A FLOW diagram approach to the logical design and mechanization of computers and other digital equipment has served as a basic design method for the formulation and construction of a medium-size data processor, the NCR 304.

The flow diagram method used by the National Cash Register, (NCR) Electronics Division resulted from an application to logical design of techniques that the computer programmer uses in developing a computer routine. That is, the programmer's method of developing a routine by constructing a flow diagram which denotes the desired sequence of computer operations necessary to the execution of a command. In the same way that a programmer may design a flow using only a general knowledge of a computer’s repertoire, it was recognized that a logical designer may lay out a command flow, having in mind only the general character of the functional units of the machine he is designing.

Once the entire command list is represented in flow diagrams the designer is then able to specify in detail the set of interconnections the computer must have, just as the programmer details the set of instructions which the blocks in his flow represent.

To discuss these concepts in more concrete terms, it is desirable to examine the NCR 304 in some detail. Table I displays the more pertinent characteristics of the 304 system. Fig. 1 is a block diagram of the 304 central processor, indicating the major functional units of the machine.

There are:
1. The main store of ferrite core memory with a flip-flop buffer register, M, which also serves as a working register.
2. A second working register, S, which is used for shifting and for intermediate storage.
3. An arithmetic unit composed of an adder and several flip-flop counters.
4. A memory address flip-flop register, L.
5. An auxiliary memory address flip-flop register, A.
6. A control unit consisting of an instruction register, J, a cycle or sequence counter, N, a decoding matrix, and a bank of operation amplifiers which furnish the outputs of the control unit.

Given the functional block diagram, the logical designer has two problems:

1. To describe, usually in Boolean algebra, what interconnections must be made between the various blocks in the functional diagram.
2. To tabulate for each computer command the time-sequence of interconnections which will produce the desired results.

In practice the designer must propose a set of functional blocks, devise a set of tentative interconnections, establish flow diagrams for those commands which would seem to have the least in common, and then proceed to lay out the remainder of the command flows, modifying the interconnections where necessary and even adding functional units to the block diagram.

Returning to the 304 Fig. 1, to illustrate some of these ideas it is necessary to describe the computer’s cycle or word-time. The word-time consists of a fixed sequence of clock intervals which are defined by the pulse counter. These intervals are allocated to:

1. Control set-up,
2. Memory read,
3. Logical manipulation of operands,
4. Memory write,

in the order listed.

The control set-up results in the

Table I. NCR 304 Data Processing System Characteristics

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>File Storage</th>
<th>Computer Structure</th>
<th>Command Types</th>
<th>Memory</th>
<th>Circuitry</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Punched</td>
<td>Paper tape</td>
<td>Magnetic</td>
<td>3-Address</td>
<td>Arithmetic</td>
<td>Coincident-time, erite</td>
<td>Transistors for counter and address decoding and for the Adder</td>
<td>A flow diagram approach used in description and preparation of computer control</td>
</tr>
<tr>
<td>paper tape</td>
<td>punch</td>
<td>tape file</td>
<td></td>
<td>Logical</td>
<td>20,000 or 40,000 character capacity</td>
<td>Transistorized Eccles-Jordan flip-flops</td>
<td>Each command level realized by the incorporation of a set of magnetic cores corresponding to the set of blocks in the command flow</td>
</tr>
<tr>
<td>photo-electric reader</td>
<td>Line printer</td>
<td></td>
<td>Serial by character, parallel by bits</td>
<td>Decision</td>
<td>Data processing</td>
<td>Transistors for amplification and inversion</td>
<td>Each core threaded by those operation lines which must be activated to realize the logical operation in that block of the flow</td>
</tr>
<tr>
<td>Punch card</td>
<td>Magnetic</td>
<td>Magnetic tape file</td>
<td></td>
<td>Editing</td>
<td>20 microseconds read-write cycle</td>
<td>Germanium diodes for all interconnecting gates and for clamping</td>
<td></td>
</tr>
<tr>
<td>photo-electric reader</td>
<td>tape</td>
<td></td>
<td>Partial word addressing</td>
<td>Data processing</td>
<td>60 bit parallel accessing</td>
<td>Molybdenum permalloy ribbon cores for logical decoding and summing of control matrix outputs</td>
<td></td>
</tr>
<tr>
<td>Magnetic tape</td>
<td>Electric</td>
<td>typewriter</td>
<td>Automatic relative addressing</td>
<td>Tape file</td>
<td>Drum file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electric typewriter</td>
<td></td>
<td></td>
<td>Automatic linking on subroutines</td>
<td>Input-output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Automatic program monitoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hudson, Edwards, Eckdahl—Computers Logical Design 59
establishment of the set of operations which are to be used in this word-time. These are simply the logical gates which are to be active during the read, logic, and write intervals.

