RESTRICTED DISTRIBUTION

Univac®

LARC PROGRAMMING

THE COMPUTING UNIT

MAY 1961

Remington Rand Univac®
DIVISION OF SPERRY RAND CORPORATION
UNIVAC ENGINEERING CENTER • PHILADELPHIA

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1
INTRODUCTION

The Univac® Larc* Computing System basically comprises two units, the Computing Unit,** and the processor. The Computing Unit is the primary computer; the processor (a secondary computer) handles the input-output operations. The two units are programmed independently, requiring only a minimum of intercommunication.

This programming manual is designed to provide the experienced programmer with the information necessary to write programs for the Larc Computing Unit. The publication Univac Larc Programming, The Processor provides similar information for programming the processor. Both manuals assume a familiarity with the essential features of the system. General information is available in the publication Univac Larc System, General Description.

The manual begins by introducing the programmer to the function of the Computing Unit in the system (section 2). This preliminary matter explains how instructions are sequenced, in general how errors and contingencies affect a program, and how the various parts of an instruction word are employed.

The main body of the manual (section 3) presents the Computing Unit instruction repertory. The individual instructions are described by class; information that applies in general to instructions in a class is given at the beginning of each class. A condensed list of all the instructions is found in appendix A.

---

* Trademark of the Sperry Rand Corporation.

** A Larc system may contain one or two Computing Units. This manual is written for the Larc System, serials 1 and 2, which contains one Computing Unit.

Section 4, on input-output operations, describes the numeric and alphanumeric codes and the input-output equipment including suggestions for the use of each of the devices. The method of communication between processor and Computing Unit is described in detail.

Operating procedures are discussed in section 5. The programmer must be familiar with this information in order to instruct the operator during debugging and final runs. Subjects such as program loading, manual intervention, visual display, and paper tape handling are covered in this section.

Section 6 consists of a complete discussion of indirect addressing. Included are the reasons for using indirect addressing, a description of the method of coding it, and coded examples of its use.

Details of tracing modes, errors, and contingencies are available in section 7. In this section, the addressable flip-flops are discussed at length, with information on the way in which they are set, tested, and reset. (A complete list of addressable flip-flops is found in appendix B.) Suggestions are made as to procedures which may be followed in the routines handling these operations.

The procedures to follow in utilizing the processor program written by Remington Rand are described in appendix C. This appendix includes an explanation of the summary orders or pseudo-code converted by the processor program into processor instructions.

SECTION 2

LOGICAL OPERATION OF THE COMPUTING UNIT

2-1. GENERAL

The Larc computer is an extremely high-speed computing device. Its high speed is obtained in part by using overlapping instructions, that is, the computer does not wait until an instruction has been executed before extracting the next from storage. Consequently, instructions follow each other rather closely through the stages of the control unit. In fact, as many as four instructions may be in the control unit at any one time.

This overlapping of instructions of course increases the complexity of the computer and imposes certain sequencing restrictions on the programmer. In some cases by careless use of instructions a programmer may increase the running time of his program. In rare instances errors can be caused by failing to observe the restrictions. Section 3 of this manual points out the more obvious restrictions on every individual instruction.

This section presents a simplified description of the Computing Unit control operations. (See figure 2-1 for a block diagram of the major units in the Computing Unit.) Because the main function of the control unit is the sequencing and execution of instructions, each component of this unit is explained and its function is described. A brief outline of the sequencing of instructions is given and a typical instruction is followed through the various stages of modification, decoding, and execution. The section ends with a brief introduction to the error and contingency routines and the tracing mode.

With this background information the programmer will be able to understand more clearly the necessity of precautions in the sequencing of instructions and in many instances will be able to work out for himself the effect of certain sequences on his program.

Before the programmer can fully understand the remainder of this section, he must be aware of the structure of the Larc Computing Unit instruction word. Hence, a brief description follows. (Further information on this topic will be found in section 3.)

A Larc word consists of 12 characters. A generalized instruction word is shown as follows:

T II AA BB MMMMM

The T digit, which need not be numeric, is known as the tracing digit. Certain values of this character lead to abnormal operation of the control unit. In general, the tracing digit will usually be a period, which will cause normal operation. The reader may assume in the discussions that follow that the tracing digit is a period unless otherwise stated.

The two I, or instruction, digits determine the operation that is to be carried out. The instruction (operation) code is discussed fully in section 3. In the present section any particular instruction code used will be explained as it appears.

The two A digits specify the address of a fast, or A, register. Fast registers are fast-access, 12-digit storage registers. Besides serving as fast-access storage, they also have some special properties which are explained in section 3. In this section the reader may assume that in any operation one of the operands will be the contents of the fast register specified by the A digits of the instruction.

The M digits of an instruction usually specify a main storage address from which the second operand of many instructions is extracted. The M digits may also specify the number of shifts in a shift instruction or the address to which control is transferred in a transfer-of-control instruction.

The B digits also specify the address of a fast register. In this case the contents of the register are used to modify the M address of the instruction. The five least-significant digits of the B fast register are known as the modifiers and are added to the M digits of the instruction to produce the modified address. Any carry produced outside the five least-significant-digit positions is ignored. Hence, an address can be increased or decreased by modification. For example, if the modifier 99999 is added to the address 06500, the modified address will be 06499. Note that modification does not alter the instruction in its storage location; the modification takes place only in the control unit. Note also that the fast registers used in modification are the same fast registers used to hold operands.

The general pattern of an arithmetic operation is that an operand from a fast register (A) and an operand from a main storage location (M) enter the control unit and the result is sent back to the fast register.

This brief explanation of instructions is sufficient to enable the reader to follow the arguments presented in the rest of this section.

2-2. THE CONTROL UNIT

The control unit may be generally described as the heart of the computer. (See figure 2-1.) Its functions are to fetch instructions from memory in correct sequence, to decode them, to bring operands from core

storage and fast registers, to perform a variety of arithmetical and logical operations on the operands, and to return the results to storage. Figure 2-1 is, of course, a highly simplified diagram and shows only a few of the many interconnections of the control unit. More complete diagrams are included in the relevant logic manuals.

The individual major components making up the control unit are described in paragraphs 2-3 through 2-7.

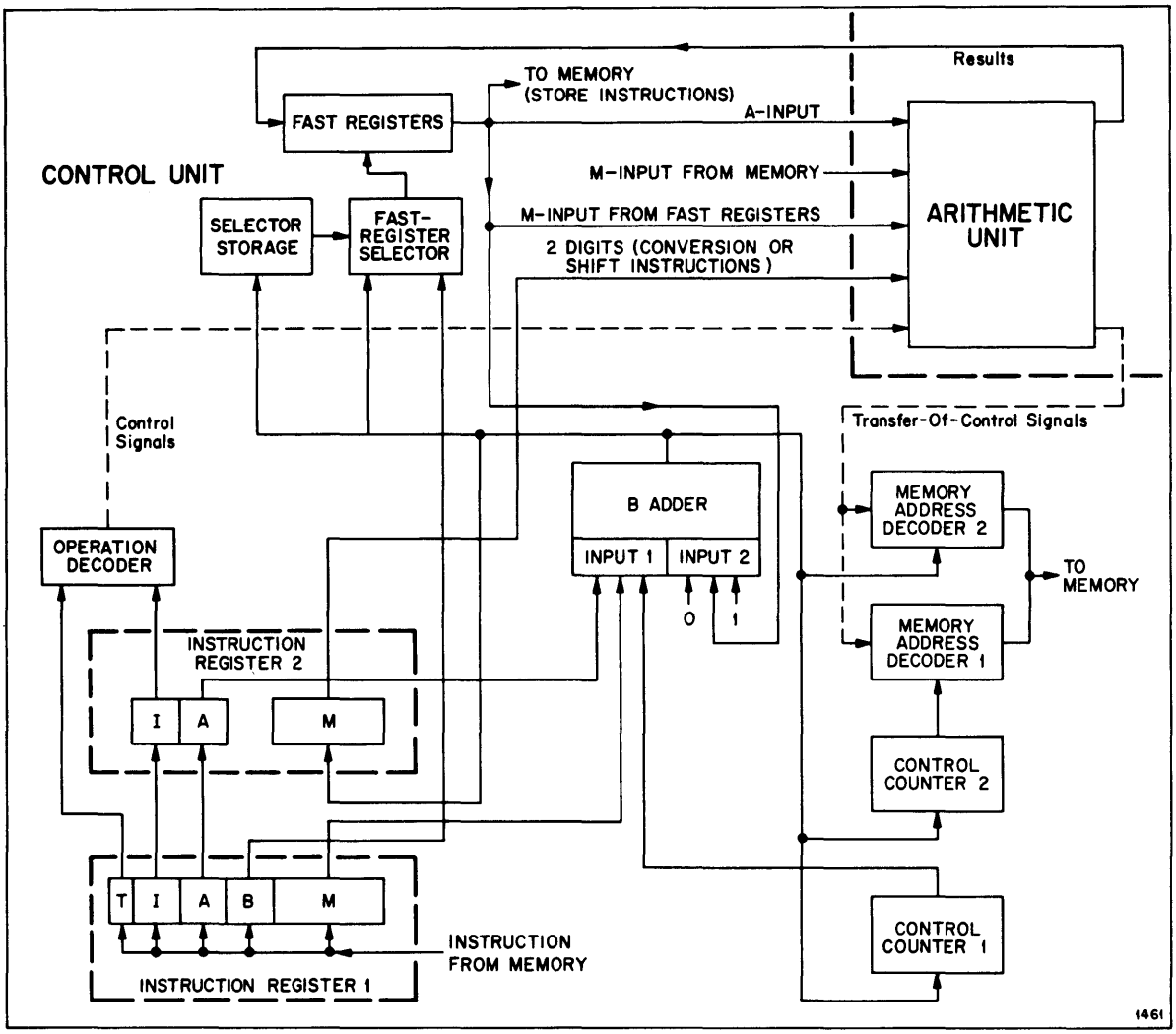

Figure 2-1. Computing Unit Block Diagram

2-3. INSTRUCTION REGISTERS

There are two instruction registers in the Larc system control unit. They are special storage registers which hold instructions while they are being decoded. All instructions entering the control unit pass through these two registers.

Instruction register 1 (IR1) is the preliminary storage register and accepts instructions directly from memory. While an instruction is waiting

in IR1, its M address is modified by the contents of the specified B register.

Instruction register 2 (IR2) is the final storage register for instructions. This register receives the I and A digits from the first instruction register and also stores the modified M address. After modification takes place the B digits are no longer required and are not stored in IR2. The instruction is held in IR2 until the final decoding is completed and the necessary information for execution of the instruction has been brought into the control unit.

2-4. CONTROL COUNTERS

A control counter is used to control the extraction of instructions from storage and to ensure that they are extracted in correct sequence. Because of the overlapping operations in the Larc Computing Unit two control counters, numbers 1 and 2, are required. One is used in the normal sequencing of instructions; the other is used when a transfer-of-control instruction is being decoded.

Control counter 1 (C1) contains the five-digit address of the instruction in IR1. In a normal sequence of instructions, that is, a sequence containing no transfers of control, the address in C1 is increased by 1 after each instruction has been executed causing the control counter to fetch the next instruction and store it in IR1.

When a transfer-of-control instruction is decoded, the modified M address is sent to control counter 2 (C2). If transfer takes place, the contents of C1 and C2 are interchanged so that C1 contains the first address of the new sequence of instructions, and C2 contains the address of the next instruction following the transfer-of-control instruction. If the transfer of control does not take place (C2) and (C1) are not exchanged. In either case, the contents of C2 remain unaltered until the next transfer-of-control instruction is decoded. Thus, at any stage in a program C2 contains the address of the instruction to which control was not transferred during the execution of the previous transfer-of-control instruction. If in instructions 80 and 81, B-modifier increment (or decrement) and transfer, no transfer of control takes place, (C2) is advanced to M + 1 before the control unit proceeds to the next instruction.

Example:

Storage location 25 contains a test-for-zero instruction.

$$(00025) = .\ 72\ \ 01\ \ 00\ \ 00010$$

The instruction tests the contents of fast register 01. If the contents equal zero, control is transferred to storage location 10; if the contents do not equal zero the control unit continues with the next instruction in sequence, address 00026.

Initially, therefore, 00026 $\longrightarrow$ C1 and 00010 $\longrightarrow$ C2.

If (O1) = 0, then (C1) and (C2) are interchanged; that is, after the transfer of control:

(C1) = 00010

(C2) = 00026

If (O1) $\neq$ 0, then (C1) and (C2) are unchanged; that is, after the completion of the instruction:

(C1) = 00026

(C2) = 00010

In the first case, the next instruction to be executed will be taken from storage location 10; in the second case it will be taken from storage location 26.

This explanation of the operation of the control counters is simplified intentionally and is inexact in one or two particulars. For the programmer's use it is entirely sufficient, however.

2-5. THE B ADDER

The B adder is a five-digit parallel adder whose primary function is the B-modification of M addresses. The adder also performs many other operations during the fetching, decoding, and execution of an instruction. Taking the execution of a simple instruction as an example, the first operation performed by the B adder is the addition of the number 00001 to the five-digit address in C1. This produces the address of the next instruction to be brought into IR1.

The B adder is also used to interchange the contents of C1 and C2 after a transfer-of-control instruction, and for various incrementing and decrementing operations.

2-6. MEMORY ADDRESS DECODERS

There are two memory address decoders in a Larc control unit. They are used to decode storage location addresses before fetching words from storage. The memory address decoders in fact partially decode the M address, selecting the cabinet, the storage unit within the cabinet, and partially selecting the storage location. The decoded information is sent over the address lines to the selected storage unit where the final decoding of the storage location takes place.

Memory address decoder 2 (MAD2) receives from the B adder the addresses it is to decode. These may be addresses of instructions or of operands. MAD2 also receives control signals which direct it to decode the next instruction address in sequence when a transfer of control is not required.

Memory address decoder 1 (MAD1) receives the addresses it is to decode from control counter 2. These addresses are only decoded and sent to the memory when control signals are received indicating that a conditional transfer of control is to occur.

2-7. OPERATION DECODER

The operation decoder decodes the operation code bits of an instruction in IR2. The decoding process generates control signals which govern the rest of the circuits during the execution of the instruction.

The decoder is also used to decode the tracing-mode digit from IR1. Depending on the character, control signals are generated to signal tracing mode, indirect-addressing mode, or normal operation.

2-8. THE ARITHMETIC UNIT

The arithmetic unit (AU) carries out all the arithmetic and logic operations in the Computing Unit. Before the AU can execute an instruction it needs operands and information about the operation to be performed. Control information enters the AU before the operands in order to give the unit sufficient time to prepare its circuits. The information derived from the decoding of the operation digits of the instruction, arrives in the form of control signals from the operation decoder.

There are two input channels to the AU for operands. These channels are known as the M-input and the A-input. In general, operands for the first input come from the memory location specified by the M digits of the instruction, whether these refer to a storage location or to a fast register. Operands for the second input come from the fast register specified in the A-digit positions of the instruction word. The two operands arrive at the AU simultaneously.

In shift and conversion instructions no operand is supplied to the M-input channel. Only the two least-significant M digits, after modification, are used and they specify the number of shifts to be performed or the conversion scale factor. They enter the AU by a special input line from IR2.

The output from the AU may be results which are sent to fast registers or storage locations, or may be control signals which govern transfer-of-control operations. Error and contingency signals are also generated by the AU.

The AU operates independently after it has been supplied with operands and control signals.

2-9. FAST REGISTERS

The fast registers in the Larc Computing Unit, known as A registers or B registers according to whether they are used as arithmetic registers or index registers, do not have their addresses decoded by the memory address decoders, but by a special unit known as the fast-register selector.

This unit accepts pairs of digits from the instruction registers or from the B adder and selects the fast register specified. The selector prepares the fast register to accept information or to transmit it when the appropriate control signal is received. In some cases fast-register-result addresses are temporarily stored in a two-digit storage before being sent to the fast-register selector. The two-digit storage is known as the selector storage unit.

2-10. ADDRESSABLE FLIP-FLOPS

Flips-flops are one-bit storage devices which, at any time, are in either an on (set) condition or an off (reset) condition. In the Larc computing system there are many of these storage devices (not shown in figure 2-1) which are used for a variety of control purposes.

Appendix B to this manual comprises a list of addressable flip-flops in the computing unit, that is, the flip-flops which can be addressed by an instruction. The list also indicates the degree of control the programmer has over the flip-flops, that is, whether he can test, set, or reset them.

The flip-flops in the Larc system have many functions. Some are solely for the use of the programmer and are entirely under his control; others are used for indicating errors or contingencies and may only be tested and reset by the programmer. Still others serve as a means of communication between Computing Unit and processor.

2-11. THE HIGH-SPEED BUS

The high-speed bus is a fast-transfer channel between the main storage units and the rest of the system. It is used by the Computing Unit, by the central processor, and by the synchronizers of the various input-output units by way of the processor dispatcher.

It requires 1/2 microsecond to transfer a word on the high-speed bus. Thus, during every memory cycle (4 microseconds) the bus can handle eight words. These 1/2-microsecond periods, or time slots, are allocated in a special way to the various units using the bus. Figure 2-2 shows how the eight time slots in a 4-microsecond memory cycle are distributed.

TIME SLOTS (1/2-MICROSECOND EACH)							
0	1	2	3	4	5	6	7
CENTRAL PROCESSOR	COMPUTING UNIT 1 INSTRUCTIONS	COMPUTING UNIT 2 OPERANDS	PROCESSOR DISPATCHER	[NOT USED]	COMPUTING UNIT 2 INSTRUCTIONS	COMPUTING UNIT 1 OPERANDS	PROCESSOR DISPATCHER

1462

Figure 2-2. Allocation of Time Slots on the High-Speed Bus

2-12. EXECUTION OF AN INSTRUCTION

In order to illustrate the over-all operation of the control unit a simple instruction will be traced through the unit. Figure 2-1 shows a simplified block diagram of the control circuits and reference should be made to this figure in following the explanation in the succeeding paragraphs.

It is assumed that initially the address in C1 is 00400, and that storage location 00401 contains the following instruction:

T	I	A	B	M
.	01	05	12	00100

and that fast register 12 contains:

000 00 00 00050

The instruction tells the control unit to add the contents of storage location 00100 (modified by the contents of fast register 12) to the contents of fast register 5 and leave the result in register 5.

First, the contents of C1 (00400) are sent to input 1 of the B adder and the number 00001 is selected for the second input. The output of the B adder is the sum of the two inputs, or 00401, the address of the next instruction. This address is returned to C1 to be stored until required for selection of the next instruction after 00401. The address is also sent to MAD2 where the cabinet and storage unit containing storage location 00401 are selected. The partially decoded information is sent from the memory address decoder over the address lines to the relevant storage unit.

In the storage unit the final decoding is done and the selected word (00401) is transmitted over the high-speed bus to IR1.

As the instruction digits enter IR1 the tracing-mode digit (.) enters the operation decoder and is decoded to signify normal operation. Simultaneously, the two B digits of the instruction are sent to the fast-register selector, which decodes the digits and selects the correct fast register (register 12). The selector causes the five least-significant digits of the register (00050) to be transmitted to input 2 of the B adder. While the B-register selection takes place, the M digits of the instruction in IR1 (00100) are sent to input 1 of the B adder, and at the same time, the I and the A digits of IR1 proceed to IR2.

The two five-digit numbers arrive simultaneously at the adder and a sum is formed. The result (00150), which is the modified M address of the instruction, is sent to the M-digit positions of IR2, which now contains the instruction with the modified M address replacing the original M address. (The T digit and the B digits of the instruction are not retained after modification has taken place.) The modified M address is also sent to MAD2 which decodes the operand address. The operand is eventually transmitted from storage to the M-input of the AU.

The operation digits of the instruction in IR2 enter the operation decoder where signals are generated to direct the remaining stages of the instruction. In particular, control signals are sent to the AU to prepare it to handle the operands.

While the M address is being decoded the two A digits of the instruction are routed from IR2 through the B adder to the fast-register selector. The selector causes the contents of the specified A register (register 5) to enter the A-input of the AU. Simultaneously, the other operand enters the M-input channel from the high-speed bus. The AU has now been provided with all the information it requires and therefore carries out the addition and enters the sum in the AU result register.

Meanwhile, the A digits of the instruction have entered the B adder for the second time and on leaving are retained in the selector storage unit. When the result of the operation has been stored in the AU result register, the two A digits go from selector storage to the fast-register selector where they are decoded to select the fast register which receives the result. In this case the result is stored in fast register 5, the A-operand register. (This is not the case, however, for all instructions.) The old contents of the A register are deleted and the new result is read in from the AU result register.

2-13. INSTRUCTION OVERLAPPING

It was stated in the introduction to this section that instructions are not processed serially by the control unit but overlap in time with preceding and succeeding instructions. The preceding paragraph explained how a single instruction was decoded and executed. The many stages in the process can be reduced to five basic operations, named in order, call instruction, B modification, operand select, execute, and result. Each of these operations can be considered to require 4 microseconds to carry out. The 4-microsecond time unit, known as a memory cycle, is the time actually required to read a word from or into a storage location.

Figure 2-3 shows a series of instructions passing through the above five stages. The instructions are staggered by a period of 4 microseconds. When instruction N is being called from storage, the previous instruction N - 1 is having its M address modified, the operands for N - 2 are being brought from memory, the AU is executing N - 3 and the results of N - 4 are being stored. In the next memory cycle, N + 1 is called from memory, N is B-modified, and so on.

The chart in figure 2-3 shows a series of simple instructions where the total execution time for each instruction is approximately 20 microseconds. Some more complex instructions, of course, take longer than 20 microseconds and succeeding instructions are delayed accordingly. Note that although each instruction takes about 20 microseconds to pass through the control unit, the next instruction is called after only 4 microseconds have elapsed. That is, for all practical timing purposes a simple instruction has an execution time of 4 microseconds. In fact, because the basic memory cycle is 4 microseconds and the time slots are fixed, any execution time is a multiple of 4 microseconds.

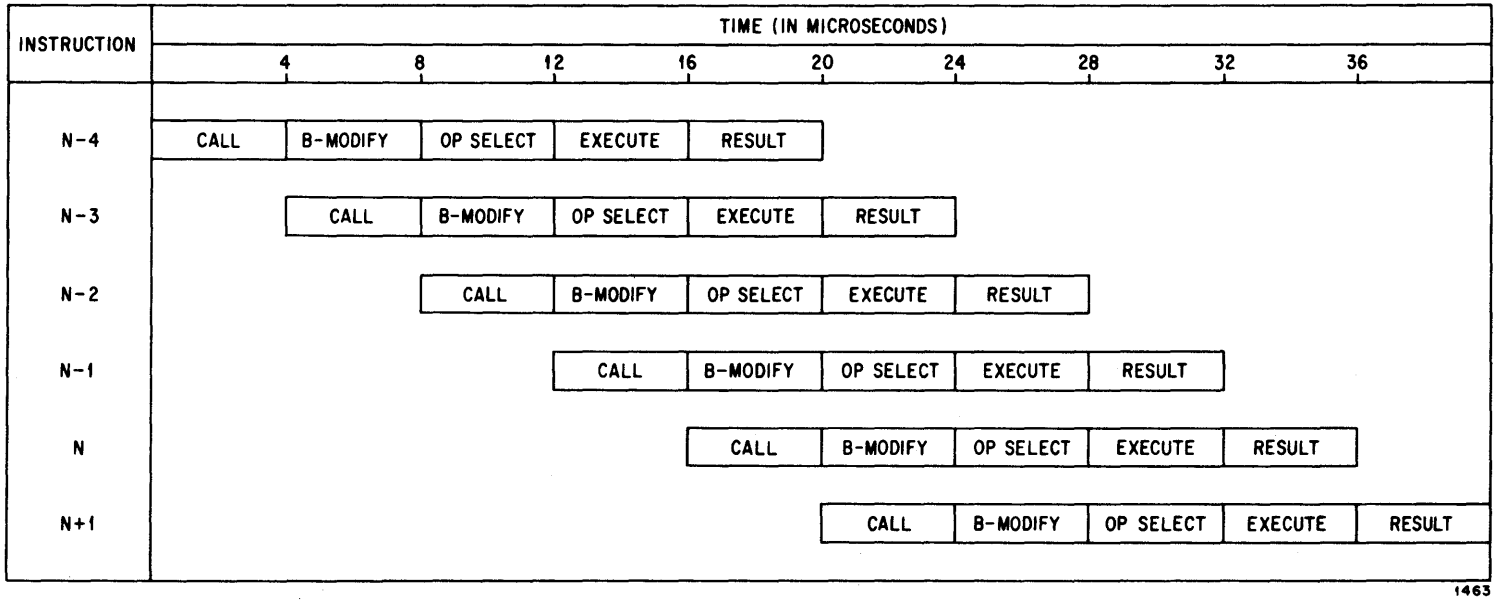

Figure 2-3. Instruction Overlapping

As an example of the use of the chart in figure 2-3, consider a sequence of instructions which contains an unconditional-transfer-of-control instruction. (See figure 2-4.) This instruction (N) merely transfers control to the instruction in the storage location specified by the modified M address of the transfer-of-control instruction. The instruction following the control transfer (N + 1) will therefore never be needed. However, due to the overlap of instructions, the instruction in location N + 1 will in fact be called for before the computer has fully decoded the unconditional-transfer instruction (operand-select cycle). Therefore, when the control unit has decoded the control-transfer instruction, a signal is sent which prevents instruction N + 1 from entering the first instruction register. The instruction specified by the M digits of the unconditional-transfer-of-control instruction will be called for in the next memory cycle. Thus, a memory cycle has been lost during the transfer of control. An unconditional-transfer-of-control instruction therefore has an effective execution time of 8 microseconds.

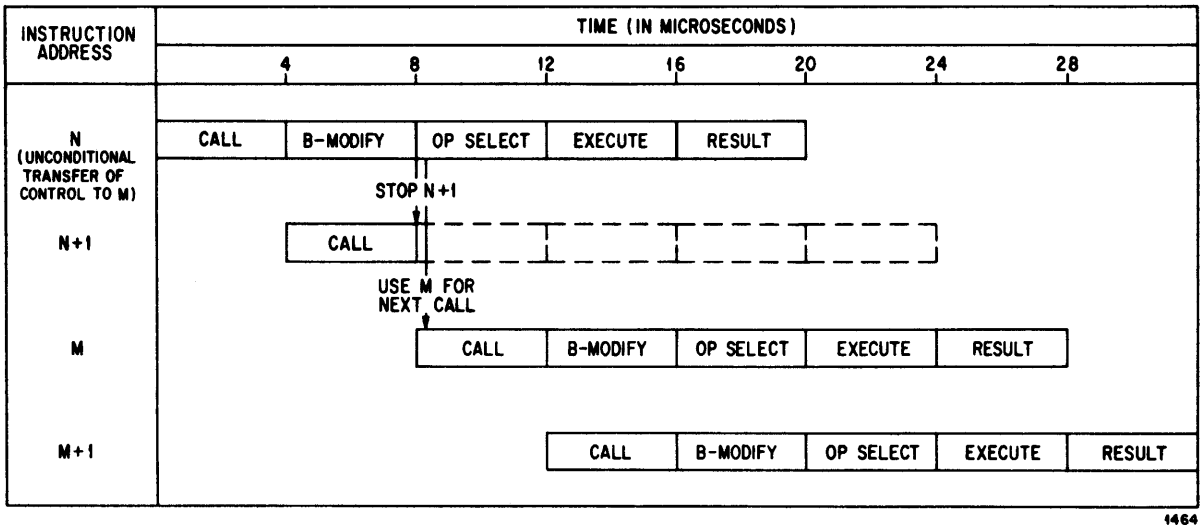

Figure 2-4. Sequencing of an Unconditional-Transfer-of-Control Instruction

2-14. CONTROL OF ERRORS, CONTINGENCIES, AND THE TRACING MODE

These three topics are dealt with at length in section 7. However, this section gives a useful brief description of the function of the control unit in these three areas.

An error is usually caused by a computer failure of some kind. However, incorrect programming also can cause errors to occur; for example, if an instruction contains the address of a non-existent storage location in the M-digit positions an error will be signified when the instruction is executed. There are several error flip-flops in the computer, each relating to a specific error. When the control unit detects an error it sets the related flip-flop or flip-flops. This automatically sets the master error flip-flop during the instruction result time. When the master error

flip-flop has been set, the control unit stores an unconditional-transfer-of-control instruction in storage location 02600 which serves as a return jump to the program. (The M digits of this instruction usually contain the address of the next instruction after the one in which the error occurred, but this is not always so.) Control is then transferred to the instruction in address 02601. Obviously, some routine for diagnosing errors must be in the computer with its first address at 02601. A general discussion of error routines is given in section 7. In general, however, when an error occurs the computer does not complete the instruction causing the error before entering the error routine, although in a few cases the instruction is actually completed. (See note, page 7-3.)

A contingency is caused by some programming error; overflow and sign anomalies are examples. The control unit handles contingencies in the same way as it does errors, that is, one or more contingency flip-flops are set, causing the master contingency flip-flop to be set. The address of the next instruction after the one in which a contingency occurred is stored as the M address of an unconditional-transfer-of-control instruction in 02700 and control is transferred to the instruction in address 02701. A contingency routine must be stored in the computer with its first instruction in storage location 02701. A general discussion of contingency routines is given in section 7.

When contingencies occur the address stored in 02700 is always that of the next instruction. An instruction in which a contingency occurs is always completed (and in a few cases the next instruction also is completed) before the computer enters the contingency routine.

The first character of an instruction, known as the tracing digit, is usually a period. Its presence causes normal execution of the instruction. The first character may also be any of the digits 1 through 9 or the ignore symbol (i).

There are nine tracing mode flip-flops in the Computing Unit corresponding to the digits 1 through 9. They may be set as the result of programmed instructions. Whenever the control unit decodes an instruction with one of the digits 1 through 9 in the tracing position, the corresponding flip-flop is tested. If it is in the reset state, the computer continues in normal operation exactly as if a period had been present. If, however, the flip-flop is in the set state, the computer enters the tracing mode before the instruction is executed.

When the computer enters the tracing mode control is in fact transferred to the error routine exactly as if an error had occurred. The error routine must be so written as to detect the tracing mode and transfer to a special tracing routine. This facility is designed to assist a programmer in debugging programs. The tracing routine, for example, may be programmed to print out the contents of certain registers before returning to the program. A general discussion of tracing routines is found in section 7. It is important to notice that instructions with tracing digits behave like any other instructions as long as the corresponding flip-flops are reset.

An ignore symbol (i) in the tracing position always causes the computer to enter the indirect-addressing mode. The M address of the instruction does not specify the operand but specifies a word where the operand

address may be found. The indirect-addressing mode is, in fact, more complex than this and a full explanation is found in section 6. The indirect-addressing mode does not use a separate routine as does the error, contingency, and tracing modes. The extraction of indirect addresses is carried out solely by the control unit circuits.

SECTION 3
INSTRUCTION DETAILS

The purpose of this section is to introduce the instruction repertory of the Larc Computing Unit. Instructions are presented by class and are described according to their normal use. The execution time in microseconds is specified for each instruction. The times given allow for overlap and each time is, in fact, the period that elapses from the end of the previous instruction to the end of the current instruction. (See appendix A for a condensed numerical list of instructions.)

3-1. INSTRUCTION FORMAT

The format of an instruction word for the Larc Computing Unit has already been described in section 2. Also in that section the mode of execution of an instruction was outlined. In this section, descriptions of the operation of individual instructions are designed specifically for the programmer's use. They are not intended to be precise descriptions of computer logic. In particular, as the contents of control counter 1 (C1) depend not only on the instruction being executed, but also on the two following (see figure 2-3), this control counter will not be mentioned in instruction descriptions. In order to avoid using C1, a hypothetical control counter C will be used. The contents of the hypothetical counter C, otherwise written as (C), will be the address of the instruction being executed; the symbolic notation, (C) + 1 $\longrightarrow$ C, means continue with the next instruction in sequence; and M $\longrightarrow$ C means transfer control to the instruction in storage location M. The programmer must remember that in fact there is no control counter such as C, and that C has been introduced merely as a device to simplify descriptions.

It will have been noted that there are three addresses in the Computing Unit instruction word. Two of these (A and B) are fast-register addresses and the third (M) may be either a fast-register address or a core-storage address. If any of these addresses in an instruction word exceeds the maximum available, errors will usually occur when the instruction is decoded. In a few instructions some of the addresses are not used and may exceed the maximum without causing errors. However, note that in all instructions modification will invariably take place and hence the M digits of the instruction must be numeric even if they are not needed. For the same reason the B address must always specify a fast register and the M digits of the fast register must also be numeric.

The original description of a Computing Unit instruction word as given in section 2 requires some amplification. The T digit is known as the tracing digit and may be one of the characters 1 through 9, period (.), or ignore symbol (i). If the tracing digit is one of the digits 1 through 9 and the corresponding tracing flip-flop is set, the execution of an instruction is delayed while the computer enters the tracing mode. (This mode is fully explained in section 7.) If the tracing digit is a period (.), the instruction is executed normally. In the examples contained in this section the tracing digit will always be a period. If the tracing digit is ignore (i), the computer operates in the indirect addressing mode. (This mode is more fully explained in section 6.)

The five M digits of an instruction word usually contain the address of a storage location. The addresses range from 00000 to Lim M; Lim M may vary from system to system but never exceeds 97499. The M digits may, however, be used to address a fast register. In this case the address would be of the form 999AA, where AA is a normal fast-register address.

The A and the B digits of an instruction usually specify fast-register addresses. These range from 01 to Lim A; Lim A never exceeds 99. In the Larc system (serials 1 and 2), Lim A is, in fact, less than 99. The references to Lim A in the manual therefore assume this fact. Note, however, that in a Larc system with a full complement of fast registers, certain of the error conditions listed in this manual will not apply.

