United States Patent [19]

Griffis

[11] **3,926,087**

[45] **Dec. 16, 1975**

[54] **COMPUTERIZED ORGAN REGISTRATION AFFECTING SYSTEM**

[76] Inventor: **Steven W. Griffis,** 4753 Irvine Ave., N. Hollywood, Calif. 91602

[22] Filed: **Oct. 4, 1974**

[21] Appl. No.: **511,996**

[52] **U.S. Cl.** **84/345**
[51] **Int. Cl.²** **G10B 3/10**
[58] **Field of Search** 84/345

[56] **References Cited**

UNITED STATES PATENTS

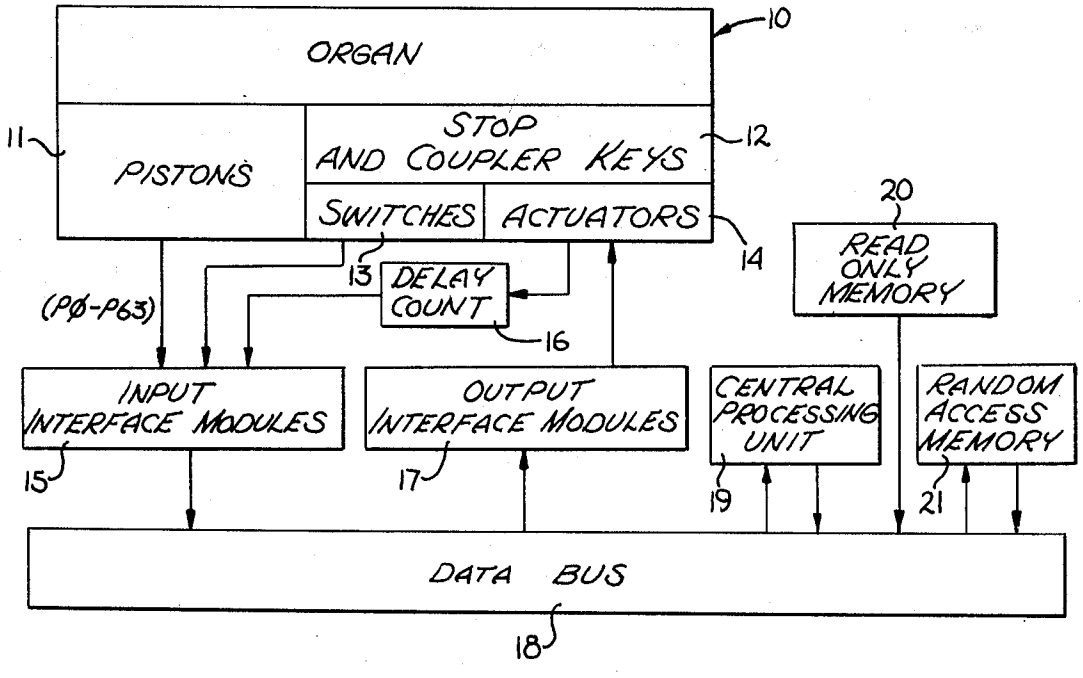
3,548,064	12/1970	Oncley	84/345 UX
3,659,488	5/1972	Deutsch	84/345
3,686,994	8/1972	Badessa	84/345

Primary Examiner—Stephen J. Tomsky
Assistant Examiner—John F. Gonzales

[57] **ABSTRACT**

A registration affecting system for electronic or pipe organs which provides for monitoring, storage and output actuation of stop and coupler key combinations selected by the operator of the organ. The translated outputs of organ pistons are input and scanned along a common data buss by a central processing unit. The central processing unit determines which registration aid function is being requested by the organ operator and causes actuation of the specific stop combination stored in a memory module. Upon a determination of the proper states of the stop and coupler keys to be actuated, the central processing unit initiates output actuation of the appropriate stop and coupler keys to select the tonal quality required.

7 Claims, 17 Drawing Figures



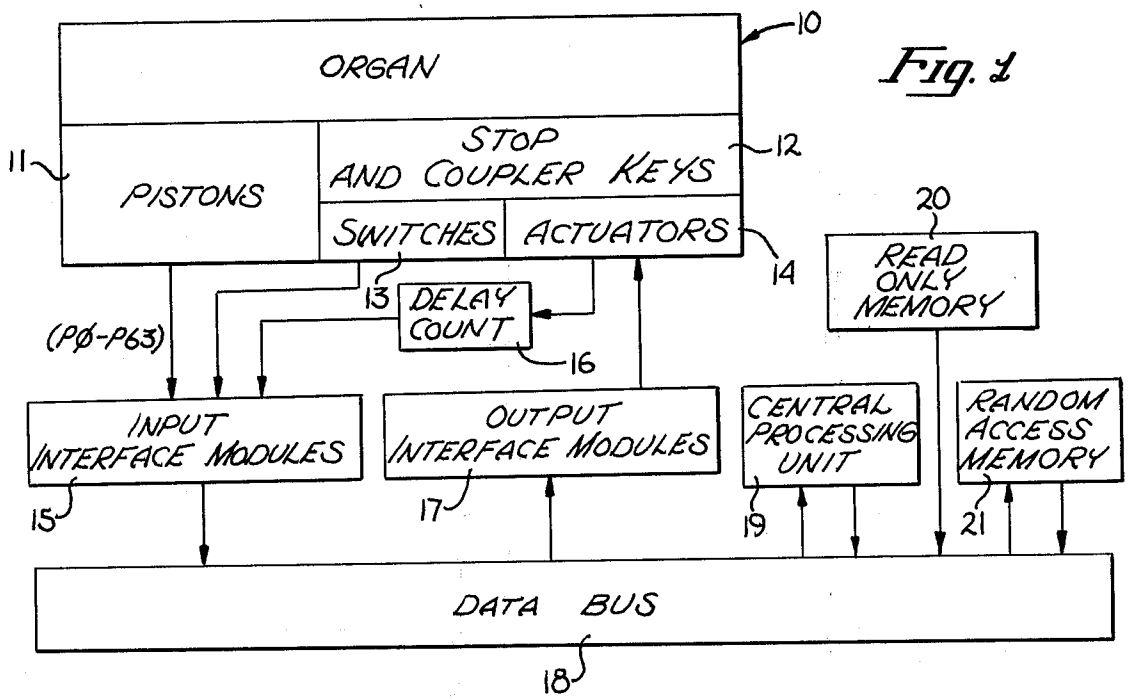

ORGAN
10
PISTONS
11
STOP AND COUPLER KEYS
12
SWITCHES
ACTUATORS
13
14
DELAY COUNT
16
(PØ-P63)
READ ONLY MEMORY
20
INPUT INTERFACE MODULES
15
OUTPUT INTERFACE MODULES
17
CENTRAL PROCESSING UNIT
19
RANDOM ACCESS MEMORY
21
DATA BUS
18

Fig. 1

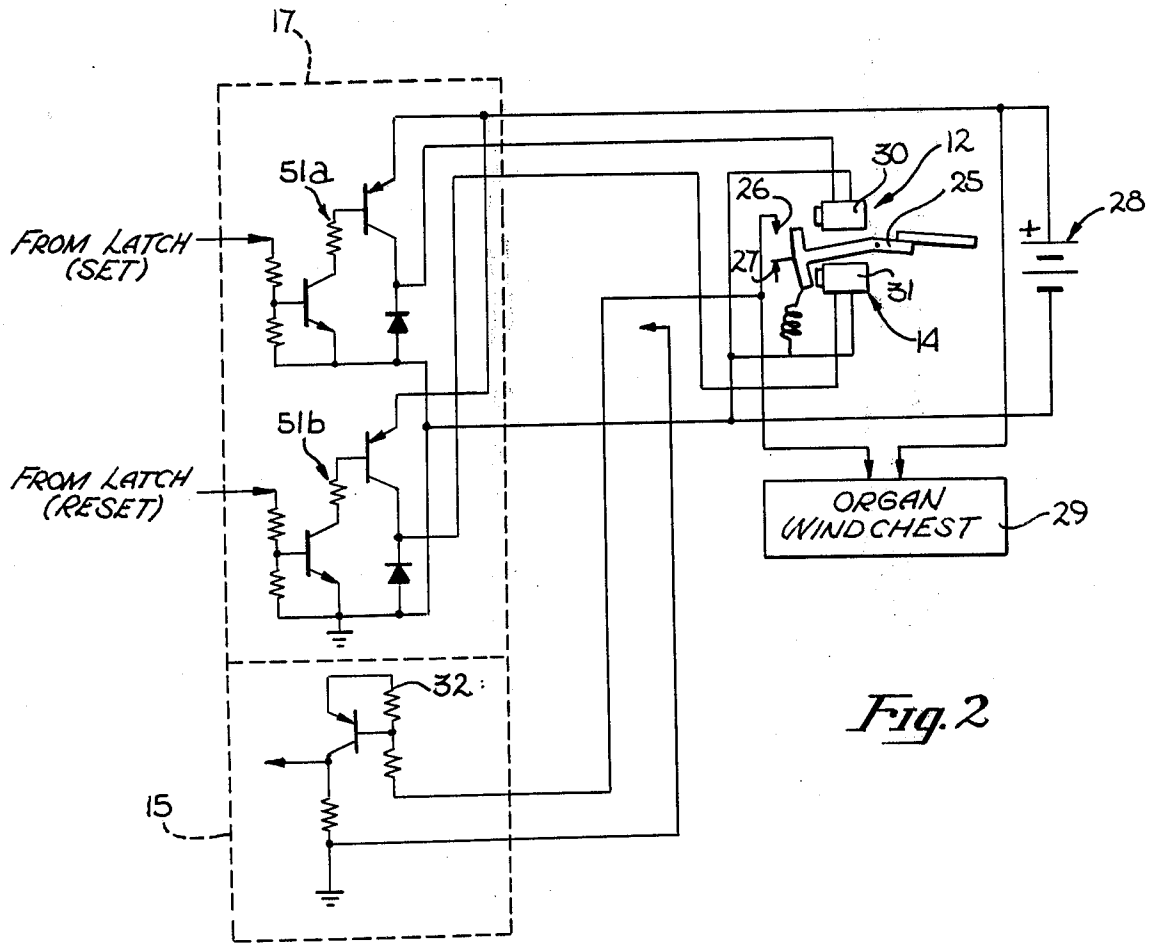

17
51a
FROM LATCH (SET)
51b
FROM LATCH (RESET)
30
12
25
26
27
31
14
28
ORGAN WINDCHEST
29
32
15

Fig. 2

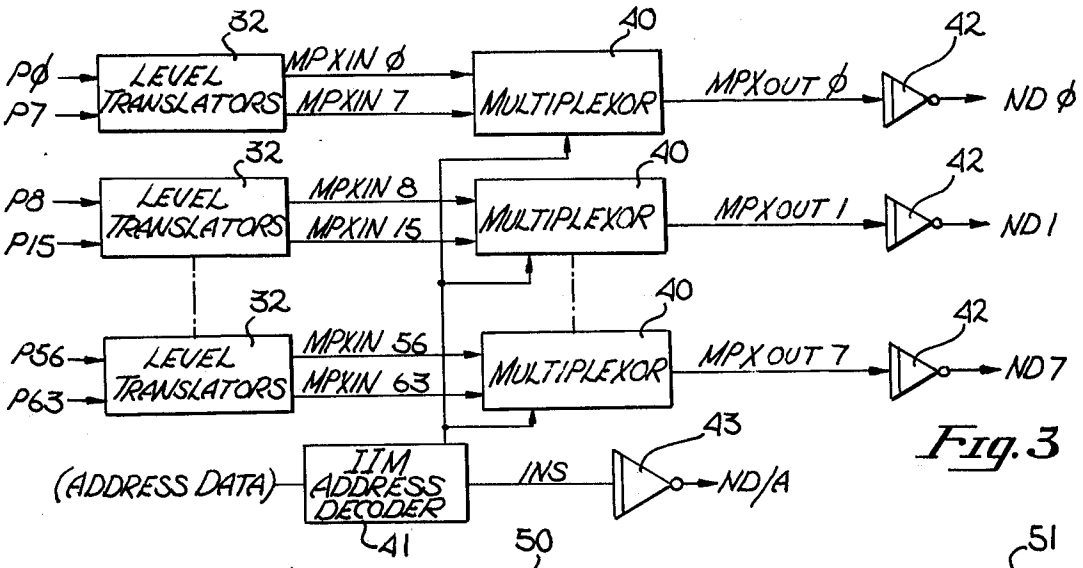
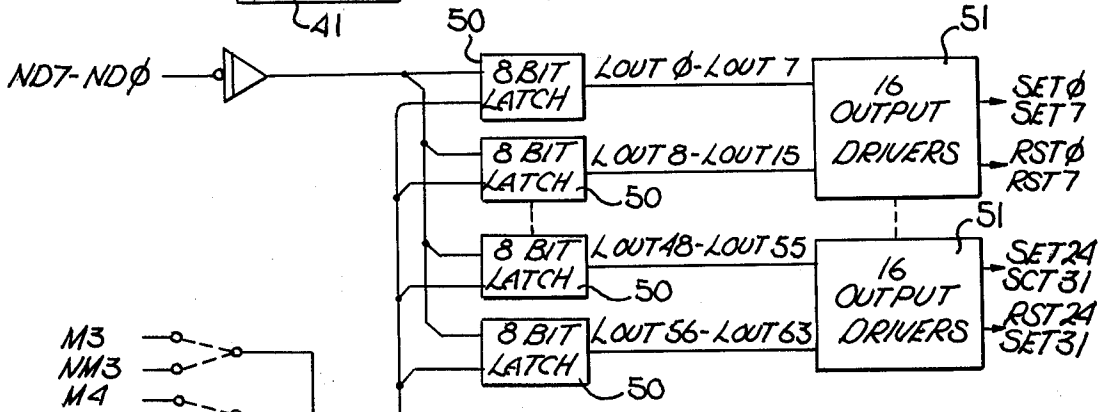


Fig. 3



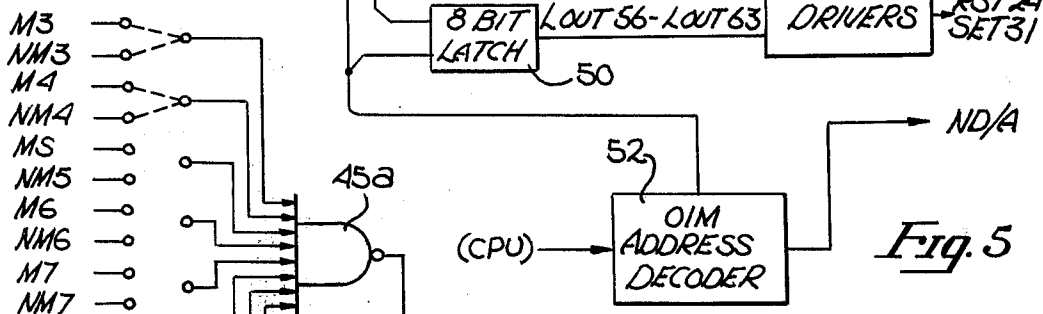


Fig. 5

Fig. 4

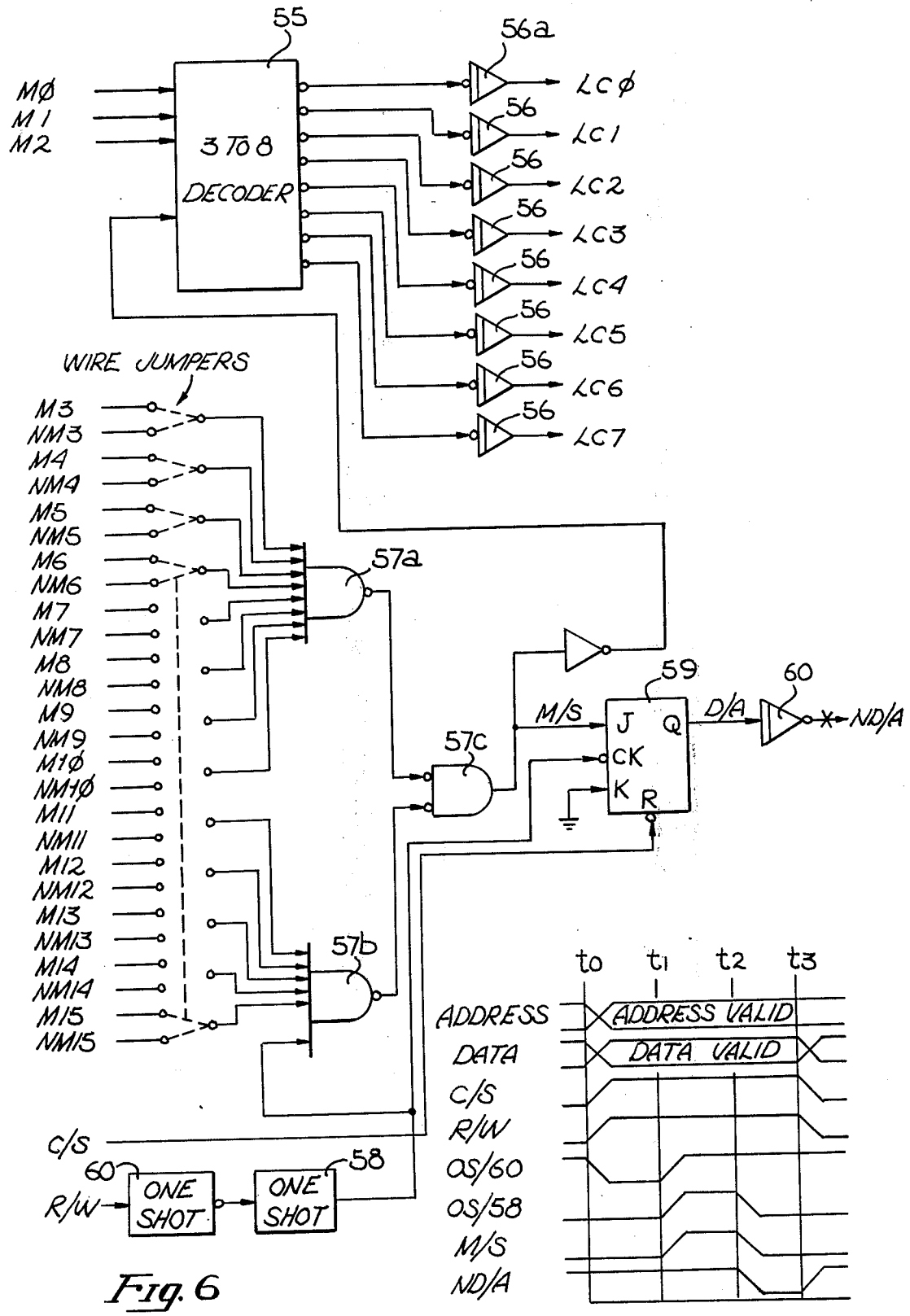

55
56a
MØ
M1
M2
3 TO 8
DECODER
LCØ
LC1
LC2
LC3
LC4
LC5
LC6
LC7
56
WIRE JUMPERS
M3
NM3
M4
NM4
M5
NM5
M6
NM6
M7
NM7
M8
NM8
M9
NM9
M1Ø
NM1Ø
M11
NM11
M12
NM12
M13
NM13
M14
NM14
M15
NM15
57a
57b
57c
59
60
M/S
J
Q
CK
K
R
D/A
N.D/A
C/S
R/W
ONE SHOT
58
to
t1
t2
t3
ADDRESS
DATA
ADDRESS VALID
DATA VALID
C/S
R/W
OS/60
OS/58
M/S
ND/A

