1968

Structure and organization of a pattern processor for handprinted character recognition

Roy James Zingg
Iowa State University

69-4295

ZINGG, Roy James, 1931-

STRUCTURE AND ORGANIZATION OF A PATTERN PROCESSOR FOR HANDPRINTED CHARACTER RECOGNITION.

Iowa State University, Ph.D., 1968
Engineering, electrical

University Microfilms, Inc., Ann Arbor, Michigan

STRUCTURE AND ORGANIZATION OF A PATTERN PROCESSOR FOR HANDPRINTED CHARACTER RECOGNITION

by

Roy James Zingg

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Ames, Iowa

1968

TABLE OF CONTENTS

INTRODUCTION

A computing system is in reality an information processor. Input information is presented to the system, transformed according to certain specified or partially specified rules and finally emerges as output information. The total information processing system includes not only the computer and its input/output devices, but also the people and machinery required to convert the input information into a form acceptable to the input devices. The central design goal in designing an information processing system should be to minimize the total cost of processing information. For a typical computing center these costs would include problem formulation and programming costs by the problem orginator and any professional programming help, machine rental or purchase costs and center operating expenses.

During the past fifteen years the most dramatic change in the cost picture in the computer industry has been the decrease in electronic cost on a per logical decision element basis. The actual electronic cost per logical decision element has gone from about two dollars to about twenty cents during this period. Projected cost for the early 1970's is a few cents per logical element (1). This cost decrease has been accompanied by increases of several orders of magnitude in both speed and reliability.

In contrast to this optimistic trend, other costs, particularly people cost, have increased during this same time period. An analysis of the cost picture of a computing center with a rather wide problem mix shows that if the cost of the logic and control circuitry in the computing

system were reduced to zero, the decrease in total information processing cost would be less than two percent (2).

If the total cost of processing information is to be minimized, perhaps one method of attack is to try to apply these inexpensive logical circuits to devices and systems which tend to minimize the people cost. Considerable work has been done in several areas pursuant to this goal. Efforts have been made in the area of directly implementing high-level languages in hardware (3, 4). The potential gain here is to replace a large part of systems programming packages with hardware. Hopefully, the total information processing cost would be decreased.

Considerable time, money and effort have recently been expended on multiprogramming and time-sharing systems (5, 6, 7). The pertinence of such systems in this discussion is that they will, at least potentially, help reduce the total cost of information processing by providing increased machine aid in the problem formulation and debugging phases of information processing. This is especially true in highly interactive systems (8, 9, 10).

A sizeable fraction of information processing cost in some computing environments is the cost of converting programs and data to a machine readable form. For example, on a typical university campus thousands of student originated programs are run per month. The cost of providing key punches and key punch operators for these one-time programs is substantial. A handprinted document processor in such an environment would be of considerable value.

Scope of Investigation

Three areas have been mentioned in which the work being done may tend to minimize information processing cost. Even with relatively low logic cost the first two research areas require substantial commitments of funds if processing hardware is to actually be built. The third area, that is a handprinted character recognition system, shows promise of being tractable in a research environment where funding is limited, but hardware as an ultimate goal is still required.

Since the problem of the recognition of handprinted characters is not solved it would be naive to undertake the design of hardware to implement some specific recognition algorithm. However, it might be feasible to build a special purpose system intended to be used as a tool for investigating various recognition algorithms. To be effective this system should include a document reader to digitize input information, a special purpose computer for the recognition function and auxiliary input/output equipment. Such a system is being conceived, designed and built as a research project, and is to be used for the purpose indicated.

This investigation is part of this research project. Its scope is to investigate possible system organizations and to arrive at a system organization for the special purpose computer in the experimental handprinted character recognition system. The resulting computer is to provided a flexible tool for the investigation of various handprinted character recognition algorithms, and at the same time be a near minimum cost machine.

REVIEW OF OPTICAL CHARACTER RECOGNITION

Optical character recognition is generally considered to be part of the more general pattern recognition problem. In the general pattern recognition problem a new pattern may or may not belong to one of the pattern classes to be recognized. However, in optical character recognition it can usually be assumed that the character being processed is some member of the allowed character set. Consequently, the problem of optical character recognition essentially consists of two steps. First, significant features of the character being processed are extracted and then these features are used in a decision procedure to determine to which class the character belongs. Of course, the non-trivial job of scanning the character and converting the visual information to some electrical form must also be done. Over the years there have been a variety of approaches to these basic steps of optical character recognition (11 - 29).

Some Approaches to Optical Character Recognition

The decision as to what approach to take in this problem area has usually been influenced by the specific job to be done. When the text to be recognized consists of a single machine written type font, template matching has been used successfully. When characters from several type fonts or handprinted characters are to be read investigators have tended toward extracting properties of the character which tend to be invariant for the expected variations in the character styles under consideration (12).

Sets of properties extracted has varied widely. Holt (13) suggests a curve tracing technique to extract features. Liu and Shelton (14) apply a series of spatially arranged groups of points called N-tuples to the character to obtain pertinent feature information. Bomba (15) extracted information about line direction and intersection orientation. This might be called local feature extraction. Doyle (16) also extracts local features, but the tests are quite different than those of Bomba. Unger (17) looks for various types of cavities and holes. Again these might be called local features. Alt (18) suggests the use of moments for feature extraction. The primary advantage of this method is that these properties are less sensitive to rotation, noise and slight variations in style than some features. Finally, Bledsoe and Browning (19) consider randomly-generated operators to obtain relatively invariant features of alphanumeric character patterns.

After the features have been extracted a decision procedure must be executed to classify the character being processed into one of a preassigned number of classes. The complexity of this task depends on the quality of the features previously extracted, the variations permitted in the characters and the number of preassigned classes. Consequently, investigators have used a variety of procedures to classify characters. Decision trees have been used (15, 17). In at least one case, a decision is made after each test in an attempt to minimize the number of tests required (20). Others suggest linear or piecewise linear decision functions (21, 22, 23). Liu and Shelton (14) base the decision procedure on a minimum distance linear classification scheme.

Adaptive Pattern Classifiers

Some of the classification schemes already mentioned are also adaptive (16, 21). That is, the decision criteria are adjusted based on experience with some of the characters to be recognized during a learning phase. Others also propose adaptive pattern classifiers (24, 25).

On-Line Character Recognition

Another variation of optical character recognition schemes for handprinted characters is on-line recognition (26, 27, 28, 29). This technique has seemingly gained favor as graphical input devices have become available for multiprogramming or time-sharing systems. On-line recognition has several advantages over static recognition systems. Sequence information of positions traced is available, and the strokes of the writing stylus, as signalled by the removal of the stylus from the writing plane, provide additional information. This additional information makes only crude measurement of stylus position necessary; whereas in static recognition inconsistancies in stylus position usually make necessary the measurement of secondary properties. This implies dividing the character plane into a relatively large number of regions. Further, if the recognized characters are displayed on a graphic output device any recognition errors can be corrected immediately by the user.

Commercial Optical Character Recognition Systems

There are a number of commercially available optical character recognition systems. The characteristics of these systems are summarized in several publications (30, 31, 32). Some of these systems will read

any of several type fonts while others will handle only a particular font. None of them will currently read a fairly complete alphabet of handprinted alphanumeric characters. One company has announced, but not delivered, a module to be added to an already expensive, existing system that will read a forty character alphabet of handprinted characters.

SYSTEM RATIONALE

Most of the processing done in investigations on handprinted character recognition has been done on general purpose computers. Even where special hardware has been proposed it has often been simulated to a large extent rather than actually built. This lack of hardware has influenced some investigations. In some cases the approach taken was determined by the type of computing general purpose computers do reasonably well. In other cases the difficulties in digitizing input data without special readers or scanners forced the investigator to test only small data samples. Special purpose equipment should facilitate investigations in this area providing it has appropriate characteristics and achieves certain objectives.

Objectives

The primary objective set for the handprinted character recognition system proposed here is that it serve as a research tool for experimental recognition algorithms. Consequently, it must be possible to implement a variety of algorithms, including adaptive recognition algorithms, and execute even complex algorithms at the designed-for character rate. Assuming an acceptable recognition algorithm, the system must have the capacity to process a reasonably large character set.

Another objective is to accomplish the primary objective at a reasonable cost. Consequently, the designed for character rate was set at 100 characters per second with processing to be done in real time. Actually the character rate is not critical for an experimental system.

This rate was chosen because it is an order of magnitude greater than keypunching, but still low enough to keep processing speed requirements reasonable.

Finally, an additional objective was set to help keep the system practical from a use point of view. The documents to be sensed are to be handprinted characters on a modified coding sheet. No special pencils or other special equipment is to be required. This last objective bears more on the design of an acceptable algorithm and on the design of a document reader than on this specific investigation. However, it is stated to clarify the intent of the overall program.

Factors Affecting System Organization

There are two factors that have had major influence on the organization of this system with a third modifying factor that has exerted influence in many of the choices made. The first is the choice of packaging technique, and the second is the type and environment of the expected usage. The modifying factor has been the desire to keep the cost of the system down.

Implications of packaging technique

Early in the development of this system the decision was made to implement it by using large two-sided printed circuit boards that plug into another two-sided printed circuit board. This interconnect board permits 162 parallel runs. All system interconnections are to be made with these 162 lines, with the possibility of breaking runs between boards to gain a limited number of additional interconnections. This scheme was developed by the Digital Research Group of Fairchild

Semiconductor (2). Their cooperation and the elegant simplicity of the technique made it a very economical choice for this system. This packaging decision was important since it implicitly forces other system decisions.

Interconnections With a limited number of interconnections a bus oriented system becomes attractive. That way many potential connections for the movement of data can be established with relatively few, regular interconnections. Also, the distribution of control information in encoded form is favored. This means that the control information must be decoded locally wherever needed. Local decoding increases the amount of logic required. However, the basic packaging technique is good from the point of view of minimizing the number of connector contacts and the number of point to point interconnections. Consequently, the net cost of local decoding is substantially less than the cost of the additional logic required.

An acceptable way to provide timing information throughout the system compatible with the packaging technique is to time each section of the system independently. The several sections of the system are then permitted to interact asynchronously with appropriate initiation and completion signals generated. When a basically synchronous device such as the document reader requires attention it is to be handled on an interrupt basis.

System partitioning The way a system is partitioned is extremely sensitive to the amount of logical capability that is available per board. Epoxy RTL integrated circuits were chosen as the basic logical building blocks in this system for primarily economic reasons. One

package of this family of integrated circuits contains either a flip flop, a pair of two-input NOR gates or a buffer. The two-sided printed circuit boards used in the chosen packaging scheme are approximately 12 inches by 14 inches. There are 484 positions for RTL integrated circuit packages with 300 packages per board being a reasonable practical limit. To give an indication of the amount of functional capability that this many integrated circuits corresponds to, an eight bit, ripple carry, binary adder can be implemented with 56 integrated circuits.

This amount of functional capability in itself would permit a variety of system partitions. However, the way in which a system is partitioned affects the number of interconnections, and also the amount of control logic in a system with local decoding. Fortunately for this system the partitioning corresponding to the minimum number of interconnections was also a satisfactory one for the amount of control logic required. Generally in this system the best partitioning corresponds to putting an entire register on a board rather than putting parts of several registers on a board.

Expansion capability In an experimental system such as this it is highly desirable to be able to expand or perhaps modify the system. The simple bus-oriented interconnection pattern with local decoding and functional partitioning are ideally suited to both expansion and modification. Additional commands can be decoded and any additional logic or temporary storage they imply can be added on additional boards and simply plugged into spare slots. Modifications in the existing structure

can usually be accomplished by replacing a board with a new board. Doing the modification in this way allows the system to be restored by replacing the new board with the old one. Naturally the extent of additions must be tempered by the number of spare slots available, power supply capacity and bus fan-in and fan-out capability.

Implications of usage

The intended use of the system proposed here is as a research system, that is, a tool for investigations of experimental recognition algorithms. Both the research environment and the specialized processing have influenced the organization of the system.

Experimental function Eventually a satisfactory recognition algorithm might be hard wired in a handprinted character recognition processor. Some of the processing might be done in an analog sense rather than digitally. However, the intent for this system is to provide the capability to experiment with various character sets and various recognition algorithms. Consequently, a NDRO (non-destructive read out) control memory was chosen as the means of implementing the experimental algorithms. The thin film control memory is electrically alterable with a capacity of 2048 20-bit words and a cycle time of 250 nanoseconds. The question of adequate capacity of this memory will be discussed later in this document after several programs are discussed. The algorithms will consist of a series of subroutines stored in the control memory. The commands making up the subroutines will be the machine language commands of the system.

Certainly an alterable memory is also necessary. Such things as

the character array being processed, the character being read, parameter information on the character set, and a macro program that selects an appropriate set of control memory subroutines must be stored and be modifiable. A 256 word, 64 bits per word, DRO (destructive read out), thin film memory with a cycle time of 250 nanoseconds was chosen for a scratchpad memory. It will take about 25 words for the two character arrays that must be preserved and 64 words (or more) for character set parameter information. Additional storage will be required for macro program and temporary storage for various purposes. The 256 words of scratchpad memory is quite feasible from an engineering and economic point of view, and appears to be adequate from the system point of view. However, since predicting required scratchpad memory capacity is difficult the addressing capability to permit doubling the scratchpad capacity is provided.

Specialized processing Many of the operations to be performed on the character arrays and on the feature information can be reduced to such operations as counting, thresholding and comparison. As a consequence the system organization especially favors such logical operations. An attempt was made to provide general logical capability without excessive cost while maintaining a reasonable processing speed.

On the basis of other investigations reported in the literature and on the basis of economic feasibility, it was assumed that the character being processed would be represented by an array 32 bits high and 24 bits wide. The document reader to be associated with this system is predicated on this decision, and certain system decisions were also based on this particular grid resolution. However, in the development of the system

organization particular care was taken not to preclude other array sizes. Finer resolution would imply longer processing time and coarser resolution would imply shorter processing time.

The main system decision that was partially based on the assumed array size was the degree of parallel processing implemented. The particular choice was based on several other factors as well. Some of the additional factors are effect on interconnections required, effect on system cost and effect on processing speed. The decision was made to provide enough parallelism to process eight bits of the array at a time.

SYSTEM STRUCTURE

In the previous section system objectives were stated and reasons were presented for making certain system decisions. Here, the structure of the resulting system will be discussed, and justifications for particular decisions given.

