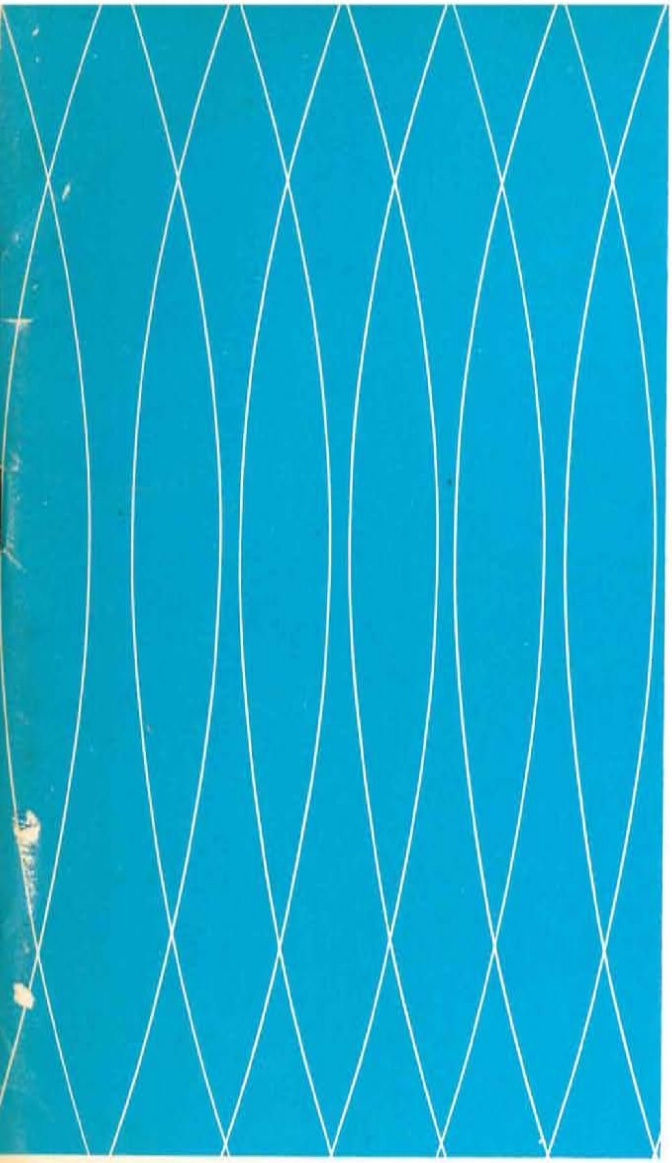


Ernie

PROGRAMMING MANUAL FOR THE MOBIDIC COMPUTER

SYLVANIA ELECTRONIC SYSTEMS
A unit of
GENERAL TELEPHONE & ELECTRONICS

PROGRAMMING MANUAL

for the

MOBIDIC A COMPUTER

APRIL 1960

SYLVANIA ELECTRONIC SYSTEMS
A Division of Sylvania Electric Products Inc.
DATA SYSTEMS OPERATIONS
189 B Street - Needham 94, Massachusetts

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

MOBIDIC is a large-scale, general-purpose, mobile digital computer developed by Sylvania Electronic Systems for the U. S. Army Signal Research and Development Laboratories. The computer is completely transistorized and has been designed for extremely reliable operation under a wide range of environmental conditions to be encountered in world-wide use with the field armies. Although MOBIDIC was originally designed for mobile installation, its advanced concepts of reliability and flexibility are equally advantageous for fixed-plant or strategic installation. Because of its operational and design characteristics, MOBIDIC offers many advantages over other existing commercial and military computers in most data-processing applications. Features of particular advantage in MOBIDIC are small size of equipment, fast arithmetic speed, large memory capacity with extremely fast access, flexible order code, wide range of input-output media, and the avoidance of the usual requirement for air-conditioning and temperature control.

The present programming manual is prepared to serve both as an introduction to the Mobile Digital Computer system (MOBIDIC) and as a programming manual for MOBIDIC A. The general organization of the MOBIDIC system is described in Section II. The structure of the information to be processed and of the machine orders is also described, and a brief explanation is given of the logical organization of the computer. The individual machine orders are described in Section III and suitable examples are given to demonstrate the capabilities of the various types of orders. Section IV is devoted to a discussion of programming techniques which may prove helpful during the problem preparation. The console operations are also discussed in this section and certain methods are suggested for program check-out and error detection.

II. MACHINE CHARACTERISTICS

A. General Organization

The MOBIDIC System is composed of the following machine units:

Central Computer
Memory Units
Input-Output Converters
Flexowriter Output Units
Paper-Tape Readers
Paper-Tape Punches
Magnetic-Tape Units
Communications Equipment
Power Supplies and Air-Conditioning Units
Control Console

These units are designed to be stored in a van and will operate when power lines are connected to the van. The general organization is shown in Figure 1. The boxes in the figure designate the various machine units and the directed lines designate the flow of information. The actual number of machine units used in each case is variable, and depends on the particular application under consideration.

In addition to the standard input-output units shown in Figure 1, other devices such as card equipment, high-speed printers, displays, etc. are available.

Although MOBIDIC is designed for installation in a van, it is equally adaptable to fixed plant operation. The rugged design, small size, and extreme resistance to environment imply that minimum installation provisions are required.

B. Information Structure

Information in the machine is represented in terms of binary digits, or bits. A binary one is represented by a voltage of a certain magnitude or by a spot on magnetic tape, while a binary zero is represented by the absence of that voltage or spot.

For purposes of information processing, the binary digits are grouped in units called words. Each word contains 38 binary digits. Of these 38 bits, 37 are information bits and one is a parity check bit chosen as zero or one in such a manner that the total number of binary ones in the word is odd. One of the 37 information bits in each word is used to represent the sign of a numeric quantity; if the information stored is not a numeric quantity, this bit is ignored.

It is necessary to distinguish between two types of words: data words and order words. Data words are words which contain information to be processed, while order words are those words which prescribe the manner in which the information is to be processed. A data word contains either purely numeric information or alphanumeric information

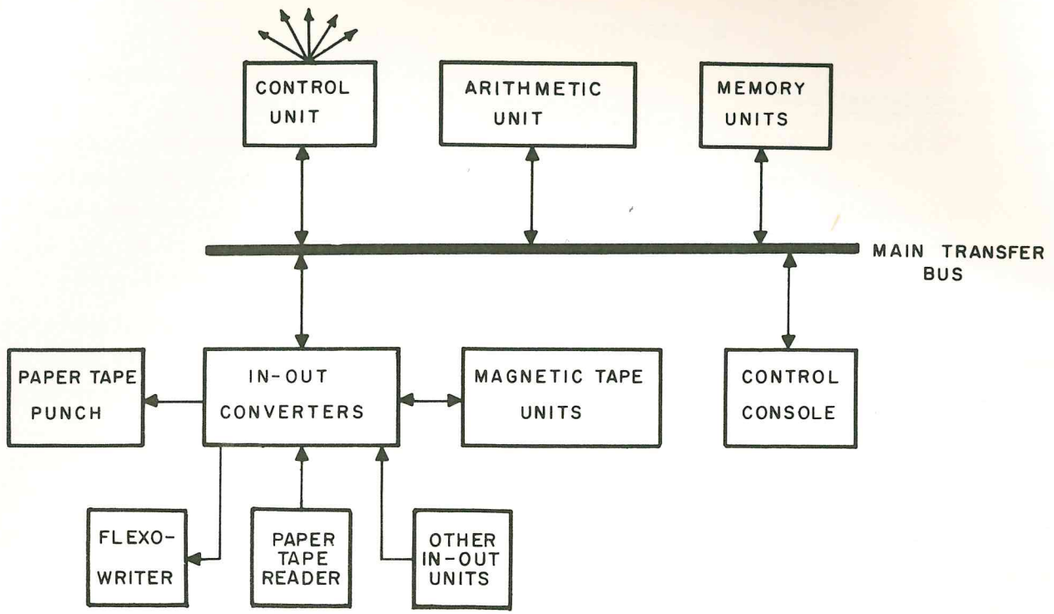


Figure 1. General Organization of MOBIDIC System

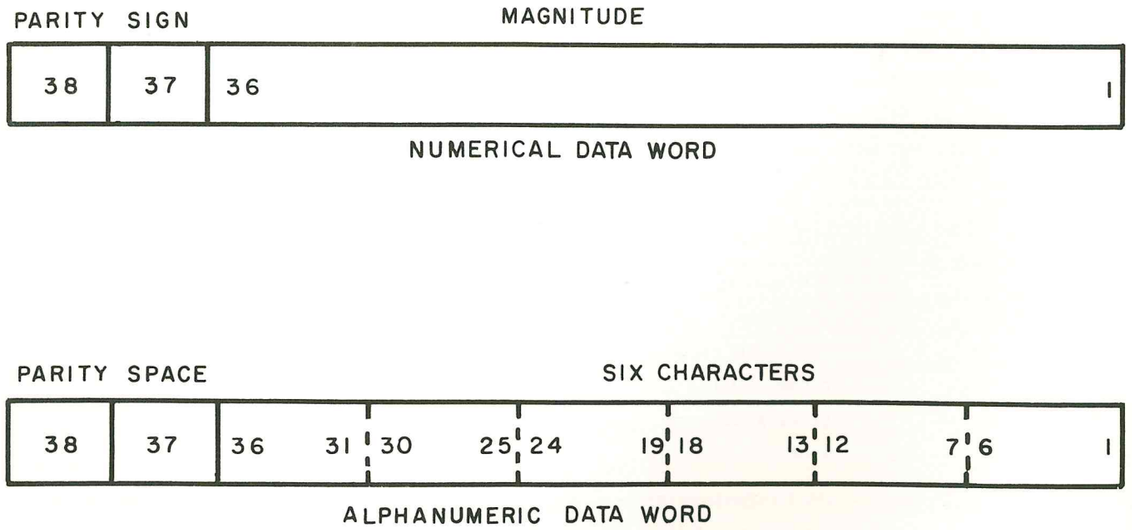


Figure 2. Data Word Format

composed of both numeric digits and alphabetic characters. The word format for the two kinds of data words is shown in Figure 2. The binary digit positions are numbered from 1 - 38 starting with the rightmost bit position. In a numerical data word successive binary digits represent successive powers of 2. Since the binary point is understood to be placed between bits 36 and 37, numbers ranging from zero to $1 - 2^{-36}$ in absolute value can be represented. For example, the number 0.10111 can be translated into decimal notation as $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} = 1/2 + 1/8 + 1/16 + 1/32 = 23/32 = 0.71875$.

In an alphanumeric data word, the binary digits are treated in groups of 6 bits, each group representing one alphanumeric character. Six alphanumeric characters can therefore be represented in one data word. For example, the letter A might be represented by 000010, and the digit 4 by 110100. Logical operations such as sorting or rearranging can be performed either on pure numerical or on alphanumeric data, while arithmetic operations are normally performed on numerical information only.

The format of the instruction words is similar to the data words. The first 36 bit positions are used to store the instruction itself; the 37th bit position is not used, and the 38th bit position is used for parity checking as before. It is necessary to distinguish between instructions which transfer information to and from the input-output units and the remaining instructions which are not concerned with the input-output units. The format in both cases is shown in Figure 3.

The 36 bit positions comprising the standard instruction words are separated into 4 parts designated respectively by alpha, beta, gamma and operation code. The alpha part is composed of bit positions 1 - 15; it specifies the address to be used in an instruction. In general the first 12 bits specify a memory address while bits 13 - 15 specify which one of 7 possible memory units is to be used. Since a number of internal storage registers are addressable, the 8th configuration for bits 13 - 15 represents internal register addresses. The actual register addressed in these cases is specified by bits 1 - 15. Bits 6 - 12 are not interpreted when addressing internal storage registers.

The beta part of the standard instructions comprises bits 16 - 27. These bit positions have several uses depending on the particular instruction being performed. The beta bits, either alone, or in combination with the gamma bits, may be used to specify a second address, or they may be used for indexing. The two low-order bits of the beta-part (bits 16 and 17) are also used to control the trapping mode, while bits 16 to 18 are used for overflow control, as explained in a later section. The gamma part composed of bits 28 - 30 is used primarily for indexing. These bit positions specify which, if any, of the index registers are to be used with the instruction. The function of the index registers is explained in more detail later. For some instructions, gamma is used as part of a second address as previously stated. Bits 31 through 36 designate the operation code which specifies the instruction to be performed. Up to 64 different operations can be defined. If the 37th bit position were made a part of the operation code, an additional 64 operations could be made available.

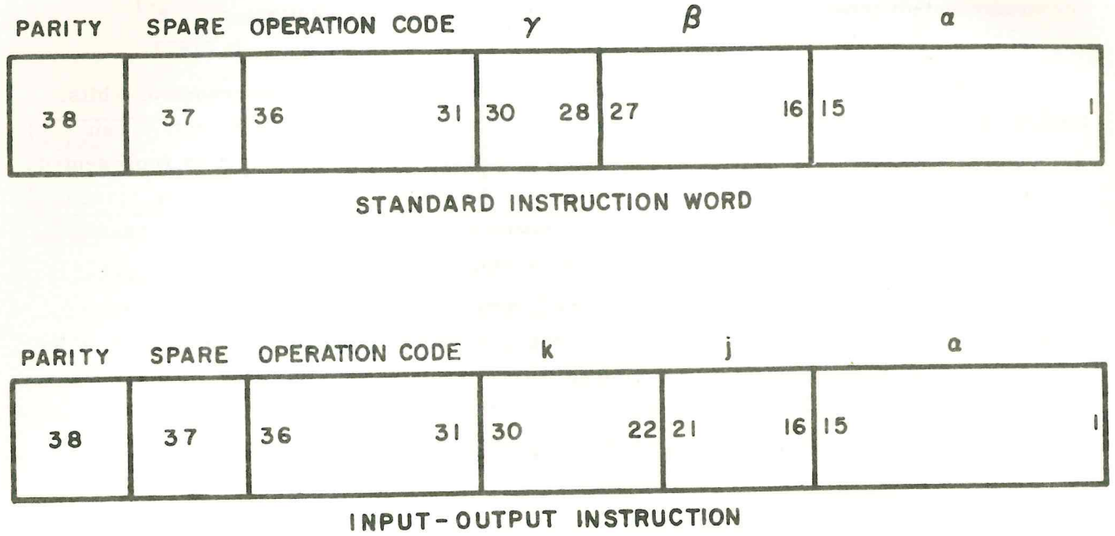


Figure 3. Instruction Word Format

The format for the input-output instructions is similar to that for standard instructions except for the assignments made to bit positions 16 - 30. Bits 22 - 30, called the k bits, are used to specify the amount of information to be processed, for example, the number of words, cards, or blocks, while bits 16 - 21, the j bits, are used to specify the particular input-output device addressed. Each input-output unit has an address and up to 63 input-output units can be used in the system.

During a transfer to or from magnetic tape a number of words may be treated as a unit, called a block. A block will normally contain up to 511 words depending on the particular problem requirements.

C. Logical Organization

1. Sequence Control. Central to the operation of the computer are the information storage units (memory units) and the information processing units. Communication between the various units is provided by the information transfer bus. Each operation is in the nature of a transfer from one unit to another through the transfer bus. For example, information may be transferred from the storage units to the processing unit, and results are then transferred back from the processing unit into storage. Transfer generally takes place in parallel for all binary digits within a given word.

The time taken to execute one complete instruction is called a basic cycle. The time required for one cycle is normally 16 microseconds. Some instructions, such as multiplication and division, may however take considerably longer. Each basic cycle is subdivided into 8 periods, each period being normally two microseconds long. For instructions requiring more than 16 microseconds, any of the 8 periods can be extended in increments of 2 microseconds. The 2-microsecond periods are defined by a timer which generates 8 gating levels in sequence. These levels identify the 8 periods of the basic cycle.

The basic cycle is shown in detail in Table I. The basic cycle operations shown on the left-hand side of Table I take place for all instructions, while the special operations shown on the right-hand side depend on the particular order being interpreted. In Table I, the operations of the ADD instruction are shown as an example.

All operations and information transfers are controlled by the sequence control unit. The instructions are interpreted by this unit and appropriate control pulses are sent to the various parts of the machine to initiate the required information transfers. When an instruction is first extracted from memory, the address specified by the instruction (bits 1 to 15) is first stored in the address register. From there these bits are transferred to the memory address register, as will be explained later. Relative addressing can be used by adding to the contents of the address register the contents of an index register to give the final absolute address.

