

[54] **DIGITAL COMPUTER WITH ACCUMULATOR SIGN BIT INDEXING**
[76] Inventors: Doyle K. Cavin, 945 Verona Drive, Fullerton; Robert B. Trousdale, 13722 Rushmore Lane, Santa Ana, both of Calif.
[22] Filed: Feb. 12, 1971
[21] Appl. No.: 114,902

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 790,735, Jan. 13, 1969, abandoned.
[52] U.S. Cl. 340/172.5
[51] Int. Cl. G06f 9/20
[58] Field of Search 340/172.5; 235/157

References Cited

UNITED STATES PATENTS

2,968,027 1/1961 McDonnell et al. 340/172.5

3,111,648 11/1963 Marsh et al. 340/172.5

OTHER PUBLICATIONS

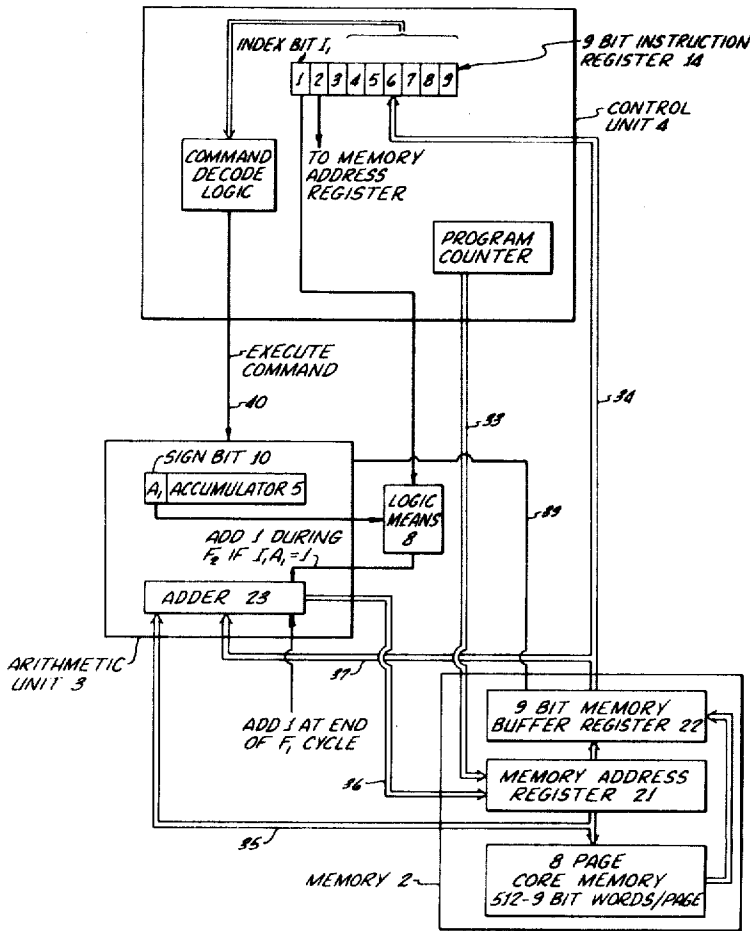
IBM 7094 Data Processing System Reference Manual, 10/21/66, Form No. A22-6703, page 48.