Fig. 6

Fig. 7

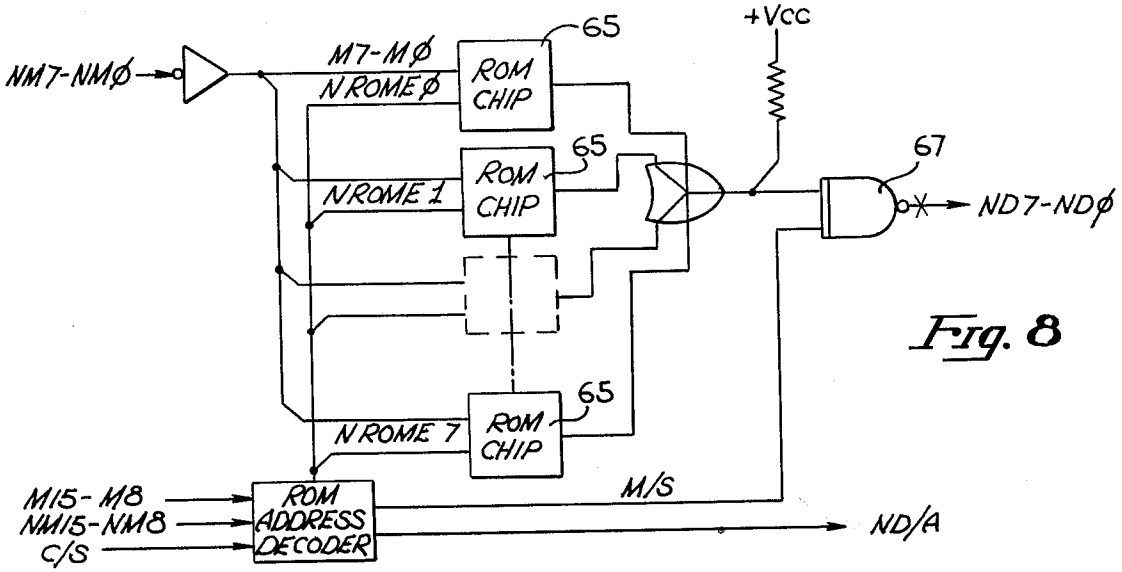


Fig. 8

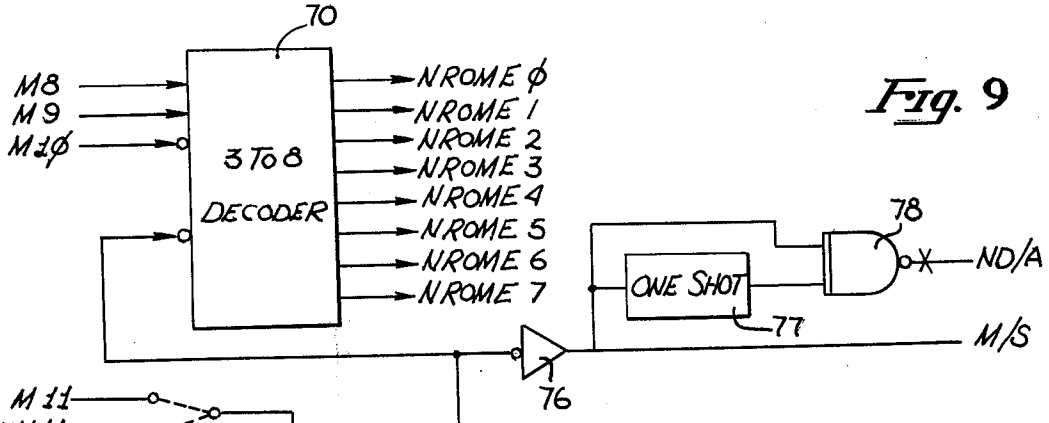


Fig. 9

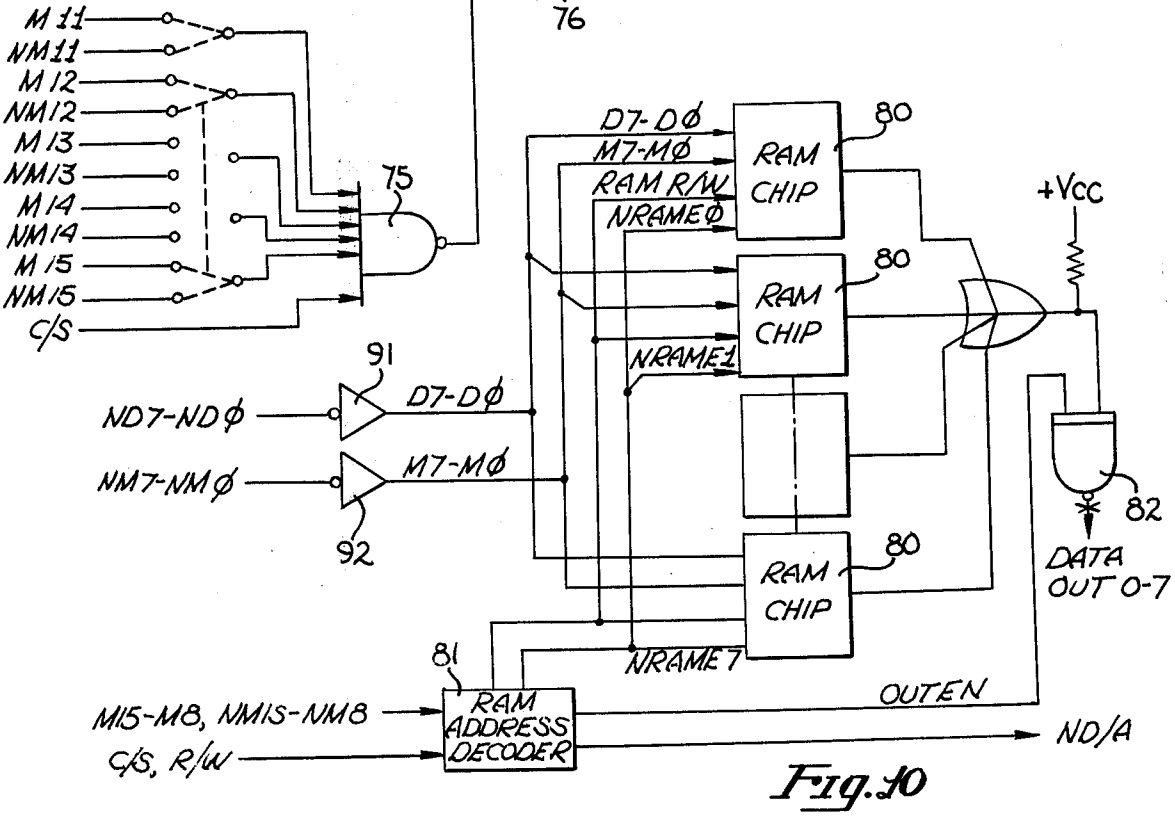


Fig. 10

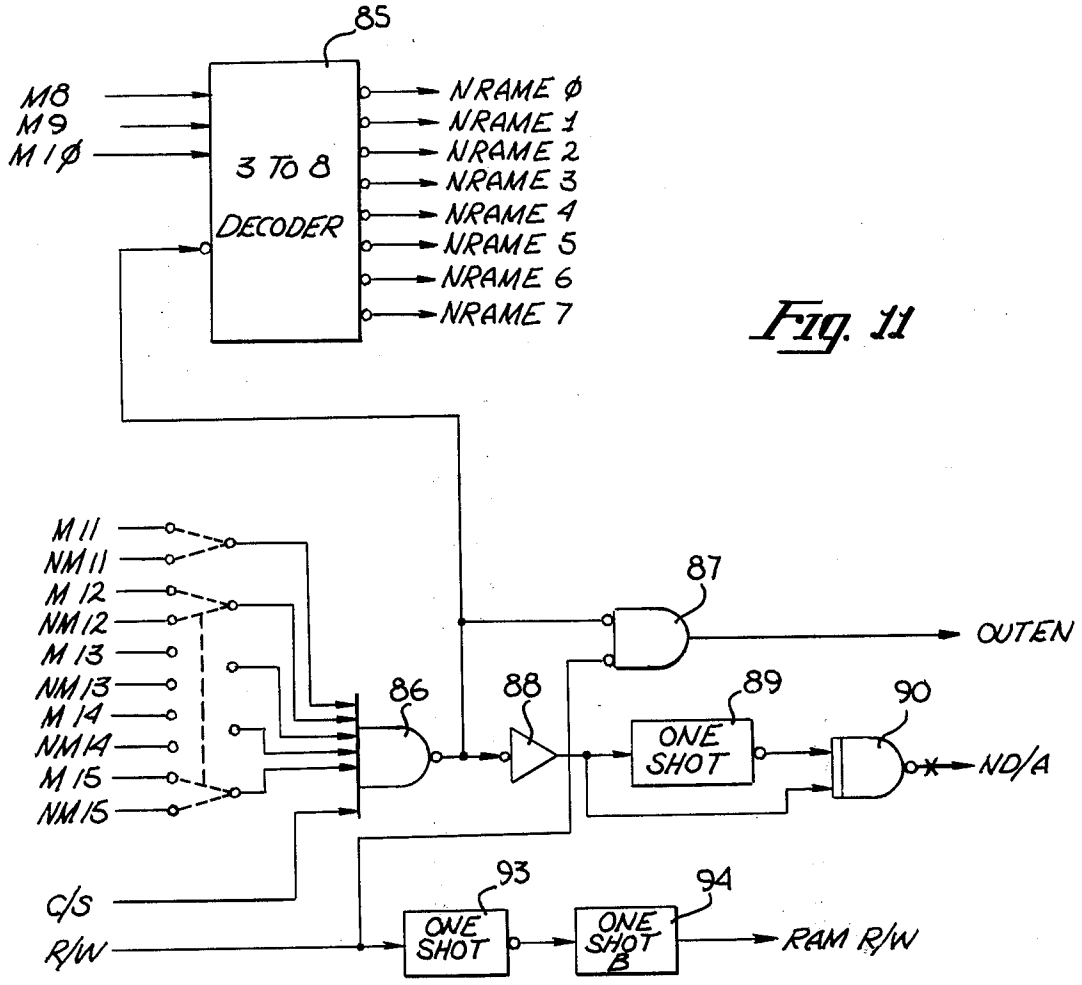

85
M8
M9
M10
3 TO 8
DECODER
NRAME 0
NRAME 1
NRAME 2
NRAME 3
NRAME 4
NRAME 5
NRAME 6
NRAME 7
M11
NM11
M12
NM12
M13
NM13
M14
NM14
M15
NM15
86
87
88
89
90
ONE SHOT
OUTEN
ND/A
C/S
R/W
93
94
ONE SHOT
ONE SHOT B
RAM R/W

Fig. 11

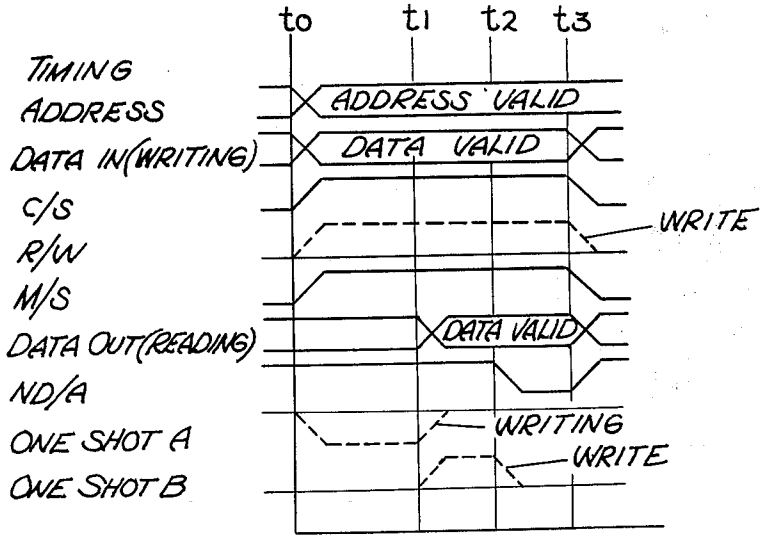

to
t1
t2
t3
TIMING
ADDRESS
DATA IN(WRITING)
C/S
R/W
M/S
DATA OUT(READING)
ND/A
ONE SHOT A
ONE SHOT B
ADDRESS VALID
DATA VALID
WRITE
DATA VALID
WRITING
WRITE

Fig. 12

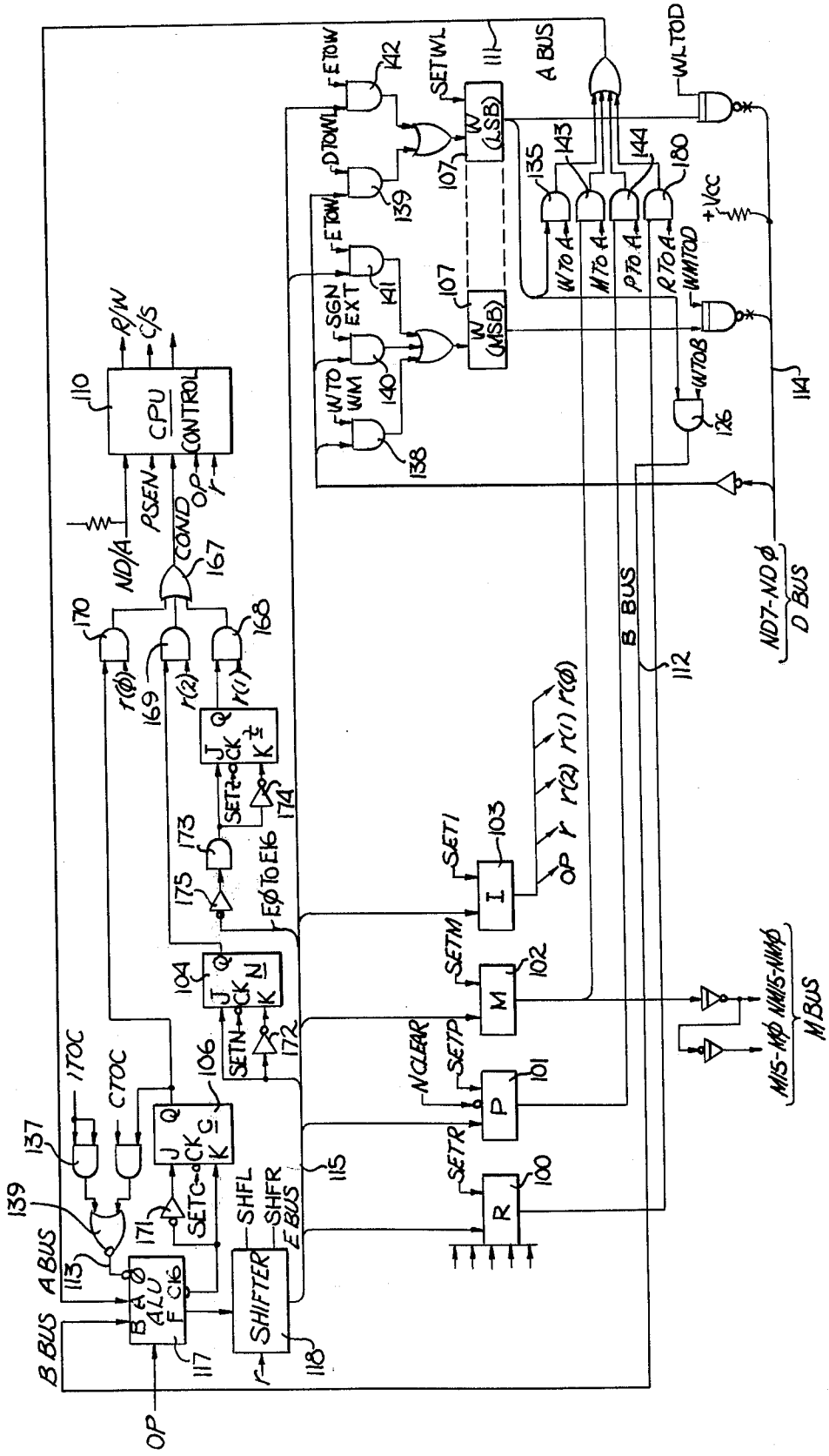

CPU CONTROL
110
R/W
C/S
PSEN
COND
OP
ND/A
167
170
168
169
r(0)
r(2)
r(1)
J Q
CK
K
SET2
174
173
175
E0 TO E16
104
SETN
172
106
1TOC
CTOC
137
139
171
SETC
SHFL
SHFR
E BUS
115
SHIFTER
118
A BUS
113
B BUS
B
A
ALU
F
C16
117
OP
SETI
I
103
SETM
M
102
SETP
P
101
SETR
R
100
NCLEAR
OP Y r(2) r(1) r(0)
B BUS
112
ETOW
142
DTOWL
139
ETOW
141
SGN EXT
140
WTO WM
138
SETWL
W (LSB)
W (MSB)
107
A BUS
111
135
143
144
180
W TO A
M TO A
P TO A
R TO A
+Vcc
WLTOD
WMTOD
WTOB
126
114
ND7-ND0
D BUS
MI5-MI0 MM15-MM0
M BUS

Fig. 13

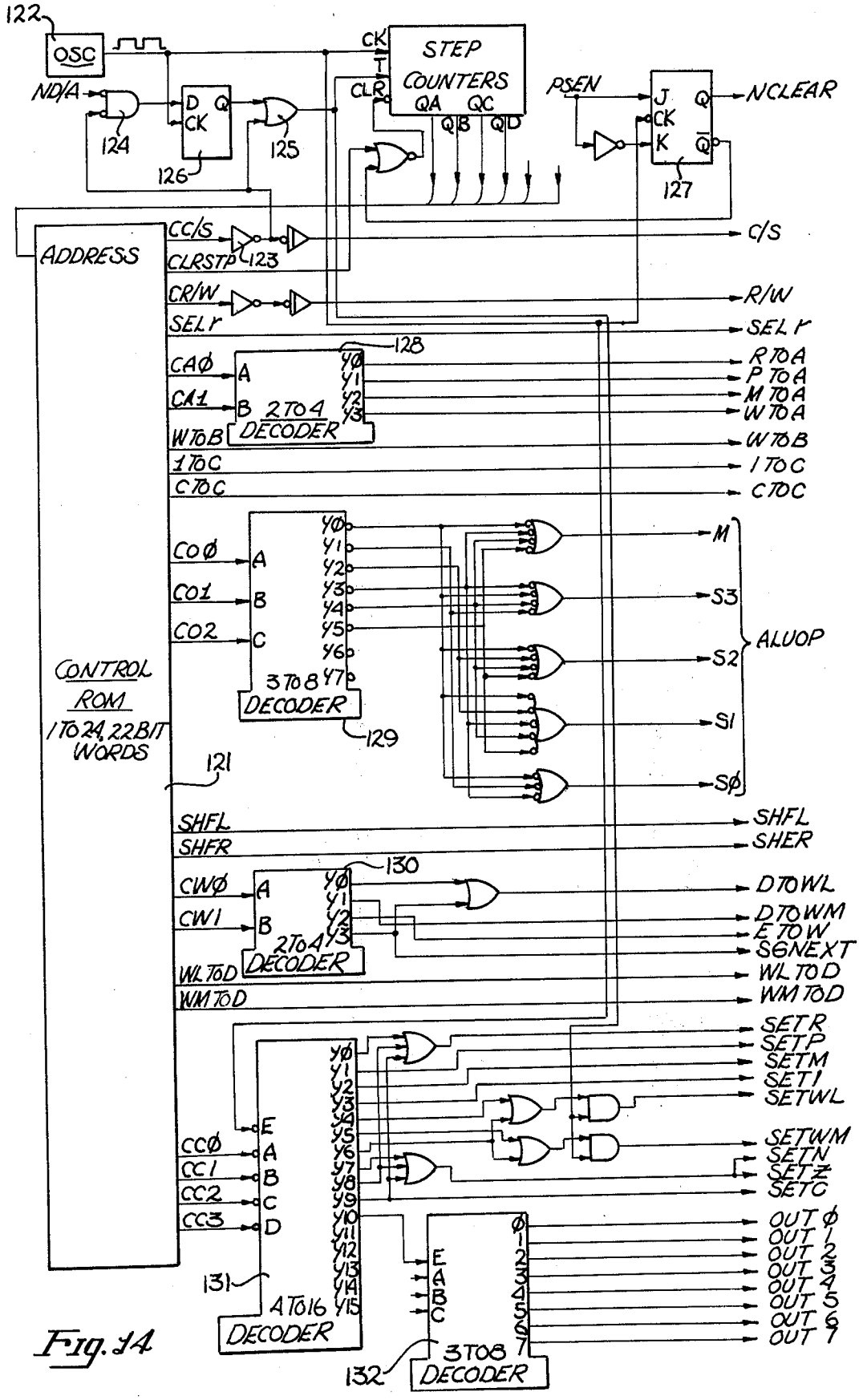

122
OSC
ND/A
124
126
125
CK
STEP
COUNTERS
QA QC
QB QD
CLR
PSEN
J Q
CK
K Q
127
NCLEAR
ADDRESS
CC/S
CLRSTP
123
CR/W
SEL r
C/S
R/W
SEL r
128
CA0
CA1
2 TO 4
DECODER
WTOB
1TOC
CTOC
R TO A
P TO A
M TO A
W TO A
W TO B
1 TO C
C TO C
CO0
CO1
CO2
3 TO 8
DECODER
129
M
S3
S2
S1
S0
ALUOP
CONTROL
ROM
1 TO 24, 22 BIT
WORDS
121
SHFL
SHFR
SHFL
SHER
130
CW0
CW1
2 TO 4
DECODER
WL TO D
WM TO D
DTOWL
DTOWM
E TO W
SGNEXT
WL TO D
WM TO D
SET R
SET P
SET M
SET I
SET WL
SET WM
SET N
SET Z
SET C
CC0
CC1
CC2
CC3
131
A TO 16
DECODER
132
3 TO 8
DECODER
OUT 0
OUT 1
OUT 2
OUT 3
OUT 4
OUT 5
OUT 6
OUT 7