For example, in a double-precision store instruction (paragraph 3-6), the use of 78 as the A address will cause errors if Lim A is equal to 78, but 99 as the A address will not cause errors if Lim A is equal to 99. The reason for this is that when the A address (99) is incremented by 1 to give the address for the second half of the store, the resulting two-digit address will be 00.

The contents of a fast register may either be interpreted as a normal operand or as a counter and modifier. In the latter case the fast register is referred to as an index register. It should be noted that any fast register may be used as an index register.

3-2. INDEX REGISTER FORMAT

The contents of a fast register used as an index register have the following format:

NNN DDDD ΔΔΔΔΔ

The three N digits are known as a cycle counter. The repertory of the computer contains index-register instructions which reduce the counter of an index register and test it for zero. The five Δ digits are known, collectively, as the address modifier. These are the digits that are used in an instruction to modify the M address. The D digits are used by index register instructions to increment or decrement the address modifier.

There is one special fast register (address 00) in the Larc Computing Unit which may be used to supply an operand consisting of a period and 11

decimal zeros (.00000000000). This operand is permanently stored in the computer and cannot be changed by the programmer.

3-3. OPERANDS

In arithmetic operations the Larc Computing Unit interprets the operands as numbers. Numbers may be stored in the Larc computer in four distinct ways as follows:

(1) Single-precision, fixed-point fractions.

(2) Double-precision, fixed-point fractions.

(3) Single-precision, floating-point numbers.

(4) Double-precision, floating-point numbers.

A single-precision, fixed-point number has the following format:

$$S_{\wedge}XXXXXXXXXXX$$

where S is the sign digit and X represents any decimal digit. The decimal point is automatically taken to be between the sign and the most significant decimal digit. A single-precision, fixed-point operand is thus a signed 11-digit fraction.

A double-precision, fixed-point number consists of two Larc computer words and has the following format:

$$S_{\wedge}XXXXXXXXXXX \quad S \quad XXXXXXXXXXX$$

This number represents a signed, 22-digit fraction. The decimal point is automatically taken to be between the sign and the most significant decimal digit of the left-hand, or most significant word. In double-precision arithmetic instructions the most significant word is addressed; the least significant word is always taken from the next higher storage location or fast register. The signs in both halves of a double-precision operand should agree.

A single-precision, floating-point number in the Larc Computing Unit has the following format:

$$SEE_{\wedge}XXXXXXXXX$$

where S and X have the same interpretation as before and the E digits represent an excess-50 power of ten, or exponent. The decimal point is taken to be between the E digits and the most significant decimal digit. A single-precision, floating-point number thus represents a signed, nine-digit fraction raised to some power of ten (EE - 50). The operand should always be normalized, that is, the most significant X digit should not equal zero. Floating-point results produced by the computer are normalized automatically. The fractional part of a floating-point number will in the future be referred to as the mantissa.

The ranges of a single-precision, floating-point number (N) are as follows:

$$-.999999999 \times 10^{49} \leq N \leq -.1 \times 10^{-50}$$

and

$$.1 \times 10^{-50} \leq N \leq .999999999 \times 10^{49}$$

The format of a double-precision, floating-point number is:

$$SEE_{\wedge}XXXXXXXXX \quad S \; XXXXXXXXXXX$$

This number represents a floating-point number with a mantissa of 20 decimal digits. The decimal point is taken to be in the most significant word immediately following the exponent. The signs of both words should agree and the mantissa should be normalized.

The ranges of a double-precision, floating-point number (N) are as follows:

$$-.99999999999999999999 \times 10^{49} \leq N \leq -.1 \times 10^{-50}$$

and

$$.1 \times 10^{-50} \leq N \leq .99999999999999999999 \times 10^{49}$$

The rules for addressing a double-precision, floating-point number are the same as those for a double-precision, fixed-point number.

The character in the sign position of a number should be one of the following:

(1) Zero (0), indicating that the number is positive.

(2) Minus (-), indicating that the number is negative.

(3) Period (.), indicating a special use of the number. (This is explained further in paragraphs 3-7 and 3-13.)

Variations in the use of the sign digit depend on the type of operation. They will be discussed under the appropriate headings in the instruction repertory.

The format of Larc alphanumeric words is discussed in section 4 in connection with input-output equipment operations.

3-4. FLOATING-POINT ARITHMETIC

The instruction repertory of the Larc Computing Unit contains instructions which carry out arithmetic operations on floating-point numbers. Some general aspects of floating-point arithmetic are reviewed here.

Overflow, as understood in the fixed-point sense, does not occur in floating-point operations. Whenever the mantissa would overflow it is shifted right and the exponent is increased by 1. However, if the exponent becomes greater than its maximum permissible value (99), an exponent overflow contingency will occur. In a similar way, if the result is found to be non-normalized, the computer shifts the mantissa to the left and decreases the exponent accordingly. If the exponent is decreased so that it becomes less than zero, an exponent underflow contingency will occur.

Normally, the result exponent after overflow or underflow has occurred will be meaningless. If the programmer needs to have this information, he may use the following general rule: calculate the result by using the true floating-point representations of the operands (that is, subtract the excess 50 from the exponents), add 50 to the result exponent, and take the ten's complement if the excess-50 result exponent is negative. (For example, the computer representation of an excess-50 exponent of -34 would be 66.) This will give the result exponent.

The following table gives the range of result exponents for all floating-point operations in which an exponent overflow or underflow contingency occurs:

Operation	Exponent Range (overflow)	Exponent Range (underflow)
Add or subtract (single precision)	00	99-91
Add or subtract (double precision)	00	99-80
Multiply (single or double precision)	00-48	99-49
Divide (single or double precision)	00-50	99-51

If the computer uses non-normalized operands in a floating-point computation it is possible for contingencies to occur. Hence, the floating-point representation of zero requires special treatment, as zero cannot be normalized. In the Larc Computing Unit, positive or negative zero will cause contingencies but the computer is designed to handle absolute zero (period and 11 zeros) without causing contingencies. In all cases where a zero is required, the programmer should use absolute zero or a small positive or negative number to avoid contingencies.

If an exponent underflow contingency occurs in a program, control will be transferred to the contingency routine. In designing a contingency routine the programmer must decide what he wishes to do in case of underflow. Underflow is caused by results too small to be in the range of floating-point numbers. The contingency routine must replace the erroneous result by some approximation to the actual value, either a small positive or negative number ($.1 \times 10^{-50}$ or $-.1 \times 10^{-50}$), or absolute zero.

It is possible for a floating-point computation to result in zero. For example, if the two operands below are added, the result will be as shown:

Operands	0 50 347263157
	-50 347263157
Result	0 41 000000000

The initial result in the arithmetic unit would, of course, be:

0 50 000000000

In attempting to normalize, the arithmetic unit would shift the mantissa nine places to the left and reduce the exponent by the number of shifts. In failing to normalize, the arithmetic unit would set the zero floating-point adder result contingency flip-flop and transfer control to the contingency routine. This routine must be designed to replace the zero result by some approximation of the true result. If we assume that, on the average, there is an error of 5 in the tenth place of the mantissa, the approximate result before normalizing would be:

0 50 000000000 5

Therefore an approximate result of:

0 41 500000000

would be the best approximation to replace the zero result. The additional time required by the contingency routine each time a zero result occurs would be a very small percentage of the complete program running time. In special cases, the programmer may wish to use absolute zero to replace a zero produced by addition or subtraction, but in general, the above procedure tends to reduce the accumulation of error terms.

3-5. PROGRAM CONVENTIONS

The following programming conventions are used in the remainder of this manual:

M — The five M digits of an instruction. These digits usually specify a core-storage address or a fast-register address.

A — The two A digits of an instruction. These digits usually specify a fast-register address.

B — The two B digits of an instruction. These digits always specify the address of a fast register which is to be used as an index register.

A_A — The two A digits of the word in fast register A.

A_B — The two B digits of the word in fast register A.

A_{AB} The two A digits and the two B digits of the word in fast register A.

A_M The five M digits of the word in fast register A.

A_I The tracing digit and the two instruction-designator digits (TII) of the word in fast register A.

M_A The same notation is used to denote a portion of the word in storage location M; that is, M_A, M_B, M_{AB}, M_M, and M_I.

.
.
.

' The address of a double precision word stored in two storage locations or two fast registers.

$$A' = A \text{ and } A + 1$$

$$M' = M \text{ and } M + 1$$

( ) The contents of
(M) means the contents of storage location M.

( )i The initial contents of

( )f The final contents of

| | The absolute value or magnitude of
|(M)| means the magnitude of the contents of storage location M.

○ Floating-point operation
(M) ⊕ (A) denotes the floating point addition of (M) and (A).

Rdd Rounded result.

3-6. DATA-TRANSFER INSTRUCTIONS

Instructions in this class transfer words of data from fast storage to memory, from memory to fast storage or from one fast register to another. Data-transfer instructions are of two types: single precision, in which the contents of one location are transferred and double precision, in which the contents of two adjacent locations are transferred. In double-precision transfers the word addressed by the instruction forms one half of the operand and the next higher word in the memory or fast storage forms the other half.

STORE S 40 4μs

T 40 AA BB MMMMM

(A) ⟶ M

Transfer the contents of fast register A to storage location M. The contents of fast register A are not changed.

The instruction will transfer words comprising any combination of legitimate Larc characters.

Storing the contents of fast register 00 is a legitimate operation; it will place in the specified storage location a word consisting of a period in the sign position and 11 decimal zeros. For convenience, this number will in the future be referred to as absolute zero, but the programmer should note that in a few instances the quantity will have no numerical significance.

Words may be stored in a fast register by writing the M address as 999AA, where AA is a fast-register address. For example, the instruction . 40 03 00 99921 will store the contents of fast register 03 in fast register 21. The M address 99900 should not be used in any store instruction because it will cause errors. Fast register 00 is a special-purpose register which cannot be written into.

STORE NEGATIVE SN 41 4μs

T 41 AA BB MMMMM

$-(A) \longrightarrow M$

Transfer the negative value of the contents of fast register A to storage location M. The contents of fast register A are not changed.

Only the sign of the word is altered during the transfer. The alteration of the sign takes place according to the rules expressed in the following table:

Sign of Word in A	0	-	.
Sign of Word in M	-	0	.

If the sign of the operand is a character other than (0), (-), or (.), a sign-anomaly contingency will occur. The store operation will take place and a zero will be deposited in the sign position of M.

The other 11 characters in the word transferred may be any combination of legitimate Larc characters.

STORE MAGNITUDE SM 42 4μs

T 42 AA BB MMMMM

$|(A)| \longrightarrow M$

Transfer the absolute value of the contents of fast register A to storage location M. The contents of fast register A are not changed.

The instruction will transfer words comprising any combination of legitimate Larc characters. Only the sign of the word is altered during the transfer; the sign is always changed to a decimal zero, regardless of its original value.

Note that the instruction . 42 00 BB MMMMM will store a word consisting of 12 decimal zeros in storage location M.

STORE, DOUBLE PRECISION SS 45 8μs

T 45 AA BB MMMMM

(A) ⟶ M

(A + 1) ⟶ M + 1

Transfer the contents of fast register A to storage location M; then, transfer the contents of fast register A + 1 to storage location M + 1. The contents of fast register A and A + 1 are not changed. The contents of the fast register may be any combination of legitimate Larc characters.

As in the single-precision store instructions, words may be stored in fast registers by writing the M address as 999AA, where AA is a fast-register address. For example, the instruction . 45 13 00 99916 will store the contents of fast register 13 in fast register 16, and the contents of fast register 14 in fast register 17.

The double-precision store instruction virtually operates as two single-precision store instructions in sequence. This gives unusual results when an instruction such as . 45 12 00 99913 is executed. In this case, the contents of fast register 12 are stored in fast register 13, and the contents of fast register 13 are then stored in fast register 14. The net effect of the double-precision store instruction is to transfer the contents of fast register 12 to fast registers 13 and 14. In particular, the instruction . 45 00 00 99901 will store absolute zero in fast registers 01 and 02.

There is a special restriction on the use of a double-precision store instruction in the last word position of a memory unit, that is, in storage locations 02499, 04999, etc. If a double-precision store instruction in one of these locations has as its M address the address of the instruction, the next instruction to be executed will be taken from the first location of the next memory unit before the second half of the store takes place. For example, if in location 02499 there is the instruction . 45 06 00 02499, the next instruction will be taken from 02500 before the contents of fast register 07 have been stored there.

This is contrary to the normal case in which such an instruction is held in any location other than the last word position of a memory unit. In this case the next instruction will be the new contents of the second storage location.

It will be seen that the use of Lim A or Lim M as A or M addresses in any double-precision store instruction will cause errors. The second half of the store will then be dealing with nonexistent fast registers or storage locations.

STORE NEGATIVE, DOUBLE PRECISION SSN 46 8μs

T 46 AA BB MMMMM

-(A) ⟶ M

-(A + 1) ⟶ M + 1

Transfer the negative value of the contents of fast register A to storage location M; then, transfer the negative value of the contents of fast register A + 1 to storage location M + 1. The contents of fast registers A and A + 1 are not changed.

The sign of each word transferred is altered independently. The signs are altered according to the rules expressed in the following table:

Sign of Word in A	0	-	.
Sign of Word in M	-	0	.

If the sign of either word is a character other than (0), (-), or (.), a sign-anomaly contingency will occur.

If the signs of both words are incorrect, both halves of the store operation will take place and a 0 will be deposited in the sign positions of M and M + 1. If the sign of the second word (A + 1) only is incorrect, storage of the first word will take place normally and storage of the second word will be carried out with a 0 deposited in the sign position of M + 1. If the sign of the first word (A) only is incorrect and M is the address of a storage location, storage of the first word will take place and a 0 will be deposited in the sign position of M; storage of the second word will take place normally. If the sign of the first word (A) only is incorrect and M is the address of a fast register, the storage of the first word will take place and a 0 will be deposited in the sign position of M; storage of the second word will take place with the following rules for the sign digit established by the rules expressed in the following table:

Sign of Word in A + 1	0	-	.	
Sign of Word in M + 1	0	0		Bad parity combination

The bad parity combination will not cause errors until a further attempt to use fast register M + 1 is made.

The other 11 characters in each word are transferred unchanged, and may be any combination of legitimate Larc characters.

The double-precision, store-negative instruction virtually operates as two single-precision store instructions in sequence. This gives unusual results in an instruction such as . 46 12 00 99913. In this case, the

negative value of the contents of fast register 12 are stored in fast register 13, and the negative value of the contents of fast register 13 is then stored in fast register 14. The net effect of the instruction is to transfer the negative value of the contents of fast register 12 to fast register 13, and the original contents of fast register 12 to fast register 14.

The special restriction on the use of a double-precision store instruction in the last word position of a memory unit applies also to the double-precision, store-negative instruction.

STORE MAGNITUDE, DOUBLE PRECISION SSM 47 8μs

T 47 AA BB MMMMM

$|(A)| \longrightarrow M$

$|(A + 1)| \longrightarrow M + 1$

Transfer the absolute value of the contents of fast register A to storage location M; then, transfer the absolute value of the contents of fast register A + 1 to storage location M + 1. The contents of fast registers A and A + 1 are not changed.

The instruction will transfer words comprising any combination of legitimate Larc characters. Only the signs of the two words are altered during the transfer. These are always changed to decimal zeros regardless of their original values.

The instruction . 47 12 00 99913 will have the net effect of transferring the absolute value of the contents of fast register 12 to fast registers 13 and 14.

The special restriction on the use of a double-precision store instruction in the last word position of a memory unit applies also to the double-precision, store-magnitude instruction. However, it is unlikely that the programmer would use this instruction to store instruction words as the tracing digit of the stored instruction would be zero, causing errors if the instruction were executed.

FETCH F 43 4μs

T 43 AA BB MMMMM

$(M) \longrightarrow A$

Transfer the contents of storage location M to fast register A. The contents of storage location M are not changed.

The instruction will transfer words comprising any combination of legitimate Larc characters.

Words may be fetched from a fast register by writing the M address as 999AA, where AA is a fast-register address. In particular, the instruction . 43 01 00 99900 will have exactly the same effect as the instruction . 40 00 00 99901.

Note that in this instruction the fast-register address 00 is not a valid A address. Errors will occur if any attempt is made to write into this fast register.

FETCH, DOUBLE PRECISION FF 48 8μs

T 48 AA BB MMMMM

(M) ⟶ A

(M + 1) ⟶ A + 1

Transfer the contents of storage location M to fast register A; then, transfer the contents of storage location M + 1 to fast register A + 1. The contents of storage locations M and M + 1 are not changed.

The contents of the two words transferred may be any combinations of legitimate Larc characters.

As in the single-precision fetch instruction, words may be fetched from fast registers by writing the M address as 999AA, where AA is a fast-register address. For example, the instruction . 48 16 00 99902 will transfer the contents of fast register 02 to fast register 16, and transfer the contents of fast register 03 to fast register 17.

The double-precision fetch instruction virtually operates as two single-precision fetch instructions in sequence. This gives unusual results when an instruction such as . 48 17 00 99916 is executed. In this case the contents of fast register 16 are transferred to fast register 17, and then the contents of fast register 17 are transferred to fast register 18. The net effect of the instruction is to transfer the contents of fast register 16 to fast registers 17 and 18. In particular, the instruction . 48 01 00 99900 will transfer absolute zero to fast registers 01 and 02.

It will be seen that the use of Lim A or Lim M as A or M addresses in the double-precision fetch instruction will cause errors. The second half of the store will then be dealing with nonexistent fast registers or storage locations.

3-7. FIXED-POINT ARITHMETIC INSTRUCTIONS

Instructions in this class perform basic arithmetic operations on words representing fixed-point fractions. Single-precision instructions are considered first, followed by double-precision instructions. The mnemonic of every instruction in this class contains an X. Read the X as 'fixed point'.

Characters in the nonsign positions of operands should be numeric only. If one or more of the nonsign positions of an operand contain a non-numeric character, an error will occur and no result will be stored. The M operand may be a fast register.

In any fixed-point addition or subtraction in which the signs of the operands are either (0) or (-), a zero result will take the sign of the A operand.

If either operand in a fixed-point addition or subtraction contains a (.) in the sign-digit position, the absolute value of the result will be the sum of the absolute values of the operands, and the sign of the result will be given by the appropriate sign table.

If a sign-anomaly contingency occurs in any fixed-point arithmetic instruction, the sign of the result will be (0). In a fixed-point addition or subtraction in which a sign-anomaly contingency occurs, the absolute value of the result will be the sum of the absolute values of the operands.

ADD, FIXED POINT AX 01 4μs

T 01 AA BB MMMMM

(M) + (A) ⟶ A

Add the fixed-point fraction in storage location M to the fixed-point fraction in fast register A. Store the sum, with correct sign, in fast register A. The contents of storage location M are not changed.

The sign position in either word may contain any of the characters (0), (-), (.), or one of the digits 1 through 9. The sign of the result is governed by the rules expressed in the following table.

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0/-	0	1 thru 9	C
-	0/-	-	-	1 thru 9	C
.	0	-	.	1 thru 9	C
1 thru 9	1 thru 9	1 thru 9	1 thru 9	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the absolute value of the sum is greater than 1, a fixed-point overflow contingency will occur.

NEGATIVE ADD, FIXED POINT NX 11 4μs

T 11 AA BB MMMMM

-(M) + (A) ⟶ A

Add the negative value of the fixed-point fraction in storage location M to the fixed-point fraction in fast register A and store the sum, with correct sign, in fast register A. The contents of storage location M are not changed. More simply, subtract the fixed-point fraction in storage location M from the fixed-point fraction in fast register A.

The sign position in either word may contain any of the characters (0), (-), (.), or one of the digits 1 through 9. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0/-	0	0	1 thru 9	C
-	-	0/-	-	1 thru 9	C
.	-	0	.	1 thru 9	C
1 thru 9	1 thru 9	1 thru 9	1 thru 9	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the absolute value of the difference is greater than 1, a fixed-point overflow contingency will occur.

MULTIPLY, FIXED-POINT ROUNDED MXR 20 8μs

T 20 AA BB MMMMM

[(M) x (A)] Rdd ⟶ A

Multiply the fixed-point fraction in fast register A by the fixed-point fraction in storage location M; store the rounded single-length product, with correct sign, in fast register A. Rounding is accomplished in the usual way. The most significant digit of the least significant half of the double-length product is examined; if it is greater than or equal to five, 1 is added to the least significant digit of the most significant half of the product. If the digit is less than five, the most significant half of the product is not altered. In either case the most significant half of the product is stored as the final result. The contents of storage location M are not changed.

The sign positions of each operand must contain one of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

No overflow contingency can occur in this instruction as the absolute value of the result is always less than that of either operand.

MULTIPLY, FIXED-POINT EXTENDED MXE 21 12μs

T 21 AA BB MMMMM

(M) x (A) ⟶ A'

Multiply the fixed-point fraction in fast register A by the fixed-point fraction in storage location M; store the result as a double-precision, fixed-point fraction in fast registers A and A + 1. The signs of (A) and (A + 1) in the result are equal. The contents of storage location M are not changed.

The sign position of each operand must contain one of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

No overflow contingency can occur in this instruction as the absolute value of the result is always less than that of either operand.

If the A address of the instruction is equal to Lim A, an error will occur.

DIVIDE, FIXED POINT DX 30 32μs

T 30 AA BB MMMMM

(A) ÷ (M) ⟶ A

Divide the fixed-point fraction in fast register A by the fixed-point fraction in storage location M; store the quotient in fast register A with correct sign. The remainder is not retained and the quotient is not rounded. The contents of storage location M are not changed.

The sign position of each operand must contain one of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

If the absolute value of the dividend is greater than or equal to that of the divisor, then the absolute value of the quotient will be greater than, or equal to, 1. This condition will cause a fixed-point overflow contingency to occur. An attempt to divide by a zero divisor will also result in an overflow.

DIVIDE, FIXED-POINT EXTENDED DXE 31 36μs

T 31 AA BB MMMMM

(A) ÷ (M) ⟶ A

Remainder ⟶ A + 1

Divide the fixed-point fraction in fast register A by the fixed-point fraction in storage location M; store the quotient, with the correct sign, in fast register A. Store the remainder, with the sign of the dividend, in fast register A + 1. The contents of M are not changed.

The sign position of each operand must contain one of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

Overflow may occur in this instruction under the same conditions as in the DX instruction.

If the programmer wishes to continue the division process to obtain a double-precision result, he may divide the remainder by the original

divisor. For example, the pair of instructions . 31 04 00 02300 and . 31 05 00 02300 will store the double-precision quotient of (04) divided by (02300) in fast registers 04 and 05.

If the A address of the instruction is equal to Lim A, an error will occur.

ADD, FIXED-POINT, DOUBLE PRECISION AAX 05 12μs

T 05 AA BB MMMMM

(M') + (A') ⟶ A'

Add the double-precision, fixed-point fraction in storage locations M and M + 1 to the double-precision, fixed-point fraction in fast registers A and A + 1; store the result, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed.

During the calculation, the sign position in the most significant half of each operand is examined. The characters in the other two sign positions are ignored and may be any legitimate Larc characters. The most significant sign positions must contain one of the characters (0), (-), or (.). The signs of both words of the result are equal and are determined by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0/-	0	C	C
-	0/-	-	-	C	C
.	0	-	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the absolute value of the sum is greater than 1, a fixed-point overflow contingency will occur.

The use of Lim A or Lim M as A or M addresses in this instruction will cause errors to occur.

In contrast to double-precision, data-transfer instructions, double-precision arithmetic instructions are true double-precision operations.

NEGATIVE ADD, FIXED-POINT, DOUBLE PRECISION NNX 15 12μs

T 15 AA BB MMMMM

-(M') + (A') ⟶ A'

Add the negative value of the double-precision, fixed-point fraction in storage locations M and M + 1 to the double-precision, fixed-point fraction in fast registers A and A + 1; store the result, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed.

During the calculation, the sign position in the most significant half of each operand is examined. The characters in the other two sign positions are ignored and may be any legitimate Larc characters. The significant sign positions must contain one of the characters (0), (-), or (.). The signs of both words of the result are equal and are determined by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0/-	0	0	C	C
-	-	0/-	-	C	C
.	-	0	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the absolute value of the difference is greater than 1, a fixed-point overflow contingency will occur.

The use of Lim A or Lim M as A or M addresses in this instruction will cause errors to occur.

MULTIPLY, FIXED-POINT, DOUBLE PRECISION MMX 26 36μs

T 26 AA BB MMMMM

(M') x (A') ⟶ A'

Multiply the double-precision, fixed-point fraction in fast registers A and A + 1 by the double-precision, fixed-point fraction in storage locations M and M + 1; store the 22 most-significant digits of the product, with correct sign in both words, in fast registers A and A + 1. The product is not rounded. The contents of storage locations M and M + 1 are not changed.

In contrast to the other double-precision arithmetic instructions, the sign position in the least significant half of each operand is used in the calculation. The characters in the other two sign positions are ignored and may be any legitimate Larc characters. The least-significant sign positions must contain one of the characters (0), (-), or (.). The signs of both words of the result are equal and are determined by the rules expressed in the following table:

Sign of Word in A + 1	Sign of Word in M + 1				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

The overflow contingency cannot occur. The use of Lim A or Lim M as A or M addresses in this instruction will cause errors to occur.

DIVIDE, FIXED-POINT, DOUBLE PRECISION DDX 35 184μs

T 35 AA BB MMMMM

(A') ÷ (M') ⟶ A'

Divide the double-precision, fixed-point fraction in fast registers A and A + 1 by the double-precision, fixed-point fraction in storage locations M and M + 1; store the quotient, with correct sign in both words, in fast registers A and A + 1. The remainder is not retained. The contents of storage locations M and M + 1 are not changed.

During the calculation, the sign in the most significant half of each operand is examined. The characters in the other two sign positions are

ignored and may be any legitimate Larc characters. The most-significant sign positions must contain one of the characters (0), (-), or (.). The signs of both words of the result are equal and are determined by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, △, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, △, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

If the absolute value of the dividend is greater than or equal to that of the divisor, then the absolute value of the quotient will be greater than or equal to 1. This condition will cause a fixed-point overflow contingency to occur. An attempt to divide by a zero divisor will also result in an overflow.

The use of Lim A or Lim M as A or M addresses in this instruction will cause errors to occur.

3-8. UNCONDITIONAL-TRANSFER-OF-CONTROL INSTRUCTIONS

Instructions in this class interrupt the normal sequence of instructions by transferring control to a specified instruction. The address of the specified instruction is held in the M digits of the transfer-of-control instruction. This address must always specify a legitimate storage location. An attempt to transfer control to a fast register will result in errors.

TRANSFER T 90 8μs

T 90 AA BB MMMMM

M ⟶ C

Transfer control to the instruction in storage location M. The A digits of the instruction are not decoded and may be any pair of legitimate Larc characters.

A T instruction need not be followed by a legitimate instruction. Although the instruction immediately following the T instruction is called for by the control unit before the transfer of control actually takes place, the instruction is prevented from entering the instruction registers. Hence, no errors can be caused by nonlegitimate characters in this instruction.

Any unconditional-transfer-of-control instruction can be stored in Lim M without causing errors. Although the control unit would attempt to obtain an instruction from Lim M + 1 and fail to do so, the transfer of control would take place before a stall error had time to occur.

It is necessary to point out here that a single-instruction loop using the T instruction, or a double-instruction loop consisting of a skip (paragraph 3-17) and a T instruction, cannot be used as a short loop to occupy control until Computing-Unit program interruption occurs as a result of intervention signals from the processor or the operator's console. For example:

(02000) = . 90 00 00 02000

or

(02000) = . 00 00 00 00000

(02001) = . 90 00 00 02000

The logic of the Computing Unit has been so arranged that intervention will not occur in such loops. The reason for this arrangement is explained in the notes on the store-last-jump instruction (paragraph 3-10). (The introductory paragraphs of section 4 deal with this problem and its solution.)

TRANSFER RETURN TR 91 12μs

T 91 AA BB MMMMM

9 90 00 00 (C) + 1 $\longrightarrow$ M

M + 1 $\longrightarrow$ C

Store, in location M, a T instruction with a tracing digit 9 and the M address equal to the address of the instruction immediately following the TR instruction; then, transfer control to the instruction in storage location M + 1. For example:

> If (04000) = . 91 00 00 00250, then, after the instruction has been executed, (00250) = 9 90 00 00 04001 and the next instruction to be executed will be taken from storage location 00251.

The A digits of the instruction are not decoded and may be any pair of legitimate Larc characters.

The instruction is intended for use as an entry to a subroutine. In the example given, if a subroutine had its first instruction in storage location 00251, the instruction in location 04000 would cause the computer to

enter the subroutine at its entry point. The subroutine should be so written as to jump to the instruction in 00250 on completion of the calculation. The T instruction in that location would then return the computer to the main program at the instruction following the entry instruction, that is, at the instruction in 04001.

A TR instruction does not have to be followed by a legitimate instruction. No error can be caused in the TR instruction by a succeeding instruction.

It is legitimate to store a TR instruction in Lim M, although it is difficult to find a use for this capability.

The use of Lim M as the M address in this instruction will result in errors when the Computing Unit attempts to transfer control to the instruction in the nonexistent storage location Lim M + 1.

TRANSFER AND STORE B-MODIFIER TB 92 8μs

T 92 AA BB MMMMM

$(C) \longrightarrow A_M$

$M \longrightarrow C$

Store the address of the instruction in the M-digit positions of fast register A (the other digits of fast register A are not changed); then, transfer control to the instruction in storage location M.

The TB instruction is intended to be used in the same way as the TR instruction for subroutine entries. For example, if a subroutine has its first instruction in storage location 00251, and the programmer wishes to enter the subroutine by an instruction in location 04000, this instruction could be . 92 13 00 00251. The last instruction in the subroutine should then be . 90 00 13 00001. This instruction will transfer control to the instruction in location 04001, that is, the instruction immediately following the TB instruction in the main program.

No error will be caused in the TB instruction if it is not followed by a legitimate instruction. Also, a TB instruction may be stored in Lim M without causing errors when it is executed.

An error will be caused if the A address of the instruction is 00. The error will occur when the computer attempts to store the control-counter address in the M-digit positions of fast register 00.

3-9. CONDITIONAL-TRANSFER-OF-CONTROL INSTRUCTIONS

Instructions in this class test one or more fast registers for a particular condition. If the condition exists, control is transferred to the instruction in the location specified by the M address of the transfer-of-control instruction. If the condition does not exist, the next instruction in sequence is processed.

A sign table is given for each instruction but it is worth noting here that, in all instructions of this class, a period in the sign position of an operand has the same effect as a zero.

If the M digits of an instruction in this class specify a fast-register address, an error will occur if the transfer of control takes place; otherwise, no error will occur.

If an instruction in this class is followed by a nonlegitimate instruction, errors will occur if the transfer of control does not take place. If the transfer of control does take place, errors may or may not occur depending on which part of the second instruction word is not legitimate.

If a sign-anomaly contingency occurs in any conditional-transfer-of-control instruction, control will be transferred to the contingency routine before any transfer of control (due to the comparison) can occur.

For convenience in presentation, only a part of the total conditional-transfer-of-control instructions in the Larc Computing Unit repertory are listed in this section. The remaining instructions are described in different paragraphs as follows:

(1) Test Flip-Flop, paragraph 3-16.

(2) B-Modifier Increment and Transfer, paragraph 3-12.

(3) B-Modifier Decrement and Transfer, paragraph 3-12.

(4) B-Modifier Increment and Continue, paragraph 3-12.

(5) B-Modifier Decrement and Continue, paragraph 3-12.

(6) Fetch from Visual-Display Register (5DD), paragraph 3-15.

(7) Fetch from Visual-Display Register (12DD), paragraph 3-15.

(8) Store in Visual-Display Register (5DD), paragraph 3-15.

(9) Store in Visual-Display Register (12DD), paragraph 3-15.

The detailed conditional-transfer-of-control instructions follow.

TRANSFER IF EQUAL TE 70

T 70 AA BB MMMMM

Are (A) = (A + 1) ?

Yes: M $\longrightarrow$ C 12μs

No: (C) + 1 $\longrightarrow$ C 4μs

Compare the contents of fast register A with the contents of fast register A + 1. If they are equal, transfer control to instruction in storage location M; if they are not equal, continue with the next instruction in sequence. The contents of fast registers A and A + 1 are not changed.

The sign position of each operand must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. The other 11 positions in each word must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The rules governing signs in the comparison are expressed in the following table:

Sign of Word in A	Sign of Word in A + 1				
	0	-	.	1 thru 9	i, Δ, +
0	11	NT	11	NT	C
-	NT	11	NT	NT	C
.	11	NT	11	NT	C
1 thru 9	NT	NT	NT	12	C
i, Δ, +	C	C	C	C	C

C = sign-anomaly contingency

NT = no transfer

11 = 11-digit comparison

12 = 12-digit comparison

An important consequence of these rules is that:

. 00000000000 = 0 00000000000

- 00000000000 ≠ 0 00000000000

- 00000000000 ≠ . 00000000000

If the programmer wishes to use this instruction for comparing two instruction words, he must remember that the instruction cannot be used to compare words with an (i) symbol in the sign position. An attempt to do so will result in a contingency.

If the A address of the instruction is equal to Lim A, A + 1 will equal Lim A + 1, a nonexistent fast register. In this case an error will occur.

TRANSFER IF GREATER TG 71

T 71 AA BB MMMMM

Are (A) > (A + 1) ?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

Compare the contents of fast register A with the contents of fast register A + 1. If the contents of fast register A are the greater, transfer control to the instruction in storage location M; if they are not the greater, continue with the next instruction in sequence. The contents of fast registers A and A + 1 are not changed.

