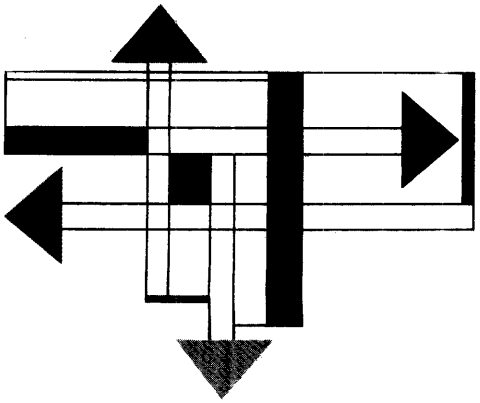
PROGRAMMER'S
REFERENCE
MANUAL

The Sylvania 9400 Data Processing System

An Important New Concept in Data Processing



SYLVANIA 9400
DATA PROCESSING SYSTEM

PROGRAMMER'S REFERENCE MANUAL

(REV. OCTOBER 1961)

SYLVANIA ELECTRONIC SYSTEMS

A Division of Sylvania Electric Products Inc.

DATA SYSTEMS OPERATIONS

189 B Street, Needham 94, Massachusetts

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

Section — Page

Section Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF APPENDICES

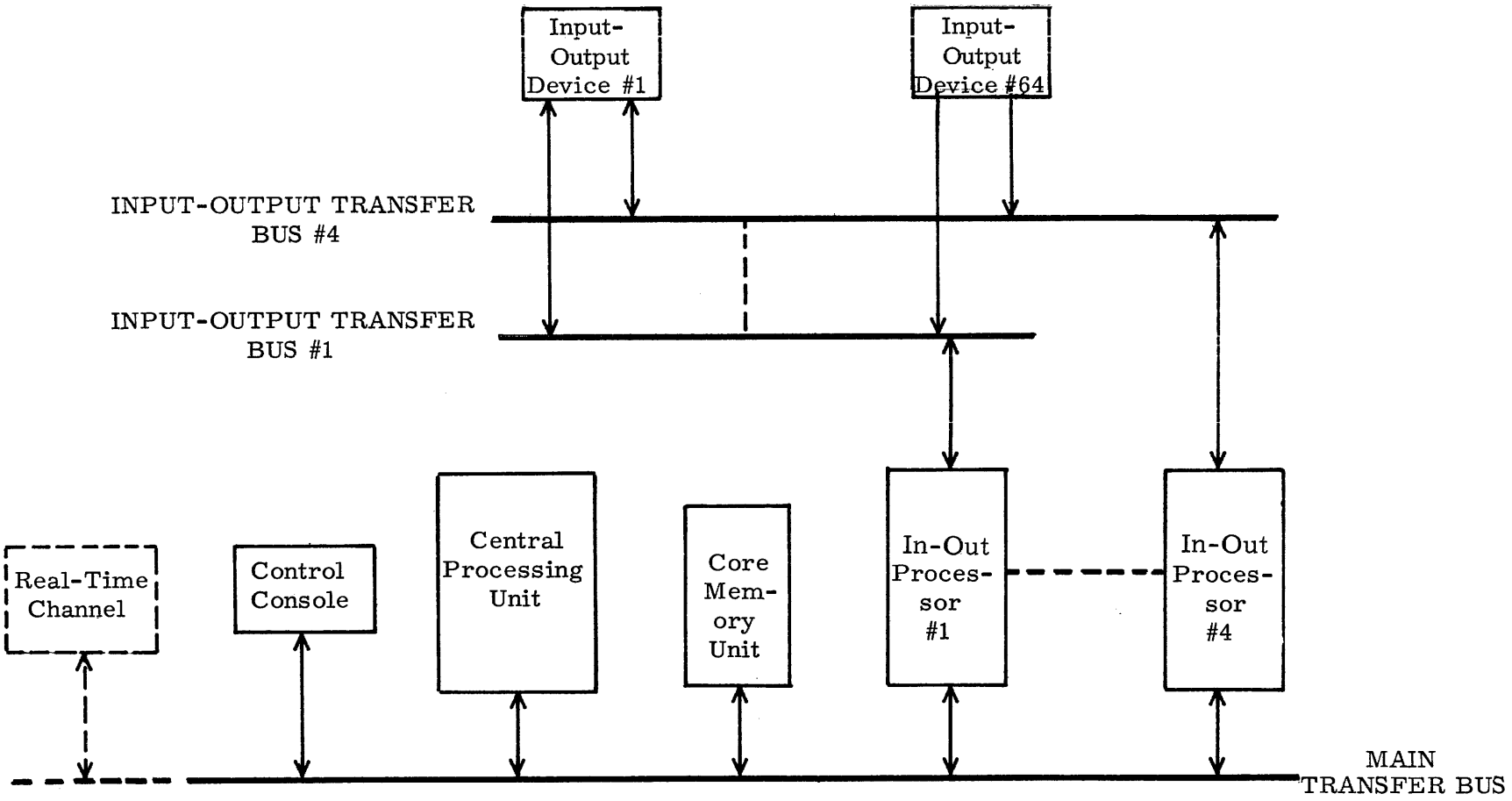


Figure I-1. Sylvania 9400 Data Processing System, Block Diagram

SECTION I
INTRODUCTION

The Sylvania 9400 System is a large-scale, general-purpose fully-transistorized digital computer system. It is characterized by high internal speed, simultaneous operations, and a real-time capability. Computer words are stored or retrieved from random locations in high-speed memory in four millionths of a second. This memory speed in conjunction with the two megacycle computer timing clock enables a basic operation rate of 125,000 instructions per second.

A representative system includes a Central Processor, a High-Speed Memory, from one to four Input-Output Processors and a wide variety of input-output devices — 64 mutually shared by all Input-Output Processors or a maximum of 256 devices. Each of the Input-Output Processors may operate simultaneously with each other and the Central Processor. Thus, the Central Processor delegates time consuming input-output operations to the Input-Output Processors, operating in parallel, while it performs the higher speed computations and logical functions.

The multiple processor approach forms the basis for an unusual degree of system expandability. For example, by expanding memory, increasing the number of Input-Output processors and peripheral devices — processing capacity can be increased several fold. This modular expansion concept eliminates the need for acquiring additional computers or the expensive exchange of an existing machine for a larger one as the application load expands.

An important feature of the 9400 system is the optional Real-Time System which provides the Central Processor with the ability to receive and read-out data from remote locations at high speed, and without costly and time consuming conversion operations such as required for punched paper tape and cards. Real-time operations also are performed simultaneously with other Input-Output and Central Processor operations. The Real-Time System commands top priority access to memory. This facility is an important consideration in large-scale integrated data processing systems.

Information to be processed is handled internally as groups of *binary digits*. A single binary bit may assume one of two states, ZERO or ONE. (Refer to Appendix A, NUMBERING SYSTEMS.)

The two-state characteristic inherent in most electronic and magnetic devices renders the binary numbering system the most feasible means of representing information within a digital computer. A group of bits of specific size inside the 9400 computer is designated as a computer word. Each word consists of 37 bits, which may be interpreted by the computer as a binary number, a set of alphabetic or numeric characters, or as a series of special symbols. In magnetic core storage, the 9400 word has attached to it an additional bit for parity checking purposes.

A block diagram of the Sylvania 9400 Data Processing System is shown in Figure I-1. Information transfer within the system takes place along transfer busses, as shown. The Main Transfer Bus connects the Central Processing Unit (CPU), the Magnetic Core Memory Unit, the Console, and the In-Out Processors. The Input-Output Transfer Busses connect the In-Out Processors to the Input-Output Devices.

The Central Processing Unit performs all the arithmetic and logical operations of the computer. The CPU also initiates input-output operations, which then proceed independently from the Central Processor.

The Magnetic Core Memory Unit provides rapid-access data and instruction storage for both the Central Processor and the In-Out Processors. Core Memory Units are made up of 37-bit word cells, each one with a unique address. Each word in Memory is available for processing in four microseconds. Appended to each memory cell is a 38th bit, used in checking accuracy during transfer of information into or out of core memory. Core Memory Units have a capacity of 32,736 words.

The 9400 System can utilize up to four In-Out Processors. Each In-Out Processor communicates with the Central Processor and the rest of the "central" system through the Main Transfer Bus. Each input-output device (card reader, magnetic tape drive) is connected to the In-Out Processors by an Input-Output Transfer Bus. As many input-output devices may be in operation simultaneously as there are In-Out Processors in the System.

The overall 9400 System is operated and monitored automatically at the Control Console. The System may be operated manually from the Console or monitored during automatic operation. In addition, it is possible for the operator to examine the contents of any addressable storage location.

In its automatic mode of operation, the 9400 System is controlled by a program stored in Core Memory. The program is made up of a set of *instruction* words

and *data* words. The Central Processor retrieves an instruction word from Memory, interrogates it through a decoding process, and causes the 9400 System to perform the operations indicated by it. As the *current* instruction is being executed, the Central Processor retrieves the next instruction word from Memory and prepares to execute it in sequence. A counter keeps track of the locations of the instruction words which are to be executed. The computer is said to be *automatically sequenced.* If an instruction calls for operation upon data, the Central Processor, in a process similar to the retrieval of an instruction word, but at a different time, retrieves the required data word from Core Memory and operates upon it. The data is selected according to the address accompanying the instruction.

The instruction repertoire of the 9400 System may be divided into two major categories; Central Processor operations, and Input-Output operations. Central Processor operations pertain in general to the processing and interrogation of data contained within the Central Processor (and core memory) itself. Input-Output operations, on the other hand, bring about the exchange of data and control information between the Magnetic Core Memory Unit (and Central Processor) and the input-output devices, through the In-Out Processors. The design of the 9400 System is such that Central Processor operations are carried out simultaneously with the execution of input-output operations, a factor greatly enhancing the speed of the overall system.

Although the 9400 System operates upon data and instructions in binary form only, the programmer will normally communicate with the computer through a symbolic language which greatly simplifies the preparation of computer programs. A packaged routine, the 9400 Assembly Program (94AP), automatically translates the programmer's symbolic notation into machine language. At the same time, the 94AP checks the symbolic program for certain format and instruction errors. See Section VII, SYMBOLIC PROGRAMMING.

SECTION II
MAGNETIC CORE STORAGE UNIT

The basic storage element of the Magnetic Core Storage Unit is a small (approximately 2 millimeters in diameter) magnetizable toroid, or core. Each core is capable of storing one binary digit. When the core is magnetized in one direction, it is considered to be storing a binary ZERO; when it is magnetized in the opposing direction, it is considered to contain a binary ONE. Thus, an array of thirty-seven cores is capable of storing a complete 9400 word. The Magnetic Core Storage Units (also referred to as Core Memories) are made up of 37-bit cells, each holding a single computer word. Appended to each cell is a 38th bit, which is used for checking the accuracy of transfer into and out of the memory unit.

The 9400 System operates with a Magnetic Core Storage Unit with a capacity of 32,736 words. In both units, each word has a unique address, providing the programmer with direct access to a specific word.

A block diagram of a Magnetic Core Storage Unit is shown in Figure II-1. The Unit communicates directly with the Central Processing Unit, the Console, each In-Out Processor, and the optional Real-Time Channel through the Main Transfer Bus (see Figure I-1). Information is stored in the Memory Array in the form of word cells. Data and instruction words are read from and written into individual Memory Cells through a buffering register referred to as the Memory In-Out Register (MO).

Inherent to the transfer of words to and from Core Memory is the verification of the accuracy of transfer by means of *parity checking*. A parity bit is added to each word stored in Memory such that the total number of ONE bits in the word, including the parity bit, is *odd*. For example, if the 37-bit word to be stored in Memory contains an even number of ONEs, the parity bit is automatically made a ONE and stored with the word. When the word is at some later time read from Memory, the computed parity bit is read out with it. At the same time, a new parity bit is calculated on the basis of the contents of the 37-bit word. The two parity bits are then compared; if they do not agree, the Memory Parity Error (MPE) alarm is turned on, indicating a malfunction within the Magnetic Core Storage Unit.

All thirty-seven bits and the accompanying parity check bit are transferred in parallel between the code array, and the Memory In-Out Register. The MO register is connected to the Main Transfer Bus and hence to the rest of the 9400 System. The transfer of words between the MO register and the rest of the 9400 System is in the form of 37-bit words.

Words in Memory are selected according to an address stored in the Memory Address Register (MA). The capacity of the Memory Address Register is fifteen bits, which enables it to contain the binary equivalent of a number which is 32 greater than the highest-numbered memory address. The last thirty-two addresses are reserved for addressable registers which are described in Section III. The Memory Address Register receives the Memory Cell Address through the Main Transfer Bus. The Memory Cell Address provides the coordinates for locating a word in Memory.

Words are read from and written into a Magnetic Core Memory Unit by a fixed sequence of events called a *memory cycle*. The memory cycle consists of a read half-cycle and a write half-cycle, each of which is accomplished in four microseconds.

The read half-cycle copies the contents of the Memory Cell specified by the contents of MA into the Memory In-Out Register. The parity check bit is copied into MO at the same time. The copying process destroys the contents of the addressed cell by resetting all bit locations to ZERO. During the read half-cycle, the word read from Memory is checked by the parity circuitry for accurate transfer.

The write half-cycle copies the contents of the Memory In-Out Register into the addressed Memory Cell. A parity bit is computed automatically and transferred, with the word, into the appropriate memory location. The contents of MO, however, are not destroyed by the transfer. A write half-cycle always occurs in conjunction with and following a read half-cycle.

Retrieval of a word from Core Memory takes place in the following manner: The Memory Address Register receives an address from the System; a read half-cycle is initiated. The contents of the addressed memory location are transferred into the Memory In-Out Register, leaving the cell in Memory blank (all ZEROes). Parity is checked. During the following write half-cycle, the contents of MO are replaced in Memory, together with the corresponding parity bit. Simultaneously with the write half-cycle, the contents of the Memory In-Out Register are also duplicated elsewhere in the 9400 System, as required by the in-

struction being executed. At the completion of an ordinary retrieval operation, where information is being read from Memory, the contents of the addressed cell in Core Storage are the same as prior to the start of the read-write cycle.

Storing of a word into Core Memory is similar to retrieval of a word from Core Memory. The only difference is that as soon as the original word has been read from the addressed Memory Cell into MO, the new word which is to be written into Memory is transferred into MO, destroying the newly-acquired word from Memory. Thus, the write half-cycle, which inevitably follows the read half-cycle, copies the new word into Memory rather than the one originally retrieved. A new parity bit is computed automatically and stored with the new word.

9400 Magnetic Core Storage Units provide *random access* storage, in that any word in Memory may be read out in four microseconds, regardless of its cell address.

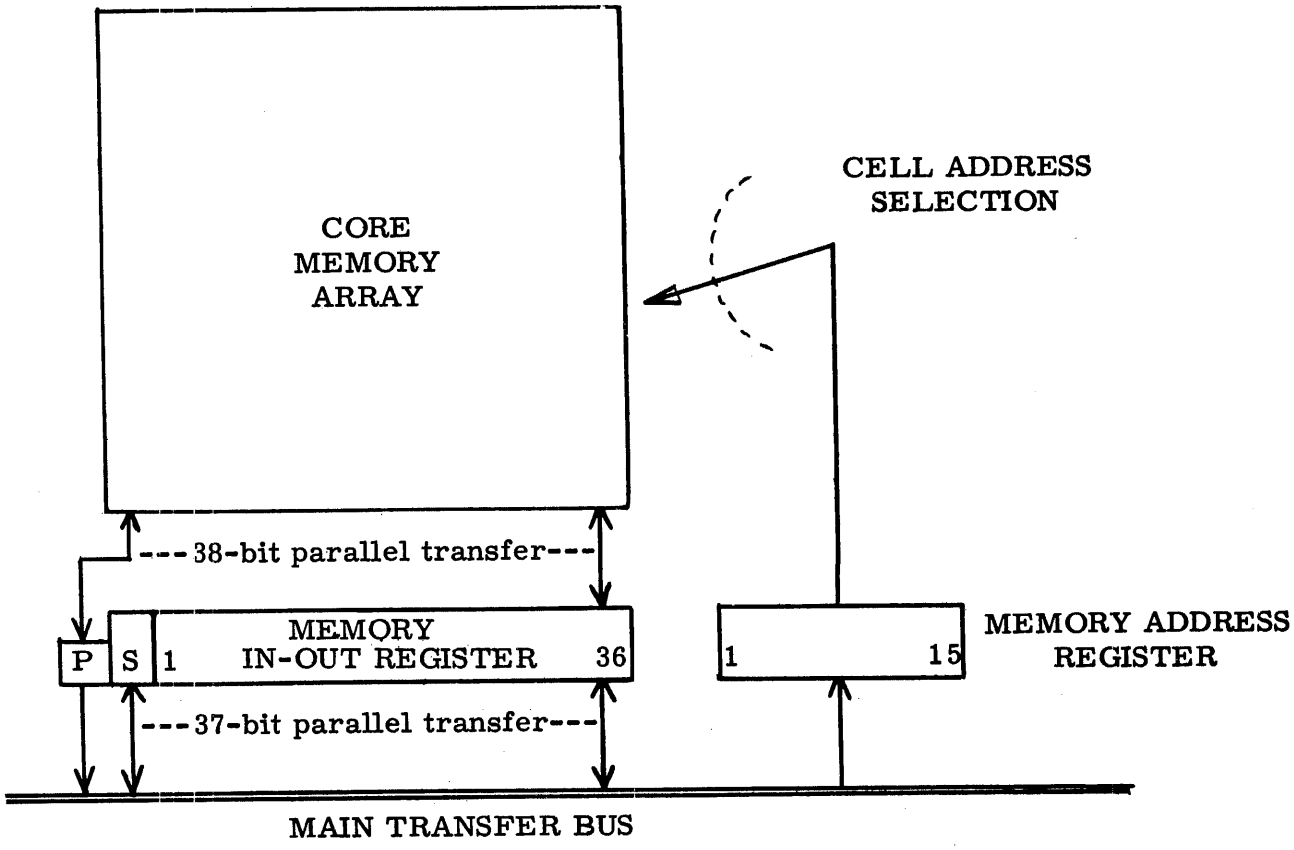


Figure II-1. Magnetic Core Storage Unit, Block Diagram

SECTION III
CENTRAL PROCESSING UNIT

GENERAL DESCRIPTION

A generalized block diagram of the Central Processing Unit (CPU) is shown in Figure III-1. The CPU is made up of two major functional parts: the Arithmetic Section and the Control Section. Each section communicates with the rest of the 9400 through the Main Transfer Bus (see also Figure I-1). In addition, certain control signals are exchanged directly between the Arithmetic Section and the Control Section.

The Control Section of the Central Processing Unit decodes and implements all computer instructions. Central Processor instructions are retrieved from the Magnetic Core Storage Unit and implemented within the Control Section of the CPU. The Control Section initiates and controls the various operations which are carried out in the Arithmetic Section.

The Control Section also retrieves and decodes input-output instructions. It initiates input-output operations, which then proceed independently from, and asynchronously with respect to, the Central Processing Unit.

The Arithmetic Section of the Central Processing Unit performs all arithmetic and related operations upon data words. The Arithmetic Section is essentially under the control of the Control Section. For certain machine instructions, however, the data words being processed within the Arithmetic Section dictate to some extent the operation of the Central Processor during the execution of these instructions. Explanations of the operation of the individual "data-dependent" instructions are given under Central PROCESSOR INSTRUCTIONS.

INTERPRETATION OF WORDS FROM MEMORY

Words stored in the Magnetic Core Storage Unit are merely strings of thirty-seven bits; an instruction is indistinguishable from a data word. The Central Processing Unit, which decodes and implements instruction words, is only able to differentiate between data words and instructions by virtue of the point in time at which they are retrieved from Core Storage.