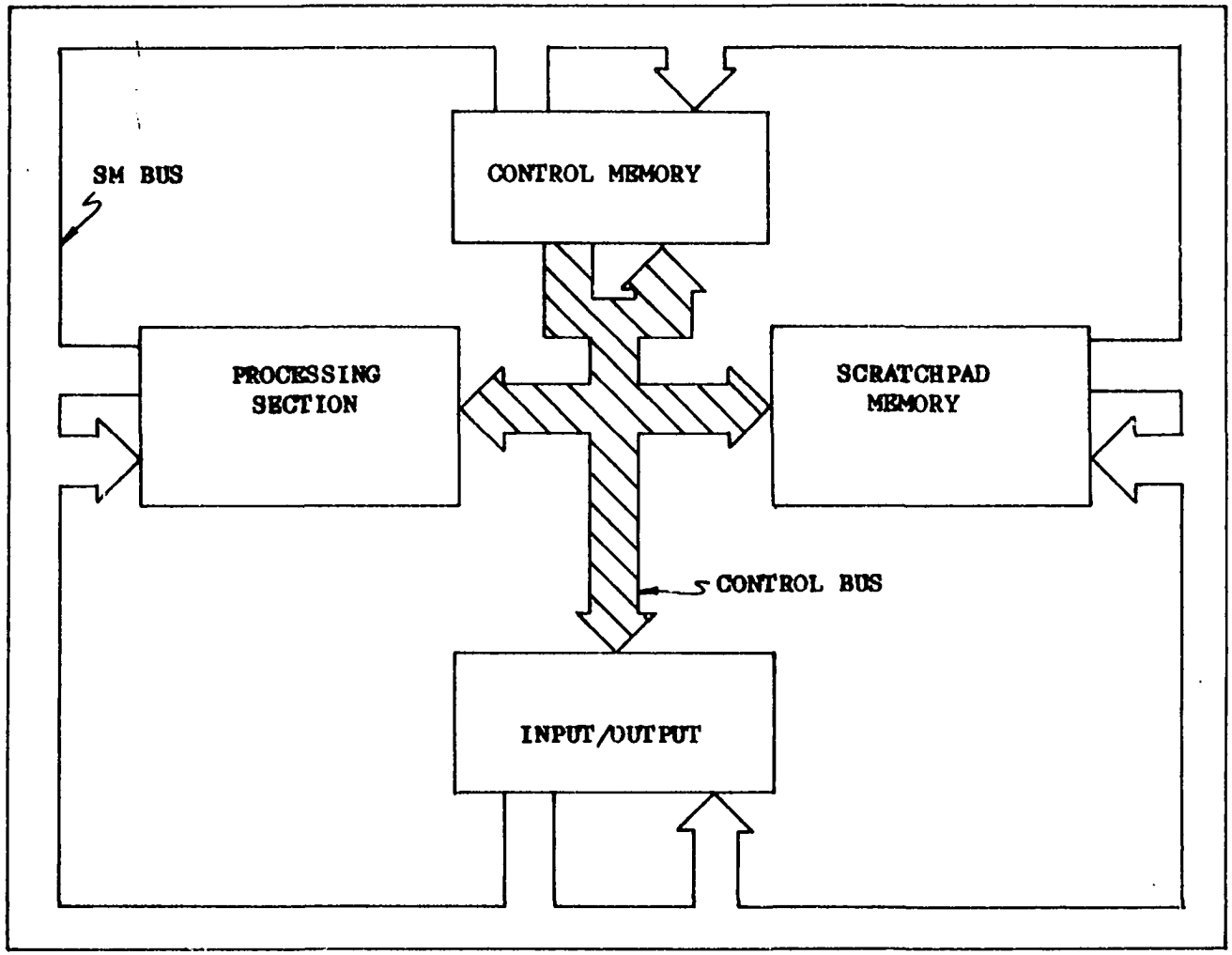
Control

The control memory is the center of the portion of the system that provides the control function. The various steps in a recognition algorithm will be written in machine language subroutines and stored in the control memory. A few housekeeping subroutines, such as one to process input interrupts, will also be stored in the control memory. Commands will be executed from the control memory in sequence unless a branch command or an input/output interrupt causes the sequence to be broken.

As indicated in Figure 1, control information flows from the control memory to other portions of the system via the control bus. When a command is read out of the control memory it is placed, still in encoded form, on the control bus. When the contents of the control bus is valid (i.e. when the memory cycle has been completed) a signal indicating this comes up, the command is decoded somewhere in the system and executed. When the execution is complete it is indicated by a completion signal. This completion signal in essence initiates the next control memory cycle.

The command set includes a substantial number of essentially control commands. A rather wide variety of conditional branch commands has been implemented to facilitate making tests on the results of processed data.

Figure 1. System block diagram


SM BUS
CONTROL MEMORY
PROCESSING SECTION
SCRATCHPAD MEMORY
CONTROL BUS
INPUT/OUTPUT

Alternatively, the results of tests can be stored in a register as the tests are made, and then the entire register contents tested against some specified bit pattern and one of two alternate courses of action taken. These commands are described in detail in Appendix A.

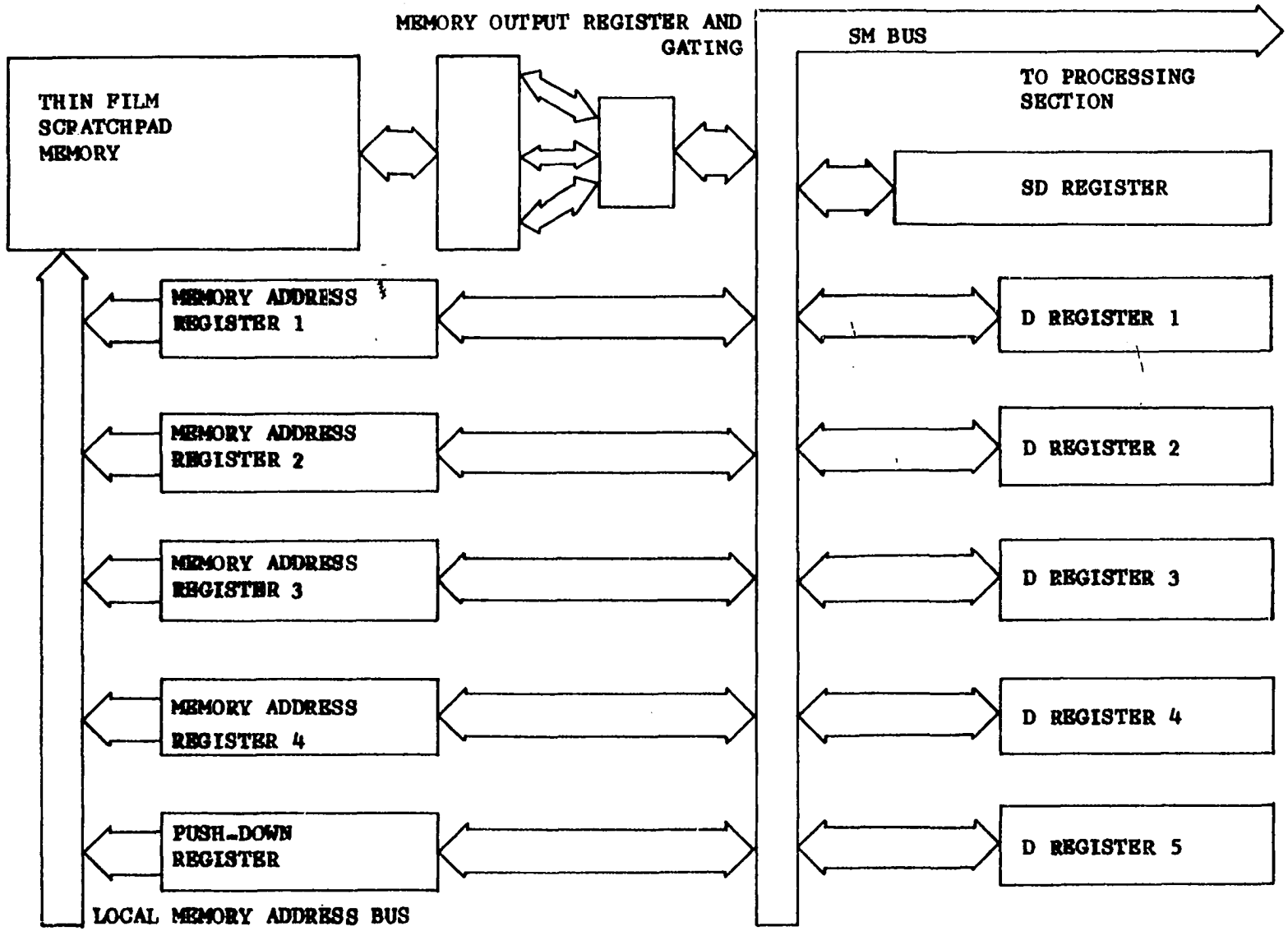
The timing of the system is slaved to the timing constraints imposed by the control memory. Obviously, the execution of a command cannot begin until it is available on the control bus. Command execution times are variable, with the only requirement being that no completion signal be generated until enough time has elapsed to allow another control memory cycle to be initiated. When an input/output interrupt occurs it is treated like a subroutine jump to a fixed control memory location, with the return address planted in a push-down list maintained in the scratchpad memory.

Data Flow and Storage

Figure 1 also shows an SM (scratchpad memory) bus linking the various parts of the system. This bus provides a mechanism for moving data and addressing information throughout the system. Raw input data will be moved from the document reader to the scratchpad memory via the SM bus. From there it will be moved to the processing section on the SM bus, processed and then returned via the SM bus.

Address information will also be moved via the SM bus. Figure 2 shows the memory address registers associated with the scratchpad memory in more detail. Often in the type of processing anticipated several streams of information will be flowing to or from the scratchpad memory. For example, perhaps at some point in time character array

Figure 2. Scratchpad memory section


MEMORY OUTPUT REGISTER AND GATING
SM BUS
TO PROCESSING SECTION
THIN FILM SCRATCHPAD MEMORY
SD REGISTER
MEMORY ADDRESS REGISTER 1
D REGISTER 1
MEMORY ADDRESS REGISTER 2
D REGISTER 2
MEMORY ADDRESS REGISTER 3
D REGISTER 3
MEMORY ADDRESS REGISTER 4
D REGISTER 4
PUSH-DOWN REGISTER
D REGISTER 5
LOCAL MEMORY ADDRESS BUS

information will be being processed eight bits at a time from an unprocessed array, and as it is processed be returned to a processed array in the scratchpad memory. At the same time, on an interrupt basis, a new character array will be coming into the scratchpad memory. Also, processing parameters that are stored in the scratchpad memory may be required for the appropriate processing of the character array. For this example four scratchpad memory locations will be of interest. It is assumed that the four memory address registers implemented will be used to hold these four addresses.

Scratchpad memory read and write commands issued from the control memory will normally specify one of the four memory address registers as the place to find the appropriate scratchpad memory address. The contents of the selected memory address register will be gated to the local memory address bus. The bit pattern on this bus is continuously decoded as the addressed word in the scratchpad memory. In the event the desired scratchpad memory address is not contained in one of the memory address registers several special scratchpad memory read and write commands have been implemented to retrieve the appropriate address from either the scratchpad memory, the control memory or a register in the processing section.

A fifth register called the push-down register can also be connected to the local memory address bus. This register, and the logic associated with it maintains a push-down list, eight addresses deep near the high address end of the scratchpad memory. This push-down list is used for subroutine linkage and linkage for input/output interrupts.

Figure 2 also shows five D registers associated with the scratchpad

memory. These registers are used primarily for indexing functions. Commands are implemented to decrement and test the contents of any of these registers.

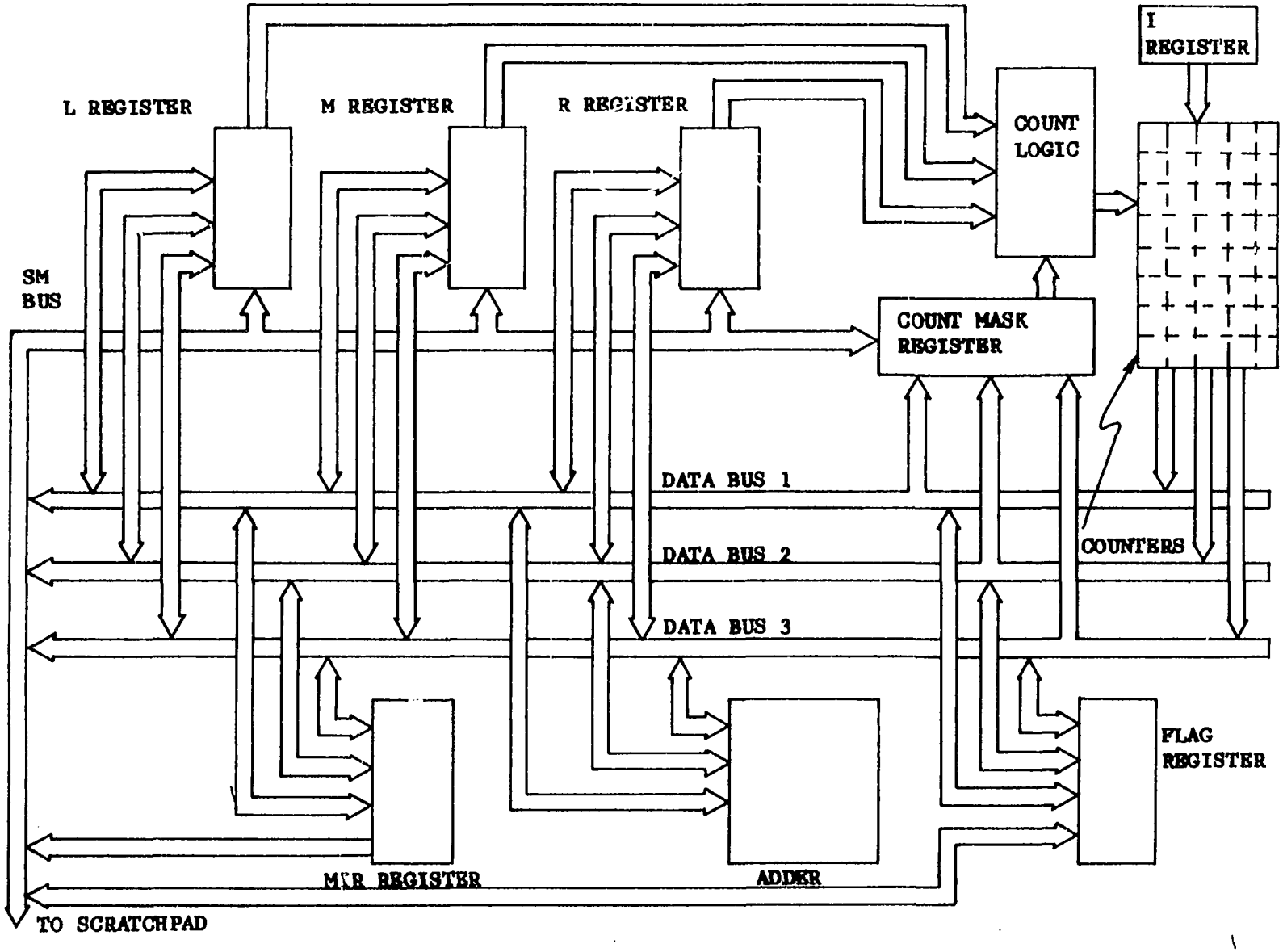
The last register shown in Figure 2, the SD register, is primarily intended to permit decrementing the memory address registers. A command is implemented to gate the contents of one of the MAR's to the SD register, decrement it by zero, one, two or eight and return the modified contents to a specified MAR. This register can also be used as a general purpose register since it can communicate with other registers via the SM bus.

The contents of all of the registers discussed as well as the scratchpad memory output register can be placed on the SM bus. Also, the contents of the SM bus can be loaded into any of these registers. The details of which lines of the SM bus are involved depends on the particular command being executed, and can be found in Appendix A.

Processing

Figure 3 shows a relatively detailed block diagram of the processing hardware. Data communication to and from the scratchpad memory is via the SM bus. Within the processing section data flow is primarily via three data buses. The possible paths for data flow are indicated on the block diagram of Figure 3. Generally, when the contents of a register is placed on a data bus the connection is maintained until a subsequent command breaks the connection. If the contents of two or more registers are placed on a particular data bus, the bus will contain the bit by bit logical OR of the registers involved. The contents of all three data

Figure 3. Processing section


I
REGISTER
L REGISTER
M REGISTER
R REGISTER
COUNT
LOGIC
SM
BUS
COUNT MASK
REGISTER
DATA BUS 1
DATA BUS 2
DATA BUS 3
COUNTERS
FLAG
REGISTER
MAR REGISTER
ADDER
TO SCRATCHPAD

buses can be gated to registers in the processing section with a single command. Also the contents of one data bus may be gated to several registers.