The sign position of each operand must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. The other 11 positions in each word must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The rules governing signs in the comparison are expressed in the following table:

Sign of Word in A	Sign of Word in A + 1				
	0	-	.	1 thru 9	i, Δ, +
0	11	T	11	NT	C
-	NT	11	NT	NT	C
.	11	T	11	NT	C
1 thru 9	T	T	T	12	C
i, Δ, +	C	C	C	C	C

C = sign-anomaly contingency

T = transfer

NT = no transfer

11 = 11-digit comparison

12 = 12-digit comparison

An important consequence of these rules is that:

0 00000000000 > - 00000000000

. 00000000000 > - 00000000000

If the programmer wishes to use this instruction for comparing two instruction words, he must remember that the instruction cannot be used to compare words with an (i) in the sign position. An attempt to do so will result in a contingency.

If the A address of the instruction is equal to Lim A, A + 1 will equal Lim A + 1, a nonexistent fast register. In this case an error will occur.

TRANSFER IF ZERO TZ 72

T 72 AA BB MMMMM

Are (A) = 0 ?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

If the contents of fast register A are equal to zero, transfer control to the instruction in storage location M; if they are not equal to zero, continue with the next instruction in sequence. The contents of fast register A are not changed.

The character in the sign position of the operand must be one of the characters (0), (-), or (.). The other 11 positions in the operand must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The rules governing the sign in the comparison are expressed in the following table:

Sign of Word in A	Operation
0	11
-	11
.	11
1 thru 9	C
i, Δ, +	C

C = sign-anomaly contingency

11 = 11-digit comparison with zero

An important consequence of these rules is that:

0 00000000000 = 0

- 00000000000 = 0

. 00000000000 = 0

If the programmer wishes to use this instruction to test an instruction word for zero content, he must remember that the instruction cannot be used to test words with an (i) or one of the digits 1 through 9 in the sign position. An attempt to do so will result in a contingency.

TRANSFER IF GREATER THAN ZERO TGZ 73

T 73 AA BB MMMMM

Are (A) > 0 ?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

If the contents of fast register A are greater than zero, transfer control to the instruction in storage location M; if they are not greater than zero, continue with the next instruction in sequence. The contents of fast register A are not changed.

The sign position of the operand must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. The other 11 positions in the operand must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The rules governing the sign in the comparison are expressed in the following table:

Sign of Word in A	Operation
0	11
-	NT
.	11
1 thru 9	T
i, Λ, +	C

C = sign-anomaly contingency

T = transfer

NT = no transfer

11 = 11-digit comparison with zero

Note that none of the three representations of zero causes a transfer of control.

TRANSFER IF LESS THAN ZERO TLZ 74

T 74 AA BB MMMMM

Are (A) < 0 ?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

If the contents of fast register A are less than zero, transfer control to the instruction in storage location M; if they are not less than zero, continue with the next instruction in sequence. The contents of fast register A are not changed.

The sign position of the operand must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. The other 11 positions in each word must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The rules governing the sign in the comparison are expressed in the following table:

Sign of Word in A	Operation
0	NT
-	T
.	NT
1 thru 9	NT
i, Δ, +	C

C = sign-anomaly contingency

T = transfer

NT = no transfer

An important consequence of the above rules is that:

- 00000000000 < 0

TRANSFER IF EQUAL, DOUBLE PRECISION TTE 75

T 75 AA BB MMMMM

Are (A') = (A + 2') ?

Yes: M ⟶ C 16μs

No: (C) + 1 ⟶ C 8μs

Compare the double-precision word in fast registers A and A + 1 with the double-precision word in fast registers A + 2 and A + 3. If they are equal, transfer control to the instruction in storage location M; if they are not equal, continue with the next instruction in sequence. The contents of fast registers A, A + 1, A + 2, and A + 3 are not changed.

The sign positions of all four words must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. Any other character in any of the four sign positions will cause a sign-anomaly contingency. The nonsign positions must contain decimal digits 0 through 9. Any other character in the nonsign positions will cause an error.

The instruction causes the computer to make a 24-digit comparison between the two double-precision operands. The transfer of control is made only if all 24 digits in one operand are equal to the corresponding digits in the other operand (a period in a sign position being equal to a zero).

If the A address of the instruction is greater than Lim A - 3, errors will occur due to nonexistent fast registers being used in the comparison.

TRANSFER IF GREATER, DOUBLE PRECISION TTG 76

T 76 AA BB MMMMM

Are (A') > (A + 2') ?

Yes: M ⟶ C 16μs

No: (C) + 1 ⟶ C 8μs

More accurately, the operation of the instruction is:

Are (A) > (A + 2) ?

Yes: M ⟶ C

No: Are (A) < (A + 2) ?

Yes: (C) + 1 ⟶ C

No: Are (A + 1) > (A + 3) ?

Yes: M ⟶ C

No: (C) + 1 ⟶ C

If the contents of fast register A are greater than the contents of fast register A + 2, transfer control to the instruction in storage location M; if the contents of fast register A are less than the contents of fast register A + 2, continue with the next instruction in sequence. If the contents of fast register A equal the contents of fast register A + 2, carry out the equivalent of a single-precision TG comparison on the contents of fast registers A + 1 and A + 3. That is, if the contents of fast register A + 1 are greater than the contents of fast register A + 3, transfer control to the instruction in storage location M; if they are not greater, continue with the next instruction in sequence. The contents of fast register A, A + 1, A + 2, and A + 3 are not changed.

The sign position of all four words must contain one of the characters (0), (-), (.), or one of the digits 1 through 9. Any other character in any of the four sign positions will cause a sign-anomaly contingency. The nonsign positions of the four words must contain only decimal digits in the range 0 through 9. Any other character in these positions will cause an error. Even if wrong characters are contained in one or both of (only) the least significant halves and these are not needed in the comparison, the contingency or error will still occur.

The sign rules for each half of the operation are the same as those for the TG instruction.

As a consequence of the mode of operation of this instruction it is possible to obtain a 'greater than' transfer when, in fact, (A') < (A + 2'). For example:

$$(A') = 0\ 35703248771\quad 0\ 73520462198$$

$$(A + 2') = 0\ 35703248771\quad -\ 86344297783$$

In the TTG instruction the computer, on finding the most significant halves equal, will compare the two least-significant halves. In this example, (A + 1) > (A + 3); accordingly, the transfer of control will take place. However, regarded as double-precision operands in all arithmetic instructions in which the sign of the most significant half is the sign of the number, (A + 2') > (A'). The programmer will have no difficulties with this instruction as long as he deals with double-precision numbers in which the signs of both halves are equal.

If the A address of the instruction is greater than Lim A - 3, errors will occur due to nonexistent fast registers being used in the comparison.

3-10. EXTRACT INSTRUCTIONS

Extraction is the process of transferring a group of digits from specified positions in one word to the corresponding positions in another word. Digits other than those in the specified positions are usually unaltered, except in the store-last-jump instruction.

Instructions in this class fall into two groups; one group comprises those instructions in which the digits transferred depend only on the operation code of the particular extract instruction, and the other comprises

those in which the digits to be transferred are specified by another computer word (usually known as a mask).

The M address of the instruction may specify a fast register as in data-transfer instructions.

EXTRACT OPERATION CODE EOP 60 4μs

T 60 AA BB MMMMM

$(M_I) \longrightarrow A_I$

Transfer the tracing digit (T) and the instruction designator digits (II) of the word in storage location M to the corresponding digit positions of fast register A. All other digits in fast register A remain unchanged. The contents of storage location M are not changed.

The contents of storage location M and fast register A may be any combination of legitimate Larc characters.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate M address.

EXTRACT A DIGITS EA 61 4μs

T 61 AA BB MMMMM

$(M_A) \longrightarrow A_A$

Transfer the A digits (AA) of the word in storage location M to the corresponding digit positions of fast register A. All other digits in fast register A remain unchanged. The contents of storage location M are not changed.

The contents of storage location M and fast register A may be any combination of legitimate Larc characters.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate M address.

EXTRACT B DIGITS EB 62 4μs

T 62 AA BB MMMMM

$(M_B) \longrightarrow A_B$

Transfer the B digits (BB) of the word in storage location M to the corresponding digit positions of fast register A. All other digit positions in fast register A remain unchanged. The contents of storage location M are not changed.

The contents of storage location M and fast register A may be any combination of legitimate Larc characters.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate M address.

EXTRACT A AND B DIGITS EAB 63 4μs

T 63 AA BB MMMMM

$(M_{AB}) \longrightarrow A_{AB}$

Transfer the A digits (AA) and the B digits (BB) of the word in storage location M to the corresponding digit positions of fast register A. All other digit positions in fast register A remain unchanged. The contents of storage location M are not changed.

The contents of storage location M and fast register A may be any combination of legitimate Larc characters.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate M address.

EXTRACT M DIGITS EM 64 4μs

T 64 AA BB MMMMM

$(M_M) \longrightarrow A_M$

Transfer the M digits (MMMMM) of the word in storage location M to the corresponding digit positions of fast register A. All other digit positions in fast register A remain unchanged. The contents of storage location M are not changed.

The contents of storage location M and fast register A may be any combination of legitimate Larc characters.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate M address.

EXTRACT LOWER EL 65 8μs

T 65 AA BB MMMMM

$(A-1) \xrightarrow[(M)]{} A$

Transfer digits from fast register A - 1 to the corresponding digit positions of fast register A. Transfer digits only from those positions where there are decimal 1's in the corresponding digit positions of the word in storage location M. All other digits in fast register A remain unchanged. The contents of storage location M and fast register A - 1 are not changed.

The contents of storage location M and fast register A and A - 1 may be any combination of legitimate Larc characters.

A (-) symbol in the sign position of the word in storage location M will also cause extraction.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate, although useless M address.

Example:

(15)i = 333333333333

(16)i = 777777777777

(04000)i = (- or 1) 36117431111

After the instruction . 65 16 00 04000 is executed the contents of fast register 16 will be:

(16)f = 377337773333

EXTRACT UPPER EU 66 8μs

T 66 AA BB MMMMM

$$(A + 1) \xrightarrow[(M)]{} A$$

Transfer digits from fast register A + 1 to the corresponding digit positions of fast register A. Transfer digits only from those positions where there are decimal 1's in the corresponding digit positions of the word in storage location M. All other digits in fast register A remain unchanged. The contents of storage location M and fast register A + 1 are not changed.

The contents of storage location M and fast registers A and A + 1 may be any combination of legitimate Larc characters.

A (-) symbol in the sign position of the word in storage location M will also cause extraction.

If the A address is 00 an error will occur. Storing in fast register 00 is not a legitimate operation; however, 99900 is a legitimate, although useless, M address. Unless Lim A is equal to 99, the use of Lim A as an A address will cause errors. If Lim A is equal to 99, then Lim A + 1 is equal to 00 and the operation is valid.

STORE LAST JUMP SLJ 93 4μs

T 93 AA BB MMMMM

This instruction differs from other extract instructions in that it manufactures a computer word, extracts a storage location address, transfers the address to its M digits, and then stores the complete word.

If M is a storage location address:

9 90 00 00 (C2) ⟶ M

If M is a fast register address:

0 00 00 00 (C2) ⟶ M

If the M address of the instruction specifies a storage location, store in the location: a 90 instruction with an M address equal to the contents of control counter 2; a 9 as tracing digit; and zero A and B addresses.

If the M address of the instruction specifies a fast register, store in the register a word with an M address equal to the contents of control counter 2 (C2), and all other digits equal to zero.

The contents of C2 are not changed.

The two A digits of the SLJ instruction are not used and may be any legitimate Larc characters.

It was explained in section 2 that when a transfer-of-control instruction is decoded, the M address of the instruction is stored initially in C2. If the transfer of control is not effected, the M address remains in C2; if the transfer is effected, the address in C2 is replaced by the address of the instruction immediately following the transfer-of-control instruction. Thus, in general, C2 contains the address of the instruction to which control was not transferred during the execution of the last transfer-of-control instruction. There is an exception to this rule: after the execution of an 80 or 81 instruction which does not transfer control, the contents of C2 will be M + 1, not M.

The SLJ instruction is most useful when a program contains a subroutine which may be entered from various points in the main program. For example:

> A subroutine begins at address 01300 and its exit line is in storage location 01370. A program is written which enters the subroutine by transfer-of-control instructions in storage locations 01500, 01600, and 01700. The instructions in these locations might be various conditional-transfer-of-control instructions, which enter the subroutine when certain conditions are satisfied. The instruction in location 01300 should be . 93 00 00 01370. This instruction, an SLJ instruction, will store a 90 instruction in 01370, the exit line of the subroutine. The instructions stored in location 01370 will be as follows:

Entry point	Instruction in 01370
01500	9 90 00 00 01501
01600	9 90 00 00 01601
01700	9 90 00 00 01701

Thus, each time the subroutine completes its operations, the computer transfers control back to the main program at the instruction immediately following the subroutine entry.

The initial instruction of the subroutine might also be . 93 00 00 99916, in which case the exit line would be . 90 00 16 00000. The effect would be exactly the same as that preceding. In this case, however, fast register 16 must not be altered by the subroutine.

An M address of 99900 will cause an error. Storing in fast register 00 is not a legitimate operation.

The programmer should use the SLJ instruction immediately after the transfer of control has taken place, as intervention can occur at any time, destroying the contents of C2. However, when a transfer-of-control instruction is executed and the transfer of control to the instruction in storage location M takes place, intervention cannot occur until the instruction in location M has been executed. If this is an SLJ instruction the computer is certain to store the required contents of C2.

3-11. SHIFT INSTRUCTIONS

Instructions in this class shift the contents of one or two fast registers either left or right. The number of places to be shifted is specified by the two least-significant M digits of the instruction. In a decimal computer, shifting a number is equivalent to multiplying or dividing the number by ten raised to a power equal to the number of shifts.

In all shift instructions there is no restriction on the characters in the fast registers to be shifted, as long as they are legitimate Larc characters.

Any attempt to shift the contents of fast register 00 will result in errors.

Even though the M digits of a shift instruction do not specify an address, modification of these instructions takes place in the normal way.

SHIFT RIGHT PR 52 4μs

T 52 AA BB MMMMM

$$(A) \times 10^{-M} \longrightarrow A$$

Shift the contents of fast register A to the right by the number of places specified by the two least-significant M digits of the instruction. The sign digit of the word in fast register A is neither shifted nor altered by this instruction. Digits shifted out of the register at the least significant end are lost from the word and decimal zeros are inserted at the most significant end to fill digit positions that are emptied by the shift.

The three most-significant M digits of the instruction are not used and may be any numeric characters.

A shift of 00 places leaves the contents of fast register A unchanged and consequently has the effect of a skip (refer to miscellaneous instructions, paragraph 3-17).

In shift instructions the overflow condition is caused by loss of digits at the most significant end of the word. Because digits cannot be lost at the most significant end of a word in a PR instruction, no overflow condition can occur.

If the number of shifts is greater than, or equal to, 11 all nonsign characters are shifted off and the contents of fast register A, after the shift, will be the sign digit and 11 decimal zeros. No error or contingency will occur.

SHIFT LEFT PL 53 4μs

T 53 AA BB MMMMM

$$(A) \times 10^{M} \longrightarrow A$$

Shift the contents of fast register A to the left by the number of places specified by the two least-significant M digits of the instruction. The sign digit of the word in fast register A is neither shifted nor altered by this instruction. Digits of the word shifted out of the register at the most significant end are lost and decimal zeros are inserted at the least significant end to fill digit positions emptied as a result of the shift.

The three most-significant M digits of the instruction are not used and may be any numeric characters.

A shift of 00 places leaves the contents of fast register A unchanged and therefore has the effect of a skip.

If significant (that is, non-zero) digits are lost at the most significant end of the word, a fixed-point overflow contingency will occur.

If a shift of 11 occurs, the nonsign positions of the word will be filled with decimal zeros and, if the word initially contained any nonzero digits, a fixed-point overflow contingency also occurs.

If the number of shifts is greater than 11, the nonsign positions will be filled with decimal zeros. If significant digits are lost during a left shift of more than 11 places, this may or may not result in an overflow contingency. In certain cases, depending on the number of shifts and the contents of the fast registers, the computer may fail to detect overflow in a shift of more than 11 places.

SHIFT RIGHT, DOUBLE PRECISION PPR 57 8μs

T 57 AA BB MMMMM

$$(A') \times 10^{-M} \longrightarrow A'$$

Shift the double-precision contents of fast registers A and A + 1 to the right by the number of places specified by the two least-significant M digits of the instruction. The sign digits of fast registers A and A + 1 are neither shifted nor altered. Digits shifted out of the least significant end of A enter the most significant end of A + 1. Digits shifted out of the least significant end of fast register A + 1 are lost and decimal zeros are inserted at the most significant end of A to fill digit positions emptied as a result of the shift.

The three most-significant M digits of the instruction are not used and may be any numeric characters.

A shift of 00 places leaves the contents of fast registers A and A + 1 unchanged and has the effect of two skips.

No overflow contingency can occur with the PPR instruction.

If the number of shifts is 22, the nonsign positions of A and A + 1 will be filled with decimal zeros.

If the number of shifts is more than 22, certain anomalous results can occur depending on the value of the least-significant M digit of the instruction. The effects are shown in the following table:

Least-Significant M Digit	Effective Number of Shifts
0,2,3,4,5,7,8,9	22
1,6	21

Example:

If the contents of fast registers 13 and 14 are the following:

(13)i = - 93247832105

(14)i = - 73512005013

and the instruction . 57 13 00 00045 is executed, the final contents of the registers will be as follows:

(13)f = - 00000000000

(14)f = - 00000000000

On the other hand, the instruction . 57 13 00 00056 will produce the following:

(13)f = - 00000000000

(14)f = - 00000000009

If the A address equals Lim A an error will occur since in this case A + 1 = Lim A + 1, the address of a nonexistent fast register.

SHIFT LEFT, DOUBLE PRECISION PPL 58 8μs

T 58 AA BB MMMMM

$$(A') \times 10^{M} \longrightarrow A'$$

Shift the double-precision contents of fast registers A and A + 1 to the left by the number of places specified by the two least-significant M digits of the instruction. The sign digits of fast registers A and A + 1 are neither shifted nor altered. Digits shifted out of the most significant end of A + 1 enter the least significant end of A. Digits shifted out of the most significant end of A are lost and decimal zeros are inserted at the least significant end of A + 1 to fill digit positions emptied by the shift.

The three most-significant M digits of the instruction are not used and may be any numeric characters.

A shift of 00 places leaves the contents of fast registers A and A + 1 unchanged and has the effect of two skips.

If significant, that is, nonzero, digits are lost at the most significant end of the word, a fixed-point overflow contingency will occur.

If the number of shifts is 22, the nonsign positions of A and A + 1 will be filled with decimal zeros. If the word originally contained any nonzero digits an overflow contingency occurs.

If the number of shifts is more than 22, certain anomalous results can occur depending on the value of the least-significant M digit of the instruction. The effects are shown in the following table:

Least-Significant M Digit	Effective Number of Shifts
0,2,3,4,5,7,8,9	22
1,6	21

Example:

If the contents of fast registers 31 and 32 are the following:

(31)i = - 83765257319

(32)i = + 63277885827

and the instruction . 58 31 00 00025 is executed, the final contents of the registers will be as follows:

(31)f = - 00000000000

(32)f = + 00000000000

On the other hand, the instruction . 58 31 00 00031 will produce the following:

$$(31)f = - 70000000000$$

$$(32)f = + 00000000000$$

If significant digits are lost during a double-precision shift left of more than 22 places, this may or may not result in an overflow contingency. In certain cases, depending on the number of shifts and the contents of the fast registers, the computer may fail to detect overflow in a shift of more than 22 places.

If the A address equals Lim A an error will occur since in this case A + 1 = Lim A + 1, the address of a nonexistent fast register.

SHIFT CIRCULAR, DOUBLE PRECISION PPC 59 12μs

T 59 AA BB MMMMM

$$(A') \times 10^{M} \longrightarrow A' \text{ (circular)}$$

Shift the double-precision contents of fast registers A and A + 1 to the left the number of places specified by the two least-significant M digits of the instruction. All 24 digits are shifted. Digits shifted out of the most significant end of A + 1 enter the least significant end of A and digits shifted out of the most significant end of A enter the least significant end of A + 1.

The three most-significant M digits of the instruction are not used and may be any numeric characters.

A shift of 00 places leaves the contents of fast registers A and A + 1 unchanged and has the effect of three skips.

Overflow cannot occur in this instruction since no digits are lost during the shift.

A shift of 24 places is equivalent to a shift of 00 places and leaves the contents of the fast registers unchanged.

An attempt to shift more than 24 places results in errors; no shifting occurs, and the contents of the fast registers are unchanged.

If the A address equals Lim A an error will occur, since in this case A + 1 = Lim A + 1, the address of a nonexistent fast register.

3-12. INDEX-REGISTER-MODIFICATION INSTRUCTIONS

Instructions in this class enable the programmer to alter the modifier (Δ) digits of an index register and also to reduce and test the counter (N) digits. The instructions may themselves be modified in the usual way.

The specified index register should contain only numeric characters. An attempt to alter non-numeric contents of an index register will result in errors.

B-MODIFIER INCREMENT BI 85 4μs

T 85 AA BB MMMMM

$$(A_{\Delta}) + (A_D) \longrightarrow A_{\Delta}$$

Add the D digits of fast register A to the Δ digits of fast register A. Only the five least-significant digits of the fast register are altered. Any carry which extends beyond the Δ part of the fast register is ignored.

The M and the B digits of the instruction are not used. The M digits may be any numeric characters and the B digits may specify any valid fast-register address.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur since the contents of fast register 00 cannot be altered.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

B-MODIFIER DECREMENT BD 86 4μs

T 86 AA BB MMMMM

$$(A_{\Delta}) - (A_D) \longrightarrow A_{\Delta}$$

Subtract the D digits of fast register A from the Δ digits of fast register A. Only the five least-significant digits of the fast register are altered. Any carry which extends beyond the Δ part of the instruction is ignored.

The M and the B digits of the instruction are not used. The M digits may be any numeric characters and the B digits may specify any valid fast-register address.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur since the contents of fast register 00 cannot be altered.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

B-MODIFIER INCREMENT AND TRANSFER BIT 80

T 80 AA BB MMMMM

$$(A_{\Delta}) + (A_D) \longrightarrow A_{\Delta}$$

$$(A_N) - 1 \longrightarrow A_N$$

Are $(A_N) = 0$?

No: M ⟶ C 8μs

Yes: (C) + 1 ⟶ C 12μs

Add the D digits of fast register A to the Δ digits of fast register A, and subtract 1 from the N digits of fast register A. If the N part of A is now not equal to zero, transfer control to the instruction in storage location M; if the N part of A is equal to zero, continue with the next instruction in sequence. The D digits of fast register A are not altered by this instruction. Any carry which extends beyond the Δ or N parts of the fast register is ignored.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur when the computer attempts to subtract 1 from the non-numeric combination .00, the N digits of fast register 00.

If $(A_N) = 000$ initially, they will equal 999 after the execution of the BIT instruction. In this case control will be transferred to the instruction in storage location M.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

When the computer loop is broken by the counter (NNN) reaching zero, the address stored in C2 will be M + 1. It is important to remember this fact if a BIT instruction is followed by an SLJ instruction.

B-MODIFIER DECREMENT AND TRANSFER BDT 81

T 81 AA BB MMMMM

$(A_\Delta) - (A_D) \longrightarrow A_\Delta$

$(A_N) - 1 \longrightarrow A_N$

Are $(A_N) = 0$?

No: M ⟶ C 8μs

Yes: (C) + 1 ⟶ C 12μs

Subtract the D digits of fast register A from the Δ digits of fast register A and subtract 1 from the N digits of fast register A. If the N part of A is now not equal to zero, transfer control to the instruction in storage location M; if the N part of A is equal to zero, continue with the next instruction in sequence. The D digits of fast register A are not altered by this instruction. Any carry which extends beyond the Δ or N parts of the fast register is ignored.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur when the computer attempts to subtract 1 from the non-numeric combination .00, the N digits of fast register 00.

If $(A_N) = 000$ initially, they will equal 999 after the execution of the BDT instruction. In this case, control will be transferred to the instruction in storage location M.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

When the computer loop is broken by the counter (NNN) reaching zero, the address stored in C2 will be M + 1. It is important to remember this fact if a BDT instruction is followed by an SLJ instruction.

B-MODIFIER INCREMENT AND CONTINUE BIC 82

T 82 AA BB MMMMM

$(A_\Delta) + (A_D) \longrightarrow A_\Delta$

$(A_N) - 1 \longrightarrow A_N$

Are $(A_N) = 0$?

Yes: M $\longrightarrow$ C 12μs

No: (C) + 1 $\longrightarrow$ C 4μs

Add the D digits of fast register A to the Δ digits of fast register A and subtract 1 from the N digits of fast register A. If the N part of A is now equal to zero, transfer control to the instruction in storage location M; if the N part of A is not equal to zero, continue with the next instruction in sequence. The D digits of fast register A are not altered by this instruction. Any carry which extends beyond the Δ or N parts of the fast register is ignored.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur when the computer attempts to subtract 1 from the non-numeric combination .00, the N digits of fast register 00.

If $(A_N) = 000$ initially, they will equal 999 after execution of the BIC instruction. In this case, the computer will continue with the next instruction in sequence.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

The addresses stored in C2 by the BIC instruction follow the normal rule.

B-MODIFIER DECREMENT AND CONTINUE BDC 83

T 83 AA BB MMMMM

$(A_\Delta) - (A_D) \longrightarrow A_\Delta$

$(A_N) - 1 \longrightarrow A_N$

Are $(A_N) = 0$?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

Subtract the D digits of fast register A from the Δ digits of fast register A and subtract 1 from the N digits of fast register A. If the N part of A is now equal to zero, transfer control to the instruction in storage location M; if the N part of A is not equal to zero, continue with the next instruction in sequence. The D digits of fast register A are not altered by this instruction. Any carry which extends beyond the Δ or N parts of the fast register is ignored.

If the fast-register address specified by the A digits of the instruction is 00, an error will occur when the computer attempts to subtract 1 from the non-numeric combination .00, the N digits of fast register 00.

If $(A_N) = 000$ initially, they will equal 999 after execution of the BDC instruction. In this case the computer will continue with the next instruction in sequence.

The resulting Δ digits in fast register A need not be a legitimate storage address. Only when the fast register is used as a modifier is it possible for an invalid address to cause an error.

The addresses stored in C2 by the BDC instruction follow the normal rule.

3-13. FLOATING-POINT ARITHMETIC INSTRUCTIONS

Instructions in this class perform basic arithmetic operations on words representing floating-point numbers. Single-precision instructions are presented first, followed by double-precision instructions.

Characters in the nonsign positions of operands should be numeric only. If one or more of the nonsign positions of an operand contain a non-numeric character, an error will occur and no result will be stored.

If the operands in a floating-point operation are normalized and the result is not zero, the result will be a normalized floating-point number. When non-normalized operands are used a normalized or a non-normalized result may be produced depending on the values of the operands and the type of operation.

The M operand in any instruction of the class may be the contents of a fast register.

In single-precision, floating-point addition or subtraction operations in which the signs of the operands are either (0) or (-), a zero result will take the inverse sign of the A operand (0 if the sign of A is -, - if the sign of A is 0).

In double-precision, floating-point addition or subtraction operations in which the signs of the operands are either (0) or (-), a zero result will take the sign of the A operand.

If either operand in a floating-point addition or subtraction contains a period in the sign-digit position, the absolute value of the result will be the sum of the absolute values of the operands, and the sign of the result will be given by the appropriate sign table.

If a sign-anomaly contingency occurs in any floating-point arithmetic instruction, the sign of the result will be a zero. In a floating-point addition or subtraction in which a sign-anomaly contingency occurs, the absolute value of the result will be the sum of the absolute values of the operands. In this case, therefore, the following contingencies cannot occur: improper operand in arithmetic subtraction, zero floating-point adder result, and exponent underflow. These contingencies also cannot occur if either operand in an addition or subtraction contains a period.

ADD, FLOATING POINT A 02 4μs

T 02 AA BB MMMMM

$(M) \oplus (A) \longrightarrow A$

Add the floating-point number in storage location M to the floating-point number in fast register A; store the normalized sum, with correct sign, in fast register A. The contents of storage location M are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0/-	0	C	C
-	0/-	-	-	C	C
.	0	-	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

ADD MAGNITUDE, FLOATING POINT AM 03 4μs

T 03 AA BB MMMMM

$$|(M)| \oplus (A) \longrightarrow A$$

Add the absolute value or magnitude of the floating-point number in storage location M to the floating-point number in fast register A; store the normalized sum, with correct sign, in fast register A. The contents of storage location M are not changed.

The sign of the word in storage location M may be any legitimate character. It is automatically converted to zero before the addition takes place. The sign of the word in fast register A may be any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0	0	0	0
-	0/-	0/-	0/-	0/-	0/-
.	0	0	0	0	0
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of the A operand is a (.), the overflow-contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

ADD UPPER, FLOATING POINT AU 04 4μs

T 04 AA BB MMMMM

(M) ⊕ (A) ⟶ A + 1

Add the floating-point number in storage location M to the floating-point number in fast register A; store the normalized sum, with correct sign, in fast register A + 1. The contents of storage location M and fast register A are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0/-	0	C	C
-	0/-	-	-	C	C
.	0	-	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

If the A address of the instruction is equal to Lim A, the result register will be the nonexistent fast register Lim A + 1, and an error will occur.

NEGATIVE ADD, FLOATING POINT N 12 4μs

T 12 AA BB MMMMM

$-(M) \oplus (A) \longrightarrow A$

Add the negative value of the floating-point number in storage location M to the floating-point number in fast register A; store the normalized sum, with correct sign, in fast register A. The contents of storage location M are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0/-	0	0	C	C
-	-	0/-	-	C	C
.	-	0	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

NEGATIVE ADD UPPER, FLOATING POINT NU 14 4μs

T 14 AA BB MMMMM

$-(M) \oplus (A) \longrightarrow A + 1$

Add the negative value of the floating-point number in storage location M to the floating-point number in fast register A; store the

normalized sum, with correct sign, in fast register A + 1. The contents of storage location M and fast register A are not changed. The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0/-	0	0	C	C
-	-	0/-	-	C	C
.	-	0	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

If the A address of the instruction is equal to Lim A, the result register will be the nonexistent fast register Lim A + 1, and an error will occur.

MULTIPLY ROUNDED, FLOATING POINT MR 22 12μs

T 22 AA BB MMMMM

$[(M) \otimes (A)]$ Rdd $\longrightarrow$ A

Multiply the floating-point number in fast register A by the floating-point number in storage location M; store the rounded normalized product, with correct sign, in fast register A. The mantissa is rounded in the usual way. The most significant digit of the least significant half of the double-length product (after normalizing) is examined. If it is greater than, or equal to, five, 1 is added to the least significant digit of

the most significant half of the product. If it is less than five, the most significant half is not altered. In either case, the most significant half of the product is stored as the mantissa of the final result. The contents of storage location M are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur. However, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

MULTIPLY, FLOATING POINT M 23 8μs

T 23 AA BB MMMMM

$(M) \otimes (A) \longrightarrow A$

Multiply the floating-point number in fast register A by the floating-point number in storage location M; store the truncated normalized product, with correct sign, in fast register A. The contents of storage location M are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

MULTIPLY UPPER, FLOATING POINT MU 24 8μs

T 24 AA BB MMMMM

$(M) \otimes (A) \longrightarrow A + 1$

Multiply the floating-point number in fast register A by the floating-point number in storage location M; store the truncated normalized product, with correct sign, in fast register A + 1. The contents of storage location M and fast register A are not changed.

The sign position in either word may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If the A address of the instruction is equal to Lim A, the result register will be the nonexistent fast register Lim A + 1, and an error will occur.

MULTIPLY EXTENDED, FLOATING POINT ME 25 12μs

T 25 AA BB MMMMM

$(M) \otimes (A) \longrightarrow A'$

Multiply the floating-point number in fast register A by the floating-point number in storage location M; store the result as a normalized, double-precision, floating-point number in fast registers A and A + 1. The signs of (A) and (A + 1) in the result are equal. As the product of the two nine-digit mantissas is an 18-digit mantissa (before normalizing), the computer will fill the two least-significant-digit positions of A + 1 with decimal zeros. The contents of storage location M are not changed.

The sign position in either operand may contain any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If the A address of the instruction is equal to Lim A, one result register will be the nonexistent fast register Lim A + 1, and an error will occur.

DIVIDE ROUNDED, FLOATING POINT DR 32 28μs

T 32 AA BB MMMMM

$[(A) \oslash (M)]$ Rdd ⟶ A

Divide the floating-point number in fast register A by the floating-point number in storage location M; store the rounded normalized quotient, with correct sign, in fast register A. The mantissa of the quotient is rounded in the usual way. The tenth significant digit of the normalized mantissa of the result is examined; if it is greater than or equal to five, 1 is added to the ninth significant digit. If it is less than five, the ninth significant digit is not altered. In either case, the mantissa is truncated to nine digits and stored as the mantissa of the final result. The contents of storage location M are not altered. No remainder is retained.

The sign of each operand may be any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If an attempt is made to divide the dividend by zero or any non-normalized divisor, a non-normalized, floating-point divisor contingency will occur.

DIVIDE UPPER ROUNDED, FLOATING POINT DUR 34 28μs

T 34 AA BB MMMMM

$[(A) \oplus (M)]$ Rdd $\longrightarrow$ A + 1

Divide the floating-point number in fast register A by the floating-point number in storage location M; store the rounded normalized quotient, with correct sign, in fast register A + 1. The mantissa of the quotient is rounded in the usual way. The tenth significant digit of the normalized mantissa of the result is examined; if it is greater than or equal to five, 1 is added to the ninth significant digit. If it is less than five, the ninth significant digit is not altered. In either case, the mantissa is truncated to nine digits and stored as the mantissa of the final result.

The contents of storage location M and fast register A are not altered. No remainder is retained.

The sign of each operand may be any of the characters (0), (-), or (.). The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If an attempt is made to divide the dividend by zero or any non-normalized divisor, a non-normalized, floating-point divisor contingency will occur.

If the A address of the instruction is equal to Lim A, the result register will be the nonexistent fast register Lim A + 1, and an error will occur.