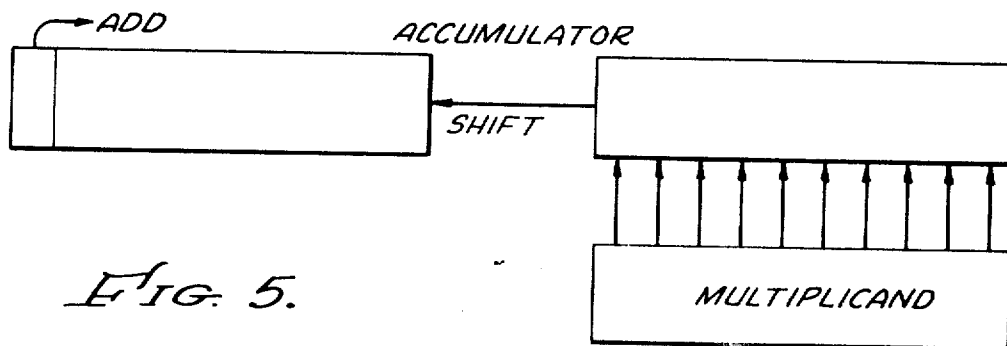
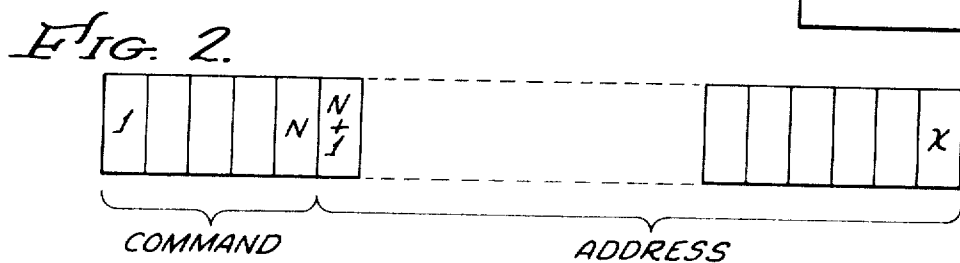
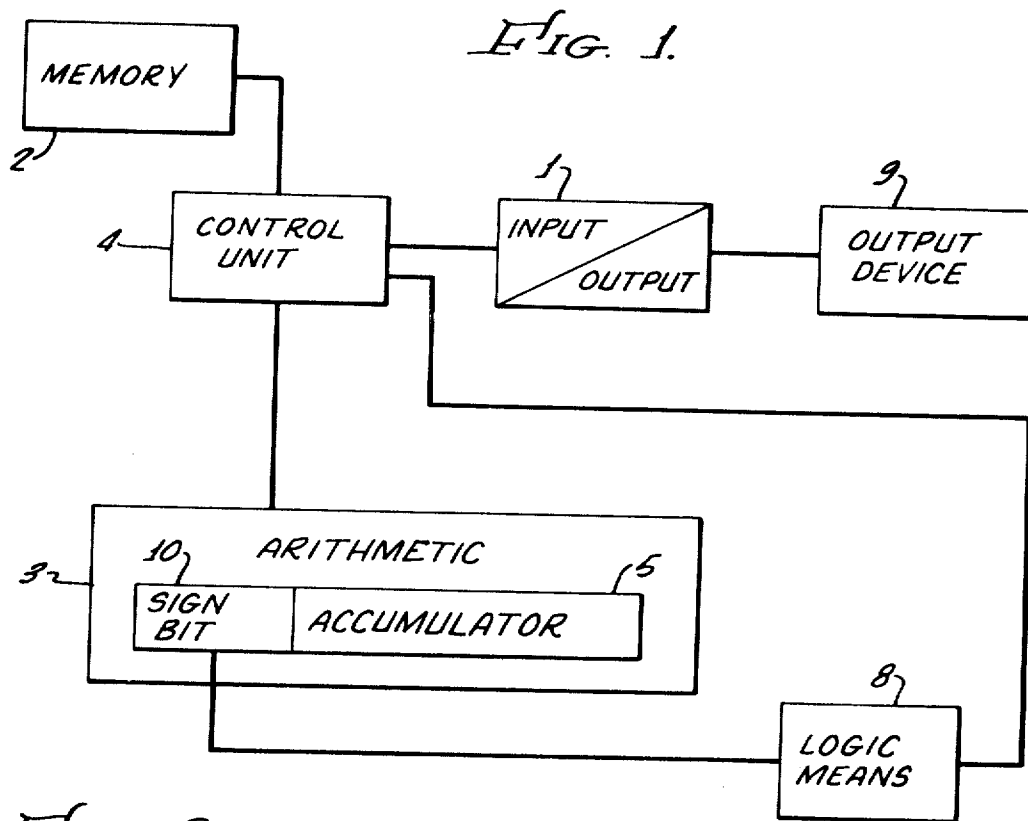
Primary Examiner—Paul J. Henon
Assistant Examiner—Mark Edward Nusbaum
Attorney—John A. Duffy and Bruce D. Jimerson

[57] ABSTRACT

In lieu of a separate indexing register, the sign bit of the accumulator may be made to function as a single bit index register to effectuate address modification. In a machine so organized the instructions have a reserved bit position which gives the programmer the option of selecting modified or unmodified addressing. Because of the versatility of this type of instruction, only a few instructions are required to in-line program a variety of normally iterate routines.

2 Claims, 9 Drawing Figures





INVENTORS.
DOYLE K. CAVIN
ROBERT B. TROUSDALE
BY

John H. Puffer
ATTORNEY.