Processing a character array will typically be done eight bits at a time, and frequently the processing of a bit involves the bit and its eight nearest neighbors. The L, M and R registers are each ten bits in length, and are intended to provide local storage for eight bits and nearest neighbor information. Thus if the center eight bits of the M register are the eight contiguous bits being processed, all of the nearest neighbors of these bits can be held in these three registers.

The L, M and R registers communicate with a series of counters via some rather simple count logic. Viewed horizontally, the counters appear as eight independent, five-bit counters. Each of these eight counters is associated with the corresponding bit position of the center eight bits of the L, M and R registers. Whether a counter is incremented by zero, one, two, or three depends on the count command being executed, the contents of the corresponding bit positions of the L, M and R registers, and may depend on the contents of the count mask register. To increase the versatility of the logic any or all of the three registers; L, M and R, may be shifted one position up or down before the counting is done and returned to their original states after the counting is done.

Part of the power of the manner in which bit processing is done here comes from the variety of ways in which the contents of the counters can be placed on a data bus, and subsequently gated to other registers. The contents of any of the eight counters can be placed on any of the data

buses. Which of the eight is determined by the contents of the I register. The I register can be incremented and tested by command. Viewed vertically, the counters can be thought of as five, eight-bit registers. The contents of any of these five registers may be placed on any of the data buses.

The adder is an eight-bit ripple carry binary adder. Each of the two input operands can be the contents or the one's complement of the contents of any of the three data buses. Or, for special purposes an input operand can be made all zeros. The binary sum of the two inputs is formed continuously and can be placed on any of the data buses. The carry into the least significant adder stage can be set to be either a one or a zero. Any additions or subtractions involving operands more than eight bits long as well as multiplication and division must be programmed.

The MIR register serves primarily as a link back to the scratch-pad memory or to other registers served by the SM bus. However, it can be loaded from or connected to any of the data buses, and thus can serve as a general purpose register for the processing section as well.

The bits of the flag register can be set as a consequence of certain of the test commands as mentioned in an earlier section. Subsequently, a conditional branch can be made on the basis of the contents of the flag register. This register is shown in this section since communication to and from it from the data buses is also implemented.

Input/Output

The primary input device for this system will be a document reader that digitizes the character array. When a slice of the character being read is registered in an input buffer an interrupt flag will be set. This will cause an interruption in the normal sequential execution of commands from the control memory. The interrupt is processed by performing a subroutine jump to location zero of the control memory. A subroutine to process input/output requests will begin at location zero.

Output will also be handled on an interrupt basis. When output is called for an interrupt flag will be set. When the output device called for is available the output interrupt will be processed in a fashion similar to the way in which an input interrupt is handled.

Hopefully, the processed output from this system will eventually be entered directly into a general purpose computer. This would permit convenient comparisons to determine the accuracy of the particular recognition algorithm being tested. However, during the initial operational phase of the system either a typewriter or a paper tape punch will be used as an output device.

PROCESSING TECHNIQUES

A rationale for this system has been presented wherein certain objectives were stated and certain factors affecting system decisions were discussed. Subsequently, the system structure was described. Also, Appendix A contains a detailed description of all of the commands that have been implemented. Further insight into the reasons for the system being as it is can be gained by a discussion of certain aspects of the system as they relate to programs meant to do specific jobs.

Addressing Techniques

The use of a control memory that is not alterable under program control led to the association of several memory address registers with the scratchpad memory. These memory address registers are used in a pseudo indirect addressing scheme. That is, a typical control command that references the scratchpad memory does not specify an address. Rather, it specifies the memory address register in which the address can be found. These memory address registers can be incremented by zero, one, two or eight as part of the execution of the referencing command. This scheme is very convenient for moving through a character array. The possible increments were chosen specifically to facilitate working through a character array in two directions, as well as for storing and retrieving one and two byte parameters.

The first two subroutines in Appendix B illustrate typical movement through character arrays stored as shown in Figure 4. Horizontal movement across a character implies incrementing by eight (one word), and vertical motion down a character implies incrementing by one (one byte).

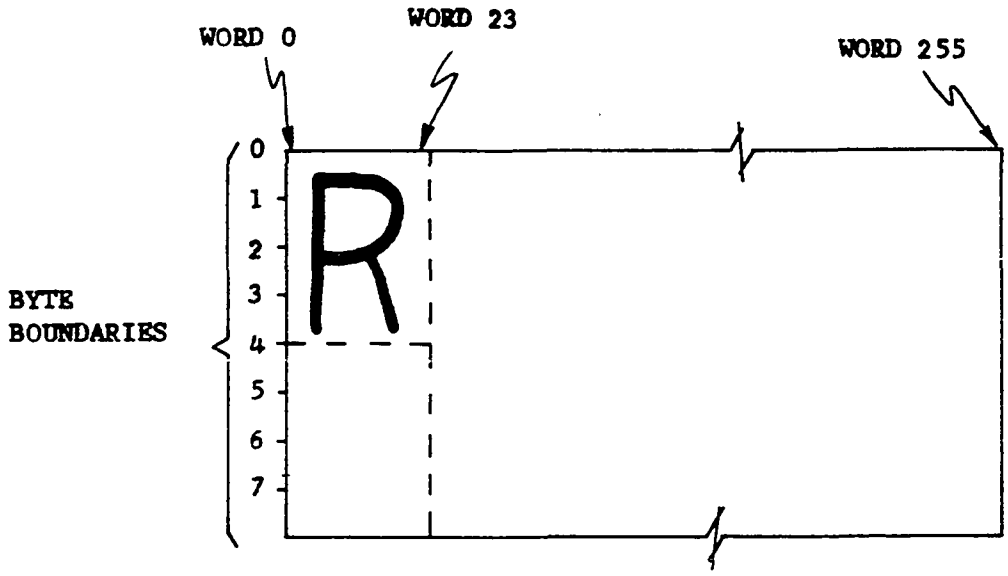


Figure 4. Typical character array storage in scratchpad memory

The SD register shown in Figure 2 was added to permit decrementing the memory address registers, not with the referencing command, but with a single command called move and decrement. Certain operations are awkward without the facility for moving in the opposite direction through an array, a list of parameters or a multiple byte operand. For example, in the addition subroutine in Appendix B each of the operands is assumed to be stored with the least significant byte in the memory location with the smallest address. The use of the SD register and the associated move and decrement command permits storing the operands in the opposite sense.

When a program calls for a single parameter or a few scattered parameters, this pseudo indirect addressing scheme is less efficient than for array processing. Also, when entering a new subroutine the contents of one or more of the memory address registers that is used by the subroutine may still be valid. To ease these situations 32 two byte groups can be addressed directly with read and write dedicated location control commands. Their mnemonics are RDL, RDLB and WDL. Several of programs in Appendix B make use of these dedicated locations in the scratchpad memory.

One additional type of command was implemented as well. When certain parameters or scratchpad memory addresses are fixed, they might as well be stored in the control memory. The line thinning program in Appendix B requires the array size to be specified. These numbers are likely to be fixed for a particular recognition algorithm. Consequently, they were carried as parameters in the control memory and moved to

registers early in the subroutine.

Note that the pseudo indirect addressing scheme, with some planning on the part of the programmer, can provide essentially completely relocatable array and parameter storage in the scratchpad. This says that the same control memory subroutines can be used regardless of where in the scratchpad memory the data is stored.

Bus Control Techniques

The major buses in this system fall into three classes as far as bus control is concerned. The control bus is the most straightforward of the three classes. There is only one data source, the output register of the control memory. Consequently, there need be no gating onto the bus. The lines can be tied to the register flip flops through amplifying buffers. Similarly, information can be taken off the control bus and put directly into local decoding networks without any control other than a signal defining the validity of the bus information.

The SM bus is shared by a number of data sources and data destinations. Since there is no convenient way of predicting future bus use at any point in time, all gating onto the bus is of a temporary nature. The data is held on the bus only long enough to be registered in the destination register. Since all destinations are registers this is quite acceptable. Also, the only acceptable way of gating from the bus to a register is to do it with pulsed transfer gates.

The three data buses also are shared by a number of data sources and destinations. However, here bus use can be determined for at least the subroutine being executed. Consequently, the control of data flow

via the data buses is handled quite differently than for the SM bus. Each register that can be connected to the data buses has two control flip flops associated with it that determine to which of the data buses it is to be connected. The alternative of no connection is also possible. One command, the RBS command, is a setup command and fixes all connections to the data buses until a subsequent setup command is executed. This is a double length command, the only one implemented, but thought to be a worthwhile exception. Note that the contents of several registers can be placed on one data bus. The result is the bit by bit logical inclusive OR of the contents of the registers connected to the data bus.

Many subroutines require only one setup command. The decision to implement three data buses was made to make this possible. The first three sample subroutines in Appendix B do require only one setup command. The multiplication subroutine requires rearranging the connections to the data buses when an addition cycle is required. Typically, the connections to the data buses would be set early in a subroutine. The particular connections would depend on the purpose of the subroutine and the whim of the programmer.

One command controls the gating from the data buses to the registers implemented as destination registers. The gating is a pulsed transfer with a particular register receiving data from only one data bus. However, the contents of one data bus can be transferred to several registers, and the contents of all three data buses can be transferred in parallel.

The interaction between the data buses and the adder inputs is slightly different. Each of the two, byte wide adder inputs has three control flip flops associated with it. Two of the flip flops determine which of the three data buses is to be connected to the input. The third determines whether the bus contents or the one's complement of the bus contents is to be connected. These six flip flops are set with one adder setup command, the BAS command. Since the connections, when established, remain until changed by a subsequent BAS command, the adder continuously forms the sum of whatever is connected as inputs. The output of the adder is treated as a potential source of data for the three data buses. One result of this adder gating scheme is that the command set does not need an add command.

Logical Manipulation of Data

The bit processing logic of this system represents the heart of the system. This is the area of the system where care was taken to provide substantial logical capability and versatility at a moderate cost. Further, the ability to perform logical manipulations in a reasonable time with a near minimum number of commands was also important since these operations tend to be inner loop operations in bit processing subroutines.

The bit processing logic consists essentially of the L, M and R registers, the logic and the commands connecting these processing registers to the counters and the eight counters. It is convenient to view this bit processing logic in two ways. First, it represents a convenient facility for counting bits, either ones or zeros, in a

character array. This same facility also represents substantial combinational logical capability for bit processing.

Counting operations

A number of the jobs in the feature extraction phase of character recognition algorithms can be reduced to the job of counting the number of ones or perhaps the number of zeros in some area of a character array. Summing the number of ones in the entire array, for example, might help to distinguish punctuation marks from alphabetic characters. Or, summing the number of ones in the upper half of an array should help to distinguish certain of the lower case letters from capital letters. An E might be distinguished from an F by counting the number of ones in the lower central portion of the array. The subroutine to sum all ones in a specified rectangular area of an array shown in Appendix B is an illustration of the type of counting application expected.

In this subroutine the array is assumed to be stored as illustrated in Figure 4. Array data is moved to the L register a byte at a time and counted with an SCLM command. At the end of each horizontal pass over the horizontal range specified the contents of the eight counters are added to the previous sum. As many horizontal passes are made as specified by the vertical range parameter. A mask byte is provided for each horizontal pass to permit the vertical range to actually be any number of bits rather than restricting it to byte boundaries. The numeric limits on the maximum row count and the total count reflect the length of the counters and the length of the adder. For the arrays expected these limits seem reasonable. However, greater effective

limits can be had with the present system by using a more involved program. If the horizontal range can exceed the counter limit two or more counter summations would have to be included in each horizontal pass. If the accumulated sum can exceed one byte this condition would have to be detected and some action taken. Perhaps noting the overflow by setting a bit in the flag register with a FSOV command would be adequate. In a situation where the actual sum is important, multiple byte addition could be programmed.

The full power of the counting commands is not used in the example subroutine. If the horizontal range can be limited to some multiple of three, or if specific checks are programmed, the array bytes can be brought into the L, M and R registers three at a time. The ones in the three bytes can be counted with one SCSM command. If the multiple-of-three restriction on the horizontal range can be tolerated this variation would reduce the execution time substantially.

This example was stated in terms of counting ones. Counting zeros can be done with a trival change in the program. In fact, there are at least two ways to count zeros rather than ones by modifying the existing subroutine. One way would be use a read complement command (RC1) rather than a read command (R1) when bringing a byte from the scratchpad to the L register. The other way would be to modify the counting parameter associated with the SCLM command so that the counters are incremented with a zero present in the register rather than with a one present.

Logical operations

Any combinational Boolean function can be generated from the OR and INVERT functions. The bit by bit OR of two or three bytes can be achieved in this system by loading the bytes in the L, M and R registers, executing an SCL command and treating the contents of the least significant vertical slice (the eight bit register called C_E) of the counters as the result. Inversion of an eight bit byte can be done by loading the byte in L, M or R, executing an SCL command and again treating the contents of C_E as the result. Since these two logical operations can be performed, any combinational function of boolean variables can be generated. This generality is important, but says little about speed or convenience of generating various functions. The following paragraphs are intended to show how several functions can be generated in a reasonable fashion.

Suppose the bit by bit AND of two bytes is desired. This can be done by loading them in the L and M registers and executing a SCS command. The bit by bit AND is the byte in the vertical counter slice C_D. The ability to invert a byte for counting purposes makes generating the AND of complemented boolean variables very simple also. That is, it costs no more in time to form $A \cdot \overline{B}$ than it does to form $A \cdot B$.

Since the variables likely to be combined in this processor are nearest neighbors of a particular bit the ability to perform a temporary, single bit shift prior to counting was included in the counting commands. For example, suppose the byte being processed is in the M register with its neighboring bytes in the L and R registers. Consider the i^{th} bit, M_i. Suppose the AND of the neighbor above and to the left, L_{i-1}, with the

neighbor below and to the right, R_{i+1}, is desired. That is, the i^{th} bit of the processed byte, P_i, is to be given by

$$P_i = L_{i-1} \cdot R_{i+1} \text{ for } i = 0, 1, 2, \cdots 7.$$

This can be done with a single SCS command in which L is shifted down one place and R shifted up one place prior to the count. Note that the register contents are returned to their original positions after the counting is done.

One of the preprocessing, noise smoothing operations that may be required for some recognition algorithms is a line thinning operation. Such a job can be stated in terms of a thresholding operation. For example, consider the following processing criterion. A given bit in the unprocessed array has eight nearest neighbors. These nine bits are to be considered in determining the corresponding bit in the processed array. If seven or more of these nine bits are ones then make the processed bit a one. Otherwise, the processed bit is to be made a zero. This operation tends to make lines in the processed array two bits narrower than corresponding lines in the unprocessed array. It also removes stray ones and fills small holes.

This type of operation is awkward on general purpose computers. This system can do this job, and other thresholding jobs quite conveniently. The second subroutine given in Appendix B is specifically intended for this job. Basically, the way in which the thresholding is accomplished is to count a bit and its eight nearest neighbors. Once the three bytes are in the L, M and R registers this takes three SCS commands. Then the counters are incremented by one. At this point the

desired byte can be found in C_B.

Other thresholds can be obtained in a similar fashion. For some thresholds the ability to obtain the OR of several of the vertical counter registers on a data bus simplifies the thresholding job. One example is for a threshold of two when the count can range from zero to nine. After the neighbors are counted the OR of vertical registers C_B, C_C and C_D would represent the processed character.