ADD, FLOATING-POINT, DOUBLE PRECISION AA 06 16μs

T 06 AA BB MMMMM

$$(M') \oplus (A') \longrightarrow A'$$

Add the double-precision, floating-point number in storage locations M and M + 1 to the double-precision, floating-point number in fast

registers A and A + 1; store the normalized sum, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed.

Only the signs in the most significant halves of the M and A operands are used in the computation. These signs may be any of the characters (0), (-), or (.). The signs in the least significant halves are ignored and may be any legitimate Larc characters. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	0/-	0	C	C
-	0/-	-	-	C	C
.	0	-	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

If the A address is equal to Lim A or the M address is equal to Lim M or Lim A, an error will occur.

NEGATIVE ADD, FLOATING-POINT, DOUBLE PRECISION NN 16 16μs

T 16 AA BB MMMMM

$-(M') \oplus (A') \longrightarrow A'$

Add the negative value of the double-precision, floating-point number in storage locations M and M + 1 to the double-precision floating-point number in fast registers A and A + 1; store the normalized sum, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed.

Only the signs in the most significant halves of the M and A operands are used in the computation. These signs may be any of the characters (0), (-), or (.). The signs in the least significant halves are ignored and may be any legitimate Larc characters. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0/-	0	0	C	C
-	-	0/-	-	C	C
.	-	0	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur. When the sign of the result can only be determined by the calculation, it is shown as 0/-.

If the result has a zero mantissa, a zero floating-point adder result contingency will occur.

If the exponent of the sum is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the overflow contingency flip-flop will be inhibited. In the event of an exponent overflow, no contingency will occur and the result will be stored with a 00 exponent.

If the A address is equal to Lim A or the M address is equal to Lim M or Lim A, an error will occur.

MULTIPLY, FLOATING-POINT, DOUBLE PRECISION MM 27 36μs

T 27 AA BB MMMMM

$(M') \otimes (A') \longrightarrow A'$

Multiply the double-precision, floating-point number in fast registers A and A + 1 by the double-precision, floating-point number in storage locations M and M + 1; store the normalized truncated double-precision product, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed.

Only the sign in the least significant half of each operand (M + 1, and A + 1) is used in the computation. The signs may be any of the

characters (0), (-), or (.). The signs in the most significant halves are ignored and may be any legitimate Larc characters. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A + 1	Sign of Word in M + 1				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If the A address is equal to Lim A or the M address is equal to Lim M or Lim A, an error will occur.

DIVIDE, FLOATING-POINT, DOUBLE PRECISION DD 36 168μs

T 36 AA BB MMMMM

$(A') \oplus (M') \longrightarrow A'$

Divide the double-precision, floating-point number in fast registers A and A + 1 by the double-precision, floating-point number in storage locations M and M + 1; store the normalized truncated double-precision quotient, with correct sign in both words, in fast registers A and A + 1. The contents of storage locations M and M + 1 are not changed. No remainder is retained.

Only the signs in the most significant halves (M and A) of the operands are used in the computation. The signs in the least significant

halves are ignored and may be any legitimate Larc characters. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If an attempt is made to divide the dividend by zero or any non-normalized divisor, a non-normalized, floating-point divisor contingency will occur.

If the A address is equal to Lim A or the M address is equal to Lim M or Lim A, an error will occur.

DIVIDE BY SINGLE-PRECISION DIVISOR EXTENDED, FLOATING POINT DSE 37 56μs

T 37 AA BB MMMMM

$(A') \oslash (M) \longrightarrow A'$

Divide the double-precision, floating-point number in fast registers A and A + 1 by the single-precision, floating-point number in storage location M; store the normalized, truncated double-precision quotient, with correct sign in both words, in fast registers A and A + 1. The contents of storage location M are not changed. No remainder is retained.

The sign of (M) and the sign of (A) are used in the computation. The sign in (A + 1) is ignored and may be any legitimate Larc character. The sign of the result is governed by the rules expressed in the following table:

Sign of Word in A	Sign of Word in M				
	0	-	.	1 thru 9	i, Δ, +
0	0	-	.	C	C
-	-	0	.	C	C
.	.	.	.	C	C
1 thru 9	C	C	C	C	C
i, Δ, +	C	C	C	C	C

NOTE

The letter C shows that a sign-anomaly contingency will occur.

A zero floating-point adder result contingency cannot occur in this instruction. An incorrect zero result, produced by invalid operands, will not be detected.

If the exponent of the result is greater than 99 or less than 00, an exponent overflow or underflow contingency, respectively, will occur; however, if the sign of either operand is a (.), the result will be stored with a 00 exponent and the overflow and underflow contingency flip-flops will be inhibited. In the event of an exponent overflow or underflow, no contingency will occur.

If an attempt is made to divide the dividend by zero or any non-normalized divisor, a non-normalized floating-point divisor contingency will occur.

If the A address is equal to Lim A an error will occur.

3-14. CONVERSION INSTRUCTIONS

Instructions in this class convert scaled fixed-point numbers to normalized floating-point numbers and the reverse. Usually, these instructions will only be used at the beginning of a program to convert fixed-point input data to floating-point form, or at the end of a program to convert floating-point results to the fixed-point format ready for output.

CONVERT TO FIXED POINT CX 50 4μs

T 50 AA BB MMMMM

(A) - FL ⟶ A - FX

Convert the single-precision, floating-point number in fast register A to a single-precision, fixed-point number; store the converted number in fast register A.

The two least-significant M digits of the instruction specify the excess-50 scale factor to be associated with the fixed-point numbers. The computer subtracts the floating-point exponent from the given scale factor, then shifts the floating-point word two places to the left to shift off the exponent. Decimal zeros are inserted at the least significant end of the word. The fixed-point fraction is then shifted to the right the number of places resulting from the difference between scale factor and exponent. During the shift, decimal zeros are inserted at the most significant end of the word to fill digit positions emptied by the shift. The sign digit does not take part in the shift and is not altered by the conversion.

As an example of conversion to fixed point, consider the instruction . 50 13 00 00056 when the contents of fast register 13 are - 51 682374831. The floating-point number stored in fast register 13 will be - 00000682374.

The three most-significant M digits of the instruction are not used and may be any Larc numeric characters.

There is no restriction on the sign digit of the word to be converted. The sign digit is not changed by the conversion.

A floating-point zero will be converted to a fixed-point zero. In particular, absolute zero will be unchanged by the conversion and behaves as a normal zero in fixed-point arithmetic.

If the floating-point exponent is greater than the fixed-point scale factor, an exponent underflow contingency will occur.

CONVERT TO FLOATING POINT C 51 4μs

T 51 AA BB MMMMM

(A) - FX ⟶ A - FL

Convert the single-precision, fixed-point number in fast register A to a normalized single-precision, floating-point number; store the converted number in fast register A.

The two least-significant M digits of the instruction specify the excess-50 scale factor associated with the fixed-point number. The computer first normalizes the fixed-point fraction by shifting it to the left until all leading zeros have been removed. Decimal zeros are inserted at the least significant end of the word to fill digit positions emptied by the shift. The number of shifts is then subtracted from the scale factor to give the exponent of the floating-point number. The fixed-point

fraction is then shifted two places to the right and the exponent is inserted to give the normalized floating-point number. The sign digit does not take part in the shift and is not altered by the conversion.

As an example of conversion to floating point, consider the instruction . 51 03 00 00047 when the contents of fast register 03 are 0 03765437994.

The contents of fast register 03, together with the scale factor, represent the number + .03765437994 x 10^{-3}. Normalizing this number we get + .37654379940 x 10^{-4}; that is, a shift of one place is required to normalize. Subtracting 1 from the scale factor (47) results in an exponent of 46, which is -4 in excess-50 representation.

The final result, then, is 0 46376543799.

The three most-significant M digits of the instruction are not used and may be any Larc numeric characters.

There is no restriction on the sign digit of the word to be converted. The sign digit is not changed by the conversion.

There is no special provision for the conversion of zero to floating-point format. During conversion, the computer examines only the 11 nonsign positions of the number to be converted, and hence will not distinguish absolute zero from positive or negative zero. The attempted conversion of any fixed-point zero will result in a zero floating-point adder result contingency. Before control is transferred to the contingency routine, a zero floating-point number will be stored in fast register A. The result exponent will be the fixed-point scale factor less 11; the computer will shift the zero 11 places to the left in attempting to normalize.

The program may either test for zero before converting or determine the result required by means of instructions in the contingency routine. Although both these methods require additional instructions, the extra time required is negligible.

If the scale factor of the fixed-point number is less than the number or leading zeros in the number, an exponent underflow contingency will occur. Note that this contingency occurs during the attempted conversion of zero to floating-point format if the scale factor is less than 11.

CONVERT TO FIXED POINT, DOUBLE PRECISION CCX 55 12μs

T 55 AA BB MMMMM

(A') - FL ⟶ A' - FX

Convert the double-precision, floating-point number in fast registers A and A + 1 to a double-precision, fixed-point number; store the converted number in fast registers A and A + 1.

The two least-significant M digits of the instruction specify the excess-50 scale factor to be associated with the fixed-point number. The mode of operation of this instruction is exactly the same as that of the

convert-to-fixed-point instruction. The sign digits of the floating-point number do not take part in the shift and are not altered by the conversion.

The three most-significant M digits of the instruction are not used and may be any Larc numeric characters.

There is no restriction on the sign digits of the word to be converted. These digits are not changed by the conversion.

A double-precision, floating-point zero will be converted to a double-precision, fixed-point zero.

If the floating-point exponent is greater than the fixed-point scale factor, an exponent underflow contingency will occur.

If the A address of the instruction is equal to Lim A, an error will occur.

CONVERT TO FLOATING POINT, DOUBLE PRECISION CC 56 12μs

T 56 AA BB MMMMM

(A') - FX ⟶ A' - FL

Convert the double-precision, fixed-point number in fast registers A and A + 1 to a normalized double-precision, floating-point number; store the converted number in fast registers A and A + 1.

The two least-significant M digits of the instruction specify the excess-50 scale factor associated with the fixed-point number. The mode of operation of this instruction is exactly the same as that of the convert-to-floating-point instruction. The sign digits of the fixed-point number do not take part in the shift and are not altered by the conversion.

There is no special provision for the conversion of zero to floating-point format. During the conversion, the computer examines only the 22 nonsign positions of the number to be converted, and hence will not distinguish absolute zero from positive or negative zero. The attempted conversion of any fixed-point zero results in a zero floating-point adder result contingency. Before control is transferred to the contingency routine, a zero floating-point number will be stored in fast registers A and A + 1. The result exponent will be the fixed-point scale factor less 22; the computer will shift the zero 22 places to the left in attempting to normalize.

The program may either test for zero before converting or determine the result required by means of instructions in the contingency routine. Although both these methods require additional instructions, the extra time required is negligible.

If the scale factor of the fixed-point number is less than the number of leading zeros in the number, an exponent underflow contingency will occur. Note that this contingency occurs during the attempted conversion of zero to floating-point format if the scale factor is less than 22.

If the A address of the instruction is equal to Lim A, an error will occur.

3-15. VISUAL-DISPLAY INSTRUCTIONS

There are two visual-display registers on the Larc Computing Unit operator's console. One is a five-digit display (5DD); the other is a 12-digit display (12DD). The 12-digit display is denoted by a K in the mnemonic of an instruction.

Information may be sent to the displays from a fast register or may be stored in a fast register from the displays by means of Computing Unit instructions.

The logic and uses of the visual-display registers, together with the associated flip-flops (connect and interlock), are explained in section 5 of the manual. In this paragraph, the instructions are given for completeness. The programmer should read the relevant part of section 5 before using any visual-display instruction.

The interlock flip-flop for a visual-display register must be set if the computer is reading from the display and must be reset if it is sending information to the display.

FETCH FROM VISUAL-DISPLAY REGISTER (5DD) FV 09

T 09 AA BB MMMMM

Is the 5DD interlock flip-flop set?

Yes: 0———0 (5DD) ⟶ 02650

(02650) ⟶ A

Reset 5DD interlock and connect flip-flops 4μs

No: M ⟶ C 12μs

Test the interlock flip-flop for the five-digit display register. If the flip-flop is set, transfer the contents of the display to the M-digit positions of storage location 02650, and fill the other digit positions in location 02650 with decimal zeros. Then, transfer the contents of storage location 02650 to fast register A. Also, reset the interlock and connect flip-flops for the five-digit display register. If the interlock flip-flop is initially reset, transfer control to the instruction in storage location M.

Storage location 02650 is used as temporary storage for the display digits before they are transferred to fast register A. When using the visual-display instructions, the programmer must remember that any information he has stored in location 02650 will be changed.

Only the contents of fast register A and storage location 02650 are altered by the FV instruction.

If the modified M address is greater than Lim M, an error will occur only if the interlock flip-flop is reset.

If the A address is greater than Lim A, an error will occur only if the interlock flip-flop is set.

FETCH FROM VISUAL-DISPLAY REGISTER K (12DD) FVK 19

T 19 AA BB MMMMM

Is the 12DD interlock flip-flop set?

Yes: (12DD) ⟶ 02650

(02650) ⟶ A

Reset 12DD interlock and connect flip-flops 4μs

No: M ⟶ C 12μs

Test the interlock flip-flop for the 12-digit display register. If the flip-flop is set, transfer the contents of the display to storage location 02650. Then, transfer the contents of storage location 02650 to fast register A. Also reset the interlock and connect flip-flops for the 12-digit display register. If the interlock flip-flop is initially reset, transfer control to the instruction in storage location M.

Storage location 02650 is used as temporary storage for the display digits before they are transferred to fast register A. When using the visual-display instructions, the programmer must remember that any information he has stored in location 02650 will be changed.

Only the contents of fast register A and storage location 02650 are altered by the FVK instruction.

If the modified M address is greater than Lim M, an error will only occur if the interlock flip-flop is reset.

If the A address is greater than Lim A, an error will occur only if the interlock flip-flop is set.

STORE IN VISUAL-DISPLAY REGISTER (5DD) SV 29

T 29 AA BB MMMMM

Is the 5DD interlock flip-flop reset?

Yes: (A) ⟶ 02650 4μs

(02650_M) ⟶ 5DD 12μs

No: M ⟶ C 12μs

Test the interlock flip-flop for the five-digit display register. If it is reset, transfer the contents of fast register A to storage location 02650. Then, transfer the M digits of storage location 02650 to the

five-digit display register. If the flip-flop is initially set, transfer control to the instruction in storage location M. The contents of fast register A are not changed.

Storage location 02650 is used as temporary storage for the display digits before they are transferred to the display register. When using the visual-display instructions, the programmer must remember that any information he has stored in location 02650 will be changed.

Only the contents of storage location 02650 are altered by the SV instruction.

If the modified M address is greater than Lim M, an error will only occur if the interlock flip-flop is set.

If the A address is greater than Lim A, an error will only occur if the interlock flip-flop is reset.

STORE IN VISUAL-DISPLAY REGISTER K (12DD) SVK 39

T 39 AA BB MMMMM

Is the 12DD interlock flip-flop reset?

Yes: (A) ⟶ 02650

(02650) ⟶ 12DD 4μs

No: M ⟶ C 12μs

Test the interlock flip-flop for the 12-digit display register. If it is reset, transfer the contents of fast register A to storage location 02650. Then, transfer the contents of storage location 02650 to the 12-digit display register. If the flip-flop is initially set, transfer control to the instruction in storage location M. The contents of fast register A are not changed.

Storage location 02650 is used as temporary storage for the display digits before they are transferred to the display register. When using the visual-display instructions, the programmer must remember that any information he has stored in location 02650 will be changed.

Only the contents of storage location 02650 are altered by the SVK instruction.

If the modified M address is greater than Lim M, an error will only occur if the interlock flip-flop is set.

If the A address is greater than Lim A, an error will occur only if the interlock flip-flop is reset.

3-16. FLIP-FLOP INSTRUCTIONS

Flip-flops were introduced in section 2. It was there explained that some are under the complete control of the programmer, some are partially

under his control, and some are outside his control altogether. The last group does not concern the programmer. The flip-flops in the first two groups, known as addressable flip-flops, are listed in appendix B of this manual. The address of a flip-flop comprises two decimal digits; in a flip-flop instruction the A-digit positions are used to hold the flip-flop address.

Twenty-two of the addressable flip-flops are completely under the control of the programmer, that is, they may be tested, reset, and set by the Computing Unit (instructions 95, 96, 97, respectively). Of these, flip-flops 00 through 09 are known as sense flip-flops; they are solely for the programmer's use and have no effect on the computer other than through Computing Unit flip-flop instructions (all other addressable flip-flops may affect the control unit during the execution of other instructions). The programmer is not restricted in his use of the sense flip-flops; there should be no occasion, however, for the programmer to ever set FF84, the cycling-unit error flip-flop. Flip-flop 15 is the manual- and IOP-intervention inhibit flip-flop; its function is described in sections 4 and 5. Flip-flops 21 through 29 are the selected tracing-mode flip-flops and are described in section 7. Finally, flip-flop 90 is the start-tape flip-flop and is described in section 5.

The remaining addressable flip-flops are only partially controlled by the programmer and are grouped as follows:

(1) Input-output flip-flops. (Refer to section 4.)

(2) Enter-tracing-mode flip-flop. (Refer to section 7.)

(3) Error flip-flops. (Refer to section 7.)

(4) Contingency flip-flops. (Refer to section 7.)

A detailed description of the three flip-flop instructions follows.

TEST FLIP-FLOP TF 95

T 95 AA BB MMMMM

Is FF A set?

Yes: M ⟶ C 12μs

No: (C) + 1 ⟶ C 4μs

Test the flip-flop at the address given by the A digits of the instruction. If the flip-flop is set, transfer control to the instruction in storage location M; if the flip-flop is reset, continue with the next instruction in sequence.

An error will occur if the A address specifies a nonexistent flip-flop.

RESET FLIP-FLOP RF 96 4μs

T 96 AA BB MMMMM

Reset FF A

Reset the flip-flop at the address given by the A address of the instruction.

If the addressed flip-flop is initially reset, the instruction will have the net effect of a skip. (Refer to paragraph 3-17 for the effect of a skip.)

The M and B digits of the instruction are not used, but are subject to the general restrictions enumerated in paragraph 3-1.

If an attempt is made to reset a flip-flop not resettable by the programmer (see appendix B) FF10, for example, an error will occur. An error will also occur if the A address specifies a nonexistent flip-flop.

SET FLIP-FLOP SF 97 4μs

T 97 AA BB MMMMM

Set FF A

Set the flip-flop at the address given by the A address of the instruction.

If the addressed flip-flop is initially set, the instruction will have the net effect of a skip.

The M and the B digits of the instruction are not used, but are subject to the general restrictions enumerated in paragraph 3-1.

If an attempt is made to set a flip-flop not settable by the programmer (see appendix B) FF20, for example, an error will occur. An error will also occur if the A address specifies a nonexistent flip-flop.

3-17. MISCELLANEOUS INSTRUCTIONS

SKIP SK 00 4μs

T 00 AA BB MMMMM

(C1) + 1 ⟶ C1

Continue with the next instruction in sequence.

The skip instruction produces no result and does not change the contents of any fast register or storage location. It provides a means of reserving storage locations within a program to allow for expansion during debugging or to allow the program to modify itself by inserting instructions.

The A digits of the instruction are not used and may be any legitimate Larc characters. The M and the B digits also are not used but are subject to the general restrictions enumerated in paragraph 3-1.

HALT H 99

T 99 AA BB MMMMM

Stop

When the computer has stopped due to a halt instruction, it may be restarted by depressing the start pushbutton on the operator's console. The computer will then continue with the instruction following the halt instruction. A processor intervention (refer to section 4) which interrupts a Computing Unit program to transfer control to the contingency routine, will also restart the Computing Unit.

The A digits of the instruction are not used and may be any legitimate Larc characters. The M and B digits also are not used but are subject to the general restrictions enumerated in paragraph 3-1.

Due to the overlap of instructions, the instruction following a halt must have a legitimate tracing digit, (., digits 1 through 9, or i). If the tracing digit is not legitimate, error flip-flop 49 will be set before the computer stops. (The flip-flop is not set if the halt instruction is in storage location Lim M.) Whenever the computer is restarted, either by processor intervention or by the start pushbutton being depressed, the master error flip-flop (98) will be set and control will be transferred to the error routine.

On the other hand, if the tracing digit of the instruction following a halt is legitimate but other digits in the instruction are incorrect, no error flip-flop will be set before the computer stops. If the computer is then restarted by the processor intervention, no error will occur. Of course, if the computer is restarted by depressing the start pushbutton, the computer will continue to execute the instruction following the halt, and errors will result.

If a halt instruction is held in the last storage location, Lim M, no error will occur before the computer stops. As in the case mentioned in the preceding paragraph, if the computer is restarted by processor intervention, no error will occur; if the computer is restarted by depressing the start pushbutton, a stall error will result.

SECTION 4

OPERATIONS OF INPUT-OUTPUT EQUIPMENT

4-1. GENERAL

The input-output operations of the Larc system are under the control of the processor, which is a separately programmed computer. However, the sequence in which operations should be performed by the processor is dependent upon the input-output requirements of the Computing Unit program. If the processor is not used to its fullest capacity in controlling input-output processes, it may be used to edit data, merge, or perform some other side routine. The Larc computer work load can be divided between the processor and the Computing Unit so that each handles the tasks for which it is best suited.

The Computing Unit program must convey its input-output requirements to the processor; in addition, the processor program must have some way to communicate with the Computing Unit so that it can notify the Computing Unit of the completion of requested operations.

Because the core storage is accessible to both the Computing Unit and the processor, it forms the main communication link between the two. Information may be deposited in the core storage by one unit for the use of the other. In addition, a special communication device, the disclosure flip-flop (FF10), connected directly to the processor is available to the Computing Unit. This device can be set and tested by the Computing Unit, and can be reset and tested by the processor. Normally, when the Computing Unit program is ready to transmit orders to the processor, it sets the disclosure flip-flop. The processor program should be designed so that it periodically tests the flip-flop and when it is found set, performs some predetermined procedure. The processor program can reset the disclosure flip-flop to indicate to the Computing Unit program that it has completed some phase of its operation. The Computing Unit program, of course, must test the flip-flop to determine when it has been reset. The exact significance of setting and resetting the disclosure flip-flop is dependent upon the design of the Computing Unit and processor programs.

In addition to the communication possibilities provided by the disclosure flip-flop, the processor has available to it a processor intervention flip-flop (FF11) which enables it to interrupt the computer program.

This flip-flop can be tested and set by the processor, and can be tested and reset by the Computing Unit.

Intervention by the processor can occur at any point in the Computing Unit program, and can be used to notify the program that certain input-output operations are now finished and that the program should go on to some other computation. The processor intervention flip-flop can also be used to indicate such things as a malfunction in the input-output equipment and errors in the input-output data. It can request that the Computing Unit perform certain operations required by the processor but which cannot be performed by it. Multiplication and division functions fall into this last category.

If the processor intervention flip-flop is set, the Computing Unit completes the current instruction; then control transfers to the contingency routine beginning in line 02701. A return jump to the current instruction plus 1 will be recorded automatically in line 02700. The contingency routine must be designed so that it tests the entire series of contingency flip-flop (11,30...34,39...45) to determine which ones have been set. When the set flip-flop is discovered, control is transferred to a subroutine designed to handle the individual contingency. (Additional details on the contingency routine and flip-flops are found in section 7.) In the case of the processor-intervention contingency, the subroutine can either operate on the assumption that each time it is entered the same processor condition has occurred, or it can require the processor to record some indication of what it requires from the Computing Unit.

The programmer may want to include short loops in his program to delay computation until processor intervention indicates that certain input-output procedures have been completed; however, because intervention cannot occur during one-line loops and during two-line loops with an unconditional transfer as the second instruction, they should not be used for this purpose. Any loop of three or more lines may be used as a delay. It is also possible to use a two-line loop which includes a conditional transfer of control. Such a loop could consist of a skip instruction followed by a 72 instruction testing fast-register zero.

Setting the processor intervention flip-flop will not cause any action in the Computing Unit if one of the following conditions exists in the Computing Unit:

(1) The master error flip-flop (FF98) is set.

(2) The Computing Unit is in an unconditional-transfer loop (see section 3).

(3) Flip-Flop 15 is set (see section 5).

(4) The master contingency flip-flop (FF99) is set.

Setting the processor intervention flip-flop will start the Computing Unit if it was stopped before the flip-flop was set. The Unit will start with the master contingency flip-flop set and control in 02701.

Resetting of the processor intervention flip-flop by the Computing Unit can be used to indicate to the processor that the required operation has been completed. Obviously, this communication requires that the processor test the flip-flop to determine when it has been reset.

4-2. DATA CODES

In the Larc computer system there is no change in the data code as data is transferred between input-output devices and core storage. (The magnetic tapes are a special case and are described in paragraph 4-4.) Data can be coded in either a numeric or alphanumeric code.

In the numeric code, a single digit represents one character. In this code, the numerals 0 through 9 and the special symbols (+), (-), (i), (∧), and (.) can be represented. This code is used for data to be operated on arithmetically.

In the alphanumeric code, a pair of digits is required to represent each character. A 12-digit word in this code will, obviously, consist of only six characters. A total of 64 characters can be represented. Decimal characters are distinguishable because the most significant digit of the pair is a 2.

If alphanumeric information is operated on arithmetically, the results will not necessarily consist of meaningful digit combinations; consequently, these characters should be translated into numeric code before they are sent through the arithmetic unit of the computer.

Table 4-1 lists the Larc computer codes and their equivalents for the Unityper* II device, electronic page recorder, and line printer. A blank space in a column indicates that the character is not available in that code or on that device. On the devices which operate in both edited and unedited modes, the functions performed by characters in the edited mode are indicated in parentheses following the character printed in the corresponding unedited mode. (In the unedited mode, all digits in a data word are represented by a printout; in the edited mode some digits may result in an action other than printout of a character.)

4-3. MAGNETIC DRUMS

The magnetic drums constitute both an input-output and a storage device. From the programmer's point of view, however, it is more useful to consider them as input-output equipment because they are handled by the Computing Unit program in the same way as other input-output devices. The drums are directly controlled by the processor program, but the Computing Unit program is indirectly responsible for activating the processor.

* Trademark of the Sperry Rand Corporation.

Table 4-1. Larc Computer Codes

Larc Computer Numeric Code	Larc Computer Alpha-numeric Code	Standard Unityper II Code	Electronic Page Recorder		Line Printer	
			Numeric Mode	Alpha-numeric Mode	Numeric Mode	Alpha-numeric Mode
\	15	i	\ (EOW)	\ (IG)	I (SP)	
^	16	^	^ (SP)	^ (SP)	△ (SP)	W (SP)
-	17	-	-	-	-	-
0	20	0	0	0	0	0
1	21	1	1	1	1	1
2	22	2	2	2	2	2
3	23	3	3	3	3	3
4	24	4	4	4	4	4
5	25	5	5	5	5	5
6	26	6	6	6	6	6
7	27	7	7	7	7	7
8	28	8	8	8	8	8
9	29	9	9	9	9	9
	32	'		=		=
	33	&		~		~
	34	(		(		(
	35	r		ε (EOL)		
	36	,		,		,
.	37	.	.	.	.	.
	40	;		▽		▽
	41	A		A		A
	42	B		B		B
	43	C		C		C
	44	D		D		D
	45	E		E		E
	46	F		F		F
	47	G		G		G
	48	H		H		H
	49	I		I		I
	52	#		<		<
	53	¢		ρ		
	54	@		π		
	55	t		α		
	56	"		]		
	57	|		[		
	60	)		)		)
	61	J		J		J
	62	K		K		K
	63	L		L		L
	64	M		M		M
	65	N		N		N
	66	O		O		O
	67	P		P		P

Table 4-1. Larc Computer Codes (Cont.)

Larc Computer Numeric Code	Larc Computer Alpha-numeric Code	Standard Unityper II Code	Electronic Page Recorder		Line Printer	
			Numeric Mode	Alpha-numeric Mode	Numeric Mode	Alpha-numeric Mode
	68	Q		Q		Q
	69	R		R		R
	72	$		>		>
	73	*		*		*
	74	?		→		
	75	Σ		Σ		
	76	β		η		
	77	:		:		:
+	80	+	+	+	+	+
	81	/		/		/
	82	S		S		S
	83	T		T		T
	84	U		U		U
	85	V		V		V
	86	W		W		W
	87	X		X		X
	88	Y		Y		Y
	89	Z		Z		Z
	92	%		∧		∧
	93	=		λ		
	94			□		

Abbreviations:

EOW - End of Word
SP - Space
IG - Ignore
EOL - End of Line

As many as 24 drum storage units can be included in the Larc system. Each drum has a maximum storage capacity of 250,000 12-digit words. A drum contains 100 circumferential bands, each of which has a storage capacity of 2500 words. Each band is divided into 25 100-word sectors; a sector is the smallest unit that can be read or written during any one reference to a drum. A read or write operation can start only at the beginning of a sector; any sector may be the first one in such an operation.

Six parallel tracks on the drum form a band. The four information bits and the parity check bit are recorded in parallel on five of the tracks. The sixth track is used to store (serially) a band number and an address for each sector. The tracks of one band are interlaced with the tracks of another.

A floating-on-air read-write head assembly is mounted above each drum; program instructions move the unit lengthwise along the drum surface so that the single assembly has access to the entire drum. The head can read or record at a pulse density of 450 pulses per inch. The drum rotates at 880 revolutions per minute, and attains a surface velocity of 1120 inches per second. The six individual read-write heads on the head assembly are spaced at twice the track spacing. Two interlaced bands are read (or recorded) by reading the first with the head in one position, then jogging the head assembly the distance of one track space to read the interlaced band. The head assembly can also be moved (stepped) from one band of an interlaced pair to the equivalent position on an adjacent pair. Movement of the head assembly can be in either direction along the axis of the drum, and can occur in parallel with head assembly movement on another drum. Also, the head movement on one drum can occur in parallel with a read or write operation performed on another drum.

The 100 bands on the drum are numbered in this order: 00,99;01,98;02, 97...47,52;48,51;49,50, where 00 and 99 are the first pair of interlaced bands, 01 and 98 the second pair, and so on. When the head assembly is over a band numbered 00 through 49 inclusive, it is said to be in the low position; when it is over a band numbered 50 through 99, it is in the high position. The most efficient method of processing data on the drum is to process an entire band (using only one instruction), and move consecutively from band 00 to band 49; then, jog the head assembly to band 50, reverse the stepping mechanism, and continue processing consecutively to band 99. This sequence of head movement expends the least time. Of course, it is also possible to use other organization of data; however, efficiency in drum utilization will be sacrificed. Information can be processed in any number of sectors up to a full band, but processing time is increased if the read or write operation must wait for the appropriate sectors to be under the read-write head. If a full band is being processed, there is no latency; however, if only one specific sector is desired, latency time may be as much as the equivalent of a complete drum revolution (68 milliseconds).

Seventy milliseconds are required to step the head assembly from one band of an interlaced pair to an equivalent position on the next pair. Jogging the head assembly from one band to the band with which it is interlaced requires 50 milliseconds, and reversing the direction of stepping requires 10 milliseconds. Note, however, that the jogging and reversing operations can be performed simultaneously.

The head assembly on a drum can be moved to the next band in sequence in less time than it takes to perform a read or write operation on another drum. This means, therefore, that the fastest way to organize data is to read (or write) alternately on two drums. Organizing data in this way means that while a read or write operation is taking place on one drum, the head assembly on another drum is being positioned over the next band.

4-4. MAGNETIC TAPES

Because tape storage references are much more time consuming than drum references, the magnetic tapes are used primarily as long-term storage media. Tapes are used to introduce instructions or data into the Larc system or to record output for long storage or for off-line conversion on an auxiliary device such as the Univac high-speed printer. Intermediate results are normally not written on tapes. Initial data is recorded on tape by a Unityper II device.

The tape system includes both individual tape reels and Uniservo* II tape-transport units. Each Uniservo II tape unit is associated with a synchronizer which controls operation of the unit. A fixed number of units are associated with each synchronizer, but only one of these may be connected to the synchronizer at any one time for reading or writing. However, any one or all other tapes can be rewound while a read or record operation is in progress. The tape synchronizers can perform in parallel with each other and with the other input-output synchronizers in the system.

By changing a plugboard, it is possible to substitute one tape unit for another under control of the same synchronizer. This would be done if one of the units was not able to function correctly.

Both 8- and 10-inch reels can be used on the Uniservo II magnetic tape units connected to the Larc computer. These reels contain, respectively, 1600 and 2400 feet of tape. Data can be recorded on the tape at a density of 104 or 208 characters per inch (cpi). (For the Larc computer, serial 2, the densities are 125 and 250 cpi.) The 104-cpi density is for tapes used on the Univac I system or on Univac off-line auxiliary devices; the 208-cpi density is for tapes used on the Larc or Univac II systems. A wider range of densities can be read by the system; this includes the 20-character-per-inch density of the Unityper I device.

* Trademark of the Sperry Rand Corporation.

The information is arranged on tapes in groups of words termed blocks. A block can be any multiple of 10 words in length; the clear area between each block is termed space between blocks (SBB). The SBB can be either 1 or 2.4 inches long. Normally, the 1-inch SBB is used on the higher density tapes.

The total word capacity of the tape reels is variable, depending on reel size, recording density, number of words per block, spaces between blocks, and bad spots on the tape. The frequency of occurrence and length of the last two items vary with individual tapes. Detection of bad spots on the tape is performed automatically by a photocell on each tape unit. Bad spots, indicated by holes punched through the center of the tape at 2-1/2 inch intervals, are skipped during a tape read or write operation. Tapes may be spliced and the splice joint rendered unusuable by punching properly spaced holes through it.

