Cascaded Variable Cycle Control as Applied to the 220 Computer

E. L. GLASER
NONMEMBER AIEE

THE subject of this paper is basically micro-programming as a design tool, the program of the meeting notwithstanding. This subject of microprogramming has received a great deal of attention in the last several years, primarily as a possible method of making the so-called general purpose computers more flexible. It has been propounded by some that microprogramming offers a technique whereby the user would be free to define, in some degree, the machine organization. However, microprogramming has other rather intriguing possibilities. Some of these have been realized recently at the ElectroData Division of the Burroughs Corporation.

Several years ago work was begun on new control techniques. One of these techniques that showed promise is the one under discussion. This method of control was first applied to the Datatron *220*. This control principle was employed on this project primarily to facilitate detail design and thus to shorten the production lead time. The method was not selected for economic reasons although it now appears that some economy of componentry has been achieved because of it.

The logical requirements for the microcontrol are similar to those of an interpretive program. A command must be fetched from the memory if necessary command arithmetic takes place, and the command is analyzed and executed. In the *220*, the model for discussion, all data is handled serial by digit, parallel by bit, except in the case of access to the parallel-parallel core memory and two broadside address transfer pads.

Fig. 1 represents the flow of data within the central processor and between this processor and the high-speed memory. The broadside shifts between the address buffer and the address register and program counter can be seen. The various microcommands available to the designer are those dealing with this configuration. The different data pads can be closed by means of gates. Any register may be shifted right, certain decades may be counted, and in the case of the adder-subtractor, either addition or subtraction can be ordered.

Since all data manipulation takes place a digit at a time, the same operation will often be applied to several digits which comprise the word or subword. Two types of pulses are used to accomplish this necessary repetition and nonrepetition of function. A digit pulse is used

E. L. GLASER is with Burroughs Corporation, Pasadena, Calif.

Acknowledgment is given to Lloyd Cali who was invaluable in the design of the *220* and for his original thinking in this basic principle of control.