Arithmetic

Much more attention has been paid to the logical manipulation of data than to arithmetic operations in this system. All arithmetic beyond single byte additions must be programmed. The third subroutine in Appendix B is an example of doing multiple byte addition. This particular example shows only the addition of positive numbers. However, if the assumption can be made that negative numbers are stored in two's complement form, addition of signed numbers follows. Further, since the provision has been made to gate the complement of the contents of the data buses to the adder inputs along with being able to set the carry into the least significant adder stage to either one or zero, the ability to present the adder with the two's complement of an operand is built in. This means that subtraction of signed numbers also follows directly from the simple addition example given.

The assumption was made that multiplication would be done by a series of additions and shifts using the L, M, R and MIR registers to hold the operands. A powerful shift command and commands to test the least significant bits of the three data buses were implemented to

make multiplication easily programmable. The final example in Appendix B shows a way of obtaining the double length product of two single byte operands.

In this subroutine the L and M registers are considered as a unit and contain the partial product. The MIR register holds the multiplicand, and the multiplier is in the R register. The multiplier is examined a bit at a time by placing the contents of the R register on a data bus and testing the least significant bit of the data bus. The outcome of the test determines whether to add the multiplicand to the parital product and then shift one place, or to simply shift one place.

The multiplication of multiple byte operands follows directly from the multiplication example given, but would be a rather time consuming operation. This could be done by performing a series of the single byte multiplications shown followed by a summing operation.

Division is also possible. This follows from the ability to subtract and the ability to shift operands and test the most significant bits of the three data buses. Division would tend to be slightly more time consuming than multiplication.

CONCLUSIONS

This investigation and the resulting system demonstrate the feasibility of building special purpose computing systems for experimenting with various handprinted character recognition algorithms. The recognition algorithms can be quite varied, and may even be adaptive in nature.

The capability for modifying the recognition algorithms comes about primarily because of two features of the system. First since an electrically alterable control memory is used for storing the recognition algorithms, modifying an algorithm or substituting a completely different algorithm is quite possible. The power and versatility of the logic of the bit processing hardware provides the capability for varied bit processing tasks. This unique combination of counters, logic and control commands is crucial to the type of processing expected.

The combination of the control memory and the pseudo indirect addressing scheme used makes the implementation of adaptive recognition algorithms convenient. For example, the subroutine, presented in Appendix B, that is used to count the number of ones in some area of a character array is basically an adaptable subroutine. All that is required to change the range of a probe is to change the range parameters or perhaps the mask parameters carried into the subroutine. The ability to change probe range might allow the accommodation of significant differences in handprinted characters.

A comparison of subroutine execution times for this system and for a general purpose system points up the desirability of having special

purpose hardware to do special purpose processing. Estimates of execution times on this system were made for the first two subroutines presented in Appendix B. These same jobs were programmed in IBM 360 machine language, and execution times calculated from instruction execution timing information for the IBM 360/65.

The subroutine to count ones is expected to take about 175 microseconds on this system for an entire 32 by 24 bit array and the line thinning operation is expected to take about 425 microseconds. Times to do these jobs on the IBM 360/65 are greater by a factor of about 25. This, even though logic speeds and effective memory speeds are quite comparable for the two systems.

A character recognition rate of 100 characters a second permits spending an average of ten milliseconds on each character. For the execution times expected the IBM 360/65 would require 425 x 25 or something over ten milliseconds to do the line thinning, preprocessing operation. Obviously, a 100 character a second character rate cannot be achieved on the IBM 360/65 if this sort of a line thinning operation is required for the recognition algorithm being used.

The line thinning operation involves the entire character array and considerable processing. It is thought to be one of the most time consumming of the expected operations. The average execution time per feature extracted is expected to be less than 200 microseconds. At this rate 25 features could be extracted in five milliseconds leaving another five milliseconds for the classification job. These tentative figures indicate that this system should be able to meet the design specification of 100 characters per second.

On the other hand this system requires about 16 microseconds to add two 32 bit numbers and about 45 microseconds to multiply two single byte numbers. These execution times are not competative with general purpose computer add and multiply times.

The average length of the four subroutines presented in Appendix B is approximately 25 control memory words. With 2048 words of control memory available, 80 such subroutines could be stored in this memory. This capacity is consistent with expected average execution times and the amount of processing anticipated, and appears to an adequate capacity.

This system also demonstrates that the special purpose processing hardware for a job such as the recognition of handprinted characters can be achieved at moderate cost with presently available devices. This system uses approximately 3000 RTL integrated circuit packages for all logic outside of memory drive and sense circuitry. This represents a cost of about $1200. The simplicity of the packaging technique used insures moderate fabrication cost, probably not more than three times component cost.

Obviously, memory cost, document reader cost and power supply cost are not included in the cost figure mentioned. These system elements must be included in any system capable of doing this job, and so do not bear on added expense for special purpose processing hardware.

Perhaps for some future system the 100 character per second rate discussed here is inadequate. One way of increasing the character rate would be to incorporate a greater degree of parallelism in the processing section. For this system very little would have to be done in any

section other than in the processing section. Neither memory would be affected, nor would most of the control except for drive requirements. In the processing section register length, the amount of logic, the number of counters, the length of the adder and data bus width would all scale up linearly with increased parallelism. These increases come at relatively low cost, since they consist chiefly of logic, power supply capacity and fabrication. It looks quite feasible to build an economic system with greater parallelism incorporated.

The packaging technique used favors a bus oriented system. Proper organization of a bus oriented system makes an expandable system possible. For example, in this system the SD register and the special command associated with it discussed in a previous section were added by adding a partially filled board. This addition made minor changes necessary on only two other boards. This is typical of the capability for adding new hardware and associated control commands for this type of system organization.

No similar claim is made for this system for the increased parallelism type of expansion. However, if this type of expansion capability rather than the capability of adding new commands in a design goal in the original design it can be achieved. This sort of thing must be planned for. It strongly influences such decisions as system partitioning.

The special purpose processing capability suggested for character recognition and special processing capability for other jobs as well might take one of two forms. It might be a complete system such as the one described here. The special purpose processing capability might

also be added as an extension to a general purpose computer, providing the design techniques used in designing the general purpose computer permit system expansions. This system demonstrates that such design techniques do exist.

A point with general digital system application is demonstrated by this investigation and the resulting system. System design rules and minimization criteria should change with changing cost ratios and component reliabilities. A gate count is fast becoming a useless exercise.

It would be quite delightful to have a nice, compact set of total system minimization rules. However, with or without such a set of rules it seems abundantly clear that low cost logic in forms ranging from now standard integrated circuits to large scale integration (LSI) will be increasingly applied to ease jobs now being done, and will also be applied in new areas beyond present machine capability.

LITERATURE CITED

1. Rice, Rex, Keith Uncapher, Tom Steel, and L. C. Hobbs. Promising avenues for computer research. AFIPS [American Federation of Information Processing Societies] Fall Joint Computer Conference Proceedings 27, Part 2: 85-100. 1965.

2. Rice, Rex. Impact of arrays on digital systems. Institute of Electrical and Electronic Engineers Journal of Solid-State Circuits 2, No. 4: 148-155. 1967.

3. Bashkow, Theodore R., Azra Sasson, and Arnold Kronfeld. System design of a FORTRAN machine. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 16, No. 4: 485-499. 1967.

4. Mullery, A. P., R. F. Schauer, and R. Rice. Adam: a problem-oriented symbol processor. AFIPS Spring Joint Computer Conference Proceedings 23: 367-380. 1963.

5. Mendelson, Myron J. and A. W. England. The SDS sigma-7: a real-time time-sharing computer. AFIPS Fall Joint Computer Conference Proceedings 29: 51-64. 1966.

6. McCullough, James D., Kermith H. Speierman, and Frank W. Zurcher. A design for a multiple user multiprocessing system. AFIPS Fall Joint Computer Conference Proceedings 27, Part 1: 611-617. 1965.

7. Gibson, Charles T. Time-sharing on the IBM system/360: model 67. AFIPS Spring Joint Computer Conference Proceedings 28: 61-78. 1966.

8. Corbato, F. J., M. Merwin-Daggett, and R. C. Daley. An experimental time-sharing system. AFIPS Spring Joint Computer Conference Proceedings 21: 335-355. 1962.

9. Bryan, G. E. JOSS: 20,000 hours at a console — statistical summary. AFIPS Fall Joint Computer Conference Proceedings 31: 769-777. 1967.

10. Shaw, J. C. JOSS: a designers view of an experimental on-line computing system. AFIPS Fall Joint Computer Conference Proceedings 26, Part 1: 455-464. 1964.

11. Stevens, M. E. Automatic character recognition — a state of the art report. National Bureau of Standards Tech. Note 112. 1961.

12. David, E. E., Jr. and O. G. Selfridge. Eyes and ears for computers. Institute of Radio Engineers Proceedings 50: 1093-1101. 1962.

13. Holt, A. W. Character recognition using curve tracing. U.S. Patent 3, 142, 818. July 28, 1964.

14. Liu, C. N. and G. L. Shelton, Jr. An experimental investigation of a mixed-font print recognition system. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 15: 916-925. 1966.

15. Bomba, J. S. Alpha-numeric character recognition using local operations. AFIPS Eastern Joint Computer Conference Proceedings 16: 218-224. 1959.

16. Doyle, Worthie. Recognition of sloppy, hand-printed characters. AFIPS Western Joint Computer Conference Proceedings 47: 1737-1752. 1959.

17. Unger, S. H. Pattern detection and recognition. Institute of Radio Engineers Proceedings 46: 1737-1750. 1959.

18. Alt, F. L. Digital pattern recognition by moments. Association for Computing Machinery Journal 9: 240-258. 1962.

19. Bledsoe, W. W. and I. Browning. Pattern recognition and reading machine. AFIPS Eastern Joint Computer Conference Proceedings 16: 225-232. 1959.

20. Fu, King-Sun, Y. T. Chien, and Gerald P. Cardillo. A dynamic programming approach to sequential pattern recognition. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 16: 790-803. 1967.

21. Duda, R. O. and H. Fossum. Pattern classification by iteratively determined linear and piecewise linear discriminant functions. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 15: 220-232. 1966.

22. Highleyman, W. H. Linear decision functions with application to pattern recognition. Institute of Radio Engineers Proceedings 50: 1501-1514. 1962.

23. Chow, C. K. An optimum character recognition system using decision functions. Institute of Radio Engineers Transactions on Electronic Computers 6: 247-254. 1957.

24. Amari, Shunichi. A theory of adaptive pattern classifiers. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 16: 299-307. 1967.

25. Bonner, R. E. Pattern recognition with three added requirements. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 15: 770-781. 1966.

26. Dimond. T. L. Devices for reading handwritten characters. AFIPS Eastern Joint Computer Conference Proceedings 12: 232-237. 1957.

27. Brown, Richard M. On-line computer recognition of handprinted characters. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 13: 750-752. 1964.

28. Teitelman, Waren. Real time recognition of hand-drawn characters. AFIPS Fall Joint Computer Conference Proceedings 26: 559-575. 1964.

29. Groner, Gabriel F. Real-time recognition of handprinted text. AFIPS Fall Joint Computer Conference Proceedings 29: 591-601. 1966.

30. Fischer, George L., Jr., Donald K. Pollock, Bernard Raddack, Mary Elizabeth Stevens, eds. Optical character recognition. New York, N.Y., Spartan Books. 1962.

31. Wilson, Robert A. Optical page reading devices. New York, N.Y., Reinhold Publishing Corporation. 1966.

32. British Computer Society. Document Handling and Character Recognition Committee, 1966. Character recognition, 1967. London, England, author. 1967.

ACKNOWLEDGMENTS

The author wishes to express his thanks to R. M. Stewart for his encouragement and remarkable patience. The author wishes to thank A. V. Pohm for serving as a sounding board for ideas, many of which wilted and died. Also, the author wishes to thank Renny, Dick and the others who are making this system a physical reality. Finally, the author wishes to thank Sheila for her help in typing the first draft of this document.

This work was partially supported by the Iowa State Affiliates Program in Solid State Electronics.

APPENDIX A. SYSTEM COMMAND SET

The commands implemented in this system are described in this appendix. The commands are 20 bits in length except for one, double length command. The general form is an eight bit operation code followed by twelve bits of parameter information.

The commands are grouped according to general function rather than in any numeric order. The command types in the order in which they appear and the members of each group are listed below.

1. Branching commands

   JSB, SR, JNZD1, JNZD2, JNZD3, JNZD4, JNZD5

   JNZ1, JNZ2, JNZ3, JOV1, JOV2, JOV3, JLS1, JLS2, JLS3

   TF0, TF1, ITI

2. Test and set commands

   FSNZ1, FSNZ2, FSNZ3, FSOV1, FSOV2, FSOV3, FSLS1, FSLS2, FSLS3

3. Scratchpad memory commands

   R1, RC1, R2, RC2, RDL, RDLB, W1, W2, W3, WDL

4. Data movement commands

   MOVE, MA1, MA2, MA3, MA4, MD

5. Data bus setup and transfer commands

   RBS, BRT, BAS

6. Clear and set commands

   CLEAR, SCI, SI, SCM

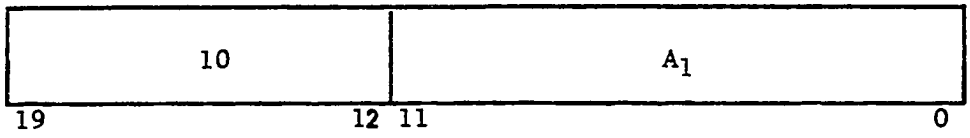
7. Shift and count commands

   SCLM, SCSM, SCL, SCS, SNC

Branching Commands

Subroutine jump

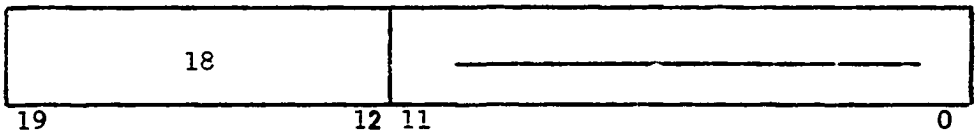
JSB A_1



First, the contents of the CMAR (next command location) is stored in a push-down stack formed in the scratchpad memory. Then an unconditional jump command is executed. The limit on nesting subroutines is eight.

Subroutine return

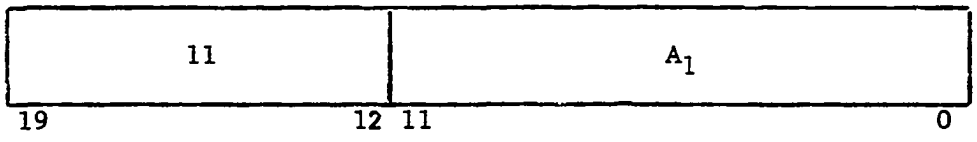
SR



Take the next command from the control memory location given by the top entry in the push-down stack.

Decrement and test D register 1

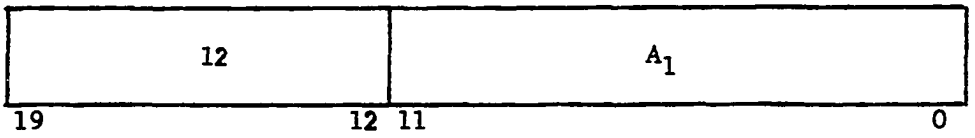
JNZD1 A_1



First, decrease the contents of D register 1 by one. Then, if the contents of the register is not equal to zero take the next command from control memory location A_1. Otherwise take the next command in sequence.

Decrement and test D register 2

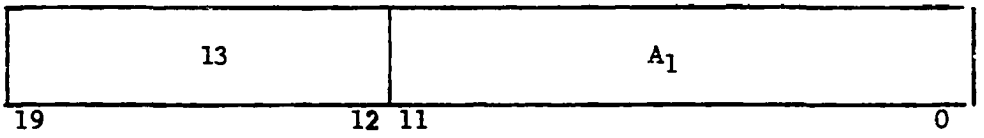
JNZD2 A_1



Same as JNZD1 except the D register 2 is involved.