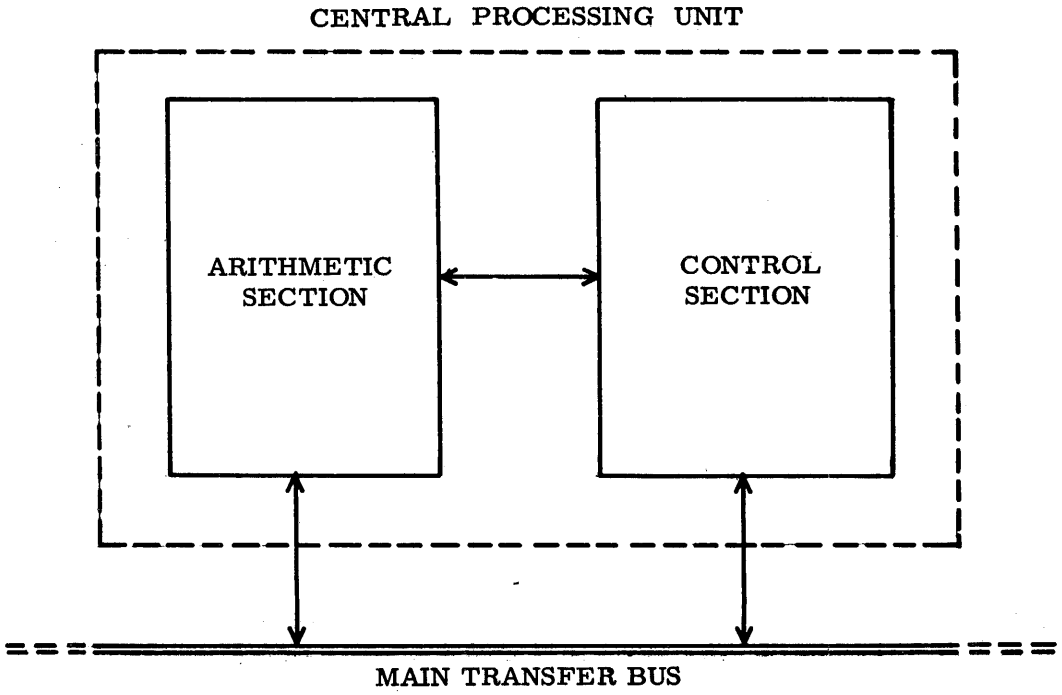


Figure III-1. Central Processing Unit, Block Diagram

The Central Processing Unit operates under the supervision of a basic machine cycle (see also INSTRUCTION TIMING AND THE BASIC MACHINE CYCLE). Instruction words are always retrieved during one specific portion of the cycle, as are data words during a different portion. Thus, words retrieved during the instruction "time" are sent to and interpreted by the Control Section of the Central Processor, and words retrieved during the data "time" are, in general, sent to the Arithmetic Section for processing.

Through proper use of programming techniques, it is possible, and often desirable, to treat instruction words as data and to operate upon them accordingly. Also, erroneous programming can cause data words to be processed as instructions and result in incorrect program operation.

CENTRAL PROCESSOR INSTRUCTION WORD FORMAT

Figure III-2 shows the Central Processor instruction word format. A 9400 word has the significance of an instruction only when it has been retrieved from Memory and transferred into the Control Section of the Central Processing Unit.

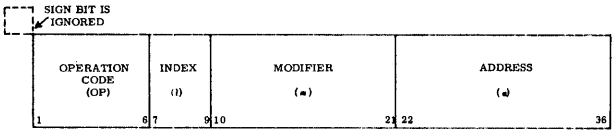


Figure III-2. Central Processor Instruction Word Format

The *sign bit* (*S*), has no significance in an *instruction* word and is ignored by the computer. The programmer may, if desired, use the sign bit of an instruction word for "tagging" purposes. Since the instruction repertoire of the 9400 System is likely to be increased in the future by including the sign bit in the operation code, extensive use of sign-bit tagging is not recommended.

The *operation code* (OP) occupies bits 1 through 6 of the word. The operation code for ADD, for example, is 001010, or 12 in octal notation.

The *first operand address* (*a*) is contained in bits 22 through 36. The number contained in the *a* portion of an instruction word is the address of a word located in Core Storage or an addressable register. In shifting and cycling operations, the number in *a* is used as a count rather than as an address.

The *modifier* (*m*) portion of the instruction word ranges from bit 10 through bit 21. In certain 9400 operations, it is possible to alter the address portions of instructions or the contents of specified registers. The modifier is involved in such operations. It is also used in controlling overflow and trapping operations (see under the appropriate headings).

The *index* (*i*) portion of the instruction word occupies bits 7 through 9. The number contained in *i* specifies a particular index register associated with the operation to be performed. Index registers enable modification of instruction addresses.

In some Central Processor instructions, the *i* and *m* portions of the instruction word are combined to form a second operand address.

The details of indexing, address modification, and the significance of first and second operand addresses are discussed in the following paragraphs and in the individual descriptions of the instructions themselves.

CONTROL SECTION AND CONTROL REGISTERS

The Control Section monitors and controls the operations of the Central Processing Unit. It also governs the Core Memory read-write cycle and causes the computer to sequence automatically through a program stored in the Core Memory. Figure III-3 is a block diagram of the Control Section of the Central Processing Unit. The Control Section consists basically of a group of registers of various lengths, together with appropriate decoding and control logic circuitry. All Control Section registers are connected to the rest of the 9400 System through the Main Transfer Bus. The Control Section mechanizes the basic machine cycle and implements all Central Processor instructions. In addition, the Control Section initiates all input-output operations. When an input-output device requires access to the Magnetic Core Storage Unit, the Control Section synchronizes the in-out memory cycle with the basic machine cycle.

The *Instruction Register* (IR) contains the 6-bit operation code for the instruction to be executed. The *Decoder Register* (D) receives and decodes the operation code which activates the Central Processing Unit.

The *G-Register* is a 3-bit register, and holds the number of the Index Register, if any, involved with the instruction to be executed (see INDEX REGISTERS). The *X-Register,* which has a capacity of 12 bits, normally contains the modifier portion of the instruction word. For instructions requiring a second operand address, the X and G-Registers function as a single 15-bit address register.

The *Address Register* (AR) is a 15-bit register, and contains the first operand address of the instruction to be executed. The *T-register* is a 9-bit counting register which keeps a count of shifting operations. It also controls the timing of instructions which are

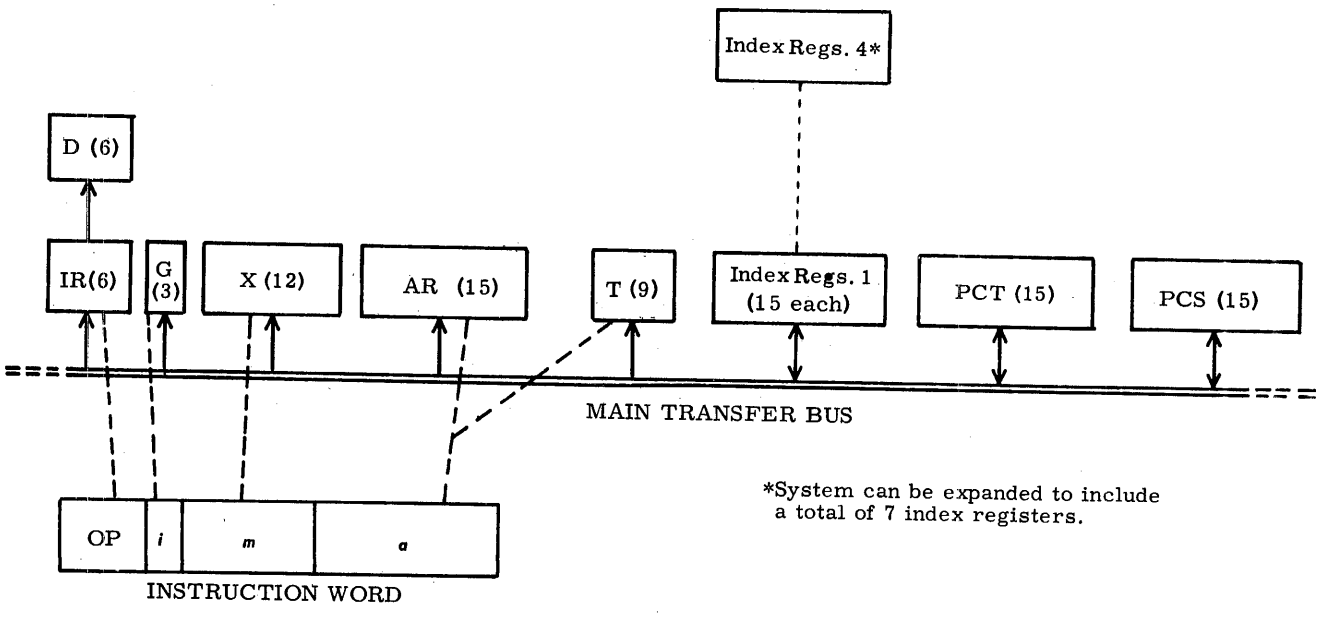


Figure III-3. Central Processing Unit, Control Section, Block Diagram

extended beyond the normal, basic machine cycle length. The T-register is also used to temporarily store the characteristic of floating point instructions.

The *Program Counter* (PCT) is a 15-bit register which contains the address of the next instruction to be retrieved from Memory. Except for transfer-type instructions, the Program Counter is advanced by a count of ONE during the execution of each instruction. The *Program Counter Store Register* (PCS) is a 15-bit register used in conjunction with two special transfer instructions, TRL and TRS, and its function is described with the description of the instructions themselves.

INDEX REGISTERS AND ADDRESS MODIFICATION

It is often desirable in programming to be able to modify the address portion of an instruction so that the same instruction may be used repeatedly to operate upon a series of data words. For this purpose, the Central Processing Unit is equipped with up to seven Index Registers. Functionally, Index Registers are part of the Control Section of the Central Processing Unit. However, because of their special characteristics and use, they are treated separately here.

An Index Register consists of a 15-bit, unsigned storage cell. It is addressable in the same manner that Core Storage Cells are addressable. In addition, for address modification purposes, Index Registers are "addressable" by the index (i) portion of an instruction word.

The instruction word shown in Figure III-4, contains an ADD operation code with an a portion of 5. This operation normally adds the contents of Core Storage Cell number 5 to the contents of an arithmetic register in the Arithmetic Section of the Central Processor. However, in the case illustrated, the i portion of the ADD instruction word contains a 2. A number in the index portion, or tag, of an instruction word indicates that address modification is to be performed. As the ADD instruction is decoded by the computer, the contents of Index Register 2 (in this case it contains a 3) are added to the address obtained from the a portion of the instruction word itself. As a result, an *effective address* of $(8)_{10}$ is formed in the Address Register of the Control Section. It is the contents of Core Storage Cell $(8)_{10}$ which are actually added to the number in the Arithmetic Section.

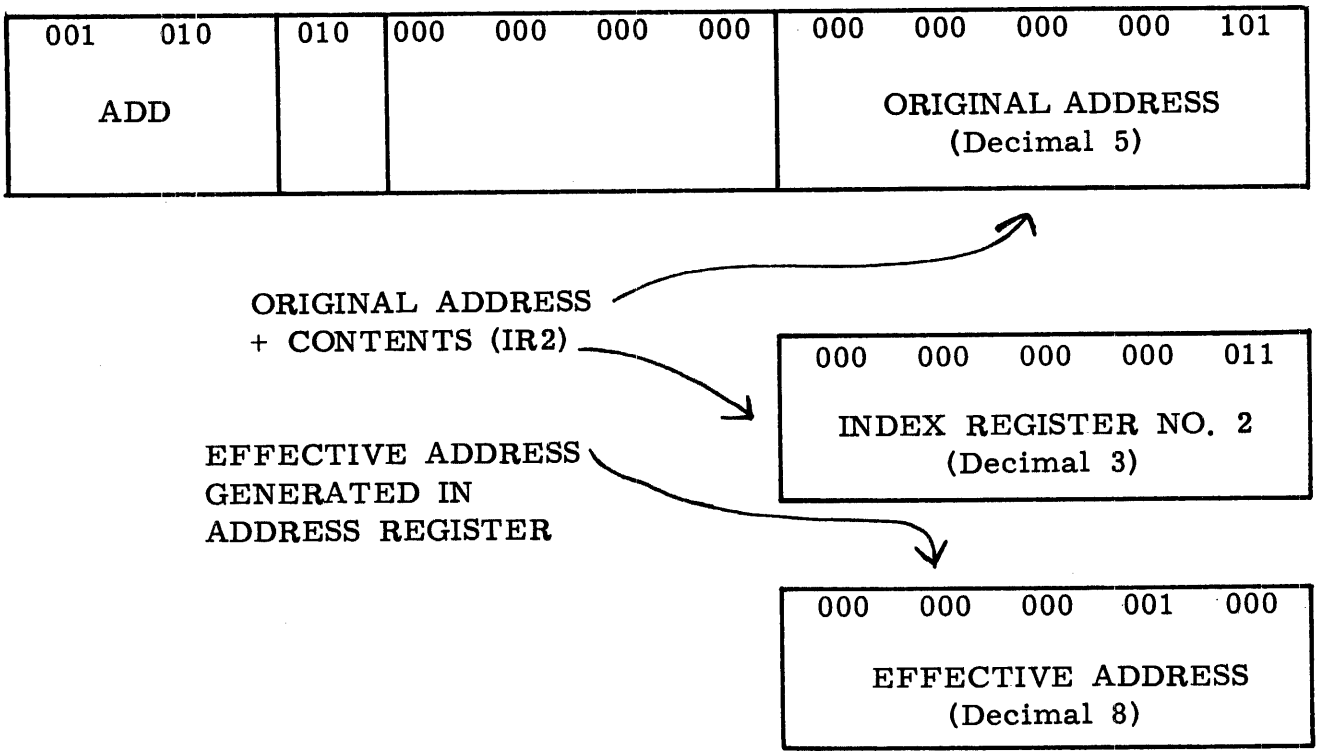


Figure III-4. Address Modification

Although tagging produces an effective address as far as execution of the tagged instruction is concerned, the original *a* portion of the instruction word is not changed. A program containing indexed instruction words is not modified except temporarily during its execution. In other words, even though the above ADD instruction word actually produced the addition of the contents of Core Storage Cell number 8, the instruction word in Memory still contains an *a* portion of 5.

For certain instructions, the contents of an Index Register may modify the second operand address in a manner similar to that for the first operand address. Specific applications for Index Registers are described under the individual instructions involved.

DATA WORD FORMATS

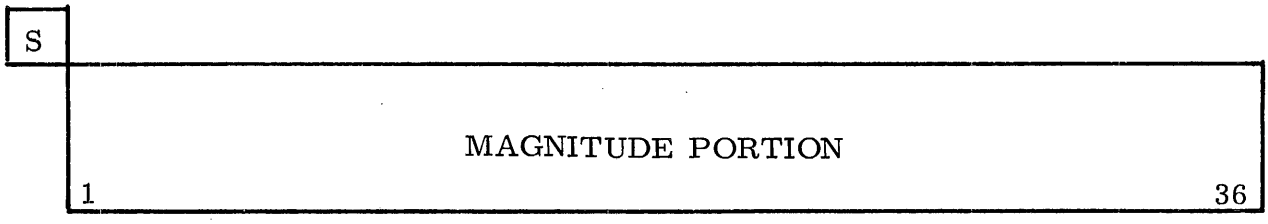
Although a 9400 word always consists of 37 bits, the computer — during the "data" portion of its basic machine cycle — interprets the 37-bit words in four different manners, depending upon the mode in which it is operating. Figure III-5 shows the four data word formats.

Data words are words which contain information to be processed. A data word contains either purely numeric information or alphanumeric information composed of both numeric digits and alphanumeric characters. Purely numeric information may be stored in memory as a conventional numerical data word or a floating point data word.

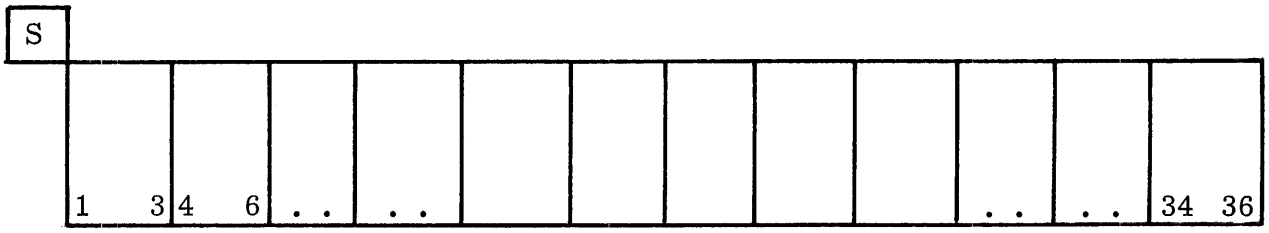
Part A of Figure III-5 illustrates a "conventional" data word. The 37-bit configuration is interpreted as a 36-bit signed number. The *magnitude* bits are numbered from 1 through 36, counting to the right. The magnitude bits of the word are thought of as a fraction, with the binary point appearing at the left-hand end of the word, between bit 1 and the sign. The sign bit (S) determines the polarity of the magnitude portion of the word. The sign bit of ONE indicates a negative word. A sign bit of ZERO indicates a positive word. Thus, a cleared (i.e., containing all binary ZEROes) computer word is interpreted, during a data retrieval, as containing positive ZERO.

Part B of Figure III-5 shows an *octal* data word. The octal notation system is a shorthand means of writing binary numbers. See Appendix A, BINARY AND OCTAL NOTATION.

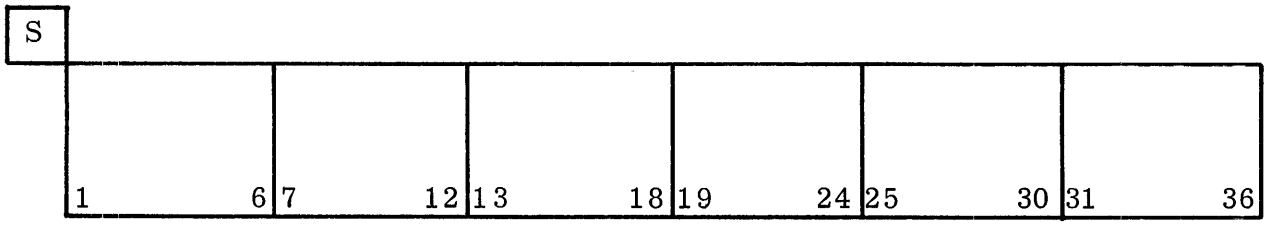
In the octal mode of operation, the computer interprets data words as consisting of twelve groups of three bits each, plus a sign. As in the conventional



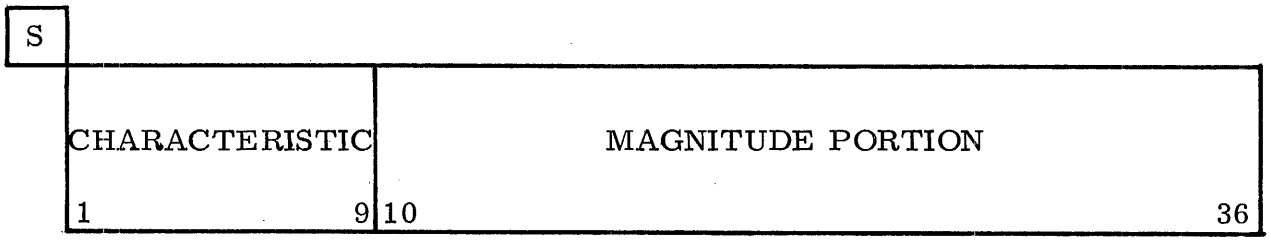
(A) "CONVENTIONAL" DATA WORD



(B) OCTAL DATA WORD



(C) ALPHANUMERIC WORD



(D) FLOATING-POINT DATA WORD

Figure III-5. Data Word Formats

word above, a ZERO sign bit indicates a positive number and a ONE sign bit a negative number. In the octal mode of operation, however, the data word is thought of as consisting of twelve octal characters with sign.

Part C of Figure III-5 shows an *alphanumeric* data word. The 9400 System is capable of processing letters and special characters as well as numbers. Representation of letters and special characters in a binary computer, however, requires special consideration, since the binary system is a method of numeric notation rather than alphanumeric. Therefore, an *alphanumeric binary code* is required. See Appendix G, 9400 ALPHANUMERIC CHARACTER CODES.

Six binary bits are used to represent a single alphanumeric character. For example, the letter A is represented by the binary configuration 000110 and a dollars sign ($) by 100111. Note also that numbers as well as letters require six bits in the alphanumeric mode of operation: A seven is given as 110111 and a nine as 111001.

An alphanumeric data word consists of six groups of six bits each, as shown. The computer interprets each six-bit group as a single alphanumeric character. In the alphanumeric data word, the sign may be interpreted or not, at the programmer's option.

Part D of Figure III-5 illustrates a *floating-point* data word. Floating-point notation greatly extends the range of numbers which can be represented by the 9400 word.

In a floating-point word, the magnitude portion is contained in bits 10 through 36. The sign of the magnitude portion is represented, as in a conventional data word, by bit S. Bits 1 through 9 of the floating-point word form the *characteristic* of the word.

The magnitude portion of a floating-point data word is considered to be a fraction with the binary point appearing between bits 9 and 10. The characteristic portion of the word is a *scaled exponent* for the magnitude portion.

A detailed description of floating-point operation and techniques is provided under FLOATING-POINT ARITHMETIC INSTRUCTIONS.