The following twelve bits of the instruction word (bits 16 to 27) are stored in the X register, while bits 28 to 30 are stored in the G register. From these two registers the

TABLE I. MOBIDIC BASIC CYCLE: ADD ORDER

Operational / Timing Function	BASIC CYCLE OPERATIONS	SPECIAL OPERATIONS FOR ADD MECHANIZATION
TF-1	1. Operand arrives at Memory Output Register	
TF-2	1. Rewrite Operand into Memory	1. Transfer Operand from Memory Output Register to B Register
TF-3	1. (Used for in-out instructions only)	1. If Signs of A and B are alike, add B to A; If Signs are different, form one's complement of B, and add B plus 1 x 2^{-36} to A
TF-4	1. Transfer Contents of Program Counter to Memory Address Register 2. Send a Read Pulse to Memory	1. If B was complemented and there was no overflow, form the one's complement of A and reverse its sign
TF-5	1. Next Instruction arrives at Memory Output Register	1. If B was not complemented and there was an overflow, set the overflow alarm flip-flop 2. If B was complemented and there was no overflow, add 1 x 2^{-36} to A
TF-6	1. Transfer appropriate bits of contents of Memory Output Register to Address Register, X Register, G Register, and Instruction Register 2. Rewrite the Instruction into Memory	
TF-7	1. Step the Program Counter 2. Add the Contents of the Index Register specified by γ to AR	
TF-8	1. Transfer the Contents of AR to the Memory Address Register 2. Send a Read Pulse to Memory 3. Transfer contents of Instruction Register to Decoder - Decoder Lines are energized	

information is transferred either to an index register (bits 16 to 27 only) or to another memory address register, or to an input-output converter, depending on the particular instruction given.

The last 6 bits which make up an instruction and which specify the operation code are transferred to the instruction register and from there they go to the decoder register to be interpreted into the proper sequence of control pulses. Figure 4 shows the break-down of an instruction into its various components before interpretation. The four storage registers which store the various parts of each instruction are not individually addressable.

During the execution of a given instruction the address of the next instruction in sequence is determined. To this effect, the program counter, which holds the address (bits 1 - 15) of each instruction before execution is stepped through consecutive memory addresses. The program counter can be individually addressed.

The program counter store can be used to store the contents of the program counter when a given sequence of instructions is to be interrupted. The original sequence can later be resumed by transferring the contents of the program counter store back to the program counter.

Another method for controlling the sequence of operations to be performed by the machine consists in using the index registers. These registers have a 12-bit capacity and are individually addressable. Any index register can be used to modify a programmed address by adding its contents to the specified address. The index registers can also be used as counters; for example, an index register might specify the number of times a given set of orders is to be executed during a calculation.

2. Input-Output Organization. The organization of the input-output units is shown in Figure 5. A set of in-out converters is provided to act as buffering devices during the transfer of information between input-output units and the other machine units. Each in-out converter is individually addressable and can store one word exclusive of the parity bit; any converter can be used in conjunction with any input-output unit. As many input-output units can normally be used simultaneously as there are converters available to effect the information transfer. However, the maximum number of input or output units of any one kind which can be used simultaneously depends also on the data transfer rates of the particular input-output unit. For example, not more than four magnetic-tape units can be used simultaneously.

The transfer of information between input-output and other machine units takes place independently of the central machine. Computation is interrupted only during access to the memory units. When a word is transferred to an output unit it is broken down into 6-bit characters in the in-out converter before the individual characters are released to the given output unit. Conversely, during an input operation, the individual characters are assembled in the converter as they come in from the input unit, and a whole word (37 bits) is then released to a machine unit when completely assembled in the converter.

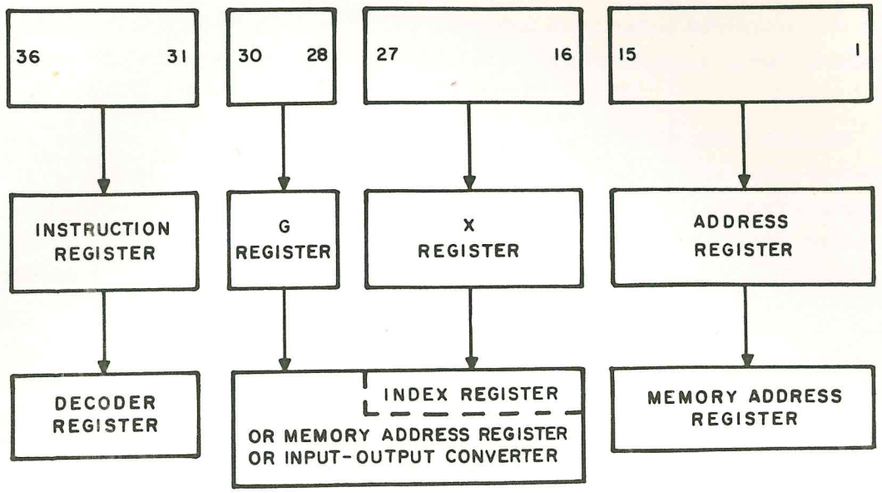


Figure 4. Decomposition of Machine Instruction

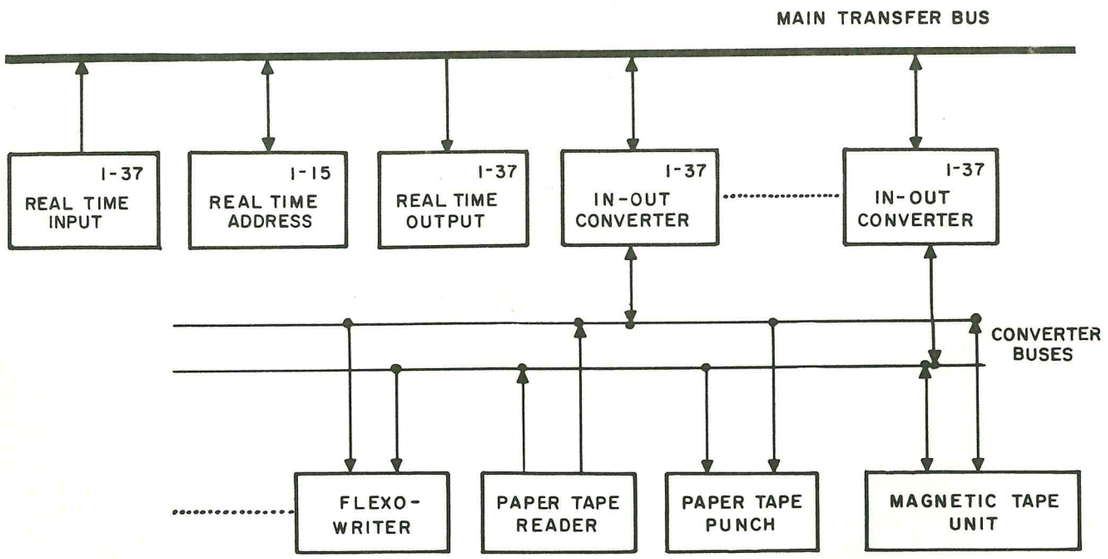


Figure 5. Input-Output Organization

A set of special 40-bit input-output registers is provided to handle asynchronous data arriving at the computer from external sources, or being transmitted to an external medium by the computer. Bits 38 to 40 are used for control functions. The real-time output register is connected to the main transfer bus and is addressable. When a word is assembled in the real-time output register, it is released to the output medium connected to that particular real-time output register.

The information loaded into the real-time input register is transferred either to memory or to an addressable storage register upon completion of the currently executed instruction. If information arrives during an instruction which takes considerable time, such as multiplication or division, the transfer occurs while the instruction is executed without delay in the main program. The real-time address register stores the 15 bits which are necessary to specify the memory location or the location of the addressable storage register which is to receive the information arriving at the real-time input register. The real-time address register can be addressed, and its contents is automatically incremented after each memory access, so as to insure that consecutive memory locations receive the input information and must be set prior to the loading of the real-time input register.

A program interrupt feature allows control to be transferred to memory location 0, if bit position number 40 of the word contained in the real-time input register contains a one. (The sending device must supply this bit). The regular program is then interrupted and the address of the last instruction executed is stored in the B register of the arithmetic unit; from here it can later be retrieved when the regular program is to be resumed. An addressable flip-flop is used to supply the program-interrupt bit for words being sent out through the real-time output register.

3. Memory Organization. Each memory unit consists of a magnetic-core matrix capable of storing 4096 words. A maximum of seven memories can be used for a total of 28,672 words. The basic memory organization is shown in Figure 6.

Each memory unit is provided with a read pulser and a write pulser which provide the control pulses required to read a word from the memory or to write a word into the memory. The memory address register is a 12-bit register which stores the address of the memory location addressed by the read or write pulsers. The memory in-out register acts as a buffer for words being transmitted to and from a memory unit. The parity count is also verified in that register for all words coming out of the memory unit.

4. Arithmetic Unit. The arithmetic unit is composed of three special storage registers, all individually addressable, and of the control circuitry necessary to perform the arithmetic operations. The organization of the arithmetic unit is shown in Figure 7.

The A-register, or accumulator, is a 37-bit register and is the main register in the arithmetic unit. Most arithmetic and comparison operations involve the contents of the accumulator. The accumulator can be reset to zero before entering new information, or else new information can be added to the contents of the accumulator without prior resetting. The accumulator holds the sum, difference, product, or remainder after addition,

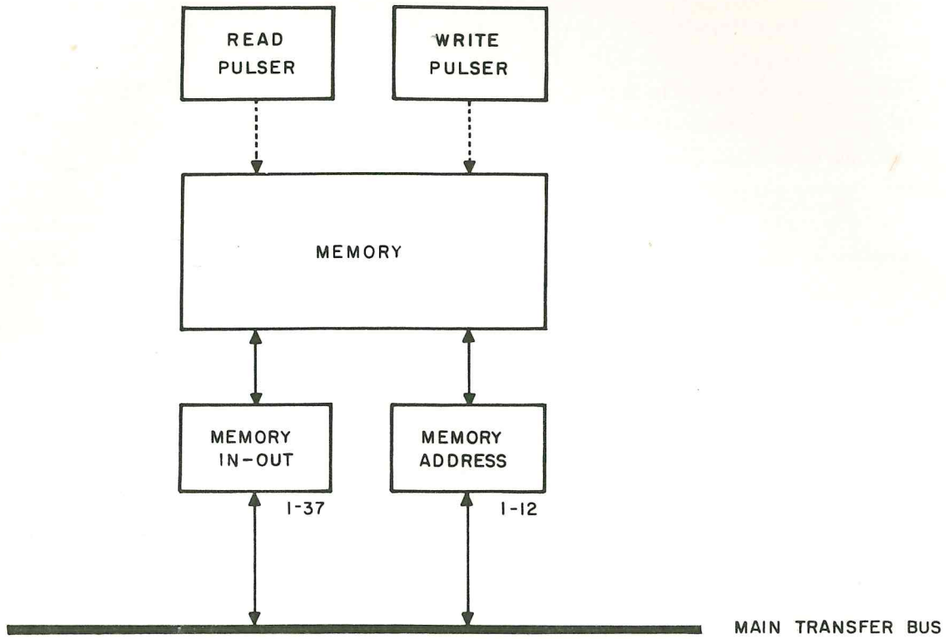


Figure 6. Memory Organization

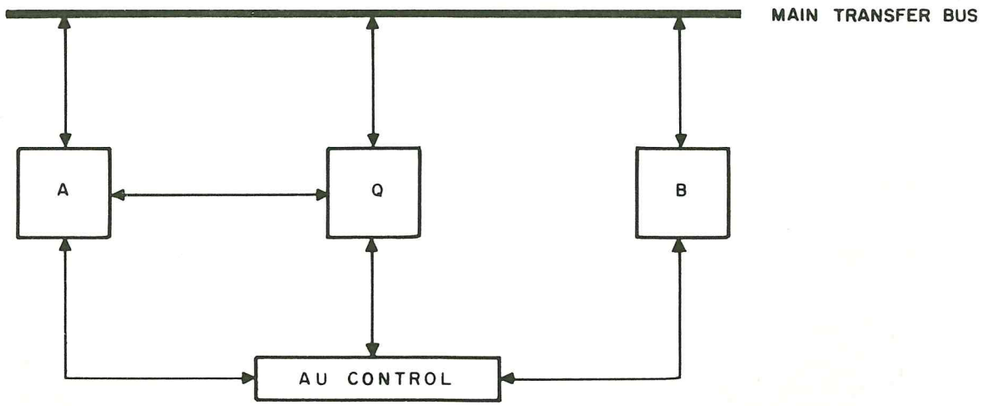


Figure 7. Arithmetic Unit

subtraction, multiplication, or division respectively. It also contains the augend, minuend, multiplicand, or dividend at the start of addition, subtraction, multiplication, or division.

The Q-register is a 37-bit register which has several uses, including the following: At the end of multiplication, it contains the low-order bits of a double-length product (that is, the 36 least significant bits of a 72-bit product). At the end of division, the Q-register always contains the quotient. At the start of division, if a double-length dividend is to be used, the Q-register holds the 36 low-order bits of the 72-bit dividend. The Q-register is also used during the "mask" instruction, as will be explained later. Finally, it can be joined with the A-register, as shown in Figure 7, to perform "shift" and "cycle" operations on double-length numbers. The Q-register will contain the low-order bits of the double-length number in each case.

The B-register is a 37-bit register used to hold one of the operands of an arithmetic operation. It is used to store the addend, subtrahend, multiplier, divisor, or another quantity during the execution of an instruction.

5. Real-Time System

The Real-Time capabilities of MOBIDIC permit its connection to certain external sources of real-time data such as radar sets, weather stations, drone aircraft, or other MOBIDIC computers by the use of a communication link and communication link terminal device called the Kineplex. The basic constituents of the MOBIDIC Real-Time System are the following: the Real-Time Input Register (RIR) which assembles incoming real-time data characters, received from the communication link via the Kineplex, into full length computer words; the Real-Time Output Register (ROR) which divides full length computer words into real-time data characters for transmission through the Kineplex and the communication link; and the Real-Time Address Register (RAR) which specifies an address (in memory or a register) for information coming into the RIR. Real-time data characters are each six bits in length. During both sending and receiving of real-time data characters, the six bits of each character are transmitted in parallel and the characters are transmitted serially. Incoming data are handled in synchronous operation with the communication link. Figure 1 shows the real-time link between 2 MOBIDICS.

MOBIDIC "A" - The RIR is a 40-bit non-addressable register in which bits 1-37 comprise the standard MOBIDIC word, bit 38 is a spare bit, bit 39 is a busy bit which is set when memory access is necessary, and bit 40 is a program interrupt bit. When bit 40 is set by the transmitting computer, it causes, in the receiving computer, an unconditional transfer to location zero, at the completion of the instruction during which the bit 40 was set, and the contents of the Program Counter are sent to the B-register.