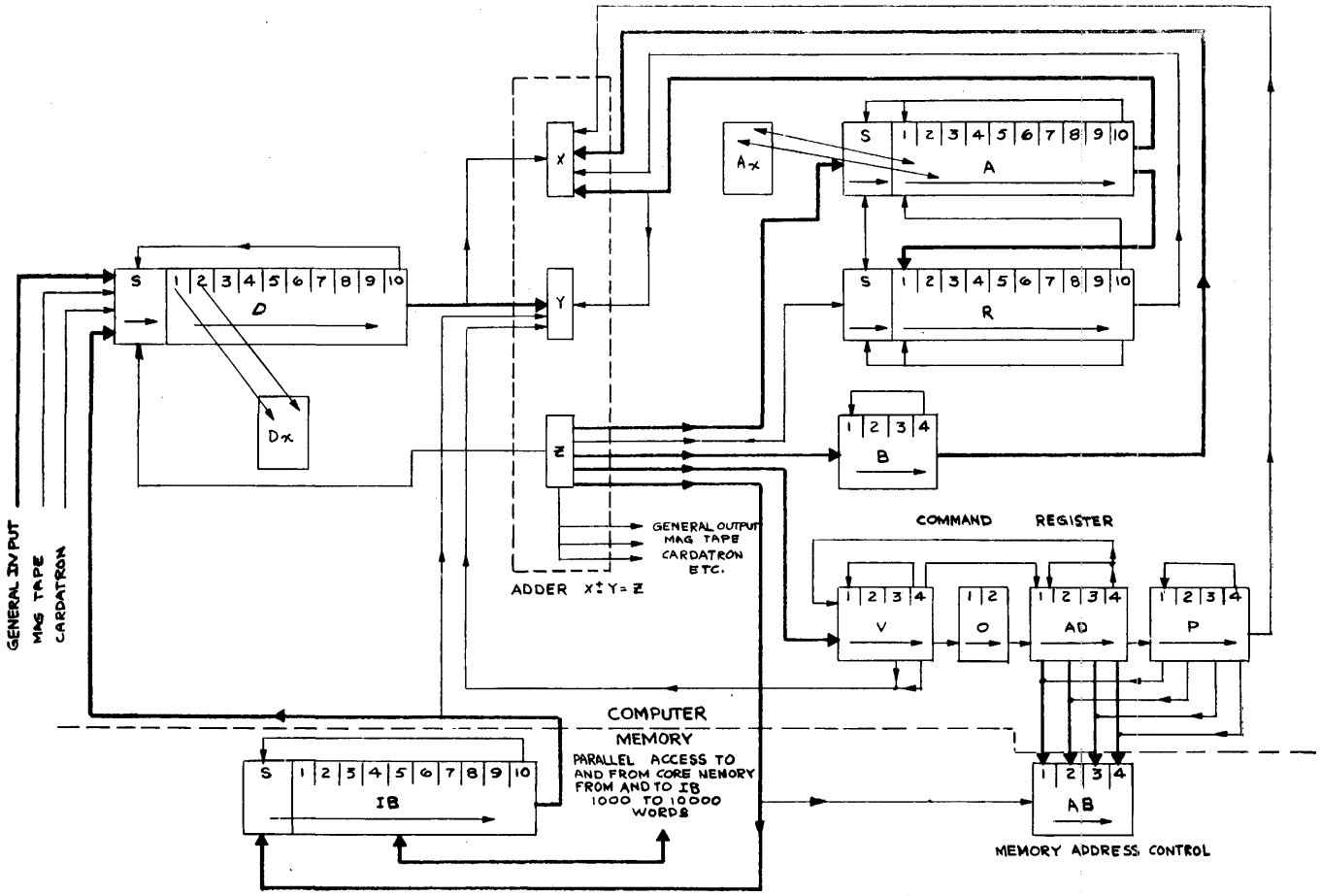

Fig. 1. Flow chart of the 220 computer

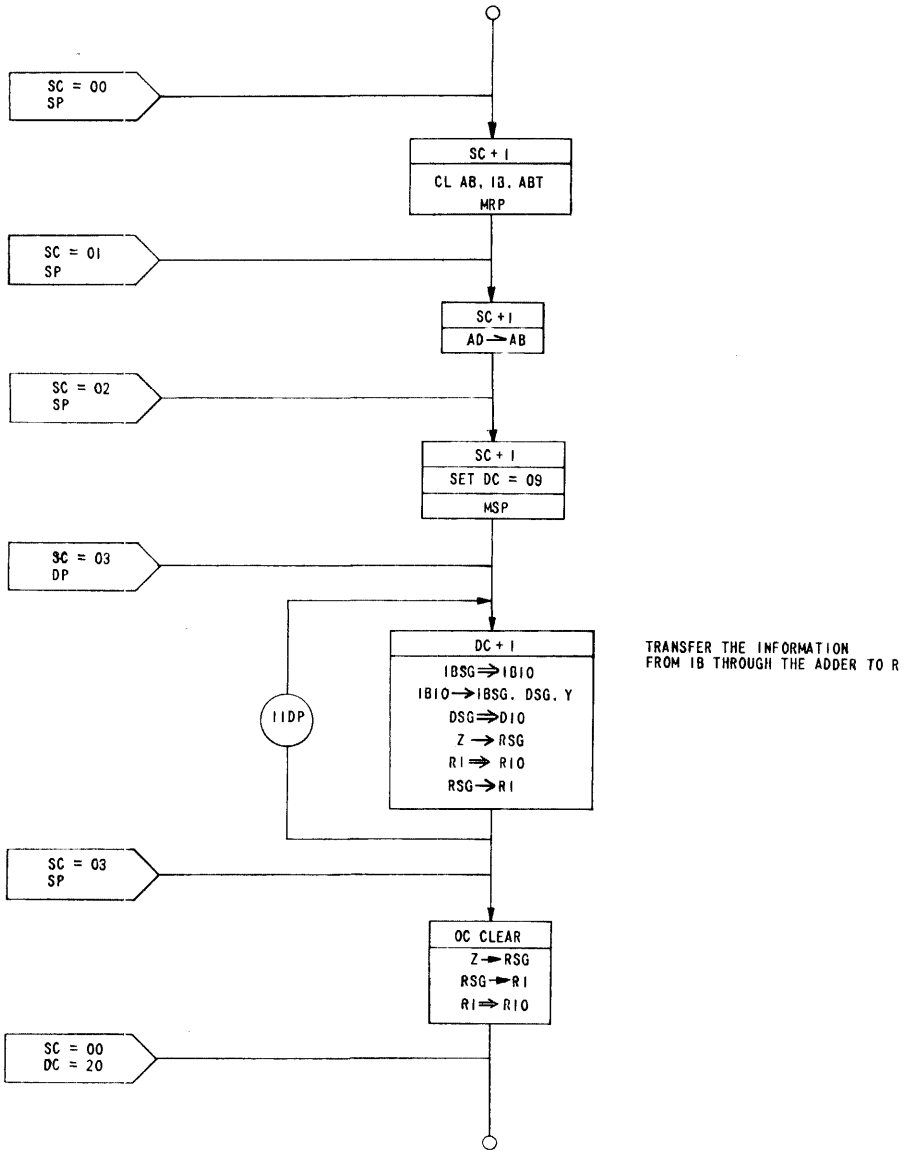

Fig. 2. Flow chart of the load R command

to apply the same operator to a number of digits. The distribution to the several digits is by means of shifting the contents of a register. Groups of these digit pulses are separated by sequence pulses. The sequence pulses may be thought of as the decision-making pulses. Any number of digit pulses may be in a group, as a consequence, the concept of word time has no meaning even though the *220* is a fixed word-length machine.