ARITHMETIC SECTION

Arithmetic, logic, and other operations carried out by the Central Processing Unit occur within the Arithmetic Section. The Arithmetic Section, as shown in Figure III-6, consists essentially of three registers. Its operation is governed by the Control Section of the Central Processing Unit. The Arithmetic Section communicates with the rest of the 9400 System through the Main Transfer Bus.

All three Arithmetic Section registers are addressable. Each has a capacity of 37 bits. For arithmetic and arithmetic-related operations, the high-order bit in each register is treated as a sign. For most other operations, the high-order bit serves simply as an extension of the register. In some cases, it is ignored.

Most arithmetic and logical operations are performed in the Accumulator (ACC). Certain Central Processor operations, such as Divide, Multiply, Shift Long, Cycle Long, and the like, combine the Accumulator and the Q-Register (QRG) into one double-length register. The Q-Register may be used independently for temporary storage. It is also involved in several Central Processor operations.

The B-Register (BRG) is essentially a temporary storage register which is used (automatically) by the computer in the execution of many of the Central Processor Instructions. Like the Q-Register, the B-Register may — with certain limitations — be employed by the programmer for temporary storage of data.

The Arithmetic Section is provided with an Overflow Alarm (OA) and an Underflow Alarm (UA).

Any computer operation which results in a number larger than the capacity of the Accumulator may, at the discretion of the programmer, set the Overflow Alarm. Left-shifting operations may also set OA if a ONE bit is shifted out the left-hand end of the Accumulator.

The Underflow Alarm operates in conjunction with *floating-point* operations and is treated separately under FLOATING-POINT ARITHMETIC INSTRUCTIONS.

SWITCHES AND INDICATORS

The 9400 System is provided with a Control Console containing various switches and indicators. By means of the Console, the computer operator may observe the status of the System at any time. He may examine the contents of any core storage cell or any addressable register. In addition, the operator may enter instruction words or data words directly from the Console to the System. Further, he may execute programs manually, as well as start or stop the computer.

Specific functions of the various switches and indicators are discussed in detail in the following paragraphs and in Section VI, CONTROL CONSOLE.

INFORMATION FLOW IN THE CENTRAL PROCESSOR

Figure III-7 is a block diagram of the Sylvania 9400 Central Processing Unit. It shows all the major pathways over which information is transmitted.

MAIN TRANSFER BUS

The "main artery" of the Central Processing Unit is the Main Transfer Bus. The transfer of all

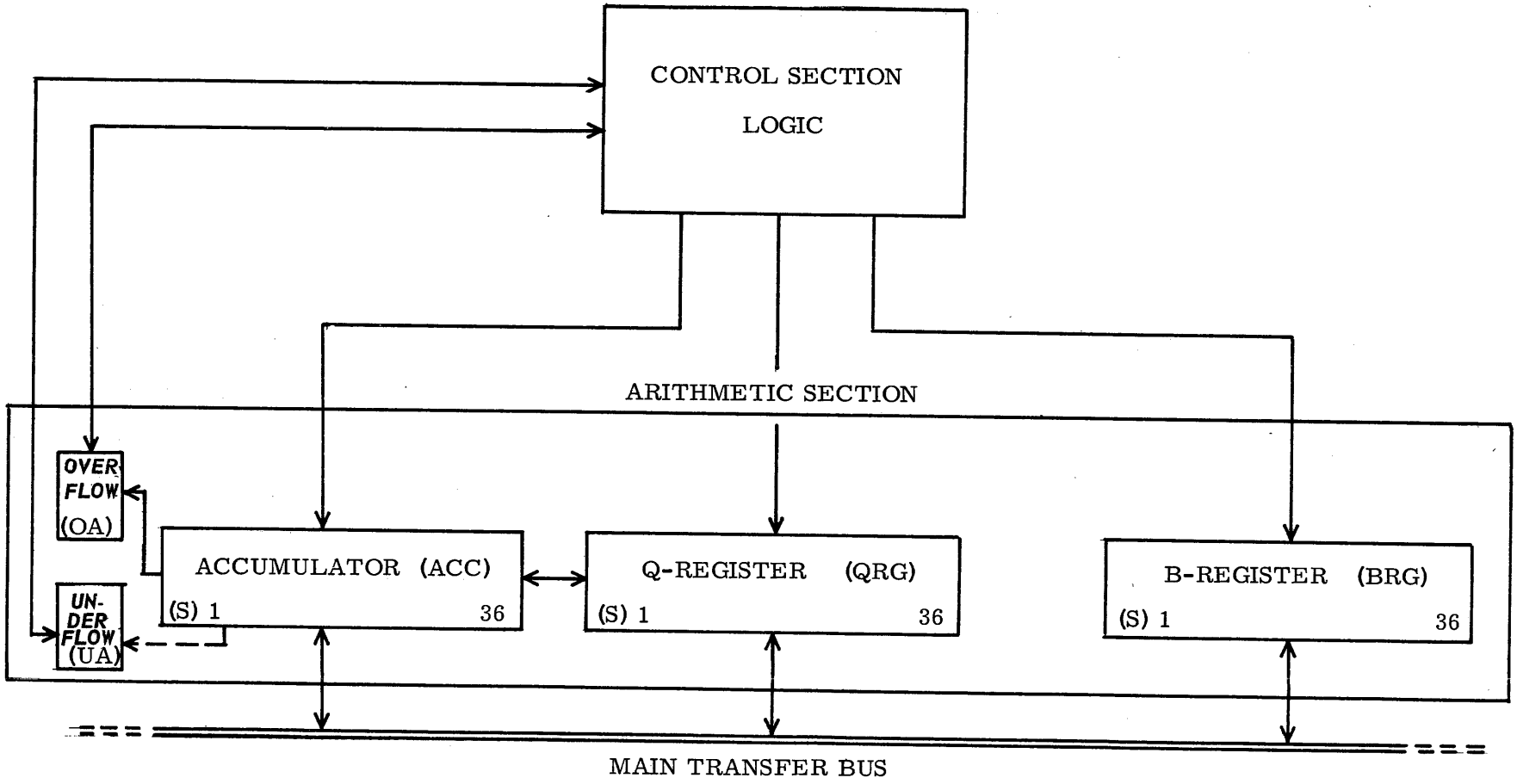


Figure III-6. Central Processing Unit, Arithmetic Section, Block Diagram

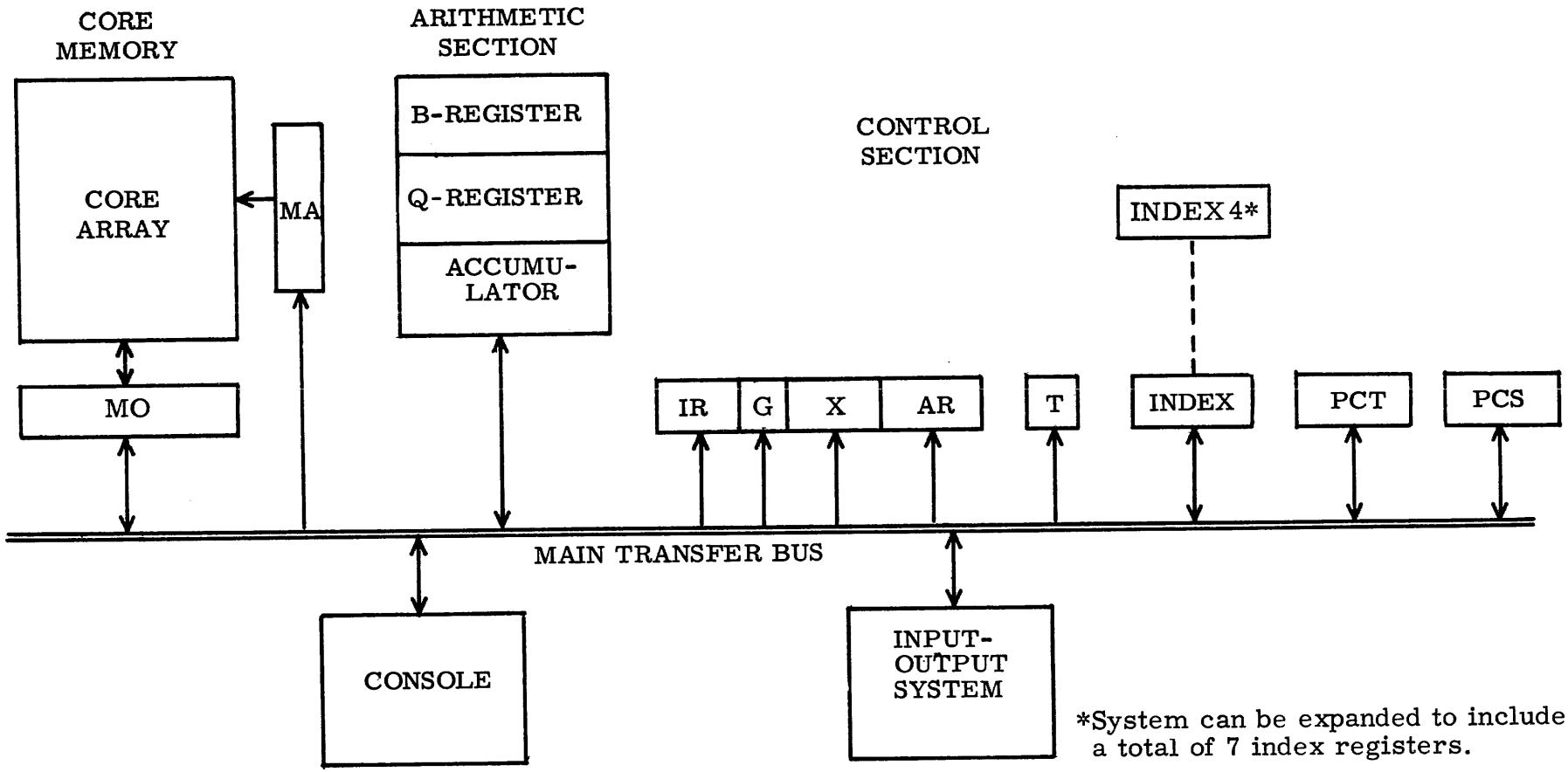


Figure III-7. Central Processing Unit and Related Equipment, Block Diagram

data words and instruction words within the Central Processor takes place along it. The Main Transfer Bus handles information in parallel, normally effecting simultaneous transfer of complete 9400 words.

RETRIEVAL AND INTERPRETATION OF AN INSTRUCTION WORD

Instruction words, like data words, are stored in the Magnetic Core Storage Unit. In order for the computer to process an instruction word, the word must first be retrieved from memory and interrogated. Instruction words are retrieved during the "instruction" portion of the basic machine cycle. The computer must be provided with the address of the instruction to be processed, in order to locate it in Memory. It is the Program Counter which holds the address of the instruction.

The contents of the Program Counter are placed on the Main Transfer Bus. The Memory Address Register (MA) receives a copy of the contents of PCT. A memory read cycle, directed by the address contained in MA, causes the word specified to be sent to the Memory In-Out Register (MO).

Once the word has appeared in MO, it is placed on the Main Transfer Bus (which has been cleared since the previous transfer along it). The word, which is to be interpreted as an instruction, is then broken down into several pieces and distributed, as follows:

1. The operation code (OP) portion goes to the Instruction Register (IR). There, it is decoded in the Decoder Register (D) and turned into the control signals which activate the Central Processing Unit.

2. The first operand address (*a*) portion normally goes to the Address Register (AR), where it remains while the instruction is executed. In the case of shifting and cycling operations, however, where the *a* portion has a *count* rather than *address* significance, it is transferred into the T-Counter, which in turn controls the counting operation.

3. The modifier (*m*) portion is transferred into the X-Register, where it is used during the execution of the instruction. Its significance varies with the instruction of which it is a part.

4. The index (*i*) portion, or *tag,* is transferred into the G-Register. It is used either for indexing operations, or, in the instructions involving a *second operand address,* as part of the second address.

EXECUTION OF AN INSTRUCTION

Once an instruction has been broken down and decoded, the computer enters the execution stage of its instruction cycle. If the operation involves a memory look-up, the contents of the Address Register (AR) are duplicated in the Memory Address Register (MA), causing the required word (operand) to appear in the Memory Output Register (MO). Then, according to the type of instruction being executed, the operand is transferred to the appropriate location, either to another memory cell or to an addressable register. In the case of the ADD instruction, for example, the contents of MO are added to the already existing contents of the Accumulator (acc), leaving the sum in the Accumulator.

If the operation is one of shifting or cycling, the contents of the Arithmetic Unit registers are shifted appropriately by the number of places contained in the T-Counter.

When address modification through indexing is indicated, the contents of the index register specified by G are added to the contents of AR to form the *effective* address. It is the effective address which then appears in MA during retrieval of the data word.

CONSOLE

The Console, like the Memory and addressable registers, communicates with the Central Processing Unit through the Main Transfer Bus. Data or instructions inserted manually through the Console are transferred along the Main Transfer Bus in the same manner as data or instructions transferred from one internal unit to another.

INPUT-OUTPUT

Information received from or transferred to the various 9400 input-output devices is also transmitted by the Main Transfer Bus. The Input-Output System is discussed separately in Section IV.

INSTRUCTION TIMING AND THE BASIC MACHINE CYCLE

The Sylvania 9400 Central Processing Unit operates in a *synchronous* fashion. The Arithmetic Section and Control Section execute Central Processor Instructions at an established rate of speed, independent for the most part of the magnitude or type of data word being processed. All Central Processor instructions, however, do not require the same amount of time for completion. A relatively simple operation like add (ADD), for example, is executed in 8 microseconds, whereas a complex operation like divide (DVD) requires 44 microseconds.

The Central Processing Unit operates under the control of a repetitive Basic Machine Cycle. The Basic Cycle enables the Central Processor to carry out fundamental processes which are common to all or several Central Processor instructions. During the running of a program, the Basic Cycle is supplemented and guided logically by the Central Processor instruction currently being executed.

Table III-1 is a verbal description of the Basic Cycle. A Basic Cycle consists of eight Timing Func-

Timing Function	*Basic Cycle Operations*	*Operations Caused by Previous Interpretation of "ADD" Instruction*
TF-1	Operand arrives at Memory Output Register	
TF-2	Operand is rewritten into Memory	Transfer operand from Memory Output Register to B-register
TF-3	Used for In-Out Instructions only	If signs of A & B are alike, Add B to A; If signs are unlike, form ones complement of B, and add B plus 1×2^{36} to A
TF-4	1. Transfer contents of Program Counter to Memory Address Register 2. Send a read pulse to memory	If B was complemented and there was no overflow, form the ones complement of A and reverse its sign
TF-5	Next instruction arrives at Memory Output Register	1. If B was not complemented and there was an overflow, set OA 2. If B was complemented and there was no overflow, add 1×2^{36} to A
TF-6	1. Transfer appropriate bits of Memory Output Register to AR, X, G, and IR 2. Rewrite Instruction Word into Memory	
TF-7	1. Add ONE to Program Counter 2. Add contents of specified Index Register to AR	
TF-8	1. Transfer contents of AR to Memory Address Register 2. Send a read pulse to Memory 3. Transfer contents of IR to Decoder. Energize decoder lines	

Table III-1. Basic Cycle/Add — Description

tions (TF), each of which is normally one microsecond in length. During each Timing Function, specific operations, related both to the Basic Cycle and to the instruction being executed, may occur.

The left-hand column of Table III-1 describes the Basic Cycle operations occurring during each Timing Function. The right-hand column describes the instruction operations occurring during the corresponding Timing Functions. The instruction illustrated is ADD.

As described under INFORMATION FLOW IN THE CENTRAL PROCESSOR, before an instruction can be executed, it must be retrieved from Core Memory. Instruction retrieval, or look-up, occurs during Timing Functions 4 through 8, as follows:

1. TF-4 time is spent in initiating read-out of the instruction word from Memory. The address of the instruction word is determined by the contents of the Program Counter (PCT).

2. At TF-5 time, the instruction word is in the Memory Output Register.

3. During TF-6 time, the instruction is broken down into its component parts. Because of the destructive nature of the read-out process, the instruction word is rewritten into memory at this time.

4. At TF-7 time, the Program Counter is advanced (for retrieving the *next instruction*), and address modification, if required, is performed.

5. At TF-8 time, the operation code portion of the instruction word is decoded. Preparation is also made to read the operand from memory by sending the contents of the Address Register (AR) to the Memory Address Register (MA).

The five preceeding operations all pertain to the retrieval and interrogation of the instruction which is to be executed, and are all under the control of the Basic Cycle, regardless of what the instruction itself consists of.

Beginning with the following TF-1 time, the computer is under the joint control of the instruction just retrieved and the Basic Cycle, as follows:

1. At TF-1 time, the operand has been read from Memory. Note that as far as the ADD instruction is concerned, nothing happens.

2. At TF-2 time, ADD logic comes into play. The operand is transferred to the B-register. Also at this time, the operand is rewritten into memory.

3. At TF-3 time, the ADD logic begins the addition process by checking signs (a complete description of the mechanization of ADD is given in the section CENTRAL PROCESSOR INSTRUCTIONS under the ADD instruction itself).

4. At TF-4 time, the addition process continues, and automatic look-up — under the control of the Basic Cycle — of the next instruction begins.

5. At the end of TF-5 time, the ADD instruction is complete. The retrieval of the next instruction continues. Thus, execution and look-up are overlapping operations.

Instructions more complicated than ADD extend into TF-6 and TF-7 times as well. Central Processor instructions are always completed before TF-8 time. The major operation during TF-8 is the exchange of information between the Magnetic Core Memory and the various in, out devices, as discussed under INPUT-OUTPUT SYSTEM.

Some Central Processor instructions are sufficiently complex as to require a Basic Cycle of more than 8 microseconds. For such instructions, the Basic Cycle is *extended*. Extension of the Basic Cycle is accomplished by extending, by increments of one microsecond, one of the Timing Functions.

Figure III-8 shows a Basic Cycle in the form of electrical "levels". Shown is an extended Basic Cycle, wherein Timing Function 5 — under control of an instruction — has been increased by two microseconds, lengthening the overall cycle to 11 microseconds.

Extension of the Basic Cycle is caused by the instruction currently being executed. The MLY (multiply) instruction, for example, takes 43 microseconds by extending Timing Function 5 by 35 microseconds.

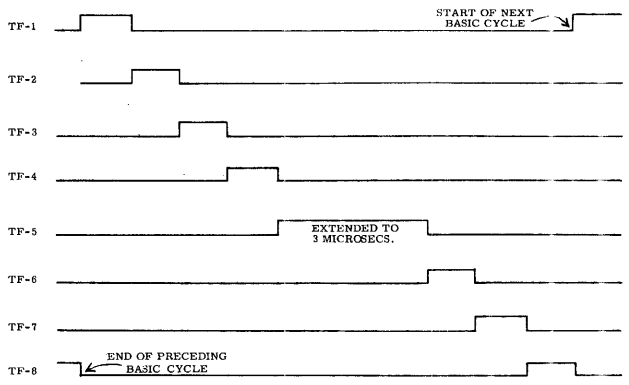


Figure III-8. Basic Cycle (Extended) Timing Chart

CENTRAL PROCESSOR INSTRUCTIONS

The following pages contain a detailed description of all Central Processor Instructions. Each instruction is shown in its symbolic form, together with a pictorial representation of its bit structure. In addition, a verbal description is provided, along with the execution time (in microseconds) of the instruction. Where an instruction is sufficiently complex and for at least one instruction in each category, a detailed explanation, together with an example, is provided.

SYMBOLIC NOTATION

The symbolic representation of each instruction is given below. The 9400 Symbolic Assembly Program (see under SYMBOLIC PROGRAMMING) accepts instructions written in symbolic form. Central Processor Operation codes consist of three-letter mnemonics, such as ADD, SUB, and MLY. The *a, i,* and *m* portions of the instruction word are designated as the *variable field.* In symbolic notation the ADD instruction is written as ADD *a, i, m,* where the three parts of the variable field are separated by commas. When one or more portions of the variable field have no significance for a particular instruction, they are omitted from the symbolic code.

PICTORIAL REPRESENTATION

The bit configuration of each Central Processor instruction word is illustrated pictorially. In addition, the operation code is given in octal as well as in binary. Portions of the instruction word which are not used are shown as shaded areas.

ABBREVIATIONS AND CONVENTIONS IN SYMBOLS

Together with the pictorial representation of each instruction, a verbal description is given, in which the following symbolic conventions are used:

C()

This symbol signifies the contents of that specified in the parentheses. C(acc), for example, indicates the contents of the Accumulator, including sign. C(*a*) specifies the contents of the memory location or addressable register, including sign, indicated by the contents of the *a* portion of the instruction word.