Data can be written on tape moving in a forward direction; it can be read from tape moving in either a backward or forward direction. The speed with which the tapes move on the Uniservo II tape units is 100-inches per second. Access to the nearest block on tape requires 15 milli-seconds; if the tape is in the rewound state, access to the first block requires an additional 1.8 seconds. Reversal of tape direction adds 0.6 second to the access time. Because the rewind operation is relatively time consuming, the programmer should avoid giving, in close succession, rewind and read or write orders to the same tape units. Normally, data which is output from one portion of a problem and input to another would be read backwards by the second portion of the program.

It is possible either to read from a tape and transfer data to main storage, or merely to move a tape a specified number of blocks to gain access to a particular block. While positioning the tape, the processor can check the information on it for readability. This procedure ensures that the recording was executed without error. A special positioning checker may be used to perform this operation. Use of the checker permits the synchronizer to be used to control both the checking operation and a simultaneous write operation on another tape.

Two types of interlock are available in the tape system. One type is a write-interlock ring which can be inserted into any individual tape reel. The ring prevents the tape from being written on when it is being used on a Uniservo tape unit. Protection of this kind is used on a tape which contains data to be preserved from one problem run to another. The second type of interlock is contained in each tape unit. It can be put to use by the processor during a rewind operation. This interlock prevents the tape from being moved forward from the rewound state until the interlock is manually released by the operator. Interlock of this type is used when the rewound tape contains output data which must be preserved.

All data on tape is in the Univac XS-3, 7-bit code. It must be translated from this code into the codes used by the Larc computer system. The reverse translation must be made when data is written onto tape. These translations are performed by built-in translators in the tape synchronizers. The translation from XS-3 code can be into either the one-digit numeric or two-digit alphanumeric code.

are multiples of ten tape words in length, tape data translated into alphanumeric code will be a multiple of twenty words long in core storage. The characters available in the XS-3 code are listed in table 4-1.

4-5. LINE PRINTER

The on-line, high-speed, line printer is used to produce a printout of the results of computation; intermediate or final results may be printed. It is normally used to print large quantities of data in either single or multiple copies. The printer prints out the 51 printable numeric and alphanumeric characters listed in table 4-1.

The printer speed is 600 lines per minute, with 130 character-positions per line, 10 characters-per-inch horizontally, and 6 lines per inch. Single or multiple line spacing can be used, and the printer accepts paper varying in width from 4-1/2 to 21 inches. The format of the data on the printed page is controlled by the programmer in his choice of data and print mode. The printer can operate in four modes: (1) numeric unedited, (2) numeric edited, (3) alphanumeric unedited, and (4) alphanumeric edited.

In the numeric unedited mode, a single digit is decoded to print a character. Both the ignore (11100) and space (00100) digits are decoded to print as I and Δ, respectively.

In the numeric edited mode all digits except the ignore and space are decoded as they are in the unedited mode. The ignore and space are both decoded to leave a space on the printed page.

In the alphanumeric modes, the two-digit combinations are decoded to determine the character to be printed or the function to be performed. In the alphanumeric unedited mode, a 16-combination is decoded to print a W; in the alphanumeric edited mode, the 16-combination is decoded to leave a space. If the data to be printed in the alphanumeric modes includes a character not available on the printer, a space is left. For a list of the characters printed in the different modes, refer to table 4-1.

In the unedited numeric mode, ten 12-digit words are printed on each line, and a space is left between each word; in the unedited alphanumeric mode, ten 6-digit words are printed on each line, with a space between each word. In the edited modes, the insertion of spaces is dependent upon the presence of space and/or ignore characters in the data being printed.

4-6. ELECTRONIC PAGE RECORDER

The electronic page recorder is used for large volume output, especially in tabular or graphical form. Output may be represented in the form of a curve plot, grid pattern, or alphanumeric characters. Or, output can be a combination of these three: that is, a plotted curve with callouts, titles, scales, grid patterns, etc. The output is displayed on

the face of a cathode ray tube and is recorded by a high-speed, 35-mm camera. For intermediate checking purposes, the display may also be recorded by a Polaroid* Land (self-developing) camera.

The electronic page recorder prints or plots characters at a maximum rate of 20,000 characters per second. Because the cathode ray tube has a single electron beam, only one character is displayed at a time. The displayed characters are registered on film, and after each frame of data has been recorded, the 35-mm camera film is advanced electromechanically. Because the camera has no shutter, the advance is made while the cathode ray tube is cut off. The film can be advanced at a maximum rate of ten frames per second. The camera accepts a 400-foot film magazine. The Polaroid Land camera requires the operator to advance the film manually. This camera produces either standard paper prints or positive transparencies; the paper print can be developed in approximately one minute and a transparency in approximately two minutes. The cameras and the cathode ray tube are enclosed so that the only light striking the film is from the tube.

There are two electronic page recorders (including cameras) with but one synchronizer serving both. Only one recorder can be connected to the synchronizer at a time. If one recorder becomes inoperative or runs out of film, the other recorder can be used to record all the output.

The format of the frame or page displayed on the electronic page recorder is a 1000 by 1000-point mesh. The center of any character being displayed can be directed to any point in the mesh. However, because the area occupied by a character is eight horizontal points by 15 vertical points, a maximum of 67 lines of 125 characters in each line can be displayed for one frame. The origin of the Cartesian coordinate system on the face of the tube is at the lower left-hand corner, and the points in the x and y directions are numbered from 000 to 999. There are 64 different characters (table 4-1) which can be recorded, any of which can be selected for use in curve plotting.

The recorder can operate in several modes and combinations of modes. In the numeric mode, a single digit selects the character to be recorded; in this case the character is either numeric or one of five special symbols. In the alphanumeric mode, two digits select the character to be recorded; this character can be any one of the 64 available. The edited and unedited modes are used when nongraphical data is to be displayed. Either the numeric or alphanumeric mode may be used in conjunction with the edited and unedited modes. In the unedited numeric mode, data is displayed in lines of ten 12-digit words; in the unedited alphanumeric mode, it is displayed in lines of ten 6-digit words. In the unedited modes the spaces between words are smaller than the normal character spacing. In the edited modes, the format is dependent upon the data to be recorded. Specific characters in the data are decoded to indicate a function to be performed rather than a character to be printed. For example, in the numeric edited mode, the ignore digit (11100) indicates the end of a word; no digits beyond it are displayed. The space digit (00100) causes a space

* A registered trademark of the Polaroid Corporation, Cambridge, Mass.

to be left in the display. In the alphanumeric edited mode, a 15-combination neither records a character, performs a function, nor leaves a space; the combination is ignored. A 16-combination causes a space to be left in the display. A 35-combination indicates the end of a line in the display.

There are two plotting modes available. In both, one plotting symbol is selected (alphanumeric), and the coordinates of the points to be plotted are specified by the data words. In the first plotting mode, two sets of x and y coordinates (two points) are specified in a single data word. The format for the data words in this mode is XXXYYYX'X'X'Y'Y'Y'. In the second plotting mode, the x and y coordinates of one point are specified in two consecutive data words in the format as follows:

xxxXXXxxxxxx
xxxYYYxxxxxx

A space symbol in a data word indicates the end of plotting. The symbol must be in the least significant digit of the appropriate section of the word.

The electronic page recorder can operate in two additional modes to plot horizontal and vertical grid lines. In these two modes, the data specifies the points on the coordinate axes through which the lines are to be drawn. In plotting vertical grid lines, two abscissas specifying two full-length lines are contained in each data word. The format for the data words is as follows:

VVV xxx V'V'V'xxx.

In plotting horizontal grid lines, two ordinates specifying two full-length lines are contained in each data word in the following format:

xxxHHHxxxH'H'H'.

As in the other plotting modes, a space symbol in the data signifies the end of plotting. It should be noted that in all modes the electronic page recorder operates on data in groups of ten words each. If less than ten words of data are to be used, an end-of-line (in alphanumeric edited) or end-of-plot (in all plotting modes) symbol should be included in the data after the last valid item. If these symbols are not used or are not available in the mode being used, the data will be filled out to ten words with the information from the locations where valid data is expected to be.

SECTION 5
OPERATING PROCEDURES

This section contains a brief description of the operator's equipment and operating procedures used in the Larc computing system. The purpose of the description is to familiarize the programmer with the equipment used in operating the system, so that he can give adequate instructions to the operator. Detailed operating information is found in the Univac Larc System, Operator's Manual.

5-1. OPERATOR'S STATIONS

There are two identical operator's stations in the Larc system, these are known as the local and remote stations, respectively. The local station, adjacent to the engineer's panel, is normally used only during maintenance periods. The remote station is a separate unit for general use. The two stations are used individually to operate both the Computing Unit and the processor.

Each station consists of an operator's console (figure 5-1), which contains the equipment necessary for transmitting information to the computing system during normal operation. Next to each console is the console printer (Flexowriter*) used for low-speed printout by the system. (The printer also includes a paper tape reader and punch.) Only one operator's station at a time may be used for communicating with the system. The required station is selected by actuating a switch on the engineer's console.

5-2. OPERATOR'S CONSOLE

An operator's console consists of a display panel which can display various types of information, a set of controls for the Computing Unit and the Processor, a console keyboard for typing small amounts of data into the computer, a console printer, and some aural monitoring controls. Those features of the console which particularly concern the Computing Unit programmer are discussed in paragraphs 5-3 to 5-6, following.

* A registered trademark of Friden, Inc., San Leandro, Calif.

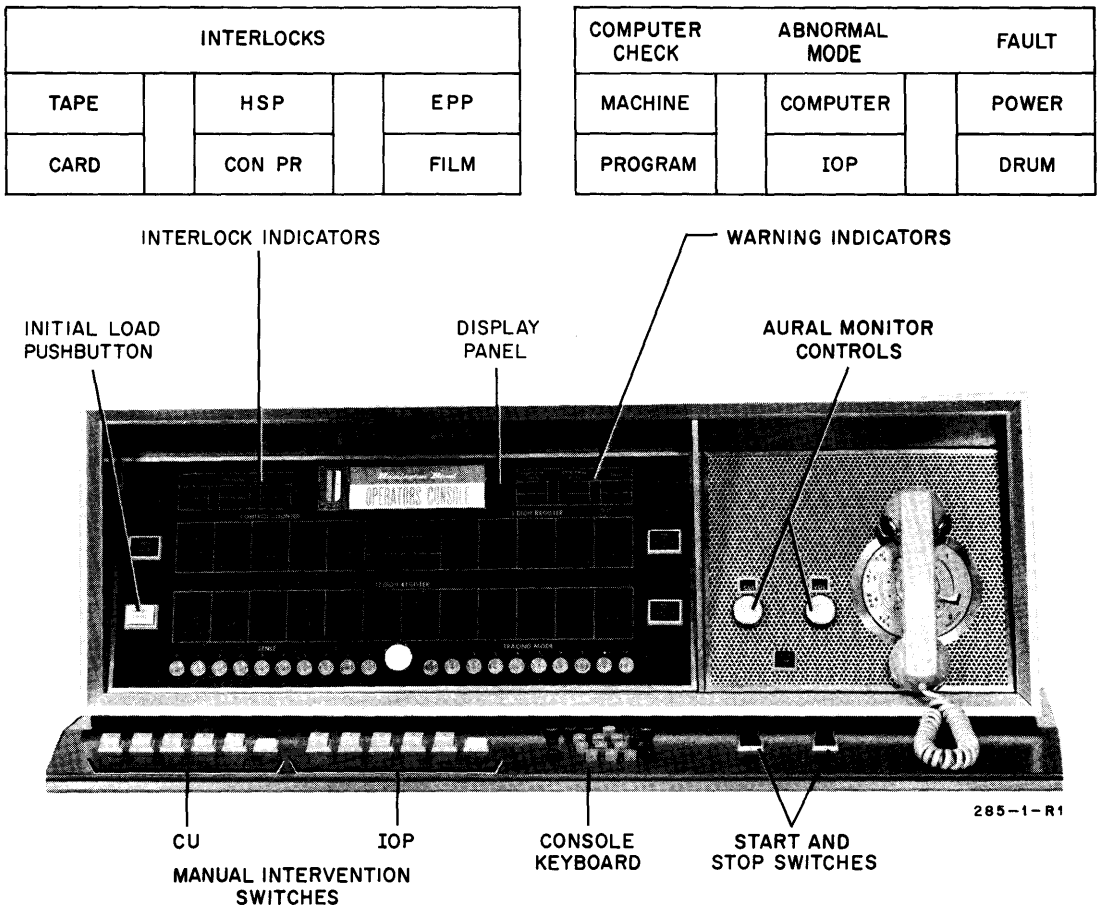

Figure 5-1. Operator's Console

5-3. DISPLAY PANEL. The elements of the display panel present computer information in visual form to the operator for reference. Three ways of displaying visual information are used in the Larc computing system, as follows:

(1) Neon bulbs.

(2) Illuminated indicators.

(3) Backlighted ground-glass indicator panels.

There are 20 neon bulbs on the display panel; ten correspond to the ten sense flip-flops (FF00 through FF09) in the Computing Unit, nine correspond to the selected-tracing-mode flip-flops (FF21 through FF29) also in the Computing Unit, and one is for the manual- and IOP-intervention-inhibit flip-flop (FF15). Each neon bulb lights when the corresponding flip-flop is set.

The illuminated indicators that concern the Computing Unit programmer are those marked CONNECT 5, INTERLOCK 5, CONNECT 12, and INTERLOCK 12. The CONNECT 5 indicator is illuminated when the five-digit-display connect flip-flop is set (that is, when the five-digit display is connected, either to the console keyboard or to the console printer). The INTERLOCK 5 indicator

is illuminated when the five-digit-display interlock flip-flop is set (that is, when the five-digit display is interlocked against use by the computer or the operator). The other two indicators carry out the same functions for the 12-digit display. Further information about the connect and interlock flip-flops is given in paragraphs 5-5 and 5-9.

There are five sets of backlighted indicators. Four of the sets are concerned with input-output equipment and abnormal operation of the system; these are not of particular interest to the programmer. The fifth set consists of three displays of computer information, as follows:

(1) CONTROL COUNTER. This display is actuated only when the Computing Unit stops. It displays the five-digit address of the instruction that was about to be fetched from storage when the computer stopped. This address is in fact the address of the last instruction completed plus 2.

(2) 5-DIGIT REGISTER. The five-digit display shows the contents of the five-digit display register. This register may be used to hold information consisting of five or less Larc characters.

(3) 12-DIGIT REGISTER. The 12-digit display shows the contents of the 12-digit display register. This register may be used to hold information consisting of 12 or less Larc characters.

The 5- and the 12-digit display registers can accept data from either the Computing Unit or the processor. Data can also be sent to the Computing Unit or the processor from the display registers. External information is stored in the display registers, either by means of the console keyboard or by reading punched paper tape.

5-4. CONTROLS. When the STOP pushbutton on the operator's console is depressed, the computer will stop immediately after it finishes executing the instruction which is in its operand-selected cycle. When the computer has stopped, the instruction following the last one completed will be in instruction register 1, and the control counter display will show the address of the instruction which was about to be called.

The START pushbutton will cause the computer to restart from the point at which it was stopped (whether by depression of the STOP pushbutton or as a result of a halt instruction). The address of the next instruction to be executed when the START pushbutton is depressed is normally given by the control counter reading less 1.

There are ten manual-intervention pushbuttons on the console. Five of these are for the Computing Unit and five for the processor. (The latter we are not concerned with in this manual.) Each of the five Computing Unit manual-intervention pushbuttons (0 through 4) sets one of the manual-intervention flip-flops (FF30 through FF34). When one of these flip-flops is set, the master contingency flip-flop (FF99) is set automatically. A transfer to the contingency routine then takes place normally immediately after the current instruction in the Computing Unit has been completed. The contingency routine (section 7) must be so written as to test flip-flops 30 through 34 and transfer control to the corresponding manual-intervention routine when one of them is set. This facility enables the

operator to modify routines during the running of a problem. Manual-intervention flip-flop 30 is also used during paper tape read operations.

Intervention in Computing Unit programs can also be caused by the processor setting the IOP intervention flip-flop (FF11). This flip-flop cannot be set by the Computing Unit or the operator. The IOP intervention has the same effect on the Computing Unit as a manual intervention. The processor will use this facility either when it requires the Computing Unit to carry out some complex arithmetic operations or when it has a need to notify the Computing Unit that certain input-output operations have been completed.

If the programmer requires that a certain part of his program be executed without interruption, he may cause the manual and IOP-intervention-inhibit flip-flop (FF15) to be set. While this flip-flop is in the set state, setting flip-flops 30 through 34 or flip-flop 11 will not set the master contingency flip-flop and no intervention can take place. However, as soon as flip-flop 15 is reset, the master contingency flip-flop will be set automatically if one or more of the intervention flip-flops are set.

The INITIAL LOAD pushbutton on the console is used during the reading of punched paper tape (paragraph 5-9).

5-5. CONSOLE KEYBOARD. The console keyboard is used by the operator to type information directly into the two visual display registers. The keyboard has 15 keys corresponding to the 15 following computer characters: digits 0 through 9; (+), (-), (∧), (.), and (i). Additionally, there are three keys which control keyboard operations, as follows:

(1) C5. Does the following:

(a) Connects the console keyboard to the five-digit display register. (Sets the five-digit-display connect and interlock flip-flops.)

(b) Disconnects the console keyboard from the 12-digit display register. (Resets the 12-digit-display connect and interlock flip-flops.)

Clears the five-digit display to decimal zeros.

(2) C12. Does the following:

(a) Connects the console keyboard to the 12-digit display register. (Sets the 12-digit-display connect and interlock flip-flops.)

(b) Disconnects the console keyboard from the five-digit display register. (Resets the five-digit-display connect and interlock flip-flops.)

Clears the 12-digit display to decimal zeros.

(3) D. Disconnects the console keyboard from the display registers to which it is connected. (Resets whichever connect and interlock flip-flops are set.)

To type into either display register the operator depresses the appropriate connect key (C5 or C12) and types the required number of characters beginning with the most significant. Shifting to the left takes place automatically.

Both the Computing Unit and the processor are prevented from storing any information in a register while it is connected to the keyboard: however, they may read from the register as soon as it is connected. The visual-display-register instructions automatically reset the connect and interlock flip-flops after reading from a display register. The Computing Unit or the processor may store information in a register when it is disconnected from the keyboard.

5-6. CONSOLE PRINTER. The console printer, located at the side of the operator's console, is a low-speed printer controlled by the processor. Normally, the programmer uses the printer to type out information necessary for running the program, namely, operator instructions, program identification, timing, and contingency and error information. Because of its slow operating rate, the console printer is not used as a data input-output device. (It prints at the rate of one character per 100 milliseconds and a carriage return from the end of one line to the start of the next requires one-half to one second.)

The printer accepts two code digits at a time and interprets and converts them into either a single print character or a printer action. The printer operates in either of two positions, upper case or lower case; the characters printed and the printer actions in the two operating positions are listed in table 5-1.

The operator may use the keyboard of the console printer to type headings and other information on the printer page.

The console printer also contains a paper tape reader and punch, described in the following paragraph.

5-7. PAPER TAPE HANDLING

The punched paper tape used by the console printer has eight information channels; five channels are used for data, and three channels are used for control symbols. The numbering of channels across the tape is 5, 4, 3, 2, 1, sprocket hole, 8, 7, 6. The permissible tape symbols are shown in figure 5-2. The data symbols are in standard Larc computer code and require no further explanation. The control symbols are explained in later paragraphs of this section.

5-8. TAPE PREPARATION

Tapes may be prepared on the paper tape equipment when the operator types information on the console printer keyboard; however, only the characters shown in figure 5-2 may be used. Similarly, when the computer (under processor control) is typing out information on the console printer, a punched paper tape record can be produced. The production of tapes in both cases is at the option of the operator, as the console printer will only punch tape when the PUNCH ON pushbutton on the keyboard is depressed.

Table 5-1. Console Printer Characters and Actions

Two-Digit Code	Upper Case	Lower Case	Two-Digit Code	Upper Case	Lower Case
10	UC	UC	46	η	F
11	LC	LC	47	:	G
2-	-	-	48		H
2∧	∧	π	49	2	I
2+	+	+	52		<
2.	.	.	53	\	ρ
2\	\	ρ	54	∧	π
15	IG	IG	55	TAB	TAB
16	SP	SP	56		]
17	-	-	57	α	[
20	0	λ	60	□	)
21	1	U	61	4	J
22	2	I	62	5	K
23	3	O	63	6	L
24	4	J	64	7	M
25	5	K	65		N
26	6	L	66	3	O
27	7	M	67		P
28	8	,	68		Q
29	9	/	69		R
32		=	80	+	+
33	ε	~	81	9	/
34		(	82		S
35	CR	CR	83		T
36	8	,	84	1	U
37	.	.	85		V
40		∇	86		W
41		A	87		X
42	>	B	88		Y
43	*	C	89		Z
44	→	D	92		Δ
45	Σ	E	93	0	λ

Abbreviations:

UC = Upper Case
LC = Lower Case
IG = Ignore
SP = Space
CR = Carriage return
TAB = Tabulate

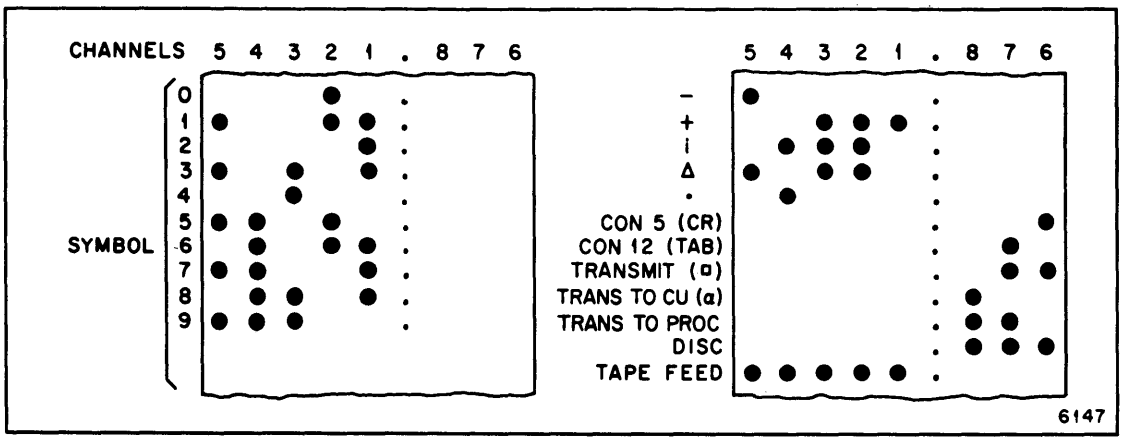

Figure 5-2. Tape Symbols

The punching of data on tape is a straightforward matter. When a key is depressed, the character is printed on the console printer page and the character is simultaneously punched onto paper tape. On the other hand, the punching of control symbols will not always cause printing. Depressing the CR CON 5 key causes the printer carriage to return and the CON 5 control symbol to be printed on the tape. Depressing the TAB CON 12 key causes the carriage to position at a preset marker and the CON 12 control symbol to be punched on the tape. Depressing the TAPE FEED key causes tape to feed through the punch and continue as long as it is depressed and to punch the tape feed control symbol on the tape in every position. Depressing the (□) key caused the symbol (□) to print and punches the transmit control symbol on the tape. Depressing the (α) key prints the symbol (α) and punches the transfer-to-Computing-Unit control symbol on the tape. The other control symbol keys merely cause the correct control symbol to be punched on the tape without causing any printing.

5-9. TAPE READING

Punched paper tapes may be read by the console printer tape reader and used to store information in memory either by means of Computing Unit instructions or by means of the operator's controls. In either case, the visual-display registers are used as buffers between the tape reader and the core storage. The sections of tape shown in figure 5-2 illustrate the way in which data and control symbols are represented on tape. The function carried out by the control symbols (shown in the right half of the figure) when they are read from tape are described below:

CON 5 (CR)

(1) Clears the five-digit display register to zero.

(2) Sets the five-digit-display connect flip-flop.

(3) Sets the five-digit-display interlock flip-flop.

(4) Resets the 12-digit-display connect flip-flop.

(5) Blocks the visual display of the contents of the five-digit display register.

CON 12 (TAB)

(1) Clears the 12-digit display register to zero.

(2) Sets the 12-digit-display connect flip-flop.

(3) Sets the 12-digit-display interlock flip-flop.

(4) Resets the five-digit-display connect flip-flop.

(5) Blocks the visual display of the contents of the 12-digit display register.

TRANSMIT (□)

(1) Stops the tape reader.

(2) Sends the contents of the 12-digit display register to the storage location whose address is in the five-digit display register.

(3) Resets the 5- and 12-digit-display connect and interlock flip-flops.

(4) Start the tape reader.

TRANS TO CU (α)

(1) Stops the reader.

(2) Resets the initial load flip-flop.

(3) Resets the completed stop flip-flop. This flip-flop is set when the Computing Unit stops. When it is reset the Computing Unit is allowed to start.

(4) By setting manual-intervention flip-flop 30, transfers control in the Computing Unit to the instruction in location 02701 (contingency routine).

TRANS TO PROC (processor)

(1) Stops the reader.

(2) Resets the initial load flip-flop.

(3) Transfers processor control to the instruction in location 00001.

DISC (disconnect)

Resets the 5- and 12-digit-display connect and interlock flip-flops.

TAPE FEED

Is ignored by the tape reader. (Tape feed symbols may be punched anywhere in a tape; punching a number of tape feed symbols at the start of a tape is useful in providing a length of tape to aid in positioning tape in the reader.)

Setting the connect flip-flops by paper tape control causes the relevant display register to be connected to the paper tape reader (refer to paragraph 5-5 to compare the action of the console keyboard connect keys).

There are two methods of reading punched paper tape: one method is to press the INITIAL LOAD pushbutton on the operator's console (initial load control); the other is by programming Computing Unit instructions to read tape (program control). Punched paper tape is, in fact, the only input medium directly available to the Computing Unit. The format of a tape depends upon which method is used to read it.

Only one format is permissible for a tape which is to be read under initial load control, as follows: a CON 5 control symbol followed by a five-digit, core-storage address; a CON 12 control symbol followed by a 12-character computer word; and finally a transmit symbol. (The first two groups of characters may be interchanged if required.) This arrangement is repeated for each word to be stored in memory. After the final transmit symbol a transfer-to-Computing Unit or a transfer-to-processor symbol must be punched.

The way in which a tape is read under initial load control is described in the next paragraph. As the name suggests, the method is used to load an initial program in the computer. (Refer to paragraph 5-10 for program load procedures.) The initial load procedure can only be used if the Computing Unit is stopped. Otherwise, the INITIAL LOAD pushbutton on the operator's console will have no effect.

When the tape has been positioned in the reader, the INITIAL LOAD pushbutton is depressed. This starts the tape feeding through the reader. The first length of tape will contain tape-feed symbols which are ignored by the tape reader. The first significant character on the tape must be a CON 5 symbol which connects the reader to the five-digit display register. The next five Larc characters (a computer address) are then read and stored in the five-digit display register. After the address, the next symbol on the tape must be a CON 12 symbol which disconnects the five-digit display and connects the 12-digit display. The next 12 Larc characters are then read and stored in the 12-digit display register. A transmit symbol on the tape then stops the tape and causes the contents of the 12-digit display register to be stored in the location whose address is in the five-digit display register. If the transfer of information from display to storage is satisfactory, the reader restarts and the cycle recommences. An error in the transfer will result in the setting of error flip-flop 52.

A transfer-to-Computing-Unit symbol or a transfer-to-processor symbol must follow the last transmit symbol. Either of these symbols stops the reader. The transfer-to-Computing Unit symbol, by setting manual-intervention flip-flop 30, causes the master contingency flip-flop (FF99) to set, resulting in an automatic transfer of control in the Computing Unit to the instruction in location 02701 (contingency routine). Control is

transferred immediately to the contingency routine. Instructions which were in the control unit when the computer stopped will not be completed (see note on page 7-3). The transfer-to-processor symbol, by setting the IOP error flip-flop, causes control of the processor to automatically transfer to the instruction in location 00001.

The format of a program-control tape may be more varied than the foregoing; however, each entry on the tape must have one of the following formats:

C5 X X X X X

C12 X X X X X X X X X X X X

C5 X X X X X C12 X X X X X X X X X X X X

in which each X may be any Larc character. The transfer-to-processor symbol may be used in place of the transfer-to-Computing Unit symbol (α).

Before the Computing Unit can read tape the 5- and 12-digit-display interlock flip-flops must both be reset. Reading starts when the Computing Unit, utilizing a set-flip-flop instruction, sets the start-tape flip-flop (FF90). If no tape is in the reader when the flip-flop is set, nothing happens until the tape is placed in position and the operator depresses the START READING pushbutton on the console printer. If either the 5- or 12-digit-display interlock flip-flop is set when the Computing Unit attempts to set flip-flop 90, the control error flip-flop (FF47) will be set instead of flip-flop 90 and control will be transferred to the error routine.

Assuming tape of the correct format in the reader, both interlock flip-flops reset, and flip-flop 90 set (by the Computing Unit), the tape will feed automatically through the reader while the Computing Unit continues with its sequence of instructions. Data from the tape will be stored in the 5- or 12-digit display registers in the same way as data from an initial load-control tape. The transfer-control symbol following the data will stop the reader, and, by setting manual-intervention flip-flop 30, will interrupt the Computing Unit program and transfer control to the contingency routine. The interruption will take place in the same way as a contingency (section 7). Note that the transmit symbol must not be used in a program-control tape. Use of this symbol sets the improper-tape error flip-flop (FF38). Storage of the data from the display registers must be carried out either by Computing Unit or processor instructions. The transfer of control executed by the transfer symbol of the tape should be to a routine which will store the information which is held in the display registers.

As the transfer symbol stops the tape reader, the Computing Unit must set flip-flop 90 again to read-in further data; however, flip-flop 30 must first be reset. It is not necessary that flip-flop 90 be reset. If the Computing Unit attempts to set flip-flop 90, the tape reader will start whether flip-flop 90 was originally reset or not.

To summarize, the Computing Unit reads tape by executing an instruction to set flip-flop 90. Information will then be read from tape into

the display registers without further Computing Unit operations being required. When the information has been stored in the display registers a transfer-control symbol on tape will stop the reader and interrupt either the Computing Unit or the processor. The programmer must arrange that the routine to which control is transferred will make use of the contents of the display registers.

Whenever the Computing Unit is about to read paper tape it should cause some identifying data to be typed out on the console printer together with a relative time. This will give a permanent record of the read operation and will notify the operator that a paper tape is needed.

Note that if the Computing Unit is operating in the program-display, manual-display, or error-display mode, flip-flop 47 will be set whenever flip-flop 90 is set. The net effect will be to transfer control to the error routine. This preserves the information in the display registers by preventing the reading of tape. The three display modes are used for engineering purposes and will not usually concern the programmer.

5-10. PROGRAM LOAD PROCEDURES

There are three basic load procedures in the Larc computing system for reading programs into the core storage. They are described briefly below.

5-11. LOAD PROCEDURE 1

Using the initial load procedure, the operator can read a short program from paper tape into core storage. The last symbol on the tape should be a transfer-to-Computing-Unit symbol. This will transfer control to the short program, which may read in additional sections of program from magnetic tape or drum file. This method is seldom used.

5-12. LOAD PROCEDURE 2

The following method may be used for the normal loading of programs during a working day. In this case it is assumed that a processor program and contingency and manual-intervention routines are already in the core storage. The operator types into one of the visual-display registers a code word indicating which Uniservo magnetic tape unit the new program tape is mounted on. He then presses a manual-intervention pushbutton which transfers control in the Computing Unit to the selected manual-intervention routine. This routine should be organized to read the code word from the visual-display register and use it to instruct the processor to read-in the program.

5-13. LOAD PROCEDURE 3

When the core storage is completely empty and it is required to read a processor program into core storage, the processor initial-read hardware can be used. This hardware is built into synchronizer 1 of the processor

and uses Uniservo tape unit 10. Before starting the operation, various controls on the engineer's console must be set to allow the initial read to take place. The program tape is then mounted on Uniservo tape unit 10. When the INITIAL READ and START pushbutton on the engineer's console are depressed simultaneously, a block of data will be read from the tape into consecutive storage locations beginning with location 00000. This block can be of any length but must be a multiple of ten words. At the end of the block an automatic transfer of control to the processor instruction in location 00001 takes place. This load procedure is normally used only for engineering purposes.

SECTION 6
INDIRECT ADDRESSING

Indirect addressing is a method of indicating the effective M address of an instruction in a location other than that of the original instruction. This provides the programmer with additional flexibility and convenience in certain programming applications.

The indirect-addressing provision can be so used that it supplies the same programming convenience as an unlimited number of index registers. Indirect addressing is particularly useful if the determination of the correct operand for an instruction depends on a variety of conditions. This usefulness is increased further if the determinations of the conditions are made at different times.

When the programmer wishes to enter the indirect-addressing mode, he places an ignore symbol (i) in the sign position of the instruction which is to use this mode. Detection of this digit by the computer inhibits the normal decoding and execution of the instruction, and initiates a process to determine the appropriate address to be used in the instruction. When the appropriate address has been selected, the instruction is executed with this address as the operand address. The programmer provides the series of possible addresses and the indication as to which of the series is to be chosen.

Each address named in the series contains the next two addresses in the chain. Also included is an index-register address. The form for the contents of each location is LLLLL BB RRRRR, where the L's and R's represent storage addresses and BB signifies an index-register address. Suppose, for example, 00220 is an address in the series. Its contents might be 00335 16 04445, where 00335 and 04445 are other addresses in the series, and 16 is an index-register address.

At each level in the process, only one of the two addresses named is actually selected to continue the sequence. Further flexibility is provided by the ability to select or reject index-register modification. The contents of the address finally selected as the operand address for the original instruction may be any legitimate operand for that instruction.

The path followed through the indirect-address chain may be varied each time the series of addresses is used. For each use the programmer must provide directions for the path to be followed through the chain. The proper address selections and index-register options are indicated by a series of key digits. The key digits are examined sequentially; the value of each one indicates the following:

(1) Whether or not the previously selected address is to be modified by an index register.