Fig. 14

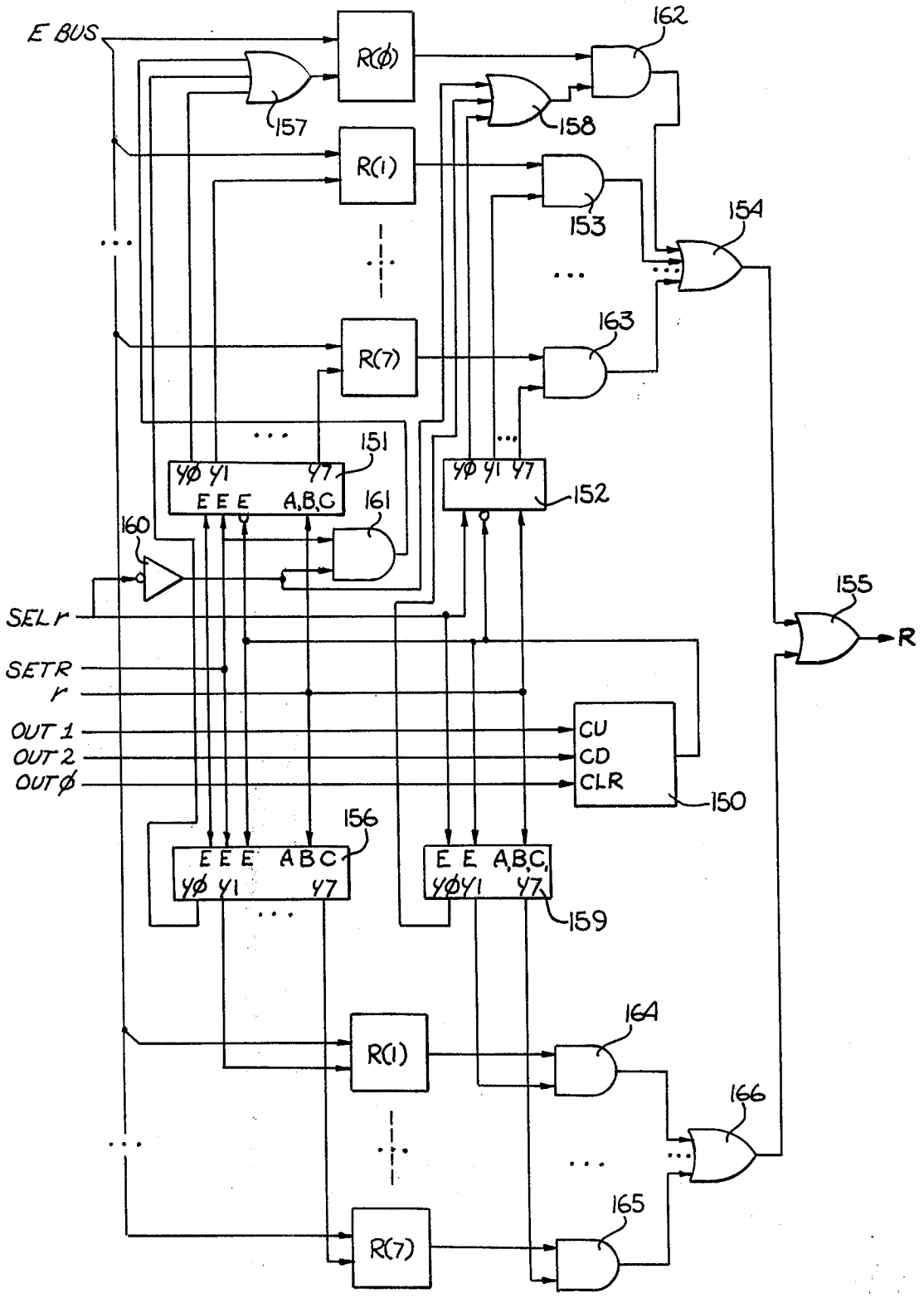

E BUS
R(Ø)
R(1)
R(7)
157
158
162
153
154
163
151
152
161
160
155
R
SELr
SETR
r
OUT 1
OUT 2
OUTØ
CU
CD
CLR
150
156
159
164
166
165
YØ Y1 Y7
E E E A,B,C

Fig. 16

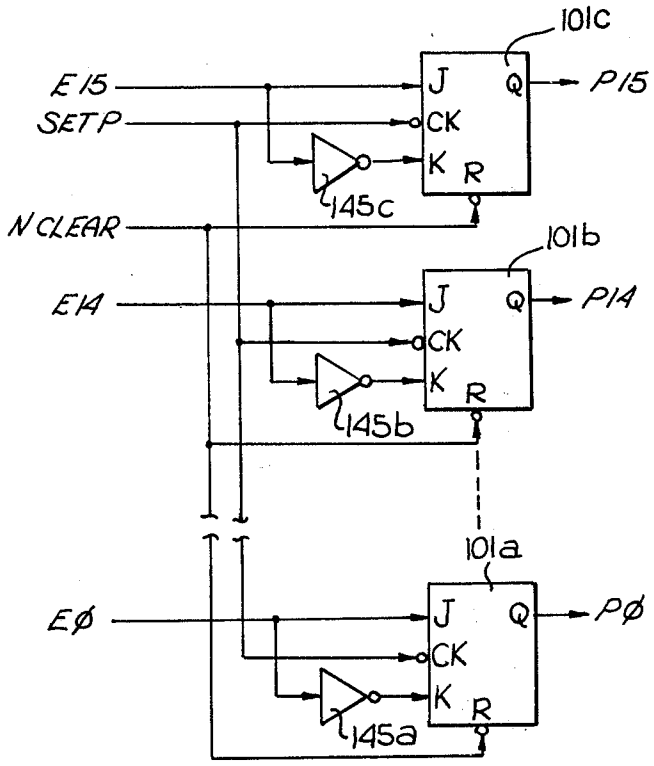

101c
E15
SETP
J
CK
K
R
Q
P15
145c
NCLEAR
101b
E14
P14
145b
101a
EØ
PØ
145a

Fig. 15

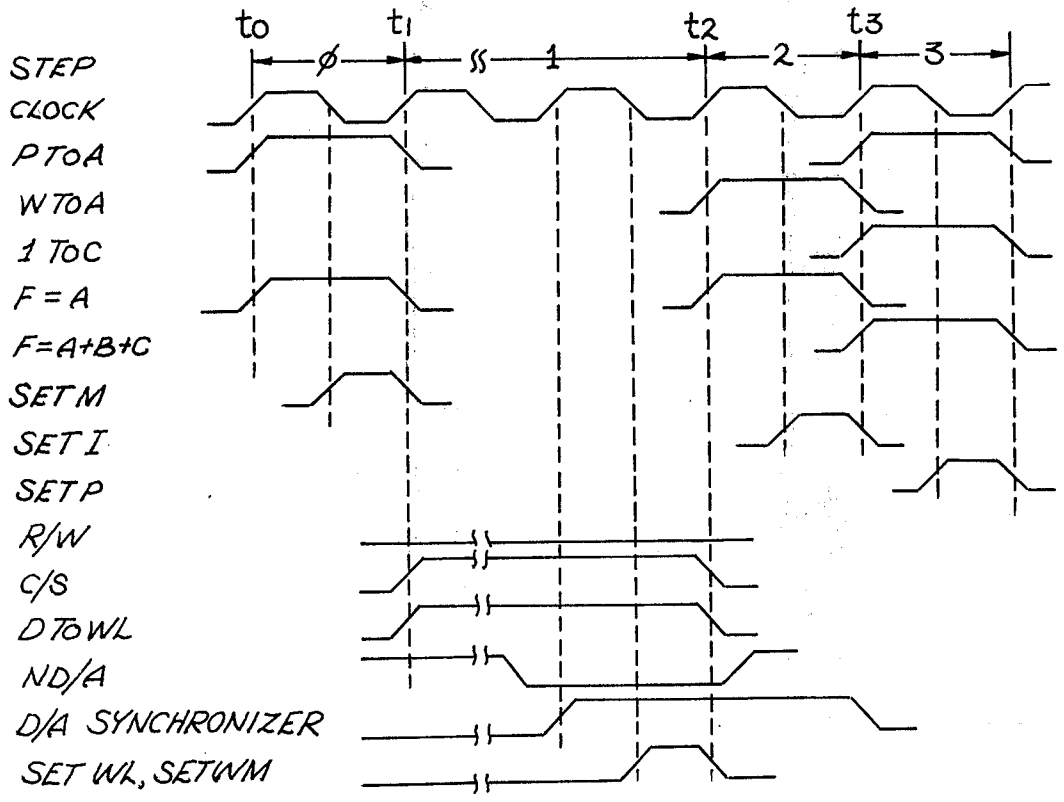

to
t1
t2
t3
Ø
1
2
3
STEP
CLOCK
PTOA
WTOA
1 TOC
F = A
F=A+B+C
SETM
SETI
SETP
R/W
C/S
DTOWL
ND/A
D/A SYNCHRONIZER
SET WL, SETWM

Fig. 17

COMPUTERIZED ORGAN REGISTRATION AFFECTING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to capture combination systems, and more particularly, to those capture combination systems usable with electronic and pipe organs which utilize digital capabilities to store and initiate actuation of appropriate stop and coupler functions.

2. Prior Art

The existence of combination systems for use with organs is well known in the prior art. The use of a combination system arises where an organist may wish to change the combination of selected stops or stop control which will thereby alter the tonal quality obtained while the organ is being played. Typically, a number of pistons or key switches are related with each division or keyboard of an organ. In a conventional organ, the pistons or key switches are located beneath each keyboard.

In a conventional organ, a piston permits actuation of a selected group of stops when that piston is actuated. Actuation of a piston permits selection of any of a number of combinations of stops in accordance with actuation of corresponding ones of the pistons. The pistons are generally arranged in categories which are identified by their control capabilities. Divisional pistons, for example, are associated with or assigned to a corresponding division and provide for control of only those voice stops and couplers associated with that respective division. Independent general pistons, on the other hand, provide for control of a combination of all of the stops provided in the organ simultaneously regardless of the relationship of groups of those stops to the specified divisions. The setting of the independent general pistons is in no way restricted by the selection of stops as established by any of the divisional pistons. Collective general pistons are related to the independent general pistons, each of the collective general pistons, when actuated, operate a corresponding piston in each group of divisional pistons.

The typical combination systems described in the prior art utilize a memory of some type for storing the combination of stops associated with each piston. Upon actuation of a piston, the combination system utilized by the prior art sets the corresponding stops as indicated in the storage medium. The prior art discloses numerous techniques to implement storage media for combination systems. The prior art systems utilize storage media such as mechanical linkages, electrical barswitches, and electronic memory systems.

The prior art discloses preset combination systems which include a storage media which has fixed wiring to select the preset storage combinations for an electronic organ. As a result, the stop combinations cannot be changed by the organist since the only way to make such changes is by rewiring the organ console. Another system discosed by the prior art is typically designated as a setter board system. In a setter board system, a switch is associated with each stop, a row of switches being provided for each divisional piston. The storage media is set by positioning the switches in the desired on or off positions. A large number of such switches are obviously required in a setter board system and accordingly, independent general pistons are typically not

provided with setter board systems, but rather only collective general pistons.

The prior art discloses combination systems which have been designated as capture combination systems, these systems permitting the organist to set information into a combination memory by means of the conventional organ controls. As an example, for the Swell division, a combination of stops can be selected which creates a tonal effect for each of the pistons of that division. The data constituting the stop combination is stored within the storage media by actuating a set piston, simultaneously actuating an available piston of that division. When the organ is being played, whenever that piston is depressed, the stops of the preselected combination which were captured in the operation with the set piston are retrieved, or set. A similar procedure is used to capture stop combinations for each divisional piston as well as the general pistons.

Although capture combinations do provide increased flexibility and ease for selecting approprite stop combination, the capture combination systems described in the prior art exhibit a number of undesirable characteristics. Where more than a single operator is using an organ, it would be typical for each operator to select a different program of stop combinations to be actuated by the pistons. Typically, the prior art capture combination systems have 25 or more pistons, each division having approximately 20 stops. Despite the apparent increased flexibility in the use with such systems, substantial delay and interruptions are often required to reset the captured combinations.

A specific capture combination system described in the prior art utilizes an internal and external memory to either manually input the selected stop combinations associated with each piston, or, through the use of a magnetic card, to input previously stored stop combinations. The difficulties inherent with such a system are substantially similar to those which are present in other capture combination systems described in the prior art. Since the settings can only be manually input, or previously stored through the use of manual selections as stored on a magnetic card, the selection of the stop combinations to be associated with each piston is again a timely operation.

The present invention substantially resolves the inadequacies existing in those devices disclosed by the prior art for determining and actuating appropriate stop combinations associated with a selected piston. Through the use of a read-only memory, no manual inputs of any kind are necessary to select the appropriate internal quality to be associated with a particular organ piston. The processing capability of what is, in effect, a general purpose computer organized to perform a specific function, permits complex selections of stop combination without the need of initially going through the manual insertion and storage procedures which are required by those systems described in the prior art.

SUMMARY OF THE INVENTION

The present invention comprises an organ registration effecting system which utilizes digital techniques. Information concerning the states of all pistons and stop and coupler keys is transmitted to input interface modules which are translated to signal levels which are compatible with the internal equipment of the present invention system. The input interface modules comprise a set of multiplexers which isolate the proper

piston data to be available on the data buss. Input data is byte oriented and will be transmitted on the data buss to the central processing unit upon demand.

The central processing unit executes a set of instructions which are stored in a read-only memory. The instruction set for the central processing unit is specifically adapted to process the data being input from a pipe or electronic organ and to set up for output those stop combinations which have been previously stored in a random access memory. The list processing techniques which are utilized by the central processing unit will correlate the multiplexed input piston data with the stored stop combinations previously input to the present invention system. To permit the present invention to operate with substantially all types of electronic and pipe organs, a delay count is input to the input interface modules to specify an optimum value for output signals. The delay count is required to allow for variations in the actuation time for stop and coupler keys. By utilizing data processing techniques for correlating piston input data and selective stop combinations which are allocated to each piston, the present invention system provides an efficient way of establishing the stop combination in accordance with the previously stored data.

It is therefore an object of the present invention to provide an improved registration affecting system for electronic and pipe organs.

It is another object of the present invention to provide a registration affecting system which can be easily used with both electronic and pipe organs.

It is still another object of the present invention to provide a registration affecting system for pipe and electronic organs which does not require the manual input of stop combination data.

It is yet another object of the present invention to provide a combination system for pipe and electronic organs which is capable of performing all registration-changing-aid functions.

The novel features which are believed to be characteristic of the invention, both as to its organization and method of operation, together with further objectives and advantages thereof, will be better understood from the following description considered in connection with the accompanying drawing in which a presently preferred embodiment of the invention is illustrated by way of example. It is to be expressly understood, however, that the drawing is for the purpose of illustration and description only, and is not intended as a definition of the limits of the invention.

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** is a schematic, block diagram of a registration affecting system for pipe and electronic organs in accordance with the present invention.

FIG. **2** is a schematic diagram of the interface between an organ stop key and the present invention registration affecting system.

FIG. **3** is a schematic, block diagram, illustrating a portion of the multiplexor interface between the organ pistons and the data buss of the present invention system.

FIG. **4** illustrates the address decoder logic for the input multiplexor shown in FIG. **3.**

FIG. **5** illustrates a schematic, block diagram of the output interface modules intermediate the present invention registration affecting system and the stop combination keys shown in FIG. **2.**

FIG. **6** illustrates the address decoder logic for the output interface module shown in FIG. **5.**

FIG. **7** illustrates the timing diagram for the output address decoder shown in FIG. **6.**

FIG. **8** illustrates a schematic, block diagram, of the read-only memory utilized by the central processing unit in accordance with the present invention.

FIG. **9** illustrates the address decoder for the read-only memory shown in FIG. **8.**

FIG. **10** illustrates a schematic, block diagram, of the random access memory modules utilized by the central processing unit in accordance with the present invention.

FIG. **11** illustrates the address decoder for the random access memory shown in FIG. **10.**

FIG. **12** illustrates the timing diagram for the address decoder shown in FIG. **11.**

FIG. **13** is a schematic, logic diagram, of the central processing unit shown in FIG. **1.**

FIG. **14** illustrates the schematic, logic diagram of the central processing unit control shown in FIG. **13.**

FIG. **15** illustrates a schematic, logic diagram, of the P register of the central processing module shown in FIG. **13.**

FIG. **16** illustrates a schematic, logic diagram, of the R registers shown in FIG. **13.**

FIG. **17** illustrates the timing diagram for the fetching of instructions from the read-only memory for the central processing unit shown in FIG. **13.**

DESCRIPTION OF THE PRESENTLY PREFERRED EMBODIMENT

Referring now to FIG. **1,** a better understanding of the interaction between the present invention organ registration affecting system and a pipe or electronic organ can be best seen, the organ being generally designated by the reference numeral **10.** As stated hereinabove, the present invention is adapted to be utilized with either an electronic or pipe organ since the interface between the organ registration affecting system and organ **10** are compatable irrespective of the type of organ being utilized. In organ **10,** pistons **11** permit actuation of a selected group of stops, couplers, or stop combinations **12** when one of the pistons **11** is actuated. Actuation of a piston **11** will permit selection of any of a number of combinations of stops **12** in accordance with a previously established program to be operated within the present invention system. The pistons **11** are generally arranged in category which are identified by their control capabilities. For the purpose of this discussion, all pistons shall be referred to by the reference numeral **11** of the specific category of the piston. Divisional pistons **11,** for example, are typically assigned to a corresponding division of organ **10** and provide for control of only those voice stops **12** and couplers associated with the respective division. Independent general pistons **11,** on the other hand, provide for control of the combination of all of the stops **12** provided in organ **10** regardless of the relationship of the groups of those stops **12** to the specified divisions. The setting of the independent general pistons **11** is in no way limited or restricted by the selection of stops **12** as established by any of the divisional pistons **11.** Collective general pistons **11** are related to the independent general pistons, each of the collective general pistons, when actuated, operating a corresponding piston **11** in each group of the divisional pistons. As shown in FIG. **1,** stop and coupler keys **12** typically comprise

members which can be manually set by switches **13** or automatically set through the use of actuators **14.** Although the implementation of switches **13** and actuators **14** is a matter of choice, a typical organ **10** will utilize a stop or coupler key **12** which has a conventional manual switch **13** and utilizes an electromagnetic relay or other like device for the implementation of actuator **14.** This shall be discussed in further detail hereinbelow.

Data available from pistons **11** are transferred to input interface modules **15.** In addition, input interface modules **15** receive data from switches **13** as well as delay count **16** which will compensate for any variation in the closure or operation of actuators **14.** Actuators **14** are coupled to output interface modules **17,** the appropriate data for selecting the stop combinations **12** being transmitted from output interface modules **17** in a manner which will be described in detail hereinbelow. The detection of when and what piston **11** has been operated is transmitted from input interface modules **15** on data bus **18,** central processing unit **19** receiving and processing all data appearing on data bus **18.** Read-only memory **20** permanently stores the special purpose program which is executed by central processing unit **19.** Data flow between central processing unit **19** and read-only memory **20** is carried out via data bus **18.** Random access memory **21** is used to store correlations between pistons **11** and the desired stop and coupler keys **12** for the combination registration function.

As a preface to the description of the logic and circuits utilized for implementation of the present invention organ registration affecting system, a description of the logic levels and circuit conventions shall be described. All signals internal to the present invention system shall be considered as true when in the high state unless the signal designation is preceded by the designation N. In the detailed description of the logic modules, the logic elements shown constitute standard symbols utilized within the computer industry, all being well known to those having skill in the computer art.

The implementation of stop and coupler keys **12** and their interface to input interface modules **15** and output interface modules **17** can be best seen by reference to FIG. **2.** Stop key **12** utilizes manual handle **25** to provide for alternate on-off positioning between contacts **26** and **27.** A conventional wiring of stop key **12** is shown including battery **28** and organ windchest **29.** As stated hereinabove, stop key **12** can be manually positioned through the use of handle **25** or can be remotely positioned through the use of actuator **14.** As shown in FIG. **2,** actuator **14** is a typical electromagnetic latching relay which can alternately be positioned by current input to the respective poles **30** and **31.** The input to actuator **14** is from the output drivers shown as part of the output interface module **17.** A set signal input to the driver circuit coupled to pole **30** will set stop key **12,** a clear signal to the circuit coupled to the pole **31** will clear stop key **12.** Contact **26** of stop key **12** is coupled to a drive circuit shown as part of input interface module **15.** The translation of the signal levels between stop key **12** and actuator **14** and the present invention system are conventional translation circuits and are used to adapt the logic levels of the present invention system to the electromagnetic or other voltage levels used within organ **10.** The circuit shown as part of input interface module **15** and output interface module **17** are conventional level translating circuits, other suitable circuits being well known to those persons having skill in in the art.

An understanding of input interface module **15** can be best seen by reference to FIG. **3.** Input interface modules **15** permit the central processing unit **19** to access the state of all input signals from organ **10.** The input interface module shown in FIG. **3** has 64 inputs which have been designated by the signal designations **P0 – P63.** The signals from pistons **11** are grouped into eight bit bytes, the total of **64** signals being grouped into eight bytes which are input into a level translator circuit **32** such as shown in FIG. **2.** The output signals from level translators **32** are designated as MPXIN-**0**–MPXIN**63** and represent the translated inputs compatible with the logic levels of the remaining portions of input interface modules **15.** The signals MPXIN-**0**–MPXIN**63** are input to conventional multiplexors **40.** In order to properly select the output signals from multiplexer **40,** interface input module address decoder **41** responds to address data transferred from central processing unit **19** and provides for a single bit to be output from multiplexors **40,** the output signals being designated as MPXOUT**0**–MPXOUT**7.** The byte of data output from multiplexors **40** are amplified and inverted by conventional inverters **42,** the data outputs being designated as ND**0**–ND**7.** The completion of the address decode function at address decoder **41** is output as signal M/S which is amplified and inverted as inverter **43** and output as signal ND/A.

As shown in FIG. **3,** the signals MPXSEL**0**–MPXSEL2 and NM/S control the operation of multiplexors **40** while the signal M/S is the source for the signal ND/A. The signal ND/A signals central processing unit **19** that the operation to read a byte of input data to central processing unit **19** has been performed. Referring now to FIG. **4,** a detailed logical schematic of input interface module address decoder **41** is shown. Since the addressing of the piston input line is to select one out of eight bits, three address bits designated as NM**0**–NM2 are utilized. The remaining address bits M3–M15 are decoded in any conventional manner and input to what is, in effect, a 14 bit AND gate designated by the reference numeral **45***a*, **45***b* and **45***c*. The output of gate **45***c* is inverted at inverter **46** thereby producing signals M/S and NM/S. As can be seen in FIG. **4,** the 14th bit is derived from signal C/S which the central processing unit **19** causes to go true when any memory operation is required. Decoding of the high order 13 address bits M3–M15 is performed by selection of either the true or complement form of each bit by appropriate decoding as shown.