$C()_{m-n}$

When less than a complete word or register is intended, the parentheses are provided with subscripts in the form of bit numbers, for example: $C(acc)_{22-36}$ means the contents of bits 22 through 36 of the Accumulator; $C(acc)_s$ indicates the contents of the Accumulator sign bit alone.

141_8

To distinguish decimal numbers from octal, the subscript *8* is used whenever the quantity involved is written in octal notation. The absence of a subscript, unless otherwise specified, indicates that the number is decimal. Because of the normally obvious characteristics of a binary number, the subscript *2* is not used except for short binary numbers where ambiguity might occur.

ADDRESSABLE REGISTERS

Table III-2 contains a list of all addressable registers and Core Memory locations. For each register, both its octal and symbolic address are given. With certain exceptions, as described in the individual instruction concerned, the addresses given in Table III-2 may be used in the *a, m,* and *im* portions of the Central Processor instruction words.

OVERFLOW CONTROL

The execution of many Central Processor instructions may result in overflow. Overflow occurs when a Central Processor operation produces a number too large for the Accumulator. Most arithmetic operations are capable of causing overflow, and all left-shift operations may cause it. The programmer is provided with a number of *options* which he may use in case of overflow.

Overflow control is implemented through the *modifier* portion (*m*) of the instruction word which may produce the overflow. Table III-3 contains a list of overflow and underflow options. (See FLOATING POINT ARITHMETIC OPERATIONS.) The configuration made up of the three low-order bits (19, 20, 21) of the *m* portion of the word determines the action taken by the Central Processor in case overflow occurs.

If overflow occurs and the 21st bit of *m* is ZERO, the OVERFLOW ALARM (OA) is SET and the Central Processor halts. Thus, any of the following ADD instructions causing overflow would also cause the Central Processor to stop:

ADD *a, i,* 0
ADD *a, i,* 2
ADD *a, i,* 4
ADD *a, i,* 6

The following ADD instructions, if overflow occurs, would SET OA, but the Central Processor would not stop:

ADD *a, i,* 1
ADD *a, i,* 5

If *no action* is desired when overflow occurs, the following modifiers are used:

ADD *a, i,* 3
ADD *a, i,* 7

The programmer also has the option of RESETTING the Overflow Alarm *before* the instruction is executed. Modifiers of 0, 1, 2, or 3 will cause OA to be RESET. Modifiers of 4, 5, 6, or 7 result in no action with regards to OA before execution of the instruction.

By proper choice of a modifier, the programmer may make effective use of an overflow condition to determine the course his program is to follow.

FIXED-POINT ARITHMETIC INSTRUCTIONS

The Sylvania 9400 Central Processing Unit performs fixed-point arithmetic operations on binary

Code	*Address**	*Name*
	00000-77737	Memory Storage Locations
ir1	77741	Index register one
ir2	77742	Index register two
ir3	77743	Index register three
ir4	77744	Index register four
acc	77750	Accumulator
qrg	77751	Q-register
brg	77752	B-register
pct	77753	Program Counter
pcs	77754	Program Counter Store
pio	77755**	Instruction register of In-Out Processor receiving the input order
asr	77756	Alarm Switch Register (see under *Program Control Instructions*)
wsr	77760	Word Switch Register
rar	77761	Real Time Address Register
ror	77762	Real Time Output Register
pir1	77770	Instruction register of first In-Out Processor
pir2	77771	Instruction register of second In-Out Processor
pir3	77772	Instruction register of third In-Out Processor
pir4	77773	Instruction register of fourth In-Out Processor
—:	——***	Non-existent register

*In octal.

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

***If a non-existent register is addressed it will be interpreted as a location which contains all "0's".

Table III-2. Addressable Registers

m Octal	*Before Execution of Instruction*	*If Instruction Causes Overflow or Underflow*
0	Clear OA and UA	Set OA or UA and HALT
1	Clear OA and UA	Set OA or UA
2	Clear OA and UA	Set OA or UA and HALT
3	Clear OA and UA	No Action
4	No Action	Set OA or UA and HALT
5	No Action	Set OA or UA
6	No Action	Set OA or UA and HALT
7	No Action	No Action

The instructions that can cause an OA are:

ADD	DVL	FLS
ADM	SHL	FAM
SUB	SLL	FLM
SBM	FLA	FLD
DVD		

UA can occur with:

FAM	FSM	FLD
FLS	FLM	

Table III-3. Overflow and Underflow Control

fractions. Calculations are carried out on fixed-point data words as if the binary point were placed at the left-hand end of the magnitude portion of the word, between the sign bit (S) and bit 1.

In his calculations, the programmer may employ not only fractions, but both mixed numbers and integers as well, so long as he understands the internal method of computation employed by the Central Processing Unit. For details, refer to Appendix A, NUMBERING SYSTEMS and to the individual instruction descriptions below.

Clear and Load Accumulator Instructions

The following instructions cause data from either the Core Memory or from an addressable register to be duplicated in the Accumulator. Although they perform no arithmetic function in themselves, they often immediately precede arithmetic operations.

CLA a,i CLEAR AND ADD

8 μ sec

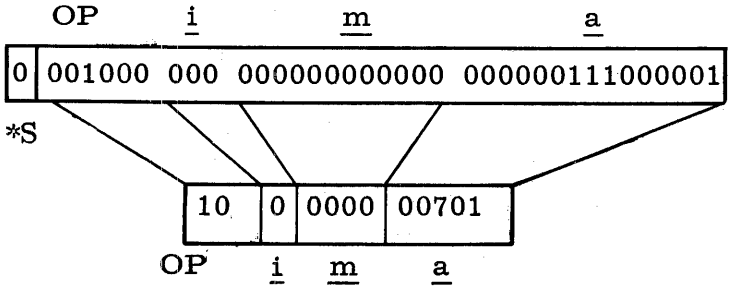
CLA replaces the contents of the Accumulator, C(acc), with the contents of a location specified by *a*.

The *a* is indexable; C(*a*) remains unchanged.

Example: In order to place the contents of memory location 449 (decimal) into the Accumulator the programmer may simply write CLA 449.

The assembly program translates CLA 449 into computer language. A binary image and an octal interpretation is shown below:

Binary Image



Octal Interpretation

*(The sign is not interpreted for instruction words)

Reconciliation:

$10_8 = OP$	This is the octal code for CLA.
$0_8 = i$	No index register is specified, therefore this example is not indexed.
$0000_8 = m$	This portion of the instruction word is never used with CLA.
$00701_8 = a$	This is the octal equivalent to the 449 decimal location of the word to be placed in the Accumulator.

CAM a,i CLEAR AND ADD MAGNITUDE

8 μ sec

CAM replaces C(acc) with the absolute value of the contents of *a*, |C(*a*)|. The Accumulator is made positive.

The *a* is indexable; C(*a*) remains unchanged.

CLS a,i CLEAR AND SUBTRACT

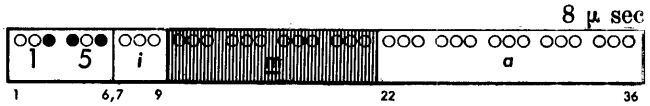
8 μ sec

CLS replaces C(acc) with the negative of C(*a*).

The *a* is indexable; C(*a*) remains unchanged.

CSM a,i CLEAR AND SUBTRACT MAGNITUDE

8 μ sec



CSM replaces C(acc) with |C(*a*)|. The Accumulator is made negative.

The *a* is indexable; C(*a*) remains unchanged.

Add and Subtract Instructions

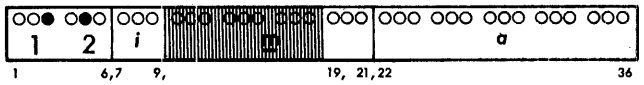
These instructions add to or subtract from the contents of the Accumulator. The contents of *a* is not changed by these instructions. While primarily intended for arithmetic operations with data, they also can be used for instruction address modification or even for modification of the instruction itself.

The C(brg) may be changed depending upon the relative signs and magnitudes of both C(*a*) and C(acc). The SUMMARY OF OPERATION CODES (Appendix I) includes a description of the variations of C(brg).

If overflow occurs during addition or subtraction, the most significant bit is lost. See OVERFLOW CONTROL.

ADD a,i,m ADD

8 μ sec

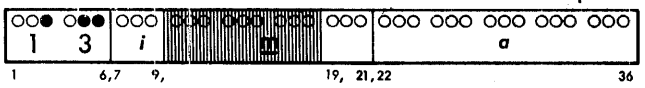


ADD algebraically adds C(a) to C(acc). The sum is developed in the Accumulator.

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

ADM a,i,m ADD MAGNITUDE

8 μ sec

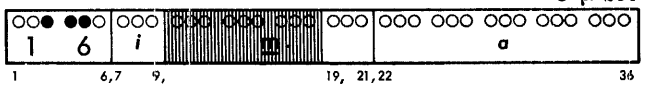


ADM is identical to ADD except that ADM adds the magnitude of C(a). In ADM, C(a) is always interpreted as a positive number.

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

SUB a,i,m SUBTRACT

8 μ sec

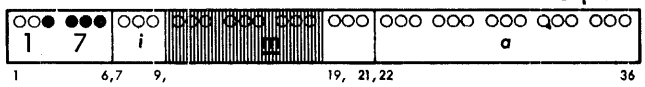


SUB algebraically subtracts C(a) from C(acc). The execution of SUB is functionally identical to ADD except for sign interpretation.

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

SBM a,i,m SUBTRACT MAGNITUDE

8 μ sec



SBM algebraically subtracts |C(a)| from C(acc). SBM is identical to SUB except that in SBM, C(a) is always interpreted as a positive number.

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

Add and Subtract Instructions Summary

The arithmetic rules for sign intepretation in the add and subtract orders are illustrated in the following table — assume a 1 decimal digit Accumulator.

	C(acc) =	+.7	+.7	−.7	+.7
	C(a) =	+.2	−.2	+.7	−.7
ADD a:	C(a) + C(acc) =	+.9	+.5	−.0	+.0
ADM a:	C(a) + C(acc) =	+.9	+.9	−.0	+1.4
SUB a:	C(acc) − C(a) =	+.5	+.9	−1.4	+1.4
SBM a:	C(acc) − C(a) =	+.5	+.5	−1.4	+.0

The "1's" to the left of the decimal point indicate overflow. Note that when the result is equal to zero the sign of the Accumulator is unchanged.

Multiply and Divide Instructions

These instructions each have two modes of operation. The Multiply and Round, MLR, and Divide, DVD, are the regular mode. Multiply, MLY, and Divide Long, DVL, enables the use of double-length products and dividends.

Control options with overflow arising from the divide operation is the same as for the add and subtract instructions.

MLY a,i MULTIPLY

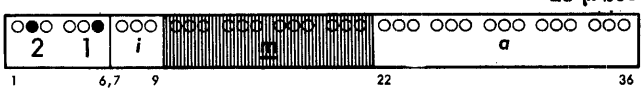
43 μ sec

MLY algebraically multiples C(a) by C(acc) and forms a 72-bit product. After multiplication the Accumulator contains the 36 high-order bits and the Q-register contains the 36 low-order bits of the product. Both acc_s and qrg_s hold the correct sign of the product. C(brg) is replaced by C(a).

The a is indexable; C(a) remains unchanged.

MLR a,i MULTIPLY AND ROUND

43 μ sec

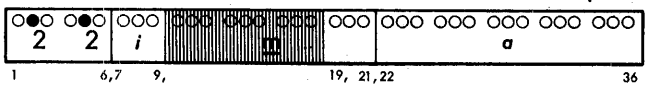


MLR performs a MLY instruction and rounds the product to 36 binary bits in the Accumulator. $C(qrg)_1$ is added to $C(acc)_{36}$ in rounding the product. C(brg) is replaced by C(a) if $C(qrg)_1$ is "1" and by zeros if $C(qrg)_1$ is "0".

The a in indexable; C(a) remains unchanged.

DVD a,i,m DIVIDE

44 μ sec



DVD algebraically divides the 36-bit dividend, C(acc), by the divisor, C(a). The Q-register is used in the DVD to collect the quotient. After the operation, qrg contains the quotient with sign, and acc contains the remainder and the sign of the dividend.

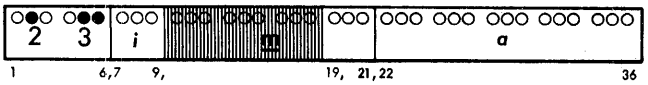
If the sign of C(a) is different from the sign of C(acc), the quotient is negative. C(brg) is replaced by C(a).

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

In DVD, |C(acc)| must be less than |C(a)| or overflow will occur. If |C(acc)| $\geqq$ |C(a)| then the resulting quotient would be a whole number. If this conditions exists, division will not take place. C(acc) and C(qrg) will not change except that C(qrg)$_s$ will be set equal to C(acc)$_s$.

DVL a,i,m DIVIDE LONG

44 μ sec



DVL is identical to DVD except that the dividend may be a 72-bit number. The 36 high-order bits of the dividend are placed in acc and the 36 low-order bits in qrg.

The a is indexable; C(a) remains unchanged. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

The quotient, including sign, is formed in the Q-register. The Accumulator contains the remainder and the sign of the dividend.

For a detailed description of binary multiplication and division, see Appendix A, NUMBERING SYSTEMS.

Shift and Normalize Instructions

The Shift and Normalize instructions complete the Fixed-Point Arithmetic instruction group. They are useful in scaling fixed-point numbers and in editing output data.

It also should be noted that shifting left or right is equivalent to multiplying or dividing by powers of two. Thus, shifting left 5 places is equivalent to multiplying by 32, (2^5). Similarly, shifting to the right 3 places is equivalent to dividing by 8, (2^{-3}).

SHR a,i SHIFT RIGHT

8-19 μ sec*

SHR shifts C(acc) to the right the number of places specified by $a_{30\text{-}36}$. The sign position bit is not shifted. Accumulator bit positions vacated to the left are replaced by "0's", and bits shifted out at the right are discarded.

The a is indexable.

SRL a,i SHIFT RIGHT LONG

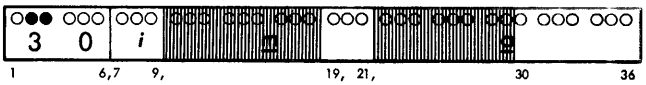
8-37 μ sec*

SRL shifts the 72 bits stored in acc and qrg to the right the number of places specified by $a_{30\text{-}36}$. The sign position bits are not shifted. Acc_{36} is shifted into qrg_1. Accumulator bit positions vacated at the left are replaced with 0's and Q-register bit positions shifted out at the right are discarded.

The a is indexable.

SHL a,i,m SHIFT LEFT

8-22 μ sec*



SHL shifts C(acc) to the left the number of places specified by only bits 30-36 of a. The sign position bit is not shifted. If a significant bit is shifted left out of acc_1, the bit is discarded and overflow occurs. Accumulator bit positions vacated at the right are replaced by "0's".

The a is indexable. Overflow is possible; optional control is specified by $m_{19\text{-}21}$.

---

*TIMING FOR SHIFT OPERATIONS

Shift left: 8 μsec if $a \leqslant 9$; if $a > 9$, $4+\frac{a}{2}$ μsec, where a, if odd, is made *even* downward to next *lowest* value.

Shift right: 8 μsec if $a \leqslant 14$; if $a > 14$, $1+\frac{a}{2}$ μsec, where a, if odd, is made *even* upward to the next *highest* value.

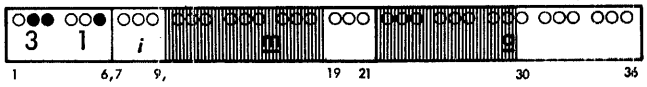
Normalize: $9 + \frac{n}{2}$ μsec, where n is the number of shifts required to normalize the word in the acc; if odd, n is made *even* downward to next *lowest* value.

Shift left long: same as shift left.

Shift right long: same as shift right.

SLL a,i,m SHIFT LEFT LONG

8-40 μ sec*



SLL shifts the 72 bits stored in acc and qrg to the left the number of places specified by $a_{30\text{-}36}$. The sign position bits are not shifted. Qrg_1 is shifted into acc_{36}. If a "1" bit is shifted left out of acc_1 the bit is discarded and overflow occurs. Q-register bit positions vacated at the right are replaced by "0's".

The *a* is indexable. Overflow is possible; control options are specified by $m_{19\text{-}21}$.

NRM a,i NORMALIZE

9-27 μ sec*

NRM shifts $C(acc)_{1\text{-}36}$ left until the most significant bit is in acc_1. The sign position bit is not shifted. The number of places shifted is "n". C(*a*), is replaced by $n \times 2^{-36}$, causing the number of shifts performed to be stored in the least significant bit position of C(*a*). If C(acc) is zero, n = 36.

The *a* is indexable.

The Normalize instruction is useful for computing fixed-point scaling factors and editing output data, but does not perform the floating-point normalize operation. Floating-point normalized data must have the characteristic and number assembled in one word and also have a scaling constant added to the characteristic.

FLOATING POINT ARITHMETIC OPERATIONS

The 9400 Computer performs floating-point arithmetic operations on either positive or negative whole numbers or fractions.

The floating-point technique extends the effective single-word computational range of arithmetic operations. The fixed-point range of approximately -10^{11} to $+10^{11}$ is extended to the floating-point range of approximately -10^{77} to $+10^{77}$. A number precision of better than 1 in 10^8 is still retained; certain operations give a number precision of better than 1 in 10^{16}.

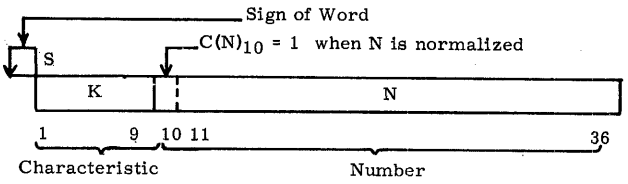
All floating-point computations are performed on floating-point-normalized numbers. The floating-point-normalize technique is essentially that of representing data in a scaled form, with a scaling factor and fraction, and assembling it in the 9400 floating-point format.

*See TIMING FOR SHIFT OPERATIONS, Page 3-17.

WORD FORMAT

A floating-point data word consists of the sign of the number, an exponent as part of the characteristic, and a fractional representation of the number itself. Floating-point operations start and end with the data in this form. Data is usually converted to and from this format by means of subroutines during input and output operations. Floating-point operations performed upon data in other formats will usually give incorrect answers.

FLOATING POINT DATA WORD



NUMBER

The number N, a numerical fraction, is 27 bits long and is located in bit positions 10 through 36 of the data word. The binary point of the number is located to the left of bit position 10. Thus, the number is a fraction. An automatic floating-point normalize operation incorporated in each floating-point instruction shifts the first significant bit into bit position 10 immediately to the right of the binary point. Therefore, as a normalized fraction, the number must be at least ½ and less than 1. The sign of the number is represented by the sign bit of the data word.

Data is represented by the expression $N \cdot 2^b$ where N is the numerical fraction, 2 is the base, and b is the exponent.

CHARACTERISTIC

The nine bits in positions 1 through 9 of the data word are used for the characteristic of the number. In order to represent both whole and fractional numbers simply, an artificial scaling constant is incorporated in the characteristic K. The binary number 2^8 (which is 100 000 000_2 or 400_8 or 256_{10}) is added to the binary exponent b so that the characteristic $K = b + 2^8$. Therefore, the actual range of the binary exponent b is from -2^8 to $+2^8$ even though the characteristic K is interpreted as a positive number ranging from 000 000 000 to 111 111 111.

For example, the exponents +5 and −7 are converted to floating-point characteristics in the following way:

	Decimal	*Binary*	*Octal*
Exponent b	+005	+000 000 101	+005
Scaling constant		+100 000 000	+400
Characteristic		+100 000 101	+405
Exponent b	−007	−000 000 111	−007
Scaling constant		+100 000 000	+400
Characteristic		+011 111 001	+371

The range of K, in octal, is from 000_8 to 777_8 (with a range of b from -400_8 to $+377_8$). K less than 000 causes underflow; K greater than 777_8 causes overflow.

The artificial scaling constant 400_8 is always present in floating-point numbers but is not manipulated during arithmetic computations. For example, when multiplying, the numerical fractions are multiplied and their exponents are added, but the artificial base remains the same. If the multiplier and multiplicand characteristics $K_1 (422_8)$ and $K_2 (403_8)$ are added, the final characteristic of the product is 425_8, not 1025_8.

During floating-point addition, the characteristics of augend and addend are made equal before the numerical fractions are added.

A special case is when N = 0. By definition K = b + 400_8. When N = 0, b = 0 and K would normally become 400_8, the number of the artificial scaling constant. However, in this special case, K is made equal to "0". This arrangement makes "0" the smallest floating-point number in the machine and allows the Compare and Transfer On Zero instructions to be used with floating-point numbers.