Decrement and test D register 3

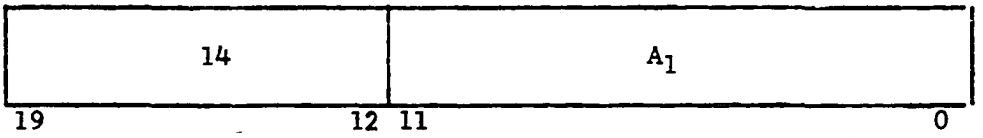
JNZD3 A_1



Same as JNZD1 except the D register 3 is involved.

Decrement and test D register 4

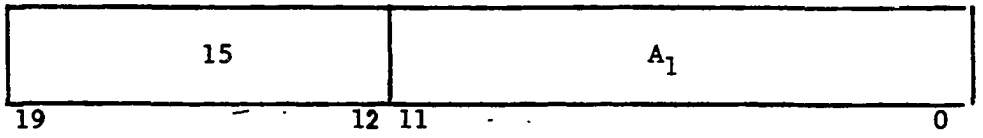
JNZD4 A_1



Same as JNZD1 except the D register 4 is involved.

Decrement and test D register 5

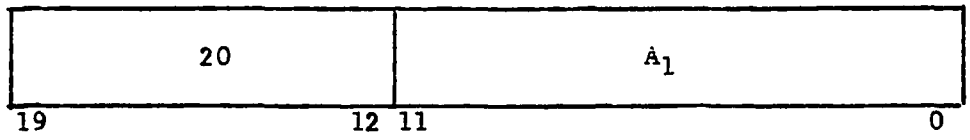
JNZD5 A_1



Same as JNZD1 except the D register 5 is involved.

Unconditional jump

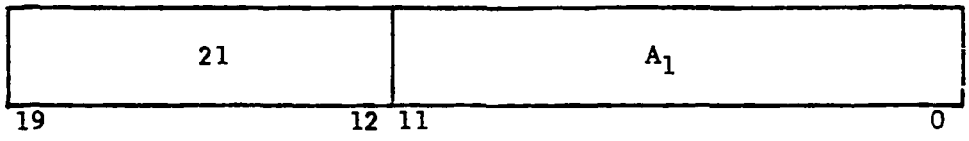
JUC A_1



The contents of the CMAR is replaced with A_1. Consequently, the next command is taken from control memory location A_1.

Non-zero jump on DB1

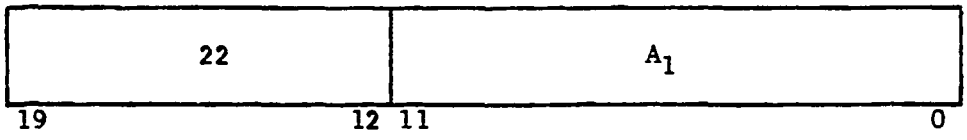
JNZ1 A_1



If the byte on DB1 (middle eight bits) is non-zero take the next command from control memory location A_1. Otherwise take the next command in sequence.

Non-zero jump on DB2

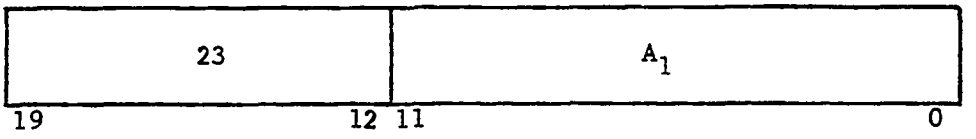
JNZ2 A_1



This command is the same as JNZ1 except the test is on DB2.

Non-zero jump on DB3

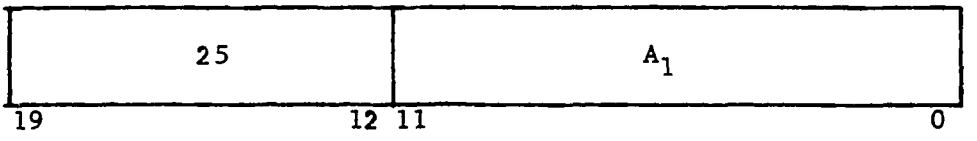
JNZ3 A_1



This command is the same as JNZ1 except the test is on DB3.

Overflow jump on DB1

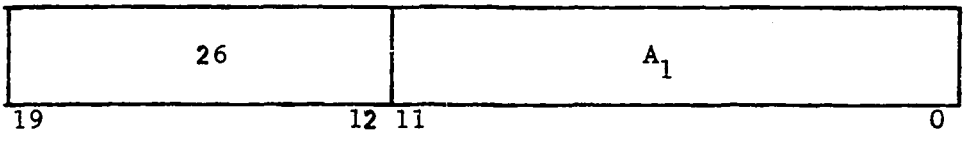
JOV1 A_1



If the most significant bit of DB1 (i.e. $DB1_{-1}$) is non-zero take the next command from control memory location A_1. Otherwise take the next command in sequence.

Overflow jump on DB2

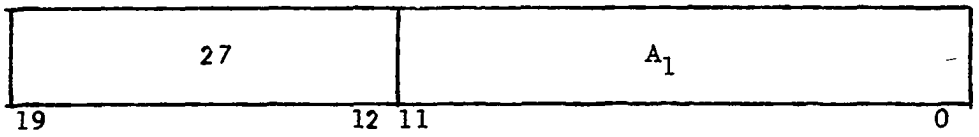
JOV2 A_1



This command is the same as JOV1 except the test is on DB2.

Overflow jump on DB3

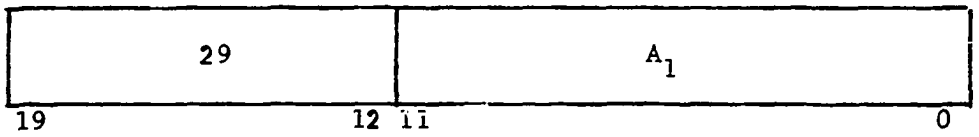
JOV3 A_1



This command is the same as JOV1 except the test is on DB3.

Least significant jump on DB1

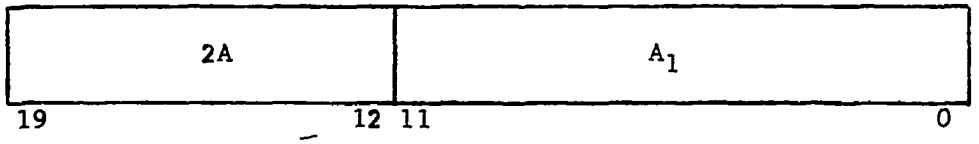
JLS1 A_1



If the least significant bit of the eight-bit byte on DB1 (i.e. DBL_7) is non-zero take the next command from control memory location A_1. Otherwise take the next command in sequence.

Least significant jump on DB2

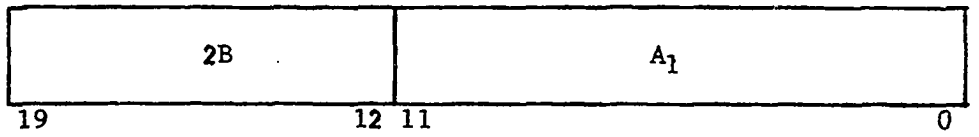
JLS2 A_1



This command is the same as JLS1 except the test is on DB2.

Least significant jump on DB3

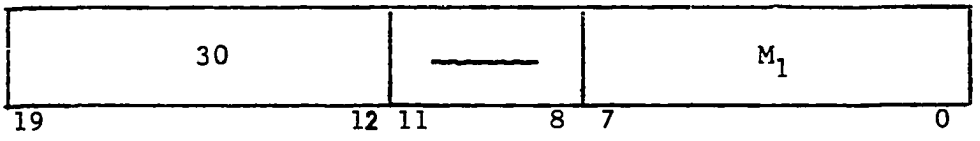
JLS3 A_1



This command is the same as JLS1 except the test is on DB3.

Test flag register for zeros

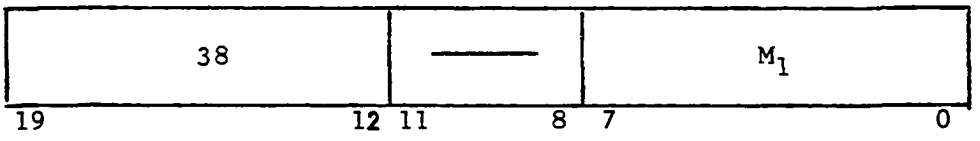
TFO M_1



If the flag register has zeros in at least the bit positions corresponding to the ones of M_1 take the next command in sequence. Otherwise skip one command (one control memory location).

Test flag register for ones

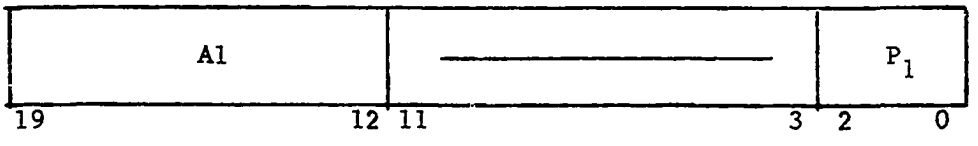
TF1 M_1



If the flag register has ones in at least the bit positions corresponding to the ones of M_1 take the next command in sequence. Otherwise skip one command (one control memory location).

Increment and test I

ITI P_1



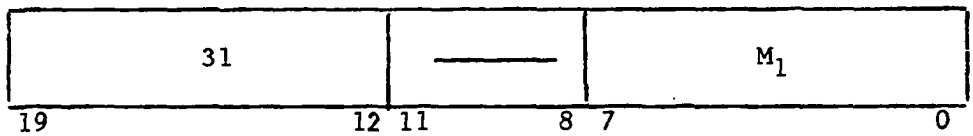
Increment the I register by one. Then if $I \neq P_1$ take the next command in sequence. Otherwise skip one control memory command (one

control memory location).

Test and Set Commands

Non-zero flag set on DB1

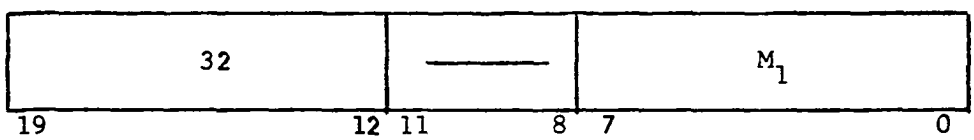
FSNZ1 M_1



If the byte on DB1 (middle eight bits) is non zero set ones in the flag register wherever there are ones in the corresponding bits of M_1.

Non-zero flag set on DB2

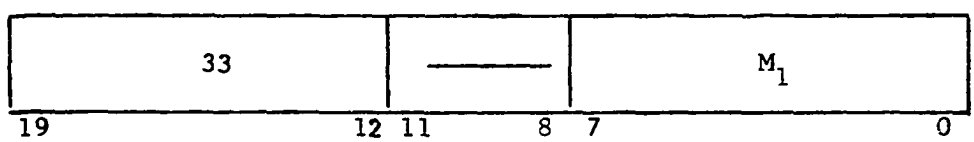
FSNZ2 M_1



This command is the same as FSNZ1 except the test is on DB2.

Non-zero flag set on DB3

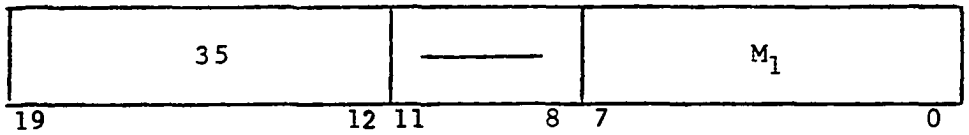
FSNZ3 M_1



This command is the same as FSNZ1 except the test is on DB3.

Overflow flag set on DB1

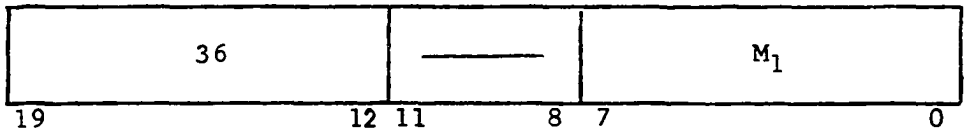
FSOV1 M_1



If the most significant bit of DB1 (i.e. $DB1_{-1}$) is non-zero set ones in the flag register wherever there are ones in the corresponding bits of the M_1.

Overflow flag set on DB2

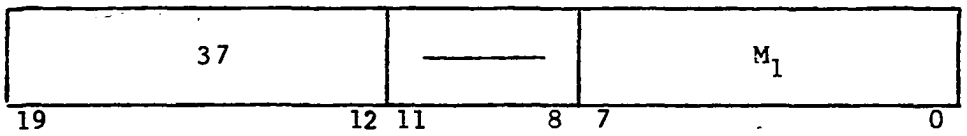
FSOV2 M_1



This command is the same as FSOV1 except the test is on the most significant bit of DB2.

Overflow flag set on DB3

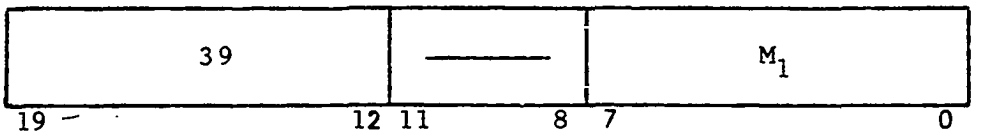
FSOV3 M_1



This command is the same as FSOV1 except the test is on the most significant bit of DB3.

Least significant flag set on DB1

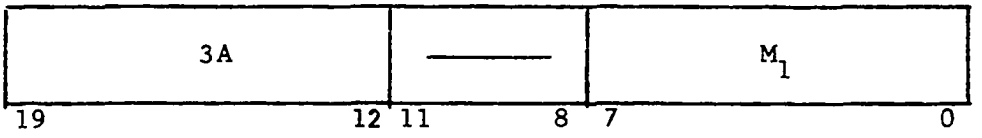
FSLS1 M_1



If the least significant bit of the eight-bit byte on DB1 (i.e. $DB1_7$) is non-zero set ones in the flag register wherever there are ones in the corresponding bits of the M_1.

Least significant flag set on DB2

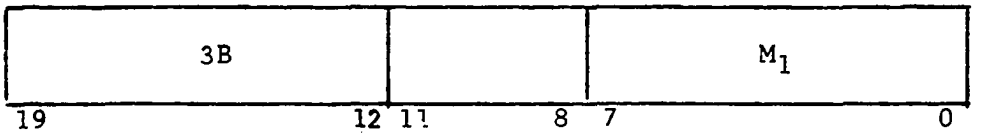
FSLS2 M_1



This command is the same as FSLS1 except the test is on DB2.

Least significant flag set on DB3

FSLS3 M_1

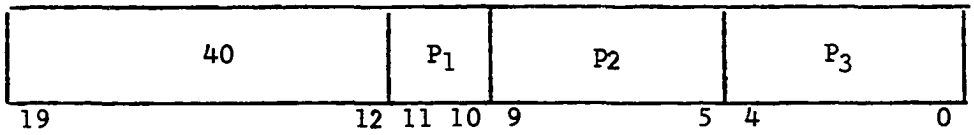


This command is the same as FSLS1 except the test is on DB3.

Scratchpad Memory Commands

Read scratchpad memory, type 1

R1 P_1, P_2, P_3



The padded byte (i.e. an eight-bit byte plus a bit on each end) specified by the contents of MAR (memory address register) P_3 is read from the scratchpad, placed on lines -1, 0, 1, ···, 8 of the SM bus, and transferred to the destination register specified by P_2. Table 1 specifies the coding for the various registers.