When the "ready" signal is set, a character arrives in stages 1-6 of the RIR. Each character is accompanied by a strobe pulse and a parity bit. The strobe pulse causes the "ready" signal to be turned off and is not set again until 2 microseconds

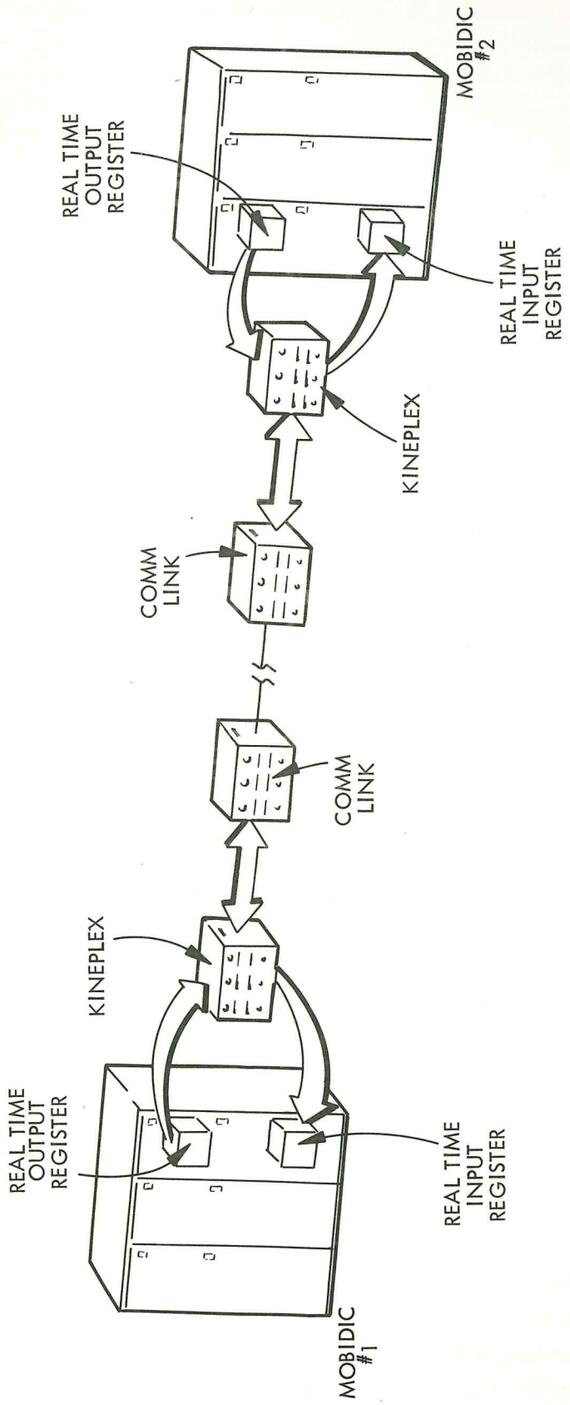
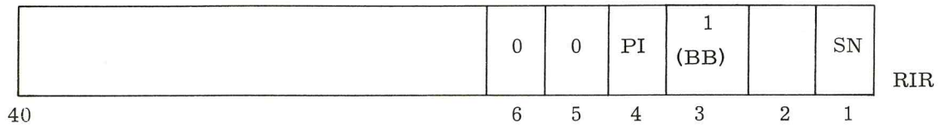
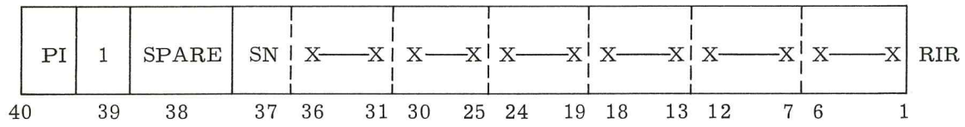


Figure 8. Real Time Link Between Two MOBIDICS

after the strobe pulse itself has been shut off. A parity check is made as each character enters RIR. If a parity error is detected, the Real-Time Parity Error Flip-Flop (RPE) is set. RPE, which is addressable, remains set until cleared by a SNR instruction in the program. The first character which has entered the RIR is a control character. It will appear as follows:



This control character is followed by six data characters. As each new character arrives, the characters previously received are shifted left in the RIR until after the sixth data character. At that time, the control character is in the following position:

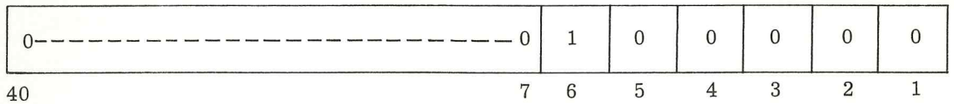


As soon as the "1" is shifted into bit position 39 of the RIR, the real-time busy bit is set and memory access takes place as soon as the busy bit is sampled by the basic timing cycle. Since one character may be received every six microseconds, and since there are seven characters per word, 42 microseconds plus 22 microseconds (maximum memory access time) or 64 microseconds is the minimum transmitting time. When memory access takes place, the contents of the RIR are placed in the memory location of one of the allowable addressable registers (ACC, Q-register, B-register, PC, PCS) specified by the 15-bit addressable Real-Time Address Register (RAR). The RAR is automatically incremented by one each time memory access takes place, except when one of the allowable addressable registers is specified.

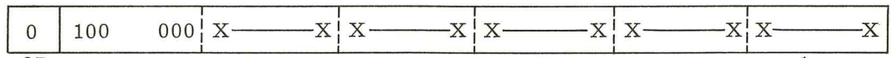
The Real-Time Output Register (ROR) is a 40-bit register, addressable by any instruction which can store a word in it. Bits 1-37 comprise the standard MOBIDIC word. Bit 38 is a sensable flip-flop (ROR 38, octal address 0106), with no particular use at this time. Bit 39 is a busy bit. It is a sensable flip-flop (ROBB, 0105) and is set when a word is stored in the ROR and reset when the last character has left the ROR. The programmer cannot set or reset this flip-flop; however, he can sense it to see whether the ROR is available for another word since there is nothing mechanical to prevent the programmer from destroying the contents of the ROR by placing a new word into it before all the characters of the previous word have left. Bit 40 is a sensable flip-flop (ROPI, 0101) which can be set by a sense and set (SNS) instruction so that a program interrupt will be caused in the receiving computer.

Other MOBIDICS - With the addition of new flip-flops and a change in the character transfer operation, the real-time system becomes even more flexible and capable. Characters may be received every four microseconds; the transmitting time per word is variable, since the number of characters per word depends upon the type of data received.

Three types of data can be received into the RIR, now a 37-bit non-addressable register. These are alphanumeric data, signed binary data and control characters. For alphanumeric data, the Real-Time Interpret Sign Flip-Flop (RISN, 0156) is set. The RIR is normally cleared when information is sent from it to the location specified by the RAR. At this time, the binary data 100 000 is placed in bits 1-6 of the RIR as follows:



The "ready" signal is set, the RIR then shifts left six places, and a character enters into BIR 1-6. Parity is checked and, if a parity error is detected, the Real-Time Parity Error (RPE, 010) is set as in MOBIDIC "A". This process is repeated five times until the RIR looks like the following:



As soon as the leading "1" is forced out of the RIR by a sixth character coming in, a busy bit flip-flop is set and memory access takes place to the location specified by the RAR. Since a maximum of 22 microseconds must be allowed for memory access, no further characters will be accepted into the RIR until the memory access takes place, the busy bit has been reset, and the RIR has been cleared.

In order to receive signed binary characters, the RISN flip-flop must first be set. The RIR will then accept seven characters per word, provided the first character received is an alphanumeric zero (110000) or 1 (110001) to indicate the sign. Reception is identical to that used in the non-interpret sign mode, except the binary code (100 000) is not inserted into RIR, 1-6; the alphanumeric sign serves in its place, so that after six characters have been received the alphanumeric sign will be in bits 36-31, and receipt of the seventh character will set the busy bit and terminate the operation. If the first character is not 110 000 or 110 001, the real-time input system will continue to accept characters from the transmitting unit. When the register is full, each new character will enter RIR 1-6 and the character in RIR 31-36 will be last. This will continue until a 1 is detected in RIR 35. When this happens, the busy bit is set and the operation terminated.

Fieldata code control characters must be handled somewhat differently since only the low-order six bits of any character are recorded in RIR (and thus in memory), and a control character cannot be recognized by only the low-order six bits. When a control character is received in the RIR, the busy bit flip-flop is immediately set and the Real-

Time Input Control Bit Flip-Flop (RINC) is set in the Central Processor. No further characters will be accepted and the contents of RIR are sent to memory. A program interrupt feature is also available. If the Real-Time Input Program Interrupt Flip-Flop (RCI) is also set, the next instruction is automatically trapped. The program can then recognize the control character by noting the address in RAR and using a suitable SNR instruction to clear the RINC.

The same three types of data evidently can also be sent from the ROR. For alphanumeric data (non-interpret sign mode), the ROR is cleared before new data is transferred in and the real-time busy bit (ROBB) is then set. If the receiving device has sent out a "ready" signal, one character is transmitted to the receiving device, accompanied by a control line (which in this mode will always be zero) and a parity line. After six characters have been transmitted, the ROBB flip-flop is cleared indicating that the ROR is ready to receive another word from the processor.

Transmitting algebraically signed data is done in the same manner as transmitting alphanumeric data, except that prior to loading ROR the program must set the Real-Time Output Interpret Sign Flip-Flop (ROSN); thus, seven characters instead of six will be transmitted.

Transmitting control characters is essentially a single character mode under program control. As before, the ROBB is set as data is transferred into the ROR. The Real-Time Output Control Character Flip-Flop (ROTC) must be set by an SNS instruction before loading ROR. After the first character is sent out, ROBB will be cleared and must be set again by the program in order to send the next character. The program itself must keep track of the number of characters.

6. Console Organization. A control console is provided as part of the MOBIDIC system. It is used by the machine operator for manual control of the machine and by maintenance personnel for repair and maintenance. The control switches and indicators permit the operation of the computer in a number of different modes; for example, the computer can be made to stop after the execution of each instruction if this seems desirable. The operator can also reset the memory and register contents to zero; he can introduce information manually into any register, and read-out the contents of any memory location or register by means of a set of indicator lights. In general, the console allows a certain amount of manual intervention during the execution of a program, and provides the required flexibility for efficient operation of the computer.

III ELEMENTS OF PROGRAMMING

The individual machine orders which are used to determine the processing sequence are described in this section. A few simple examples are given to illustrate the use of the various instructions. A number of orders whose function is related are grouped together; thus orders for internal data handling are first described, followed by arithmetic orders, editing orders, orders for sequence control, indexing orders, and input-output orders.

In each case, the functional name of the order is followed by its operation code in octal notation and by its mnemonic abbreviation. The time required to execute the operation is also indicated. Finally a brief explanation is given of the action performed by each order. Unless otherwise noted, each order can be indexed, the particular index register to be used being indicated by the γ part of the instruction.

The following conventions and abbreviations are used in the remainder of this section:

a) Unless otherwise stated, all numeric data words and order words are in octal notation.

b) Since the sign position of an instruction has no influence on its operation, it is normally disregarded.

c) The symbol C( ) is used to indicate the contents (sign and magnitude) of the location given in parentheses. For example, C(A) indicates the contents of the accumulator.

d) A subscript shown with C( ) specifies the contents of the associated bit position(s) of the word. For example, $C(141)_{17}$ indicates the content of bit position 17 of memory location 00141. $C(A)_{1-15}$ stands for the contents of bit positions 1 through 15 (inclusive) of the accumulator. $C(Q)_{sn}$ refers to the value of the sign of the Q register.

e) A numeric subscript shown with either A, Q, or B specifies a bit position of the word. For example, A_{10} stands for bit position 10.

f) Normally, an abbreviated instruction is written: OPERATION CODE γ, β, α. However, if a given subdivision of the instruction can have no effect on its operation, its designation is left out. For example, CLA$\gamma\alpha$ indicates that the CLEAR AND ADD instruction is independent of the bit configuration shown in the beta part.

Table II shows the octal addresses assigned to the various addressable locations within the computer. These addresses are generally shown in the α part of the instructions.

TABLE II

ADDRESSABLE REGISTERS AND MEMORY UNITS

Register Address (Octal)	Register Name	Code
00000 - 07777	Memory Unit Zero	
10000 - 17777	Memory Unit One	
20000 - 27777	Memory Unit Two	
. . .	. . .	
60000 - 67777	Memory Unit Six	
70001	Index register one	(I^1)
70002	Index register two	(I^2)
70003	Index register three	(I^3)
70004	Index register four	(I^4)
70010	Accumulator	(A)
70011	Q register	(Q)
70012	B register	(B)
70013	Program Counter	(PC)
70014	Program Counter Store	(PCS)
70015*	Instruction register of converter receiving the input order	(CRI)
70020	Word Switch Register	(WSR)
70021	Real Time Address Register	(RAR)
70022	Real Time Output Register	(ROR)
70030	Instruction register of first in-out converter	(CIS1)
70031	Instruction register of second converter	(CIS2)

*When used as the address of an input instruction, it tells the converter to store the contents of its buffer register into its instruction word register.

A. Internal Data Handling

The instructions which are used primarily for moving words or parts of words from one element of the central computer to another are described first.

LOAD, 51, LODγβα, 18 μsec.

Replace the contents of the addressable register (not a memory location) specified by β with C(α); α refers to a memory address or an addressable register. C(α) is unchanged. C(B) is replaced by C(α).

STORE, 50, STRγα, 16 μsec.

Replace C(α) with C(A). C(A) remains unchanged.

CLEAR AND ADD, 10, CLAγα, 16 μsec.

Replace C(A) with C(α). C(α) remains unchanged.

REPLACE ADDRESS, 54, RPAγα, 22 μsec.

Replace $C(\alpha)_{1-15}$ with $C(A)_{1-15}$. C(A) and $C(\alpha)_{16-sn}$ remain unchanged. C(B) is replaced by C(A).

REPLACE THROUGH MASK, 55, MSKγα, 22 μsec.

Replace the bits of C(α) including sign bits with those bits of C(A) corresponding to the "ones" of C(Q). C(A), C(Q), and the remainder of C(α) remain unchanged. C(B) is replaced by C(A).

MOVE, 52, MOVγβα, 26μ sec.

Replace C(γβ) with C(α); C(α) remains unchanged. This is a two address instruction in which α specifies one 15-bit address, and γ and β together specify the second 15-bit address. C(B) is replaced by C(α). The MOVE instruction cannot be indexed. If a REPEAT order precedes the MOVE order, a special sequence is followed, as explained later in this section.

The six orders previously defined will now be described in more detail. The LOAD order can be used to transfer a word from any register or memory location to an addressable register. For example (octal notation will be used):

OP CODE	γ	β	α
51	0	0011	00400

will transfer the contents of memory location 400 to the Q register, and

51	0	0011	70001

will replace C(Q) with the contents of index register 1.

CLEAR AND ADD is a special-purpose load order used for transferring a word into the accumulator. The instruction

10	0	0000	00701

will transfer the contents of memory location 701 into the accumulator.

The STORE instruction performs the inverse operation of the CLEAR AND ADD. The contents of the accumulator are transmitted to another register or to a memory location.

REPLACE ADDRESS is normally used to modify the address part of an instruction. The operation of this instruction is illustrated by the following example:

Let the Accumulator contain +073 002 113 211 and memory location 371 contain +400 000 002 765; then the instruction

54 0 0000 00371

stores +400 000 013 211 in memory location 371.

The operation REPLACE THROUGH MASK is similar to that of REPLACE ADDRESS except that it allows the replacement of any bit or combination of bits in a memory location or addressable register. The choice of bits to be replaced is controlled by the binary "ones" of the contents of the Q register; bit positions of α corresponding to zeros of Q are not affected. The following examples will clarify the operation of this instruction.

Example 1:

Let C(A) = +123 456 701 234 = ... 010 011 100
C(Q) = +000 000 000 077 = ... 000 111 111
C(721) = +000 000 000 000 = ... 000 000 000

Execution of 55 0 0000 00721 will leave

C(721) = +000 000 000 034

Example 2:

Let C(A) = -765 432 101 234
C(Q) = -070 605 040 302
C(1200) = +432 107 654 321

After translation into binary form

C(A) = 1 111 110 101 100 011 010 001 000 001 010 011 100
C(Q) = 1 000 111 000 110 000 101 000 100 000 011 000 010
C(1200) = 0 100 011 010 001 000 111 110 101 100 011 010 001

Execution of 55 0 0000 01200 will alter the contents of memory position 1200 as follows:

C(1200) = 1 100 110 010 101 000 010 110 001 100 010 010 001
= - 4 6 2 5 0 2 6 1 4 2 2 1

A short program will now be given using the instructions introduced so far. An instruction is located at address 1361. It is required to isolate the gamma bits (28-30) and address bits (1-15) of the instruction and store them at 1500 and 1501 respectively (memory location 600 contains the word +007 000 000 000):


```
10  0  0000  70000      Clear Accumulator
50  0  0000  01500      Store Zero in 1500
50  0  0000  01501      Store Zero in 1501
10  0  0000  01361      Clear and Add C(1361)
51  0  0011  00600      Load Mask
55  0  0000  01500      Store Gamma in 1500
54  0  0000  01501      Store Address in 1501
```

The first three instructions of the above program are used to clear memory locations 1500 and 1501 to +000000000000. The first instruction can be used to clear the accumulator becauze a zero operand is provided whenever a nonexistent register (in this case 70000) is addressed.

The MOVE instruction is used primarily for transferring words from one memory location to another. It is the only instruction which will transfer the contents of an addressable register, other than A, directly to the memory. The program previously given can be shortened by one instruction if the following two move orders are used to clear memory locations 1500 and 1501:

```
52  0  1500  70000
52  0  1501  70000.
```

The last four instructions of the original program remain unchanged.

As was mentioned above, whenever data is called for from a nonexistent register the result is the same as calling for information from a location containing zero.