A digit counter is used to count the number of digit pulses in a group. This counter is designed to seek a terminal value which is, in the *220*, equal to the number 20. When this counter is set to some value less than 20 it will immediately start counting in a positive sense until the value of 20 is reached. A terminal value of zero could have been used in conjunction with a decreasing count; however, the present method was picked because of easier training. This counter can, in any event, be used to define both sequence and digit pulses. Digit pulses occur, by definition, when the digit counter is less than 20. These pulses are used to increase the counter in order to meter the digit pulses. Sequence pulses occur when the counter is equal to 20, the terminal or rest state.

A second counter is employed in the micro-control for the purpose of defining the subcycles in any operation. This sequence counter in the *220* is a 4-stage, binary counter taking on all values from 0 through 15. This counter is set and counted by sequence pulses only. Thus, any sequence pulse is defined by the setting of the sequence counter and in turn, defines the next setting of the sequence counter. A digit pulse is defined by the setting of the sequence counter. The setting of the digit counter is not cited here since its purpose is to meter the number of digit pulses. Its initial setting for each group of digit pulses is the only value of interest and this setting is made and defined by a sequence pulse. As a result, the same pulse that sets the sequence counter sets the digit counter.

The contents of the sequence counter is not in itself sufficient to define the set of micro-operations to be carried out by the associated digit and sequence pulses. It is therefore necessary to have some method by which the coded machine order can be related to the microprogramming cycles, or subroutines. This need is satisfied by the mechanism of a standard-order decoding matrix affixed to the order portion of the command register. During command execution, the output of this matrix defines the order and is gated by the output of the sequence counter to define the suborder. The three functions of digit counter, sequence counter, and order matrix could have been combined; however, the attending logical complexity was too fearsome to contemplate for long. The maximum values for the contents for both control counters was picked as minimal for the *220* system.

Fig. 2 is a flow chart of the load *R* command. This command loads the *R* register from memory. The first pulse *SP* 0 is common to all commands and merely clears the debris from the previous fetch. The second pulse sets up the address in the address buffer and the third pulse orders a memory read. All of these pulses are sequence pulses and each defines the next settings of the control counters. So far the digit pulse groups have been of length zero. *SP* 2 also sets the digit counter to 09 which is 20 minus 11. Since 11 pulses are needed to shift the word to the *T* register, this setting is the correct one. The final *SP* is used to shift the *R* register into proper position and at the same time all control imaginable is cleared, or otherwise disposed of.

It can be deduced from this explanation, hopefully, that some sort of timing flip-flop is still needed to differentiate between the two basic states of a classic stored program computer. When this flip-flop is in the fetch state, the output of the order matrix is blithely ignored and the sequence counter and the timing flip-flop alone control the fetch cycle. During execution this flip-flop is in the opposite state. This state is changed at the end of each half cycle of the computer. The fetch state is also used during the control of