Referring now to FIG. **5,** a better understanding of the output interface modules can be best gained. By proper orientation of data, and through the use of lists of pointers to stop and coupler keys **12,** central processing unit **19** may affect any single or combination of stop keys **12.** Referring now to FIG. **5,** the output signals from inverters **42** are gated to eight, eight bit latches, the output thereof being designated as LCUT-**0**–LOUT**63.** Central processing unit **19** can temporarily store in latches **50** data representing the desired states of the various keys **12.** Signals LOUT**0**–LOUT**63** are connected to output driving circuits **51***a* and **51***b* as shown in FIG. **2.** As was explained in connection with FIG. **2,** output driver **51***a* will set actuator **14** and therefore set stop key **12,** circuit **51***b* resetting actuator **14** and therefore clearing the associate stop key **12.**

In order to properly activate the appropriate stop coupler keys **12**, output interface modules **52** address the appropriate eight bit latches thereby permitting the activation of appropriate actuators **14**. The output interface module address decoder **52** must strobe data into the latches during a time when the data is stable. As soon as the data has been strobed into latches **50**, signal ND/A as shown in FIG. **5** is initiated to terminate the operation.

The operation of output interface module address decoder **52** can be best seen by reference to FIGS. **6** and **7** where address decoder **52** and the associated timing diagram are illustrated. The sixteen bit address field, i.e., **M0–M15**, is used to generate the appropriate signals needed to operate output interface module **17**. **M0-M2** define eight binary states which are needed to enable eight bit latches **50**. **M0**, M1 and M2 are input to decoder **55**, the eight binary states being amplified at amplifier **56** to produce signals LC0–LC7. As can be seen from FIG. **5**, signals LC0–LC7 are used as enable signals to latches **50**. The remaining portion of address field, i.e., M3–M15, and the negative complements thereof, are input to the effective AND gates **57***a*, **57***b* and **57***c*. Between memory operations from central processing unit **19**, signals R/W and C/S are in the false or low state. One shot multivibrator **58** will be reset thereby having the effect of holding signal M/S in the low or false state and thereby disabling decoder **55** and allowing no clock pulses to reach latches **50**. Signals C/S resets flip-flop **58** which outputs signal D/A.

An operation is initiated when central processing unit **19** provides an address field M0–M15, data and raising signals C/S and R/W. At time T0 as shown in FIG. **7**, signals C/S and R/W go to the high or true state, signal R/W triggering one shot multivibrator **60**. At time *t*1, one shot multivibrator **60** times out triggering one shot multivibrator **58** which in turn initiates signal M/S and clocks the D/A flip-flop **59** at time *t*2. Signal ND/A, is the output of inverter **60** and will be sent by central processing unit **19** which in turn resets signals C/S and R/W at time *t*3. Once signals C/S and R/W are reset, i.e., following time *t*3. the data and address may or may not change, the states thereof depending on the next program instruction to be executed.

Read-only memory module **20** provides storage for all programs and supplementary data lists required by central processing unit **19**. In a typical implementation, information is distributed over one or more read-only memory chips, the composite of the chips making up read-only memory **20**. In a typical implementation, read-only memory **20** is made of up chips each holding 256 bytes of data as shown in FIG. **8**. Any byte may be caused to appear on the data bus by applying the address field MO–M7 or the complement thereof, and the chip enable signal NROME0–NROME7. Typical ROM chips **65** have output characteristics such that the output line will not be driven to either a high or low state when the chip **65** is not in an enabled state. As a result, the output lines from each ROM chips **65** may be joined in a common connection which provides for a simple multiplexing arrangement for sending data from the enabled chip **65** to the data buss drivers.

When the output lines from ROM chips **65** are coupled together in a floating, multiplexing scheme, to prevent a floating state from appearing on the data bus, signal M/S which is output from ROM address decoder **66** is in the low or false state. The low state of signal M/S will turn gate **67** off thereby disabling the data bus drivers. A typical implementation for ROM address decoder **66** is shown in FIG. **9**. Address bits **M8–M10** are input to decoder logic **70**, the eight binary states of bits M8–M10 producing the eight output states in NROME0-NROME7. The remaining bits M11–M15, and the complements thereof, are input to gate **71** along with enabling signal C/S. The output of gate **75** is passed through inverter **76** to produce signal M/S. ROM address decoder **66** transforms the address signals into eight chip enables and signal M/S. Upon the loading of the address signals, and when signal C/S goes to the true state, signal M/S will go to the high state triggering one shot multivibrator **77**. One shot multivibrator **77** establishes the access time for ROM chips **65** and upon timing out produces output signal ND/A from gate **78**. Signal ND/A signals central processing unit **19** that the access of ROM memory **20** is complete.

Referring now to FIG. **10**, an understanding of the random access memory module **21** utilized by central processing unit **19** can be best seen. In the embodiment of the present invention shown, random access memory module **21** can store up to 2,048 bytes of random access data used for the storage of stop combinations. As with the case of read-only memory module **20**, random access memory **21** is comprised of RAM chips **80**. A typical RAM chip **80** usable to implement random access memory module **21** is organized internally as 256 one bit storage locations. Where such chips **80** are used, eight chips operate in parallel to achieve storage and retrieval of bytes of data, each byte comprising eight bits of data. Each chip has a single data input and a single output data line. Eight address lines M0–M7 select one of the 256 locations to be written into or be read out of.

Chip enable signals, NRAME0–NRAME7 provide for individual enable signals for each group of chips **80**. When a chip enable signal is in the low or false condition, a read operation is selected. As in the case with the read-only memory chips **65**, when a RAM chip **80** is not enabled the output is in a floating state. RAM address decoder **81** receives a portion of the address field, i.e., bits M8–M15, and control signals C/S and R/W. When neither a read or write operation is being performed, signal OUTEN disables gate **82** to prevent the data bus drivers from being in a floating state. As with the case of ROM address decoder **66**, RAM address decoder **81** produces signal ND/A to signal central processing unit **19** when the read/write operation is complete.

The operation of RAM address decoder **81** can be best understood by reference to FIGS. **11** and **12** wherein the logic and timing diagrams thereof are shown. The portion of the address field designated as bits M8–M10 are input to eight bit decoder **85**. The eight binary states of address bits M8–M10 are decoded to the eight individual states NRAME0–NRAME7. The remaining bits of the address field, i.e., bits M11–M15 and the complements thereof, are input to gate **86**. The output of gate **86** is also input to decoder **85**. Central processing unit **19** initiates a read operation by providing an address on lines M8–M15 and raises control signal C/S at time *t*0. During a read operation, signal R/W is in the low or false state, the high or true state of signal R/W being the write state. As stated, address bits M8–M10 are decoded thereby enabling one of the eight groups of RAM chips **80**. Address bits M0–M7 select one of 256 bytes which will appear on the chip output line after the expiration of

chip access time. Since signal R/W is in the low state, the output of gate **87** will be signal OUTEN which will be in the high state and therefore enable the data bus drivers. When signal C/S is raised to the high state at time $t0$, the output of gate **86** is inverted at inverter **88** and causes one shot multivibrator **89** to trigger. When the output of one shot multivibrator **89** times out, the output of gate **90** will be signal ND/A which will signal central processing unit **19** that the read operation has been completed at time $t2$. As can be seen from FIG. **12,** the time $t2$, the data out of gate **82** will be valid and therefore can be read on the data buss drivers.

Central processing unit **19** writes a byte of data into a RAM chip **80** by providing the byte of data at inverter **91,** address bits M0–M7 at inverter **92** and the remaining address bits at decoder **85** and gate **86.** After providing the address and data information, control signals C/S and R/W are raised to the high state at time $t0$. Raising signal R/W to the high state triggers one shot **93** which times out at time $t1$. Upon the timing out of one shot **93,** one shot **94** is triggered at time $t1$. The combination of one shot **93** and **94** produces output signal RAMR/W which will cause the data to be stored in the appropriate location of RAM chips **80.** As described hereinabove, signal ND/A signals central processing unit **19** that the read/write operation for the random access memory module **21** is complete.

All functions performed by the present invention organ registration affecting system come about as a result of central processing unit **19** executing programs stored in read-only memory module **20.** An understanding of central processing unit **19** can be best gained by reference to FIG. **13** wherein a schematic, logic diagram of central processing unit **19** can be best seen. All program accessible registers are collectively referred to as R registers **100.** All R registers **100** are two bytes wide, i.e., 16 bits, and are organized into pages of eight registers per page. At any point in time, only the R registers **100** on a current data page are accessible. For the purpose of example, the embodiment of the present invention is considered to be organized into only two pages of data, it being obvious that the number of R registers **100** can easily be expanded. R registers **100** on a page are numbered from **0–7** and are designated as R[**0**] to R[7]. On all pages, the R register **100** is designated as R[**0**] and constitutes the accumulator. The accumulator R[**0**] is also referred to as the A register. The program counter **101** is designated as the P register and is two bytes wide and will always point to the next instruction to be executed, the instructions being stored in read-only memory module **20.** Memory address register **102** is designated as the M register and is two bytes wide and constitutes the source of the addresses for input interface modules **15,** output interface modules **17** and all other modules. The instruction register **103** is designated as the I register and comprises one byte of data, I register **103** storing the instruction being executed by central processing unit **19** or the first byte of a multiple byte instruction. The N register **104,** Z register **105** and C register **106** all constitute one bit storage registers and are used for special functions. N register **104** indicates a negative arithmetic result, Z register **105** indicates a zero arithmetic result, and C register **106** indicates a carry function.

As indicated previously, R registers **100** are each two bytes wide. A necessity for a two byte register is dictated by the need to hold two byte addresses in the R registers **100** as pointers to various locations in input and output modules **15** and **17** and in memory modules **20** and **21.** In order to fill an R register **100,** the transfer of data between central processing unit **19** and interface modules **15** and **17** and memory modules **20** and **21** occur as a succession of two single byte transfers. W register **107** assembles the two bytes received from memory modules **20** and **21** into a single operand. Also, an operand is broken down into two bytes for transmission to interface module **17** and memory module **21.** Instructions are transferred from read only memory module **20** to central processing unit **19** on a byte by byte basis.

Table 1 set forth hereinbelow, sets forth the instruction set which is executable by central processing unit **19.** Table 2 illustrates the instruction format for all instructions shown in Table 1. Columns N, Z and C of Table 1 indicate when the corresponding result register, i.e., registers **104, 105** and **106,** will be affected by execution of the respective instructions.

TABLE 1

CPU INSTRUCTION SET			
MNEMONIC	FORMAT TYPE	PROCESS	N Z C
LA	1	A ← M[R[r]]	* *
LA2	1	A ← M[R[r]]; next R[r] ← R[r]+2	* *
SA	1	M[R[r]] ← A	
SA2	1	M[R[r]] ← A; next R[r] ← R[r]+2	
AA	1	A ← A+M[R[r]]	* * *
AA2	1	A ← A+M[R[r]]; next R[r] ← R[r]+2	* * *
LRI	2	R[r] ← im	
ARI	2	R[r] ← R[r]+im	* * *
LRD	3	R[r] ← M[d]	* *
SRD	3	M[d] ← R[r]	
BR	1	P ← R[r]	
BLD	3	R[r] ← P; next P ← d	
CBR	4	(COND=1) → P ← P+s	
CBD	3	(COND=1) → P ← d	
CNR	4	(COND=0) → P ← +s	
CND	3	(COND=0) → P ← d	
AD	1	A ← A+R[r]	* * *
ADC	1	A ← A+R[r]+C	* * *
SB	1	A ← A−R[r]	* * *
SBC	1	A ← A−R[r]+C	* * *
AND	1	A ← A ∧ R[r]	* *
OR	1	A ← A ∨ R[r]	* *
XOR	1	A ← A v R[r]	* *
CMPR	1	N□Z ← A−R[r]	* *

TABLE 1-continued

CPU INSTRUCTION SET			
MNEMONIC	FORMAT TYPE	PROCESS	N Z C
ADI	1	R[r] ← R[r]+1	* * *
SUI	1	R[r] ← R[r]−1	* * *
LD	1	A ← R[r]	* *
ST	1	R[r] ← A	
SLL	1	$A \leftarrow A \times 2^{R+1}$	* *
SRL	1	$A \leftarrow A \div 2^{R+1}$	* *
OUT	1	DISCRETE OUTPUT[r] ← 1	

TABLE 2

INSTRUCTION FORMAT				
TYPE	FORM			
1	OP	r		
2	OP	r	im_0	im_1
3	OP	r	d_0	d_1
4	OP	r	s	

The process column of Table I indicates the data and control operation being carried out by the specific instructions. M[ ] indicates that the data comes from, or goes to, the memory location given by the address within the brackets. *r* is three bits in the first instruction byte specifying one of the current R registers **100** or is used for other purposes. Instructions LA2, SA2, and AA2 each provide for automatic stepping of a memory pointer. Note that the pointer is incremented by two to affect data transfers involving two bytes. *im* refers to two bytes of data immediately following the first instruction byte in read-only memory **20.** *d* is an address (2 bytes) which follows the first instruction byte. COND is the logic as shown:

$$(N \wedge r[2]) \vee (Z \wedge r[1]) \vee (C \wedge r[0])$$

which is the condition on which the conditional branches are based. The term *r*[2] is the most significant bit of the R field. *s* is a short address, i.e., only one byte, and is used as a relative address. N□Z←specifies that the result of N register **104** and Z register **105** are to be set on values based on the result of the operation given to the write of the arrow. Instructions SLL and SRL are logical shift operations, i.e., the bits of the accumulator are shifted left or right with bits shifted out being lost and zeroes inserted into the opposite end of the register. The OUT instruction is a means for providing a pulse on the specified discrete output line, OUT0–OUT7. These lines can be used for a variety of purposes, in the preferred embodiment of the present invention, OUT0–OUT1 and OUT2 control the R register **100** page control hardware.

The hardware needed to implement central processing unit **19** can be best gained by reference to FIGS. **13 – 16.** Referring now to FIG. **13,** CPU control **110,** A bus **111,** B bus **112,** C bus **113,** D bus **114,** E bus **115** and M bus **116** comprise the major control and data buss members of central processing unit **19.** The other major elements of central processing unit **19** comprise registers **100, 101, 102, 103, 104, 105** and **106** which have been described hereinabove. Other major features of central processing unit **19** is the arithmetic and logic unit **117** and the shift register **118.**

The logic details of CPU control unit **110** can be best seen by reference to FIG. **14.** CPU control **110** provides all of the control signals for the remaining central processing unit hardware and creates the control signals R/W and C/S used in the operation of interface modules **15** and **17,** read-only memory **20** and random access memory **21.** The pertinent components of CPU control **110** are step counter **120** and the control read only memory **121.** The count stored in step counter **120,** the operand field of the current instruction, and the function defined as COND serve as an address to the control read-only memory **121.** Each word read from control read-only memory **121** is decoded and activates various control signals. A sequence of words from the control read-only memory **121** will result in execution of the selected instruction.

The output terminals from step counter **120** are designated as QA, QB, QC and QD with output terminal QA representing the least significant bit. Oscillator **122** provides a continuous sequence of clock pulses to input CK. On the leading edge of a clock pulse emanating from oscillator **122,** step counter **120** will be incremented by one, if the T and CLR input are high. The T input is a means for stopping step counter **120** during the time when central processing unit **19** is waiting for a D/A signal from interface modules **15** and **17,** read-only memory **20** or random access memory **21.** The circuit for implementing step counter **120** is conventional and known to those persons having skill in the computer art. When T is in the low state, the step counter **120** will be halted. A low state on input CLR will cause step counter **120** to be reset to zero on the next clock pulse eminating from oscillator **122** irrespective of the state of the signal at input P.

Signal designated as CC/S is output from control read-only memory **121.** During that portion of instruction execution where a memory cycle is required, signal CC/S will be in the high state. Signal CC/S is inverted at inverter **123** and input to gate **124.** The output of inverter **123** is also gated through OR gate **125** and causes the T input of step counter **120** to be in the low state since latch **126,** which is, in effect, the D/A synchronizer, is normally in the reset state. As a result, step counter **120** is in a waiting condition. When signal D/A occurs, in its complemented form, signal ND/A goes to the low state, the output of gate **124** will go high and on the next clock pulse from oscillator **122,** flip-flop **126** will be set. As soon as flip-flop **126** goes to the set state, the output of gate **125** will go high permitting step counter **120** to commence counting. Step counter **120** will be incremented on the next clock pulse from oscillator **122.** The next word read from control read-only memory **121** will have the signal CC/S in the low state. The T input of step counter **120** will now remain in the high state irrespective of the output of the D/A synchronizer **126.** Subsequent to signal CC/S going to the low state, the signal D/A emanating from one of the

interface modules **15** or **17** or memory modules **20** or **21** will cause the signal to go to the low state. As soon as signal D/A goes to the low state, the output of gate **124** will go low and D/A synchronizer **126** will be reset.

All instructions being executed by central processing unit **19** do not require the same number of procedural steps for execution. Signal CLRSTP emanating from control read-only memory **121** provides means to reset step counter **120** and begin execution of the next instruction. During the last word read from control read-only memory **121** of the current instructions step sequence, signal CLRSTP will go to the high state forcing the CLR input to step counter **120** to the low state. On the next clock pulse emanating from oscillator **122,** step counter **120** will be reset and commence counting again from zero.

When power is first applied to the present invention organ registration affecting system, means are required to initialize central processing unit **19** so that it commences execution of instructions at a particular location in memory, i.e., location zero. Signal PSEN and flip-flop **127** perform the reset function. Signal PSEN is a control signal derived from the power supply. Signal PSEN is derived in a conventional manner and is low for a few milliseconds after power is applied. During this delay time, the power supply for the logic has an opportunity to stabilize and oscillator **122** commences operation. After the delay, signal PSEN goes to the high state and remains high until power is removed. Flip-flop **127** will be reset after power is applied. The output from flip-flop **127,** i.e., NCLEAR, will be in the low state and cause resetting of P register **101** (FIG. **13**). The false output of flip-flop **127** will be in the high state and cause resetting of step counter **120.** Central processing unit **19** is now initialized and will commence execution after flip-flop **127** is set causing signal NCLEAR to go to the high state.

The remaining logic of CPU control **110** constitutes conventional decoders **128, 129, 130, 131** and **132** and the decoding logic associated therewith. Table **3** set forth hereinbelow sets forth the decoding of the output from control read only memory **121** and the output signals shown along the right edge of FIG. **14.**

TABLE 3

CPU CONTROL DECODING

A BUS CONTROL

ROM 121 OUTPUT		SIGNALS			
CA1	CA0	RTOA	PTOA	MTOA	WTOA
L	L	H	L	L	L
L	H	L	H	L	L
H	L	L	L	H	L
H	H	L	L	L	H

ALUOP

ROM 121 OUTPUT			SIGNALS				
CO2	CO1	CO0	M	S3	S2	S1	S0
L	L	L	H	H	H	H	H
L	L	H	L	H	L	L	H
L	H	L	L	L	H	H	L
L	H	H	H	H	L	H	H
H	L	L	H	H	H	H	L
H	L	H	H	L	H	H	L
H	H	L	L	L	L	L	L
H	H	H	L	L	L	L	L

W ENTRY CONTROL

ROM 121 OUTPUT		SIGNALS			
CW1	CW0	DTOWL	DTOWM	ETOW	SGNEXT
L	L	H	L	L	L
L	H	L	H	L	L
H	L	L	L	H	L
H	H	H	L	L	H

REGISTER CLOCKS AND OUTPUT ENABLES

ROM 121 OUTPUT				SIGNALS									
CC3	CC2	CC1	CC0	SETR	SETP	SETM	SETI	SETWL	SETWM	SETN	SETZ	SETC	OUT
L	L	L	L	H	L	L	L	L	L	L	L	L	L
L	L	L	H	L	H	L	L	L	L	L	L	L	L
L	L	H	L	L	L	H	L	L	L	L	L	L	L
L	L	H	H	L	L	L	H	L	L	L	L	L	L
L	H	L	L	L	L	L	L	H	L	L	L	L	L
L	H	L	H	L	L	L	L	L	H	L	L	L	L
L	H	H	L	L	L	L	L	H	H	L	L	L	L
L	H	H	H	L	L	L	L	L	L	H	H	L	L
H	L	L	L	H	L	L	L	L	L	H	H	L	L
H	L	L	H	H	L	L	L	L	L	H	H	H	L
H	L	H	L	L	L	L	L	L	L	L	L	L	H
H	L	H	H	L	L	L	L	L	L	L	L	L	L
H	H	L	L	L	L	L	L	L	L	L	L	L	L
H	H	L	H	L	L	L	L	L	L	L	L	L	L
H	H	H	L	L	L	L	L	L	L	L	L	L	L
H	H	H	H	L	L	L	L	L	L	L	L	L	L

As stated previously six data buses are employed in the operation of central processing unit **19.** Referring again to FIG. **13,** A and B buses **111** and **112** respectively, are two bytes wide and transfer operands from two registers to the arithmetic and logic unit **117.** Entry of an operand onto A bus **111** is governed by control signals WTOA, MTOA, PTOA and RTOA. These were

shown in Table 3 under A BUS CONTROL. B bus 112 is controlled by signal WTOB. It is to be noted that signals WTOA and WTOB give the entire sixteen bit W register 107 onto A bus 111 and B bus 112 respectively via gates 135 and 136. R register 100 is gated to A bus 111 via gate 180, the gate being enabled by signal RTOA. C bus 113 is one bit wide and is derived from flip-flop 106 and associated gates 137, 138 and 139. C bus 113 introduces a third operand into the arithmetic and logic unit 117 during specific operations, the third operand being either a carry being held in flip-flop 106 or a binary one. A byte of data arriving on D bus 114 from a interface module 15 or 17 from memory modules 20 or 21 is in inverted form, i.e., each bit is complemented, and can be gated to the most significant bit portion of W register 107 at gate 138 being enabled by signal DTOWM. The data can be gated to the least significant bit portion of W register 107 at gate 139 and using signal DTOWL.