B. Arithmetic Instructions

Arithmetic operations are performed in MOBIDIC on binary numbers, including sign and magnitude. As noted earlier the binary point is fixed and assumed to be between bit position 36 and the sign position. The range of numbers which can be represented in MOBIDIC is, therefore, between $-(1 - 2^{-36})$ and $+(1 - 2^{-36})$. However, by using proper scale factors, the programmer is able to perform arithmetic operations in any number range. The introduction into the machine of a number whose absolute value is greater than or equal to one results in overflow. Normally, the computer will set the overflow alarm and halt when an overflow occurs in the accumulator. However, instead of halting the computer, several other options are available to the programmer for handling overflow. These options will be described in detail after the arithmetic instructions have been introduced.

ADD, 12, ADDγβα, 16 μsec.

This instruction replaces C(A) with the algebraic sum of C(A) and C(α); overflow is possible. C(α) remains unchanged. The contents of the B-register depend on the signs of both the accumulator and of memory location α. A detailed explanation is given in Table V at the end of this manual.

SUBTRACT, 16, SUBγβα, 16 μsec.

This instruction replaces C(A) with the algebraic difference between C(A) (minuend) and C(α) (subtrahend); overflow is possible. C(α) remains unchanged. C(B) is again dependent on the signs of A and of α as explained in Table V. If overflow occurs during addition or subtraction, the overflow bit is lost. However, the machine may be halted as will be explained later.

ADD MAGNITUDE, 13, ADMγβα, 16 μsec.

This instruction replaces C(A) with the algebraic sum of C(A) and the absolute value of C(α); overflow is possible. C(α) remains unchanged. C(B) depends on $C(A)_{sn}$ and $C(\alpha)_{sn}$ as explained in Table V.

SUBTRACT MAGNITUDE, 17, SBMγβα, 16 μsec.

This instruction replaces C(A) with the algebraic difference between C(A) (minuend) and the absolute value of C(α) (subtrahend); overflow is possible. C(α) remains unchanged. C(B) depends on $C(A)_{sn}$ and $C(\alpha)_{sn}$ as before.

The following example serves to illustrate the operation of the various Add and Subtract instructions.

C(A)	C(α)	Instruction	New C(A)
+000 013 726 130	+011 620 351 072	ADD α	+011 634 277 222
		SUB α	-011 604 422 742
		ADMα	+011 634 277 222
		SBM α	-011 604 422 742

Since C(α) is positive ADD and ADM give the same result; so do SUB and SBM. The next example illustrates these operations when C(α) contains a negative number.

C(A)	C(α)	Instruction	New C(A)
+125 013 301 075	-734 320 574 001	ADD α	-607 305 272 704
		SUB α	+061 334 075 076*
		ADMα	+061 334 075 076*
		SBM α	-607 305 272 704

*Indicates overflow

It should be noted that if the result of an arithmetic operation is equal to zero, the sign of the accumulator at the start of the operation will be taken as the sign of the result. The following example illustrates this rule.

C(A)	C(α)	Instruction	New C(A)
-123 456 701 234	+123 456 701 234	ADD α	-000 000 000 000
		SUB α	-247 135 602 470
		ADMα	-000 000 000 000
		SBM α	-247 135 602 470

CLEAR AND SUBTRACT, 14, CLSγα, 16 μsec.

This instruction replaces C(A) with the negative of C(α).

CLEAR AND ADD MAGNITUDE, 11, CAMγα, 16 μsec.

C(A) is replaced by C(α) and A_{sn} is set equal to zero.

CLEAR AND SUBTRACT MAGNITUDE, 15, CSMγα, 16 μsec.

This instruction replaces C(A) with C(α) and sets A_{sn} equal to 1.

MULTIPLY, 20, MLYγα, 86 μsec.

This instruction forms the 72-bit algebraic product of C(A) (multiplicand) and C(α) (multiplier). The high-order 36 bits of the product are left in A and the low-order 36 bits in Q. Both A_{sn} and Q_{sn} hold the sign of the product. C(B) is replaced by C(α).

MULTIPLY AND ROUND, 21, MLRγα, 86 μsec.

This instruction forms the 72-bit algebraic product of C(A) and C(α). The high-order 36 bits of the product are left in A and the 36 low-order bits in Q. If $Q_{36} = 1$, then $\pm 1 \cdot 2^{-36}$ is added to C(A), depending on whether $C(A)_{sn}$ is positive or negative. C(B) is replaced by C(α) if $Q_{36} = 0$ and by +000 000 000 000 if $Q_{36} = 1$.

[For purposes of scaling it is often convenient to assume the existence of a fictitious binary point. After a multiplication, the number of bits to the right of the (fictitious) binary point in the 72-bit product is equal to the sum of the number of bits to the right of the binary points in both the multiplicand and the multiplier. For example, if C(A) = +000 000 000 133 and C(1401) = -000 000 007 102, then execution of the instruction 20 0 0000 01401 leaves C(A) = -000 000 000 000 and C(Q) = -000 001 210 566. Under the assumption that the binary point of the multiplicand lies between A_{10} and A_{11}, and the point of the multiplier between positions 4 and 5 of memory location 1401, the binary point of the product will lie between Q_{14} and Q_{15}.]

DIVIDE, 22, DVDγβα, 88 μsec. (18 μsec. if overflow)

This instruction forms the unrounded 36-bit algebraic quotient of C(A) (dividend) and C(α) (divisor). The quotient is left in Q and the remainder in A. The sign of the remainder agrees with the sign of the dividend. If $|C(A)| \geq |C(\alpha)|$ overflow occurs, the division does not take

place, and C(A) and C(Q) are unchanged with the exception of Q_{sn} which is set equal to A_{sn}. C(B) is replaced by C(α).

DIVIDE LONG, 23, DVLγβα, 88 μsec. (18 μsec if overflow)

This instruction is identical with DIVIDE except that the dividend is a 72-bit number. The 36 high-order bits of the dividend are stored in the accumulator and the 36 low-order bits in the Q-register. The sign of the dividend will be the sign of A.

[If the fictitious binary point of the dividend and divisor are in the same relative position, the binary point of the quotient will be to the left of Q_{36} and the binary point of the remainder to the left of A_{36}. Displacement of the point in the dividend results in an equal displacement of the points of both the quotient and the remainder. Displacement of the point of the divisor results in a change of equal magnitude but opposite direction in the quotient.]

NORMALIZE, 37, NRMγα, 18 + n - n(mod 2) μsec., n = number of shifts

$C(A)_{1-36}$ is shifted to the left until a one appears in A_{36}. A_{sn} is not shifted. C(α) is replaced with $+n \cdot 2^{-36}$ where n is the number of shifts executed. If C(A) is zero, then n = +36, and C(A) remains zero with A_{sn} unchanged. The NORMALIZE instruction is useful for computing scale factors during calculations. It is also used for floating point operations and to edit output data.

SHIFT RIGHT, 32, SHRγα, 16 μsec if $\alpha \leq 14$; 2 + α + α (mod 2) μsec if $\alpha > 14$

C(A) is shifted α (mod 128) places to the right. The bit positions vacated at the left are replaced with zeros, and the bits shifted out at the right are lost. A_{sn} is not shifted.

SHIFT RIGHT LONG, 33, SRLγα, 16 μsec if $\alpha \leq 14$; 2 + α + α (mod 2) μsec if $\alpha > 14$

The 72 bits stored in A and Q are shifted α (mod 128) places to the right. The bit positions vacated at the left of A are replaced with zeros and the bits shifted out at the right of Q are lost. A_{sn} and Q_{sn} are not shifted, so that A_1 is shifted into Q_{36}.

SHIFT LEFT, 30, SHLγβα, 16 μsec if $\alpha \leq 9$; 8 + α - α (mod 2) μsec if $\alpha > 9$

C(A) is shifted α (mod 128) places to the left. The bit positions vacated at the right of A are filled with zeros and the bits shifted out at the left of A are lost. A_{sn} is not shifted. If a non-zero bit is shifted out of A_{36}, overflow occurs.

SHIFT LEFT LONG, 31, SLLγβα, 16 μsec if $\alpha \leq 9$; 8 + α - α (mod 2) μsec if $\alpha > 9$

The 72 bits stored in A and Q are shifted α (mod 128) places to the left. The bit positions vacated at the right of Q are filled with zeros and bits shifted out at the left of A are lost. A_{sn} and Q_{sn} are not shifted. Q_{36} is shifted into A_1. If a non-zero bit is shifted out of A_{36}, overflow occurs.

The instructions which can cause overflow are ADD, ADM, SUB, SBM, DVD, DVL, SHL, AND SLL. As mentioned previously, several options are available to the programmer

to handle overflow. A flip-flop called the overflow alarm (OA) is used to indicate to the programmer that overflow has occurred. Normally an overflow causes the machine to halt and to set OA. However, the three low-order bits of the beta part (bits 16, 17, and 18) of instructions which can cause overflow may be used to control the computer in case of overflow. The following table shows the effect of each of the eight possible configurations formed by bits 16, 17, and 18 in case of overflow:

Bits 18	17	16	Numerical Value of β	Before Execution of Instruction	If Instruction Causes Overflow
0	0	0	0	Clear OA	Set OA and halt
0	0	1	1	Clear OA	Set OA
0	1	0	2	Clear OA	Set OA and halt
0	1	1	3	Clear OA	No action
1	0	0	4	No action	Set OA and halt
1	0	1	5	No action	Set OA
1	1	0	6	No action	Set OA and halt
1	1	1	7	No action	No action

The arithmetic instructions previously given are illustrated by the following simple example: It is desired to find

$$Z = \frac{ax + b}{c}.$$

In this example it is assumed that all four parameters are scaled at 2^0, that is, 36 bits are placed to the right of the binary point. It is further assumed that overflow is not possible. Let C(100) = a, C(101) = b, C(102) = c, and C(200) = x, and let it be required to store Z at 500. The following program will calculate the required function:

10	0	0000	00100	Place a in A
21	0	0000	00200	Multiply by x
12	0	0000	00101	Add b
22	0	0000	00102	Divide by c
52	0	0500	70011	Store Z

The beta parts of the ADD and DVD instruction have been coded to halt the machine in case of overflow.

C. Editing Instructions

This class of instructions is used primarily to change the format of internally stored information. This is often required in preparation for an output operation when it is necessary to insure that the information appear in the proper form on the output document. Editing orders are also used to arrange information in a form convenient for some later operation, such as a comparison.

The various instructions are described in detail in the next few paragraphs. Both the NORMALIZE (NRM) and the REPLACE THROUGH MASK (MSK) instructions, which properly belong to this category, were introduced previously for convenience. Examples which illustrate the use of these instructions are given as part of the elementary subroutines in Section IV.

CYCLE SHORT, 34, CYSγα, 16 μsec if α ≤ 14; 2 + α + α(mod 2) μsec if α > 14

The cycle-short instruction forms a closed ring, consisting of the 36 bits of the A register, and cycles the bits α(mod 128) places to the left. Bits passing out of A_{36} enter A_1, so that no information is lost as it is in an ordinary shift operation. A_{sn} remains unaltered.

CYCLE LONG, 35, CYLγα, 16 μsec if α ≤ 14; 2 + α + α(mod 2) μsec if α > 14

The cycle-long instruction is similar to CYS except that the ring is formed by the complete A and Q registers. All 74 bits are cycled in such a way that bits passing out of A_{36} enter A_{sn}, bits passing out of A_{sn} enter Q_1, bits passing out of Q_{36} enter Q_{sn}, and bits passing out of Q_{sn} enter A_1.

LOGICAL ADD, 03, LGAγα, 16 μsec.

This instruction forms the logical sum of C(A) and C(α) including the signs. A Boolean "or" function is performed on a bit-by-bit basis, and the result is left in A. For example, if C(A) = 0···0001011 and C(α) = 0···0011001, then the logical sum C(A) = 0···0011011.

LOGICAL MULTIPLY, 02, LGMγα, 16 μsec.

This instruction performs the Boolean "and" function of C(A) and C(α) including the signs. The result is again left in A. For example, if C(A) = 0···01101011 and C(α) = 0···00111000, then the logical product C(A) = 0···00101000.

LOGICAL NEGATION, 04, LGNγα, 16 μsec.

This instruction places the one's complement of C(α) in the accumulator, including the sign. For example, if C(α) = 0···01011101, then this operation will place 1···10100010 in the accumulator.

The logical operations performed by the last three instructions define a complete Boolean algebra and permit the computation of all binary functions of binary variables. Such functions, often called "truth functions," or "switching functions" are used in logic and in the theory of switching, and are generally useful in the design and application of computing machinery.

D. Sequencing Instructions

The MOBIDIC order code includes several instructions which allow the programmer to control the sequence in which the computer executes the instructions. Most of these "transfer" instructions cause the machine either to leave the normal sequence of instructions or not, depending upon the result of some test or comparison. Such instructions are called "conditional transfers." Transfers of control which do not depend on the results of some test or comparison are called "unconditional transfers."

Transfers may be made only to a memory address. Attempts to transfer to an addressable register not a memory location will result in a halt.

UNCONDITIONAL TRANSFER, 40, TRUγβα, 16 μsec.

The next instruction is taken from memory location α. The beta bits of this instruction are used to control the trapping mode as will be described later in Section IV A.

Transfer instructions are often used to form a "loop." In a loop the same set of instructions is executed repetitively. The program for the computation of:

$$Z = \frac{ax + b}{c}$$

is rewritten below in the form of a loop to show how the same instructions can be used to compute Z for different values of x. The x_i are assumed stored in consecutive memory locations starting at 200 and the Z_i are to be stored consecutively starting at 500. Since this program refers to the storage locations of some instructions, addresses are assigned starting, arbitrarily, at location 1000.

1000	10	0	0000	00100	Place a in A
1001	21	0	0000	00200	Multiply by x_i
1002	12	0	0000	00101	Add b
1003	22	0	0000	00102	Divide by c
1004	52	0	0500	70011	Store Z_i
1005	10	0	0000	01001	Place MLR order in A
1006	12	0	0000	01014	Increment i
1007	54	0	0000	01001	Store new address in MLR order
1010	10	0	0000	01004	Place MOV order in A
1011	12	0	0000	01015	Increment i
1012	50	0	0000	01004	Store new MOV order
1013	40	0	0000	01000	Return to beginning
1014	+00	0	0000	00001	Constant for modifying x_i
1015	+00	0	0001	00000	Constant for modifying Z_i

This program could not actually be used for the computation of Z_i since there is no provision made for terminating the program. Several instructions will be introduced later which can be used to transfer out of a loop after the loop program has been performed a given number of times.

TRANSFER ON NEGATIVE ACCUMULATOR, 46, TRNγα, 16 μsec.

If A_{sn} is a one, the next instruction is taken from α. If A_{sn} is a zero, the sequence is continued in order. Only the sign bit of the accumulator is considered.

TRANSFER ON POSITIVE ACCUMULATOR, 44, TRPγα, 16 μsec.

If A_{sn} is a zero, the next instruction is taken from α. If A_{sn} is a one, the sequence is continued in order.

TRANSFER ON ZERO ACCUMULATOR, 45, TRZγα, 16 μsec.

If $C(A)_{1-36}$ is zero, the next instruction is taken from α. If $C(A)_{1-36}$ is non-zero, the sequence is continued in order.

HALT, 00, HLT, 16 μsec.

The computer operations are halted after completion of any input or output operations currently in progress. If the START AT PC switch is actuated after the computer is halted, the next instruction is taken in sequence.

COMPARE, 47, TRCγα, 22 μsec.

C(A) and C(α) are compared algebraically. If C(A) < C(α) the computer continues in sequence. If C(A) > C(α), the next instruction is skipped. If C(A) = C(α) the next two instructions are skipped. C(A) and C(α) remain unchanged. C(Q) is replaced by C(A). C(B) will contain α. In the execution of this order, minus zero is considered equal to plus zero.

SENSE, 05, SENγβα, 16 μsec.

If the flip-flop specified by β is set, the next instruction is taken from α. If it is reset, the sequence is continued in order.

SENSE AND SET, 06, SNSγβα, 16 μsec.

If the flip-flop specified by β is originally reset, it is set by this instruction and the next instruction is taken from α. If the flip-flop is originally set, the sequence is continued in order.

SENSE AND RESET, 07, SNRγβα, 16 μsec.

If the flip-flop specified by β is originally set, it is reset by this instruction and the next instruction is taken from α. If the flip-flop is originally reset, the sequence is continued in order.