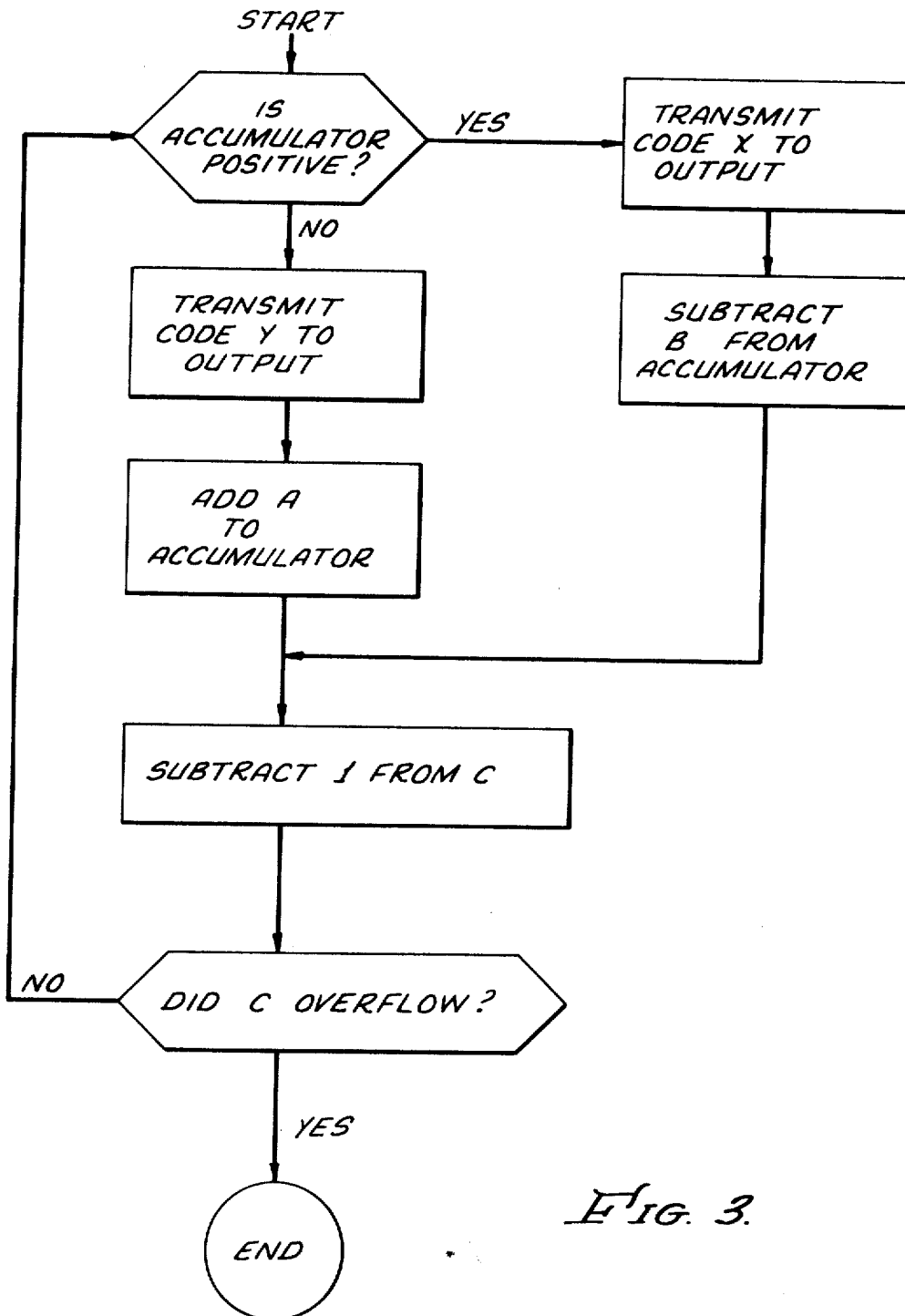


FIG. 3.

INVENTORS.
DOYLE K. CAVIN
ROBERT B. TROUSDALE
BY

John A. Duff
ATTORNEY.