P_1 specifies the amount by which MAR P_3 is to be incremented after the memory cycle as follows:

00 — no increment

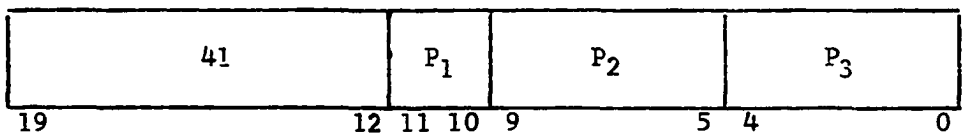
01 — increment by 1

10 — increment by 2

11 — increment by 8.

Read scratchpad memory complement, type 1

RC1 P_1, P_2, P_3



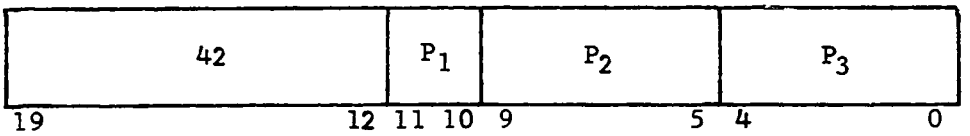
This command is identical to R1, except the bit by bit complement of the 10 bits in memory is loaded in the specified register.

Table 1. Destination specification for read scratchpad memory commands

Register	Destination specification
CMAR	00000
MAR1	00001
MAR2	00010
MAR3	00011
MAR4	00100
L	00101
M	00110
R	00111
D1	01001
D2	01010
D3	01011
D4	01100
D5	01101
CM	10001
FLAG	10010
SD	10011
OUTPUT DEVICE 1	11001
OUTPUT DEVICE 2	11010

Read scratchpad memory, type 2

R2 P_1, P_2, P_3



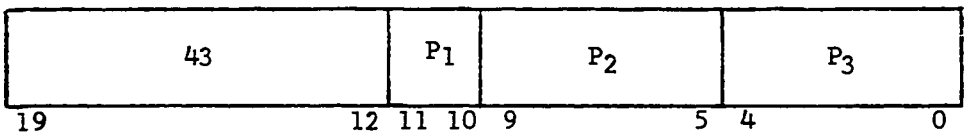
This command is identical to R1 except 16 bits from memory (starting at an even byte boundry) are put on lines 0-15 of the SM bus and transferred to the destination register.

When the destination register is a 12-bit register the contents of lines 4-15 are actually registered.

When the destination is an 8-bit register the contents of lines 0-7 are actually registered.

Read scratchpad memory complement, type 2

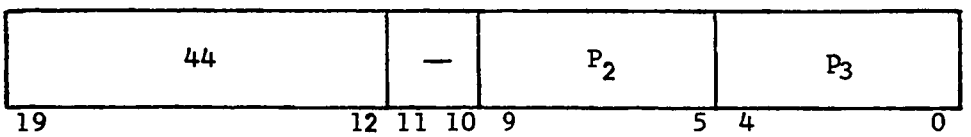
RC2 P_1, P_2, P_3



This command is identical to R2 above except the bit by bit complement of the 16 bits in memory are transferred.

Read scratchpad memory, dedicated location, type 1

RDL P_2, P_3

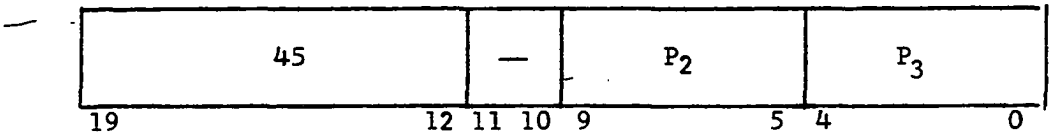


This command permits the reading of the last 8 words in variable

memory as 32 16-bit units. P_3 specifies which of the 32 units is to read. P_2 specifies the register in which the read information is to be registered.

Read scratchpad memory, dedication location, type 2

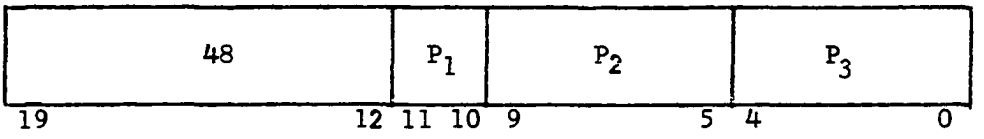
RDLB P_2, P_3



This command permits the reading of the least significant half only of the 32 dedicated 16-bit units of memory. P_3 specifies which of the 32 units to be read. P_2 specifies the register in which the read information is to be registered.

Write scratchpad memory, type 1

W1 P_1, P_2, P_3



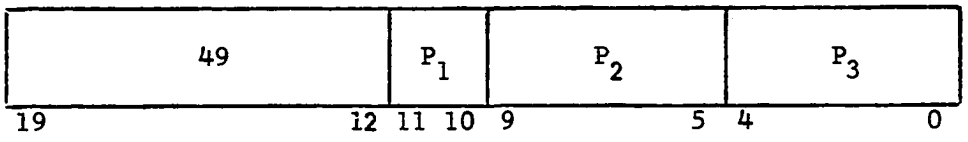
The eight-bit byte on lines 0, 1, ···, 7 of the SM bus is stored in the scratchpad location specified by the contents of MAR P_3. The source of the byte is the register specified by P_2. Table 2 specifies the coding for the various registers. MAR P_3 is then incremented by P_1 (see R1).

Table 2. Source specification for write scratchpad memory commands

Register	Source specification
CMAR	00000
MAR1	00001
MAR2	00010
MAR3	00011
MAR4	00100
DB1	00101
DB2	00110
DB3	00111
PDR	01000
D1	01001
D2	01010
D3	01011
D4	01100
D5	01101
CM	10001
FLAG	10010
SD	10011
MIR-DB3	10000
INPUT DEVICE 1	11011
INPUT DEVICE 2	11100

Write scratchpad memory, type 2

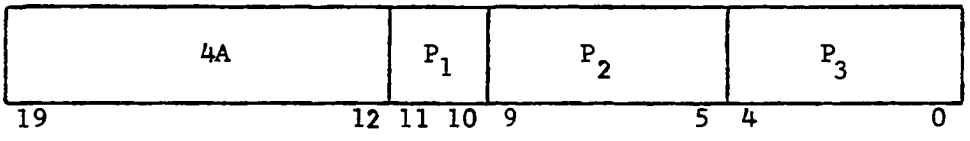
W2 P_1, P_2, P_3



This command is the same as W1 except 16 bits are stored. These 16 bits come from the specified register via lines 0-15 of the SM Bus. Storage begins on an even byte boundary.

Write scratchpad memory, type 3

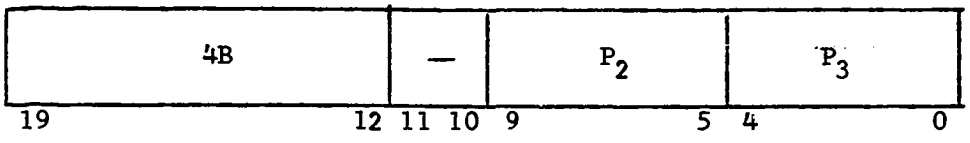
W3 P_1, P_2, P_3



This command is like W1 except 32 bits are stored. These 32 bits come via lines 0-31 of the SM Bus. Storage begins on a half-word boundary.

Write variable memory, dedicated location

WDL P_2, P_3

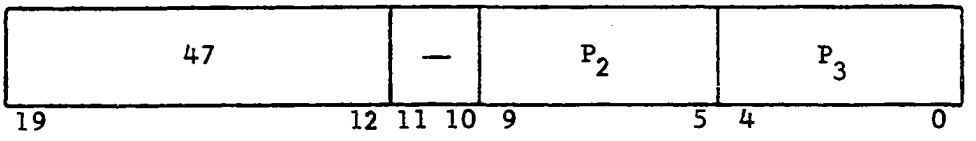


This command permits writing 16 bits in one of the last 8 memory words. P_3 specifies which of the 32 16-bit units to be written into. P_2 specifies the source of the information to be written.

Data Movement Commands

Move on SM bus

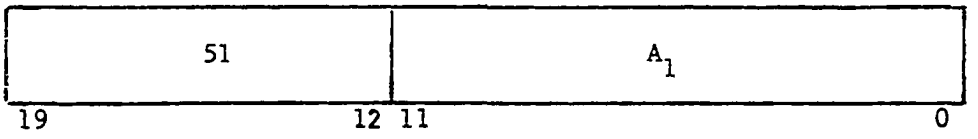
MOVE P_2, P_3



Move the contents of the register specified by P_3 to the register specified by P_2. When source and destination registers are different lengths, normal SM bus connections will be maintained. Zeros will be transferred to register bit positions not defined by the source register.

Move address 1

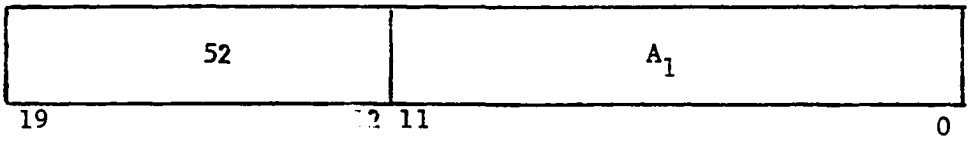
MA1 A_1



Move the address A_1 to MAR1.

Move address 2

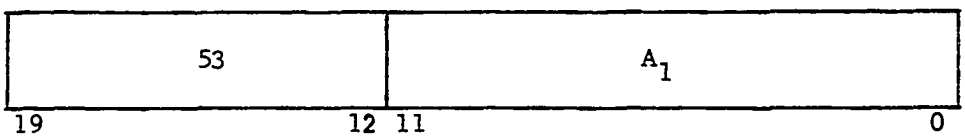
MA2 A_1



Move the address A_1 to MAR2.

Move address 3

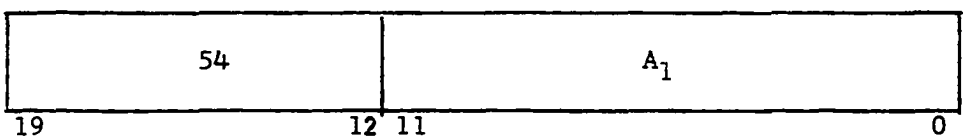
MA3 A_1



Move the address A_1 to MAR3.

Move address 4

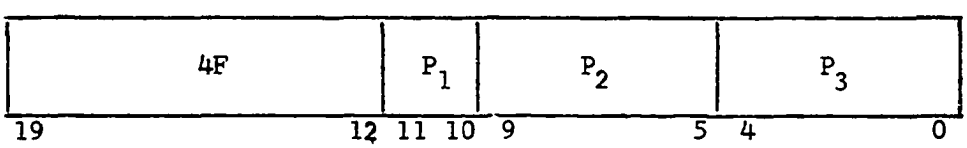
MA4 A_1



Move the address A_1 to MAR4.

Move and decrement

MD P_1, P_2, P_3



Transfer the contents of the MAR defined by P_3 to the SD register, decrement the contents of the SD register by an amount specified by P_1 and return the modified contents of the SD register to the MAR specified by P_2.

The possible decrements and their specifications are:

00 — no decrement

01 — decrement by 1

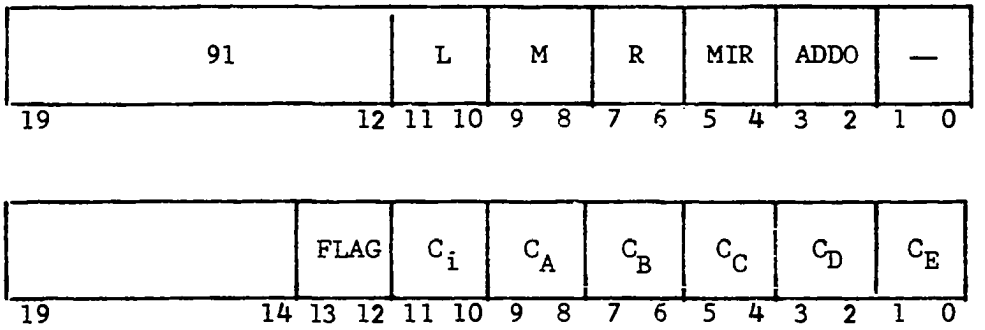
10 — decrement by 2

11 — decrement by 8

Data Bus Setup and Transfer Commands

Register to data bus setup

RBS list



Connect the register to the data buses as specified. The specification for a register is as follows:

00 no connection

01 register is connected to DB1

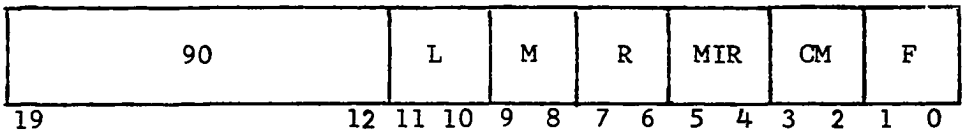
10 register is connected to DB2

11 register is connected to DB3.

Once a connection is established it will exist until a subsequent RBS command changes it. Note that this is a double length command.

Data bus to register transfer

BRT list



Transfer the contents of the data buses to the registers as specified. The specification for a register is as follows:

00 — no transfer

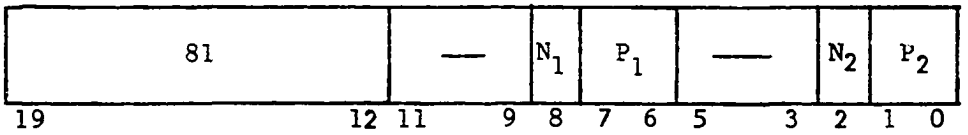
01 — transfer from DB1

10 — transfer from DB2

11 — transfer from DB3.

Data bus to adder setup

BAS N_1, P_1; N_2, P_2



Setup the inputs to the adder as specified. P_1 and P_2 specify the data bus to be connected to adder inputs one and two respectively. The specification is as follows:

00 — NUL

01 — DB1 is connected

10 — DB2 is connected

11 — DB3 is connected.

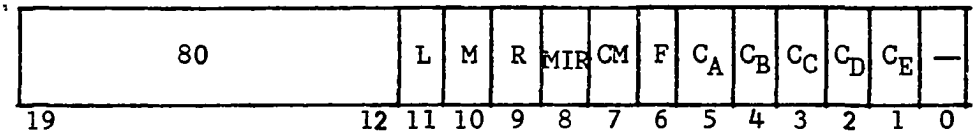
N_1 and N_2 specify whether the bus contents or the bit by bit complement of the bus contents is to be connected to the respective inputs

(0 — bus contents, 1 — complement of bus contents). These connections exist changed by a subsequent BAS command.

Clear and Set Commands

Clear registers

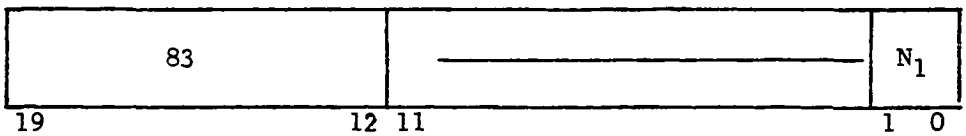
CLEAR list



Clear the register(s) specified by ones in the corresponding address bit position to zero.