(2) Which portion of the contents of the resultant address is the next address in the series.

The value of the key digit also indicates when the selected address is to be used as the final operand address, thus ending the indirect-address chain.

The original instruction, the key-digit series, and the beginning of the address chain are set up in the following form:

Original instruction:	$iIIA_0A_0B_0B_0M_0M_0M_0M_0M_0$
(M_0 modified by the contents of B_0)	$K_1K_2K_3K_4K_5B_1B_1M_1M_1M_1M_1M_1$
(M_1)	$L_1L_1L_1L_1L_1B_2B_2R_1R_1R_1R_1R_1$
(L_1)	$L_2L_2L_2L_2L_2B_3B_3R_2R_2R_2R_2R_2$
(R_1)	$L_3L_3L_3L_3L_3B_4B_4R_3R_3R_3R_3R_3$

where M_0 is modified by the contents of B_0 (note that this modification is always carried out) contains 5 key digits, K_1 through K_5, and the first address, M_1, and index register, B_1, specification in the series. M_1 is modified by (B_1) if K_1 indicates modification. The contents of M_1 (or M_1 modified) are the next possible addresses in the series; K_1 indicates which of these is selected. K_2 is then picked to determine whether L_1 or R_1 is to be modified by B2; K_2 also selects either the left or right portion of L_1 (or L_1 modified) or R_1 (or R_1 modified). The sequence continues in this way with L_2, R_2, L_3, and R_3, each specifying two more storage addresses and an index-register address. Only the final address in the series (that used as an operand address) may be a fast-register address.

If more than five key digits are required to determine the operand address, $M_0 + 1$ is used for additional digits. These will be examined in sequence after digits K_1 through K_5 have been examined. Similarly, if more than 17 key digits are required, $M_0 + 2$ is used for additional key digits. These are used after those in $M_0 + 1$ have been used up. This process continues until the final operand address is selected.

The key digit functions in the following manner:

(1) If K is 5 or greater than 5, the address selected by the preceding key is modified by the associated B box; if the key is less than 5, no B-box modification occurs. In either case, the word indicated by the address is called from storage.

(2) If the key digit is 0 or 5, the left-hand address is selected as the next address in the chain; if the key digit is 1 or 6, the right-hand address is selected. In these cases, the next key digit is picked up and the process is repeated. However, if the key is 4 or 9, the address selected by the previous key is to be used as the operand address in the original instruction. If $K_1 = 4$ or 9, then M_1 with the appropriate index register modification is the operand address. Keys of 2, 3, 7, 8 should not be used, as they may cause errors. The possibility of errors occurring depends on the combination of the current key and the preceding key.

Figure 6-1 illustrates the steps in the process of address selection in the indirect-addressing mode.

The following examples illustrate the mechanism of indirect addressing. The circled numbers on the left correspond to points on the flow chart, figure 6-1.

The contents of these index registers apply to all 3 examples:

(17) = 000000000007
(18) = 000000000008
(19) = 000000000009
(20) = 000000000010
(22) = 000000000012
(23) = 000000000013

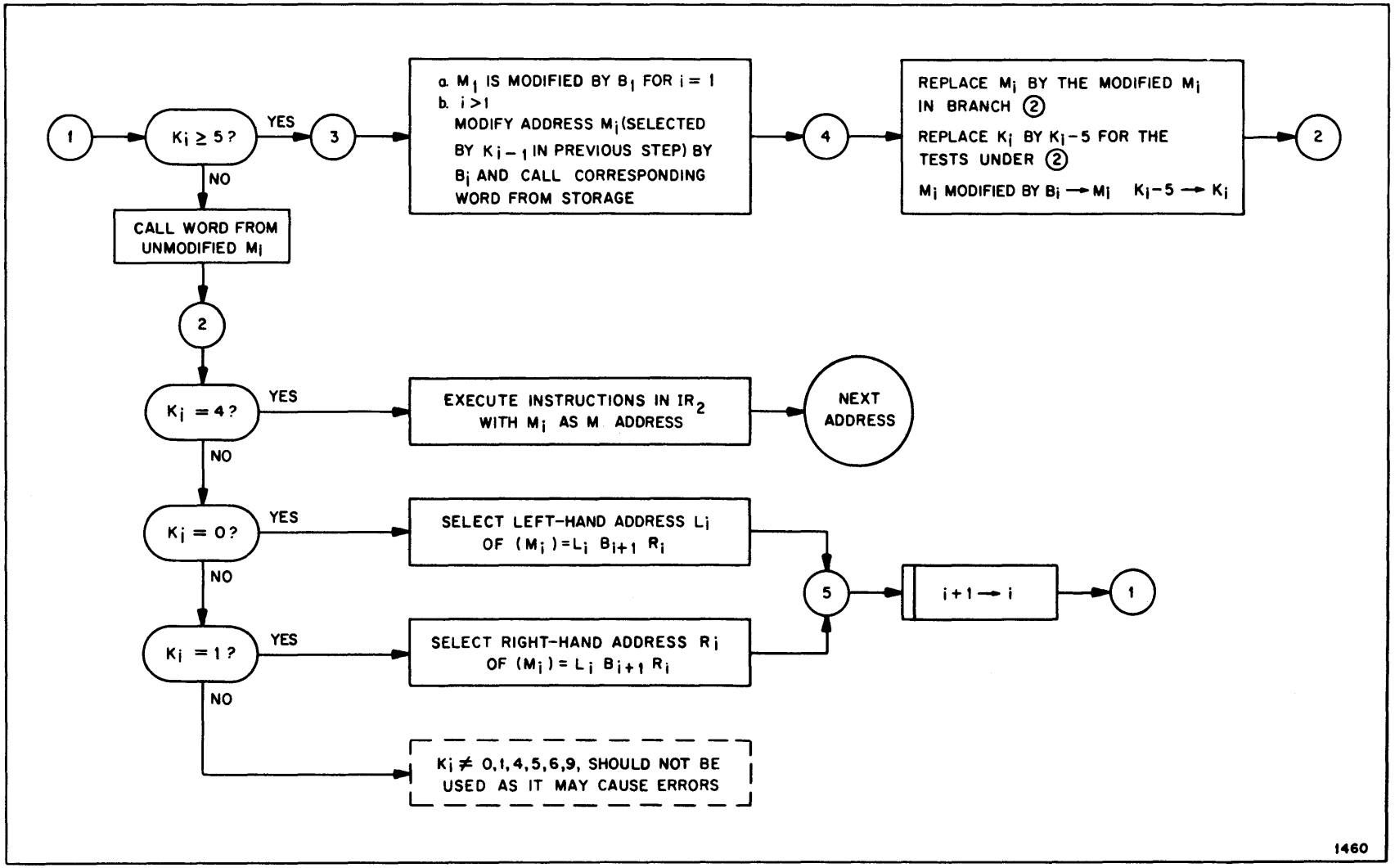

Figure 6-1. Address Selection in Indirect Addressing

Example 1

Comments

(5009) = i 01 05 00 07500

(7500) = 50040 17 07500

i = 1	①		$K_1 = 5$, $M_1 = 7500$	$B_1 = 17$; modification to occur
	③	(a)	$(B_1) = (17) = 000000000007$	$M_1 = 7500$
	④		$M_1 + (B_1) = 7507 \longrightarrow M_1$	M_1 is modified by B_1 and becomes $M_1 = 7507$
			$0 \longrightarrow K_1$	$K_1 - 5 \longrightarrow K_1$
	②		$K_1 = 0$, $(M_1) = 07501\ 18\ 07502$	$B_2 = 18$
			$M_2 = 7501$ is selected	$L_1 = 7501$
	⑤		$2 \longrightarrow i$	
i = 2	①		$K_2 = 0$, $M_2 = 7501$	
	②		$(M_2) = (7501) = 07503\ 19\ 07504$	$B_3 = 19$
			$M_3 = 7503$ is selected	$L_2 = 7503$
	⑤		$3 \longrightarrow i$	
i = 3	①		$K_3 = 0$, $M_3 = 7503$	
	②		$(M_3) = (7503) = 07505\ 20\ 07506$	$B_4 = 20$
			$M_4 = 7505$ is selected	$L_3 = 7505$
	⑤		$4 \longrightarrow i$	
i = 4	①		$K_4 = 4$, $M_4 = 7505$	
	②		$M_4 = 7505$ = address to be used in original instruction	

Example 2

			Comments
		(3214) = i 43 01 00 00500	
		(500) = 01595 17 07500	B_1 = 17
			M_1 = 7500
i = 1	①	K_1 = 0, M_1 = 7500	
	②	(M_1) = (7500) = 07501 18 07502	B_2 = 18
		M_2 = 7501 is selected	L_1 = 7501
	⑤	2 ⟶ i	
i = 2	①	K_2 = 1, M_2 = 7501	
	②	(M_2) = (7501) = 07503 19 07504	B_3 = 19
		M_3 = 7504 is selected	R_2 = 7504
	⑤	3 ⟶ i	
i = 3	①	K_3 = 5, M_3 = 7504	
	③(b)	(B_3) = (19) = 000000000009	M_3 is modified by B_3 and becomes M_3 = 7513
		M_3 + (B_3) = 7513	
	④	7513 ⟶ M_3, 0 ⟶ K_3	K_3 - 5 ⟶ K_3
	②	K_3 = 0, (M_3) = (07513) = 07505 20 07506	B_4 = 20, L_3 = 7505
		M_4 = 7505 is selected	
	⑤	4 ⟶ i	
i = 4	①	K_4 = 9, M_4 = 7505	
	③(b)	(B_4) = (20) = 000000000010	M_4 is modified by B_4 and becomes M_4 = 7515
		M_4 + (B_4) = 7515	
	④	7515 ⟶ M_4; 4 ⟶ K_4	K_4 - 5 ⟶ K_4
	②	M_4 = 7515 is selected as address to be used in original instruction.	

Example 3			Comments
		(2413) = i 02 0400 00915	
		(915) = 15690 17 07500	$B_1 = 17$
i = 1	①	$K_1 = 1$, $M_1 = 7500$	
		$(M_1) = (7500) = 07501\ 18\ 07502$	$B_2 = 18$
		$M_2 = 7502$ is selected	$R_1 = 7502$
	⑤	$2 \longrightarrow i$	
i = 2	①	$K_2 = 5$, $M_2 = 7502$	M_2 is modified by B_2 and becomes $M_2 = 7510$
	③	$(B_2) = (18) = 000000000008$	
		$M_2 + (B_2) = 7510$	
	④	$7510 \longrightarrow M_2$, $0 \longrightarrow K_2$	$K_2 - 5 \longrightarrow K_2$
	②	$K_2 = 0$, $(M_2) = 07513\ 22\ 07514$	$B_3 = 22$
		$M_3 = 7513$ is selected	$L_2 = 7513$
	⑤	$3 \longrightarrow i$	
i = 3	①	$K_3 = 6$, $M_3 = 7513$	
	③	$(B_3) = (22) = 000000000012$	M_3 is modified by B_3 and becomes $M_3 = 7525$
		$M_3 + (B_3) = 7525$	
	④	$7525 \longrightarrow M_3$, $1 \longrightarrow K_3$	$K_3 - 5 \longrightarrow K_3$
	②	$K_3 = 1$, $(M_3) = (7525) =$ 07515 23 07516	$B_4 = 23$
		$M_4 = 7516$ is selected	$R_3 = 7516$
	⑤	$4 \longrightarrow i$	
i = 4	①	$K_4 = 9$, $M_4 = 7516$	
	③(b)	$(B_4) = (23) = 000000000013$	M_4 is modified by B_4 and becomes $M_4 = 7529$
		$M_4 + (B_4) = 7529$	
	④	$7529 \longrightarrow M_4$, $4 \longrightarrow K_4$	$K_4 - 5 \longrightarrow K_4$
	②	$M_4 = 7529$ is selected as address to be used in original instruction.	

The time taken by an instruction which uses indirect addressing is increased by the number of key digits used multiplied by 8 microseconds. For example, a normal 4-microsecond instruction would need a total of 28 microseconds if three key digits were used. Note that the number of key digits used includes the one that indicates that the address is the operand address. There is no limit to the number of levels through which the indirect-addressing chain may pass. The stall condition, which normally occurs if a new instruction is not read into IR2 within 3000 microseconds is blocked during the indirect-addressing process.

The indirect-addressing mode of the Larc computer system can be used to perform a table lookup operation. This process consists of locating a given item in a table. The items in the table can be in consecutive or random storage locations.

The procedure to follow in locating an item depends on the evaluation of a series of criteria related to the item. Suppose, for example, a table consists of people's names. The location of each item is determined by the following classification in which each major classification is divided into two classes:

(1) Sex: male or female

(2) Marital status: married or single

(3) Place of residence: east or west of the Mississippi River

(4) Initial letter of last name in the range: A-L or M-Z

(5) And so on.

The order of items in the table can be illustrated by the tree in figure 6-2.

In order to locate a particular name in the table, it is necessary to determine whether the person is male or female, married or single, lives east or west of the Mississippi River, and so on. Each choice indicates which branch of the tree is followed to reach the next choice point. Thus, by always halving the chosen area of the table, the name is converged on. This technique is particularly powerful if the examination of the criteria is separated in time by many other calculations.

To use the indirect-addressing mode with this technique, three things must be present in storage:

(1) The table

(2) A group of storage locations set aside to form a tree of locations. (The indirect-addressing mode travels down this tree until it arrives at the proper entry of the table.)

(3) The key digits which indicate the branches of the tree to be followed.

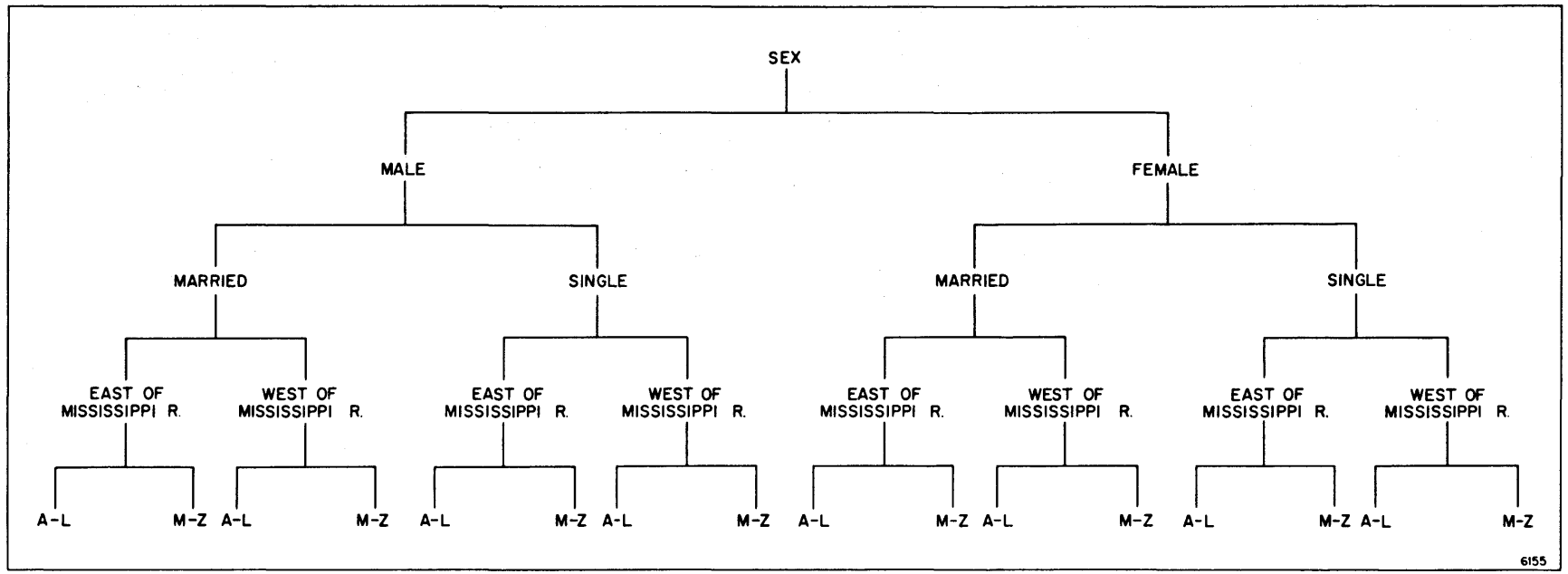

Figure 6-2. Example of Classification Tree

A sample table is illustrated in table 6-1. The indirect-address tree is illustrated in figure 6-3, and the contents of the corresponding group of storage locations are shown in table 6-2. Figure 6-4 is a flow chart which illustrates the sequence of testing the criteria, setting up the appropriate key digits, and performing the actual table lookup operation. Following the flow chart is some sample coding which assumes that the evaluation of the criteria (conditions 1, 2, and 3) are indicated by setting and resetting sense flip-flop 01.

Table 6-1. The Table

(04999) = the 000(0) entry of the table
(00063) = the 001(1) entry of the table
(12500) = the 010(2) entry of the table
(18732) = the 011(3) entry of the table
(04000) = the 100(4) entry of the table
(03000) = the 101(5) entry of the table
(03999) = the 110(6) entry of the table
(00062) = the 111(7) entry of the table

Table 6-2. Storage Locations for Indirect-Address Tree

(02833) = 04033 00 05010
(04033) = 06022 00 03400
(06022) = 04999 00 00063
(03400) = 12500 00 18732
(05010) = 02399 00 02867
(02399) = 04000 00 03000
(02867) = 03999 00 00062

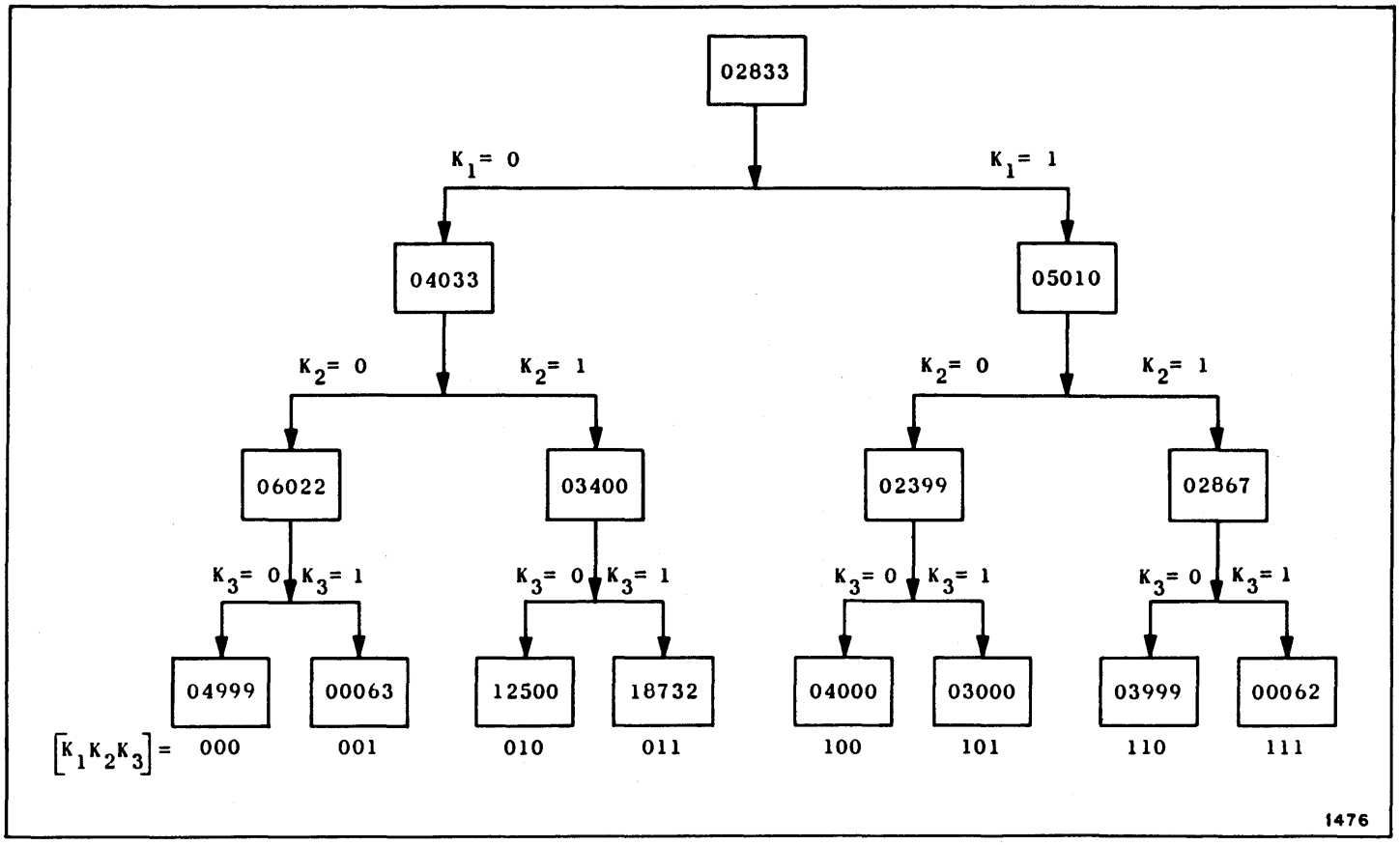

Figure 6-3. Indirect-Address Tree

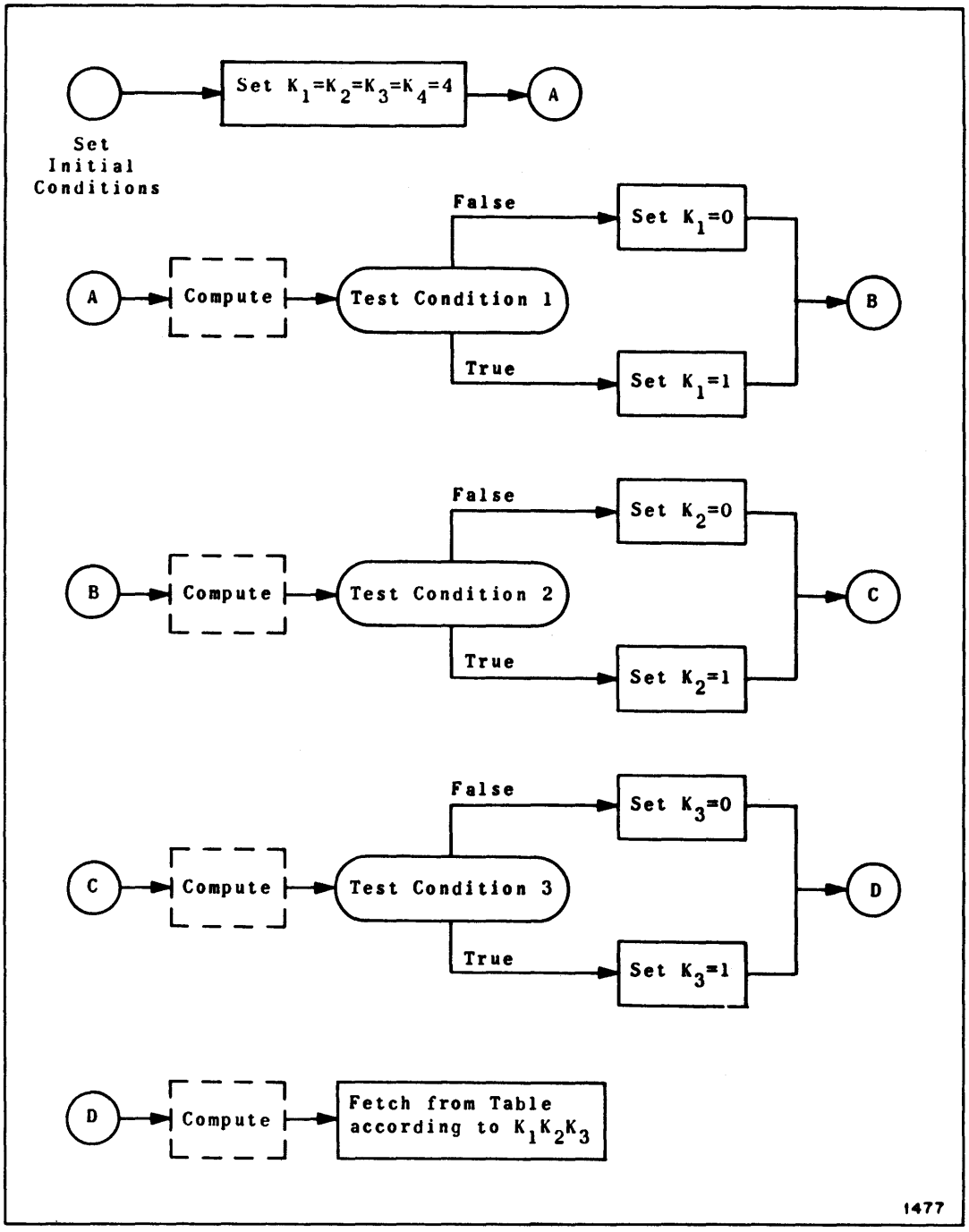

Figure 6-4. Determining Indirect Addressing Sequence

Table Lookup with Indirect Addressing

```
                .
10200           .
                .                    Compute: Part of Computation
                .                    will set FF01 if first con-
                .                    dition is true, reset FF01
                .                    if first condition is false.
                .
10205   .430200125525
10206   .950100102088                Test Condition 1
10207   .430200125266
10208   .430100125200
10209   .660100125277                Set K1
10210   .400100125200
```

```
10310           .
                .                    Compute: Part of computation
                .                    will set FF01 if second con-
                .                    dition is true, reset FF01
                .                    if second condition is false.
                .
10315   .430200125525
10316   .950100103188                Test Condition 2
10317   .430200125266
10318   .430100125200
10319   .660100125288                Set K2
10320   .400100125200
```

Table Lookup with Indirect Addressing

```
10421                .
                     .
                     .                    Compute: Part of computation
                     .                    will set FF01 if condition 3
                     .                    is true, or reset FF01 if
                     .                    condition 3 is false.
10426    . 4 3 0 2 0 0 1 2 5 2 5
10427    . 9 5 0 1 0 0 1 0 4 2 9          Test condition 3
10428    . 4 3 0 2 0 0 1 2 5 2 6
10429    . 4 3 0 1 0 0 1 2 5 2 0
10430    . 6 6 0 1 0 0 1 2 5 2 9          Set K3
10431    . 4 0 0 1 0 0 1 2 5 2 0

10552    i 4 3 0 2 0 0 1 2 5 2 0          Fetch entry from table

12520    4 4 4 4 0 0 0 0 2 8 3 3          M0
                     .
                     .
                     .
                     .
12525    1 1 1 0 0 0 0 0 0 0 0 0          K = 1
12526    0 0 0 0 0 0 0 0 0 0 0 0          K = 0
12527    1 0 0 0 0 0 0 0 0 0 0 0          Extractors for K1
12528    0 1 0 0 0 0 0 0 0 0 0 0          Extractors for K2
12529    0 0 1 0 0 0 0 0 0 0 0 0          Extractors for K3
```

SECTION 7

CONTINGENCIES, ERRORS, AND TRACING MODES

7-1. INTRODUCTION

Included in the Larc computing system are circuits for automatic detection of error conditions. The occurrence of an error condition in the system sets a flip-flop and causes a transfer of control to a predetermined location where the programmer places coded instructions to handle the condition. Error conditions can be divided into those caused by programming mistakes and those caused by machine faults; the former are termed contingencies, the latter, errors. (Certain programming mistakes also result in error indications. These mistakes are noted in paragraph 7-4.)

Contingency and error conditions are discussed in paragraphs 7-2 and 7-4. These paragraphs also include brief descriptions of the contingency and error flip-flops and of suitable contingency and error routines, which will in many cases automatically correct the contingency and error conditions.

The section concludes (paragraph 7-6) with a detailed treatment of the tracing modes and their application in monitoring the progress of Computing Unit programs as they are running.

7-2. CONTINGENCIES

Contingency conditions can be either caused by programming error conditions, or planned by the programmer when he wants automatic notification of certain occurrences. The occurrence of each contingency condition causes a flip-flop associated with the specific condition to be set. Setting of this associated flip-flop causes the master contingency flip-flop (FF99) to be set. Control is then automatically transferred to location 02701 and a return jump to the next main program instruction is recorded automatically in location 02700. The programmer must place in location 02701, the first instruction (or a transfer thereto) of a routine to handle contingency situations. Twenty microseconds is added to the time of the instruction causing the contingency to allow for recording of the return jump and transfer to the contingency routine.

The master contingency flip-flop is set after completion of the execution phase of the instruction causing the contingency. The result of the instruction is stored before control transfers to the contingency routine. The transfer of control occurs before execution of the next instruction; however, if full overlap is achieved and a store instruction follows an instruction which caused a contingency, the store instruction will be completed.

At the time control reaches the contingency routine, the instruction causing the contingency is not contained in any of the instruction registers. It is accessible, however, because the return jump in 02700 contains the address of the instruction following the one which caused the contingency.

The master contingency flip-flop is automatically set when any of the individual flip-flops are set; it is automatically reset when all of the individual flip-flops are reset. The individual contingency flip-flops provide for arithmetic contingencies (flip-flops 39 through 45), processor intervention (flip-flop 11), and manual intervention (flip-flops 30 through 34).

The operation of the master contingency flip-flop can be varied by controls on the engineer's console. These controls, marked CONTINGENCY OPTION, consist of normal (N), ignore (I), and stop (S) pushbuttons. When the N pushbutton is depressed, the contingency flip-flop operates as already described. If the I pushbutton is depressed, there is no transfer of control to the contingency routine when flip-flop 99 is set. If the S pushbutton is depressed, the Computing Unit stops and there is no transfer of control when flip-flop 99 is set.

The exact nature of the contingency routine may be varied from one program to the next, since it depends, in part, on the methods used in the particular program. In general, the routine will consist of a central control portion and a series of subroutines. The central control portion will make the initial flip-flop tests to ascertain which contingency has caused entrance into the routine. Control can then be transferred to the subroutine written to handle the particular contingency. The function of the subroutine depends on the contingency and the design of the main program. In typical problems, the subroutine may print out the data which caused the contingency, indicate the point in the program at which it occurred, and (in some arithmetic contingencies) substitute a predetermined quantity for the given (contingency-causing) result.

After a subroutine has completed its operation, it must arrange for the individual flip-flop to be reset. If the given contingency were the only one which occurred, the master contingency flip-flop would be reset automatically. However, if several contingencies occurred, the master flip-flop would remain set. The main control routine must therefore test the master contingency flip-flop to determine if it remains set. If it does, the flip-flops must be retested to determine which additional ones are set. As each set flip-flop is discovered, the correction subroutine is executed and the flip-flop is reset. This procedure is repeated until

all individual contingency flip-flops are reset, and the master contingency flip-flop is reset. Control is then returned to the main program through the return jump recorded in 02700.

If a contingency occurs while the master contingency flip-flop is set (control is in the contingency routine), the individual flip-flop is set, but the return jump is not recorded. This insures that control will always return to the main program when the contingency routine is completed. If the return jump were altered when a contingency occurred during the contingency routine, the jump would be addressed to a location in the contingency routine, and the main program address would be lost.

It is possible that if a given contingency occurs, the programmer would not want a return to the main program. In this case, control would not be transferred to the main program, but would be transferred to a routine that would wind up the program and bring in a new one.

NOTE

If a contingency or error occurs during the execution of an instruction N and either N + 1 or N + 2 is a halt instruction, control transfers normally to the contingency or error routine; however, depending on the type of error and whether N + 1 and N + 2 are subject to any delay, the Computing Unit may stop after completing the first instruction in the subroutine.

The halt instruction has no effect on the error routine if it follows an instruction which is traced (FF20) or one in which occurs a decoding error in tracing-mode selector digit, (FF49); an instruction odd-even error, (FF51); or a fast register odd-even error in M-address modification, (FF53).

When any of the following individual flip-flops are set, the master contingency flip-flop is also set, causing transfer of control to the contingency routine. The flip-flops are as follows:

(1) Processor-intervention flip-flop, FF11.

(2) Console manual-intervention flip-flops, FF30 through FF34.

(3) Improper operand in arithmetic subtraction flip-flop, FF39.

(4) Zero floating-point adder result flip-flop, FF40.

(5) Non-normalized divisor flip-flop, FF41.

(6) Exponent overflow flip-flop, FF42.

(7) Exponent underflow flip-flop, FF43.

(8) Fixed-point overflow flip-flop, FF44.

(9) Sign-anomaly flip-flop, FF45.

7-3. CONTINGENCY FLIP-FLOPS

Operation of the individual contingency flip-flops is described in the following paragraphs. The conditions listed as causing the arithmetic contingencies are representative but not exhaustive. Sign rules for all instructions are covered in section 3.

PROCESSOR-INTERVENTION CONTINGENCY FLIP-FLOP (FF11)

The processor intervention flip-flop is set by the processor program when it requires certain Computing Unit functions or when it is necessary to notify the Computing Unit of specific conditions in the processor. The processor may require multiplication, division, or complex editing functions from the Computing Unit, or may have a need to notify the Computing Unit of the state of input-output operations or error conditions which have occurred in the input-output equipment. Normally, the processor will place in the contingency subroutine some notification of the condition causing it to interrupt the Computing Unit program. This notification may take the form of a transfer-of-control instruction placed in a fixed location and addressed to a location containing the instructions for handling the specific condition. The Computing Unit contingency routine would then transfer control to the instruction recorded by the processor. This instruction would, in turn, transfer control to the instructions written to satisfy the processor's requirements. In using this sort of communication, it is imperative, of course, that both the Computing Unit and processor programs be designed to use the same locations for the same functions; that is, if the processor places a transfer-of-control instruction addressed to 01150 when a multiplication is required, the Computing Unit program must include a multiplication routine in 01150. Agreement must also be reached on the location of input and output for such an operation.

The programmer is cautioned against attempting to use one-line or certain two-line loops to delay the Computing Unit operations while waiting for processor notification (through intervention) of completion of certain input-output operations. (Refer to paragraph 4-1.)