Table III shows the addresses of the various flip-flops in the computer and indicates how they can be affected by the console or the program. For example, if the overflow alarm is set, the instruction

07 0 0100 02511

will reset it and transfer control to 2511.

TABLE III

MOBIDIC SENSE FLIP FLOPS

Octal Address	Designation	Function	Individual Console Switch	Can be Cleared By Program	Can be Set By Program
0001-0077*	IOD	In-Out Device Busy Signals	No	No	No
0100	OA	Overflow Alarm	No†	Yes	Yes
0101	ROPI	Real Time Output Program Interrupt	No	Yes	Yes
0102	ISN	Interpret Sign	Yes	Yes	Yes
0103	NHC	Ignore In-Out Converter Error	Yes	Yes	Yes
0104	RPE	Real Time Parity Error	No	Yes	No
0105	ROBB	Real Time Output Busy Bit	No	No	No
0106	ROR_{38}	Real Time Output Reg. Bit No. 38	No	Yes	Yes
0110-0117	SFF1-SFF8	General Sense Flip Flops	Yes	Yes	Yes
0120-0127	SFF9-SFF16	General Sense Flip Flops	Yes	Yes	Yes
0130	IOA^1	In Out Alarm Converter 1	No†	Yes	No
0131	IOA^2	In Out Alarm Converter 2	No†	Yes	No
0135	TPE	Tape Erase	Yes	Yes	Yes
--	TRA	Trapping Mode	Yes	Yes**	Yes**

*These addresses correspond to the address of the device being sensed. They are levels, not flip-flops.

**Done only by β bits of TRU order or when an order is trapped.

†Can be reset by CLEAR ERROR button.

All flip-flops shown in Table III, except TRA, are addressable only by SEN, SNR, and SNS instructions, except that RPE and IOA^1 and IOA^2 cannot be set (SNS); ROBB and IOD can only be sensed (SEN) and the state of TPE cannot be determined (SEN), but it can be set (SNS) and reset (SNR). In each case, the flip-flop's address appears in the beta-part of the instruction.

E. Indexing Instructions

It was mentioned in Section II that a set of index registers is provided in the MOBIDIC system. The contents of these index registers can be added to the programmed addresses before execution of the instructions in order to allow for more flexible programming. The indexing operation itself, that is, the addition of the contents of the index register to a memory address, proceeds generally without loss of time in the main program. Address modification is used not only to operate on different variables with the same set of instructions, but also to permit the programming of routines independently of the memory locations which will be used to store these routines. The programmer is then unconcerned with the actual storage assignment until all routines are completely programmed.

When an order is "indexed," the address α is internally increased by the contents of the specified index register before the order is executed. The contents of the memory position containing the order are unchanged.

Since the index registers are used for address modification, instructions must be available to modify the contents of these index registers and to specify memory addresses. A set of indexing instructions is therefore provided to increment the contents of the index registers by some specified amount, or to specify addresses by other means. Such instructions are described below. If there are n index registers and $\gamma = n$, then $\gamma + 1 = 1$; if $\gamma = 0$ or $\gamma > n$, then no data can be transferred into the specified (nonexistent) index register; if data is called for from a nonexistent index register by any instruction, the number zero is supplied.

LOAD INDEX, 53 LDX$\gamma\beta\alpha$, 16 μsec.

This instruction replaces $C(I^{\gamma})$ by β and $C(I^{\gamma+1})$ by α. The LOAD INDEX instruction always modifies the contents of two index registers. If it is desired to load only one index register from memory, the LOAD instruction should be used. For example,

```
51    0    0002    07512
```

could be used to load index register 2 with $C(7512)_{1-12}$. The LOAD INDEX instruction cannot be indexed. If a nonexistent index register is addressed, the instruction is interpreted as a vacuous (skip) order.

TRANSFER ON INDEX, 43, TRX$\gamma\beta\alpha$, 22 μsec.

If $C(I^{\gamma+1}) = 0$, the instructions are continued in sequence. If $C(I^{\gamma+1}) \neq 0$, $C(I^{\gamma+1})$ is replaced by $C(I^{\gamma+1}) - 1$. If $C(I^{\gamma+1})$ is now equal to zero the sequence is continued in order; otherwise $C(I^{\gamma})$ is replaced by $C(I^{\gamma}) + \beta$, and the next instruction is taken from α. In any case C(B) is replaced by α. This instruction cannot be indexed. If a nonexistent index register is addressed, the instruction is interpreted as a vacuous order; C(B), however, is replaced by α.

ADD BETA, 24, ADB$\gamma\beta\alpha$, 26 μsec.

First β is added to $C(\alpha)_{1-12}$. The sum is then placed in A, $C(\alpha)$, and I^{γ}. C(Q) is replaced by the original contents of A. The contents of B depend on both C(A) and $C(\alpha)$, as shown in Table V at the end of the manual. Overflow is ignored. The ADD BETA instruction cannot be indexed.

SUBTRACT BETA, 25, SBB$\gamma\beta\alpha$, 26 μsec.

β is subtracted from $C(\alpha)_{1-12}$. The difference is placed in A, $C(\alpha)$, and I^{γ}. C(Q) is replaced by the original contents of A. C(B) depends on C(A) and $C(\alpha)$ as before. Overflow is again ignored. The subtract beta instruction cannot be indexed.

LOAD PCS AND TRANSFER, 41, TRL$\gamma\beta\alpha$, 16 μsec.

The location of this instruction incremented by one is stored in PCS; β is then loaded into I^{γ} and the next instruction is taken from α. This instruction cannot be indexed.

TRANSFER TO PCS, 42, TRS, 16 μsec.

The next instruction is taken from the location specified by PCS.

REPEAT, 01, RPT$\gamma\beta\alpha$, 16 μsec.

The instruction following the REPEAT order is executed $\alpha + C(I^{\gamma}) + 1$ times provided it is not an input or output order. After each execution of that instruction, its α part is increased by β of the REPEAT instruction. After completion of the repeated order, $I^3 = 0$ and $I^4 = \beta$. The REPEAT instruction and the instruction being repeated are left unchanged in memory. A sense or transfer instruction following a REPEAT order will transfer in the proper manner only when it has been executed $\alpha + C(I^{\gamma}) + 1$ times.

Two MOBIDIC instructions exhibit a special mode of operation when preceded by the REPEAT order. These are MOVE (MOV), and COMPARE (TRC).

The REPEAT-MOVE combination allows the block transfer of an arbitrary number of words from one part of memory to another. When the MOVE instruction follows a REPEAT, the α address of the MOVE order is incremented as described under the REPEAT order, and the $\gamma\beta$ address is indexed by $C(I^2)$.

As an example, the following program will move a block of one hundred consecutive words starting at location 250 to consecutive locations starting at 1200:

```
00071    51     0    0002    00074
00072    01     0    0001    00143      Note: 143(8) = 99(10)
00073    52     0    1200    00250
00074   +00     0    0000    00001
```

In this program memory location 74 contains a constant used to load I^2. Since the high-order six bits of C(74) represent the operation code for the HALT instruction, the machine will stop after completing the block move. It should be noticed that, by substituting different values for the beta part of C(72) and for the alpha part of C(74), a large variety of block moves can be made. Thus, blocks of words can be increased or reduced in size, and the arrangement of the words within a block of data can be substantially altered.

The REPEAT operation can be used in conjunction with a COMPARE (TRC) operation to allow for comparisons between the contents of the accumulator and a specified sequence of memory locations. Such comparisons are useful in table look-up operations and in operations which make it necessary to rearrange the order of input or output data.

Consider the following sequence of orders:

Location	Order
y - 1	RPT $\gamma_1\beta_1\alpha_1$
y	TRC γ_2 - α_2
y + 1	-
y + 2	-
y + 3	-

Since both the REPEAT and the COMPARE orders can be indexed, the number of executions of the COMPARE order is given by $R = \alpha_1 + C(I^{\gamma 1}) + 1$, while the effective address of the COMPARE order is $x = \alpha_2 + C(I^{\gamma 2}) + n\beta$, where n = 0, 1, 2, etc.

The operation of the REPEAT-COMPARE sequence will now be explained in detail. The contents of the accumulator are first compared with the contents of x. If C(A) < C(x), x is stored in the B register and transfer is made to location y + 1, that is, to the address immediately following the TRC order. In a table look-up operation, this case corresponds to one where the argument used to search the table is smaller than any of the functional values stored in memory in increasing order of magnitude.

If C(A) = C(x), x is again stored in the B register, and transfer is made to location y + 3. This case corresponds to one where the argument stored in the accumulator is equal to the functional value stored in location x.

If C(A) > C(x) the argument is larger than the functional value stored in x, and in general a further comparison with a larger functional value should take place. For this reason, a further test is made to determine if R (the number of repetitions of the COMPARE order) is equal to zero. If R > 0, R is decreased by 1, and the address of the next memory location, x, to be compared is increased by β_1 of the REPEAT order; the whole program is then started over by a new comparison between the contents of the accumulator and the contents of the new x. If, on the other hand, R = 0, that is, if all allowable repetitions have already been performed, a transfer is made to location y + 2. This case is the one where the argument has already been tested against all available functional values.

At the end of the complete sequence, the contents of the accumulator will appear in the Q-register, while C(A) and C(x) are unchanged. The actual number of comparisons executed is given by $\alpha_1 + C(I^{\gamma 1}) + 1 - C(I^3)$.

The complete RPT-TRC sequence is shown in flow-chart form in Figure 9. The round boxes indicate comparisons. The TRC order is assumed to be stored in location y as before. The operations coded at locations y + 1, y + 3, and y + 2 can be unconditional transfer orders which transfer control to different locations depending on whether the argument is smaller than, equal to, or larger than any of the given functional values.

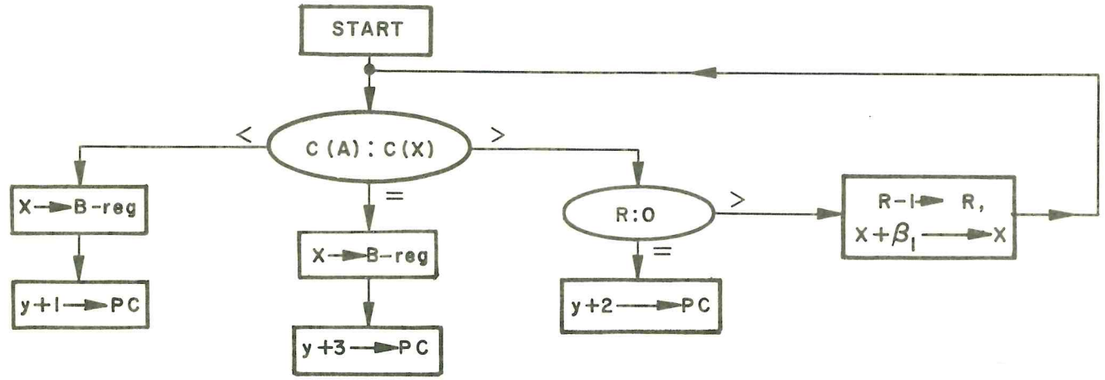


Figure 9. Operation of the REPEAT-COMPARE Sequence

F. Input-Output Instructions

The input-output system of MOBIDIC consists of one or more in-out converters and a set of peripheral equipment. Each converter contains storage units and controls to operate any of the peripheral devices. As many devices can operate simultaneously as there are available converters.

When an in-out order is decoded the first free converter is selected and the instruction is stored in that converter. The computer then proceeds to the next instruction in sequence. In the event that all converters are in use, or that the input-output device requested is occupied, the sequence is interrupted until the device is free and a converter is available; when these conditions are obtained the instruction is processed.

Before giving a detailed description of the in-out orders, a few comments must be made concerning the magnetic and paper tape formats. Characters are recorded on magnetic tape in eight parallel channels. Six of the channels correspond to the alphanumeric character code, one is used to store a parity bit for each character, and the last is used for control purposes. The density of recording is 280 bits-per-linear-inch, and tape speed is 150 inches-per-second. If the sign of a word is to be read or recorded together with the remaining

characters of that word, the Interpret Sign Flip-Flop (ISN) must be set by a SENSE AND SET order prior to executing the actual in-out instruction. The ISN Flip-Flop will be reset as part of the transfer of the in-out instruction to the converter. In the interpret sign mode, the sign is then converted to the correct eight-bit pattern and recorded on tape as an individual character.

The paper tape format is similar to the magnetic tape format, in that information is punched in eight parallel channels: six information channels, one parity channel, and one control channel. However, the code assignment for the individual characters is somewhat different for paper tape at either 60, 120, or 270 characters-per-second, depending on the paper tape reader; punching is done at the rate of 60 characters-per-second.

When reading or writing, the Non-Halt On Converter Error flip-flop (NHC) can be set, if a parity error is to be ignored. Thus, if the programmer wishes to read 5 words with sign from magnetic tape No. 1 and ignore parity errors, the following program can be used if the five words are to be stored starting in location 1000:

Address					
100	06	0	0102	00101	Sense and set ISN, transfer to 101
101	06	0	0103	00102	Sense and set NHP, transfer to 102
102	70	005	01	01000	Read 5 words from tape No. 1.

The input-output instructions will now be described in more detail.

WRITE ALPHANUMERIC, 74, WANkjα, 16 μsec + 8 μsec per word + tape time

The WRITE ALPHANUMERIC order is used to write k words on output device j. The location of the first word is specified by the address α. If the addressable Interpret Sign flip-flop (ISN) has not been set, the converter will ignore the sign, and split the remaining word into six 6-bit characters. The high-order characters of each word are recorded first. Thus, bits 31-36 will be recorded first, followed by bits 25-30, and so on until bits 1-6 are recorded. Since k consists of only 9 bits in the order word, in general there is a maximum of 511 words per block.

If the Interpret Sign flip-flop is set prior to giving the WAN order, a six-bit character defined as the sign of the word is recorded, followed by the six characters of the word. Thus, in this mode, each word is composed of seven characters, the first character being the sign.

If the device j is a magnetic-tape unit, special symbols known as block marks are automatically recorded before the first word and after the k^{th} word. An interblock gap is left between each terminating block mark and the beginning block mark of the next block. When a WAN order is executed, this gap introduces a delay of approximately 6.2 milliseconds before the first word is transmitted from the memory. Approximately 11 milliseconds are required for the tape to pass over a complete interrecord gap. Thereafter, words are recorded at the rate of 24 μsec-per-character (144 μsec or 168 μsec per word). During the record operation, internal computations can take place; an interruption of 8 μsec occurs only when a

converter requires access to the memory. This interruption is handled automatically, and is not under program control.

WRITE OCTAL, 76, WOKkjα, 16 μsec. + 8 μsec. per word + tape time

The WRITE OCTAL order cannot be used for magnetic tape. In the WRITE OCTAL mode, the sign is always interpreted and the 36-bit word is split into 12 octal digits preceded by a sign digit. Each octal digit, consisting of three bits, is converted to its equivalent six-bit form and sent to the output device. This is done because the output devices respond only to a six-bit code. Thirteen 6-bit characters are recorded per word. In all other respects the WOK order is similar to the WAN order previously described.

READ OCTAL, 72, ROKkjα, 16 μsec. + 8 μsec. per word + tape time

The READ OCTAL order reads k words from device j and places them in memory starting at the address specified by α. This order cannot be used for magnetic tape. Its primary purpose is to provide a means for converting information on punched paper tapes directly into machine code. This is required since information is not recorded on paper tapes in pure binary form. The six-bit characters are converted to equivalent 3-bit octal digits. A machine word is assembled from 13 octal digits; the first is treated as the sign, and the remaining twelve comprise the 36-bit word. However, the non-octal numbers and English text may be used between octal words but will be ignored by the computer.

Because it is impractical always to specify the exact number of words to be read, a special character defined as "stop code" is provided. When this code is read from paper tape by the photoelectric tape reader, the reading is terminated, even though less than k words have been read. The stop code can be punched on tape either by the Flexowriter or under program control.

Other special characters are provided to control the Flexowriter action when using a write order. Since the Flexowriter has both an upper and a lower case, symbols are provided to shift both up and down. Other symbols control the space, tabulator, carriage return, and back-space actions. All these special controls are available for both upper and lower case. All the characters, including alphanumeric characters and special arithmetic symbols are provided in either upper or lower case, but only letters are provided in both.