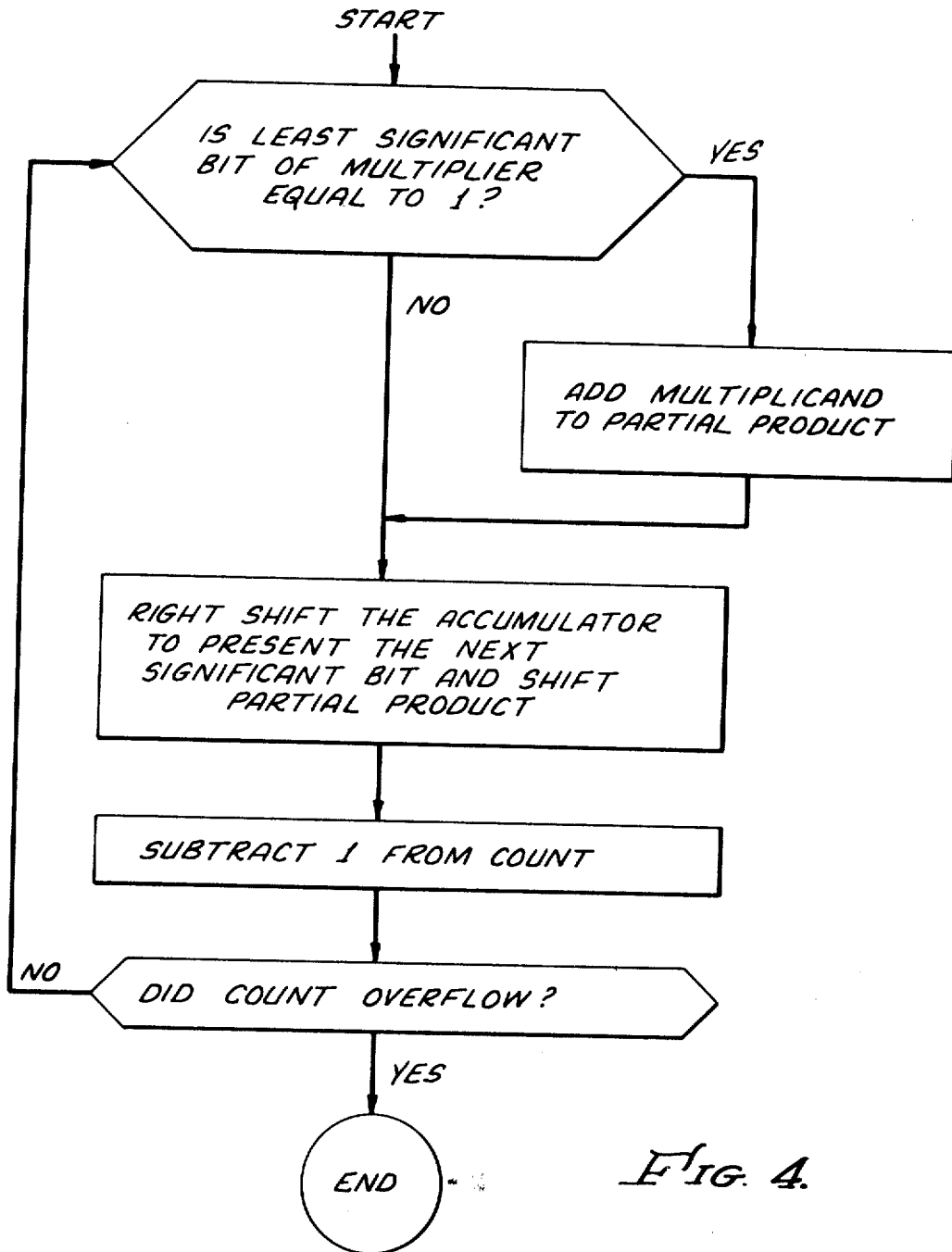


FIG. 4.

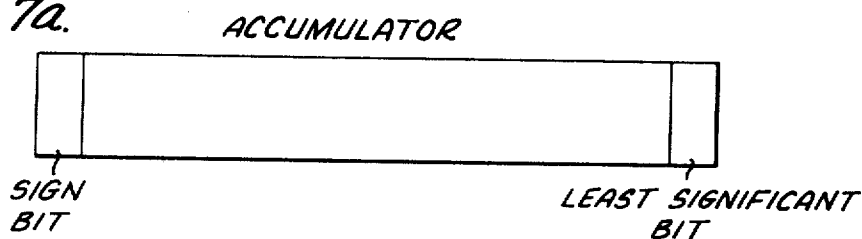
INVENTORS:
DOYLE K. CAVIN
ROBERT B. TROUSDALE
BY

John H. Hoff
ATTORNEY.

STEP	ACCUMULATOR	
	UPPER HALF	LOWER HALF
1. START:	0 1 1 0	0 0 0 0
2. ADD 0's:	0 1 1 0	0 0 0 0
3. SHIFT:	1 1 0 0	0 0 0 0
4. ADD MULTIPLICAND:	1 1 0 0	0 0 1 1
5. SHIFT:	1 0 0 0	0 1 1 0
6. ADD MULTIPLICAND:	1 0 0 0	1 0 0 1
7. SHIFT:	0 0 0 1	0 0 1 0
8. ADD 0's:	0 0 0 1	0 0 1 0
9. SHIFT:	0 0 1 0	0 1 0 0
10. SHIFT RIGHT:	0 0 0 1	0 0 1 0

FIG. 6.

FIG. 7a.



LEAST SIG. BIT	SIGN BIT	LOCATION OF NEXT INSTRUCTION
EVEN	POS.	α
EVEN	NEG.	$\alpha + 1$
ODD	POS.	β
ODD	NEG.	$\beta + 1$

FIG. 7b.

INVENTORS.
DOYLE K. CAVIN
ROBERT B. TROUSDALE
BY
John A. Duffy
ATTORNEY.