As stated previously, one byte in the conditional branch instructions, i.e., the *s* field, is the relative address to the present contents of P register 101, the instruction branching to that address if a condition is satisfied. The relative address can be either positive or negative to allow forward or backward branching. The magnitude of the address will be restricted between 0 and 127. When the *s* byte arrives on D bus 114, it is expanded to two bytes by the control signals SGNEXT and DTOWL and gated into the sixteen bit W register 107. Signal DTOWL allows the *s* byte into the least significant half of W register 107. Signal SGNEXT enables gate 140. Signal SGNEXT and D7, the most significant bit of the *s* byte enters a byte of all zeroes, if the D7 is false, or the byte of all ones, if bit D7 is true, the byte being entered into the most significant half of W register 107. Bit D7 will be true when the *s* byte is negative in value.

E bus 115 transfers the results from shift register 118 to one of the registers 100, 101, 102, 103 or 107. Some registers only use a part of E bus 115, i.e., N register 104 and I register 103. Data on E buss 115 is available to registers 100, 101, 102, 103, 104 and 105 at all times. Entry into W register 107 from E bus 115 is under control of signal ETOW which enables gate 141 and 142 at a proper time so as not to interfere with D bus 114. M register 102 is two bytes wide, the output of M register appearing on M buss 116 as well as being gated to gate 143 and being enabled by signal MTOA. The use of bus 116 will be described in detail hereinbelow.

As stated previously P register 101 is the instruction counter. Incrementing of P register 101 is performed by gating the contents of P register 101 at gate 144 via enable signal PTOA, the contents appearing on A bus 111 being gated to the arithmetic and logic unit 117. The incrementing one is introduced on C bus 113 and arithmetic and logic unit 117 demanded to add the contents of P register 101 and the binary one. The arithmetic result appearing in the arithmetic and logic unit 117 is transferred through shift register 118, on the E bus 115 and entered back into P register 101. When an instruction is to be fetched, the contents of the P register 101 are gated through the arithmetic and logic unit 117 and shift register 118 as described hereinabove and then transferred through M register 102 to bus 116 whereupon a memory read operation is commanded. As was discussed in connection with FIGS. 2, 6, 8, 9, 10 and 12, the data on M bus 116 constitutes the memory address which is used to select the appropriate memory location in interface modules 15 or 17 or in read-only memory 20 or random access memory 21. The details of P register 101 can be best seen by reference to FIG. 15 wherein a schematic logic diagram of the sixteen flip-flops represented by the reference numerals 101*a*, 101*b* and 101*c*. The inputs to all of the flip-flops of P register 101 are derived from E bus 115, the signals appearing on the set inputs of flip-flops 101*a*, 101*b* and 101*c* being inverted through inverters 145*a*, 145*b* and 145*c* to provide a complementary input to the reset input of the respective flip-flops. Information is stored in P register 101 from E bus 115 upon the trailing edge of signal SETP. As stated hereinabove, signal NCLEAR acts as a direct reset of all flip-flops 101*a*, 101*b* and 101*c* upon the initiation of power to the present invention system.

A better understanding of the operation and implementation of R registers 100 can be best seen by reference to FIG. 16. As stated hereinabove, the embodiment of the present invention disclosed herein utilizes two pages of data. FIG. 16 illustrates two R register pages. Page 0 is represented in the upper half of FIG. 16, page 1 being represented by the lower portion of FIG. 16. It is to be noted that only a single R[0] is used. A conventional up/down binary counter 150 is provided for keeping track of the current R page. Up/down counter 150 comprises a single flip-flop since only two pages of data are represented in the present embodiment. Counter 150 would comprise additional storage elements where additional pages of data are required. Counter 150 has three inputs to implement the functions of countdown, countup and clear. The input signals to counter 150 are OUT0, OUT1 and OUT2, the signals being shown in FIG. 14. On the leading edge of a pulse from the signal OUT1, counter 150 is decremented by one; the signal OUT2 causes counter 150 to be incremented by one; the signal OUT0 resets counter 150 to zero.

When counter 150 holds a value of zero, the output thereof will provide an enable signal to decoder 152 which provides eight binary output signals for the three coded input signals collectively referred to as a *r*. When control signal SEL*r* goes true, decoder 152 is enabled and proceeds to decode the *r* field into one of the eight output signals. The one true output from decoder 152 enables an AND gate such as gate 153. In this case, the contents of the R register 100 designated as R[1] are enabled, signal being enabled through gates 153, 154 and 155. If, during the time SEL*r* is true, the signal SETR is pulsed, information on E bus 115 would be stored in R[1] since decoder 151 is enabled. In a like manner, each of the registers R[0]–R[7] may be gated to the output of gate 155 and loaded from E bus 115.

When counter 150 has a value of one, decoder 151 is disabled and decoder 156 enabled. As a result, the registers R[0]–R[7] are active. It is to be noted that register R[0] for page 1 is the same for page 0 through the use of OR gates 157 and 158. When signal SEL*r* is in the low state, all decoders 151, 152, 156 and 159 are disabled. Inverter 160 enables gates 161 and 162 with a high level thereby enabling the A register, i.e., R[0] to be read from or stored into. When gate 162 is enabled, register R[0] can be read as the output of gate 155 and be loaded from E bus 115 by signal SETR. Gate 163 operates with register R[7] in the same manner as gate 153 operates with register R[1]. Registers 164 and 165 operate with the register of page 1, gate

164 gating the output of register R[1'], gate 165 gating the output of register R[7']. OR gate 166 gates the output of the page 1 registers to gate 155.

M registers 102, I register 103 and the W register 107 are similar to P register 101 in construction and operation. Information in the one byte I register 103 is broken down into several fields designated as OP, *r*, *r*(0), *r*(1) and *r*(2). The five most significant bits are the OP field and are transmitted to CPU control 110. The remaining three bits are the *r* field and are sent as a unit to CPU control 110, shift register 118 and R registers 100. Individually, the *r* bits are used to form the signal COND which is the output of gate 167. Gates 168, 169 and 170 are the terminus of bits *r*(0), *r*(1) and *r*(2) respectively. As was stated previously, the two halves of W register 107 can be loaded individually or simultaneously. Each half of register 107 can be gated on to D bus 114 or the entire W register 107 can be gated onto A bus 111.

The arithmetic logic unit 117 can perform two arithmetic operations and four logic operations on two sixteen bit operands. As shown in FIG. 13, the arithmetic logic unit 117 has three operand inputs A and B via A bus 111 and B bus 112 respectively, and via C bus 113, and operation on the five bit OP field. The resulting output S is input to shift register 118. During arithmetic operations, a carry-in may be introduced from C register 106 on C bus 113. The carry-out from arithmetic logic unit 117 will be coupled directly to C register 106 and the input of the complementary inverter 171.

The two arithmetic operations which can be carried out by arithmetic logic unit 117 are add and subtract functions. In either case, no account is taken of a sign bit, the operands being considered to be in the absolute range of 0 to $(2^{16}-1)$. Arithmetic logic unit 117 includes a conventional binary adder and follows the conventional principals of binary addition. In the special case of incrementing an operand by one, a zero is entered at the B input, and a one is entered into the CO input. The arithmetic logic unit 117 performs the subtract operation by forming the one's complement of the B operand and then performs an addition with that which is on A bus 111. Normal subtraction can be achieved by introducing a one into input C0 along with the A and B operands. Any carry or borrow which is acquired during the addition or subtraction operation is indicated by the output appearing at terminal C16 which goes to a false level during the operation.

The four logical operations are designated as PASS, OR, AND and EXCLUSIVE OR. PASS is simply the gating of the A operand through to the S output, i.e., to shift register 118. OR, AND and EXCLUSIVE OR are conventional logic operations which are performed on a bit by bit basis utilizing the inputs from A bus 111 and B bus 112.

Shift register 118 is sixteen bits in length and holds a sixteen bit operand. Upon operation of the shift register, the output thereof constitutes a sixteen bit shifted copy of the operand. Bits in the original operand which are shifted past either the most significant or least significant bit positions are lost, zeroes filling any vacated positions. A shift operation toward the most significant end of the register is indicated by the signal SHFL, the field *r* specifying the magnitude of the shift. A shift right is designated by the signal SHFR, the *r* field designating the magnitude of the shift. If neither signal SHFL or SHFR is raised to the true state, the input operand is gated unchanged to the output thereof and onto E bus 115.

C register 106 receives the output from arithmetic logic unit 117 via the output terminal C16 of arithmetic logic unit 117. This enables the central processing unit 19 to detect overflows which occur through a conditional branch on the contents of C register 106. In addition, C register 106 stores the carry/borrow function which occurs between cycles of a programmed multiple precision add/subtract. N register 104 is clocked by signal SETN, the set input thereof being coupled to E bus 115, the reset input to N register 104 being coupled through inverter 172 to E bus 115. N register 104 will receive the value of the most significant bit of E bus 115. Z register 105 will be set to one during certain instructions if the contents of E bus 115 are all zero. Register 105 is a single flip-flop which is set by the output of gate 173 and reset through inverter 174. Inverter 175 schematically depicts sixteen inverters which are each receiving inputs from one of the sixteen lines of E bus 115. The output of all inverters 175 are and'ed at gate 173 to implement the function of Z register 105. Therefore, only when E bus 115 contains all zeroes, will register Z register 105 indicate the true state. As stated previously, the outputs from C register 106, N register 104 and Z register 105 are respectively input through gates 168, 169 and 170. The outputs thereof are logically or'ed at gate 167, the output of gate 167 comprising the signal COND. Signal COND is examined by CPU control 110 during conditional branch instructions to determine whether a branch operation is called for.

TABLE 4

	INSTRUCTION EXECUTION										
STEP	R CONTROL	A BUS GATE	B BUS GATE	C BUS GATE	ALUOP	SHIFT CONTROL	W IN CONTROL	WTOD CONTROL	REGISTER CLOCKS	STORAGE CONTROL	STEP CONTROL
INSTRUCTION READ											
0		PTOA			F=A				SETM		
1							DTOWL		SETWL,M[1]	C/S	WAIT
2		WTOA			F=A				SETI		
3		PTOA		ITOC	F=A+B+C				SETP		
LA											
4	SELr.	RTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		ITOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		WTOA			F=A				SETN,Z,R[2]		CLRSTP
LA2											
4	SELr	RTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT

TABLE 4-continued

INSTRUCTION EXECUTION											
STEP	R CON-TROL	A BUS GATE	B BUS GATE	C BUS GATE	ALUOP	SHIFT CONTROL	W IN CONTROL	WTOD CONTROL	REGISTER CLOCKS	STORAGE CONTROL	STEP CONTROL
6		MTOA		1TOC	F=A+ B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		WTOA			F=A				SETN,Z,R		
9	SELr	MTOA		1TOC	F=A+B+C				SETR		CLRSTP
SA											
4		RTOA			F=A		ETOW		SETWL,M		
5	SELr	RTOA			F=A				SETM		
6								WLTOD		C/S,W[3]	WAIT
7		MTOA		1TOC	F=A+B+C				SETM		
8								WMTOD		C/S,W	WAIT, CLRSTP
SA2											
4		RTOA			F=A		ETOW		SETWL,M		
5	SELr	RTOA			F=A				SETM		
6								WLTOD		C/S,W	WAIT
7		MTOA		1TOC	F=A+B+C				SETM		
8								WMTOD		C/S,W	WAIT
9	SELr	MTOA		1TOC	F=A+B+C				SETR		CLRSTP
AA											
4	SELr	RTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		RTOA	WTOB		F=A+B+C				SETN,Z,C,R		CLRSTP
AA2											
4	SELr	RTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		RTOA	WTOB		F=A+B+C				SETN,Z,C,R		
9	SELr	MTOA		1TOC	F=A+B+C				SETR		CLRSTP
LRI											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8	SELr	WTOA			F=A				SETR		
9		MTOA		1TOC	F=A+B+C				SETP		CLRSTP
ARI											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WATT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8	SELr	RTOA	WTOB		F=A+B+C				SETN,Z,C ,R		
9		MTOA		1TOC	F=A+B+C				SETP		CLRSTP
LRD											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		MTOA		1TOC	F=A+B+C				SETP		
9		WTOA			F=A				SETM		
10							DTOWL		SETWL	C/S	WAIT
11		MTOA		1TOC	F=A+B+C				SETM		
12							DTOWM		SETWM	C/S	WAIT
13	SELr	WTOA			F=A				SETN,Z,R		CLRSTP
SRD											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		MTOA		1TOC	F=A+B+C				SETP		
9		WTOA			F=A				SETM		
10	SELr	RTOA			F=A		ETOW		SETWL,M		
11								WLTOD		C/S,W	WAIT
12		MTOA		1TOC	F=A+B+C				SETM		
13							WMTOD			C/S,W	WAIT, CLRSTP
BR											
4	SELr	RTOA			F=A				SETP		CLRSTP
BLD											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8	SELr	MTOA		1TOC	F=A+B+C				SETR		

TABLE 4-continued

INSTRUCTION EXECUTION											
STEP	R CON-TROL	A BUS GATE	B BUS GATE	C BUS GATE	ALUOP	SHIFT CONTROL	W IN CONTROL	WTOD CONTROL	REGISTER CLOCKS	STORAGE CONTROL	STEP CONTROL
9		WTOA			F=A				SETP		CLRSTP
[CBR∧(COND=0)]∨[CNR∧(COND=1)]											
4		PTOA		1TOC	F=A+B+C				SETP		CLRSTP
[CBD∧(COND=0)]∨[CND∧(COND=1)]											
4		PTOA		1TOC	F=A+B+C				SETP		
5		PTOA		1TOC	F=A+B+C				SETP		CLRSTP
[CBR∧(COND=1)]∨[CNR∧(COND=0)]											
4		MTOA			F=A		ETOW		SETWL,M		
5		PTOA			F=A				SETM		
6		WTOA			F=A				SETP		
7							DTOWL, SGNEXT		SETWL,M	C/S	WAIT
8		PTOA	WTOB		F=A+B+C				SETP		CLRSTP
[CBD∧(COND=1)]∨]CND∧(COND=0)]											
4		PTOA			F=A				SETM		
5							DTOWL		SETWL	C/S	WAIT
6		MTOA		1TOC	F=A+B+C				SETM		
7							DTOWM		SETWM	C/S	WAIT
8		WTOA			F= A				SETP		CLRSTP
AD											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB		F=A+B+C				SETN,Z,C,R		CLRSTP
ADC											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB	CTOC	F=A+B+C				SETN,Z,C,R		CLRSTP
SB											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB	1TOC	F=A−B+C				SETN,Z,C,R	CLRSTP	
SBC											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB	CTOC	F=A−B+C				SETN,Z,C,R		CLRSTP
AND											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB		F=A B				SETN,Z,R		CLRSTP
OR											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB		F=A B				SETN,Z,R	CLRSTP	
XOR											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		RTOA	WTOB		F=A B				SETN,Z,R		CLRSTP
CMPR											
4	SELr	RTOA		1TOC	F=A		ETOW		SETWL,M		
5		RTOA	WTOB		F=A−B+C				SETN,Z		CLRSTP
AD1											
4	SELr	RTOA		1TOC	F=A+B+C				SETN,Z,C,R,		CLRSTP
SU1											
4	SELr	RTOA			F=A−B+C				SETN,Z,C,R		CLRSTP
LD											
4	SELr	RTOA			F=A		ETOW		SETWL,M		
5		WTOA			F=A				SETN,Z,R		CLRSTP
ST											
4		RTOA			F=A		ETOW		SETWL,M		
5	SELr	WTOA			F=A				SETR		CLRSTP
SLL											
4		RTOA			F=A	SHFL			SETN,Z,R,		CLRSTP
SRL											
4		RTOA			F=A	SHFR			SETN,Z,R		CLRSTP
OUT											
4									OUT[r]		CLRSTP

Notes:
[1]SETWL and SETWM
[2]SETN and SETZ and SETR
[3]C/S and Write

Table 4 illustrates the control signals which are active during all steps needed to execute instructions by central processing unit **19.** Steps 0 – 3 constitute the signals which are active during the fetching of an instruction, steps 4 and beyond constituting actual execution of the instructions shown in Table 3. In step 0, the contents of P register **101** constitute the address of the next instruction, the contents of P register **101** being gated onto A bus **111,** through arithmetic logic unit **117** and into M register **102.** Signal SETM is pulsed at the end of Step 0 loading the address into M register **102.** Referring now to FIG. **17,** the timing diagram for the reading of an instruction is shown. Times $t0$, $t1$, $t2$ and $t3$ represent the initiation of Steps 0, 1, 2 and 3 respectively. As can be seen in FIG. **17,** signal PTOA goes to the true state at time $t0$ thereby enabling the transfer of the contents of P register **101** onto A bus **111.** At time $t1$, signal C/S is raised causing the memory read operation. Signal DTOWL enables gate **139** in anticipation of receiving a byte of data on D bus **114.** Read only memory module **20** decodes the address sent from M register **102** and returns the requested byte following the required access time. As shown in connection with FIG. **8** and FIG. **9,** read only memory module **20** activates signal D/A to indicate the memory access completion thereby setting D/A synchronizer **126** (FIG. **14**). The setting of D/A synchronizer **126** enables the gates for outputting signals SETWL and SETWM to go to the true state following the setting of D/A synchronizer **126.** The byte of data is latched into the least significant half of W register **107,** all zeroes being loaded into the most significant half of W register **107.** At the initiation of Step 2, the contents of W register **107** are transferred to I register **103.** In implementing this transfer, signal WTOA goes to the true state at time $t2$ and enabling the output from register **107,** signal SETR going through to load the instruction byte into I register **103.** At time $t3$, Step **3** is initiated, and the contents of P register **101** are incremented by one. As stated previously, P register **101** is incremented by transferring the contents of P register **101** via the enabling signal PTOA to arithmetic logic unit **117** and adding a binary one to that figure. After the contents of the program counter have been incremented, the contents of the arithmetic logic unit **117** are shifted to shift register **118.** As shown in FIG. **17,** signal SETP transfers the contents of shift register **118** via E bus **115** back to P register **101.**

The instruction LA is a mnemonic for the function load A. In this instruction the contents of R register **100** is used to address memory, retrieve two bytes of data and store them in the A register, i.e., R[**0**]. In Step 4, as shown in Table **4,** the contents of the R register **100** designated as R[r] is copied to M register **102.** At Step 5, a byte of data is read from memory into the least significant half of W register **107.** In Step 6, the contents of M register **102** are incremented to point to the next byte, the contents being placed back into M register **102.** The byte of data which is designated is read from memory in Step 7 and placed in the most significant half of W register **107.** At Step 8, the contents of W register **107** are copied into the A register, i.e., R[**0**]. In addition, during Step 8, N register **104** and Z register **105** are set to states which reflect the characteristic of the data which were sent from W register **107** to the A register, i.e., R[**0**]. Upon the occurence of signals CLRSTP, instruction will be terminated and Step 0 re-entered.

Instruction LA2 is a mnemonic for the function of load A register and increment R[r] by two. Instruction LA2 is identical to LA through Step 8. In Step 9, the contents of M register **102** are incremented and sent to that portion of R register **100** designated as R[r] thereby effecting an increment of R register **100** by two.

The instruction SA is a mnemonic for the instruction store acumulator. Instruction SA copies the low order byte of the portion of R register **100** designated as R[**0**] to the memory location which is given by R register **100** R[r]. The high order byte goes to the next succeeding byte of the location pointed to by the R register **100.** During Step 4, the contents of that portion of the R register **100** designated as R[**0**] are copied into W register **107.** The contents of R register R[r] is copied into M register **102** during Step 5. During Step 6, the low order byte of W register **107** is stored in memory. During Step 7, the contents of M register **102** are incremented. During Step 8, the high order byte in W register **107** are stored in memory.

Instruction SA2 is a mnemonic for the execution of the process store accumulator and increment by two. Instruction SA2 is identical to instruction SA through Step 8. During Step 9, instruction is completed by incrementing the contents of R register R[r] by 2.

Instruction AA is a mnemonic for the arithmetic instruction add to accumulato. Instruction AA adds two bytes which are addressed by the R register **100** designated as R[r]. The contents of the memory module location which is addressed are added to the contents of the R register **100** designated as R[**0**], i.e., the accumulator the result being left in the accumulator. During Step 4, M register **102** is set to the address stored in R register R[r]. During Step 5, a byte of data is read from memory which is placed in the least significant half of W register **107.** During Step 6, the contents of M register are incremented and the next byte is read during Step 7 into the most significant half of W register **107.** During Step 8, the contents of W register **107** are added to the contents of the accumulator, the results being placed in the accumulator. As stated hereinabove, N register **104,** Z register **105** and C register **106** are set according to the appropriate arithmetic results.

Instruction AA2 is a mnemonic for the arithmetic operation of adding a sum to the contents of the accumulator and incrementing by two. Instruction AA2 is identical to AA through Step 8. Step 9 terminates instruction AA2 and completes the operation by incrementing the R register **100** designated as R[r] by two.

Instruction LRI is a mnemonic for the function load register immediate. Instruction LRI loads the second and third instruction bytes into the R register **100** designated as R[r]. The address of the second instruction byte is stored in P register **101.** Step 4 of the execution of instruction LRI copies the contents of P register **101** into M register **102.** During the execution of Step 5, the instruction byte is copied into the least significant half of W register **107.** During the execution of Step 6, the contents of M register **102** are incremented and the third instruction byte transferred to the most significant half of W register **107.** During the execution of Step 8, the contents of W register **107** are transferred to the R register **100** designated as R[r]. During the execution of Step 9, the contents of M register **102** are incremented and loaded into P register **101.** After Step 9, the contents of P register **101** point to the next instruction to be executed.