DECIMAL TO FLOATING-POINT CONVERSION

Floating-point instructions operate on numbers in the floating-point normalized format only. Conversion between decimal and floating-point normalized format is usually performed by means of subroutines during input or output operations.

The conversion from decimal to floating-point normalized can be described as occurring in four steps:

Convert the decimal data to binary.

Determine the exponent.

Add the scaling constant to the exponent.

Assemble in the floating-point normalized format.

The following example illustrates the simplicity of the conversion operation.

Given the decimal number 497.6875_{10} *convert* the number to binary (also written as octal for simplicity).

$$497.0000_{10} = 761.00_8 = 111\ 110\ 001.000\ 000_2$$
$$.6875_{10} = .54_8 = .101\ 100_2$$
$$497.6875_{10} = 761.54_8 = 111\ 110\ 001.101\ 100_2$$

Determined the exponent by shifting the converted number so that it is a fraction and count the number of places shifted — 3 for the octal number or 9 for the binary number. Remember that $8^3 = 2^9$. The 9 place shift results in an exponent of 9_{10} or 11_8.

Add the scaling factor, 400_8, to the exponent.

Exponent	11
Scaling factor	400
Characteristic	411_8

Assemble in floating-point normalized format.

Characteristic	411.00000_8
Converted number	$.76154_8$
Normalized number	411.76154_8

OVERFLOW AND UNDERFLOW

An overflow alarm, OA, is set whenever the characteristic of the resulting operand is more than 777_8. That is, the OA is set if a characteristic tries to go more positive than $111\ 111\ 111_2$ after compensation for the artificial base in both characteristics.

An underflow alarm, UA, which is similar to the overflow alarm, OA, is set whenever the characteristic of the resulting operand is less than 000_8. For example, in division, the characteristic of the divisor is subtracted from the characteristic of the dividend and the UA is set if the difference is smaller than $000\ 000\ 000_2$ after compensation for the artificial base in both characteristics.

The explanation of optional program control after overflow or underflow, and a table of control numbers and functions, is given under OVERFLOW CONTROL.

FLOATING-POINT CONVENTIONS AND ABBREVIATIONS

The conventions and abbreviations explained earlier for fixed-point operations are expanded here to encompass floating-point operations. The abbreviations, N, for number, and K, for characteristic, were described above.

An additional abbreviation "F" preceding an expression is used to designate floating-point normalized data. When used to convey the meaning, "the floating-point-normalized content of", it replaces the letter "C" which simply designates "the contents of" without qualification and is ordinarily applied to non-normalized numbers. Therefore, F(a) is identical to C(a) only if C(a) is a floating-point normalized number. Similarly, FK and FN refer to a characteristic and number of normalized floating-point data.

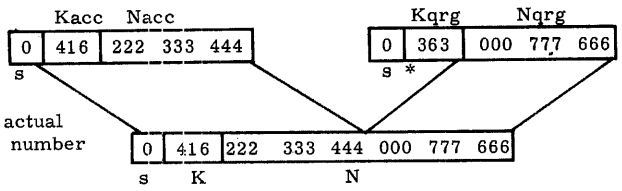
The letters K and N may be prefixed to a location designation to indicate reference to only the characteristic or number portion of the location. The bit position suffixes refer to the entire location.

For example, F(Nacc)$_{18}$ refers to the floating-point contents of the number portion of the Accumulator, specifically, bit position 18 of the Accumulator.

All floating-point instructions except the divide form 72-bit normalized floating-point results using both the Accumulator and the Q-register. Each register contains a sign, a number and the associated characteristic. The Accumulator contains the normalized 27 high-order bits of the number with appropriate characteristic, previously defined as the exponent of the normalized number plus 400_8.

The low-order word or portion of a 72-bit normalized floating-point result in the Q-register is normalized with respect to the high-order word, but not necessarily as an entity. That is C(Nqrg)$_{10}$ and following bits may be 0. The low-order characteristic is 33_8 (or 27_{10}) less than the high-order characteristic, C(Kqrg) = C(Kacc) -33_8.

Example: Assume an answer in acc and qrg:



*Same as for acc.

Note that Kqrg = Kacc -33_8, 416_8 -33_8 $=363_8$

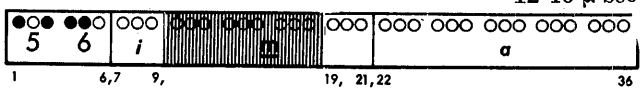
Thus, in the example, the number Nqrg is not normalized as an entity, but is correctly normalized with respect to the Accumulator; Nqrg being 27 places (33_8) to the right of Nacc.

FLOATING-POINT INSTRUCTIONS

There are six floating-point arithmetic instructions, each including an automatic normalize operation after the arithmetic operation. The following descriptions include notes on results of operations, especially after alarms. These notes form the basis of error detection and correction.

FLA a,i,m FLOATING POINT ADD

12-46 μ sec



FLA algebraically floating-adds F(a) to F(acc) and forms the 72-bit floating-point sum.

After the operation acc contains the normalized high-order bits of the sum; qrg contains the low-order bits of the sum. The acc and qrg also contain the respective high-order and low-order characteristics.

The a is indexable; C(a)remains unchanged. Overflow or underflow is possible; optional control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

Some conditions which may result from FLA, FAM, FLS, and FSM to alter the normal contents of acc, qrg, or brg are as follows:

No alarm, but C(Nqrg) = 0
C(qrg) = 0

Overflow, when C(Kacc) $>777_8$
C(Kacc) = 777_8 instead of 1000_8
C(Kqrg) = 745_8, ($1000_8 - 33_8$) which assumes Kacc = 1000_8

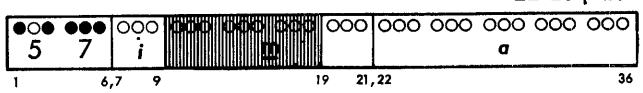
Underflow, when C(Nacc)$_{10}$ = 0
C(Kacc) = C(Kqrg) = 0
C(Nacc), C(Nqrg) = the un-normalized number, consistent with C(Kacc) = 0. This occurs when normalization fails.

Underflow, when C(Nacc)$_{10}$ = 1 and C(Kacc) $<33_8$.
C(Kqrg) = 0, but should be <0.

In all cases C(brg) = 0.

FAM a,i,m FLOATING POINT ADD MAGNITUDE

12-46 μ sec

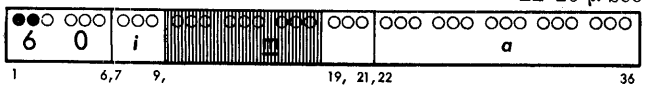


FAM algebraically floating-adds F(a) to F(acc). It is identical to FLA except that FAM adds the magnitude of F(a). In FAM, F(a) is always interpreted as a positive number.

The a is indexable; C(a) remaining unchanged. Overflow is possible; optional control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

FLS a,i,m FLOATING POINT SUBTRACT

12-46 μ sec

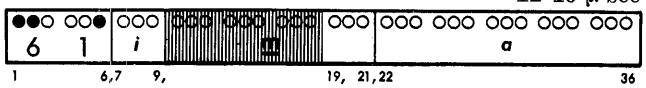


FLS algebraically floating-subtracts F(a) from F(acc). The execution of FLS is functionally identical to FLA except for sign interpretation.

The a is indexable; C(a) remains unchanged. Overflow or underflow is possible; optional control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

FSM a,i,m FLOATING POINT SUBTRACT MAGNITUDE

12-46 μ sec

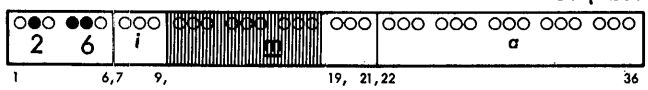


FSM algebraically floating-subtracts F(a) from F(acc). It is identical to FLS except that FSM subtracts the magnitude F(a).

The a is indexable; C(a) remains unchanged. Underflow is possible; option control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

FLM a,i,m FLOATING POINT MULTIPLY

37 μ sec



FLM floating-multiplies F(a) by F(acc).

After the operation, acc contains the normalized high-order bits of the product and qrg contains the low-order bits.

The a is indexable; C(a) remains unchanged. Overflow or underflow is possible; optional control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

Certain conditions resulting from FLM may alter the normal contents of acc and qrg as follows:

No alarm but C(Nqrg) = 0
C(qrg) = 0

No alarm when C(acc) or C(a) = 0
C(acc) = 0; C(qrg) = 0

Overflow when C(Kacc) + C(Ka) $>777_8$
C(acc) = 0; C(qrg) = original C(acc)

Underflow when $C(Nacc)_{10} = 0$
C(Kacc) = 0; C(Nacc) is correct but unnormalized
C(qrg) = 0 or C(Kqrg) = 0 but should be <0, C(Nqrg) is correct but unnormalized

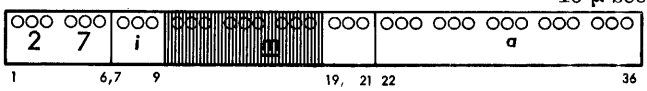
Underflow when $C(Nacc)_{10} = 1$, and C(Kacc) $<33_8$
C(Kqrg) = 0 incorrect because when C(Kacc) $<33_8$, then C(Kacc) $-33_8 < 0$

Underflow when C(Kacc) + C(Ka) $<2^8$
C(acc) = 0, C(qrg) = original C(acc)

In all cases C(brg) = C(a).

FLD a,i,m FLOATING POINT DIVIDE

40 μ sec



FLD floating-divides F(acc) by F(a).

After the operation qrg contains the normalized quotient and characteristic and acc the remainder.

The a is indexable; C(a) remains unchanged. Overflow or underflow is possible; optional control is specified by $m_{19\text{-}21}$. Original C(qrg) and C(brg) are discarded.

The conditions resulting from FLD which alter the normal contents of acc, qrg, and brg are as follows:

Overflow, when $C(a)_{10} = 0$
C(Kacc) = C(Kacc) −C(Ka), C(Nacc) is unchanged
F(qrg) = original F(acc)
$C(brg)_8 = C(a)_8$; $C(brg)_{1\text{-}36}$ equals the complement of $C(a)_{1\text{-}36}$

Overflow when C(Kacc) − C(Ka) $> 777_8$
The same as above except $C(Nbrg)_{10} = 0$, after complementing

Overflow when C(C(Kacc) − C(Ka) $= 777_0$, and C(Nacc) $\geq$ C(Na)
C(acc) = remainder
$C(qrg)_8$ = sign of quotient
C(Kqrg) = 777_8 instead of 1000_8 C(Nqrg) = number of quotient
F(brg) = F(a)

Underflow when C(Kacc) − C(Ka) <0
The same as overflow when $C(a)_{10} = 0$ except $C(Nbrg)_{10} = 0$, after complementing

Underflow when C(Kacc) $\leq 33_8$, or when C(Kacc) = 0, and C(Nacc) = C(Na)
C(Kacc) = 0, it should be < 0
C(Nacc) = Remainder
F(brg) =F(a)

In all cases C(brg) = C(a).

DATA TRANSFER INSTRUCTIONS

The following data transfer instructions transfer single data words between registers or memory locations. They are logically similar to the Clear-Accumulator instructions.

STR a, i STORE

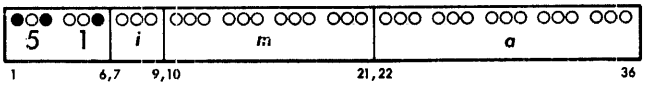
8 μ sec

STR replaces C(a) with C(acc). C(acc) remains unchanged. The contents of the Accumulator are transmitted to an addressable register or to a memory location.

The a is indexable.

LOD a,i,m LOAD

9 μ sec



LOD replaces C(m) with C(a), where m refers exclusively to an addressable register. However, a may refer to an addressable register or a memory location. C(brg) is replaced by C(a).

The a is indexable.

Example:

Replace C(qrg) with C(400).

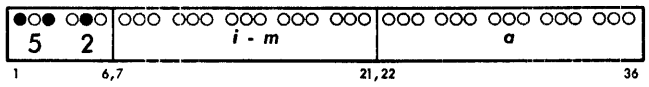
INSTRUCTION	WRITE
LOD *a,i,m*	LOD 400, 0, qrg

The Assembly Program interprets qrg and inserts the numerical address 7751_8.

Actually, the address of qrg is 77751, but the high-order 7 is inserted automatically during execution when an addressable register is specified by m.

MOV a,i,m MOVE

13 μ sec



The MOV instruction moves C(a) to the location specified by im. This is a two-address instruction in which a specifies one 15-bit address, and i and m together specify the second 15-bit address. Either a or i-m may be an addressable register. The MOV instruction cannot be indexed. C(brg) is replaced by C(a).

If a REPEAT instruction precedes the MOVE instruction, a special sequence is followed, as explained under the REPEAT instruction.

PROGRAM CONTROL INSTRUCTIONS

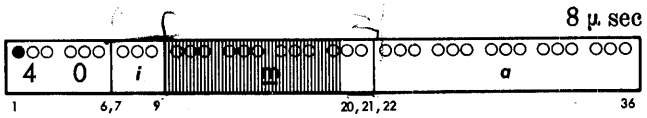
The Program Control instructions enable the programmer to alter the sequence of events within his program. Loops and subroutines, for example, are usually initiated and terminated by Program Control instructions.

The SENSE instructions, which are included in this group, are frequently used in controlling and monitoring simultaneous In-Out Processor and Central Processor operations. Table IV-6 contains a list of *Sensable Controls,* all of which may be "addressed" by the SENSE instructions.

All Program Control instructions except COMPARE are useable in the Trapping Mode (see under TRAPPING MODE).

TRU a,i,m UNCONDITIONAL TRANSFER

8 μ sec

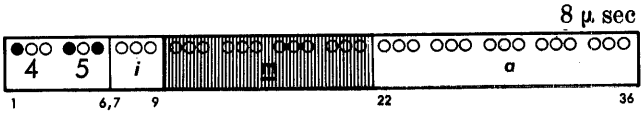


TRU unconditionally transfers the program control to the instruction in memory location a. Bits $m_{20\text{-}21}$ of this instruction are used to control the trapping mode.

The a is indexable.

TRZ a,i TRANSFER ON ZERO ACCUMULATOR

8 μ sec



TRZ transfers the program control to the instruction in memory location a if $C(acc)_{1\text{-}36} = 0$. If $C(acc)_{1\text{-}36} \neq 0$, the program continues in sequence.

The a is indexable.

TRP a,i TRANSFER ON POSITIVE ACCUMULATOR

8 μ sec

TRP transfers the program control to the instruction memory location a if the sign of the Accumulator is positive, $C(acc)_s = 0$. Otherwise, the program continues in sequence. Only the sign bit of the Accumulator is interpreted.

The a is indexable.

TRN a,i TRANSFER ON NEGATIVE

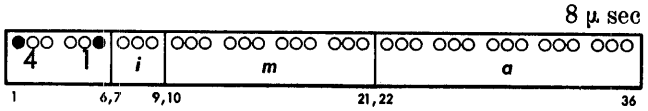
8 μ sec

TRN transfers the program control to the instruction in memory location *a* if the sign of the Accumulator is negative, $C(acc)_s = 1$. Otherwise, the program continues in sequence. Only the sign bit of the Accumulator is interpreted.

It should be noted that TRZ, TRP, and TRN are not mutually exclusive. Combinations of TRZ and TRP or TRZ and TRN may be used to test the sign of zero.

TRL a,i,m LOAD PCS AND TRANSFER

8 μ sec



TRL loads the Program Counter Store (PCS) with the location of this instruction incremented by one. Program control is transferred to the instruction in memory location *a*, not indexable.

The TRL instruction stores the location of the next instruction in sequence, thereby enabling the program to return to the original sequence after completing the loop or subroutine that starts at *a*.

TRL also loads *m* into the Index Register specified by *i*, ir*i*. The *i* and *m* portions are usually used to implement the indexing to be performed by the loop or subroutine.

TRS TRANSFER TO PCS

8 μ sec

TRS transfers the program control to the memory location stored in the Program Counter Store (PCS).

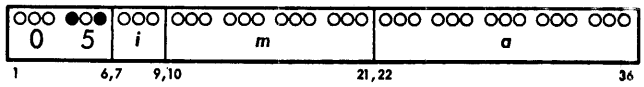
This instruction is usually used in conjunction with TRL, which stores an address in PCS.

SENSE INSTRUCTIONS

Sense Instructions provide a program with the ability to reference the state of various accessible internal switches (Sense Switches) which indicate specific conditions pertinent to the processing function of the program. In addition to the sensing function, Sense Instructions can be used to *set* or *reset* specific Sense Switches. The *m* portion of a Sense Instruction will contain a code which indicates the specific Sense Switch to be sensed (see Table IV-6).

SEN a,i,m SENSE

8 μ sec



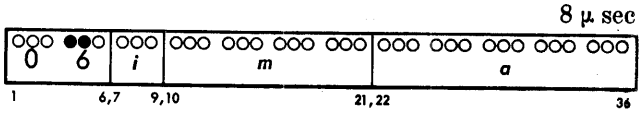
SEN transfers the program control to the instruction in memory location *a* if the switch specified by *m* is SET. Otherwise, the program continues in sequence.

That is, if the switch specified by *m* is SET, transfer to memory location *a*. If the switch is RESET, the program continues in sequence. The Sense Switch remains unchanged.

The *a* is indexable.

SNS a,i,m SENSE AND SET

8 μ sec

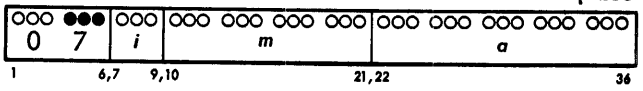


If the switch specified by *m*, is RESET, it is SET by SNS, and the program control is transferred to the instruction in location *a*. If the switch is SET, the program continues in sequence.

The *a* is indexable.

SNR a,i,m SENSE AND RESET

8 μ sec



If the switch specified by *m*, is SET, it is RESET by SNR, and the program control is transferred to memory location *a*. If the switch is originally RESET, the sequence is continued in order.

The *a* is indexable.

TRC a,i COMPARE

11 μ sec

TRC algebraically compares C(acc) with C(a). One of three possible program control functions occur.

If C(acc) < C(a) the program executes the first instruction after TRC.

If C(acc) > C(a) the program skips the first instruction and executes the second instruction after the TRC.

If C(acc) = C(a) the program skips the first two instructions and executes the third instruction after the TRC.

C(qrg) is replaced by C(acc), and C(brg) is replaced by a. C(acc) and C(a) remain unchanged.

In the execution of this instruction, minus zero is considered equal to plus zero, $-0 = +0$.

The a is indexable.

HLT HALT

8 μ sec

HLT suspends the Central Processor program immediately and stops the entire system upon completion of any input-output operations in progress when HLT is decoded. If the operator desires to restart operations with the next instruction in sequence, he need only actuate the START AT PC switch.

If HLT is decoded while an In-Out Processor is in the Order-Sequence mode, the entire order sequence is completed before the computer stops. All input-output operations resulting from a previous single instruction are also continued to termination although the Central Processor halts.

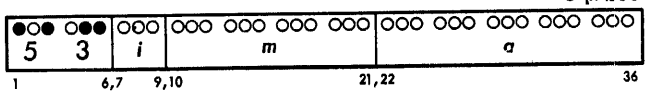
INDEX CONTROL INSTRUCTIONS

The two Index Control instructions, Load Index and Transfer on Index, are multifunction instructions for implemeting indexing techniques. These instructions each deal simultaneously with two Index Registers.

In controlling program loops, one Index Register controls the number of times the program sequences through the loop while the second Index Register increments the addresses of data being processed.

LXS a,i,m LOAD INDICES

8 μ sec



LXS always loads two Index Registers. The contents of the Index Register specified by i, C(ir$i$), are replaced by a. The contents of the next Index Register in sequence, C(ir(i + 1)), are replaced by m. LXS is not indexable.

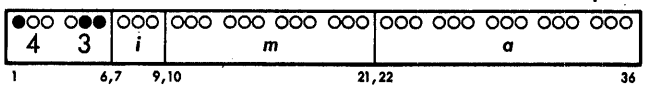
INDEX REGISTER NUMBERING

The LXS instruction is effective for any value of i, so long as $i <$ n, where n is the number of Index Registers in the 9400 System. If i = n, then $i + 1 = 1$, by design. However, if $i = 0$ or i > n, the i will refer to a non-existent Index Register, the instruction will not be executed, and the program will proceed to the next instruction in sequence.

If only one Index Register is to be loaded, the regular LOD instruction should be used.

TRX a,i,m TRANSFER ON INDEX

11 μ sec



TRX uses two Index Registers and transfers program control conditionally. The program control continues in sequence if either of two tests are satisfied.