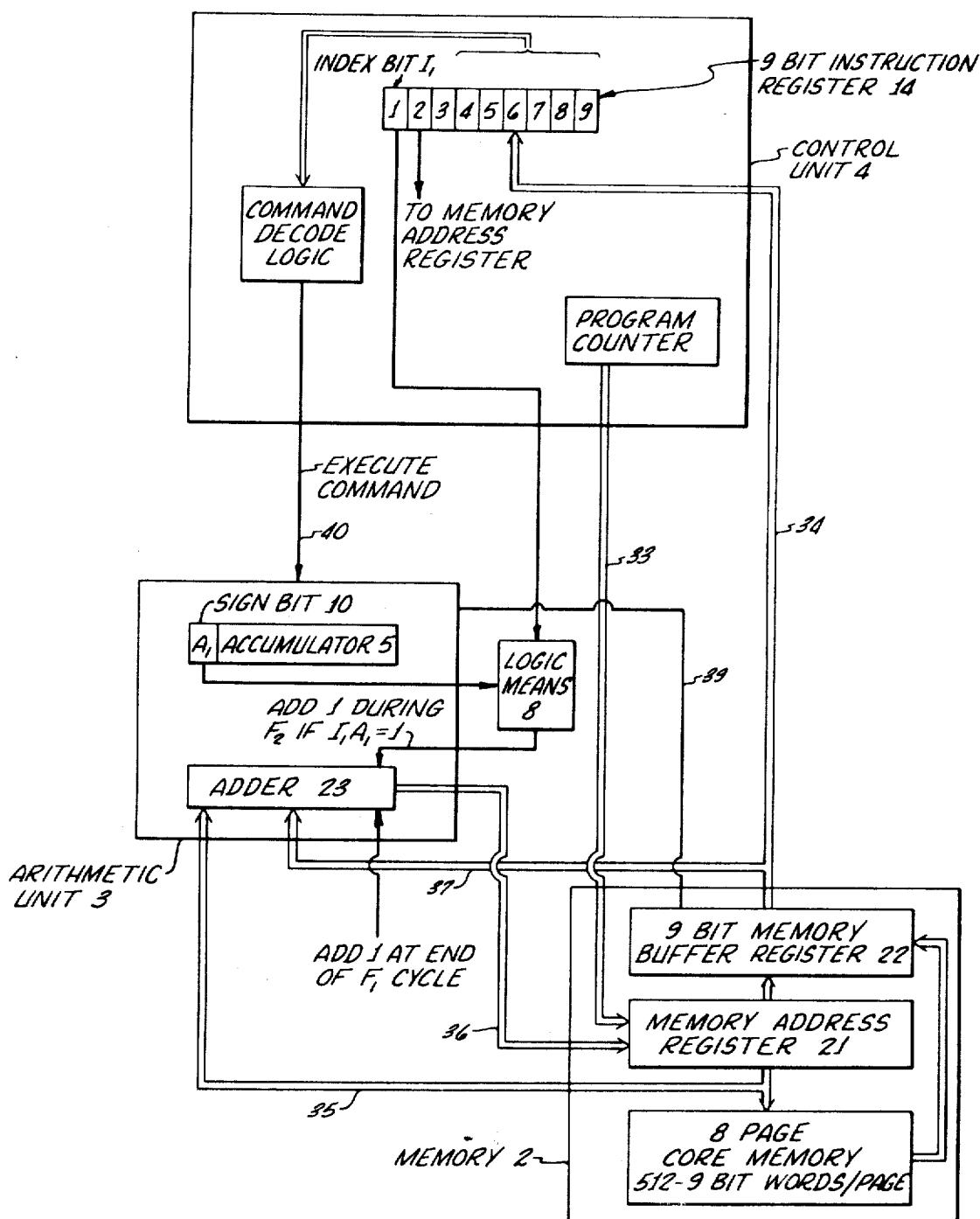


FIG. 8.

INVENTORS.
 ROBERT B. TROUSDALE
 DOYLE K. CAVIN
 BY
 Bruce D. Jimmason
 ATTORNEY.

DIGITAL COMPUTER WITH ACCUMULATOR SIGN BIT INDEXING

BACKGROUND OF THE INVENTION

This application is a Continuation-in-Part of an earlier filed copending application Ser. No. 790,735, now abandoned, filed Jan. 13, 1969.

The utilization of special and general purpose digital computers for problem solving is well-known in the art. Typically, such decision making machines are designed to execute a variety of sequential instructions upon a quantity of input data and generate therefrom the desired results whether they be mathematical answers, a series of operations for algorithm solutions, or merely the outputting of particularly specified stored data. The actual number of available instructions which a given machine is capable of executing will of course depend upon the complexity of the machine. In a large general purpose computer, the instruction set may comprise hundreds of instructions, whereas a small special purpose machine may only utilize a few basic instructions which can be combined in various ways to perform more complicated functions. One disadvantage of the small machine having a very limited set of instructions is that a number of basic instructions must be written in the program to perform even very simple arithmetic operations. Thus, in a simple machine having an instruction set consisting only of add, shift, and output, a multiplication operation would require the programmer to separately command each of the repetitive additions with a consequent consumption of a great amount of man and machine time. On the other hand, the more sophisticated and lengthy the instruction set, the more complex the machine. What is actually desired is a machine organization which is responsive to an instruction set consisting of a few very powerful basic instructions which may be efficiently combined to enable simple in-line programming of numerous standard routines which normally require complicated iterative program loops. In some of the larger machines this operation is accomplished by external address modification via individually accessible auxiliary index registers. This same function may be conveniently accomplished in even relatively small computers in accordance with the teachings of the present invention wherein, in lieu of separate indexing registers the machine is structured to have the capability of being able to respond to and directly operate upon information contained in the standard computational registers. In particular, the sign bit of a standard accumulator may be utilized as a single bit index register to control alternate choice operations.

Accordingly, it is an object of the present invention to provide a computer having an instruction set comprising a small number of very versatile instructions.

It is another object of the invention to provide a computing machine having an internal means for data and address modification which is not dependent upon separate indexing registers.

It is a further object of the invention to provide a computer which utilizes the accumulator sign bit as an index register.