READ ALPHANUMERIC, 70, RANkjα, 16 μsec. + 8 μsec. per word + tape time

If j specifies a magnetic-tape unit, then the READ ALPHANUMERIC order reads $k \pmod{2^8}$ words or blocks into consecutive memory locations starting at address α. If the ninth bit (the most significant bit) of k is a one, then the remaining eight bits determine the number of blocks to be read; if this bit is a zero, then the other eight bits determine the number of words to be read. The RAN order can be used with the Interpret Sign flip-flop in a set or reset state, as explained previously.

When a magnetic-tape unit is selected by this order, the converter first searches for a beginning block mark before reading words into the machine. For example, if the order 70 005 01 00300 is to be executed, the converter will search for the next beginning block mark on magnetic tape unit one, read five words into memory starting at address 300, and if the number of words in the block is greater than five, search for the terminating block mark and stop.

If the device addressed by the RAN order is a paper-tape reader and the Interpret Sign flip-flop (ISN) is set, the RAN order will assemble incoming characters consisting of six 6-bit characters preceded by a sign character. The reading will be terminated either by a stop code on paper tape or after k(mod 2^9) words have been read, depending on which occurs first.

Consider, for example, the following orders:

```
3000   06   0     0102   03001
3001   70   777     36   00100
```

The first order sets the ISN flip-flop (address 0102); thereafter control is transferred to location 3001. The next order reads up to 511 words and stores them starting at location 100. If a stop code is encountered the reading operation will cease; otherwise all 511 words will be read.

If the Interpret Sign flip-flop is not set and the paper-tape reader is addressed, two modes of paper-tape reading are possible, depending on whether the most significant bit of k is a one or a zero. If this bit is a one, six characters will be grouped to form a word to be stored in memory. When the stop code is read, the reading will cease and the stop-code combination will appear as a character in the last word. Zeros will be supplied in the low-order positions to fill the last word read into memory.

If the highest-order bit of the k-part in the instruction is a zero, only one character is read and placed in memory location a as a full machine word. This character will occupy the low-order six bits; the remainder of the word will be filled with zeros. A stop-code will be read into memory the same as any other character.

SKIP, 66, SKPkj, 16 μsec. + tape time

This order applies to magnetic tape only. Tape unit j will skip k(mod 2^9) blocks forward. The a bits of the order are not used.

BACKSPACE, 67, BSPkj, 16 μsec. + tape time

This instruction is identical with SKP except that skipping takes place in the reverse direction.

READ REVERSE, 71, RRVkja, 16 μsec. + 8 μsec. per word + tape time

This instruction is identical with the RAN order except that reading takes place in reverse direction. The sign bit still appears in its proper position.

REWIND, 77, RWDj, 16 μsec. + tape time

This instruction rewinds magnetic-tape unit j. After the rewind starts, the converter is then disconnected and the rewind operation proceeds independently. The converter is then available for further in-out instructions.

RE-WRITE ALPHANUMERIC, 75, WWAkja, 16 μsec. + 8 μsec. per word + tape time

When this order is given, tape unit j starts in the read mode and searches for the next beginning block mark. When this block mark is detected, the tape unit switches to the record mode and writes k(mod2^9) words beginning at memory location a followed by a terminating block mark. If a beginning block mark is not found, the whole tape will be searched.

IV PROGRAMMING TECHNIQUES

A number of techniques which may be useful in operating and programming the computer are presented in this section. The trapping mode is first described followed by a description of the control console. Methods are suggested for starting the calculations and for the check-out of problems. Some useful program subroutines are then given to illustrate the use of the orders introduced in the previous section.

A. The Trapping Mode

The trapping mode is used mainly in diagnostic procedures. It can be controlled either by the program or by the operator. The following discussion applies only to program control.

If the trapping mode flip-flop (TRA) is set, and the machine is given any of the instructions in the list below, it will not perform the instruction; it will instead take the next instruction from location 0, the contents of the Program Counter will be sent to the B register, and the trapping mode flip-flop will be reset.

TRL	TRN	SEN
TRS	TRP	SNS
TRX	TRZ	SNR

The trapping mode flip-flop (TRA) is not addressable. It may be set by using the unconditional transfer (TRU) order, with the 16th and 17th bit positions set in accordance with the scheme shown in Table IV.

TABLE IV TRAPPING MODE CONTROL

β_{17} of TRU Order	β_{16} of TRU Order	Numerical Value of β	TRA Flip-Flop (Before)	Trapping Action	Effect on PC	TRA Flip-Flop (After)	Comments or Summary
0 or 1	0	0 or 2	0	No	$\alpha + I^{\gamma} \rightarrow PC$ $PC \rightarrow B$	0	No special action
0 or 1	0	0 or 2	1	Yes	$0 \rightarrow PC$	0	Order is trapped
0	1	1	0 or 1	No	$\alpha + I^{\gamma} \rightarrow PC$	1	Sets TRA Flip-Flop if not already set
1	1	3	0 or 1	No	$\alpha + I^{\gamma} \rightarrow PC$	0	Resets TRA Flip-Flop if not already reset

B. Console Operations

A slightly simplified diagram of the console is given in Figures 9 and 10. It may be noticed that the control elements on the console consist of a set of switches and a set of display lights. The display lights exhibit the state of the flip-flops which store the binary digits within each word.

The contents of the following storage registers can be displayed in the bus indicator register:

A, B, and Q Registers
Program Counter Store
Memory Output Registers (1 - 7)
Index Registers (1 - 4)
Converter Instruction Registers (1 - 4)

The register selection is provided by the register selector switches, located below the bus indicator register. Each one of the 15 switches controls one of the registers listed above. The seven memory output registers can be used to monitor any word in memory after first transferring that word from memory to the appropriate memory output register. This transfer can be achieved manually by means of a read-out switch provided as part of the initiating control switches.

The instruction word register is used to display the last instruction performed by the computer. This register displays the contents of the instruction register, the G and X registers, and the address register, as explained in Section II. A third set of indicators is used to display the contents of the program counter, that is, the address of the next instruction to be performed by the computer.

The program counter displays the location from which the next order will be taken.

The mode control switches (Single Pulse, Run, One Instruction) are used to fix the mode of operation. The computer can be made to operate continuously under program control (Run), or it can be made to stop after each instruction is performed during program checking (One Instruction), or it can be made to stop after each pulse time for maintenance purposes (Single Pulse).

The initiating control switches (Read Out, Start at ASR, Manual Instruction, Read In, Start at PC, Program Read In) are used to start computer operations, either at the location specified by the contents of the program counter (Start at PC), or at the location specified by the contents of the address switch register (Start at ASR). The address switch register consisting of 15 switches provided on the console can be set manually to store any computer address. A set of 15 neon indicators located immediately above the address switches is used to display the contents of the address switch register. The initiating controls are also used to read-in programs into the computer from punched paper tape (Program Read In). This mode is often used when the computer must first be loaded with a set of programs.

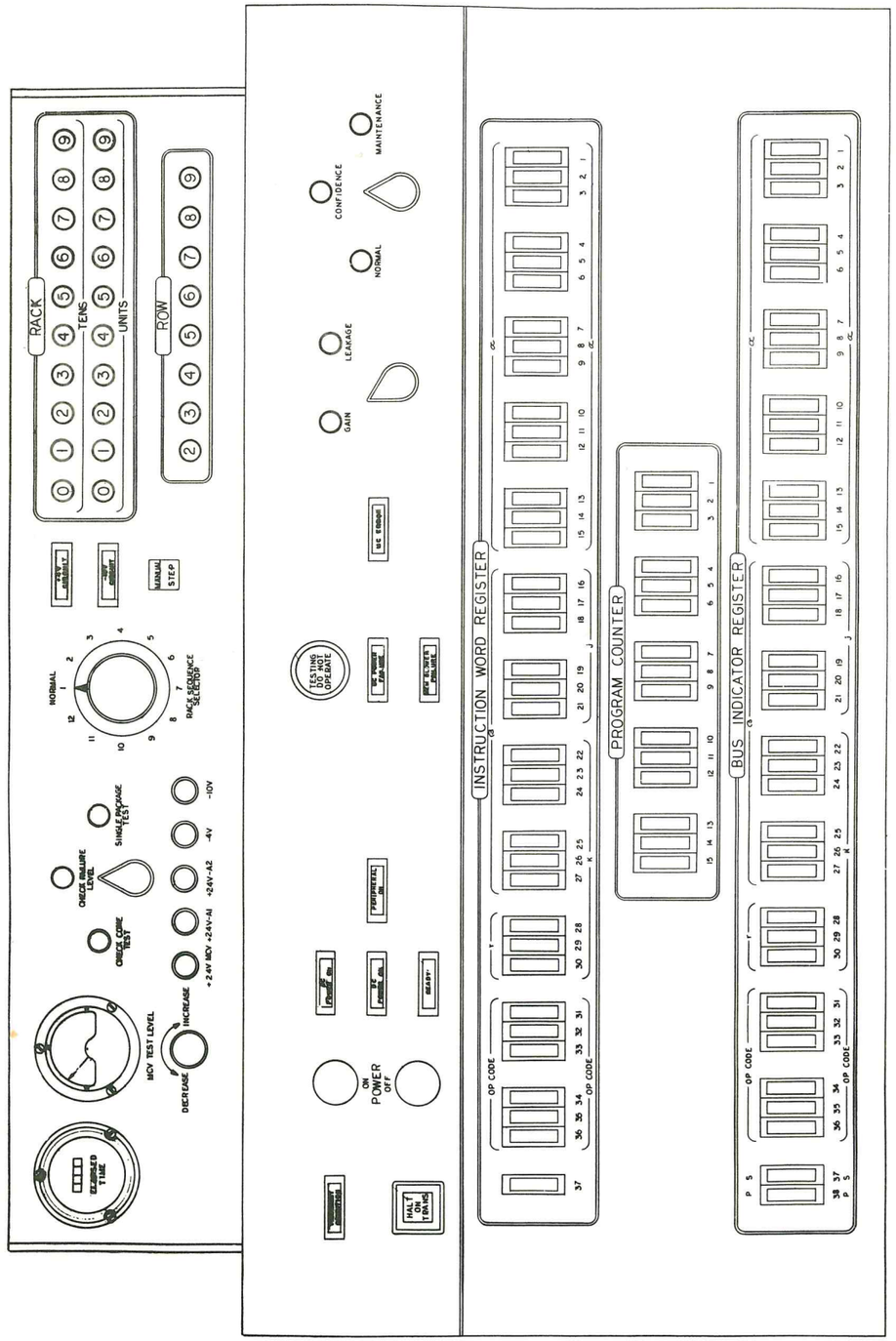


Figure 10. Organization of the Upper Half of the Control Console: The Display Panel

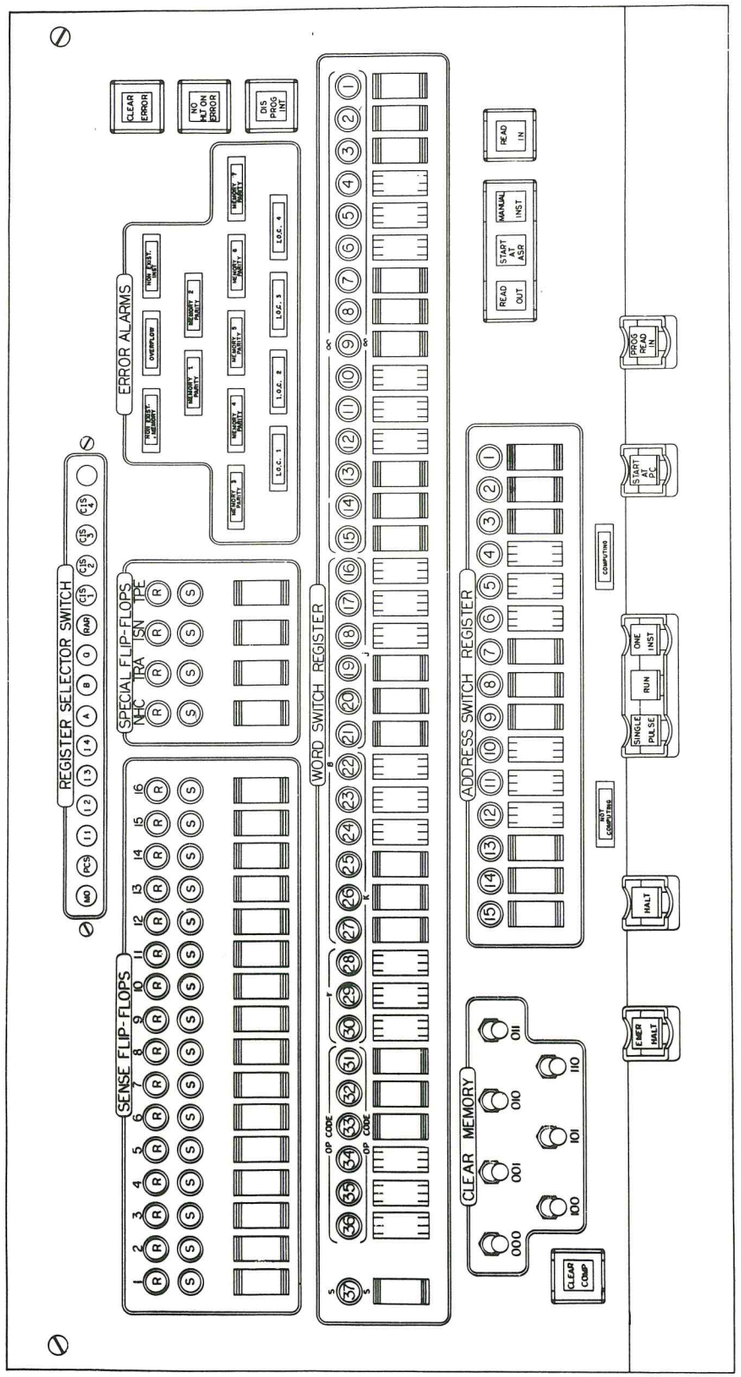


Figure 11. Organization of the Lower Half of the Control Console: The Control Panel and Step Panel

The initiating controls also permit manual read-in (Read In) and read-out (Read Out) operations. To this effect, the word switch register consisting of 37 switches and associated display indicators is provided on the console. A word can be set-up in the switches and can then be read into the register or memory location specified by the address switch register; alternatively, an instruction specified in the word switch register can be executed by means of the manual initiating controls (Manual Instruction). This register (WSR) is individually addressable. To read-out a word stored in memory, the memory address is set in the address switch register; the word is then transferred from memory to the memory output register in order to be displayed in the bus indicators.

A halt switch and an emergency halt switch are provided on the console; these switches can be used to stop the computer manually. The halt switch stops the machine after completion of the current instruction except that all input-output instructions currently being executed are finished. The emergency halt stops the machine without waiting for completion of the current instruction. In the latter case, all input-output devices are stopped and the associated converters cleared.

The clear memory function provides a method for resetting to zero the contents of all memory locations of a given memory unit, while the clear computer switch clears all registers (but not the memory) except the halt flip-flop and the flip-flop of the timing function generator. Each memory unit is provided with a clear memory switch.

A set of three-position sense and control switches are provided which can be set or reset either manually or under program control. In the neutral position, these flip-flops operate under program control only, and can be interrogated by a SENSE order, as previously explained. In the set or reset positions, the flip-flops are manually controlled and cannot be altered by the program. Sixteen sense flip-flops are provided and four special control flip-flops designated respectively by NHC (no halt on converter error), ISN (interpret sign), TPE (tape erase), and TRA (trapping mode).

The NHC flip-flop is used to prevent the halting of the machine if a converter error is detected, while the ISN flip-flop controls the treatment of the sign of a number during input-output operations. The TPE flip-flop permits tape erasure. It is used in conjunction with a write order and it erases a length of tape equal to the length of k words plus its associated interblock gap.

The TRA Switch is also shown on the control console. When this switch is reset manually, all transfers to the trapping mode are inhibited.

A set of error and alarm display lights is provided on the console to detect various program and machine failures. Thus, for example, an alarm is set if a nonexistent instruction is used, or if the program counter contains a nonexistent memory location. Overflow resulting from an arithmetic operation can also be detected by the overflow alarm flip-flop, which is under program control. A separate error alarm is provided for each memory unit to detect parity check errors, and for each in-out converter.

Under normal circumstances the computer will halt if an error is detected by any of the error alarms. However, a No Halt on Error switch provided on the console can inhibit the halt action in such cases, if this is desired. This No Halt switch must be set manually. If an error is detected and the machine stops, the error alarm is reset by a Clear Error switch, which is located below the error alarms on the console. The Clear Error switch operates as a common reset control for all error alarms.