Set carry in

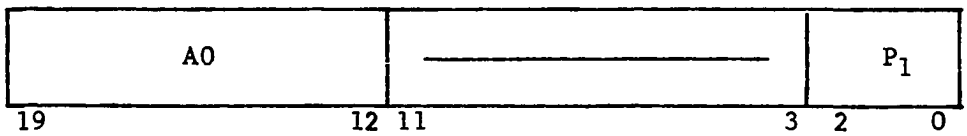
SCI N_1



Set the carry into the least significant adder stage to agree with N_1.

Set I

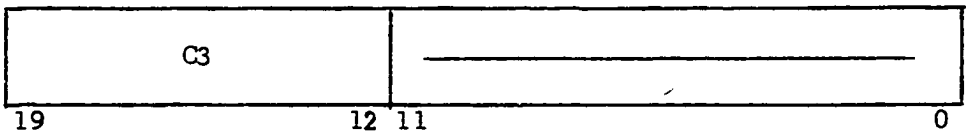
SI P_1



Set the I register to the value given by P_1.

Set count mask register

SCM

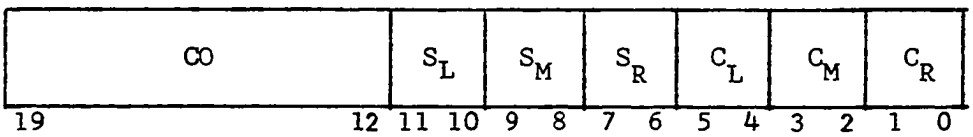


Set all bits of the CM register to one.

Shift and Count Commands

Shift and count logical (masked)

SCLM S_L, S_M, S_R; C_L, C_M, C_R



In this command three steps are done in sequence.

1. L, M and R are shifted one place according to the values of S_L, S_M and S_R respectively. The specification is:

   00 — no shift

   01 — shift down

   10 — shift up

   11 — not allowed

2. The counting called for by C_L, C_M and C_R is done. A count may occur only on the counter levels for which the CM (count mask) register contains a one. Suppose level j of CM is one. Then for C_L equal to:

00 no count regardless of L_j

01 count if $L_j = 0$

10 count if $L_j = 1$

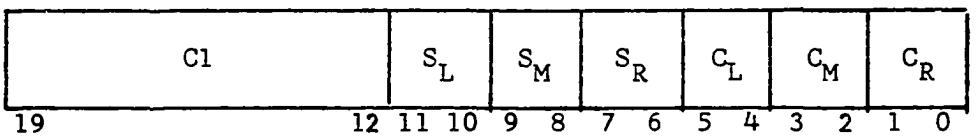
11 count regardless of L_j.

Similar specifications hold for C_M applying to the M register and C_R applying to the R register. The outcome of the tests on the three registers are OR'ed together so a particular counter may be incremented by one or not at all.

3. Finally, the registers are shifted back to their original positions.

Shift and count successively (masked)

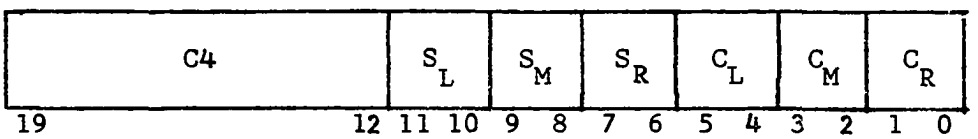
SCSM S_L, S_M, S_R; C_L, C_M, C_R



This command is identical to SCLM except for one difference. Here the count specification and register contents are examined successively for the three registers involved. A particular counter may be incremented by zero, one, two or three as a consequence of this command.

Shift and count logical

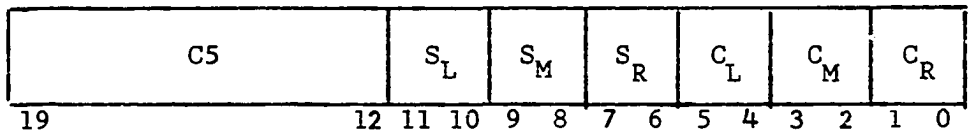
SCL S_L, S_M, S_R; C_L, C_M, C_R



This command is identical to SCLM except that the CM register is ignored.

Shift and count successively

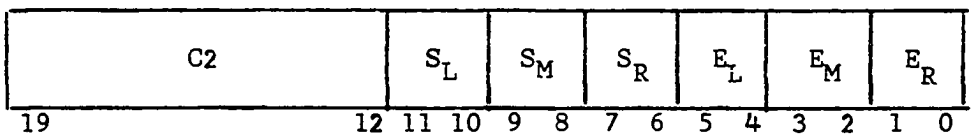
SCS S_L, S_M, S_R; C_L, C_M, C_R



This command is identical to SCSM except that the CM register is ignored.

Shift, no count

SNC S_L, S_M, S_R; E_L, E_M, E_R



Shift L, M and R one place according to the values of S_L, S_M and S_R respectively. (See SCLM for the shift specification rules.) The end connections are set by the E specifications. For a particular register the specifications are:

00 — shift a zero in

01 — shift a one in

10 — independent circular shift

11 — shift in contents of opposite end of adjacent register

(See Figure 5).

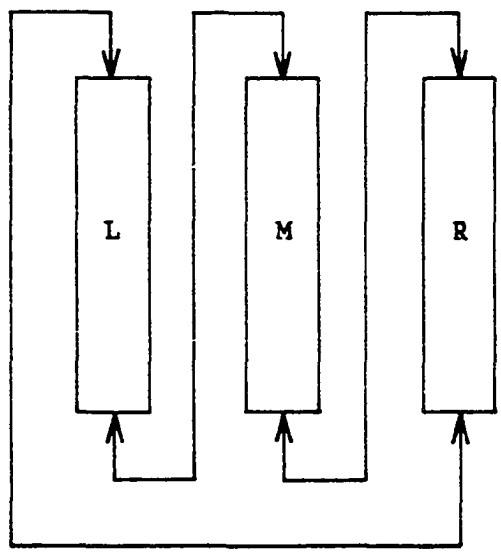


Figure 5. End connections for SNC command

APPENDIX B. SAMPLE PROGRAMS

Several symbolic programs are presented to show ways in which the command set and the hardware can be used for several typical processing jobs expected.

Sum all Ones in a Specified Rectangular Area

The flowchart for this subroutine is given in Figure 6, and the symbolic program is presented as Table 3. The purpose of this subroutine is to sum the total number of ones in a specified rectangular area of the array. The area is specified by a list of parameters stored in the scratchpad memory.

The initial assumption is that MAR3 contains the location of the first parameter in the list. The list consists of the following parameters stored in the order listed:

1. initial byte location (IBLOC)
2. horizontal range in bits (HR)
3. vertical range in bytes (VR)
4. as many mask-bytes as there are bytes of vertical range.

A dedicated scratchpad location is used for temporary address storage (DLOCT). The area over which the summing is done is variable, but it is assumed that no horizontal row has over 31 ones and that the total number of ones is less than 256. The result is left in MIR.

Figure 6. Flowchart for summing all ones in a specified rectangular area

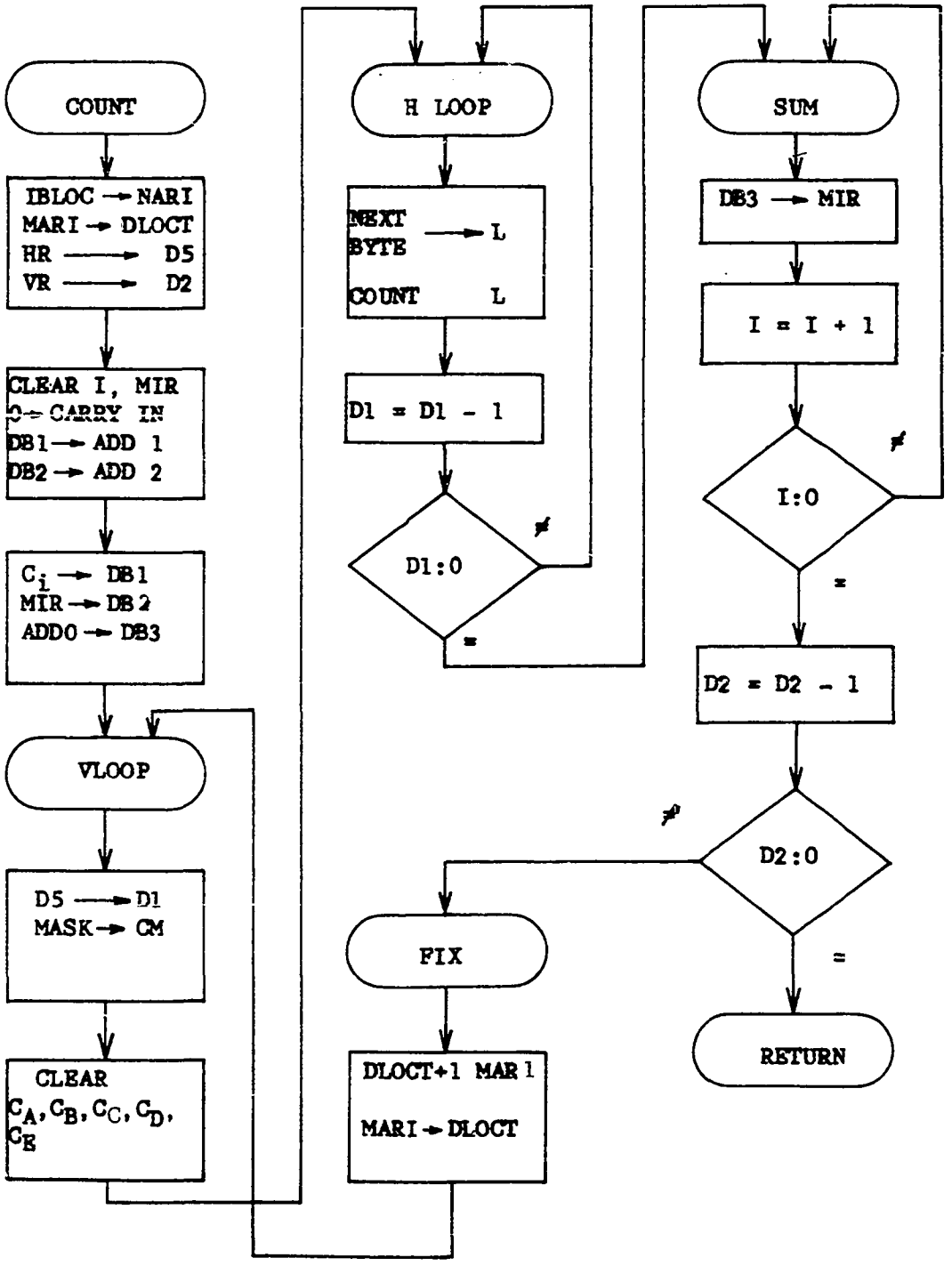

COUNT
IBLOC → NARI
MARI → DLOCT
HR → D5
VR → D2
CLEAR I, MIR
0 → CARRY IN
DB1 → ADD 1
DB2 → ADD 2
C_i → DB1
MIR → DB2
ADD0 → DB3
VLOOP
D5 → D1
MASK → CM
CLEAR
C_A, C_B, C_C, C_D, C_E
H LOOP
NEXT BYTE → L
COUNT L
D1 = D1 - 1
D1:0
≠
=
SUM
DB3 → MIR
I = I + 1
I:0
≠
=
D2 = D2 - 1
D2:0
≠
=
FIX
DLOCT+1 MAR1
MARI → DLOCT
RETURN

Table 3. Symbolic program for summing all ones in a specified rectangular area

Label	Symbolic operation	Symbolic parameters	Comments
COUNT	R2	2, MAR1, MAR3	obtain initial byte location
	WDL	MAR1, DLOCT	store it in a temporary location
	R1	1, D5, MAR3	put horizontal range in D5
	R1	1, D2, MAR3	put vertical range in D2
	SI	0	clear I
	CLEAR	MIR	
	SCI	0	set carry in to 0
	BAS	0, DB2; 0, DB1	setup adder inputs
	RBS	C_i; MIR; ADDO	setup data buses
VLOOP	MOVE	D5, D1	copy horizontal range in D1
	R1	1, CM, MAR3	mask to CM
	CLEAR	C_A, C_B, C_C, C_D, C_E	clear counters
HLOOP	R1	8, L, MAR1	byte to L
	SCLM	; 10	count ones
	JNZD1	HLOOP	
SUM	BRT	;; MIR	adder output to MIR
	ITI	0	
	JUC	SUM	
	JNZD2	FIX	check for more processing
	SR		return to main program

Table 3 (Continued)

Label	Symbolic operation	Symbolic parameters	Comments
FIX	RDL	MAR1, DLOCT	three commands to prepare for next pass
	R1	1, NUL, MAR1	
	WDL	MAR1, DLOCT	
	JUC	VLOOP	

Line Thinning and Stray Bit Deletion

The flowchart for this subroutine is given in Figure 7, and the symbolic program is given in Table 4. The purpose of the subroutine is to reduce the thickness of lines in a character array. It does this by performing a thresholding operation. Consider a given bit in the unprocessed array and its eight nearest neighbors. These nine bits are to be considered in determining the corresponding bit in the processed array. If seven or more of these nine bits are ones then make the processed bit a one. Otherwise the processed bit is to be made a zero.

The initial assumptions are that the array dimensions are parameters in the control memory, and that the beginning byte locations of the unprocessed array and the processed array are stored in two dedicated locations in the scratchpad (DLOC1, DLOC2). At the end the processed array and the unprocessed array are both stored in the scratchpad.