Other objects and advantages of the present invention will become obvious from the detailed description of a preferred embodiment given hereinbelow.

SUMMARY OF THE INVENTION

The accumulator sign-bit indexing computer described herein comprises an arrangement for organizing a general purpose machine which will utilize the information provided by the sign of the accumulator contents in a variety of alternate choice operations. In essence the accumulator sign bit functions as a single bit index register to facilitate address modification in accordance with the dictates of certain indexing instructions. Each such accumulator indexing instruction has reserved a designated bit position for the purpose of controlling which of two addresses will be accessed by the particular instruction.

DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a general purpose digital computer.

FIG. 2 illustrates a typical instruction format.

FIG. 3 is a flow chart for performing the digital plotting of a straight line.

FIG. 4 is a flow chart for multiplying two binary numbers together.

FIG. 5 shows how a double length accumulator functions in a multiplication routine.

FIG. 6 shows the flow of the multiplication algorithm of four bit binary numbers to multiply three by six.

FIG. 7a shows how the accumulator sign bit and least significant bit may be utilized to effectuate a four way jump.

FIG. 7b is a table of the four possible combinations of sign bit and least significant bit.

FIG. 8 shows in block diagram form the requisite blocks and flow paths involved in carrying out a typical indexing operation.

DETAILED DESCRIPTION OF A PARTICULAR EMBODIMENT

FIG. 1 shows a block diagram of a typical general purpose computer having the standard elements comprising an input/output means 1, a memory 2, arithmetic center 3, and control unit 4. For the purposes of the present invention, the memory 2 may be of any type such as a random access core, thin film, MOS or serial accessed drum, disk or delay line. The computer may be organized to have a fixed or variable word size of any length (number of bits). The number system employed may be full binary with negative quantities expressed in one's or two's, complement notation, or a sign and magnitude form may be employed using binary coded decimal or any other system in which the sign of information representing a given data word is represented by a particular bit at a fixed location in the word. The arithmetic center 3 comprises an accumulator 5 which may consist of actual hardware (such as a set of flip flops or the like) or which may occupy some fixed or alterable location in the memory 2. The capacity of the accumulator 5 may be fixed or variable and may or may not be related to the word size of stored values in the memory 2. The logic means 8 in FIG. 1 is typically an "AND" gate.

The instruction set defining the computational capability of the computer should include a standard "add" or "subtract" instruction so that the accumulator contents may be modified by the contents of selected stored values in the memory to produce a resultant quantity in the accumulator having a positive or nega-

tive sign. In addition, the instruction set should include a standard "load" instruction whereby the contents of a particular address in the memory 2 may be placed in the accumulator 5; an output instruction whereby the contents of a particular location in the memory 2 or accumulator 5 may be transferred to an output device 9, such as a magnetic tape, typewriter, electronic plotter, etc.; a conditional jump instruction whereby the sign of a particular memory address (or of the accumulator) is examined to determine the address of the next instruction; an accumulator shift instruction; an input instruction, a load accumulator instruction, and a store instruction.

Although the application of the internal indexing and address modification features of the present invention are described in detail with respect to a few of the above enumerated instructions, it will be understood that the instruction set may include numerous other instructions and that the basic concepts of the present invention (to be described below) may also be combined therewith.

Adverting to FIG. 2, a typical standard instruction comprises a command (bits 1-N) specifying some basic operation, e.g., "ADD", "SUBTRACT", "STORE", "LOAD", "OUTPUT", "INPUT", etc., together with an operand address (bits N + 1 to X) which points to a specific location in the memory which contains the operand to be used in the instruction. Where iterative loops are to be programmed, it is convenient to be able to modify the address portion of the instruction. Thus, assume for example that it is desired to form the sum of a long column of numbers which are located in consecutive addresses of the computer memory from address 030 to 077. IT is thus possible to use the same add instruction for successive additions provided that the address of the "ADD" instruction is modified before each addition so as to add the contents of the next successive address to the partial sum each time around. In prior art digital computing devices, address modification is effectuated by an auxilliary means so that the operand actually accessed by the instruction can be varied as a function of the contents of the auxilliary means. Conventionally, the auxilliary means employed is an independent index register, the contents of which is normally modified by a special instruction which is independent of the instructions which change the accumulator contents. Hence, in the above example, an index register may be loaded with the value of the first address and then incremented one count each time an addition is made to access consecutive memory addresses.

An important feature of the present invention is the facility to directly modify an address without resorting to special indexing instructions and additional registers apart from the accumulator. In essence, the sign bit 10 of the accumulator 5 functions as a single bit index register. In addition, the computer is organized to be responsive to a set of instructions having a control bit I_1 for enabling the programmer to select unmodified or modified (indexed) addressing. Thus, if indexed addressing is selected, the operand address is taken to be the direct value specified in the instruction when the accumulator contains a positive number (sign bit positive) whereas if the sign bit is negative the operand address is taken to be the direct value plus a known increment (typically +1 or +2) which may vary depending

upon the specific instruction. This function is accomplished by the logic means 8 as shown in FIG. 1.