If C(ir(i + 1)) = 0 continues in sequence

If C(ir(i + 1)) ≠ 0 subtract one, then if

C(ir(i + 1)) = 0 continue in sequence or if

C(ir(i + 1)) ≠ 0 add m to C(ir(i) and the next instruction is taken from a.

These tests are made automatically (both before *and* after subtracting 1 from C(ir(i + 1)) in order to simplify counting and insure that the Index Register will not inadvertently pass zero and fail to terminate the loop after it is executed the proper number of times. A flow diagram of the TRX instruction is shown in Figure III-9.

The TRX instruction is effective for the same values of i specified in the instruction in the same manner as those defined under LXS "Index Register Numbering".

The execution of TRX causes C(brg) to be replaced by a. If a nonexistent index register is addressed, the instruction will not be executed, and the program will proceed to the next instruction in sequence, C(brg), however, is replaced by a.

Generally, one register, ir(i), modifies the addresses of data being processed during each loop of a series of loops. A second register, ir($i + 1$), counts the number of times a program loop is performed and tests for completion of the desired number of loops.

The i bits of the TRX instruction designate the address-modifying register. The second Index Register is, by design, the next higher numbered index, ir($i + 1$). Execution of conditional functions of TRX depend on C(ir($i + 1$)).

At the start of the series of loops, LXS can be used to load the number of loops into ir($i + 1$) and the address modifier for the first loop in ir(i).

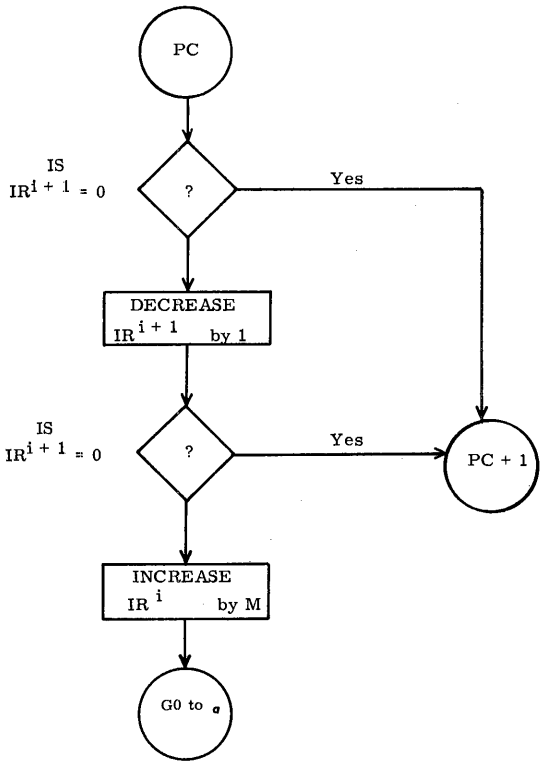


Figure III-9. TRX Instruction, Flow Diagram

ADDRESS MODIFICATION INSTRUCTIONS

The Address Modification instructions provide the programmer with single instructions to modify the address portion only of words. The Accumulator is used to perform the arithmetic operation, and the results may be equally applicable to memory locations, Index Registers, or other addressable registers.

The Add Modifier, ADB, and Subtract Modifier, SBB, instructions automatically modify both the specified address and the specified Index Register.

RPA a,i REPLACE ADDRESS

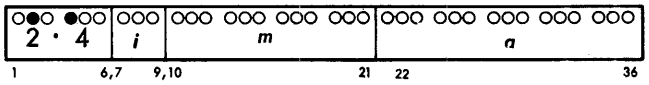
11 μ sec

RPA replaces the address portion of location a with the address portion of the Accumulator: $C(a)_{22\text{-}36}$ is replaced by $C(acc)_{22\text{-}36}$. The remainder of location a, $C(a)_{s1\text{-}22}$ is unchanged. The Accumulator is unchanged, but C(brg) is replaced by C(acc) in the process.

The a is indexable.

ADB a,i,m ADD MODIFIER

13 μ sec



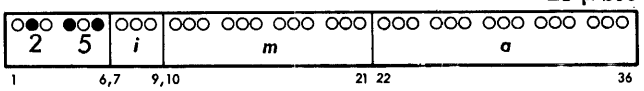
ADB forms in the Accumulator the algebraic sum of C(a) plus m. The sum in acc is then used to replace both the original C(a) and C(ir$i$). The low order bits of C(acc) replace C(ir$i$).

The original C(acc) replaces C(qrg). C(brg) is dependent upon the relative signs and magnitude of C(a) and C(acc) as indicated in Appendix J, SUMMARY OF OPERATIONS.

ADB is not indexable. Overflow is ignored.

SBB a,i,m SUBTRACT MODIFIER

13 μ sec



SBB forms in acc the algebraic difference of C(a) minus m. The difference in acc replaces both the original C(a) and C(ir$i$). The low order bits of C(acc) replace C(ir$i$).

The original C(acc) replaces C(qrg); C(brg) depends on the relative signs and magnitudes of C(a) and C(acc) as indicated in Appendix J, SUMMARY OF OPERATIONS.

SBB is not indexable. Overflow is ignored.

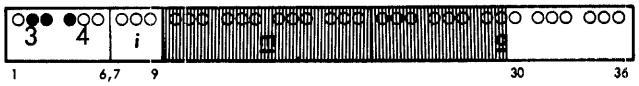
WORD MODIFICATION INSTRUCTIONS

The word modification instructions complete the set of operations by which the programmer may modify word structures. Their main use is in data handling and editing, though they have important specialized applications in many areas.

The cycle operations permit the sequential rearrangement of data without the potential bit loss inherent in the shift operations. The mask and logical operations modify word structures bit-by-bit without the bit-carry of arithmetic operations. In addition, the logical operations provide for Boolean manipulation of data.

CYS a,i CYCLE SHORT

8-19 μ sec*



The Cycle Short instruction forms a closed ring, consisting of the 36 bits of the Accumulator, and cycles the bits left the number of places specified by *a*. Bits are cycled to the left out of acc_1 and enter acc_{36}, so that no information is lost as in an ordinary shift operation. $C(acc)_s$ remains unaltered.

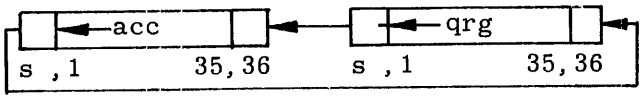
The *a* is indexable.

CYL a,i CYCLE LONG

8-38 μ sec*

The Cycle Long instruction is similar to CYS except that a 74-bit ring is formed by the complete Accumulator and Q-Register. The 36 bits plus sign in each register, are cycled left the number of places specified by *a*.

The *a* is indexable.



As illustrated above, bits pass from:

qrg_1 to qrg_s; qrg_s to acc_{36}; acc_1 to acc_s; acc_s to qrg_{36}.

MSK a,i REPLACE THROUGH MASK

11 μ sec

The MSK operation can be thought of as a selective store operation. It replaces portions of C(*a*) with the corresponding portions of C(acc), as specified by the mask in qrg. The replacement is controlled by the "1's" in qrg.

**Timing for Cycle Operations:* 8 μsec for a $\leqslant 14$. For $a > 14$: $1 + \frac{a}{2}$ μsec, where *a* if odd, is made *even* to next *highest* value.

Only those bits of C(*a*) corresponding to the "1" bits in qrg, the mask, are replaced by C(acc). The C(acc), C(qrg), and the remainder of C(*a*) remain unchanged. C(brg) is replaced by C(acc).

The *a* is indexable.

Example:

Replace *i* and *m*, bits 7-21, of location *a* with bits 7-21 of the Accumulator. The mask is in memory location 50.

$$C(50) = 0\,000\,000\;\;111\cdots111\;\;000\cdots\cdots000$$
$$\qquad\quad s,1 \qquad 6,7 \qquad\quad 21,22 \qquad\quad 36$$

Then

LOD 50, 0, 7751 — This places C(50), into C(qrg). The "1's" in $C(qrg)_{7\text{-}21}$ comprise the effective mask. The corresponding bits of $C(a)_{7\text{-}21}$ are replaced by $C(acc)_{7\text{-}21}$ when the MSK instruction is executed.

MSK *a* — This replaces $C(a)_{7\text{-}21}$ with $C(acc)_{7\text{-}21}$. In the event that C(*a*) contained a MOV instruction, this sequence would modify the second address. MSK could also be used to modify the first address, but the RPA is a more efficient and specialized instruction.

LGA a,i LOGICAL ADD

8 μ sec

This operation forms in acc the logical sum of C(*a*) and C(acc), including the sign. The logical sum is formed on a bit-by-bit basis using corresponding bits of the C(acc) and C(*a*).

This logical sum is called an inclusive "OR" function because the result is a "1" in C(acc) if the corresponding bit positions of either the original C(acc) OR C(*a*) contains a "1".

The *a* is indexable. C(*a*) remains unchanged.

Example:

Form the logical sum of the Accumulator and location *a*.

Sign	*Number*
C(acc) = 0	001 001 100 111 ... 000
C(*a*) = 1	000 101 001 110 ... 000
Final C(acc) = 1	001 101 101 111 ... 000

LGM a,i LOGICAL MULTIPLY

8 μ sec

This operation forms the logical product, in acc, of C(a) and C(acc), including the sign. The logical product is formed on a bit-by-bit basis using corresponding bits of the acc and a.

The logical product is often called an AND function because the result is a "1" if the corresponding bit positions of C(acc) AND C(a) both contain "1's".

The a is indexable; C(a) remains unchanged.

Example:

Form the logical product of the Accumulator and location a.

Sign	*Number*
C(acc) = 0	000 101 001 110 000
C(a) = 1	000 001 100 111 000
Final C(acc) = 0	000 001 000 110 000

LGN a,i LOGICAL NEGATION

8 μ sec

This instruction forms the complement in acc of C(a), including the sign. In binary notation the complement of "1" is a "0" and likewise the complement of a "0" is "1". Therefore, LGN forms the one's complement of C(a).

The a is indexable; C(a) remains unchanged.

Example:

Form the logical negation of C(a), i.e., LGN a.

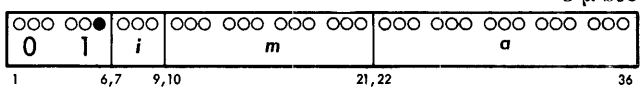
Sign	*Number*
C(a) = 1	001 001 100 111 000
Final C(acc) = 0	110 110 011 000 111

THE REPEAT FUNCTION

The repeat instruction enables the programmer to handle a frequently encountered type of program loop. The loop consists of repetitive application of an instruction to a block of data. The repeat instruction defines the size and location of the block by using two index registers, as a repetition counter and as an effective address increment. It is immediately followed by the affected instruction. Together, the two instructions perform the *repeat function.*

RPT a,i,m REPEAT

8 μ sec



RPT causes the next instruction in sequence to be executed a + C(ir$i$) + 1 times. The effective address of the repeated instruction is modified in the manner described below.

A flow diagram of the RPT instruction is shown in Figure III-10.

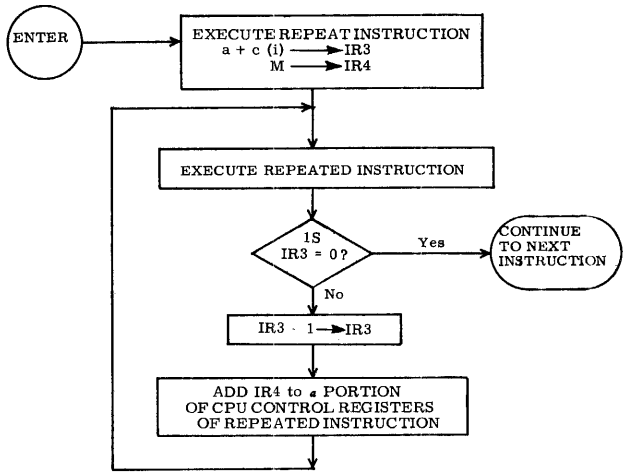


Figure III-10. RPT Instruction, Flow Diagram

RPT places a plus the contents of Index Register i in Index Register number three, ir3; and m in Index Register number four, ir4.

The instruction following the RPT is executed in its normal mode, then C(ir3) is tested for equality to zero.

If C(ir3) ≠ 0, a "1" is subtracted from C(ir3), and C(ir4) is cumulatively added to C(Address Register) to form the new effective address of the instruction being repeated. The loop proceeds with the execution of the instruction upon its new effective address, testing for C(ir3) = 0, and cumulatively indexing until the repeated instruction has been executed a + C (ir$i$) + 1 times.

When C(ir3) = 0, either initially or after execution a + C(ir$i$) + 1 times; the Program Counter is incremented by one, and the program returns to its original sequence.

The effective address of the repeated instruction is a^2 + C(iri^2) + nm^1, when "n" is the number of repetitions. Superscript "1" refers to the RPT instruction, and superscript "2" refers to the repeated instruction. The subscript refers to the index register number.

Example: Add the numbers in memory locations 105 through 114 to the contents of the Accumulator. Assume that the C(ir2) = 2 and will be used to form the effective address of the REPEAT instruction. Assume the C(ir1) = 5 and will be used to form the effective address of the ADD instruction which will be repeated. Prior to the Repeat these index registers must be loaded independently. Before the REPEAT instruction the contents of the various index Registers are:

ir1	ir2	ir3	ir4
5	2	0	0

The REPEAT and ADD instructions are now performed.

RPT a^1, i^1, m^1 ADD a^2, $i_1{}^2$, m^2	RPT 5, 2, 1 ADD 100, 1, 0				
		ir1	ir2	ir3	ir4
ADD a^2 + c($i_1{}^2$)	ADD 105	5	2	7	1
ADD a^2 + c($i_1{}^2$) + c(ir4)	ADD 106	5	2	6	1
ADD a^2 + c($i_1{}^2$) + 2·c(ir4)	ADD 107	5	2	5	1
ADD a^2 + c($i_1{}^2$) + 3·c(ir4)	ADD110	5	2	4	1
ADD a^2 + c($i_1{}^2$) + 4·c(ir4)	ADD 111	5	2	3	1
ADD a^2 + c($i_1{}^2$) + 5·c(ir4)	ADD 112	5	2	2	1
ADD a^2 + c($i_1{}^2$) + 6·c(ir4)	ADD 113	5	2	1	1
ADD a^2 + c($i_1{}^2$) + 7·c(ir4)	ADD 114	5	2	0	1

It is seen from the above that the effective address of the ADD instruction is a^2 + C(iri^2) + 7. Where 7 = "n" is the number of repetitions of the *repeated* instruction. The first ADD is not under REPEAT control. That is why the number of executions is equal to a^1 + c(iri^1) + 1.

Any instruction which can normally be indexed can also be indexed when used in conjunction with RPT. This is due to the fact that when the instruction is read out of memory the address portion is placed in the Address Register and immediately incremented by the contents of the Index Register specified in that instruction.

Since the Address Register is never cleared until the repeat operation loop is completed the normal operation is performed only during the first execution. Upon subsequent executions of the instruction the Address Register is incremented by C(ir4). Therefor, after "n" repetitions of the instruction the Address Register contains the effective address a^2 + (C(iri^2) + nm^1 where m^1 = C(ir4).

In the preceding example both a^2 and i^2 were used together to specify the initial effective address of 105, in other words, ADD was indexed.

Assume 14 was previously loaded with ir1, then make a^2 = 87 and i^2 = 1. The effective address of any indexable instruction is a + C(ir$i$); in this example, a^2 + C(ir1) = 87 + 14 = 101. The remainder of the RPT-ADD example would be unaffected by this initial indexing of a^2.

Any order in the instruction repertoire can be used in conjunction with the RPT instruction without causing machine error. However, several of these are ineffective. Any input or output instruction when used with RPT *will ignore* the *RPT instruction* and will proceed in its normal mode. The only effect of the RPT will be to set C(ir3) = a + C(ir$i$) and C(ir4) = m.

When a sense or transfer instruction is used with RPT, the machine performs the number of loops specified by RPT prior to the normal execution of the order. When the order is executed C(ir3) = 0, C(ir4) = m.

REPEAT-MOVE

The Move (MOV) and Compare (TRC) instructions are particularly powerful when used with RPT. The RPT-MOV combination facilitates the transfer of an arbitrary number of words from one part of memory to another.

The MOV instruction is a two address instruction. Therefore, the repetition count and the two effective address modifiers (one for each address) must be specified. The first effective address is initially speci-

fied by *a* of the MOV instruction and is subsequently modified by C(ir4). The second effective address is initially specified by *i-m* of the MOV instruction and is subsequently modified by C(ir2). The MOV instruction itself is unaltered by the RPT-MOV function. A diagram of the RPT-MOV instructions are shown in Figure III-11. The ir3 is still used as the counter.

Example: Move every other number in position 100_{10}-200_{10} inclusive to every 4th location starting at 300_{10}.

The RPT instruction will load ir3 which determines how many times the MOV instruction will be REPEATED and ir4 which determines the effective address a of the MOV instruction, but prior to the RPT ir2 must be loaded independently. For this problem ir2 is loaded with 4. Before the REPEAT instruction the contents of the various index registers are:

i r 1	i r 2	i r 3	i r 4
0	4	0	0

The REPEAT and MOV instructions are now performed.

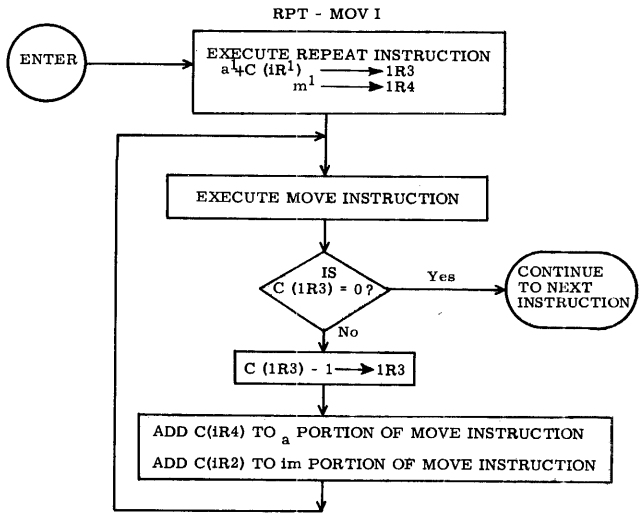


Figure III-11. RPT-MOV Instruction, Flow Diagram

RPT a^1, i^1, m^1	RPT 50, 0, 2				
MOV a^2, i^2, m^2	MOV 100, 300				
		i r 1	i r 2	i r 3	i r 4
		0	4	$(50)_{10}$	2
MOV a^2, i^2m^2	MOV 100, 300	0	4	$(50)_{10}$	2
MOV a^2 + c(i r 4), i^2m^2 + c(i r 2)	MOV 102, 304	0	4	$(49)_{10}$	2
MOV a^2 + 2·c(i r 4), i^2m^2 + 2·c(i r 2)	MOV 104, 310	0	4	$(48)_{10}$	2
MOV a^2 + 3·c(i r 4), i^2m^2 + 3·c(i r 2)	MOV 106, 314	0	4	$(47)_{10}$	2
. . .	. . .	.	.	.	.
MOV $a^2 + (50)_{10}$·c(i r 4), $i^2m^2 + (50)_{10}$·c(i r 2)	MOV 200, 500	0	4	0	2

From the above it is seen that the instruction is executed a^1 + C(iri) + 1 times or 51 times. The a^2 part of the MOV instruction is increased by m^1 of the repeat instruction, after each execution of that instruction. The i^2m^2 of the MOV instruction is indexed by ir2 on each execution after the first. Before each MOV instruction is executed its effective a is placed in Q.

REPEAT-COMPARE

The TRC instruction can be used in conjunction with RPT to compare the contents of the Accumulator and a specified sequence of incremented Memory Locations.

The RPT-TRC function is primarily designed for table look-up where the data is placed in ascending numeric order. However, its use is not restricted and is applicable to a wide range of situations including the rearrangement of input or output data.

In the following sequence of instructions assume that the data to be compared is in the Accumulator. The location of the TRC order in Memory is designated symbolically by "y".

Since the TRC order can be indexed it will be executed using the effective addresses $a^2 + C(iri^2) + nm^1$ where n = 0, 1, 2, and so forth, it will be executed $a^1 + C(iri^1) + 1$ times.