A Disable Program Interrupt switch is provided to inhibit the program interrupt feature available with the real time input system.

A number of power control indicators are provided to indicate whether the main power is turned-on; indicators will also show whether the peripheral equipment and the memory units are ready to operate. A Not Computing indicator, connected to the halt flip-flop, is turned-on if the machine is stopped but ready to compute, while a Computing indicator is turned-on when the machine is operating.

C. Program Preparation and Check-Out

Programs are normally prepared on punched paper tape. However, if off-line equipment is available, magnetic tapes or punched cards can also be used. The program read-in mode is used to transfer a program into memory. To initiate the program read-in operation, two instruction words must be provided on a special paper tape. The first word on the tape is a READ instruction which specifies the address of the input device containing the program, the number of words to be read into the computer, and the memory storage location for the program. The second word contains the address to which control is transferred after completion of the program read-in.

During check-out of a program, the error and alarm flip-flops should be enabled so as to allow detection of nonexistent instructions and memory addresses, and of possible overflow during arithmetic operations. Furthermore, it is convenient to operate the computer in the trapping mode during program check-out. A transfer to memory location zero will then occur for each TRANSFER and SENSE order in the program. The program at memory location zero can provide for type-out of the contents of important memory locations; moreover, the location from which transfer was made to memory location zero must be saved from the B register so that the program can be resumed at the correct location after interruption.

Since trapping mode permits verification of the conditions which control a transfer of control operation, the programmer can follow the programs as they are executed to insure that the various paths traced on the flow diagram are actually executed. Instead of stopping the computer for each TRANSFER and SENSE operation, the contents of certain key registers can be recorded on tape and the program resumed automatically from where the interruption originally occurred. At the end of the complete computation, the program can then be checked-out with the help of the recorded data without tying up the computer.

The sense flip-flops can also be used for diagnostic purposes, since they can cause transfer of control when set, while not affecting the program when reset. When used in this manner they function as "break-points" in the program.

For maintenance and detection of machine failures, the machine can be operated cycle-by-cycle or one pulse at a time. Special programs can also be used to test the various units of the computer in order to narrow down the areas where errors are likely to have originated. Such programs will be described in a separate maintenance manual.

Special program packages are also made use of while preparing a problem for solution. Such programs are frequently designed to be used repeatedly in a given calculation and are of great help to the programmer. Some routines of this nature are described in the next few paragraphs.

D. Elementary Subroutines

A subroutine can be defined as a set of self-contained orders designed to perform a specified task which may be required repeatedly during a given calculation. Such subroutines are often stored in a fixed location in memory through the computations, and control is transferred to the particular location whenever the routine is needed. Some of the routines which are used most frequently for scientific applications include the trigonometric function routines; square root, logarithm, and exponential routines; and routines for complex, high-accuracy (double-length), and floating-point arithmetic. Routines which are used in data-processing applications include the merging, classification, and ordering of data; the extraction of certain characters from a large block of information, and the loading and unloading of the memory locations.

A number of preliminary routines are presented in this section. The methods used represent typical solutions to the problems chosen. It is not suggested, however, that these routines are necessarily to be included in a subroutine library for MOBIDIC.

1. Frequency Count. It is often desirable to obtain a frequency count of a set of words or characters. For example, in linguistic analysis it might be desirable to determine how often each word is used in a given text; alternatively, a frequency count of the individual letters within each word might be desired.

The routine which follows can be used to obtain a frequency count of alphanumeric characters. One hundred blocks of twenty words each are stored on magnetic tape. The six alphanumeric characters which are included in each word are individually tested and a count is kept for each of the 64 possible configurations of 6 bits. This count is kept in memory locations 00 000 to 00 077 (the first 64 memory locations in memory unit 0) in such a way that memory location α is used for configuration α. The program will process one block of words while the next block of words is read into memory. The total time required by the program is a function of the read-in time only; no additional time is required by the internal processing.

MOBIDIC MACHINE
CODING FORM

PROGRAM Frequency Count
ROUTINE _____
PROJECT/W.O. _____

LOCATION	±	OP	K γ	J β	α	COMMENTS
0						
100 1		70	4	01 | 20	00124	BLOCK I
100 2		53	4	00 | 00	00144	100_{10}->I^1, 0->I^4
100 3		10	0	00 | 00	70000	
100 4		01	0	00 | 01	00077	} clear counters
100 5		50	0	00 | 00	00000	
100 6		70	4	01 | 20	00100	BLOCK II
100 7		07	0	01 | 10	01010	0->SFF1
101 0		24	0	00 | 24	01012	modify process
101 1		51	0	00 | 03	00151	20_{10}->I^3
101 2		51	3	00 | 11	00077	word->Q
101 3		51	0	00 | 04	00150	6->I^4
101 4		10	0	00 | 00	70000	clear acc.
101 5		31	0	00 | 00	00006	
101 6		54	0	00 | 00	01020	process
101 7		10	0	00 | 00	70011	6 characters
102 0		24		00 | 01		
102 1		43	3	00 | 00	01014	character test
102 2		43	2	00 | 00	01012	word test
102 3		43	4	00 | 00	01025	block test
102 4				|		Exit
102 5		05	0	01 | 10	01006	sense to determine next read
102 6		70	4	01 | 20	00124	read BLOCK I
102 7		25	0	00 | 24	01012	modify process
103 0		06	0	01 | 10	01011	1-> SFF1
1				|		
2				|		LOC CONTENTS
3				|		0-77 Counters
4				|		100-123 BLOCK II
5				|		124-147 BLOCK I
6				|		150 68
7				|		151 248

A sense flip-flop is used to indicate whether the computer is ready to treat block 1 or block 2. Index register 3 (I^3) is used to count the words in each block, I^4 counts the characters in each word, and I^1 counts the blocks themselves. The inner loop which processes the characters (locations 1014 - 1021) is reduced to a minimum, since each 6-bit character is also treated as the address of the counter for that character.

2. Merging. The operation which consists in taking several sets of ordered numbers and forming from these a single ordered set of numbers is called merging. Merging is frequently used in business and other data-processing applications for file maintenance, and during sorting operations.

The program presented here takes two sets of positive numbers arranged in ascending order and merges them into a single set of numbers in ascending order. The location of the first word of set 1 is taken to be an unspecified memory location x, while location y will store the first word of the second set. The merged list will be stored starting at memory location S.

Index register number 4 (I^4) is used as a counter to step the storage addresses of the merged sequence, so that successive numbers are stored in successive locations in memory. I^2 is similarly used to step from one storage address to the next for the first set of unmerged numbers, and I^3 steps from address to address for the second set of unmerged numbers. The last number in each set is a sentinel identified by a negative sign. The program tests each number to find if it is negative. When the first negative number is found, one set of numbers is exhausted; the remaining numbers from the other set are then taken in order and added to the merged list.

In this routine it is assumed that the total number of words to be merged does not exceed the number of locations available in memory.

3. Classification. The operation which consists in breaking down a single set of items into several classes according to some specified criterion is called classification. For example, the familiar punched-card sorter arranges a deck of cards into ten pockets according as a particular card column contains each one of the ten decimal digits.

Classification is one of the basic business data-processing operations. It is performed when a certain class of items in stock must be segregated from the other items in an inventory control problem, or when a given class of customers must be processed separately in an accounts-receivable operation. Classification can be performed both with tabulating equipment and with large-scale computing machinery.

The routine presented here classifies one hundred integers into ten classes according to the value of the lowest order digit. Since all integers might conceivably end in the same digit, one hundred storage locations are reserved for each class of integers. Thus, all integers which end in zero are stored starting at location 2001, those ending in 1 are stored starting in 2146, those ending in 2 are stored starting in 2313, and so on. One extra memory location is reserved for each class of integers to store the number of integers in each class.

MOBIDIC MACHINE
CODING FORM

PROGRAM Merge Routine

LOCATION	±	OP	K γ	K β	J	α	COMMENTS
0		LDX	2		(X)	(Y)	L(X)→I^2; L(Y)→ I^3
1		LOD	0	0 0	04	00036	L(S)→I^4
2		CLA	2	0 0	00	00000	X
3		TRN	0			00030	Test for last X
4		CLA	3			00000	Y
5		TRN	0			00022	Test for last Y
6		TRC	2			00000	Y:X
7		TRU	0			00016	<
10		TRU	0			00011	>
11		CLA	2			00000	=
12		STR	4			00000	X→S
13		ADB	0	00	01	70002	X + 1→X
14		ADB	0	00	01	70004	S + 1→ S
15		TRU				00002	
16		STR	4			00000	Y→S
17		ADB	0	00	01	70003	Y + 1→Y
20		ADB	0	00	01	70004	S + 1→S
21		TRU	0			00004	
22		CLA	2			00000	X
23		TRN	0			EXIT	Test for last X
24		STR	4			00000	X→S
25		ADB	0	00	01	70002	X + 1→X
26		ADB	0	00	01	70004	S + 1→S
27		TRU	0	00	00	00022	
30		CLA	3	00	00	00000	Y
31		TRN	0	00	00	EXIT	Test for last Y
32		STR	4			00000	Y→S
33		ADB	0	00	01	70003	Y + 1→ Y
34		ADB	0	00	01	70004	S + 1→S
35		TRU	0			00030	
36						L(S)	
7							

MOBIDIC MACHINE
CODING FORM

PROGRAM Classification

LOCATION	±	OP	K γ	K β	J	α	COMMENTS
01000		10	0	00	00	70000	
01001		01	0	01	45	00009	} clear counters
01002		50	0	00	00	02000	
01003		53	2	00	00	00144	100_{10}→I^3, 0→I^2
01004		10	0	00	00	70000	0→ACC
01005		51	2	00	11	01513	N→Q
01006		23	0			01512	N (mod 10) = rem
01007		50	0			70001	rem→I^1
01010		10	1	00	00	01500	C_r
01011		54	0			01012	
01012		24	0	00	01		C(C_r + 1→C(C_r)→A
01013		12	1			01500	L(C_r) + C(C_r)→A
01014		54	0			01016	
01015		10	2			01513	N
01016		50	0				N→L(C(C_r) + L(C_r))
7		43	2	00	01	01004	
0							EXIT
1							
2							LOCATION CONTENTS
3							1500 2000 }
4							1501 2145 } add.
5							1502 2312 } of
6							= } col.
7							1511 2615
0							
1							1512 12_8
2							
3							1513
4							= } 100_{10} numbers
5							1657
6							
7							

Thus, locations 2000, 2145, 2312, ..., 2615 will serve as counters for integers ending in 0, 1, 2, ..., 9 respectively.

To determine the class to which a given integer belongs, each integer is divided by 10 (12 in octal notation). The remainder after the division appears in the accumulator and specifies the class of the integer. The sum of the contents of each counter plus the address of that counter determines the effective storage address for the next integer to be stored in each class.

4. Justification. Justification of a word consists in shifting the word to the right until the first non-zero digit appears in the lowest order position of the word. All zeros appearing at the low-order end of the word are eliminated. For example, a word containing the characters AB0500 will contain 00AB05 after justification. This operation is used mostly for editing purposes when output data must be recorded in such a way that each word ends at a fixed right-hand margin.

Index register number 2 (I^2) is used as a counter. Initially it stores the number of words to be justified; when I^2 is reduced to zero the process is terminated. Index register number 1 is used as a counter, starting at zero. The address of the first word (1000) is added to the contents of I^1 to determine the storage locations of the consecutive words before and after justification. Index register number 3 is first cleared, and is then used as a counter to store the number of shifts required for justification. The number of shifts required to justify the various words are stored in consecutive locations starting at 2000. I^4 stores the number 36, the maximum number of shifts permitted for each word. If 36 shifts are required, the word originally stored is equal to zero.

5. Floating-Point Addition. It is desired to add two numbers, each one being defined by a 36-bit mantissa and an exponent (characteristic). The exponent can range in magnitude from $2^{36} - 1$ to $-(2^{36} - 1)$ and is assumed to be stored in the memory location immediately following the location of the corresponding mantissa. A zero exponent indicates that the mantissa is a normalized fraction with zero significant digits to the left of the binary point (the binary point lies between bit 36 and bit 37). When the mantissa is shifted n places to the right, its exponent is increased by n; conversely, when the mantissa is shifted to the left, the exponent is correspondingly decreased.

Before two numbers which carry different exponents can be added, the number with the smaller exponent is shifted to the right by a number of places equal to the difference of the two exponents. A zero mantissa is supplied with the smallest possible exponent $(-2^{36} + 1)$ so as to insure that it is always considered as the smaller of the two numbers to be added. Since the shift circuits in MOBIDIC operate modulo 128, a shift of more than 127 places will in general result in an incorrect shift. If, therefore, the difference of the two exponents is greater than 127, the number with the larger exponent is taken as the sum of the two numbers.

MOBIDIC MACHINE CODING FORM

PROGRAM Justification

LOCATION	±	OP	K γ	K β	J	α	COMMENTS
0							
1		53	1	00	00	N	$N \rightarrow I^2$, $0 \rightarrow I^1$
2		53	3	00	00	00044	$36_{10} \rightarrow I^4$, $0 \rightarrow I^3$
3		10	1			01000	$N_i \rightarrow A$
4		02	0	00	00	00020	Low Order bit of N_i in A
5		45	0			00012	If Zero, Justify
6		10	0			70003	If One, $I^3 \rightarrow A$
7		50	1			02000	No. of Shifts → Storage
10		43	1	00	01	00002	Get Next Word
11		40	0	00	00	00017	Exit Line
12		10	1			01000	$N_i \rightarrow A$
13		32	0	0		00001	Justify
14		50	1	00	00	01000	Justified $N_i \rightarrow L(N_i)$
15		43	3	00	01	00004	Repeat
16		40	0			00006	
17							EXIT
20	0	00	0	00	00	00001	
1							
2							
3							LOC
4							1000-1000 + N N numbers
5							2000-2000 + N no. of shifts
6							
7							
0							Use $I^3 + I^4$ to count shifts
1							Use $I^1 + I^2$ to count words
2							
3							
4							
5							
6							
7							

MOBIDIC MACHINE
CODING FORM

PROGRAM Floating ADD
ROUTINE _____
PROJECT/W.O. _____

LOCATION	±	OP	K γ	K β	J	α	COMMENTS
0		CLA	0			70002	
1		STR	1			00061	Store I^2
2		LOD	0	00	02	70014	PCS→I^2
3		ADB	0	00	04	70014	PCS + 4→PCS
4		CLA	2			00001	Char. A
5		SUB	2			00003	Char. A - Char. B
6		SRL	0			00007	
7		TRZ	1			00014	Out to Routine
10		TRN	1			00056	to neg. char ≤ -128
11		CLA	2			00000	Mant A
12		LOD	2	00	11	00001	Char.A
13		TRU	1	00	00	00031	L(Store I^2)
Routine 14		SLL	0	00	00	00007	
15		TRN	1			00034	Char. B > Char. A
16		RPA	1			00020	
17		CLA	2			00002	Mant. B
20		SHR	0				
21		LOD	2	00	11	00001	Char. A→Q
22		ADD	2		1	00000	Mant. A
23		SNR	1	01	00	00043	Exit on Overflow
24		TRZ	1			00053	Exit on Zero Mantissa
25		NRM	1			00062	
26		CYL	0			00045	Mant.→Q Char→A
27		SUB	1			00062	Adjust Char..
30		CYL	0			00045	Mant.→A Char.→Q
31		LOD	1	00	02	00061	Restore I^2
32		TRS					EXIT
(Spare) 33	0	00	0	00	00	00001	L(Lo Order One)
Neg. 34		RPA	1			00037	
35		LOD	2	00	11	00003	Char. B→Q
36		CLA	2			00000	Mant. A
37		SHR	0	00	00		Shift Right (Char. A - Char. B) Places

MOBIDIC MACHINE
CODING FORM

PROGRAM Floating ADD
ROUTINE _____
PROJECT/W.O. _____

LOCATION	±	OP	K γ	K β	J	α	COMMENTS
40		ADD	2	00	01	00002	Mant. B
41		TRU	1			00023	to test for overflow
(Spare) 42	0	40	0	00	00	00000	L(Hi-order one)
Overflow 43		ADD	1			00033	Lo-order one
44		SHR	0			00001	
45		ADD	1			00042	Hi-order one
46		CYL	0			00045	
47		ADD	1			00033	Lo-order one
50		CYL	0			00045	
51		TRU	1			00031	to exit
(Spare) 52	1	77	7	77	77	77777	L (all ones)
Zero Mantissa 53		CLA	0			70000	Clear Accumulator
54		LOD	1	0	11	00052	All ones
Dif in Char. -128 55		TRU	1			00031	to EXIT
56		CLA	2	00	00	00002	Mant. B
57		LOD	2	00	11	00003	Char. B
60		TRU	1			00031	to EXIT
61				STORAGE			STORE I^2
62				STORAGE			STORE number of shifts
3							
4							
5							
6							
7							
0							
1							
2							
3							
4							
5							
6							
7							

In the routine presented here the numbers and corresponding exponents are assumed to be arranged in storage as shown below:

The two mantissas and corresponding exponents are stored in the four memory locations immediately following the TRL instruction which transfers control to the subroutine. This transfer instruction stores the address of the first mantissa (location n + 1) in the program counter store. This permits reference to the four quantities which figure in the operation without determining their actual addresses.