In order to illustrate the advantages of a digital computer so organized, consider the following instruction set:

LDA(Y),0 - Load accumulator with contents of location Y

LDA(Y),1 - Load accumulator with the contents of location Y if the accumulator (A) is presently positive or with the contents of Y + 1 if A is presently negative.

ADD(Y),0 - Add contents of location Y to contents of accumulator.

ADD(Y),1 - Add contents of location Y if accumulator is positive or contents of location Y + 1 in accumulator is presently negative.

OUT(Y),0 - Transfer the contents of location Y to an output device.

OUT(y),1 - Transfer the contents of location Y to an output device if accumulator is positive or contents of Y + 1 if accumulator is presently negative.

DIJB(W) - Decrement the contents of the memory cell immediately following that in which this instruction is stored. If the cell contents does not underflow, jump back N instructions (i.e., take next instruction from a location N memory cells prior to the location of the *DIJB* instruction)

With only these few selected instructions, a computer program may be written to perform, for example, a commonly used algorithm for approximating a straight line by incremental plotting. Thus, assume that the X and Y directional components of the line are given as having a ratio of A to B and that the line is to have a magnitude of "C" increments. If the plotting device is responsively coupled to the computer as an output device so as to execute incremental movements parallel to either the X or Y axis according to whether code X or code Y respectively is transmitted to the output, the line may be plotted according to the flow chart shown in FIG. 3. The complete program may be written simply as:

OUT(X),1

ADD(A),1

DIJB(2)

where the constant code X and code Y are stored in two successive locations so that the *OUT(X),1* instruction will cause code X to be selected if the accumulator is positive or code Y to be selected if the accumulator is negative. Similarly, the constants A and B are stored in two successive locations so that either A is added to the accumulator or B is subtracted from the accumulator depending upon the accumulator sign with the *ADD(A),1* instruction. The count value C - 1 is stored in the location immediately following the *DIJB* instruction so that it may be decremented once each recursion until underflow occurs.

The three-step routine described above for performing the common straight line algorithm is typical of the program simplicity which result when the computing machine is adapted to utilize the sign of the accumulator contents to facilitate address modification. To further illustrate the versatility of utilizing the special index in combination with the information provided by the accumulator sign bit to effectuate alternate choice operation, consider the flow chart shown in FIG. 4 for multiplying two binary numbers together. In this appli-

cation the sign bit is employed as a value indicator for determining whether or not the multiplicand is added. Assume that the computer is arranged to have a double length accumulator, as shown in FIG. 5, so that initially the multiplier may be placed in the left most bits (upper half) and the lower half may be set to zero. The sign bit of the accumulator is used to determine whether or not the multiplicand is to be added to the partial sum after each left shift of the accumulator. In essence, each of the multiplier bits is successively examined as to sign and discarded by a left shift operation which also has the effect of left shifting the partial sum for proper addition of the multiplicand (or zero) depending upon the sign of the next multiplier digit. Thus, if the sign bit = 1, addition of the multiplicand is performed, otherwise not. When a number of shifts equal to the number of bits of the multiplier has been performed, the multiplication routine will be complete and the product will be contained in the double length accumulator. By adding three more instructions, namely:

RSA - Right Shift Accumulator

LSA - Left Shift Accumulator

ADR(M),1 - Add contents of memory location (*M*) to right half of accumulator as indexed by sign of accumulator, a loop for implementing the algorithm using sign bit indexing may be written as:

ADR(M),1

LSA

DIJB(2)

RSA wherein *M* contains the value zero, *M* + 1 contains the multiplicand and *DIJB(2)* is followed by the number of bits of the multiplier. In FIG. 6 an example is given showing the various states of the accumulator in executing the multiplication of two four-bit numbers according to the dictates of the above loop. The multiplicand is taken to be 3(0011) and the multiplier is taken to be 6(0110). Initially, the upper half of the accumulator contains the multiplier (0110) and the lower half is coded with zeros. It will be seen that one right shift of the accumulator contents (step 10) is required to properly position the product 18 (00010010) in the accumulator following the last left shift (step 9).

As a further example of the potency of sign bit indexing, consider the application of the concept to a conditional jump instruction. In the standard jump instruction, the computer is instructed to take the next instruction in sequence or, alternatively, to take the next instruction from some part of the memory other than the usual sequence. The decision as to which course is to be followed will typically depend upon a comparison of stored values, e.g.,

Negative Jump - If accumulator is negative, take next instruction from address α . If accumulator is positive, continue in sequence.

Positive Jump - If accumulator is positive, take next instruction from address α . If accumulator is negative, continue in sequence.