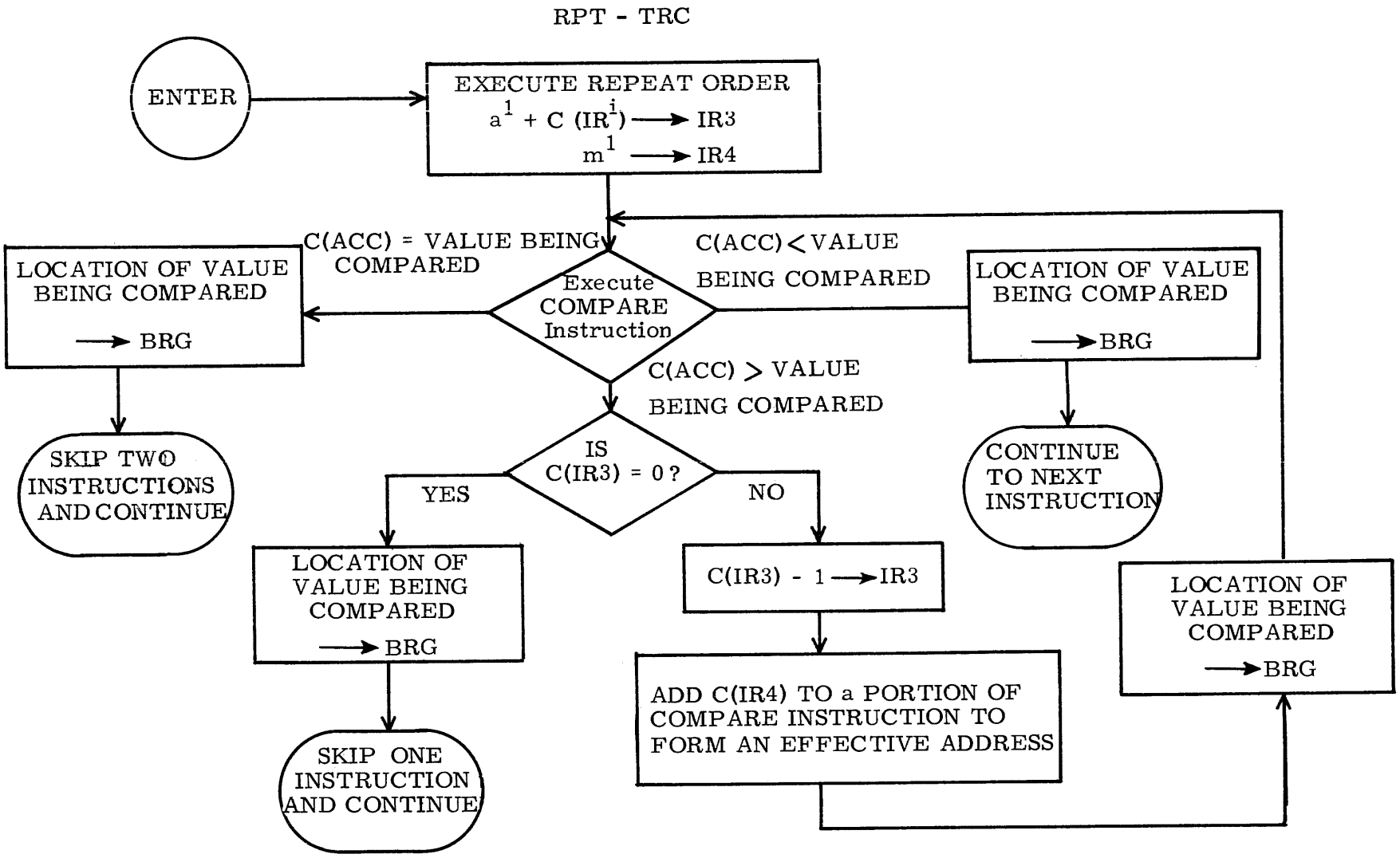


Figure III-12. RPT-TRC Function, Block Diagram

Location	*Instruction*
y − 1	RPT a^1, i^1, m^1
y	TRC a^2, i^2
y + 1	Sub-routine initiating instruction
y + 2	Sub-routine initiating instruction
y + 3	Sub-routine initiating instruction

The RPT order loads a^1 + C(iri^1) into ir3 and m^1 into ir4. The contents of the Accumulator are compared to the contents of location a^2 + C(iri^2) initiating one of the following three lines of action. A block diagram of the RPT-TRC function is shown in figure III-12.

Let x equal the effective address portion of the TRC instruction. It is the location at the word currently being compared. Initially x = a^2 + C(iri^2); as the RPT-TRC operation proceeds, x is incremented successively by C(ir^4).

Then, if C(acc)= C(x), x is stored in the B-register and the loop ends with a transfer to location y + 3; that is, the program counter is incremented by three, to y + 3. The termination is the result of finding the desired equality.

If C(acc) < C(x), x is stored in brg and the loop ends with a transfer to location y + 1. In a table look up operation, this case corresponds to one where the value compared against the table is smaller than any of the functional values stored in memory in increasing order to magnitude.

If C(acc) > C(x), repeat comparison. The C(ir3) is tested and if greater than zero the Address Register is incremented by C(ir4) and the TRC loop repeated. The looping ceases when C(ir3)= 0 or one of the other branch conditions is reached. At the time of the first execution of TRC the C(acc) is also placed in the Q-register where it remains until disturbed by later instructions.

If a program interrupt occurs during a Repeat-Compare sequence, the effective address in the B-register will be destroyed. To prevent the destruction of required B-register contents, program interrupt should be delayed until the contents of the B-register are stored in some memory location. The following set of instructions should be included in the program when using RPT-TRC, if the contents of the B-register are important:

```
    SET SPI
    RPT                 A1  CLA BRG
    TRC                     STR (Memory Location)
    TRU A1                  RESET SPI
    TRU A2                  TRU A3
    CLA BRG             A2  CLA BRG
    STR (Memory Location)   STR (Memory Location)
    RESET SPI               RESET SPI
A3  Next Instruction        TRU A3
```

TRAPPING MODE

GENERAL DESCRIPTION

The Trapping Mode is a special mode of Program Interrupt (see Section IV, INPUT-OUTPUT SYSTEM). Since it is concerned exclusively with Central Processor operations, however, it is discussed in this section.

The Trapping Mode is a tool for checking the progress and accuracy of a program without the necessity of stopping the automatic operation of the computer. When a program is running in the Trapping Mode, it

Symbolic m Portion	*Bit m_{20} of TRU*	*Bit m_{21} of TRU*	*TRA Sw. Before*	*Trapping Action?*	*Effect on Program Counter**	*TRA Switch After*	*Comments or Summary*
0 or 2	0 or 1	0	0	No	a + C(ir$i$) → pct	0	No special action
0 or 2	0 or 1	0	1	Yes	C(pct) → brg 0 → pct	1	Order is trapped*
1	0	1	0 or 1	No	a + C(ir$i$) → pct	1	Leaves TRA SET
3	1	1	0 or 1	No	a + C(ir$i$) → pct	0	Leaves TRA RESET

NOTE: When a transfer order is trapped, TRA is left SET, and both TOT and SPI are SET.
* Only if SPI is RESET.

Table III-5. Trapping Mode Control

is interrupted at certain points, and control is transferred to a specific memory location. Beginning at the location to which control is transferred, the programmer can have stored a program, or subroutine for evaluating the status of his main program. Once the program or subroutine has been completed, control is normally returned to the main program until another "trap" occurs. By proper use of trapping, the programmer can obtain data pertaining to the dynamic operation of his program for debugging purposes.

TRAPPING MODE OPERATION

When the Central Processor is executing a program in the Trapping Mode, the program is interrupted whenever a transfer instruction is encounterd. For example, if the Central Processor is to execute a series of instructions, one of which is a transfer instruction, the computer will "trap" as soon as it has decoded the transfer instruction. Before the transfer instruction is executed, the program sequence is interrupted and control is transferred to Core Memory Location 00000. At the same time, the address (i.e., location) of the transfer instruction which caused the trap is stored in the B-register.

When a program is trapped, the programmer procides a SNR instruction to see whether TOT is set. If it is he transfers control to another memory location which carries out a series of special operations. If it is he transfers control to another memory location which carries out a series of special operations. If TOT is not set some other type of program interrupt caused the trapping action (see Program Interrupt). For example, the trap program may cause the contents of all registers to be output. The final action of the trap program is normally to return control to the main program (making use of the address saved in the B-register to do so), so that it may continue in sequence.

TRAPPING MODE CONTROL SWITCHES

Trapping operations are controlled by two switches, one of which (TOT) is completely under electronic control, and another (TRA) which may be operated either manually from the Console or automatically by the program. The Trapping Mode Activity Switch (TRA) determines whether or not the computer is to operate in the Trapping Mode. When TRA is SET, Trapping Mode is specified. Trapping can only occur when TRA has been SET prior to the decoding of a transfer instruction. When TRA is SET and the Central Processor encounters a transfer instruction, an internal switch, Transfer Order Trapped (TOT), is SET automatically. If TRA and TOT are both SET, the Central Processor program becomes trapped and control is transferred to memory location 00000.

The one transfer-type instruction which does not always cause trapping during the Trapping Mode is *Unconditional Transfer* (TRU). This is because the TRU instruction is used to SET or RESET TRA, depending upon the contents of the *m* portion of the TRU instruction itself. Table III-5 lists the various configurations of the *m* portion of the TRU instruction and the effect these configurations have on the status of the TRA switch, together with the operation of the TRU instruction during trapping operations.

The TRA switch may also be SET and RESET from the Console (see Section VI, CONSOLE). Even though the TRA switch may have been operated upon by the Console control, it may still be altered by the program.

STOP PROGRAM INTERRUPT

In order to prevent trapping while the computer is executing a trap program (in other words, a trap within a trap), a Stop Program Interrupt Switch (SPI) is provided. SPI is always SET automatically when a program is trapped. It is the programmer's obligation to RESET it at the end of the trap program, otherwise all further trapping operations will be inhibited. SPI may be SET, RESET and SENSED by the *Sense Instructions* (see Section III, CENTRAL PROCESSOR INSTRUCTIONS).

In addition to RESETTING the SPI at the end of a trap program, the programmer must also RESET the Transfer Order Trapped Switch (TOT) before returning control to the main program. If he neglects to do so, the program will be caught in a "loop".

Additional types of program interrupt, as related to input-output operations and alarm conditions are discussed in Section IV, INPUT-OUTPUT SYSTEM, under PROGRAM INTERRUPT.

SECTION IV

INPUT-OUTPUT SYSTEM

GENERAL DESCRIPTION

The Sylvania 9400 Data Processing System employs a variety of peripheral input-output devices. The input-output devices operate at varying speeds, both with respect to each other and to the Central Processing Unit. Further, the format of the information processed by each type of input-output device is unique and different from the internal binary formats employed by the computer.

The differences in operating speeds require that the Central Processing Unit be, during memory access times, synchronized with the particular peripheral device involved. The differences in data format require that the information exchanged between the Magnetic Core Memory Unit and the external devices be converted to the appropriate format and code. The 9400 Input-Output System performs the above conversions and synchronization. In addition, the Input-Output System allows input-output operations to occur simultaneously with Central Processor operations and with each other.

The 9400 Input-Output System consists basically of In-Out Processors, each with its own In-Out Transfer Bus; Device Switching Units, Buffers, and the peripheral devices themselves. Each In-Out Processor may communicate with each input-output device. The 9400 System may have up to 64 input-output devices attached to it.

The relationship of 9400 Input-Output System to the Magnetic Core Memory Unit is shown in Figure IV-1. Data words and instruction words are transferred between the Core Memory Unit (or the addressable registers in the Central Processor) and the Input-Ouput System via the Main Transfer Bus. Instruction words are sent from Core Memory and various switches, respectively, to the Control Section of an In-Out Processor. Data is exchanged between the Data-Handling Section of a Processor and the Core Memory Unit (or addressable registers).

Information exchanged between a device and an In-Out Processor travels along the In-Out Transfer Bus attached to the Processor. The Device Switching Units determine which Processor is attached to the particular device addressed.

INTERPRETATION OF A WORD FROM MEMORY

Instruction words and data words transferred from Core Memory (or an addressable register), to the Input-Output System are merely groups of 37 bits. An instruction is indistinguishable from a data word until it leaves the Memory Unit. Like the Central Processing Unit, an In-Out Processor of the Input-Output System interprets a word according to the time it receives it. Thus, the Input-Output System employs a *control cycle* and a *data cycle.*

During the control cycle, a word is sent from Core Memory through the Central Processor Control Unit and placed in the Control Section of the selected Processor. During a data cycle, the word received from memory is treated as data and transmitted to the appropriate input-ouput device.

INTERPRET SIGN

During input-output operations, the status of the Interpret Sign switch (ISN) determines the manner in which data words are to be interpreted. The ISN switch may be SET, RESET and interrogated by the various SENSE instructions (see Section III, CENTRAL PROCESSOR INSTRUCTIONS). If the ISN switch is RESET, words exchanged between the Magnetic Core Memory Unit (and addressable registers in the Central Processor) are considered signless. If ISN is SET, the signs of words exchanged with the Input-Output System are transmitted along with their magnitude portions.

An exception to ISN control is in the *octal* mode of operation. Exchange of octal information between the Input-Output System and the "central" computer is always in interpret sign mode (refer to INPUT-OUTPUT INSTRUCTIONS, specifically READ OCTAL (ROK) and WRITE OCTAL (WOK)).

IN-OUT PROCESSORS

The prime controlling element of the Input-Output System is the In-Out Processor. A 9400 System may have from one to four Processors. Each Processor is capable of connecting, through its own In-Out Transfer Bus, any input-output device with the

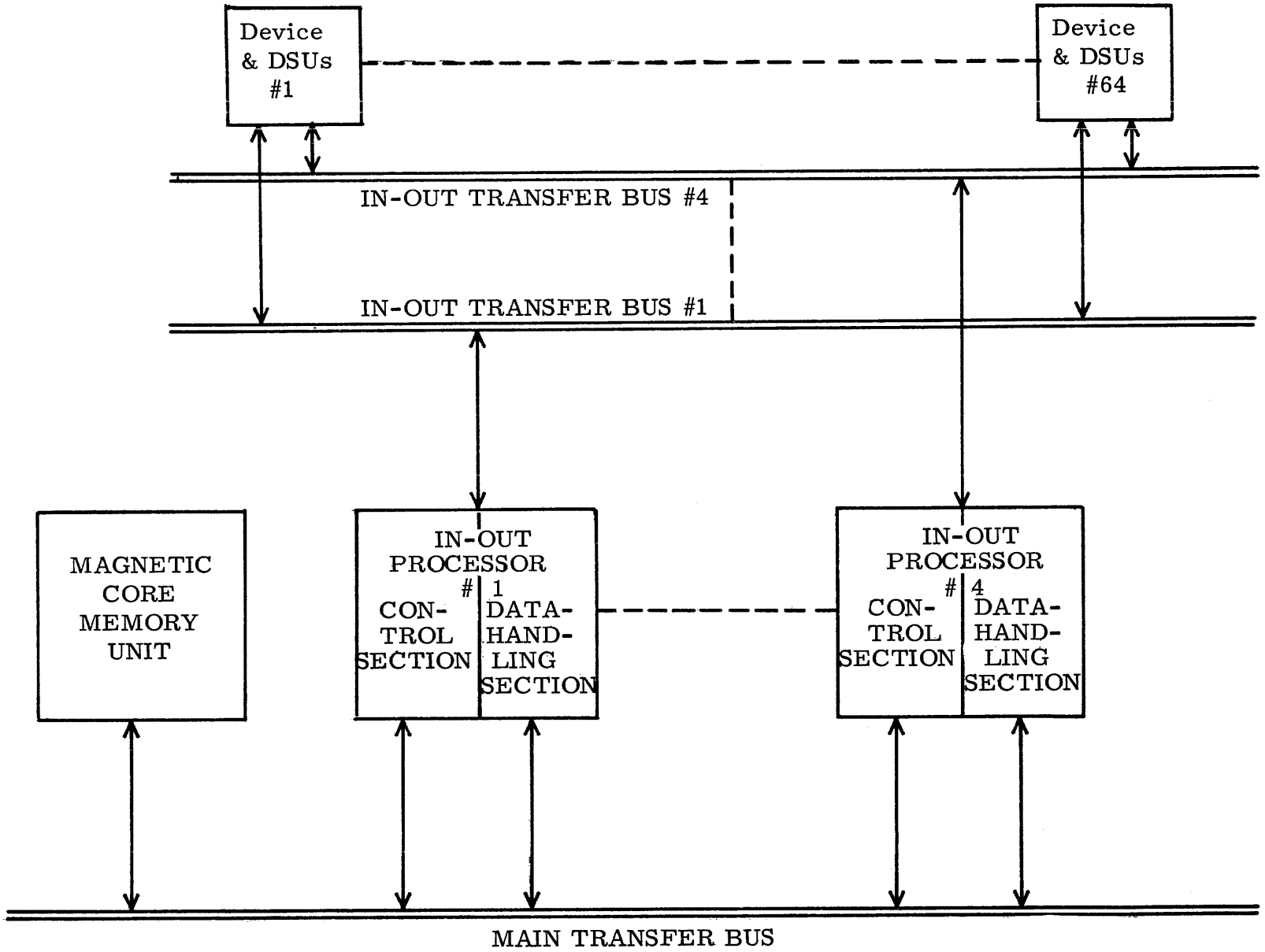


Figure IV-1. Input-Output System, Block Diagram

Magnetic Core Unit or with an addressable register in the Central Processing Unit. As many input-output devices as the System has In-Out Processors may operate simultaneously. Each In-Out Processor consists of a Control Section and a Data-Handling Section (see Figure IV-2).

The Control Section is made up of two groups of registers: the Processor Instruction Register (PIR) and the Order Sequence Registers (DAS and OSR).

The PIR register, which is addressable as a *unit,* is broken down into four sub-registers, as follows:

1. The Instruction Register (ISR) holds the current input-output operation code, where it is interpreted and implemented by the In-Out Processor.

2. The Word-Block Counter (WBC) contains the count of the number of pieces of data to be transferred between the Input-Output System and the central computer. During the execution of an input-output operation, the contents of WBC are decremented by ONE each time one complete 9400 character, word, or block of words (depending upon the mode of operation) is transferred between the peripheral device and the central computer. Normally, when the contents of the Word-Block Counter reach ZERO, the current input-output operation is terminated.

3. The Device Address Register (DAR) contains the address of the selected input-output device.

4. The Address Counter (ADC) holds the first address to or from which data is to be transferred. When the address contained in ADC specifies a core memory location, the contents of ADC are incremented by ONE following the transfer of each 9400 character or word between the input-output device and the central computer (An exception is READ REVERSE (REV), where the contents of ADC are *decremented* by ONE each time). If an addressable register is specified, the contents of ADC are not changed during the input-output operation.

The Order Sequence Register (DAS) stores the various options pertaining to the input-output operation being carried out. The Order Sequence Register (OSR), like the Program Counter (PCT) for the Central Processing Unit, stores the address of the next Order Sequence Order to be executed (see ORDER SEQUENCE MODE).

The Data-Handling Section of the In-Out Processors consists of six registers (see Figure IV-2). The Buffer Register (BFR) is capable of storing a complete 37-bit word. The function of this register is to provide a link and word-assembly point between the Central Processing Unit and the input-output devices.

Data is exchanged between the In-Out Processors and all input-output devices in the form of eight-bit characters. Six of the bits in each eight-bit character are *data* bits, the seventh is a *control* bit, and the eighth is a *parity* bit. Information being sent from the Central Processing Unit to an output device must be divided into six-bit characters with the appropriate control and parity bits added to form eight-bit characters. Conversely, information being read from an input device must be assembled into standard 37-bit words. The control and parity bits of incoming characters are interrogated and removed as they arrive at the In-Out Processor.

The above operations are carried out in the Data-Handling Section of the In-Out Processors. The five Character Buffer Registers (CIR, BSR, BCR, BXR, and TAR) are involved in the character-to-word and word-to-character conversion. They transfer data to and from the Buffer Register one character at a time. In addition, they check and generate parity bits and special control characters, such as block-start and block-end markers.

A Device Switching Unit (DSU) is essentially a gate between each input-output device and each In-Out Processor. For example, in a 9400 System with three In-Out Processors, each peripheral device is "gated" to each of the three Processors through a separate Device Switching Unit.

INFORMATION FLOW IN THE INPUT-OUTPUT SYSTEM

PROCESSOR AND INPUT-OUTPUT DEVICE SELECTION

When the Central Processing Unit encounters an input-output instruction during the execution of a program, an attempt is made to select an In-Out Processor. If a Processor is available, and if the device selected is also available, the Processor is selected and the input-output instruction is transferred, via the Main Transfer Bus, to the appropriate PIR register. If the selected device is busy, or if no Processors are available, the central processing program is interrupted if the device busy or processor busy decision switch is SET and SPI switch is RESET, otherwise it hangs up during Timing Function 8, see under PROGRAM INTERRUPT. The input-output operation which caused the interrupt may be executed as soon as both the device and a Processor become available. Other input-output operations are not held up, however.