The core memory is then accessed at the cell specified by the address register, or, under certain control conditions, at one of a number of special working cells, M1, M2, M3, etc.

The logical manipulation may take the form of a copying of the memory buffer, into the working register, S, or a combining of M and S through the adder; or a transfer of a portion of M to the A-register. Or several of these operations may be carried out at the same time. The simultaneous manipulation of data in several registers is a common occurrence in the mechanization of the 304.

At the completion of the ten clock intervals of the logic period, the memory write circuitry copies the contents of the memory buffer, M, into the cell which was cleared during memory read. And the cycle starts again.

Consider the flow diagram, Fig. 2, which is very nearly the add flow for the 304. Each block in the flow has two aspects. One is that the block represents a fixed interval of time, the word-time. The second is that each block is a unique control configuration which implies a unique combination of operations of active logical gates.

In block 0, the control selects a special memory cell, M1, in which is kept the address of the next instruction. This address is transferred from the memory buffer to the memory address register, L, and to the auxiliary address register, A1, during the logic interval. Also during this interval the adder is used to augment the address which is to be restored to the cell, M1, for the next instruction address.

In block 1, the first instruction word, which contains the instruction code and three addresses, is read from the cell selected by the address register, L. The instruction code is transferred to the instruction register, I; the three addresses are copied from the memory buffer into the S register; and the auxiliary address register, A, is augmented by one. The memory cell is then restored with this first instruction word, and as this block is terminated, the augmented address from the A register is copied into the L register in parallel.

It is then possible in block 2 to access the cell containing the second instruction word. This word contains partial word extractors which are copied into counters in the arithmetic section, a relative address designator which is copied into the A register, a monitor code which is compared with certain console switches, and a mode digit which is copied into an extension of the instruction register.

As this word is restored to its memory cell, the relative address cell location in A is transferred into L in parallel.

The computer then advances to block 3 where the memory cell specified by L is accessed. The resulting contents of the memory buffer are added to the S register, producing absolute addresses for command execution.

In block 4 the two operand addresses, a and b, are transferred from the S register to L and A, and the putaway address, c, is transferred to the memory buffer from which it is copied into a special working cell, M2, for later referral.

These five blocks which have just been described are almost identical for all command levels in the 304. This is the process by which the machine's registers are set up for command execution.

Before proceeding to the remainder of the add flow, it is instructive to summarize the characteristics of the machine which have appeared in these first blocks.

Fig. 3 is a more detailed flow chart with a shorthand notation for indicating the operations which are to occur in each block.

Notice that certain operations have been used several times in these first 5 blocks, e.g., M-select, L-register, M→A, A→L.

It is also characteristic of the 304 that the adder is used quite frequently for bookkeeping, such as augmenting the command location as in block 0, augmenting the A register as in block 1, and for adding the relative addresses to the index cell as in block 3.

Finally, there are a total of 48 distinct and independent operations in these first 5 set up blocks in the actual 304 flow. These 5 blocks for all commands amount to 1/5 of the total number of blocks in the machine, and require 1/10 of the 480 available machine operations.

Referring again to Fig 2, the a and b operand addresses are held in L and A respectively in block 4. In block 5, the cell containing the a field is accessed, and a is extracted from M into S under the control of the counter containing the partial word designator for a.

In block 6, the a field is right-justified as it is transferred into the working cell, M2. The putaway location, c, which was stored here temporarily in block 4, is transferred to the A register, and the b address in A is copied into the L register. The a address is discarded.

In block 7, the b field is read into M and extracted into S under the control of the counter containing the partial word designator for b. The signs of both operands being now available are compared, the comparison having three possible results:

1. Signs are the same, implying no complementing of either operand is desired.
In Fig. 2, Add flow

2. a is positive, b is negative implying that b is to be complemented as the addition occurs.

3. b is positive, a is negative implying that a is to be complemented as the addition takes place.

The flow from block 7 illustrates graphically the action that the machine control must take for each of these eventualities. It will advance to block 8 if the signs are the same; it will skip to block 9 if a is positive, b negative; it will skip to block 11 if a is negative, b positive.

This skipping ability which the machine control has allows the logical designer to specify as many as three alternate arbitrary jumps from each block in a flow as well as the ability to count to the next block. These jumps or transformations of the control counter are another result of the operation lines furnished by the control section. In effect, the control establishes those operations which are to occur in one block time, and the numbers of those blocks which may follow, one of which is chosen by decisions made within the block time.