CONSOLE MANUAL-INTERVENTION CONTINGENCY FLIP-FLOPS (FF30 THROUGH FF34)

Depressing one of the manual-intervention pushbuttons on the operator's console causes the corresponding manual-intervention flip-flop to be set. The caution regarding one-line loops, discussed under processor intervention, also applies to manual intervention. The use of the manual-intervention pushbuttons and the use of FF30 with paper-tape-read operations is discussed in section 5.

IMPROPER OPERAND IN ARITHMETIC SUBTRACTION CONTINGENCY FLIP-FLOP (FF39)

This flip-flop is set during floating-point addition or subtraction if one of the operands is not in proper form, that is, not normalized. The flip-flop is set, for example, when the following conditions occur simultaneously in an addition:

(1) Unequal exponents.

(2) Unlike signs.

(3) Magnitude of number with larger exponent is less than that of number with smaller exponent. (This implies that the number with the larger exponent is not normalized. The other number may or may not be normalized.)

If the flip-flop is not set when the operand is not normalized, then the result is correct. If the flip-flop is set, the result obtained from such an operation is meaningless.

ZERO FLOATING-POINT ADDER RESULT CONTINGENCY FLIP-FLOP (FF40)

This flip-flop is set during floating-point addition and subtraction operations when the result is zero. It also occurs in conversion from a fixed to a floating-point instruction if the number to be converted is zero.

NON-NORMALIZED DIVISOR CONTINGENCY FLIP-FLOP (FF41)

This flip-flop is set during floating-point division if the divisor is zero or if it is not normalized.

EXPONENT OVERFLOW CONTINGENCY FLIP-FLOP (FF42)

This flip-flop is set during any floating-point arithmetic operation if the excess-50 exponent of the result exceeds 99.

EXPONENT UNDERFLOW CONTINGENCY FLIP-FLOP (FF43)

This flip-flop is set during any floating-point arithmetic operation or fixed-to-floating-point conversion operation if the excess-50 exponent of the result is less than 00. This flip-flop is also set if in floating-to-fixed-point conversion operations the scale factor is less than the exponent. In this case, significant digits will be lost.

FIXED-POINT OVERFLOW CONTINGENCY FLIP-FLOP (FF44)

This flip-flop is set during fixed-point addition, subtraction, or division operations if the absolute result is equal to or greater than 1. In all such cases (except division when the divisor is equal to or less than the dividend) the result is equal to the true sum minus 1. In the division case cited, the result is incorrect and meaningless.

This flip-flop is also set during a shift-left instruction if the number of shift places is greater than the number of significant zeros in the number to be shifted. This means that significant non-zeros are shifted off.

SIGN-ANOMALY CONTINGENCY FLIP-FLOP (FF45)

This flip-flop is set when the rules governing the sign digit have been violated.

7-4. ERRORS

An error condition in the Larc Computing Unit causes an error flip-flop associated with the particular condition to be set. An error will usually be traced to a machine fault, but a programming mistake may cause errors in certain cases; for example, if a program attempts to store information in a nonexistent fast register or storage location, or if the operands in an arithmetic operation are not numeric, errors will result.

The setting of one or more error flip-flops will cause the master error flip-flop (FF98) to be set. The master error flip-flop will be set during the result phase of an instruction; this instruction will normally be the instruction in which the error occurred, but may be some other instruction. This will result in an automatic transfer of control to the instruction in location 02601. (The programmer should store, in the area of core storage beginning at location 02601, a routine to handle errors.) When the transfer of control to location 02601 takes place, a return jump to the main program will automatically be stored in location 02600. This jump instruction will be of the form 9 90 00 00 MMMMM, where M is normally the address of the instruction following the one in which the error occurred. When an error occurs, an additional 20 microseconds are taken by the computer in storing the return jump and transferring control to the error routine. (See Note, page 7-3.)

The master error flip-flop will not necessarily be set immediately after the execution of the instruction causing the error. (The address stored in location 02600 is therefore not necessarily the address of the instruction following that which caused the error.) Table 7-1 is a guide for the programmer indicating the range of possible addresses stored in location 02600. In this table E signifies the address of the instruction causing the error. Two of the error flip-flops, (FF38 and FF84), are not listed in the table. In the case of these two flip-flops, the address stored in location 02600 will have a special meaning which will be discussed when the flip-flops are described in detail in paragraph 7-5.

Table 7-1. Contents of 02600
After Transfer to 02601 Occurs

Address	E-2	E-1	E	E+1	E+2
FF Number	50	50,56	50,56,57,58	20; 46 thru 70	46,50

An error routine will usually attempt to find out the cause of an error and print out any relevant data. The following remarks will be useful in attempting to analyze this data. The instruction which is in its result phase when the master error flip-flop is set will be prevented from storing a result in a fast register. (However, if this instruction stores a result in a storage location it will be completed.) If the next instruction is a store instruction, and is in its execution phase when the master error flip-flop is set, it will be completed.

There are 28 error flip-flops in the Computing Unit. One of them (FF20) is the enter-tracing-mode flip-flop which is automatically set when an instruction is decoded which has a tracing digit corresponding to a set selected-tracing-mode flip-flop (covered in paragraph 7-6). Each of the other error flip-flops is used to indicate an error condition.

Any of the 12 "digit flip-flops" (FF71 through FF82) may be set, in addition to the error flip-flops, when certain errors occur. The setting of one of these flip-flops indicates an incorrect bit pattern in the corresponding digit position of an operand or a result, but will not of itself cause the master error flip-flop to be set. However, it is not possible for a digit flip-flop to be set without some error flip-flop first being set.

The master error flip-flop is automatically set when any of the individual error flip-flops are set and is automatically reset when all of the individual error flip-flops are reset. Flip-flop 84 is an exception to the general rule. It is discussed separately later on in this section in the detailed list of error flip-flops.

Error flip-flops 52 and 59 through 70 (which include all the arithmetic flip-flops and two flip-flops connected with inputs to the arithmetic unit) have the following special property: if one or more flip-flops in the group are set (several flip-flops in the group may set simultaneously), the other flip-flops in the group are subsequently prevented from setting.

One or more of the 12 "digit flip-flops" (FF71 through FF82) will be set when any of the following error flip-flops are set: FF51 through FF55, FF59, FF60, FF67, FF68 and FF70. However, once one of these error flip-flops has been set and the relevant digit flip-flops have also been set, no subsequent error will alter the state of the digit flip-flops.

No fixed rules can be given for the organization of an error routine; however, the following outline may be of assistance to the programmer.

Since the enter-tracing-mode flip-flop (FF20) causes a transfer to the error routine, the routine must test this flip-flop and transfer control to the tracing routine if it is set. This subject is discussed in paragraph 7-6.

The error routine must also test all error flip-flops and digit flip-flops, and list those which have been set. Details of the error should be printed on the console printer. Such details might be the flip-flops which were set, the address of the instruction causing the error, the instruction itself, and the addresses and contents of registers or storage locations relevant to the instruction.

When all the error flip-flops have been reset and details of the error have been printed out, the error routine may attempt to repeat the instruction causing the error, if this is possible. If the instruction is repeated without error, then control may be transferred to the main program using the jump instruction stored in location 02600. This approach will only be useful if the error was caused by an intermittent machine fault. If it is impossible to repeat the instruction or if the error occurs a second time, the error routine may either stop the computer or cause the processor to bring in a new program.

If an error and a contingency occur simultaneously, both the master error flip-flop (FF98) and the master contingency flip-flop (FF99) will be set, but control will be transferred to the error routine and the return jump will be stored in location 02600. Hence, when the error routine has completed testing the error flip-flops, it must also test flip-flop 99, the master contingency flip-flop. If flip-flop 99 is set, control must be transferred to the contingency routine; however, the programmer should note that if an error occurs in the contingency routine, the error circuits will function in the normal way and a transfer to the error routine will occur. In this case also, the error routine will find that flip-flop 99 is set. These two situations require the following different treatments by the error routine:

(1) If flip-flop 99 is set and the address in 02600 is a main program address, store the instruction located in 02600 in location 02700 and transfer control to 02701.

(2) If flip-flop 99 is set and the address in 02600 is a contingency-routine address, execute the jump instruction located in 02600 and leave the contents of 02700 unchanged.

In both of these cases, it is assumed that the error routine has completed its work satisfactorily and has successfully repeated the erroneous instruction.

There is a possibility for a contingency to occur in the error routine while the master error flip-flop is still set. The master contingency flip-flop will then be set but no transfer to the contingency routine will occur. This situation will be indistinguishable from case 1 above. This however is not likely to cause trouble, as normally the error routine

results in a contingency only when it is attempting to repeat the erroneous instruction in the main program, in which case the procedure outlined in (1) will be satisfactory. In any event, a contingency in an error routine is an unlikely occurrence.

When the error routine has reset all the error flip-flops, the master error flip-flop will automatically be reset. Once this has occurred another error will result in a transfer of control to 02601 and the storing of a return jump in 02600. Unless precautions are taken, the return jump to the main program which was originally stored in location 02600 will be destroyed by such a sequence. This eventuality may be avoided by causing the error routine to test the address in 02600 at the beginning of the routine. If, after test it is found not to be an error-routine address, the instruction in location 02600 should be preserved in a special location; however, if it is an error-routine address, the instruction in location 02600 should not be preserved in the special location. This organization of the error routine will ensure the preservation of the return jump to the main program. Note that if an error occurs while the master error flip-flop is still set, the computer will automatically stop. This condition is known as a second-error stop.

Although not normally of use to the programmer, two special modes of operation are available for handling errors. Mode selection is controlled at the engineer's console by three adjacent CHECK OPTION pushbuttons. These pushbuttons, which are individually marked N (normal), I (ignore), and S (stop), control the operation of the error circuits as follows:

(1) N: the error circuits operate as previously explained.

(2) I: if errors occur and flip-flop 98 sets, control will not transfer to the error routine. The Computing Unit program will continue without interruption.

(3) S: if errors occur and flip-flop 98 sets, the computer will stop. The computer may be restarted by manually resetting the error flip-flops and depressing the START pushbutton.

7-5. ERROR FLIP-FLOPS

The operation of the individual error flip-flops is described in the following paragraphs. The treatment is not exhaustive and attempts primarily to explain how a programming mistake may set a particular error flip-flop. Further details about the error flip-flops and machine faults are found in the logic manuals covering the control system and the arithmetic system.

Setting any of the 28 error flip-flops listed, unless otherwise stated, will set the master error flip-flop and cause an automatic transfer of control to the instruction in location 02601.

ENTER TRACING MODE FLIP-FLOP (FF20)

Flip-flop 20 is set when the instruction to be executed has in its tracing mode position a digit which corresponds to a set selected-tracing-mode flip-flop (FF21 through FF29). Control is transferred to the error routine before the instruction is executed. When FF20 is set, the address recorded with the return jump in location 02600 is always that of the instruction following the one with the selected tracing mode digit.

IMPROPER-TAPE ERROR FLIP-FLOP (FF38)

This flip-flop is set if the paper tape reader attempts to read a transmit symbol when the operation is being controlled by the Computing Unit through flip-flop 90. When flip-flop 38 is set, the Computing Unit will normally be executing a sequence of instructions. The error will interrupt the sequence and transfer control to the error routine. The address stored in location 02600 will be the address of the instruction following the last one completed before the transfer of control took place.

STALL ERROR FLIP-FLOP (FF46)

A stall error can occur when the control unit sends a call for the contents of a core storage location and no memory-not-busy signal is received. If at the end of 3 milliseconds the control unit is still waiting for the signal, FF46 will be set. The stall error will occur if the modified M address of an instruction exceeds Lim M. The error will also occur if an attempt is made to transfer control to an instruction in a fast register.

CONTROL ERROR FLIP-FLOP (FF47)

If an instruction tests, sets, or resets a nonexistent flip-flop, error flip-flop 47 is set. Flip-flop 47 is also set if an instruction tests, sets, or resets an existing flip-flop and for some reason the execution of the instruction is not completed (refer to paragraph 5-9).

FAST-REGISTER CONTROL ERROR (ON RESULT TIME) FLIP-FLOP (FF48)

This flip-flop is set only as a result of machine error.

DECODING ERROR (IN TRACING-MODE SELECTOR DIGIT) FLIP-FLOP (FF49)

This flip-flop is set if the tracing digit of the instruction being executed is (0), (∧), (-), (+), or is not a legitimate Larc character.

B ADDER ODD-EVEN ERROR (ON INSTRUCTION OR OPERAND CALL) FLIP-FLOP (FF50)

This flip-flop is set when the output from the B adder on an operand or instruction call contains odd-even errors. This flip-flop will be set if the M address or modifier of an instruction contains non-numeric characters.

INSTRUCTION ODD-EVEN ERROR FLIP-FLOP (FF51)

This flip-flop is set when the instruction called contains an odd-even error.

OPERAND ODD-EVEN ERROR FLIP-FLOP (FF52)

If an operand called from memory contains odd-even errors, flip-flop 52 is set.

FAST REGISTER ODD-EVEN ERROR (IN M-ADDRESS MODIFICATION) FLIP-FLOP (FF53)

This flip-flop is set when M-address modification is attempted using a B modifier which contains odd-even errors.

FAST REGISTER ODD-EVEN ERROR (ON TIME SLOT M) FLIP-FLOP (FF54)

If odd-even errors exist in the contents of the fast-register operand read out on the M slot, this error flip-flop is set. For example, the flip-flop is set if an attempt is made to read from a non-existent fast register or a fast register which contains odd-even errors.

FAST REGISTER ODD-EVEN ERROR (ON RESULT TIME) FLIP-FLOP (FF55)

This flip-flop is set if an attempt is made to store in a non-existent fast register or in a fast register which contains characters with odd-even errors.

B ADDER ODD-EVEN ERROR (ON OUTPUT TO C1, TO HIGH-SPEED BUS, OR TO ARITHMETIC UNIT) FLIP-FLOP (FF56)

This flip-flop is set only as a result of machine error.

B ADDER ODD-EVEN ERROR (ON OUTPUT TO FAST-REGISTER SELECTOR, TO SELECTOR STORAGE, OR TO M DIGITS OF IR2) FLIP-FLOP (FF57)

This error flip-flop can be set by non-numeric characters appearing in the A, B, or M fields, or in the B modifier.

B ADDER ODD-EVEN ERROR (ON OUTPUT TO C2) FLIP-FLOP (FF58)

This flip-flop is set only as the result of machine error.

ADDER OUTPUT ODD-EVEN OR NON-NUMERIC ERROR FLIP-FLOP (FF59)

This flip-flop is set if non-numeric characters are added to other characters in the adder. This flip-flop is also set if odd-even errors exist in the output of the adder. These conditions can occur in comparison, shift, conversion, index, or arithmetic instructions.

SHIFTER OUTPUT ODD-EVEN ERROR FLIP-FLOP (FF60)

If the signals which indicate how much to shift are lacking, error flip-flop 60 is set. If a circular-shift instruction indicates a shift greater than 24, error flip-flop 60 is set. (Refer also to FF63.) Flip-flop 60 can also be set when non-numeric characters are present in nonsign positions of operands in certain fixed-point and floating-point arithmetic instructions. A non-numeric character in an indirect addressing key word will also set flip-flop 60.

COMPARATOR ERROR (SINGLE-PRECISION DIVISION) FLIP-FLOP (FF61)

This flip-flop is set only as a result of machine error.

MULTIPLIER, QUOTIENT, AND EXTRACTOR ERROR FLIP-FLOP (FF62)

This flip-flop is set only as a result of machine error.

SHIFT CONTROL ERROR FLIP-FLOP (FF63)

If the shift digits in a circular-shift instruction specify a shift which is greater than 24 places, error flip-flop 63 is set.

ADDER-OVERFLOW ERROR FLIP-FLOP (FF64)

This flip-flop can be set when non-numeric characters are present in nonsign positions of operands in certain arithmetic and conversion instructions.

ARITHMETIC-UNIT PROGRAM COUNTER AND DECODER ERROR FLIP-FLOP (FF65)

This flip-flop is set only as a result of machine error.

ENDING-PULSE ERROR FLIP-FLOP (FF66)

This flip-flop is set only as a result of machine error.

AH REGISTER ODD-EVEN ERROR FLIP-FLOP (FF67)

This flip-flop can be set if non-numeric characters are contained in nonsign positions of the multiplier or divisor in fixed-point or floating-point multiplication or division.

AD REGISTER ODD-EVEN ERROR FLIP-FLOP (FF68)

This flip-flop is set only as a result of machine error.

SIGN DIGIT ODD-EVEN ERROR FLIP-FLOP (FF69)

When an odd-even error occurs in the sign position of the operand read out on the M slot, error flip-flop 69 is set. Flip-flop 69 will be set, for example, in a store instruction in which the A address exceeds Lim A or in which the sign position of fast register A contains an odd-even error.

A REGISTER ODD-EVEN ERROR (ON TIME SLOT A) FLIP-FLOP (FF70)

This error flip-flop is set when an odd-even error is detected in the contents of a fast register when they are read out on time slot A. Flip-flop 70 will be set, for example, in a single-precision, fixed-point add instruction in which the A address exceeds Lim A or in which the contents of fast register A include odd-even errors.

CYCLING-UNIT ERROR FLIP-FLOP (FF84)

This error flip-flop is set if the cycling unit stops or if there is a component failure in the cycling unit.

The cycling unit provides the basic timing pulses for the Larc system Computing Unit. If the cycling unit fails completely then flip-flop 84 will set and the computer will stall (FF46 will set); the lack of timing pulses will prevent the setting of the master error flip-flop, control will not transfer to the error routine, and nothing will be stored in location 02600. If the cycling unit fails for only a short time and then restarts, flip-flop 84 will set and flip-flop 98 will set after the restart. In addition, other error flip-flops will invariably be set. In any event, control will be transferred to the error routine. The address stored in location 02600 cannot be precisely defined in this case, but will be within one or two locations of the instruction that was in its execution phase when the cycling unit failed.

It is possible for the programmer to set flip-flop 84 by means of a set-flip-flop instruction. In this case, the master error flip-flop will not be set and control will not be transferred to the error routine. Note however, that if flip-flop 84 is set and the master error flip-flop is also set, then the latter flip-flop will not reset until flip-flop 84 is reset.

7-6. TRACING MODES

It is well known that a computer program will rarely work correctly the first time it is run. Sometimes programmers may write short programs which are free of errors, but in a complex program of any length, it is usual for numerous typographical errors and mistakes in program logic to remain even after the program has been checked visually many times.

The remaining errors in a program must be discovered by running the program on the computer, usually with a set of test data. Typographical errors and many mistakes in program logic will often result in machine errors or contingencies. In the Larc Computing Unit the occurrence of errors or contingencies will transfer control to special routines which will print out information relating to the errors or contingencies. Using the printouts, the programmer should be able to correct the more obvious programming mistakes.

Some programming mistakes cannot be so easily corrected. For example, if a typographical error in a program results in an erroneous jump, a machine error might occur many instructions after the mistake in the program. Or, as another example, a program might appear to run without error, but produce incorrect results. In these cases it would be useful to be able to monitor the progress of programs as they are running. This facility is provided in the Larc Computing Unit in the form of tracing modes.

Normally, a Larc Computing Unit instruction word will have a period in the most significant, or tracing digit, position. When such an instruction is decoded, it will be executed normally. If a programmer wishes to monitor a program he will place one of the digits 1 through 9 in the tracing-digit position of selected instructions, and at the beginning of his program he will set the corresponding selected-tracing-mode flip-flop (FF21 through FF29). When an instruction with one of the digits 1 through 9 is decoded and the corresponding selected-tracing-mode flip-flop is set, the execution of the instruction will be inhibited and the computer will operate in the selected tracing mode. In detail, what happens is this: the control circuits set the enter-tracing-mode flip-flop (FF20). A skip is then inserted automatically into IR2 and executed. While the skip is being executed the master error flip-flop (FF98) is set, and control is transferred to the error routine in the normal way. The error routine entry (location 02601) is used as a common entry point for the error and tracing routines. Computing Unit control is transferred to the instruction in location 02601 and a return jump to the main program is stored in location 02600. This jump is of the form 9 90 00 00 MMMMM, where M is the address of the instruction following the instruction being traced.

After testing the various error flip-flops, the error routine must test flip-flop 20 and, if it is set, transfer control to the tracing routine. The tracing routine may be written by the programmer to suit the special purposes of his program. The organization of such a routine might be as follows: After preserving the return jump instruction stored in location 02600, the tracing routine should reset the enter-tracing-mode flip-flop; then, if no error flip-flops were set, the master error flip-flop would automatically reset. (The jump instruction must be preserved so that a possible error in the tracing routine will not destroy it.) The routine should next bring from storage the instruction being traced. After examining the tracing digit of the instruction to see which tracing mode is required, the routine should change the tracing digit to a period and cause the instruction to be executed. The tracing routine should then transfer control to a subroutine corresponding to the particular tracing

mode. This subroutine might then cause the processor to print out information required by the programmer for the monitoring of his program. Such information might be intermediate results, contents of fast registers, the states of counters, etc. After the tracing routine has completed its monitoring work, it may transfer control back to the main program by executing the jump instruction which was originally stored in location 02600, and was later preserved by the tracing routine. If flip-flop 29 is set, however, the tracing digit (9) in the jump instruction must be replaced by a period before the jump is executed. Otherwise, control will be transferred immediately back to the error routine and tracing routine.

There are two things the programmer must be aware of when using the tracing routine. First, the tracing routine must reset flip-flop 20 which will automatically reset the master error flip-flop (provided that no error flip-flops are set). If this is not done, the occurrence of an error or the tracing of another instruction will cause the computer to stop. This stop is known as a second-error stop. Secondly, if a set flip-flop instruction sets one of the selected-tracing-mode flip-flops (FF21 through FF29), the two instructions immediately following the set-flip-flop instruction cannot normally be traced in that particular mode. The reason for this is that by the time the flip-flop is set, the two instructions will have already passed the point in their execution cycle where their tracing digits are compared with the corresponding selected-tracing-mode flip-flop. This is not a serious restriction because usually the required selected-tracing-mode flip-flops will be set at the start of a program before any instruction is encountered which requires tracing. For the same reason, if a reset flip-flop instruction resets one of the selected-tracing-mode flip-flops, the two instructions immediately following the reset-flip-flop instruction can still be traced in that particular mode.

A tracing digit in an instruction will have no effect unless the corresponding tracing mode flip-flop is set. Accordingly, when a program has been monitored and corrected by use of the tracing routine, it requires only slight alteration to remove the tracing facility. The removal of those instructions which set the selected-tracing-mode flip-flops at the start of the program will prevent all tracing. The tracing digits in instructions do not need to be changed to periods.

The discussion in the following paragraphs covers the special purposes for which tracing modes 1 and 9 are used in the Larc computing system.

Tracing mode 1 is used in conjunction with the engineer's console and the visual-display registers for engineering monitoring of special registers and signals. The special use of tracing mode 1 is activated by setting up the program display mode by controls on the engineer's console. When the Computing Unit is operating in this mode, the decoding of an instruction with a 1 as the tracing digit will produce special displays in the visual-display registers according to the engineer's needs. The actual tracing function in the Computing Unit is not affected in any way by the program mode.

Tracing mode 9 may be used as a common printing routine for errors and contingencies, as the return instructions stored in locations 02600 (for errors) and 02700 (for contingencies) have a 9 as the tracing digit. If the error or contingency routine following resetting of the master error or master contingency flip-flop executes the return instruction, and flip-flop 29 is set, control will be transferred to the tracing routine corresponding to digit 9. This routine should be so written that it causes a printout of information previously stored in special locations by the error or contingency routines.

If tracing mode 9 is used for this purpose, the error routine must make special provision for preserving the return jump stored in location 02600. If this instruction is not preserved, the reentry to the error routine caused by the decoding of tracing digit 9 will destroy the instruction and prevent a return to the main program. If the error routine and contingency routine do not use tracing mode 9, the mode may be used by the programmer in the normal way.

The programmer should note that both the SLJ and the TR instructions store jump instructions with a 9 tracing digit. If tracing mode 9 is used as a print routine, these jump instructions will require special attention.

APPENDIX A

NUMERICAL LIST OF INSTRUCTIONS

This appendix contains a list of Computing Unit instructions in numerical order. The instructions are listed by numeric code number, and the information covers the mnemonic code, symbolic notation, time of execution, and a page reference to the text.

Numeric Code	Mnemonic Code	Symbolic Notation	Time (μs)	Page Reference
00	SK	Skip (C1) + 1 ⟶ C1	4	3-68
01	AX	(M) + (A) ⟶ A	4	3-13
02	A	(M) ⊕ (A) ⟶ A	4	3-45
03	AM	|(M)| ⊕ (A) ⟶ A	4	3-46
04	AU	(M) ⊕ (A) ⟶ A + 1	4	3-47
05	AAX	(M') + (A') ⟶ A'	12	3-18
06	AA	(M') ⊕ (A') ⟶ A'	16	3-55
09	FV	If interlock set, 0——0 (5DD) ⟶ 02650; then (02650) ⟶ A and reset connect and inter- lock FF's If interlock reset, M ⟶ C	4 12	3-64
11	NX	-(M) + (A) ⟶ A	4	3-13
12	N	-(M) ⊕ (A) ⟶ A	4	3-48
14	NU	-(M) ⊕ (A) ⟶ A + 1	4	3-48
15	NNX	-(M') + (A') ⟶ A'	12	3-19
16	NN	-(M') ⊕ (A') ⟶ A'	16	3-56
19	FVK	If interlock set, (12DD) ⟶ 02650; then (02650) ⟶ A and reset connect and inter- lock FF's If interlock reset, M ⟶ C	4 12	3-65
20	MXR	[(M) × (A)] Rdd ⟶ A	8	3-14
21	MXE	(M) × (A) ⟶ A'	12	3-15
22	MR	[(M) ⊗ (A)] Rdd ⟶ A	12	3-49
23	M	(M) ⊗ (A) ⟶ A	8	3-50

Numeric Code	Mnemonic Code	Symbolic Notation	Time (μs)	Page Reference
24	MU	(M) ⊗ (A) ⟶ A + 1	8	3-51
25	ME	(M) ⊗ (A) ⟶ A'	12	3-52
26	MMX	(M') × (A') ⟶ A'	36	3-19
27	MM	(M') ⊗ (A') ⟶ A'	36	3-57
29	SV	If interlock reset, (A) ⟶ 02650; then (02650_M) ⟶ 5DD	4	3-65
		If interlock set, M ⟶ C	12	
30	DX	(A) ÷ (M) ⟶ A	32	3-16
31	DXE	(A) ÷ (M) ⟶ A Remainder ⟶ A + 1	36	3-17
32	DR	[(A) ⊜ (M)] Rdd ⟶ A	28	3-53
34	DUR	[(A) ⊜ (M)] Rdd ⟶ A + 1	28	3-54
35	DDX	(A') ÷ (M') ⟶ A'	184	3-20
36	DD	(A') ⊜ (M') ⟶ A'	168	3-58
37	DSE	(A') ⊜ (M) ⟶ A'	56	3-59
39	SVK	If interlock reset, (A) ⟶ 02650; then (02650) ⟶ 12DD	4	3-66
		If interlock set, M ⟶ C	12	
40	S	(A) ⟶ M	4	3-7
41	SN	-(A) ⟶ M	4	3-8
42	SM	|(A)| ⟶ M	4	3-8
43	F	(M) ⟶ A	4	3-11
45	SS	(A) ⟶ M (A + 1) ⟶ M + 1	8	3-9
46	SSN	-(A) ⟶ M -(A+1) ⟶ M + 1	8	3-10

Numeric Code	Mnemonic Code	Symbolic Notation	Time (μs)	Page Reference
47	SSM	$\|(A)\| \longrightarrow M$ $\|(A + 1)\| \longrightarrow M + 1$	8	3-11
48	FF	$(M) \longrightarrow A$ $(M + 1) \longrightarrow A + 1$	8	3-12
50	CX	$(A) - FL \longrightarrow A - FX$ M: scale factor	4	3-61
51	C	$(A) - FX \longrightarrow A - FL$ M: scale factor	4	3-61
52	PR	$(A) \times 10^{-M} \longrightarrow A$ (right shift M places)	4	3-36
53	PL	$(A) \times 10^{M} \longrightarrow A$ (left shift M places	4	3-37
55	CCX	$(A') - FL \longrightarrow A' - FX$ M: scale factor	12	3-62
56	CC	$(A') - FX \longrightarrow A' - FL$ M: scale factor	12	3-63
57	PPR	$(A') \times 10^{-M} \longrightarrow A'$ (right shift M places)	8	3-37
58	PPL	$(A') \times 10^{M} \longrightarrow A'$ (left shift M places)	8	3-39
59	PPC	$(A') \times 10^{M} \longrightarrow A'$ (circular) (left circular shift)	12	3-40
60	EOP	$(M_I) \longrightarrow A_I$	4	3-32
61	EA	$(M_A) \longrightarrow A_A$	4	3-32
62	EB	$(M_B) \longrightarrow A_B$	4	3-32
63	EAB	$(M_{AB}) \longrightarrow A_{AB}$	4	3-33
64	EM	$(M_M) \longrightarrow A_M$	4	3-33
65	EL	$(A - 1) \longrightarrow A$ (M)	8	3-33
66	EU	$(A + 1) \longrightarrow A$ (M)	8	3-34

Numeric Code	Mnemonic Code	Symbolic Notation	Time (μs)	Page Reference
70	TE	(A) = (A + 1) ?		3-24
		Yes: M ⟶ C	12	
		No: (C) + 1 ⟶ C	4	
71	TG	(A) > (A + 1) ?		3-26
		Yes: M ⟶ C	12	
		No: (C) + 1 ⟶ C	4	
72	TZ	(A) = 0 ?		3-27
		Yes: M ⟶ C	12	
		NO: (C) + 1 ⟶ C	4	
73	TGZ	(A) > 0 ?		3-28
		Yes: M ⟶ C	12	
		No: (C) + 1 ⟶ C	4	
74	TLZ	(A) < 0 ?		3-29
		Yes: M ⟶ C	12	
		No: (C) + 1 ⟶ C	4	
75	TTE	(A') = (A + 2') ?		3-30
		Yes: M ⟶ C	16	
		No: (C) + 1 ⟶ C	8	
76	TTG	(A') > (A + 2') ?		3-30
		Yes: M ⟶ C	16	
		No: (C) + 1 ⟶ C	8	
80	BIT	$(A_\Delta) + (A_D) \longrightarrow A_\Delta$		3-41
		$(A_N) - 1 \longrightarrow A_N$		
		New $(A_N) \neq 0$; M ⟶ C	8	
		New $(A_N) = 0$; (C) + 1 ⟶ C	12	
81	BDT	$(A_\Delta) - (A_D) \longrightarrow A_\Delta$		3-42
		$(A_N) - 1 \longrightarrow A_N$		
		New $(A_N) \neq 0$; M ⟶ C	8	
		New $(A_N) = 0$; (C) + 1 ⟶ C	12	

Numeric Code	Mnemonic Code	Symbolic Notation	Time (μs)	Page Reference
82	BIC	$(A_\Delta) + (A_D) \longrightarrow A_\Delta$ $(A_N) - 1 \longrightarrow A_N$		3-43
		New $(A_N) = 0$; M $\longrightarrow$ C	12	
		New $(A_N) \neq 0$; (C) + 1 $\longrightarrow$ C	4	
83	BDC	$(A_\Delta) - (A_D) \longrightarrow A_\Delta$ $(A_N) - 1 \longrightarrow A_N$		3-43
		New $(A_N) = 0$; M $\longrightarrow$ C	12	
		New $(A_N) \neq 0$; (C) + 1 $\longrightarrow$ C	4	
85	BI	$(A_\Delta) + (A_D) \longrightarrow A_\Delta$	4	3-41
86	BD	$(A_\Delta) - (A_D) \longrightarrow A_\Delta$	4	3-41
90	T	M $\longrightarrow$ C	8	3-21
91	TR	990——0 (C) + 1 $\longrightarrow$ M M + 1 $\longrightarrow$ C	12	3-22
92	TB	(C) $\longrightarrow A_M$ M $\longrightarrow$ C	8	3-23
93	SLJ	If M is storage location, 990——0 (C2) $\longrightarrow$ M If M is fast register, 0——0 (C2) $\longrightarrow$ M	4	3-34
95	TF	Test FF A:		3-67
		If set, M $\longrightarrow$ C	12	
		If reset, (C) + 1 $\longrightarrow$ C	4	
96	RF	Reset FF A	4	3-68
97	SF	Set FF A	4	3-68
99	H	STOP	-	3-69

APPENDIX B

ADDRESSABLE FLIP-FLOPS IN THE COMPUTING UNIT

Flip-Flop	Title	Controllable by Computing Unit Program Instruction		
		Test	Reset	Set
00...09	Sense	95	96	97
10	Disclosure (can be tested and reset by processor)	95		97
11	Processor-intervention contingency (can be tested and set by processor)	95	96	
15	Manual- and IOP-intervention inhibit	95	96	97
20	Enter tracing mode	95	96	
21...29	Selected tracing mode	95	96	97
30...34	Console manual-intervention contingency	95	96	
38	Improper-tape error	95	96	
39	Improper operand in arithmetic subtraction contingency	95	96	
40	Zero floating-point adder result contingency	95	96	
41	Non-normalized-divisor contingency	95	96	
42	Exponent-overflow contingency	95	96	
43	Exponent-underflow contingency	95	96	

Flip-Flop	Title	Controllable by Computing Unit Program Instruction		
		Test	Reset	Set
44	Fixed-point overflow contingency	95	96	
45	Sign-anomaly contingency	95	96	
46	Stall error	95	96	
47	Control error	95	96	
48	Fast-register control error (on result time)	95	96	
49	Decoding error (in tracing-mode selector digit)	95	96	
50	B-adder odd-even error (on instruction or operand call)	95	96	
51	Instruction odd-even error	95	96	
52	Operand odd-even error	95	96	
53	Fast-register odd-even error (in M-address modification)	95	96	
54	Fast-register odd-even error (on time slot M*)	95	96	
55	Fast-register odd-even error (on result time)	95	96	
56	B-adder odd-even error (on output to control counter 1, or to the high-speed bus, or to the arithmetic unit)	95	96	
57	B-adder odd-even error (on output to the fast-register selector, or to selector storage, or to the M digits of instruction register 2)	95	96	
58	B-adder odd-even error (on output to control counter 2)	95	96	

* Time slot M is the time at which the contents of a fast register are read out when addressed by the M digits of an instruction. In certain instructions a fast register addressed by the A digits is read out on time slot M.

