the user to manually initiate the basic functions that may be performed by the calculator without an additional second memory unit, and a second plurality of keys employed alone or with one or more of the first plurality of keys for enabling the user to manually initiate the added functions that may be performed by the calculator when each of the additional second memory units is plugged into one of the receptacles in the calculator.

12. An electronic calculator as in claim 11 including a keyboard overlay for designating the added functions that may be initiated by the second plurality of keys when one of the additional second memory units is plugged into one of the receptacles in the calculator.

13. An electronic caluclator as in claim 1 wherein said calculator includes adaptor means for accommodating one or more additional first memory units to internally expand the storage capacity of the basic first memory unit.

14. An electronic calculator as in claim 13 wherein said basic first memory unit includes a first section for storing data and a separate second section for storing program steps, and said adaptor means is operable for accommodating the additional first memory units to separately, internally expand the data and program-step storage capacity of the first and second sections of the basic first memory unit.

15. An electronic calculator as in claim 14 wherein said processing means includes addressing means coupled to the basic first and second memory units and to the adaptor means for addressing the basic first and second memory units and each added first memory unit, and accessing means coupled to the basic first and second memory units and to the adaptor means for accessing the basic first and second memory units and each added first memory unit.

16. An electronic calculator as in claim 15 wherein said addressing means comprises a memory address register and a first bus system for coupling the memory address register to the basic first and second memory units and to one or more electrical connectors of the adaptor means, and said accessing means comprises a memory access register and a second bus system for coupling the memory access register to the basic first and second memory units and to the electrical connectors of the adaptor means.

17. An electronic calculator as in claim 16 wherein said basic first memory unit is a read-write memory, and each of said basic and additional second memory units is a read-only memory.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 3,859,635

DATED : January 7, 1975

INVENTOR(S) : Robert E. Watson, Jack M. Walden and Charles W. Near

It is certified that error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

Column 369, claim 1, line 16, "routine" should read -- routines --;

Column 370, claim 11, lines 10 and 11, delete "one of the receptacles in";

Column 370, claim 12, line 5, delete "one of the receptacles in".

Signed and Sealed this

Twenty-fifth Day of January 1977

[SEAL]

Attest:

RUTH C. MASON
Attesting Officer

C. MARSHALL DANN
Commissioner of Patents and Trademarks