The instruction designated as ARI is a mnemonic for execution of the function add to register immediate. Instruction ARI adds the second and third instruction bytes to the accumulator, i.e., R register **100** designated as R[**0**] leaving the results stored in the accumulator. The steps of instruction ARI are identical to LRI except that during execution of Step 8, the contents of W register **107** and the contents of the accumulator are added together, the result being stored in the accumulator. In addition, N register **104,** Z register **105** and C register **106** are set to the appropriate conditions.

Instruction LRD is a mnemonic for the function of load register direct. The two bytes to be loaded into the R register **100** designated as R[*r*] are to be found at the memory location addressed by the second and third instruction byte, i.e., the direct address. During the execution of Steps 4 – 7, the address is obtained. During the execution of Step 8, the contents of P register **101** are set to point to the location of the next instruction. During the execution of Steps 9 – 12, the direct address is used to obtain the two bytes of data which are to be loaded into the R register **100** designated as R[*r*]. The loading of the two bytes of data are actually performed during Step 10. N register **104** and Z register **105** are set to the appropriate conditons.

Instruction SRD is a mnemonic for the function of the store register direct. Instruction SRD is identical to instruction LRD to the execution of Step 9. The difference between instruction SRD and LRD arise during Steps 10 – 13. During the execution of Steps 10 – 13, the two bytes stored in the R register **100** designated as R[*r*] are stored in memory at the location designated by the direct address.

Instruction BR is a mnemonic for the function branch using register. Instruction BR is an unconditional branch instruction which utilizes the contents of the R register **100** designated as R[*r*] as the address of the next instruction to be executed. During the execution of Step 4 of instruction BR, the contents of the R register **100** designated as R[*r*] are transferred to P register **101** thereby effecting the branch procedure. Instruction BR is usually used in conjunction with instruction BLD which will be discussed in detail hereinbelow.

Instruction BLD is a mnemonic for the process steps branch and link direct. Instruction BLD uses the direct address portion of the instruction field to specify the next instruction which is to be executed. In addition, the address of the instruction which follows the instruction BLD, the link, is placed in the R register **100** designated as R[*r*]. During the execution of Steps 4 – 7, inclusive, the direct address is read from read-only memory module **20** and assembled in W register **107** as is done with all other instructions utilizing direct addresses. During the execution of Step 8, the contents of M register **102** are incremented to give the address of the instruction following instruction BLD, the result being stored in the R register **100** designated as R[*r*]. During the execution of Step 9, P register **101** is set to the value of the direct address thereby completing the branch and link instruction. Under normal operation, instruction BLD is used to branch to a subroutine. The last instruction of the subroutine which is executed is the instruction BR which returns control to the main program.

The instructions designated by the mnemonics CBR, CBD, CNR and CND are all conditional branch instructions. The signal COND is the output of gate **167** (FIG. **13**) and is used as an address line to the control read only memory **121** (FIG. **14**). The occurance of the signal COND forces each conditional branch instruction into one of two step sequences depending upon its value. When instruction CBR is being executed, and where the signal COND is a zero, no branch will be made. In this case, during the execution of Step 4, the contents of P register **101** will be incremented over the *s* field and thereby point to the next sequential instruction, i.e., no branch. When instruction CNR is being executed, and when signal COND is a logical one, the contents of P register **101** are incremented to indicate the next instruction. During the execution of instruction CBD, and when signal COND is a logical zero, the contents of P register **101** are incremented during Steps 4 and 5 to point to the next instruction. This is due to the two bytes of the direct address. In the same manner as occurs during the execution of instruction CBD, during the execution of instruction CND, where the signal COND is a logical one, the contents of P register **101** are incremented to indicate the next instruction.

When the execution of instruction CBR or CNR do indicate a branch, five steps are required to effect the logical branch operation. During the execution of Step 4, the address of the branch instruction is found in M register **102,** the contents of M register **102** being transferred to W register **107** where it is temporarily stored. During the execution of Step 5, the contents of P register **101** point to the *s* field, the contents of P register **101** being copied into M register **102.** The contents of W register **107** are copied into P register **101** which thereby frees W register **107** to the receive and expand the *s* byte which is read during the execution of Step 7. During the execution of Step 8, the contents of P register **101** and W register **107** are added together thereby yielding the branch address which is then stored in P register **101.** When the execution of instruction CBD or CND result in a branch operation, five additional steps are required to complete execution of the instructions. During the execution of Steps 4 – 7, inclusive, the direct address is read and transferred into W register **107.** During Step 8, the address stored in W register **107** is copied into the program counter, i.e., P register **101,** thereby completing the branch instruction.

The instruction AD is a mnemonic for the function ADD register. Instruction AD adds together the contents of the R registers **100** designated as R[*r*] and R[O], i.e., the accumulator. During the execution of Step 4, the contents of the R register **100** designated as R[*r*] are copied into W register **107** so that during the execution of Step 5, the contents of W register **107** can be added to the contents stored in the accumulator, the results being stored in the accumulator. The contents of N register **104,** Z register **105** and C register **106** are set accordingly.

The instruction ADC is a mnemonic for the function add register with carry. Instruction ADC is similar in operation to the previously described instruction AD except that the contents of C register **106** are added in with the contents of the R register **100** designated as R[*r*] and R[**0**].

The instruction SB is a mnemonic for the function subtract register. Instruction SB is similar to the operation of instruction AD except that the B operand transferred to the accumulator is one's complemented be-

fore adding to the A operand. A one is introduced as the C operand on C bus **113** to effect addition of the two's complement of the B operand.

The instruction SBC is a mnemonic for the function subtract register with carry. Instruction SBC is similar to instruction SD except that the contents of C register **106** are introduced in place of the one as the C operand.

The logical instructions AND, OR and XOR are mnemonic for the logical functions AND, OR and EXCLUSIVE OR. During the execution of each instruction, the R register **100** designated as R[*r*] is copied into W register **107** during the execution of Step **4.** During the execution of Step 5, the A operand is operated upon. Only N register **104** and Z register **105** are affected by these instructions.

The instruction CMPR is an mnemonic for the function compare register. Instruction CMPR is similar to instruction SB but the result is not transferred to the R register **100** designated as R[**0**]. As was the case with the logic instructions described hereinabove, only N register **104** and Z register **105** are set to the appropriate values during the execution of these instructions.

The instructions AD1 and SU1 are mnemonics for the functions add 1 and subtract respectively. During the execution of both instructions, the contents of the R register **100** designated as R[*r*] is read into R[**0**]. During execution of the instruction AD1, a one is added to the contents of the R register **100** designated as R[*r*]. During the execution of the instruction SU1, a zero is subtracted from the contents of the R register **100** designated as R[*r*] which thereby reduces the value of the contents of the R register **100** designated as R[*r*] by one. After performing the arithmetic operation, the result is sent back to and stored in the R register **100** designated as R[*r*] simultaneously with the setting of N register **104,** Z register **105** and C register **106** to the appropriate values.

The instructions LD and ST are mnemonics for the functions load and store register respectively. During the execution of the LD instruction, the W register **107** is used to temporarily store the contents of the R register **100** designated as R[*r*]. During the execution of instruction ST, the W register **107** is used to temporarily store the contents of the accumulator. This occurs during the execution of Step 4. During the execution of Step 5, the instruction LD causes the data to be transferred to the accumulator. During the execution of instruction ST, the data is transferred to the R register **100** designated as R[*r*] during the execution of Step 5. Execution of instruction LD causes N register **104** and Z register **105** to be set according to the contents of the R register **100** designated as R[*r*].

The instruction SLL and SRL are mnemonics for logical shift operations. During the execution of these instructions, the contents of the accumulator are transferred to shift register **118** where the contents thereof are shifted left or right in the number of locations to be shifted is given by the quantity *r*. The contents of N register **104** and Z register **105** are set accordingly.

The instruction OUT is a mnemonic for the function discrete output. When instruction OUT is being executed, a pulse which is equivalent to the low portion of a clock cycle is transmitted on a discrete output line, the output line being designated by the quantity *r*.

The prior discussion has dealt with the physical hardware utilized to implement the present invention, including the input interface modules **15,** output interface modules **17,** central processing unit **19,** read-only memory **20** and random access memory **21.** The following discussion describes the overall operation of the present invention organ registration affecting system which is generally designated in FIG. **1.** The present invention combines the operation of the organ **10** with the computerized system which has been described in detail. Initially, the details of organ **10** as it relates to the operation of the present invention system is described, the details given relating organ **10** to the input interface modules **15** and the output interface modules **17.** Finally, the operation of central processing unit **19** and the implementation of the setting of appropriate stop combinations **12** as a result of input indicia from pistons **11** are described.

Organ **10** is one which typically has four divisions which are in effect funtionally separate suborgans, the divisions being called the Pedal, Great, Swell and Choir organs. Table 5 displays the number of the stop and coupler keys **12** within each division. Each key **12** is referred to by division and number within the division rather than by the actual name of the key. As an example, the Pedal organ stop keys **12** will be called pedal stop **1,** pedal stop **2** etc. Pedal stop **13** is the last of the pedal stop keys **12.** The coupler keys **12** are called pedal coupler **1,** pedal coupler **2,** etc.

As described previously each division of organ **10** has combination action pistons **11,** the number typically being **12.** The combination action pistons **11** are conventionally called combons. A divisional combon affects a combination on only its division stop and coupler keys **12.** Twelve coupler combons affect all coupler keys **12** regardless of division boundaries. Twelve independent general combons affect all stop and coupler keys **12.** Each of the twelve collective general combons causes activation of all correspondingly numbered divisional combons. No combinations can be stored on a collective general, they can only be read. To indicate a combination is to be stored, and not read, with the pressing of any other type of combon, a setter piston **11** is provided.

Other registration-changing-aids are divisional cancels. Each divisional cancel clears all stop and coupler keys **12** of the associated division. A coupler **12** and general cancel are also provided. Another piston **11** is the swell tremolo off which cancells swell stop **16** (the tremolo). A double off piston cancels pedal stops **1, 2** and **9,** swell stops **1** and **12** (the **16'** pitch which is double the usual **8'**). The last piston **11** is the great coupler **1** reversable which alternates the position of the great coupler key **1** with each pressing.

TABLE 5

STOPS, COUPLERS AND PISTONS OF THE ORGAN CONSOLE

Pedal Organ Stops, Couplers

Stop Number[1]	Stop Name	Pitch[2]
1	Principal	16'
2	Gedeckt	16'
3	Octave	8'
4	Hohlflote	8'
5	Rohrflote	8'
6	Choral Bass	4'
7	Rohrflote	4'
8	Mixture	III
9	Fagotto	16'
10	Trumpet	8'
11	Fagotto	8'
12	Krummhorn	4'
13	Fagotto	4'

Number of couplers--6

TABLE 5-continued

STOPS, COUPLERS AND PISTONS OF THE ORGAN CONSOLE

Great Organ Stops, Couplers

Stop Number	Stop Name	Pitch
1	Principal	8′
2	Hohlflote	8′
3	Octave	4′
4	Gedeckt	4′
5	Fifteenth	2′
6	Fourniture	III
7	Trumpet	8′
Number of couplers—9		

Swell Organ Stops, Couplers

Stop Number	Stop Name	Pitch
1	Rohrflote	16′
2	Rohrflote	8′
3	Viola	8′
4	Viole Celeste	8′
5	Principal	4′
6	Rohrflote	4′
7	Nasard	2 2/3′
8	Blockflote	2′
9	Tierce	1 3/5′
10	Larigot	1 1/3′
11	Mixture	II
12	Fagotto	16′
13	Trumpet	8′
14	Fagotto	8′
15	Clarion	4′
16	Tremolo[3]	
Number of couplers—3		

Choir Organ Stops, Couplers

Stop Number	Stop Name	Pitch
1	I	8′
2	Gedeckt	8′
3	Viola	8′
4	Voce Umana	8′
5	VIII	4′
6	XV	2′
7	Hohlflote	2′
8	XIX	1 1/3′
9	XXII	1′
10	XXVI	2/3′
11	XXIX	1/2′
12	XXXIII	1/3′
13	Ripieno	V
14	Krummhorn	8′
15	Tremolo	
Number of couplers—6		

Registration Changing Aids

Type	Number of Pistons
Pedal combon[4]	12
Great combon	12
Swell combon	12
Choir combon	12
Coupler combon[5]	12
Coll Gen combon[6]	12
Indep Gen combon[7]	12
Pedal Cancel	1
Great Cancel	1
Swell Cancel	1
Choir Cancel	1

Pedal Organ Stops, Couplers

Stop Number[1]	Stop Name	Pitch[2]
Coupler Cancel		1
Swell Tremolo Off		1
Doubles Off[8]		1
General Cancel		1
Great Rever[9]		1
Setter		1

Notes

[1]Stop number is for reference only

[2]Given under pitch is the length of the lowest pitch pipe of the stop or a Roman numerial which is a code for various pitches which are sounded simultaneously

[3]Tremolo is a tonal special effect

[4]Combon is short for combination button (or piston)

[5]Coupler combons affect only the couplers

[6]Coll Gen is short for Collective General. Activation of a Coll Gen combon activates corresponding Pedal, Great, Swell, and Choir combons

[7]Indep Gen is short for Independent General. Indep Gen combons affect all stops in the organ regardless of membership in any of the four divisions (Pedal, Great, Swell, and Choir)

[8]Cancels all stops of the 16′ pitch

[9]Each pressing of the Great Rever (Reversible) piston causes Great division coupler key 1 to change position

In the embodiment of the present invention described herein, four input interface modules **15** and three output interface modules **17** are required to interface the stop and coupler keys **12** and pistons **11** described in connection with the organ disclosed in Table 5. Each of the modules **15** and **17** has its address decoder wired to be responsive to the addresses set forth hereinbelow: **8n**

IIM module	1	Addresses 2048	through 2055
	2	2056	2063
	3	2064	2071
	4	2072	2079
OIM module	1	2560	2567
	2	2568	2575
	3	2576	2583

Table 6 and Table 7 indicate the correlation between each signal which emanates from or is transferred to the console of organ **10** and the corresponding bits of the words in the input interface module **15** and output interface module **17** respectively. It will be recalled that a word is two bytes wide, the word being the net unit of information which is transferred beteen modules **15** and **17** and central processing unit **19.** The convention used to number the bits of a word is such that bits zero through **7** of a word correspond to bits zero through **7** of the low order byte. Bits **8** – **15** of a word correspond to bits zero through **7** of the high order byte. A word can comprise any of two adjacent bites of data.

TABLE 6

IIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Input	
2048	15	Pedal Combon	1
	14		2
	13		3
	12		4
	11		5
	10		6
	9		7
	8		8
	7		9
	6		10
	5		11
	4		12
	3	Great Combon	1
	2		2
	1		3
	0		4

TABLE 6-continued

IIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Input	
2050	15		5
	14		6
	13		7
	12		8
	11		9
	10		10
	9		11
	8		12
	7	Swell Combon	1
	6		2
	5		3
	4		4
	3		5
	2		6
	1		7
	0		8
2052	15		9
	14		10
	13		11
	12		12
	11	Choir Combon	1
	10		2
	9		3
	8		4
	7		5
	6		6
	5		7
	4		8
	3		9
	2		10
	1		11
	0		12
2054	15	Coupler Combon	1
	14		2
	13		3
	12		4
	11		5
	10		6
	9		7
	8		8
	7		9
	6		10
	5		11
	4		12
	3	Collective General Combon	1
	2		2
	1		3
	0		4
2056	15		5
	14		6
	13		7
	12		8
	11		9
	10		10
	9		11
	8		12
	7	Independent General Combon	1
	6		2
	5		3
	4		4
	3		5
	2		6
	1		7
	0		8
2058	15		9
	14		10
	13		11
	12		12
	11	Pedal Cancel	
	10	Great Cancel	
	9	Swell Cancel	
	8	Choir Cancel	
	7	Coupler Cancel	
	6	Swell Tremolo Off	
	5	Doubles Off	
	4	General Cancel	
	3	Great Reversible	
2060	0	Pedal Stop	1
	1		2
	2		3
	3		4
	4		5
	5		6
	6		7
	7		8
	8		9
	9		10
	10		11
	11		12

TABLE 6-continued

IIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Input	
	12		13
	13	Pedal Coupler	1
	14		2
	15		3
2062	0		4
	1		5
	2		6
	3	Great Stop	1
	4		2
	5		3
	6		4
	7		5
	8		6
	9		7
	10	Great Coupler	1
	11		2
	12		3
	13		4
	14		5
	15		6
2064	0		7
	1		8
	2		9
	3	Swell Stop	1
	4		2
	5		3
	6		4
	7		5
	8		6
	9		7
	10		8
	11		9
	12		10
	13		11
	14		12
	15		13
2066	0		14
	1		15
	2		16
	3	Swell Coupler	1
	4		2
	5		3
	6	Choir Stop	1
	7		2
	8		3
	9		4
	10		5
	11		6
	12		7
	13		8
	14		9
	15		10
2068	0		11
	1		12
	2		13
	3		14
	4		15
	5	Choir Coupler	1
	6		2
	7		3
	8		4
	9		5
	10		6
	11	Setter	
2070	0	Outer Delay Count Bit	0
	...	...	
	15		15
2072	0	Inner Delay Count Bit	0
	...	...	
	15		15

TABLE 7

OIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Output	
2560	0	Pedal Stop	1 Clear
	1		2
	2		3
	3		4
	4		5
	5		6
	6		7
	7		8
	8	Pedal Stop	1 Set

TABLE 7-continued

OIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Output
	9	2
	10	3
	11	4
	12	5
	13	6
	14	7
	15	8
2562	0	Pedal Stop 9 Clear
	1	10
	2	11
	3	12
	4	13
	5	Pedal Coupler 1 Clear
	6	2
	7	3
	8	Pedal Stop 9 Set
	9	10
	10	11
	11	12
	12	13
	13	Pedal Coupler 1 Set
	14	2
	15	3
2564	0	Pedal Coupler 4 Clear
	1	5
	2	6
	3	Great Stop 1 Clear
	4	2
	5	3
	6	4
	7	5
	8	Pedal Coupler 4 Set
	9	5
	10	6
	11	Great Stop 1 Set
	12	2
	13	3
	14	4
	15	5
2566	0	Great Stop 6 Clear
	1	7
	2	Great Coupler 1 Clear
	3	2
	4	3
	5	4
	6	5
	7	6
	8	Great Stop 6 Set
	9	7
	10	Great Coupler 1 Set
	11	2
	12	3
	13	4
	14	5
	15	6
2568	0	Great Coupler 7 Clear
	1	8
	2	9
	3	Swell Stop 1 Clear
	4	2
	5	3
	6	4
	7	5
	8	Great Coupler 7 Set
	9	8
	10	9
	11	Swell Stop 1 Set
	12	2
	13	3
	14	4
	15	5
2570	0	Swell Stop 6 Clear
	1	7
	2	8
	3	9
	4	10
	5	11
	6	12
	7	13
	8	Swell Stop 6 Set
	9	7
	10	8
	11	9
	12	10
	13	11
	14	12
	15	13
2572	0	Swell Stop 14 Clear
	1	15

TABLE 7-continued

OIM ADDRESS ASSIGNMENTS

Word Address	Bit Position	Output
	2	16
	3	Swell Coupler 1 Clear
	4	2
	5	3
	6	Choir Stop 1 Clear
	7	2
	8	Swell Stop 14 Set
	9	15
	10	16
	11	Swell Coupler 1 Set
	12	2
	13	3
	14	Choir Stop 1 Set
	15	2
2574	0	Choir Stop 3 Clear
	1	4
	2	5
	3	6
	4	7
	5	8
	6	9
	7	10
	8	Choir Stop 3 Set
	9	4
	10	5
	11	6
	12	7
	13	8
	14	9
	15	10
2576	0	Choir Stop 11 Clear
	1	12
	2	13
	3	14
	4	15
	5	Choir Coupler 1 Clear
	6	2
	7	3
	8	Choir Stop 11 Set
	9	12
	10	13
	11	14
	12	15
	13	Choir Coupler 1 Set
	14	2
	15	3
2578	0	Choir Coupler 4 Clear
	1	5
	2	6
	8	Choir Coupler 4 Set
	9	5
	10	6

The assignment of pistons **11** to input interface module **15** words begins at location **2048**, bit **15** and continues sequentially to location **2058**, bit **3**. Bit **15** of a word follows bit **0** of the previous word. Stop and coupler key **12** assignments begin at location **2060**, bit **0** and proceeds in ascending bit order. In this case, bit **0** follows bit **15**. At the end of the assignment of stop and coupler keys **12**, setter piston **11** assignment must be made. The following two words provide the delay count needed to account for actuation times of actuated **14** as has been described previously.

The assignment of output interface modules **17** include the additional complication that two lines are required for each stop and coupler key **12**, the set and reset lines associated with actuators **14**. In addition, assignments of output interface module **17** must have a specific relationship to the assignment of the stop and coupler keys **12** to the input interface modules **15**. As an example, pedal stops **1 – 8**, inclusive, provide information on their state, i.e, set or reset, through bits **0 – 7** of word **2060**. The stops are cleared by means of bits **0 – 7** of word **2560** and set by bits **8 – 15**. Pedal stops **9 – 13**, inclusive, and couplers **1 – 3** provide their states in bits **8 – 15** of word **2060**. These stop and coupler

keys **12** are controlled in word **2562.** It should be noted that the relationship between the assignments of input interface modules **15** and output interface modules **17** is not an inherent restriction of the present invention organ regsitration affecting system but has been a selection of choice to simplify the programming of central processing unit **19.**

As stated previously, the central processing unit **19** of the present invention organ registration affecting system executes the following functions: (1) senses the activation of a piston level; (2) identifies the registration-changing-aid function being requested by the activation of the piston **11;** executes the registration-changing-aid function through the output of discrete output signals. All programs executed by central processing unit **19** are stored in the read-only memory module **20,** read-only memory module **20** responding to address locations **0 – 2047.** Program required data are also stored in read-only memory module **20** in the form of lists, the list format being set forth in Table 8 which is displayed hereinbelow.