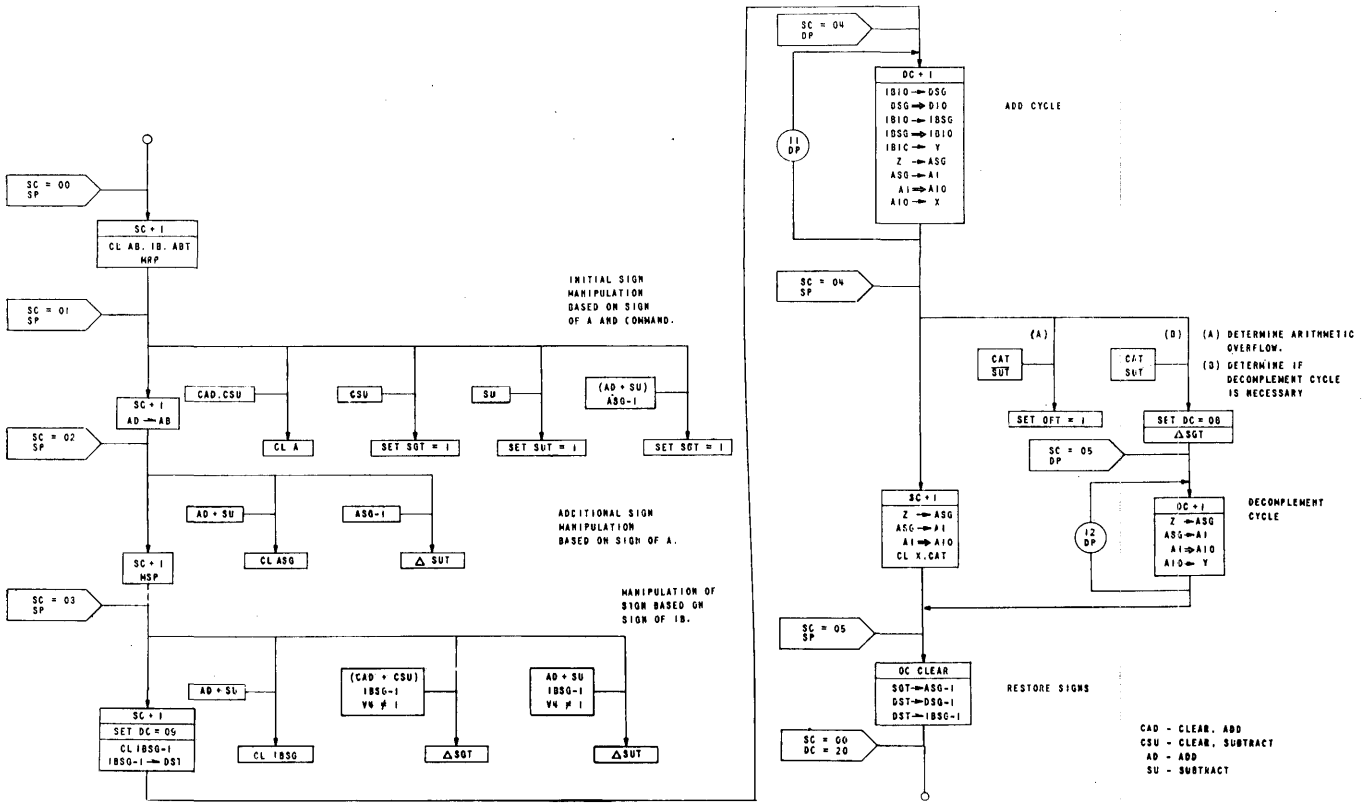

Fig. 3. Flow chart

cycles from the console. Micro-operations are not, and should not be, dependent on the contents of the order register.

A number of extraneous flip-flops are still needed in the control section to store binary decisions. Several are there because of the command list. These include the overflow and compare indicators. Some are there because of the nature of the universe. The carry flip-flop is a good example. One or two are used to make life easier for the designer. An example of this class is the subtract flip-flop. This controls the adder-subtractor. In all cases, combinational logic could replace this flip-flop; however, the presence of this flip-flop makes possible the use of sequential rather than combinational logic at a reasonable saving in complexity. Since the data for controlling this flip-flop are available only in a sequential fashion, it follows that no loss of performance is encountered.

A second example of command control is the fixed point add group. Two types of decisions must be made in this group. The entire group of fixed point add commands are compressed into one cycle so that the exact micro-order to be followed is the usual function of the sequence counter and the order matrix. The sign control also enters into the choice of micro-order. A second class of decision is that which is necessary after a complement addition. If the result of the addition is in complementary form, a decomplement cycle is ordered. In all other cases this cycle is skipped. This decision is made by conditional setting of both the digit and sequence counter. (See Fig. 3.) In the case in point, only the digit counter is involved. However, the conditional setting of the sequence counter is prevalent.

An analogy may be drawn between this type of microprogramming and normal coding. The machine order to be designed is the subroutine name. The setting of the sequence counter is equivalent to the step number. The micro-orders are equivalent to the normal commands in programming. The digit counter is similar in operation to a repeat command as used in some machines. The use of this format for command decision has already proved its worth in making it possible to employ young, relatively untrained engineers in the detail stages of design. This also serves as a good basic training ground for these young engineers and gets them into design that much faster.

The translation from the flow chart to a print is not as formidable as one might think. The use of data processing on the flow charted information can group inputs to circuits and produce loadings on outputs. A set of circuit and wiring lists can be produced and from these to a print is straight forward although tedious.

This method of design is far from a panacea. Its main disadvantage is that it does not facilitate optimization of circuitry. Its prime value is in the speed that can be realized in the early detailing of a large system.