Flip-Flop	Title	Controllable by Computing Unit Program Instruction		
		Test	Reset	Set
59	Adder output odd-even or non-numeric error	95	96	
60	Shifter output odd-even error	95	96	
61	Comparator error (single-precision division)	95	96	
62	Multiplier, quotient, and extractor error	95	96	
63	Shift-control error	95	96	
64	Adder-overflow error	95	96	
65	Arithmetic-unit program counter and decoder error	95	96	
66	Ending-pulse error	95	96	
67	AH register odd-even error	95	96	
68	AD register odd-even error	95	96	
69	Sign digit odd-even error	95	96	
70	A register odd-even error (on time slot A*)	95	96	
71...82	Odd-even error, digit position	95	**	
84	Cycling-unit error	95	96	97
90	Start tape	95	96	97
98	Master error	95	***	
99	Master contingency	95	****	

* Time slot A is the time at which the contents of a fast register are normally read out when addressed by the A digits of an instruction. Refer to note on FF54.

** Flip-flops 71 through 82 are automatically reset when all the following error flip-flops are reset: 51...55; 59, 60, 67, 68, 70.

*** FF98 is automatically reset when all of the error flip-flops (20, 38, 46...70, and 84) are reset.

**** FF99 is automatically reset when all of the contingency flip-flops (11, 30...34, and 39...45) are reset.

APPENDIX C
REMINGTON RAND UNIVAC PROCESSOR PROGRAM

It is possible to write many different processor programs which are specifically tailored for each program run on the computer; however, this would be costly and time consuming. Because many input-output procedures will not vary from one problem to the next, it is possible to write a general-purpose processor program which will work relatively efficiently for most applications.

Remington Rand has developed such a general-purpose processor program.[1] In using this processor program, the Computing Unit programmer writes summary orders to express commands to the processor. A summary order is a pseudo-command to the processor, and indicates to the processor program the exact series of operations which must be performed. The processor program picks up and analyzes the summary orders, and according to the analysis, executes the required instructions.

This program occupies all of the first storage unit, which is permanently interlocked against computer access. In addition, the program uses the last 1500 words of the second storage unit. These locations should not be altered by the computer program.

The first two digits of the summary order specify the type of operation to be performed; the remaining digits indicate which particular device is to be used, which part of the device is involved, and what storage locations, if any, are involved. The exact format depends on the specific summary order.

Summary orders are transferred to the processor in what are called packets. A packet may consist of a single summary order or a group of summary orders placed in sequential storage locations. An end-of-packet word must be placed in the storage location immediately following the last summary order in the packet. The end-of-packet word provides a means of determining when summary orders which transfer data into or out of storage have been completed. The time of completion of non-storage summary orders, however, will not usually be of interest to the Computing Unit programmer.

---

[1] H. Bruch, The Standard Processor Program, Remington Rand Univac Division of Sperry Rand Corporation, 1961.

The processor performs these orders in sequence, but they do not affect data or equipment required by the Computing Unit program.

The end-of-packet word can be in either of two formats: a computer-check end-of-packet word or a processor-interrupt end-of-packet word.

The computer-check end-of-packet word is used when the Computing Unit program is going to use the input-output data when it is free and able to do so. The processor-interrupt end-of-packet word is used when the Computing Unit intends to use the data from the summary orders as soon as it is available. The processor-interrupt end-of-packet word provides the Computing Unit program with immediate notification of the completion of summary orders through automatic transfer of control to that part of the Computing Unit routine which is to handle the summary order information. The end-of-packet word must be included even though there are no data-transfer summary orders in the packet. If there are no data-transfer summary orders the completion indication is made as soon as the summary orders have been either executed or filed for execution. The computer-check end-of-packet word has the following format:

0 00 00 00 MMMMM

where MMMMM is the address of a core-storage location in which the processor program stores a word made up of all zeros (.00000000000) when all storage summary orders in the packet are complete. By checking MMMMM for zeros, the computer program can determine when the summary orders have been carried out. It is also possible to use MMMMM in the following manner: if the programmer has a loop to perform while waiting for the summary orders to be completed, he can use MMMMM as an unconditional-transfer-of-control instruction at the end of the loop; then, when the processor stores zeros in MMMMM, it will function as a skip instruction, and control will pass to the instructions following MMMMM.

The processor-interrupt end-of-packet word has the following format:

. 90 00 00 MMMMM

where MMMMM is the address of a location to which control is to be transferred when the storage summary orders have been completed. This transfer of control is effected in the following manner: when the summary orders have been completed, the processor places the end-of-packet word (. 90 00 00 MMMMM) in location 02799 and sets the processor intervention flip-flop; the Computing Unit then enters the contingency routine, and through the processor intervention subroutine, executes the . 90 00 00 MMMMM instruction, transferring control to location MMMMM.

Note that digits 4 through 7 of both types of end-of-packet word must be zeros. These digits are used by the processor program, and must not be altered by the Computing Unit program until the processor has completed processing the packet.

If a packet is to be used more than once, the programmer may list a series of end-of-packet words after the packet, and each time the packet is issued, a different end-of-packet word address is listed in the disclosure word. An end-of-packet word which precedes the effective one is treated as a skip summary order.

After the summary orders are placed in desired locations, the Computing Unit program places a disclosure word in storage location 02500 and sets the disclosure flip-flop. When the processor finds the disclosure flip-flop set, it picks up the disclosure word and resets the flip-flop. After the flip-flop is reset, the computer can set it again for another group of summary orders. The processor will not check the disclosure flip-flop again until the current summary orders are being executed or are distributed to the appropriate files to await execution.

The disclosure word provides the programmer with one method of determining whether the program is in an endless loop. There are two forms of the disclosure word, a primary disclosure word and a secondary disclosure word. A primary disclosure word has the format 15 NNNNN MMMMM, and the secondary disclosure word has the format 14 NNNNN MMMMM; NNNNN is the address of the first summary order in the packet and MMMMM is the address of the end-of-packet word. The processor records the time it picks up the disclosure word. If it is a primary disclosure word, and there is no other primary disclosure word given within 5 minutes of this time, there is a printout on the console printer indicating this situation. The printout will be repeated every 5 minutes until there is another primary disclosure word.

If the disclosure word is a secondary disclosure word, and there is no other secondary disclosure word given within 5 minutes, there will be a printout on the console printer indicating this fact. In the case of the secondary disclosure word, the printout is not repeated. The secondary disclosure word is generally used in a side routine being performed when the main program does not occupy the full time of the Computing Unit. In this situation, the programmer needs to be able to distinguish between a tieup in a loop in the secondary program and a tieup in the main program.

The disclosure word printout is only effective in detecting loops which do not include delivery of summary orders. This mechanism will not be able to indicate a loop in which the same disclosure word is continually being issued.

If it is important to the programmer to know if the last packet of summary orders has been picked up, he can issue a pseudo-disclosure word (000000000000 or .00000000000) and set the disclosure flip-flop. The processor does not examine the disclosure flip-flop and pick up a new disclosure word (and therefore does not reset the flip-flop) until the preceding packet has either been executed or distributed to the proper files. Therefore, the program is aware that if the disclosure flip-flop is reset (the disclosure word picked up) after the pseudo-disclosure word is issued, the preceding packet has been filed or executed. The pseudo-disclosure word, then, permits the programmer to check the progress of the last packet without issuing any additional summary orders. The pseudo-disclosure order does not initiate any other action in the processor.

After the processor has picked up the disclosure word, it begins to execute the summary orders in the sequence in which they are listed. Execution of a summary order may be delayed if the device it calls on or the required synchronizer is already in use, or if the storage area it affects is in use or reserved to be used by a prior summary order.

To ensure that the summary orders are executed according to the correct priority, the processor program keeps files (device files) of the summary orders waiting to be executed. A device file is maintained for each input-output device. As the device completes one summary order, the next summary order in the file is picked up and executed.

A storage-conflict file is maintained for summary orders which refer to storage areas. The file consists of ten 3-word slots and one conflict slot. The 3-word slots are used to contain the beginning and ending addresses of the storage area affected by the summary order and an instruction which is to be executed when the summary order is completed. This instruction subtracts 1 from the summary order count connected with the end-of-packet word. The conflict slot is used when the current summary order refers to a storage area which overlaps with a storage area used by a previous uncompleted summary order. In this case, zeros are placed in the current summary order's place in the device file. When the storage area is released by the completion of the conflicting summary order, the current summary order is removed from the conflict slot and placed in the device file.

Figure C-1 is a flow chart illustrating the steps by which a summary order is executed or filed to await execution. It is not intended to show details of procedures performed by the processor program.

As it stands, the processor program does not include any provision for notifying the Computing Unit of errors it detects, except for the errors discovered by the synchronizers which cause printouts on the console printer. The processor program does detect errors such as bad summary orders and bad disclosure words, but as it is written, the program merely stops the processor when such conditions occur. It is up to the individual programmer or installation to determine the method by which error information is conveyed to the Computing Unit program. Processor and Computing Unit programs must be written according to the same communication system. One suggested method is as follows: store a list of instructions to transfer to fixed locations. The fixed locations contain instructions for handling specific errors. When an error occurs the processor program picks up the appropriate transfer instruction, places it in a specified location in the contingency routine, and sets the processor intervention flip-flop. When this flip-flop is set, the contingency routine transfers control to the specified location; control is then transferred to the appropriate instructions to handle the error.

The summary orders are listed in paragraphs C-1 through C-4, following, according to the input-output device to which they apply. Miscellaneous summary orders which do not apply to specific devices are listed in paragraph C-5. An x in the format of a summary order indicates a digit position which is not interpreted by the processor program; however, this position must contain one of the digits 0 through 9.

The summary orders for each device are followed by gross timings for phases of the input-output operations. These timings are maximum timings assuming the equipment is available. It would be misleading to give total timing for each summary order, because the time is dependent upon the position of the equipment at the time the order is received, and the amount of movement required to complete the order.

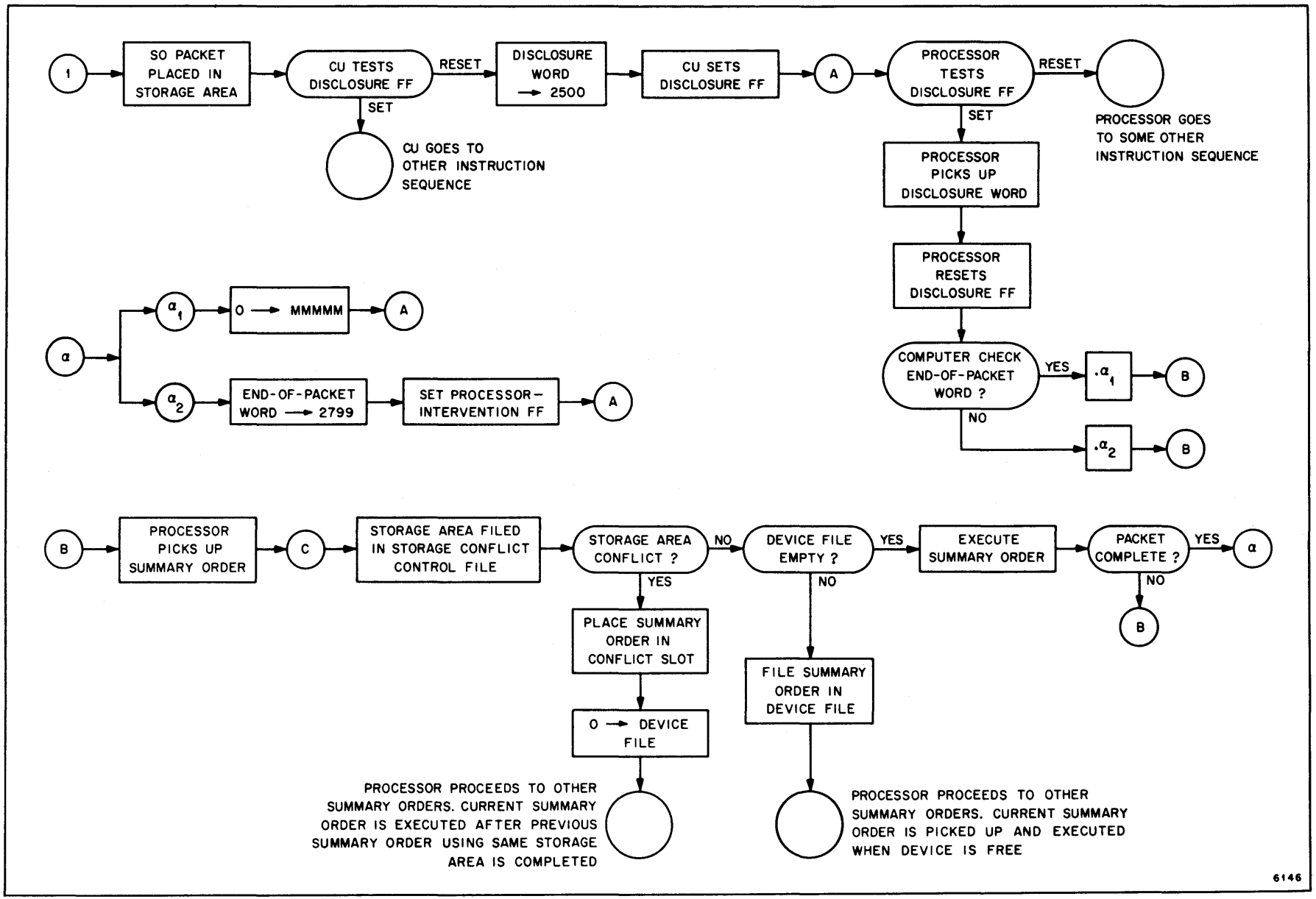

Figure C-1. Summary Order Execution and Filing

C-1. MAGNETIC DRUM SUMMARY ORDERS

The magnetic drum summary orders available to the Computing Unit programmer include instructions for reading, writing, positioning the head assemblies, interlocking the drum, and setting up the drum format.

It is possible for the programmer to limit the effective number of bands on each drum. If this is desired, he can give a drum-format summary order specifying the number of bands to be used and the actual band which is to be considered as the new 00 band. The drum-format summary order has the following format:

93 000 DD SB x HB

where DD is the drum number, SB is the new starting band (new 00 band), and HB is one-half the number of bands to be used. In addition to setting up the drum format, this summary order causes the head assembly to be positioned over band SB. (Note that SB + HB must be less than, or equal to, 50.) If the complete drum is to be used, SB will equal 00 and HB will equal 50. The format specified in this summary order will be in effect for the given drum until another drum-format order is given.

Before a drum read or write instruction is given, the programmer must be sure that the head assembly is positioned over the correct band. If the head is not correctly positioned, the programmer must first give one of the summary orders which move the head assembly. The next-lower-band summary order has the following format:

94 xxx DD xxxxx

where DD is the drum involved. This summary order causes the head assembly to be positioned over the next lower band. If the head assembly is already over band 00, this summary order causes it to be positioned over the highest numbered band. Both the 00 band and the highest numbered band are in accordance with the specifications in the most recently executed summary order 93 for drum DD.

The next-higher-band summary order has the following format:

95 xxx DD xxxxx

where DD is the drum number. This summary order causes the head assembly to be positioned over the next higher band on drum DD. If the head is already positioned over the highest numbered band, the order causes it to be positioned over band 00. Both band 00 and the highest numbered band are those specified by the most recently executed summary order 93 for drum DD.

The position-head summary order positions the head assembly of the specified drum over the indicated band (relative to SB as specified in the last summary order 93 from the drum). The format for this summary order is as follows:

96 x BB DD xxxxx

where BB is the band ($0 \leq BB \leq 2HB-1$) and DD is the drum.

The read summary order causes the words of a specified number of sectors to be read from the drum and transferred to sequential storage locations. The summary order has the following format:

8 SS NN DD MMMMM

where SS is the starting sector, NN is the number of sectors to be read, DD is the drum, and MMMMM is the first of the storage locations into which the words are to be transferred. To read a full band in minimum time, SSNN should equal 0025.

The write summary order has the following format:

9 SS NN DD MMMMM

where SS is the first of NN sectors to be written on drum DD. The words are written from sequential storage locations beginning with location MMMMM. To write a full band in minimum time, SSNN should equal 0025.

In order to prevent destruction of information which has been recorded on the drum, the programmer gives an interlock summary order, which prevents the execution of any future write summary orders addressed to the indicated drum. This summary order has the following format:

98 xxx DD xxxxx

where DD is the drum number.

To remove the drum write interlock and permit write operations on the interlocked drum, a remove-interlock summary order is given. The format for this summary order is as follows:

99 xxx DD xxxxx

where DD is the drum number.

The following table shows the gross timing for selected drum operations:

Operation	Time in Milliseconds
Drum head step or jog	50
Drum head reversal	10
Drum revolution	68
Drum connect to synchronizer	10
Full band read or write	80
Sector processing time	2.7

C-2. MAGNETIC TAPE SUMMARY ORDERS

The first summary order which is given when a new tape (or Uniservo) unit is used is one of the block-format orders. The fixed-block-length summary order has the following format:

63 K L x ST xx WWW

and indicates that the length of all blocks on the tape will be equal to WWW0 core-storage words. The tape number is indicated by digits ST, where S represents the synchronizer to which tape unit T is to be connected. If, in a write operation, the space between blocks (SBB) is to be one inch, L = 0; if the SBB is to be 2.4 inches, L = 1. In a read instruction, L should equal 0 to read a Larc computer-generated tape, or 1 to read a Univac computer-generated tape. The K digit indicates the type of translation to be performed. If K = 1, data read from the tape is translated into Larc computer one-digit numeric code and data written on the tape will be translated from this code. If K = 2, the Larc computer two-digit alphanumeric code is used in connection with the tape. The variable-block-length summary order has the following format:

64 KL x ST MMMMM

and means that the blocks on tape ST are (or will be) of different lengths (but always multiples of ten words). Following read operations in the variable mode, the address of the last location loaded, $\pm$ 1 (+1 for a 73 instruction, -1 for a 74 instruction), will be deposited in location MMMMM (MMMMM $\geq$ 05000). The format of the word in location MMMMM will be 0000000 (M' $\pm$ BBB0) where M' is the beginning location and BBB the limit specification in the read summary order. Digits K and L are interpreted as they are in summary order 63. Block-length summary orders are in effect for all other tape summary orders on a given tape until new block-length summary orders are given for the same tape.

After the tape format has been established, the tape must be properly positioned. If the tape is already in position, no summary order need be given for this; otherwise, one of the two position-summary orders should be issued. The position-forward order, has the following format:

71 CD x ST EEEEE

where EEEEE is the number of blocks through which tape ST is moved forward. If the programmer wishes to check the readability of the tape, he sets C = 0; otherwise, C = 1. A special positioning checker may be used for checking tape readability if the programmer needs to use the synchronizer to write on another tape concurrently with the checking operation. In this case, he sets D = 1; otherwise, D = 0, and synchronizer S is used for the checking operation during which it is not free for either a read or write operation. When two checking operations occur simultaneously, the longer operation should be assigned to the positioning checker, so that the other synchronizer performing the checking operation will sooner be free to perform read and write operations. (Note that the positioning checker should not be used to check-read a Univac I tape.)

The position-backward summary order has the following format:

72 CD x ST EEEEE

and moves tape ST backward through EEEEE blocks. Digits C and D are interpreted as they are in summary order 71.

The read summary orders are used to transfer data from tape into core storage. The read-forward summary order has the following format:

73 BBB ST MMMMM

In the fixed mode this order indicates that tape ST is to be read in a forward direction, and the data is to be stored in locations MMMMM to MMMMM + BBB0 - 1, inclusive. In the variable mode it indicates that the next block of data on ST is to be read into storage locations beginning with MMMMM. If the block extends beyond location MMMMM + BBB0 - 1, an error is indicated.

The read-backward summary order has the following format:

74 BBB ST MMMMM

This order is analogous to the read-forward summary order, but the tape is read in backward direction and the words are stored in locations MMMMM to MMMMM - BBB0 + 1.

The two write summary orders are used to transfer data from the core storage to tape. The write-density-200 summary order has the following format:

75 BBB ST MMMMM

and causes data to be written on tape at a density of 200 characters per inch. (On Larc computer, serial 2 it writes at a 250 character-per-inch density.) In the fixed mode, BBB0 words are written from storage locations MMMMM to MMMMM + BBB0 - 1. A space between blocks is inserted after each group of WWW0 words, as specified in the block-length summary order. In the variable mode, BBB0 words are written as one block.

The second-write summary order has the following format:

77 BBB ST MMMMM

This order is the same as summary order 75, except that the data is written on tape at a density of 100 characters per inch. (On Larc computer, serial 2, the density is 125 characters per inch.)

In order to rewind a tape, the programmer issues one of the rewind summary orders. These summary orders have formats as follows:

66 xxx ST xxxxx

and

68 xxx ST xxxxx

the 66 is a plain rewind order and the 68 is a rewind with interlock order.

The following table shows the gross timing for tape operations:

Operation	Time in Milliseconds
First block delay	1800
Tape reversal	800
Servo setup	10
Space between blocks:	
(a) 1 inch	10
(b) 2.4 inches	24
Process 10 words on tape:	
(a) 104 characters per inch	11.52
(b) 125 characters per inch	9.6
(c) 208 characters per inch	5.76
(d) 250 characters per inch	4.8

C-3. LINE PRINTER SUMMARY ORDERS

The page format on the line printer can be established in a summary order which has the following format:

43 PP S SL xxx FL

where S is the printer number (5 or 6 for printer 1 or 2, respectively), PP is the number of lines on each page (normal size paper is 66 lines), SL the number of the line to which each page is advanced before any printing occurs, and FL the last line of printing on the page. FL must be greater than SL. This summary order provides for automatic paper advance for the situations when less than full pages of printing is desired. If the programmer on occasion wants to print above line SL, he must then use summary order 42 below, to position the paper. The processor program does not permit any printing below line FL.

The advance-paper summary order has the following format:

42 LL S xx xxxx C

If C = 0, the paper on printer S is advanced LL lines. If C = 1, the paper is advanced to the top of the next page, then advanced LL lines.

The format for the print summary order is as follows:

40 NN SQ P MMMMM

The summary order causes a number of words of data to be printed on printer number S. The words to be printed are those in storage locations beginning with MMMMM. The paper is advanced Q lines (Q > 0) before each line is printed. (Q = 1 for single spacing, Q = 2 for double spacing, etc.) Digit P indicates one of the following print modes:

1 = Numeric edited

2 = Numeric unedited

3 = Alphanumeric edited

4 = Alphanumeric unedited

The number of words that are printed is a multiple of the number of words per line (n_p) which is fixed for each mode according to the following table:

P	n_p
1	11
2	10
3	22
4	10

The number of lines, K, (and, consequently, the total number of words) which will be printed can be calculated from the following relationships: $K \times n_p \leq NN \times 10$ and $(K + 1) \times n_p > NN \times 10$. The first relationship does not hold in the case where $n_p = 11$ and NN = 1. In this case, K will equal 1 and $K \times n_p$ will be greater than NN x 10.

The following table shows gross timings for on-line printer operations:

Operation	Time in Milliseconds
Print one line:	
(a) numeric	80
(b) alphanumeric	100
Advance paper one line	10

C-4. ELECTRONIC PAGE RECORDER SUMMARY ORDERS

In the following summary orders for the electronic page recorder, S, Z, L, X, and Y must be expressed as a number of points in a 1000 x 1000-point mesh. One normal line spacing is equal to 15 points.

The connect summary order must be given before the electronic page recorder can be used. This order has the following format:

59 xxxx C xxxxx

It causes recorder C to be connected to the synchronizer, and advances the film.

If the programmer wants to alter the set spacing, he can execute a line-spacing summary order which has the following format:

52 SSS xxxx ZZZ

where SSS is the number of spaces the beam is to be moved immediately and ZZZ is the number of spaces to be left between succeeding lines or groups of lines. ZZZ is counted from the first line of one group to the first line of the next group. If S = 0, no immediate spacing is executed; if Z = 0, there is no change in the spacing. The spacing indicated by Z is in effect until another summary order 52 is given.

The position summary order has the following format:

53 x LLL XXX YYY

It causes the beam to be positioned at point XXX, YYY before each frame is displayed. In addition, it indicates that the display is to be terminated after line LLL (LLL < YYY) has been recorded, and at that point the film is to be advanced and the beam repositioned at XXX, YYY. This summary order is in effect for all print orders until a new 53 order is given. The check to determine whether line LLL has been reached is made after each group of W x 10 words (see summary order 50 following) has been displayed. Therefore, if W > 1, it is possible to record beyond line LLL.

If the display is to be in the plotting modes, the plotting character must be selected. This is done in the select-plotting-character summary order, which has the following format:

56 xxxxxxxx KK

where KK is the character to be used. Character KK, which must be alphanumeric, is used for all plotting until a new plotting character is specified.

The summary order for printing or plotting has the following format:

50 BBB W P MMMMM

where BBB x 10 words are to be printed or plotted, beginning with the word in location MMMMM. If the display is printed rather than plotted, it appears in groups of W x 10 words. (In the alphanumeric modes, a word is 12 digits long as a data word, but only 6 characters long as a printed word.) For the plotting modes, W must equal 0. The mode for the display is indicated by P, where P can have one of the following values and meanings:

1 = Numeric edited mode

2 = Numeric unedited mode

3 = Alphanumeric edited mode

4 = Alphanumeric unedited mode

5 = Graphing with the X and Y coordinates for two points contained in each data word. The format of the data word is XXXYYYX'X'Y'Y'Y'.

6 = Graphing with the X and Y coordinates for a single point contained in two successive data words. The format for a pair of data words is xxxXXXxxxxxx and xxxYYYxxxxxx.

7 = Plotting vertical grid lines with two abscissas for two full-length vertical grid lines contained in each data word. The data words are in the format, VVVxxxV'V'V'xxx.

8 = Plotting horizontal grid lines with two ordinates for two full-length horizontal grid lines contained in each data word. The data words are in the format, xxxHHHxxxH'H'H'.

The contents of each line in a group of words being printed are dependent upon the mode. In the edited modes, the characters are printed one after another for a maximum of 125 characters to a line. A word can be broken between the end of one line and the beginning of the next. Spacing is inserted only as indicated in the data words. If an end-of-line symbol is reached, the program will adjust its counters as if it had completed printing a block of ten words. This must be kept in mind when specifying W in the alphanumeric edited mode. Note that to allow the printing of a full line (twenty 6-character words) in the alphanumeric edited mode, W must equal 2. Within a group of W x 10 words, the line spacing is the normal 15 points to a line. However, between groups of words the spacing is determined by ZZZ as specified in the last summary order 52. Lines within

a group are printed starting at the left margin (X = 0); however, the first line of a group is indented to the value of X specified in the last summary order 53.

The print-or-plot-and-advance summary order has the following format:

51 BBB W P MMMMM

This summary order is the same as summary order 50 except that after the display the film is advanced and the beam repositioned to XXX, YYY as specified in the last summary order 53.

To open the shutter of the Polaroid Land camera, the programmer gives the open-shutter summary order, 57 xxxxx xxxxx.

For the data to be recorded on the film, this summary order must be given before the print or plot summary order for the desired data is given. The close-shutter summary order, in the format 58 xxxxx xxxxx is given after the print or plot summary order. To operate the Polaroid camera, the programmer need only surround the appropriate print or plot order with the open- and close-shutter orders. The timing problems are handled by the processor program. A printout on the console printer notifies the operator to pull the developing tab on the camera.

The following table shows the gross timing for the electronic page recorder operations:

Operation	Time in Milliseconds
Camera movements:	
(a) Advance film (35 mm)	100
(b) Open shutter (Polaroid)	110
(c) Close shutter (Polaroid)	110
Print 10 words:	
(a) Modes 1 and 2 (120 characters)	9
(b) Modes 3 and 4 (60 characters)	4.5
Plot 10 words:	
(a) Mode 5 (20 points)	4
(b) Mode 6 (5 points)	1
(c) Modes 7 and 8 (20 grid lines)	12

C-5. MISCELLANEOUS SUMMARY ORDERS

If the programmer wants to prevent the computer program from writing in a given core-storage unit, he executes a storage-unit-write-interlock summary order. This summary order has the following format:

08 xxx xx MMMMM

where MMMMM is any storage location in the 2500-word storage unit to be interlocked. To remove this interlock, the programmer executes a remove-interlock summary order with the following format:

09 xxxxx MMMMM

If the storage location in summary order 09 is in the same storage unit as the processor program, the operator will be notified to this effect by a printout on the console printer.

A storage-interlock summary order prevents the execution of any summary order which would alter the contents of a specified storage area. This summary order has the following format:

28 NNNNN MMMMM

where NNNNN is the first location and MMMMM the last location of the protected area. If the processor program receives a summary order which would alter the specified area, it records in location 02799 a transfer of control to NNNNN and sets the processor intervention flip-flop. (This assumes that the processor intervention subroutine in the contingency routine will include a control transfer to location 02799, and that location 02799 will contain instructions to handle the conflict situation.) At any one time, only one storage area may be interlocked by a summary order 28. In addition, this interlock can be removed only by issuing another summary order 28. If the programmer wants to remove this interlock without interlocking another storage area, he can address a summary order 28 to an area occupied by the processor program. Because this area is interlocked anyhow, the programmer will not be affecting usable storage.

To instruct a console printer to print, the programmer issues a summary order in the following format:

20 NN D W P MMMMM

This summary order causes a carriage return on printer D, followed by a printout of NN words of data beginning with the word stored in location MMMMM. For the engineer's console printer, D = 1; for the operator's console printer, D = 2. The printing is done in mode P, with P equal to one of the following:

1 = Numeric edited mode

2 = Numeric unedited mode

3 = Alphanumeric edited mode

After every W words (W = 1 through 9) a carriage return is executed, except that in mode 3 carriage returns are indicated in the data words. Note that

100 milliseconds is required to print one character on the console printer. Carriage return time may take as long as 1/2 to 1 second.

If either numeric mode is specified, the processor program translates each digit of the data words into the two-digit code required by the printer. In mode 1, the printer prints digits 0 through 9 and the symbols (+), (-), and (.). The code for space (^) and ignore (\) are translated into the codes for the corresponding printer actions. Mode 2 differs from mode 1 only in that the codes for space and ignore cause the printing of the appropriate character rather than a printer action. In mode 3 the data must be prepared in the two-digit code listed in table C-1. The only printer function codes which may appear in the data in mode 3 are 35 (carriage return), 55 (tabulate), 15 (ignore), and 16 (space).

If, using the processor program, the programmer wants information displayed in the visual display registers, he uses the visual-display summary order,* which has the following format:

10 C x TTT MMMMM

If the display is to be in the 12-digit display register, C = 1, and MMMMM is the location of the word to be displayed. If the 5-digit display register is to be used, C = 2. In this case, the display will consist of the contents of the five least-significant-digit positions of MMMMM. Digits TTT indicate the number of seconds the information is to be displayed.

The Computing Unit programmer can time an operation through the use of the time-limit summary order which has the following format:

03 TTTTT MMMMM

This summary order causes a .900000 MMMMM instruction to be placed in location 02799 and the processor intervention flip-flop to be set TTTTT seconds after the summary order is executed. The programmer should include in the contingency routine a transfer to 02799 when the processor intervention flip-flop is found set. Control will then transfer from 02799 to MMMMM. Location MMMMM should contain the first instruction required after the timed interval has elapsed.

A stop summary order is given when the Computing Unit program terminates a certain phase of operation. The stop summary order has the following format:

06 S xxxx MMMMM

Normally, when the computer is ready to switch to a new problem, it would issue this summary order with S = 0. The processor would then stop accepting summary orders but would complete those already accepted.

* This summary order unusable with the processor program supplied with Larc System, Serial 2.

Table C-1. Alphanumeric Code for the Console Printer

Two-Digit Code	Printer Symbol or Action	Two-Digit Code	Printer Symbol or Action
2.	.	53	ρ
2-	-	54	π
2∧	∧	55	TAB
2+	+	56	]
2\	\	57	[
15	IG	60	)
16	SP	61	J
17	-	62	K
20	0	63	L
21	1	64	M
22	2	65	N
23	3	66	O
24	4	67	P
25	5	68	Q
26	6	69	R
27	7	72	>
28	8	73	*
29	9	74	→
32	=	75	Σ
33	~	76	η
34	(	77	:
35	CR	80	+
36	,	81	/
37	.	82	S
40	∇	83	T
41	A	84	U
42	B	85	V
43	C	86	W
44	D	87	X
45	E	88	Y
46	F	89	Z
47	G	92	Δ
48	H	93	λ
49	I	94	□
52	<		

SP = Space
IG = Ignore
CR = Carriage return
TAB = Tabulate

Computer control would be transferred (through the contingency routine and location 02799) to location MMMMM, and the processor would begin again to accept summary orders. Location MMMMM would be the beginning of the new problem.

In case of uncorrectable difficulty in the Computing Unit and/or processor program, S should be set to 1 in the stop summary order. The processor will then stop executing summary orders and will transfer the contents of storage to drum 1, bands 10 through 15 and 84 through 89. (This information is thus available for analysis, if desired.) The processor hardware is set to initial conditions, and a new copy of the processor program is transferred into core storage. Computer control will be transferred through the contingency routine and location 02799 to location MMMMM. Location MMMMM would be a rerun point.

If a word with zeros in the first two digit positions occurs in a summary order packet, it will be considered a skip order and will be ignored.