TABLE 8

LIST FORMATS

List 0:
- Setter loop count
- Setter addr.
- Delay count addr.
- Begin. OIM addr.
- Loop count, OIM clear
- Mask
- Begin. IIM addr.
- Loop count, IIM search

List 1:
- Loop count, IDENT search
- {Compare item
- {Param. list addr.
- . . .

List 2:
- Divisional/Independent General/Coupler Combon Routine List
  - DIGCID
  - Combination length
  - Combination base addr.
  - Repeat count
  - {READ loop count
  - {Begin. OIM addr.
  - . . .
  - {Low mask
  - {Begin. IIM addr.
  - {Loop count
  - {High mask
  - . . .
- Collective General Combon Routine List
  - COLGEN
  - Loop count
  - {DIGCID param. list addr.
  - . . .
- Single Cancel Routine List
  - SCID
  - OIM addr.
  - Cancel data
- Consecutive Multiple Cancel Routine List
  - CMCID
  - OIM addr.
  - Low cancel data
  - Loop count
  - High cancel data
- Random Cancel Routine List
  - RCID
  - Loop count
  - {OIM addr.
  - {Cancel data
  - . . .
- Reversible Routine List
  - REVID
  - IIM addr.
  - OIM addr.
  - Mask

The list of addresses, data required and function being performed in connection with the list format designated in Table 8 are fully identified and displayed in Table 9 which is set forth hereinbelow.

TABLE 9

LISTS

List 0:

Address	Data	Comment
1024	4	Setter loop count
1026	2068	Setter addr.
1028	2070	Delay count addr.
1030	2560	Beginning OIM addr.
1032	10	Loop count, OIM clear
1034	$FFF8_{16}$	Mask
1036	2048	Beginning IIM addr.
1038	5	Loop count, IIM search

List 1:

Address	Data	Comment
1040	16	Loop count, IDENT search
	16411	Pedal combon compare item
	1106	
	16423	Great combon compare item
	1126	
	16435	Swell combon compare item
	1146	
	16447	Choir combon compare item
	1166	
	16459	Coupler combon compare item
	1186	
	16471	Collective general combon
	1242	compare item
	16483	Independent general combon
	1254	compare item
	16484	Pedal cancel compare item
	1274	
	16485	Great cancel compare item
	1284	
	16486	Swell cancel compare item
	1294	
	16487	Choir cancel compare item
	1304	
	16488	Coupler cancel compare item
	1314	
	16489	Swell tremolo off compare item
	1346	
	16490	Doubles off compare item
	1352	
	16491	General cancel compare item
	1372	
Address	Data	Comment
	16492	Great reversible compare item
1104	1382	

List 2:

Address	Data	Comment
1106	DIGCID	Pedal combon list
1108	6	
1110	3584	
1112	0	
1114	3	
1116	2560	
1118	$FFFF_{16}$	
1120	2060	
1122	1	
1124	0707_{16}	
1126	DIGCID	Great combon list
	6	
	3656	
	0	
	3	
	2564	
	$F8F8_{16}$	
	2062	
	1	
	0707_{16}	
1146	DIGCID	Swell combon list
	6	
	3728	
	0	
	3	
	2568	
	$F8F8_{16}$	
	2064	
	1	
	$3F3F_{16}$	
1166	DIGCID	Choir combon list
	8	
	3800	
	0	
	4	
	2572	

TABLE 9-continued

LISTS

	$COCO_{16}$	
	2066	
	2	
	0707_{16}	
Address	Data	Comment
1186	DIGCID	Coupler combon list
	14	
	3896	
	3	
	2	
	2562	
	2	
	2566	
	1	
	2572	
	2	
	2576	
	$EOEO_{16}$	
	2061	
	0	
	0707_{16}	
	$FCFC_{16}$	
	2063	
	0	
	0707_{16}	
	3838_{16}	
	2066	
	8000_{16}	
	0000_{16}	
	$EOEO_{16}$	
	2068	
	0	
	0707_{16}	
1242	COLGEN	Collective general combon
1244	4	list
	1106	
	1126	
	1146	
	1166	
1254	DIGCID	Independent general
	20	combon list
	4064	
	0	
	10	
	2560	
	$FFFF_{16}$	
	2060	
	8	
	0707_{16}	
1274	CMCID	Pedal cancel list
	2560	
	$OOFF_{16}$	
	1	
	0007_{16}	
1284	CMCID	Great cancel list
	2564	
	$OOF8_{16}$	
	1	
	0007_{16}	
1294	CMCID	Swell cancel list
	2568	
	$OOF8_{16}$	
	1	
	$003F_{16}$	
1304	CMCID	Choir cancel list
	2572	
	$OOCO_{16}$	
	2	
	0007_{16}	
1314	RCID	Coupler cancel list
	7	
	2562	
	$OOEO_{16}$	
	2564	
	0007_{16}	
	2566	
	$OOFC_{16}$	
	2568	
	0007_{16}	
	2572	
	0038_{16}	
	2576	
	$OOEO_{16}$	
	2578	
	0007_{16}	
1346	SCID	Swell tremolo off list
	2572	
	0004_{16}	
1352	RCID	Doubles off list
	4	
	2560	
	0003_{16}	
	2562	
	0001_{16}	
	2568	
	0008_{16}	
	2570	
	0040_{16}	
1372	CMCID	General cancel list
	2560	
	$OOFF_{16}$	
	8	
	0007_{16}	
1382	REVID	Great reversible list
	2063	
	2566	
	0404_{16}	

Execution of the program routines to permit the present invention organ registration affecting system to operate in accordance with its objectives is fully identified by the routines and subroutines which are identified in Tables 10 – 20 inclusive. The following sets forth a summary of Tables **10** – **20** and the respective routines:

Table 10	Search Routine
Table 11	Identify Routine
Table 12	Division/Independent General/ Coupler Combon Routine
Table 13	Collective General Combon Routine
Table 14	RAMAD Subroutine
Table 15	Set Subroutine
Table 16	CONVT Subroutine
Table 17	Read Subroutine
Table 18	Wait Subroutine
Table 19	Cancel Routine
Table 20	Reversible Routine

The search routine identified in Table 10 has three functions to perform: (1) initialization of the hardware; (2) clear the output interface modules **17**; (3) strobe the input interface modules **15** looking for an activated piston **11**.

TABLE 10

SEARCH ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	SEARCH	OUT	0		Emit reset pulse, set reg. page 0
1		LRI	1	1030	Load R[1] with pointer to params.
2		LA2	1		Load beginning OIM address
3		ST	7		Move to R[7]
4		LA2	1		Load loop count
5		ST	6		Move to R[6]
6		XOR	0		Clear accumulator
7	REPT1	SA2	7		Store 0 in 2 bytes of the OIM
8		SU1	6		Decrement count by 1
9		CNR	N, Z	REPT1	Branch if count > 0

TABLE 10 -continued

SEARCH ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
10		LA2	1		Load mask
11		ST	5		Move to R[5]
12		LA2	1		Load begin. IIM addr.
13		ST	4		Move to R[4]
14		LA2	1		Load loop count
15		ST	3		Move to R[3]
16	REPT2	LD	4		Load begin. IIM addr.
17		ST	7		Move to R[7]
18		LD	3		Load loop count
19		ST	6		Move to R[6]
20	REPT3	LA2	7		Load 2 bytes from IIM
21		CNR	Z	IDENT	Branch if accum. $\neq$ 0
22		SU1	6		Decr. count by 1
23		CNR	N, Z	REPT3	Branch if count > 0
24		LA2	7		Load last 2 bytes from IIM
25		AND	5		Apply mask
26		CNR	Z	IDENT	Branch if accum. $\neq$ 0
27		CNR	0	REPT2	Branch to REPT2

Register useage:
R[0] = accum.
R[1] = pointer to params.
R[3] = loop count, IIM search
R[4] = begin. IIM addr.
R[5] = mask
R[6] = loop counter
R[7] = pointer to OIM or IIM

35 bytes

The first instruction of the search routing, statement number **0**, is located at location zero of read-only memory module **20.** When power is applied to the present invention organ registration affecting system, statement **0** is the first instruction to be executed. R page counter **150** (FIG. **16**) is reset to zero by the execution of instruction OUT. Statement **1** sets the R register **100** designated as R[1] to point to list **0** (Table 9) where various parameters required for the remaining statements will be obtained. Statements **2** and **3** place the first parameter, i.e., the lowest address of the output interface module **17** (address number **2560**) into the R register **100** designated as R[7] and increments the list zero pointer to the next parameter. Statements **4** and **5** establish an iterative loop count into the R **100** register designated as R[6]. Statement 7 clears the accumulator, i.e., R[0]. The iterative loop consisting of statements **7, 8** and **9** copy the logical zero stored in R[0] to successive locations in output interface module **17.** The R register **100** designated as R[7] gives the addresses and the R registers **100** designated as R[6] establishes the loop iterations. The clearing process insures that the stop and coupler keys **12** of organ **10** are under manual control of the operator. Statements **10 – 15,** inclusive, place into the R register **100** designated as R[5], R[4] and R[3] the parameters required for the next loop which will search for the activation of a piston **11.**

The R **100** register designated as R[5] now holds a mask which will be applied to read-only memory **20** word **2058** so that when the location thereof is read, the unused bits are ignored. The R **100** register designated as R[4] holds the lowest input interface module **15** address, i.e., **2048.** The R **100** register designated as R[3] contains the loop count. Statements **16–19,** inclusive, copy the contents of those portions of R register **100** designated as R[4] and R[3] to the portions designated as R[7] and R[6] respectively. R[7] and R[6] are altered by the inner loop, statements **20–23,** inclusive, so that R[4] and R[3] will be used to initialize them for another pass through the input interface module **15** if such is required. The inner loop fetches a word from the input interface module **15** and branches to the identify routine (Table 11) if the word contains any one-bits. If no one-bits are found, another word is fetched. The loop count causes word **2056** to be the last read. Statements **24 – 26,** inclusive, read word **2058,** apply the mask and check for a one-bit. Failing to find an activated piston **11,** statement **27** causes the execution of a branch instruction to statement **16** and the strobing of the input from input interface modules **15** repeated. When an activated piston **11** is found by strobing the inputs from input interface modules **15,** the contents of R register **100** are as follows: (1) R[0] contains the input interface module **15** word with the one-bit; (2) R[1] points to the beginning of list one (Table 9); (3) R[7] contains the address of the input interface module **15** word plus **2.**

The identify routine is shown in Table 11 and uses the contents of the R register **100** portions R[0], R[7], and list one to cause a branch branch to a routine which performs the registration-changing-aid function corresponding to the activated piston **11.** In addition, a pointer to a new list is passed to that routine.

TABLE 11

IDENTIFY ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or _im field	Comment
0	IDENT	ST	2		Move value in accum. to R[2]
1		LD	7		Load R[7] into accum.
2		SLL	2		Shift accum. left 3 bits
3		ST	7		Put back into R[7]

TABLE 11-continued

IDENTIFY ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or _im field	Comment
4		LRI	6	15	Load loop limit count into R[6]
5		LD	2		Bring back piston data
6	REPT4	LD	0		Obtain sign of accum.
7		CBR	N	CONT1	Branch if accum. < 0
8		AD1	7		Incr. R[7] by 1
9		SLL	0		Shift accum. left 1 bit
10		SU1	6		Decr. count by 1
11		CNR	N	REPT4	Branch if count ≥ 0
12		CNR	O	SEARCH	Error-start over
13	CONT1	LA2	1		Load loop count
14		ST	6		Move to R[6]
15	REPT5	LA2	1		Load compare item
16		CMPR	7		Compare accum. with R[7]
17		CNR	N	CONT2	Branch if accum. ≥ R[7]
18		ARI	1	2	Skip to next compare item
19		SU1	6		Decr. count by 1
20		CNR	N, Z	REPT5	Branch if count <
21		CNR	0	SEARCH	Error-start over
22	CONT2	SB	7		Subtract R[7] from accum.
23		ST	7		Hold result in R[7]
24		LA	1		Load new param. list addr.
25		ST	1		Move to R[1]
26		LA2	1		Load addr. of handler routine
27		BR	0		Branch to handler

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = temp.
R[6] = loop counter
R[7] = piston no. data

38 bytes

Statements **0 – 12** convert the contents of those portions of R register **100** designated as R[**0**] and R[**7**] into a single number called the piston ID. The piston ID has a value which is established by the following equation:

$$\text{Piston ID} = [(R[7])*(8) + [15 - (\text{Bit position of one-bit in } R[0])]$$

Statement **0** places the contents of the R register **100** designated as R[**0**] into R[**2**] for temporary storage. Statements **1 – 3** cause an effective multiplication of the contents of R[**7**] by eight. In preparation for entrance into a loop, statement **4** places the loop count into R[**6**]. The only function of the loop is to act as a safety stop. The input interface module **15** word is transferred back to R[**0**] during the execution of statement **5.** At statement **6,** the register **100** designated as R[**0**] is loaded with the identical quantity, this having the effect of setting N register **104** and Z register **105** to the appropriate setting. If the most significant bit in R[**0**] was the one-bit, execution of statement **7** will cause a branch to statement **13.** If the most significant bit of R[**0**] was not the one-bit, the contents of R[**7**] is incremented and R[**0**] shifted left one position through the execution of statements **8** and **9.** The loop count established by statement **4** is decremented and the loop re-entered through the execution of statements **10** and **11.** The execution of statement **12** insures that the one-bit in the original input interface module **15** word, if it was there, was not lost. The above described loop completes the formation of the piston ID.

When statement **13** is reached, the piston ID is stored in that portion of R register **100** designated as R[**7**]. A loop count established from list one (Table 9) is loaded into R[**6**] upon the execution of statements **13** and **14.** The next loop comprising statements **15 – 21** compares the piston ID to successive words from list one. The data words compared from list one are equal in value to the piston ID for the highest numbered piston **11** of the associated registration-changing-aid function. For example, word **1042** has the value 16411 which is the piston ID for pedal combon, Statement **15** loads the word from list one into the R register **100** designated as R[**0**]. The word is compared with the piston ID at statement **16.** If the contents of R[**7**] is less than or equal to the contents of R[**0**] then a branch to statement **22** is made. If the contents of R[**7**] are not less than or equal to the contents of R[**0**] the contents of R[**1**] are incremented to establish same for the next compare item at statement **18.** The loop count is decremented and the loop re-entered through the execution of statements **19** and **20.** The loop count and statement **21** insure that no items pass the end of list one are read during the loop should an error have occurred during execution.

Statement **22** of the subroutine is entered when the item which was read from list one results in a proper comparison. The portion of R register **100** designated as R[**7**] is subtracted from the contents of R[**0**], the result thereof being provisionally designated as a pseudo-piston number by statement **22.** The pseudo-piston number is placed in the portion of R register **100** designated as R[**7**] upon executing statement **23.** The contents of the portion of R register **100** designated as R[**1**] points to the word in list one which follows the compared item. Statements **24** and **25** result in the retrieval of the word pointed to and result in the storage of same within that portion of R register **100** designated as R[**1**]. R[**1**] now points to the beginning of the new list. The first item in the new list is the address of the next routine to be executed. The address is loaded into that portion of R register **100** designated as R[**0**] and a branch instruction is executed pursuant to statements **26** and **27.** Following the termination of the identify routine, the portion of R register **100** designated as R[**7**] contains the pseudo-piston number, R[**1**] storing the list pointer for the next routine.

The routine set forth in Table 12 is applicable for any combon except a collective general. The routine set forth in Table 12 is executed following the identify routine. Two modes of operation of the routine in Table 12 are possible for each piston, the two possible modes being read or set a selected combination. Each combination consists of one or more words. The format of data within every word is such that when the combination is read to specified locations of the output interface modules **17** by the read subroutine, the appropriate keys **12** will assume the states that existed when the set subroutine was executed. The functions performed by the routines set forth in Table 12 are as follows: (1) call the RAMAD subroutine, the execution of same calculating the locating in the random-access memory module **21** for the combination; (2) determine if the setter piston **11** has been activated; (3) call the set subroutine if the setter piston **11** has been activated; (4) call the read and wait subroutines if the setter piston **11** has not been activated.

TABLE 12

DIV/INDEP GEN/COUPLER COMBON ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	DIGCID	BLD	2	RAMAD	Determine RAM addr.
1		LRD	6	1024	Load setter loop count into R[6]
2		LRD	0	1026	Load setter addr. into accum.
3		LA	0		Load setter data
4	REPT6	SU1	6		Decr. count by 1
5		CBR	N	CONT3	Check if setter bit positioned
6		SLL	0		Shift accum. left 1 bit
7		CNR	0	REPT6	Branch to REPT6
8	CONT3	LD	0		Obtain sign of accum.
9		CBR	N	SET	Branch if setter bit is true
10		BLD	2	READ	Perform READ operation
11		CNR	0	WAIT	Branch to WAIT

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = linkage
R[6] = setter loop counter
R[7] = RAM addr.
24 bytes

Statement **0** causes a branch and link to the RAMAD subroutine (Table 14). The link, or return address, remains stored in that portion of R register **100** designated as R[**2**]. The RAMAD subroutine will calculate a unique beginning address for the combination and return it to the portion of R register **100** designated as R[**7**]. The RAMAD subroutine returns control to statement **1.** The state of the setter piston **11** is checked during the execution of statements **1 – 9,** inclusive. The portion of R register **100** designated as R [**6**] receives a loop count from list one in statement **1.** The portion of R register **100** designated as R[**0**] receives the input interface module **15** address of the word containing the setter state which is then used to load the word during the execution of statements **2** and **3.** The loop established during the execution of statements **4 – 7** uses the loop count to shift the setter bit into the most significant bit position. With the setter bit thus positioned, statements **8** and **9** determine its value and cause a branch to the set subroutine (Table 15) if its value is a one. If the value of the setter bit is not a one, a branch and link to the read subroutine is followed, upon return, by a branch to the wait subroutine (Table 18).

If a collective general piston **11** is activated, the collective general combon routine designated in Table 13 is activated. The routine shown in Table 13 is branched to from the identify routine (Table 11) described hereinabove.

Table 13

COLLECTIVE GENERAL COMBON ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	COLGEN	LA2	1		Load coll. gen. loop count
1		ST	6		Move to R[6]
2	REPT7	LA2	1		Load DIGCID param. addr.
3		OUT	2		Change to page 1
4		ST	1		Move to R[1]
5		OUT	1		Change to page 0
6		LD	7		Load piston no. data
7		OUT	2		Change to page 1
8		ST	7		Move to R[7]
9		BLD	2	RAMAD	Determine RAM addr.
10		BLD	2	READ	Perform READ operation
11		OUT	1		Change to page 0
12		SU1	6		Decr. count by 1
13		CNR	N, Z	REPT7	Branch if count > 0
14		CNR	O	WAIT	Branch to WAIT

Register useage, page 0
R[0] = accum.
R[1] = ptr. to coll. gen. params.
R[2] = linkage
R[6] = loop counter
R[7] = piston no. data

page 1:
R[0] =accum.
R[1] = ptr. to DIGCID params.
R[2] = linkage
R[7] = piston no. data, RAM addr.

21 bytes

Using the pseudo-piston number passed from the identify routine, the general collective combon routine copies the combination of each similarily numbered divisional piston **11** to the output interface module **17.** The routine is entered with those portions of R register **100** designated as R[7] and R[1] containing the pseudo-piston number and a pointer to word **1244** of list two respectively. The routine contains a single loop comprising statements **2 – 13,** inclusive. Each iterative pass through the loop causes one complete combination to be transferred to output interface module **17.** Statements **0** and **1** load the portion of R register **100** designated as R[**6**] with the loop count. The loop begins by loading R[1] of page 1 with a pointer to that portion of list two which corresponds to one of the divisional combons. The pseudo-piston number is then stored in that portion of R register **100** designated as R[7]. Using the registers of page 1, the RAMAD subroutine (Table 14) calculates the beginning address of the first combination, the pseudo-piston number will be replaced by this address. During the execution of statement **10,** the read subroutine (Table 17) copies a combination to the output interface module **17.** Returning to Page 0, statement **12** executes a decrement operation of the collective general loop count. Branching back to statement **2,** another combination from another division will be read. After the four combinations are loaded into the output interface module **17,** execution of statement **14** causes a branch to the wait subroutine (Table 18) to give the actuators **14** sufficient time to be positioned. As stated previously, the delay count **16** as identified in FIG. **1** establishes the waiting period.

The RAMAD subroutine is set forth in Table 14 and is used to establish the address for the random-access memory **21.**

The RAMAD subroutine is commenced with the contents of those portions of R register **100** designated as R[7], R[2] and R[1] containing the pseudo-piston number, the return address and a pointer to the appropriate portion of list two, respectively. As an example of the pointer, the pointer would be to word **1108** if a pedal divisional combon had been activated. The RAMAD subroutine will form a random-access memory address from a base address found in list two, i.e., word **1110,** and an off-set which is equivalent to the product of the length of a combination for the particular registration-changing-aid function and the pseudo-piston number. Statement **0** clears the accumulator (R[**0**]) in preparation for the loop established in statements **1 – 4,** inclusive. The loop decrements the pseudo-piston number and branches to the branch identifier CONT **4** if the result is negative. If the result is not negative, a combination length (in bytes) is added to the contents of the portion of R register **100** designated as R[**0**] and the loop is repeated. Upon reaching statement **5,** the contents of R[**0**] constitutes the off-set. Upon reaching statement **5,** the list two pointer is incremented and the base address obtained and added to the off-set in the following statement. The final address is moved to that portion of R register **100** designated as R[7] and the divisional combon routine (Table 12) is returned to.