At the end of the floating-point routine, control is transferred back to the main routine (location n + 5). The mantissa of the sum will appear in the accumulator and the exponent of the sum is stored in the Q-register. Before the transfer back to the main routine takes place, the sum must be tested for possible overflow. In case of overflow, the sum is shifted to the right by one place, a one is entered into bit position 36, and 1 is added to the exponent of the sum. If there is no overflow, the sum is normalized and the number of shifts required is subtracted from the exponent.

The location used to store the first instruction of the subroutine is stored in index register number 1 (I^1), and reference to all memory locations within the subroutine are made relative to this first location. The subroutine will therefore operate unchanged no matter where it is eventually stored in memory.

Index register number 2 is used to store address n + 1, that is, the address of the first mantissa. However, the original contents of I^2 are saved before this register is used in the subroutine, and are restored later. Thus, the subroutine will destroy only the contents of the registers in the arithmetic unit and the contents of I^1. All other machine locations can be used by the main program.

6. Order Code Summary. The order codes which were presented in Section III and illustrated in Section IV are summarized in Table V which follows. The input-output instructions are treated separately at the end of the table.

The state of all internal registers and memory locations affected by the various internal orders is shown as it appears at the end of the corresponding instruction. The table also indicates whether a given instruction can be indexed and whether overflow is possible. The programmer should use this table as a permanent reference during problem preparation.

TABLE V. SUMMARY OF OPERATION CODES (cont.)

Instruction	α	A	Q	B	I^γ	$I^{\gamma+1}$	Notes	Time
ADB 24	C(α) + β	C(α) + β	C(A)	$\sigma_{SN} = +$; $\beta \rightarrow B$ $\sigma_{SN} = -\begin{cases} \text{if } \lvert C(\alpha)\rvert \geq \beta; & C(\alpha)' \rightarrow B,\ B_{SN} = \sigma_{SN} \\ \text{if } \lvert C(\alpha)\rvert < \beta; & 0 \rightarrow B \end{cases}$	$C(A)_{1-12}$		Cannot be indexed Overflow possible but not detected	26μs
ADD 12	C(α) + β	C(A) + C(α)		$A_{SN} = \sigma_{SN}$; $C(\alpha) \rightarrow B$ $A_{SN} \neq \sigma_{SN} \begin{cases} \text{if } \lvert C(A)\rvert \geq \lvert C(\alpha)\rvert; & C(\alpha)' \rightarrow B,\ B_{SN} = \sigma_{SN} \\ \text{if } \lvert C(A)\rvert < \lvert C(\alpha)\rvert; & 0 \rightarrow B \end{cases}$			Overflow is possible	16μs
ADM 13	C(α) + β	C(A) + C(α)		$A_{SN} = +$; $C(\alpha) \rightarrow B$ $A_{SN} = -\begin{cases} \text{if } \lvert C(A)\rvert \geq \lvert C(\alpha)\rvert; & C(\alpha)' \rightarrow B,\ B_{SN} = \sigma_{SN} \\ \text{if } \lvert C(A)\rvert < \lvert C(\alpha)\rvert; & 0 \rightarrow B \end{cases}$			Overflow is possible	16μs
CAM 11	C(α) + β	|C(α)|						16μs
CLA 10	C(α) + β	C(α)						16μs
CLS 14	C(α) + β	C(α), $\sigma_{SN} \rightarrow A_{SN}$						16μs
CSM 15	C(α) + β	-|C(α)|						16μs
CYL 35	C(α) + β	A, Q cycled left amod 128 places						$\alpha \leq 14$; 16μs $\alpha > 14$; $2 + \alpha + \alpha \pmod 2$μs
CYS 34	C(α) + β	A cycled left amod 128 places					Sign not included	$\alpha \leq 14$; 16μs $\alpha > 14$; $2 + \alpha + \alpha \pmod 2$μs
DVD 22	C(α) + β	Remainder	Quotient				Overflow is possible	88μs (18μs if overflow)
DVL 23	C(α) + β	Remainder	Quotient				Overflow is possible	88μs (18μs if overflow)
HLT 00	C(α) + β						Computer stops	16μs
LDX 53	C(α) + β				β	α	Cannot be indexed	16μs
LGA 03	C(α) + β	Logical Sum C(α) + C(A)					Signs Included	16μs
LGM 02	C(α) + β	Bit by bit Logical Product C(A)C(α)					Signs Included	16μs

TABLE V. SUMMARY OF OPERATION CODES (cont.)

Instruction	α	A	Q	B	$I^{\gamma+1}$	I^{γ}	Notes	Time
LGN 04		$C(\alpha_{1-37})'$					C(β) = 1; 0 —> C(β), α —> PC C(β) = 0; C(PC) + 1 —> PC	16μs
LOD 51				C(α)			C(β) = 0; 1 —> C(β), α —> PC C(β) = 1; C(PC) + 1 —> PC	16μs
							Signs not included	$\alpha \leq 14$; 16μs $\alpha > 14$; 2 + α + α(mod 2)μs
MLR 21		C(A)C(α) High order bits rounded	C(A)C(α) Low order	Q_{36} = 0; C(α) —> B Q_{36} = 1; 0 —> B			Overflow is possible	16μs
MLY 20		C(A)C(α) High order	C(A)C(α) Low order	C(α)			C(A) < C(α); C(PC) + 1 —> PC C(A) > C(α); C(PC) + 2 —> PC C(A) = C(α); C(PC) + 3 —> PC	22μs
MOV 52			α on RPT-MOV only	C(α)			C(PC) + 1 —> PCS α —> PC Cannot be Indexed	16μs
MSK 55	Bit by bit logic Q_i = 1; $\alpha_i = A_i$ Q_i = 0; $\alpha_i = \alpha_i$			C(A)	β		A_{SN} = + ; C(PC) + 1 —> PC A_{SN} = - ; α —> PC	16μs
NRM 37	Number of shifts (x 2^{-36}) C(A) = 0, 36 x 2^{-36}	High order zeros shifted off; low order zeros inserted					A_{SN} = - ; C(PC) + 1 —> PC A_{SN} = + ; α —> PC	16μs
RPA 54	$C(A_{1-15})$ $C(\alpha)_{1-15}$			C(A)			PCS —> PC	16μs
RPT 01							β_{16} = 0, TRA = 1; 0 —> PC Otherwise; α —> PC	16μs
SBB 25	C(α) - β	C(α) - β	C(A)	$^{\alpha}SN = +\begin{cases} |C(\alpha)| \geq \beta; \beta' \rightarrow B, B_{SN} = \alpha_{SN} \\ |C(\alpha)| < \beta; 0 \rightarrow B \end{cases}$ $^{\alpha}SN = -; \beta \rightarrow B$		C(1^γ) + β If $1^{\gamma+1} \neq 0$; Otherwise no change	C(1^γ) ≠ 0; α —> PC C(1^{γ+1}) = 0; C(PC) + 1 —> PC Cannot be indexed	22μs
SBM 17	C(A) - |C(α)|			$^{A}SN = -; C(\alpha) \rightarrow B$ $^{A}SN = +\begin{cases} |C(A)| \geq |C(\alpha)|; |C(\alpha)'| \rightarrow B, B_{SN} = \alpha_{SN} \\ |C(A)| < |C(\alpha)|; 0 \rightarrow B \end{cases}$		C(1^γ) - 1 If $1^{\gamma+1} \neq 0$	C ⌊$(A)_{1-36}$⌋ = 0; α —> PC C ⌊$(A)_{1-36}$⌋ ≠ 0; C(PC) + 1 —> PC	16μs
SEN 05*				TRA = 1; C(PC) —> B				
SHL 30	C(A) Shifted left α mod 128							
SHR 32	C(A) Shifted right α mod 128							
SLL 31				C(A, Q) Shifted left α mod 128				

*Trapping is possible

TABLE V. SUMMARY OF OPERATION CODES (cont.)

Instruction	α	A	Q	B	Notes	Time
SNR 07*				TRA = 1; C(PC) —> B		16μs
SNS 06*				TRA = 1; C(PC) —> B		16μs
SRL 33		C(A, Q) Shifted right α mod 128 places				
STR 50	C(A)					16μs
SUB 16		C(A) - C(α)		$A_{SN'} = {}^{\alpha}SN \begin{cases} \geq & C(A) - C(\alpha)'; C(\alpha)' \rightarrow B, B_{SN} = {}^{\alpha}SN \\ < & C(A) - C(\alpha); 0 \rightarrow B \end{cases}$ $A_{SN} \neq {}^{\alpha}SN; C(\alpha) \rightarrow B$	Overflow is possible	16μs
TRC 47			C(A)	α		
TRL 41*				TRA = 1; C(PC) —> B		16μs
TRN 46*				TRA = 1; C(PC) —> B		16μs
TRP 44*				TRA = 1; C(PC) —> B		16μs
TRS 42*				TRA = 1; C(PC) —> B		16μs
TRU 40*				β_{16} = 0, TRA = 1; C(PC) —> B Otherwise; B unchanged		16μs
TRX 45*				TRA = 1; C(PC) —> B TRA = 0; α —> B		16μs
TRZ 45*				TRA = 1; C(PC) —> B		16μs

TABLE V. SUMMARY OF OPERATION CODES (cont.)

Instruction	Operation	Notes	Time (Excluding tape time)
WAN 74	Writes k (mod 2^9) words on IOD_j (Input Output Device). First word is at location a. Words record six bits at a time. If interpret sign flip-flop is set, the sign is treated as a six-bit character. If j is magnetic tape, block marks inserted before and after k words.	Six characters per word plus one character for sign.	16μs + 8μs per word
WOK 76	Three-bit octal digit is converted to six-bit equivalent and written on paper tape. Sign is always interpreted.	Cannot be used for magnetic tape. Thirteen Six-bit characters per word.	16μs + 8μs per word
ROK 72	Reads k (mod 2^9) words from IOD_j and stores in memory beginning at a. Treats thirteen six-bit characters as octal word and sign. If Stop code is read, order is terminated whether or not k = 0.	Cannot be used for magnetic tape.	16μs + 8μs per word
RAN 70	If j is magnetic tape unit, k (mod 2^8) words or blocks are read into memory beginning at location a. If k_9 = 1, k_{1-8} signify number of blocks; if k_9 = 0, k_{1-8} signify number of words. With paper tape, ISN = 1 gives continuous reading until a stop code is received or until k_{1-9} = 0; if ISN = 0 and k_9 = 1, reading will continue until a stop code is received; if ISN = 0 and k_9 = 0, only one character is read in.	Low order zeros inserted if full word is not used.	16μs + 8μs per word
SKP 66	Tape unit j skips forward k (mod 2^8) blocks.	Magnetic tape only.	16μs
BSP 67	Tape unit j backspaces k (mod 2^9) blocks.	Magnetic tape only.	16μs
RRV 71	Same as RAN except reads in opposite direction.	Magnetic tape only.	16μs + 8μs per word
RWD 77	Rewinds tape j.	Magnetic tape only.	16μs
WWA 75	Searches for beginning block marks. Then writes k (mod 2^9) words starting at memory location a and puts block mark after k (mod 2^9) words.	If beginning block mark is not found, whole tape is searched.	16μs + 8μs per word

TABLE VI
SUMMARY OF OPERATION CODES
IN NUMERICAL ORDER
WITH PROGRAMMING MANUAL
PAGE REFERENCES

OCTAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	μSEC	OPERATION	PAGE	INDEXABLE?
00	HLT		16	Halt	26	No
01	RPT	γβα	16	Repeat	29	Yes
02	LGM	γα	16	Logical Multiply	24	Yes
03	LGA	γα	16	Logical Add	24	Yes
04	LGN	γα	16	Logical Negation	24	Yes
05	SEN	γβα	16	Sense	26	Yes
06	SNS	γβα	16	Sense & Set	26	Yes
07	SNR	γβα	16	Sense & Reset	27	Yes
10	CLA	γα	16	Clear & Add	17	Yes
11	CAM	γα	16	Clear & Add Mag.	21	Yes
12	ADD	γβα	16	Add	19	Yes
13	ADM	γβα	16	Add Magnitude	20	Yes
14	CLS	γα	16	Clear & Subtract	21	Yes
15	CSM	γα	16	Clear & Subtract Mag.	21	Yes
16	SUB	γβα	16	Subtract	20	Yes
17	SBM	γβα	16	Subtract Magnitude	20	Yes
20	MLY	γα	86	Multiply	21	Yes
21	MLR	γα	86	Multiply & Round	21	Yes
22	DVD	γβα	88 (18 if overflow)	Divide	21	Yes
23	DVL	γβα	88 (18 if overflow)	Divide Long	22	Yes
24	ADB	γβα	26	Add Beta	29	No
25	SBB	γβα	26	Subtract Beta	29	No
26						
27						
30	SHL	γβα	α ≤ 9 = 16; α > 9, 8 + α - α(MOD 2)	Shift Left	22	Yes
31	SLL	γβα		Shift Left Long	22	Yes
32	SHR	γα	α ≤ 14 = 16; α > 14, 2 + α - α(MOD 2)	Shift Right	22	Yes
33	SRL	γα		Shift Right Long	22	Yes
34	CYS	γα	α ≤ 14 = 16; α > 14, 2 + α + α(MOD 2)	Cycle Short	24	Yes
35	CYL	γα		Cycle Long	24	Yes
36						
37	NRM	γα	18 + n - n(Mod 2) n = # of Shifts	Normalize	22	Yes
40	TRU	γβα	16	Uncond. Transfer	25	Yes

OCTAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	μSEC	OPERATION	PAGE	INDEXABLE?
41	TRL	γβα	16	Load PCS & Transfer	29	No
42	TRS		16	Transfer to PCS	29	No
43	TRX	γβα	22	Transfer on Index	28	No
44	TRP	γα	16	Transfer on Positive Acc.	26	Yes
45	TRZ	γα	16	Transfer on Zero Acc.	26	Yes
46	TRN	γα	16	Transfer on Negative Acc.	26	Yes
47	TRC	γα	22	Compare	26	Yes
50	STR	γα	16	Store	17	Yes
51	LOD	γβα	18	Load	17	Yes
52	MOV	γβα	26	Move	17	No
53	LDX	γβα	16	Load Index	28	No
54	RPA	γα	22	Replace Address	17	Yes
55	MSK	γα	22	Replace Through Mask	17	Yes
56						
57						
60						
61						
62						
63						
64						
65						
66	SKP	kj	16 + tape time	Skip	34	No
67	BSP	kj	16 + tape time	Backspace	34	No
70	RAN	kjα	16 + 8 per word + tape time	Read Alphanumeric	33	No
71	RRV	kjα	16 + 8 per word + tape time	Read Reverse	34	No
72	ROK	kjα	16 + 8 per word + tape time	Read Octal	33	No
73						
74	WAN	kjα	16 + 8 per word + tape time	Write Alphanumeric	32	No

OCTAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	μSEC	OPERATION	PAGE	INDEXABLE?
75	WWA	kjα	16 +8 per word +tape time	Rewrite Alphanumeric	35	No
76	WOK	kjα	16 + 8 per word +tape time	Write Octal	33	No
77	RWD	j	16 + tape time	Rewind	35	No

SYLVANIA ELECTRONIC SYSTEMS
A Division of Sylvania Electric Products Inc.
63 SECOND AVENUE, WALTHAM, MASS.

DIVISION FIELD OFFICES

Dayton, Ohio	333 W. First St.
Los Angeles, Calif.	6506 E. Gayhart St.
Philadelphia, Pa.	4700 Parkside Ave.
Rome, N. Y.	225 N. Washington St.
Washington, D. C.	734 15th St., N.W.