Zero Jump - If accumulator is zero, take next instruction from address α ; otherwise continue in sequence. As evidenced by each of the above standard instructions, the programmer has a choice between two locations. With a sign bit indexing computer however, a four-way jump is available. Thus, assume that the jump

instruction provides that a jump is to be made to a new location β , if the contents of the accumulator is odd. Accordingly, a jump instruction stored in location ($\alpha - 1$) will cause the next instruction (α) to be taken in sequence or, alternatively, the next instruction will be taken from cell β depending upon the state of the least significant bit of the accumulator as indicated in FIG. 7a. Now, if accumulator sign bit indexing is employed, and if the low order bit is odd, the address β will be indexed by one if the contents of the accumulator is negative, i.e., the next instruction will be taken from memory cell β or $\beta + 1$ depending upon the algebraic sign of the contents of the accumulator. If, however, the contents of the accumulator is even, the next instruction will be taken from α or $\alpha + 1$ depending upon the sign bit of the accumulator. The four possible combinations may therefore be summarized as shown in FIG. 7b.

FIG. 8 shows a block diagram illustrating the various flow paths in a preferred arrangement for practicing the invention. The memory 2 is divided into eight pages. Each page corresponds to a word capacity of 512 words. Each word in the memory is nine bits. A complete instruction word for the machine is 18 bits but only the first nine bits are stored in the nine bit instruction register 14. The last nine bits specify the address of the operand - this information being stored in the next word in memory. The first bit I_1 of the instruction register 14 indicates whether or not the address of the operand is to be indexed. The second bit is used to designate the page in the memory, i.e., the present page or page one of the memory. The third bit is used for indirect addressing. The remaining six bits of the instruction register specify the command (i.e., add, subtract, multiply, etc.). The indexing feature requires three cycles (two fetch and one execute cycle) which may be described as follows:

During the first fetch cycle F_1 the memory address register is loaded (via line 33) with the address of the instruction to be executed. The instruction stored in this address is transferred to the memory buffer register 22. These nine bits specify the command portion of the instruction and are transferred during F_1 to the instruction register 14 as indicated by the path 34.

The contents of the memory address register 21, are also augmented by one at the end of the F_1 cycle via the adder 23. This operation is indicated by the paths 35 and 36. At this time the memory address register will thus specify the address of the second portion of the instruction.

During the F_2 cycle, the second part of the instruction (the operand address) is transferred to the memory buffer register 22 and then to the adder 23, as indicated by path 37. This address is then incremented by one via adder 23, if the index bit I_1 of the instruction register 14 is true (i.e., indexing is commanded) and if the contents of the accumulator is negative (sign bit = 1). Logic means 8 is typically an "AND" gate whose output is true only if A_1 and I_1 are both true. The result of this addition is transferred from the adder to the memory address register via path 36.

During the execute cycle E , the operand specified by the address stored in the memory address register 21 is transferred to the buffer register 22 and then to the arithmetic unit (as indicated by 39) for execution in accordance with the instruction command as decoded by the command decode logic (path 40). The program counter 24 is then augmented by two counts and the

next F_1 cycle is commenced by storing the address of the first portion of the next instruction in the memory address register 21.

The indexing feature may be used to perform almost any type of alternate choice operation. Furthermore, the sign of the accumulator may be taken to represent the sign of a number (as in the first example) or the value of a selected bit (as in the second and third examples). As previously mentioned, the indexing may be applied to numerous other instructions such as load, store, subtract, subtract outputs, inputs, etc. In computers containing indirect addressing capability (i.e., where the instruction is used to specify not the address of the operand, but rather the location where the address of the operand may be found), the indexing feature of the present invention may be employed to provide an alternative between different pointers of addresses of operands. In this application, the operands of interest may be stored anywhere in the memory and the instruction is used to effectuate the choice between two addresses which specify the actual operand locations.

Although preferred embodiments of the present invention have been shown and described herein, it is understood that the invention is not limited thereto, and that numerous changes and modifications may be made without departing from the spirit of the invention.

We claim:

1. A digital computer (comprising) having a control unit, a memory for storing instructions having a command field and an operand address field, and an arithmetic unit including an accumulator wherein the improvement comprises:

(an accumulator having) a particular bit position in said accumulator for representing the sign of information contained in said accumulator;
logic means for identifying the state of said sign bit;

instruction register means within said control unit responsive to the transfer of an instruction from said memory for storing the instruction currently being executed;

arithmetic means within said arithmetic unit responsively connected to said logic means for modifying the operand address of the instruction which is stored in said instruction register means and is currently being executed, based upon whether the state of said sign bit of said accumulator is a binary one or a binary zero.

2. In a digital computer (comprising:)

having a memory for storing instructions having a command field and an operand address field and having an arithmetic unit which includes an accumulator, the improvement which comprises:

(and accumulators having) a particular bit position in said accumulator for representing the sign of information contained in said accumulator;

instruction register means for storing the instruction currently being executed;

control means responsive to unmodified instructions for executing the instruction upon an operand specified by the address of the instruction irrespective of the state of said sign bit;

means for sensing the sign bit of said instruction register;

(control) arithmetic means responsively connected to said memory and responsively connected to said means for sensing the sign bit modified instructions, for altering the address where the operand of the instruction being executed is stored based upon whether said sign bit is a binary one or a binary zero.

* * * * *

40

45

50

55

60

65