This facility also makes it possible to mechanize iterative processes with the same sequence of blocks by testing for a certain condition at the end of the loop, and either continuing or going back to the start of the sequence.

In Fig. 2, if the signs are the same, the control proceeds to block 8 and the b operand is right-justified as it is added to a which has been copied from M2 into the M register. The resulting sum in M is restored to M2. From block 8 an un­conditional jump is made to block 13.

In block 13, M2 is copied into the M register and then is left-justified in S in preparation for putaway. The C address is transferred from A to L.

In block 14, the C cell is copied into the M register, and an insert type of operation transfers the results from S into the allocated portion of M. M is then copied back into the C cell location, and the control returns to block 0.

If the a sign is positive, but b negative the control goes from block 7 to block 9. As the addition takes place, the b input to the adder is complemented. Also the adder "carry" indicates at the end of the operation whether the result is valid or an uncomplementing operation is required. Here again the control establishes a possible skip to block 13 or a count to block 10 for uncomplementing.

A like procedure occurs in blocks 11 and 12 for the case of a negative, b positive.

The add command requires then 15 possible control configurations for its execution. Each add command specifies the following actions:

1. Augment the a, b, and c portions of the first instruction word by the corresponding a, b, and c portions of index cell P, to obtain the absolute addresses for operands and putaway.
2. Extract and right-justify operand a.
3. Extract and right-justify operand b.
4. Obtain sum of a and b and left-justify.
5. Insert results into putaway cell.

**Fig. 3. Detailed flow diagram**

_Hudson, Edwards, Eckdahl—Computers Logical Design_ 61
A total of 82 operations are required to effect these steps. There are 40 instructions in the 304 of which approximately 1/3 are relatively simple commands similar to the add flow. Another 1/3 of the commands are for data processing and editing. And a final 1/3 are input, output, and tape file commands with extensive bookkeeping and branching requirements.

Turning to the mechanization of the control for this rather formidable command list, a matrix of ribbon cores is used to decode the instruction register and control counter into the individual blocks of a command flow. The matrix consists of a 40 × 40-array, but with cores wired in only for the blocks actually used. As an example, there are 40 intersections on the add command level, but only the first 15 have cores.

The matrix has a bias winding which holds all cores in a "minus 1" state. One of the 40 lines decoded from the instruction register will carry a current sufficient to cancel the bias in those cores which it threads. Then as the control counter changes from state to state, its decoded outputs will, one by one, turn over the cores at the intersections with the instruction line. Hence, as the control counter changes, a change of flux occurs in the core corresponding to the new state, and this flux change must activate those operation amplifiers which are to be used during that block time. For this purpose each operation amplifier is linked to all those cores for which that operation is required. As an example, Fig. 4 shows 5 cores which represent the first 5 blocks of a command set up as discussed previously. There are 20 sense windings in this portion of the array corresponding to the 20 distinct operations which were listed in Fig. 3. Some of these only thread one core of this section; others thread 2 or 3 cores. Each sense winding eventually terminates on a latching amplifier which will be activated if any core on its winding is turned over.

In the 304 matrix, there are approximately 900 cores, with an average of 15 sense windings threading each core. There are 480 sense windings emanating from the matrix, each terminating on a latching amplifier which activates a set of logical gates by supplying proper voltages to the product resistors of these gates. One would imagine that 480 operations would imply a certain redundancy. This is actually the case and arises for two reasons. One is the difficulty in optimizing a design of this complexity; but more important is the necessity of minimizing command execution time. This latter reason accounts for most of the redundancy in the operation list of the NCR 304.

Another implication of this large operation list is that given this list and given the requirement of a new command, a logical design can be prepared for this new command which will probably not require any additions or changes to the functional units other than the installation of a core plate wired for the new command.

In summarizing, a little thought leads one to the following conclusions about the 304 command mechanization. In executing a design from specifications involving rather complex commands, a flow diagram approach results in a presentation of the types of interconnections and the sequence in which these interconnections are made in the various command levels. This presentation shows the compromise which the designer made between execution time and minimization of equipment.

The control mechanization is nearly optimum for the 304 command list for three reasons.

1. Flexibility in modifications of the machine is achieved by specifications of different wiring patterns for the sense windings in the control matrix.
2. Additional command capabilities were more easily included than would have been the case with previous types of control because these were achieved by the addition only of cores to the matrix, and modifications to the sense windings' paths.
3. The logical design of commands such as multiply, divide, merge, sift, summarize, which have an iterative character are more efficiently executed using the "count" and "skip" features to effect these iterative loops.

Finally, it is felt that the flow diagrams are one of the most easily understood methods of machine description and facilitate a rapid training of maintenance personnel.
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 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 which comprises 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