The set subroutine is identified in Table 15. The set subroutine creates a combination and places it in the random-access memory module **21** beginning at the address which is passed from the RAMAD subroutine (Table 14), i.e., in that portion of R register **100** designated as R[7]. Any divisional, independent general, or coupler combination can be created, the choice thereof being solely dependent upon the parameters from list two.

TABLE 14

RAMAD SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	RAMAD	XOR	0		Clear accum.
1	REPT8	SU1	7		Decr. piston no. data by 1
2		CBR	N	CONT4	Branch if result < 0
3		AA	1		Add in RAM block length
4		CNR	0	REPT8	Branch to REPT8
5	CONT4	ARI	1	2	Incr. param. ptr. to next item
6		AA2	1		Add in RAM base addr.
7		ST	7		Move to R[7]
8		BR	2		Return

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = linkage
R[7] = piston no. data
13 bytes

Table 15

SET SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	SET	LA2	1		Load repeat count
1		ST	4		Move to R[4]
2	REPT13	ARI	1	4	Advance ptr. over READ params.
3		SU1	0		Decr. count by 1
4		CNR	N	REPT13	Branch if count ≥ 0
5		OUT	2		Change to page 1
6		LRI	3	$OOFF_{16}$	Load const. into R[3]
7		OUT	1		Change to page 0
8	REPT14	LA2	1		Load low mask
9		ST	5		Move to R[5]
10		LA2	1		Load begin. IIM addr.
11		OUT	2		Change to page 1
12		ST	7		Move to R[7]
13		BLD	2	CONVT	Load and convert data
14		OUT	1		Change to page 0
15		AND	5		Apply mask
16		SA2	7		Move to RAM
17		LA2	1		Load loop count
18		CBR	N	CONT6	Branch if count < 0
19		CBR	Z	CONT5	Branch if count $= 0$
20		ST	5		Move to R[5]
21	REPT15	OUT	2		Change to page 1
22		BLD	2	CONVT	Load and convert data
23		OUT	1		Change to page 0
24		SA2	7		Move to RAM
25		SU1	5		Decr. count by 1
26		CNR	Z	REPT15	Branch if count $\neq 0$
27	CONT5	LA2	1		Load high mask
28		ST	5		Move to R[5]
29		OUT	2		Change to page 1
30		BLD	2	CONVT	Load and convert data
31		OUT	1		Change to page 0
32		AND	5		Apply mask
33		SA2	7		Move to RAM
34		CNR	0	CONT7	Skip next instruction
35	CONT6	LA2	1		Skip over high mask
36	CONT7	SU1	4		Decr. repeat count by 1
37		CNR	N	REPT14	Branch if count ≥ 0
38		CND	0	SEARCH	Branch to SEARCH

Register useage, page 0:
R[0] = accum.
R[1] = ptr. to params.
R[4] = repeat counter
R[5] = masks, loop counter
R[7] = ptr. to RAM

page 1:
R[0] = accum.
R[2] = linkage
R[3] = $OOFF_{16}$
R[7] = ptr. to IIM

57 bytes

Each combination word created by the set subroutine is derived from information contained in a byte of the input interface module **15.** Every byte of key data provided through the input interface module **15** (bytes **2060 – 2069**) will be expanded into a full word in order to recreate, at a later time, when it is tranferred to the output interface module **17,** the present key positions. Expansion is necessary because two bits in the otuput interface module word are necessary to establish the new state of any key.

The first section of the set subroutine creates the first combination word from the lowest address byte of those to be processed. In general, only a portion of this byte is relevant. For example, when setting combination on a great combon, only bits **3 – 7** of byte **2062** are allowed to establish set and clear bits in the combination word. The second subroutine section processes all bytes which are used in their entirety. With the great combon, this would only be byte **2063.** The third section processes the upper most byte of data. This would be bits **0 – 2** of byte **2064** for the great combon. All three sections are enclosed in a loop to permit their use more than once as for a coupler combon.

Statement **0** extracts a parameter from link **2** (such as the one in word **1112**) whose value is one less than the number of times the subroutine main sections are to be executed. The parameter is copied into that portion of R register **100** designated as R[**4**] for later use. The parameter is still stored in R[**0**] to serve as a loop counter for the loop consisting of statements **2 – 4** inclusive. The loop advances the list two pointer over parameters for the read subroutine (such as words **1114** and **1116**). Upon the execution of statements **5 – 7,** a constant of value of OOFF, in hexadecimal, is loaded into R[**3**] of register **100** for page 1.

Upon reaching statement **8,** the set subroutine's first main section is entered. Statement **8** and statement **9** load that portion of R register **100** designated as R[**5**] with the low order mask (such as in word **1118**) which is to be used later in the subroutine. Following statement **9,** the portion of R register **100** designated as R[7] of page 1 is loaded with the address of the first byte of the input interface module **15** to be processed (such as that in word **1120**). A branch and link to the CONVT subroutine (Table **16**) is made in statement **13.**

The CONVT subroutine shown in Table **16** obtains one byte of key-state-information from the input interface module **15** and creates a combination word.

TABLE 16

CONVT SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	CONVT	LA	7		Load data from IIM
1		ADI	7		Incr. IIM ptr. by 1
2		SLL	7		Shift accum. left 8 bits
3		ST	4		Hold in R[4]
4		SRL	7		Shift accum. right 8 bits
5		XOR	3		Complement low accum. byte
6		OR	4		Bring back set data
7		BR	2		Return

Register useage:
R[0] = accum.
R[2] = linkage
R[3] = $OOFF_{16}$
R[4] = temp.
R[7] = ptr. to IIM
8 bytes

Statement **0** of the convert subroutine loads one word from the input interface modules **15** into that portion of R register **100** designated as R[**0**]. Next, the input interface module **15** pointer is incremented by one to reflect that only the low order byte of this word is relevant, the high order byte becoming the low order byte of the next word read from the input interface module **15.** Statement **2** shifts the contents of that portion of R register **100** designated as R[0] toward the more significant end, the shift constituting eight bit positions. The desired byte is not in the high order byte position and the low order byte position contains all zeroes. The contents of R[**0**] are copied into R[**4**] for temporary storage in statement **3.**

Upon the execution of statement **4,** the byte of input interface module **15** data stored in R[**0**] is shifted back into the low order position filling the high order byte with zeroes. The byte of data is complemented by means of the EXCLUSIVE-OR with the contents of that portion of R register **100** designated as R[**3**]. The original input interface module **15** byte and its complement are then placed in the same word upon the execution of statement **6.** At statement **7,** the convert subroutine branches back to the set subroutine at statement **14** thereof.

Statement **14** of the set subroutine is the next to be executed. The combination word returned by the convert subroutine will typically contain erroneous data, i.e., bits which would cause the wrong keys to assume new states after the word is transferred to the output interface module **17.** Statement **15** applied a mask to the combination word which in effect zeros-out the incorrect data. The corrected word is stored in random-access memory module **21** upon the execution of statement **16.**

Upon entry to statement **17,** a parameter is obtained from list two which indicates if the subroutines second and third sections are to be executed, i.e., such as the parameter in word **1122.** If the value of the parameter is negative, neither section is to be executed. Statement **18** detects the value of the parameter and causes a branch on the value. If the parameter has a value of zero, only the third section of the set subroutine is to be executed. Statement **19** causes a branch to the third section. If the parameter read is positive, its value specifies the number of times section two is to be executed before the third section is to be executed.

Assuming a parameter value of one or more exists, entrance to statement **20** will transfer the parameter to that portion of R register **100** designated as R[**5**] the value to be used as a word counter. Statements **21 – 23** create a combination word. The entire word is valid and therefore it is stored in the random-access memory module **21** immediately. Statements **25** and **26** decrement the loop count stored in that portion of R register **100** designated as R[**5**] and cause re-entry of the second section of the set subroutine if the count remains positive.

Section three commences at statement **27** by loading that portion of R register **100** designated as R[**5**] with the high mask. This mask serves the same function as the low mask in the first section. The last combination word is created by statements **29 – 31.** The mask is applied and the word stored in the random access memory module **21.** Statement **35** is executed only when sections two and three have been bypassed. Statement **35** serves only to advance the list two pointer over a dummy high mask. Statements **36** and **37** determine if the three sections are to be re-executed. When all combination words have been created and stored in the random-access memory module **21,** statement **38** causes a branch to the search subroutine (Table 10) to strobe for the activation of another piston **11.**

The read subroutine is disclosed in Table 17. The read subroutine copies a combination from the random-access memory module **21** to specific locations in the output interface modules 17. The subroutine is entered with a list two pointer pointing to the repeat count, i.e., such as that which appears in word **1112.**

Table 17

READ SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	READ	LA2	1		Load repeat count
1		ST	4		Move to R[4]
2	REPT9	LA2	1		Load READ loop count
3		ST	6		Move to R[6]
4		LA2	1		Load begin. OIM addr.

Table 17-continued

READ SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s._d._or_im field	Comment
5		ST	5		Move to R[5]
6	REPT10	LA2	7		Load word from RAM
7		SA2	5		Move to OIM
8		SU1	6		Decr. count by 1
9		CNR	N,Z	REPT10	Branch if count > 0
10		SU1	4		Decr. repeat count by 1
11		CNR	N	REPT9	Branch if count ≥ 0
12		BR	2		Return
0					

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = linkage
R[4] = repeat loop counter
R[5] = ptr. to OIM
R[6] = READ loop counter
R[7] = ptr. to RAM

15 bytes

Statements **0** and **1** load the repeat count in that portion of R register **100** designated as R[**4**] for use as a loop counter. The count indicates the number of times the read subroutine will be executed. Statements **2** and **3** of the subroutine commence the actual processing routine by loading the member of combination words to be moved from random-access memory **21** into working register R[**6**]. Next, working register R[**5**] is loaded with the first output interface module **17** address. Statements **6 – 9** transfer each word of the combination from the random-access memory module **21** to the output interface module **17.** The transfer commences at the address calculated by the RAMAD subroutine and returned to that portion of R register **100** designated as R[7]. If another combination is to be transferred, statements **10** and **11** will cause a branch to statement **2.** If no other combination is to be transferred, a return to the calling subroutine is executed.

The wait subroutine is disclosed in Table 18. The function of the wait subroutine is to delay return to the search subroutine by central processing unit **19** until sufficient time has elapsed to permit keys **12** to assume their new state as a result of activation of actuators **14.**

count **16** (FIG. **1**) to that portion of R register **100** designated as R[**7**]. The inner delay count is then placed in that portion of R register **100** desdignated as R[**0**]. Statement **4** commences a loop which uses the outer delay count as a loop counter. This statement copies the inner delay count from R[**0**] to R[**6**] where it serves as a loop counter for the loop beginning at statement **5.** The loop comprised of statements **5** and **6** simply decrement the contents of that portion of R register **100** designated as R[**6**] until the contents become negative. The function of decrementing the count uses up an interval of time which is proportional to the contents of R[**6**]. If the result did not go negative, statement **7** decrements the outer delay count and causes re-execution of the loop set by statement **5** and statement **6.** If the contents of R[**6**] go negative, the time delay count established by delay counter **16** has been met and the search routine returned to.

Table 19 illustrates the three cancel routines which have been designated by the mnemonics SCID, CMCID and RCID as well as the common subroutine designated by the mnemonic SNGLC. The swell tremolo off piston **11** uses the SCID routine. The divisional and general cancels use the CMCID routine. The coupler

TABLE 18

WAIT SUBROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s._d._or_im field	Comment
0	WAIT	LRD	5	1028	Load delay count addr.
1		LA2	5		Load outer delay count
2		ST	7		Move to R[7]
3		LA	5		Load inner delay count
4	REPT11	ST	6		Move to R[6]
5	REPT12	SU1	6		Decr. inner count by 1
6		CNR	N	REPT12	Branch if count ≥ 0
7		SU1	7		Decr. outer count by 1
8		CNR	N	REPT11	Branch if count ≥ 0
9		CND	0	SEARCH	Branch to SEARCH
0					

Register useage:
R[0] = accum., inner count
R[5] = temp.
R[6] = inner loop counter
R[7] = outer loop counter

16 bytes

In statement **0,** the portion of R register **100** designated as R[**5**] is loaded with the address of the outer delay count set in list **0.** The outer delay count is then copied from the input interface module **15** as a result of delay

cancel and double off piston **11** use the RCID routine. Each of the routines is entered from the identify routine (Table 11) with that portion of R register **100** designated as R[**1**] pointing to a portion of list two.

TABLE 19

CANCEL ROUTINES

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	SCID	BLD	2	SNGLC	Branch to single cancel
1		CNR	0	WAIT	Branch to WAIT
0	CMCID	BLD	2	SNGLC	Branch to single cancel
1		LA2	1		Load loop count
2		CBR	N, Z	CONT8	Branch if count $\leq$ 0
3		ST	6		Move count to R[6]
4		LRI	0	$OOFF_{16}$	Load cancel data
5	REPT16	SA2	7		Move data to OIM
6		SU1	6		Decr. count by 1
7		CNR	N, Z	REPT16	Branch if count > 0
8	CONT8	LA2	1		Load high cancel data
9		SA	7		Move to OIM
10		CNR	0	WAIT	Branch to WAIT
0	RCID	LA2	1		Load loop count
1		ST	6		Move to R[6]
2	REPT17	BLD	2	SNGLC	Branch to single cancel
3		SU1	6		Decr. count by 1
4		CNR	N, Z	REPT17	Branch if count > 0
5		CNR	0	WAIT	Branch to WAIT
0	SNGLC	LA2	1		Load OIM addr.
1		ST	7		Move to R[7]
2		LA2	1		Load cancel data
3		SA2	7		Move to OIM
4		BR	2		Return

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = linkage
R[6] = loop counter
R[7] = ptr. to OIM
38 bytes

The SCID routine consists of executing the SNGLC subroutine followed by the wait subroutine (Table 18). Statement **0** and **1** of the SNGLC subroutine loads that portion of R register **100** designated as R[7] with the output interface module **17** address of the key or keys **12** to be cancelled. Subsequently, the data word which will cause the desired cancellation is fetched from list two and transferred to the output interface module **17.** After transferring the data word to output interface module **17,** statement **4** branches back to the calling routine.

The CMCID routine causes cancellation of keys **12** which are connected to two or more consecutive locations in the output interface module **17.** Statement **0** cancels the specified keys **12** of the first output interface module **17** word using the SNGLC subroutine described hereinabove. The next item obtained from list two is an indicator of the number of output interface module **17** words which are to receive cancellation data. If the item has a value of zero, a branch is made by the execution of statement **2,** the branch being to statement **8.** Should the item have a value of one or more, it is used as a loop count for the loop consisting of statements **5 – 7,** inclusive. The loop comprising statements **5 – 7** cancel all keys **12** for each output interface module **17** word. The portion of R register **100** designated as R[**6**] stores the loop count and the portion designated as R[**0**], i.e., the accumulator, stores the cancellation data word. The data word is transmitted to successive output interface module **17** location until the loop count is decremented to zero. The last cancellation data word is loaded from loop **2** and moved to the output interface module **17** during the execution of statements **8** and **9.** Subsequent to the execution of statement **9,** the wait subroutine (Table **18**) is entered. The RCID routine executes the SNGLC subroutine repetitively until the loop count is decremented to zero. The wait subroutine (Table 18) terminates this routine as described in connection with the other cancellation routines.

The reversible routine is described in Table 20. The reversible routine obtains the present state of a particular key **12** from the input interface module **15** and creates a data word which reverses the state when placed in the output interface module **17.**

TABLE 20

REVERSIBLE ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s,_d,_or_im field	Comment
0	REVID	LA2	1		Load IIM addr.
1		ST	7		Move to R[7]
2		LA2	1		Load OIM addr.
3		ST	6		Move to R[6]
4		LRI	2	$OOFF_{16}$	Load $OOFF_{16}$ into R[2]
5		LA2	1		Load mask
6		ST	4		Move to R[4]
7		LA	7		Load stop data
8		AND	2		Mask off high 8 bits
9		ST	3		Hold in R[3]

TABLE 20-continued

REVERSIBLE ROUTINE

Statement No.	Branch Identifier	Instruction Mnemonic	r_field	s._d,_or_im field	Comment
10		XOR	2		Compl. low 8 bits of accum.
11		SLL	7		Shift accum. left 8 bits
12		OR	3		Bring back clear data
13		AND	4		Apply mask
14		SA	6		Move data to OIM
15		CND	0	WAIT	Branch to WAIT

Register useage:
R[0] = accum.
R[1] = ptr. to params.
R[2] = constant $OOFF_{16}$
R[3] = temp.
R[4] = mask
R[6] = ptr. to OIM
R[7] = ptr. to IIM
20 bytes

Statements **0** and **1** load that portion of R register **100** designated as R[7] with the address of the byte in the input interface module **15** which contains the desired key state. A portion of R register **100** designated as R[**6**] receives the output interface module **17** word address for which the key can be affected. The constant **00FF,** in hexadecimal, is placed in that portion of R register **100** designated as R[2]. A mask is required to insure only that one key changes state. The mask is stored in that portion of R register **100** designated as R[**4**].

Statement **7** of the reversible subroutine fetches a word from the input interface module **15.** The key state is stored in the low order byte position. The high order byte position is set to all zeroes. The resultant word is copied into that portion of R register **100** designated as R[**3**]. Statement **10** complements the key state data. The byte of complemented data is shifted into the high order position, the uncomplemented byte being placed in the low order position. After a mask is applied, only a single one-bit remains which will cause the desired key state change. The word is placed in the output interface module **17** and the wait subroutine (Table 18) is executed.

It can therefore be seen that the present invention organ registration affecting system provides a systematic method for setting stop and coupler combinations for organ **10** based upon input data provided by pistons **11.** The use of the central processing unit **19** which is specifically designed to operate on this application in the most efficient manner possible provides a highly efficient and flexible system for dealing with either pipe or electronic organs. Upon activiation of a piston **11,** the input interface module **15** is strobed. Central processing unit **19** identifies the piston **11** which has been activated, and retrieves the desired stop combination from the list stored in memory. By outputting discrete output signals to the appropriate stop and coupler keys **12** via actuators **14,** the states of the stop and coupler keys are appropriately positioned in a manner which surpasses the systems disclosed in the prior art.

I claim:

1. An organ registration affecting system for use with electronic and pipe organs having a plurality of pistons and a plurality of stop and coupler keys and a plurality of actuators for setting and resetting the stops and couplers, said system comprising:

a. input interface means for translating and presenting signals responsive to a selected piston, the state of stop and coupler keys and engagement time of the organ actuators;

b. output interface means for setting and resetting the states of selected stop and coupler keys and actuators;

c. a central processing unit including a program counter, address register and arithmetic and logic unit and input and output means for transferring data to and from said central processing unit respectively, said input means being coupled to said input interface means and said output means being coupled to said output interface means;

d. read-only memory means for program storage, said central processing unit input means being coupled to said read-only memory means;

e. random-access memory means for storing data correlating the signal responsive to a selected piston and the selected actuator, the input and output means of said control processing unit being coupled to said random access memory means; and

f. at least two pages of working registers, each page comprising eight registers of two bytes each, said working registers being coupled to said arithmetic and logic unit of said central processing unit whereby a selected organ piston causes said central processing unit to correlate said piston with signals responsive to selected stop and coupler actuators, said signals responsive to said actuators being output to the organ by said output interface means.

2. An organ registration affecting system as defined in claim **1** wherein said input interface means comprsies a plurality of voltage translation circuits coupled to each of the stop and coupler keys and pistons, decoding means for an input line to be strobed, said decoding means being coupled to said address register, the output of said decoder means selecting one of the plurality of voltage translating circuits, a data multiplexer coupled to said voltage translation circuit and the output of said decoding means, said multiplexer outputting a signal responsive to the addressed voltage translating circuit.

3. An organ registration affecting system as defined in claim **1** wherein said output interface means comprises a plurality of binary latches equal in number to the organ actuators, output decoder means for selecting one of said plurality of said binary latches, said output decoder means being connected to selectively enable each of said latches, and output driver means for transmitting the output signal of said binary latches

to the organ actuators.

4. An organ registration affecting system as defined in claim 1 wherein said central processing unit includes a shift register having a parallel, 16 bit capacity, the input of said shift register being connected to the arithmetic and logic unit, OR means coupled to the output of said shift register for outputting a signal responsive to a binary one in any one of the shift register bit locations, said signal being coupled to a zero indication register.

5. An organ registration affecting system as defined in claim 4 wherein said central processing unit includes a carry-borrow register coupled to the output of said arithmetic and logic unit.

6. An organ registration affecting system as defined in claim 5 wherein said central processing unit includes a most significant bit indication register coupled to the most significant bit of the output of said shift register, conditional branching means coupled to said zero indication register, said carry-borrow register and said most significant bit indication register, said conditional branching means for producing a signal responsive to the function $(N \wedge r[2]) \vee (Z\, r[1]) \vee (C \wedge r[0])$.

7. An organ registration affecting system as defined in claim 4 wherein said program counter includes means for resetting said program counter to zero upon the initiation of power to said central processing unit.

* * * * *