Figure 7. Flowchart for a line thinning operation

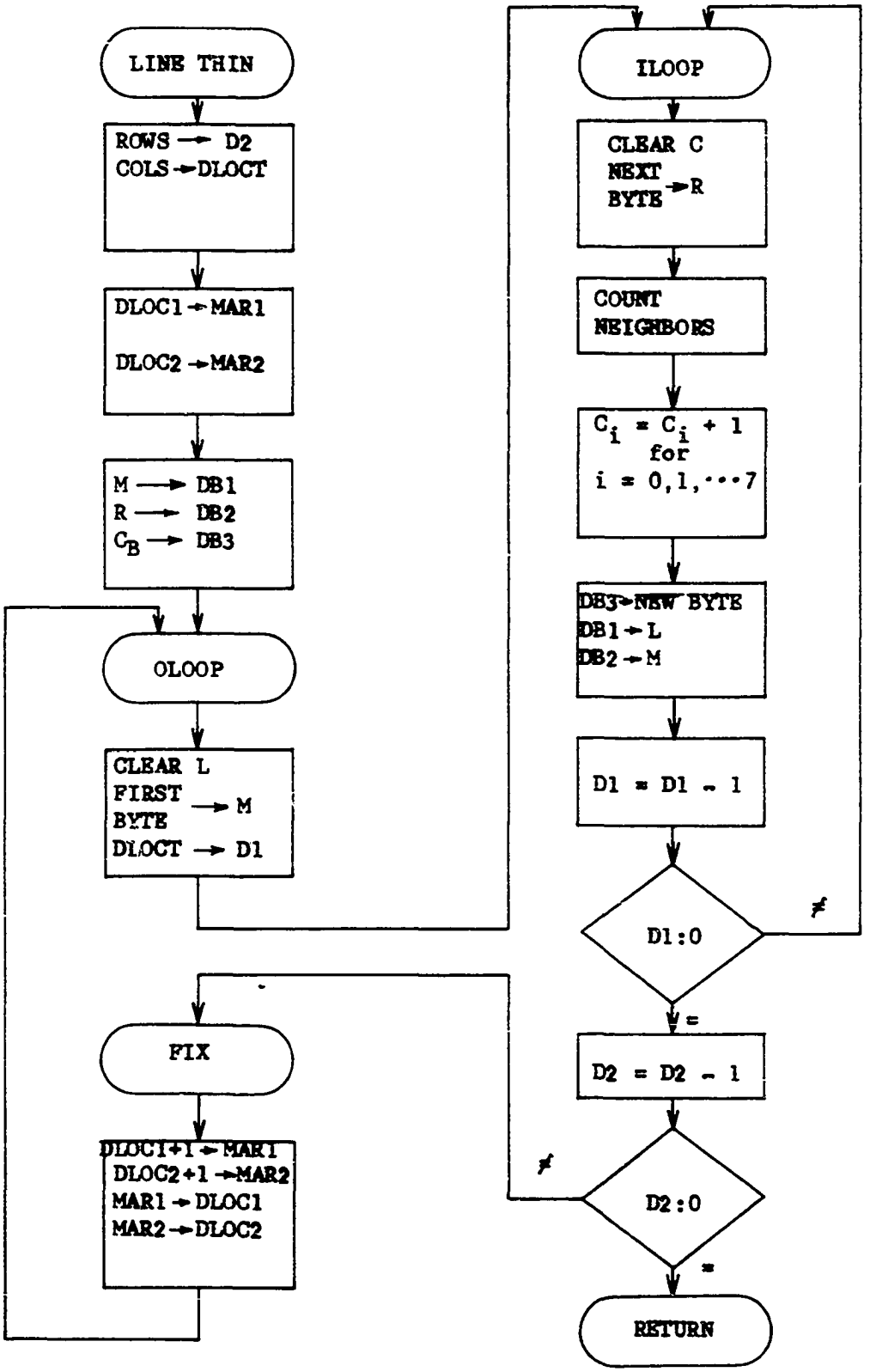

LINE THIN
ROWS → D2
COLS → DLOCT
DLOC1 → MAR1
DLOC2 → MAR2
M → DB1
R → DB2
C_B → DB3
OLOOP
CLEAR L
FIRST BYTE → M
DLOCT → D1
FIX
DLOC1+1 → MAR1
DLOC2+1 → MAR2
MAR1 → DLOC1
MAR2 → DLOC2
ILOOP
CLEAR C
NEXT BYTE → R
COUNT NEIGHBORS
$C_i = C_i + 1$ for $i = 0,1,\cdots 7$
DB3 → NEW BYTE
DB1 → L
DB2 → M
D1 = D1 - 1
D1:0
≠
=
D2 = D2 - 1
D2:0
≠
=
RETURN

Table 4. Symbolic program for a line thinning operation

Label	Symbolic operation	Symbolic parameters	Comments
LINE THIN	MA1	ROWS, COLS	parameters to MAR1
	WDL	MAR1, DLOCT	store temporarily
	RDL	D2, DLOCT	put rows in D2
	RDL	MAR1, DLOC1	location of first byte of unprocessed array
	RDL	MAR2, DLOC2	location of first byte of processed array
	RBS	M, R, C_B	setup data buses
OLOOP	CLEAR	L	
	R1	8, M, MAR1	first byte to M
	RDLB	D1, DLOCT	columns to D1
ILOOP	CLEAR	C_A, C_B, C_C, C_D, C_E	
	R1	8, R, MAR1	next byte to R
	SCS	L, M, R↓; L, M, R	three commands to count neighbors
	SCS	; L, M, R	
	SCS	L, M, R↑; L, M, R	
	SCL	; 11	increment counters by one
	W1	8, DB3, MAR2	store processed character
	BRT	L, M	
	JNZD1	ILOOP	
	JNZD2	FIX	
	SR		return to main program

Table 4 (Continued)

Label	Symbolic operation	Symbolic parameters	Comments
FIX	RDL	MAR1, DLOC1	the next several commands set MAR's for new row
	R1	1, NUL, MAR1	
	WDL	MAR1, DLOC1	
	RDL	MAR2, DLOC2	
	R1	1, NUL, MAR2	
	WDL	MAR2, DLOC2	
	JUC	OLOOP	

Multiple Byte Addition

This subroutine can be used to compute the sum of two positive, multiple byte operands. The flowchart is given in Figure 8, and the symbolic program is given as Table 5.

The assumptions used in writing the program are that the beginning locations (ALOC1, BLOC1) of the two operands, A and B, are given as parameters in the first two subroutine commands, the sum, S, is placed in the locations formerly occupied by A, the length of the operands in bytes is given as a parameter in the subroutine and that the operands are stored with the least significant bytes first.

Figure 8. Flowchart for the addition of two positive multiple byte numbers

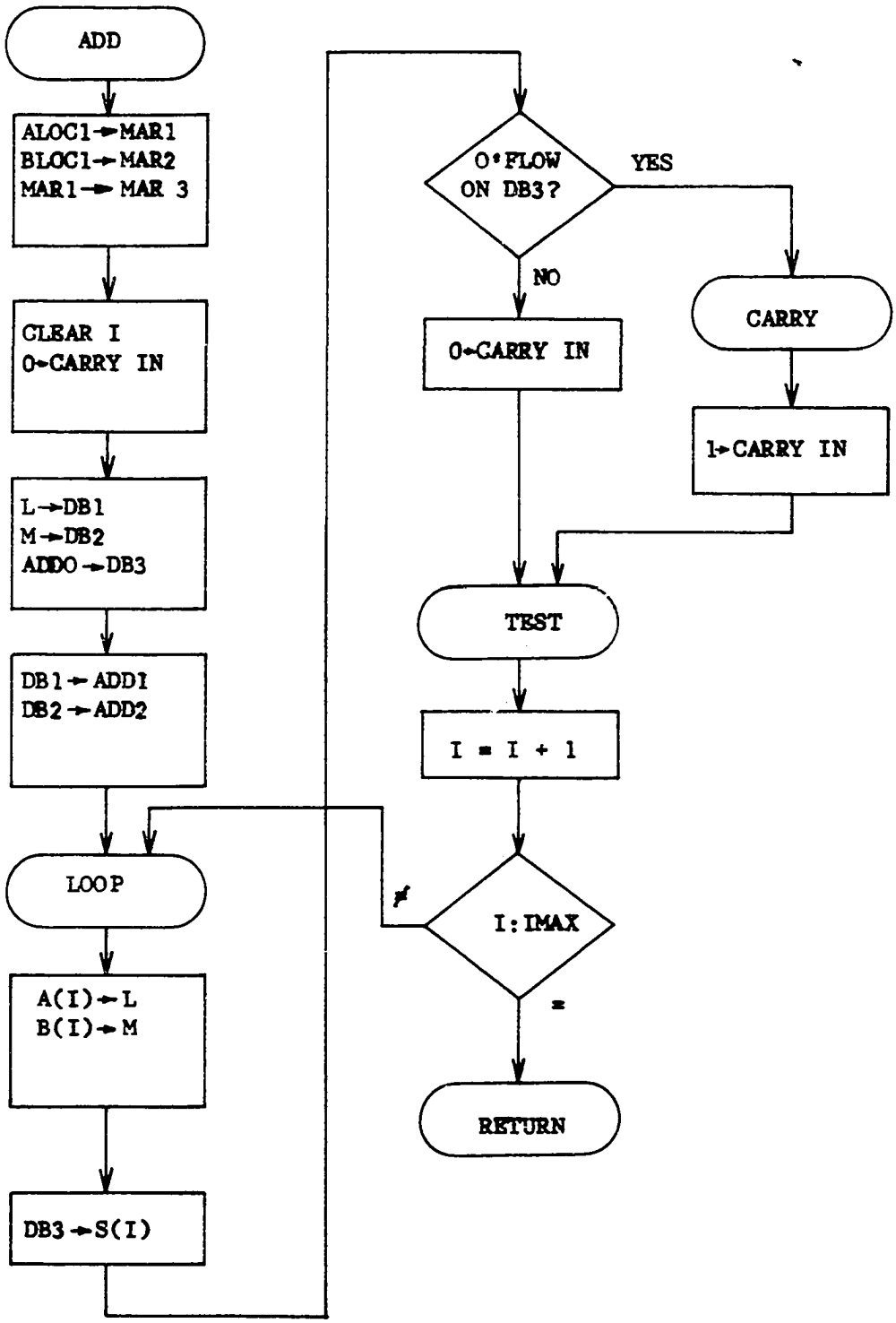

ADD
ALOC1→MAR1
BLOC1→MAR2
MAR1→MAR 3
CLEAR I
0→CARRY IN
L→DB1
M→DB2
ADDO→DB3
DB1→ADD1
DB2→ADD2
LOOP
A(I)←L
B(I)←M
DB3→S(I)
O'FLOW ON DB3?
YES
NO
0→CARRY IN
CARRY
1→CARRY IN
TEST
I = I + 1
I:IMAX
≠
=
RETURN

Table 5. Symbolic program for the addition of two positive multiple byte numbers

Label	Symbolic operation	Symbolic parameters	Comments
ADD	MA1	ALOC1	operand A location to MAR1
	MA2	BLOC1	operand B location to MAR2
	MOVE	MAR3, MAR1	operand C location to MAR3
	SI	0	clear the I register
	SCI	0	set carry in to zero
	RBS	L; M; ADDO	setup data buses
	BAS	0, DB1; 0, DB2	setup adder inputs
LOOP	R1	1, L, MAR1	get i^{th} byte of operand A
	R1	1, M, MAR2	get i^{th} byte of operand B
	W1	1, DB3, MAR3	store i^{th} byte of operand
	JOV3	CARRY	test to determine carry into next byte
	SCI	0	carry in is zero
TEST	ITI	IMAX	is addition complete?
	JUC	LOOP	not complete
	SR		return to program
CARRY	SCI	1	carry in is one
	JUC	TEST	back to see if addition is complete

Single Byte Multiplication

The flowchart for this subroutine is given in Figure 9, and the symbolic program is presented in Table 6. The subroutine can be used to compute the two-byte product, P, of two single byte operands, A and B.

It is assumed that A and B are located in successive byte locations of the scratchpad with A in the location called ALOC. ALOC is considered to be a parameter in the first subroutine command. The product P is placed in the two byte locations that originally held A and B.

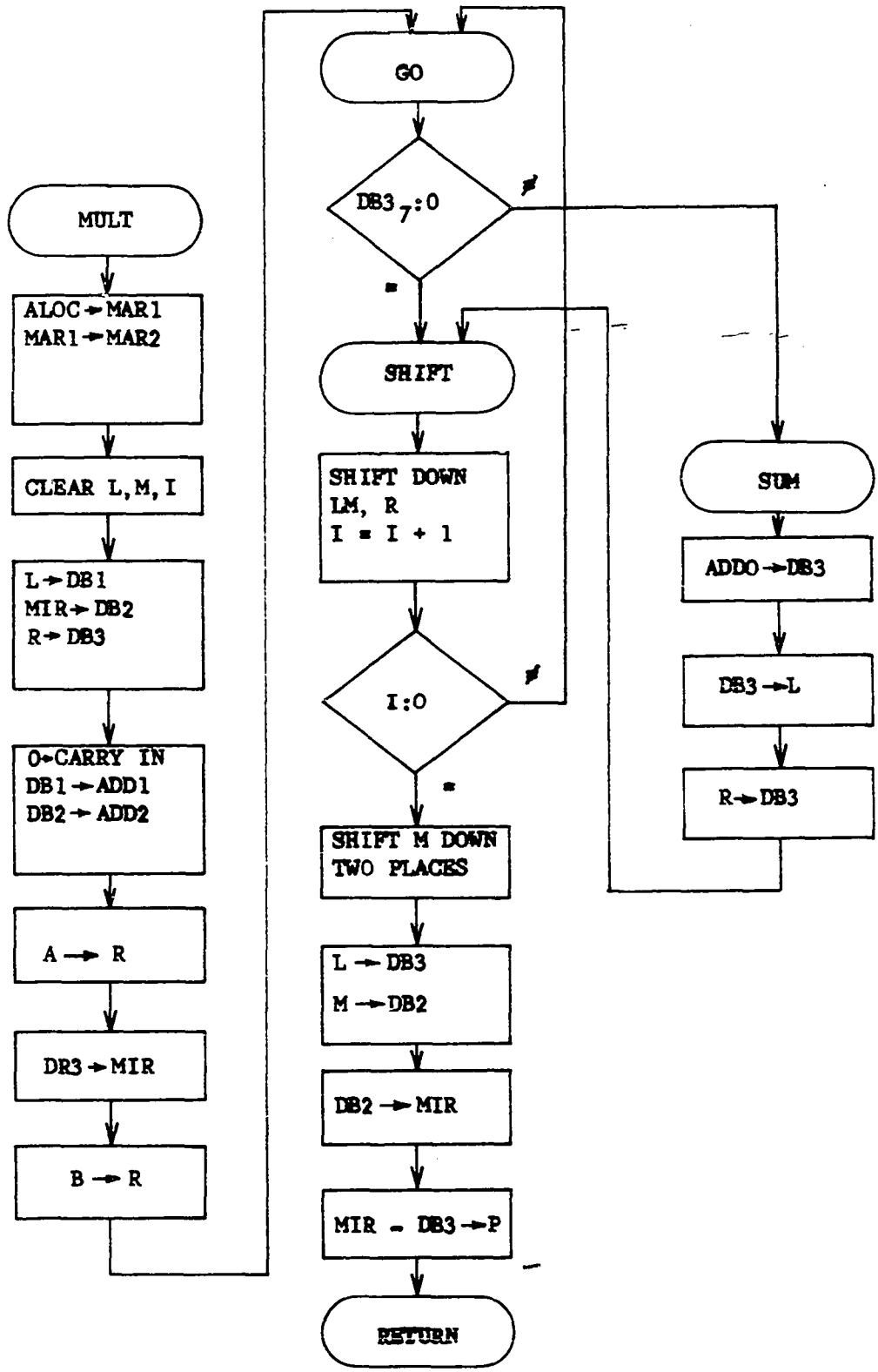
Table 6. Symbolic program for muliplying two single byte members

Label	Symbolic operation	Symbolic parameters	Comments
MULT	MA1	ALOC	operand A location to MAR1
	MOVE	MAR2, MAR1	operand C location to MAR2
	CLEAR	L, M	
	SI	0	clear I
	RBS	L; MIR; R	setup buses
	SCI	0	set carry in to zero
	BAS	0, DB1; 0, DB2	setup adder inputs
	R1	1, R, MAR1	operand A to R register
	BRT	; ; MIR	move A to MIR
	R1	1, R, MAR1	operand B to R register
	JLS3	SUM	

Table 6 (Continued)

Label	Symbolic operation	Symbolic parameters	Comments
SHIFT	SNC	L, M, R↓; 00, 11, 10	shift partial product and multiplier
	ITI	0	complete?
	JUC	GO	not complete
	SNC	M↓; , 10,	shift M down
	SNC	M↓; , 10,	shift M down
	RBS	; M ; L	setup buses
	BRT	; ; MIR	first byte of P in MIR
	W2	, MIRDB3	store P
RETURN	SR		return to main program
SUM	RBS	; ; ADDO	next three commands to perform the summation
	BRT	; ; L	
	RBS	; ; R	
	JUC	SHIFT	

Figure 9. Flowchart for single byte multiplication


MULT
ALOC → MAR1
MAR1 → MAR2
CLEAR L, M, I
L → DB1
MIR → DB2
R → DB3
0 → CARRY IN
DB1 → ADD1
DB2 → ADD2
A → R
DR3 → MIR
B → R
GO
DB3 7:0
≠
=
SHIFT
SHIFT DOWN
LM, R
I = I + 1
I:0
≠
=
SHIFT M DOWN
TWO PLACES
L → DB3
M → DB2
DB2 → MIR
MIR - DB3 → P
RETURN
SUM
ADDO → DB3
DB3 → L
R → DB3