Once the input-output instruction has reached the In-Out Processor, all transfers and control signals are made along the In-Out Transfer Bus associated with the selected Processor.

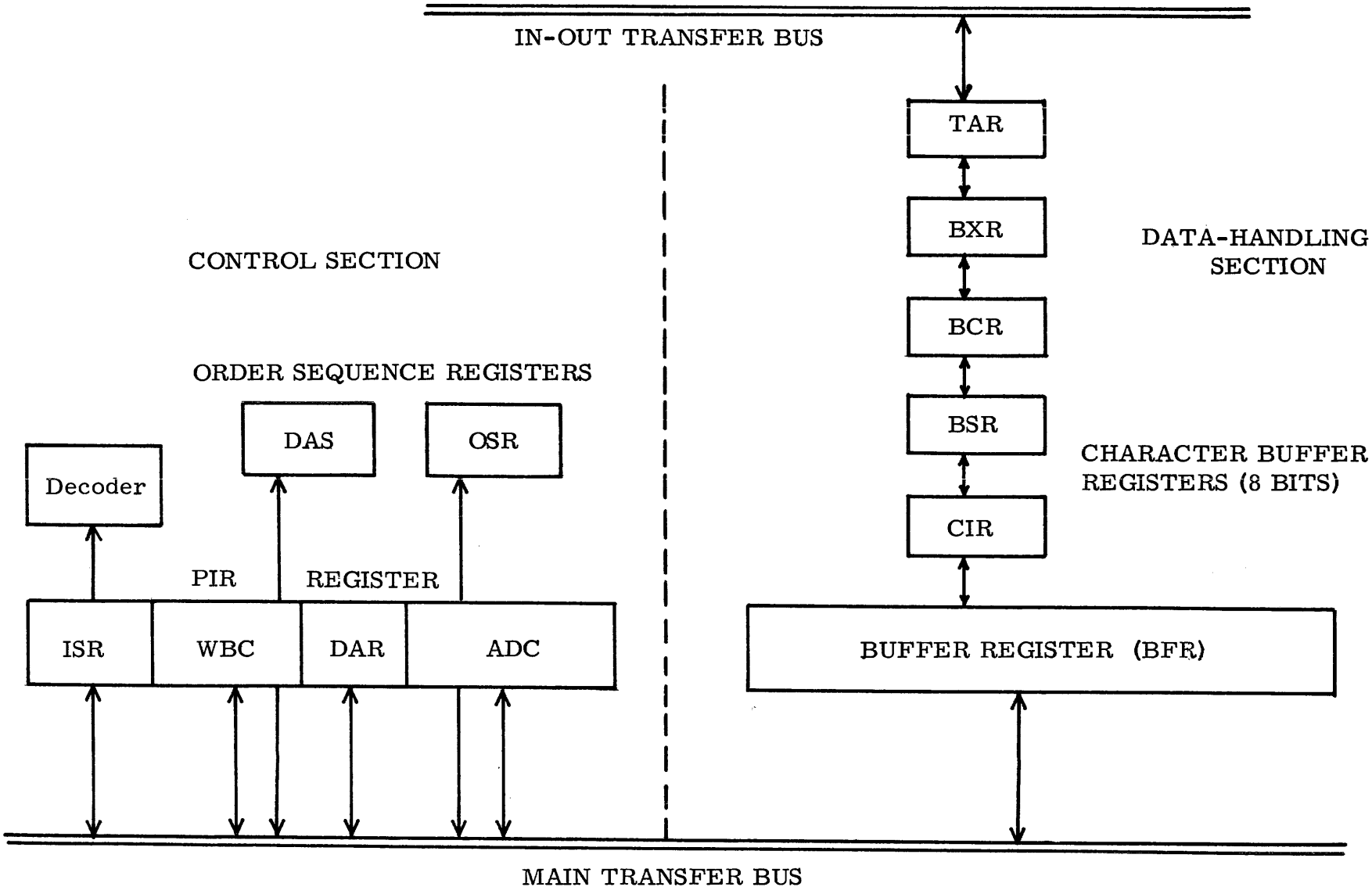


Figure IV-2. In-Out Processor, Block Diagram

WRITE OPERATION

A write operation consists of the transfer of information from the Magnetic Core Memory or from an addressable register, to an output device. An inherent part of the write operation is the format and code translation of the output information.

Following Processor and device selection, a write operation begins with the starting of the selected device, and with the parallel transfer along the Main Transfer Bus of a 9400 word from the Memory Unit or the Central Processing Unit to the Buffer Register (BFR). The address of the selected word is contained in the ADC portion of the PIR register.

Once the word reaches BFR, and in synchronization with the output device, it is broken down into six-bit groups, starting at the high-order end of BFR. If the sign is to be included, it is also converted into a standard six-bit code (see INTERPRET SIGN). As each six-bit group leaves the Buffer Register, it passes sequentially through the five Character Buffer Registers. During its passage, appropriate control and parity bits are added to it. Finally, the newly-formed eight-bit characters are transferred, one at a time, out from the TAR register onto the In-Out Transfer Bus. The Device Switching Unit allows the characters to pass through to the appropriate output device.

READ OPERATION

A read operation involves the transfer of information from an input device to the Magnetic Core Memory Unit or to an addressable register. As in an output operation, format and mode translation of the transferred information takes place.

Once the Processor and device have been selected, the read operation begins by starting the appropriate input device. As eight-bit characters arrive at TAR on the In-Out Transfer Bus, they are transmitted sequentially through the five Character Buffer Registers toward BFR. Each character is interrogated and checked for parity in the CIR register before being transferred into the BFR.

Once an incoming character leaves CIR, it is stripped of the control and parity bits, allowing a six-bit character (three bits for octal characters, see specific octal input and output instructions) to enter BFR at the low-order end. As each new character reaches BFR, the preceeding character is shifted left to make room for it. When BFR is filled, the entire newly-formed word is transferred, in parallel, along the Main Transfer Bus, to the location specified by the contents of ADC.

SINGLE INSTRUCTION MODE

GENERAL DESCRIPTION

The 9400 Input-Output System operates in two modes, the Single Instruction Mode (SIM), and the Order Sequence Mode (OSM). In SIM, a single input-output *instruction* is initiated as part of the main program sequence. In OSM, which is described under ORDER SEQUENCE MODE, a series of input-output *orders* are executed in parallel with the main program.

The Single Instruction Mode operates under the control of nine input-output instructions. Each instruction, according to its requirements, contains an operation code, the "address" of the selected input-output device, the address of the data to be transferred, and a count of the amount of information to be transferred. When the Central Processor encounters an input-output instruction in the main program sequence, the required input-output device is connected to the "central" computer through an In-Out Processor. Once the Processor has been connected, the main program sequence resumes its operation. Normally, Processor selection does not delay the main program.

The input-output operation causes the In-Out Processor to transfer information between the Core Memory Unit (or addressable registers) during the execution of the main program. The Input-Output System is synchronized with the Central Processor only when the Input-Output System requires access to Core Memory or to an addressable register.

An input-output instruction causes a single input-output operation, in that as soon as the required amount of data has been transferred between the Input-Output System and the "central" computer, the In-Out Processor, and hence the input-output device, are logically disconnected. Another input-output instruction following in the main program sequence will initiate a new Processor and cause a device to be connected again.

INSTRUCTION WORD FORMAT

The format for a standard 9400 input-output instruction is shown in Figure IV-3. As with the Central Processor instruction word, the sign bit, S, of the input-output instruction word is ignored. The magnitude portion of the input-output instruction is broken up into four sections, as follows:

1. The Operation Code (OP) is contained in bits 1 through 6 of the instruction word.

2. The Word-Block Count (c) occupies bits 7 through 15. The word-block count determines how many 9400 characters, words, or blocks of words are to be transferred between the selected input-output device and the Central Processing Unit.

3. The Selected Device Code (s) occupies bits 16 through 21. Each input-output device is identified by its own unique six-bit code. The s portion of an input-output instruction word identifies the input-output device which has been selected.

4. The Address Portion (a) is contained in bits 22 through 36. Initially, the a portion of an input-output instruction word specifies the address in core memory (or adressable register) of the first word to be processed by the input-output operation.

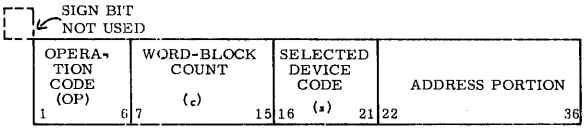


Figure IV-3. Standard Input-Output Instruction Word Format

INPUT-OUTPUT ALARMS

The No Halt on Processor Error Switch (NHP) is used to prevent the halting of the computer if an In-Out Processor error is detected. NHP has to be SET before each input-output instruction for which In-Out Processor errors are to be ignored. NHP is addressable and may be SET by an SNS instruction.

When an input-output error is detected the appropriate Input-Output Alarm (IOA), is immediately SET. Subsequent action depends, in general, on the status of NHP. If NHP is SET, execution of the input-output instruction involved will be continued. However, if NHP is RESET, any data transmission implied by this input-output instruction will cease, the instruction will continue until its normal termination, and the computer will halt.

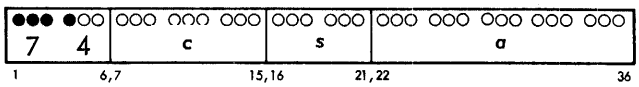
Some of the possible input-output alarms, and their results are listed below:

1. Input Parity Error (IPE)· If there is an input parity error, the IOA is SET immediately.
2. Interpret-Sign Error (ISE) If characters have been written in the non-ISN mode, and a read operation in the ISN mode is attempted, the IOA is SET immediately upon discovery of an illegitimate character in the sign position.
3. Improper Order (IMO) If the instruction in the Processor is meaningless, the IOA is SET.
4. Device Alarm (DVA) A device failure (e.g. power loss) will SET the IOA and the computer will stop, regardless of the condition of NHP. The Processor will disconnect immediately.
5. Timing Read Error (TRE) If there is a block mark out of place on the tape, or an illegitimate character is detected in the middle of a word (e.g. a non-octal character in the middle of an octal word when using the ROK order), IOA is SET.

OUTPUT INSTRUCTIONS (see table IV-1, IV-3, IV-4, IV-5)

Output instructions WRITE data on output devices such as magnetic tape, high speed printers, paper tape, punched cards, or typewriters. Each output instruction specifies how much data is to be written on what output device and where the data is found — usually where the first of a series of data words is located in memory.

WAN a,s,c WRITE ALPHANUMERIC INSTRUCTION



WAN writes c (up to 511) words on output device s from sequential memory locations beginning at a.

NON-INTERPRET SIGN

In this mode, the sign of C(a) is ignored. The In-Out Processor divides each memory word into six 6-bit units. The six high-order bits of the memory word make up the first 6-bit unit.

INTERPRET SIGN

If the Interpret-Sign mode is selected, the Interpret-Sign switch must be set by an SNS instruction prior to WAN. In the Interpret-Sign mode, the In-Out Processor divides each memory word into seven 6-bit units. The sign is interpreted and translated as the first of the seven 6-bit units. The six high-order bits of the memory word make up the second 6-bit unit.

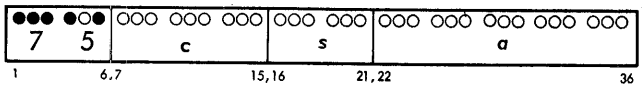
WRITE VARIABLE LENGTH BLOCKS

The WAN instruction will write any length block up to 511 words. If the device specified by s is a magnetic tape unit, special Block-Start (BLS) and Block-End (BLE) marks are written automatically before the first word and after c words, respectively, for each WAN instruction. It is possible to re-write blocks written on magnetic tape as described under the RE-WRITE (WWA) instruction.

WRITE FILES

It is frequently desirable to combine a variable number of blocks into larger record groups called files. The programmer need only specify that an End-of-File (EOF) mark be written by the last WAN instruction used in writing the file; the Write End-of-File switch (WEF) must have been SET prior to executing the last WAN instruction used in writing the file. WEF is sensable. It is RESET automatically upon transfer of the WAN instruction to the In-Out Processor. EOF marks are automatically written after each WAN, instead of BLE if WEF was SET prior to the instruction.

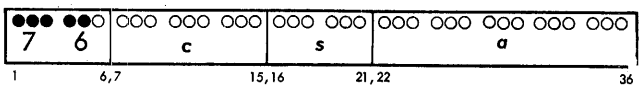
WWA a, s, c REWRITE ALPHANUMERIC INSTRUCTION



The WWA instruction writes *c* (up to 511) words on the magnetic tape specified by *s*. WWA enables information to be rewritten in existing blocks. WWA must write the same number of words, as were contained in the original block.

WWA starts in the read mode and searches forward on magnetic tape until the first Block Start mark is detected. Thereafter, it performs the same as the WAN instruction. If a BLS is not detected, the whole tape will be searched.

WOK a,s,c WRITE OCTAL INSTRUCTION



WOK writes *c* (up to 511) words on output device *s* from sequential memory locations beginning at *a*. It is similar to WAN except that it always writes in the Interpret-Sign mode, regardless of the state of the ISN switch.

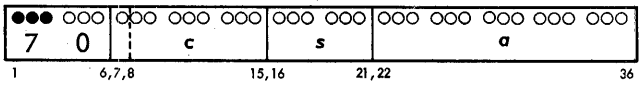
Each memory word is written as thirteen 3-bit octal numbers. The sign bit of the memory is expanded to a 3-bit octal number. The next 12 octal characters comprise the 36 data bits of the word.

WOK may *not* be used with magnetic tape or the card punch or Mass Memory.

INPUT INSTRUCTIONS (see table IV-1, IV-2, IV-3)

Input instructions READ data from input devices. Each input instruction specifies how much data is to be read from what input device and where the data is to be placed, usually the address of the first word of a series of data words is to be placed in memory.

RAN (c) a,s,c READ ALPHANUMERIC INSTRUCTION



MAGNETIC TAPE UNIT OR MASS MEMORY

If *s* specifies a magnetic tape unit or a Mass Memory Unit, the RAN reads to an EOF mark or *c* (up to 255) blocks or words into consecutive memory locations starting at address *a*. If bit 7 is a one, then the remaining eight bits in *c* determine the number of blocks to be read; if this bit is a zero, then the other eight bits determine the number of words to be read. When an EOF mark is detected during a RAN instruction the EOF switch in the Central Processor is SET and the instruction is terminated.

PAPER TAPE READER

If *s* specifies a paper tape reader the RAN reads up to 511 words inclusive or until STOP CODE is encountered if the Interpret Sign Mode is specified. The RAN reads until the STOP CODE if bit 7 is a "1" and Non Interpret Sign is specified. The RAN reads up to 255 characters if bit 7 is a "0" and Non Interpret Sign is specified.

CARD READER

RAN reads up to 255 cards from the Card Reader specified by *s* into sequential memory locations starting at *a*.

NON-INTERPRET SIGN

In the Non-Interpret-Sign mode six 6-bit units are assembled and read into memory as a 37-bit word. The sign bit position is made ZERO. The first 6-bit unit occupies the six high-order bits of a full memory word.

INTERPRET SIGN

In the Interpret-Sign mode the first of every 7 units comprising a word is interpreted as the sign of the word when read into memory. The next six 6-bit units comprise the data. The second 6-bit unit makes up the six high-order bits of a full memory word.

Information is usually read in the same mode in which it was written. If an attempt is made to read data in the Interpert-Sign mode when the data was written in the Non-Interpret-Sign mode, every seventh unit is interpreted as a sign. If a unit which is not a sign character is interpreted as a sign, the Interpret-Sign-Error switch (ISE) will be SET, and the 6-bit unit will be lost.

READ FILES

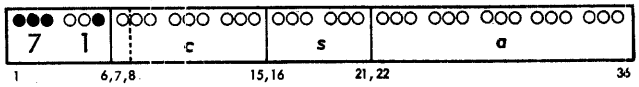
When an EOF mark is detected during a RAN instruction, the EOF switch in the Central Processor is SET and the instruction is terminated.

EXTENDED RAN INSTRUCTION

RAN may be written as an extended instruction by adding one of the termination control suffixes shown below to the basic mnemonic operation code.

Control Symbols	*Control*	*Control Bits*
		0 c,
RAN-		Bit 7
W	Read c words	0
B	Read c blocks	1
Examples:	RANW, RANB	

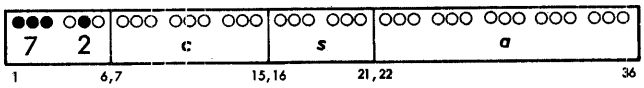
RRV(c) a,s,c READ REVERSE INSTRUCTION



MAGNETIC TAPE

RRV reads *c* (up to 255) words or blocks in reverse direction from magnetic tape unit *s* and stores them in decreasing memory locations starting at *a*. Words are arranged in memory in the reverse direction as read. Therefore, the data is stored in memory in the same order as if it had been read forward by a RAN instruction. Data cannot be read reverse into memory location 0-40. If the tape is still traveling in the reverse direction at the beginning of tape DVA will occur.

ROK a,s,c READ OCTAL INSTRUCTION



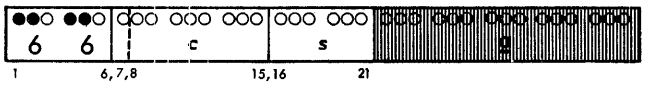
ROK reads *c* (up to 511) octal words in the Interpret-Sign mode from Paper Tape Reader *s* into sequential memory locations starting at *a*.

ROK is primarily used to convert characters on paper tape directly into machine code, and is not applicable to magnetic tape. During the reading operation each character is converted into its octal equivalent. A computer word is assembled from 13 octal numbers, with the low-order bit of the first octal number interpreted as the sign of the word. The remaining 12 octal numbers comprise the 36-bit word. If a STOP code is encountered before *c* words are read, the operation terminates.

DEVICE CONTROL INSTRUCTIONS

The device control instructions are magnetic tape instructions which initiate or control tape motion.

SKP (c) s,c SKIP INSTRUCTION



SKP causes magnetic tape *s* to skip forward *c* (up to 255) blocks or files. If bit 7 is a "1", *c* determines the number of files to be skipped. If bit 7 is a "0", *c* determines the number of blocks to be skipped. If the tape has reached the physical end and no files appear a DVA will occur.

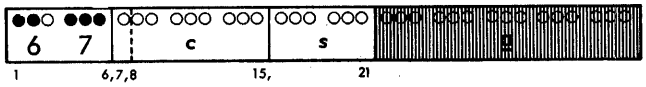
EXTEND SKP INSTRUCTION

SKP may be written as an extended instruction by adding one of the following termination control suffixes shown below to the basic mnemonic operation code.

Control Symbols	*Control*	*Control Bits*
		0
		c,
SKP-		Bit 7
B	Skip c blocks	0
F	Skip c files	1

Examples: SKPB, SKPF

BSP(c) s,c BACKSPACE INSTRUCTION

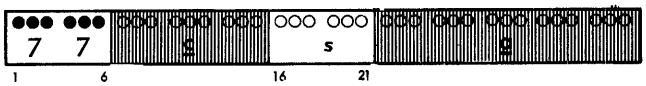


BSP backspaces *c* (up to 255) blocks or files on magnetic tape *s*. BSP is identical to SKP except backspace skips in the reverse direction.

When backspacing files, the tape stops at the beginning of the block containing the EOF mark. Therefore, in order to backspace "n" files, the programmer must backspace n + 1 files and skip one block.

BSP may be written as an extended order in the same manner as SKP. See SKP instruction for control symbols.

RWD s REWIND INSTRUCTION



RWD rewinds Magnetic Tape device *s*. Once rewind has begun the In-Out Processor disconnects and the magnetic tape transport completes the rewind independent of external control. The device performing the rewind cannot be re-selected until rewinding is completed. If the tape is already rewound when giving this instruction rewind behaves as if it is just completed.

ORDER SEQUENCE MODE

GENERAL DESCRIPTION

In the Order Sequence Mode (OSM), as opposed to the Single Instruction Mode (SIM), the 9400 Input-Output System executes input-output *programs* in parallel with the Central Processor program. As in Single Instruction Mode, an In-Out Processor is selected and connected to the addressed input-output device. Once a Processor has been selected, the Order Sequence Mode of operation allows the Processor to execute a *series* of input-output *orders* (an Order-Sequence program) while the main program in the Central Processor proceeds simultaneously. As many input-output programs may be in operation simul-

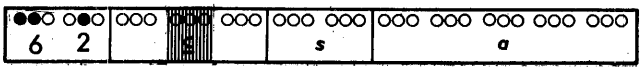
taneously as there are In-Out Processors, all of them in parallel with the main program.

The Order Sequence Mode operates under the control of Order Sequence Orders. The Order Sequence Orders make up an input-output program. A special instruction, Start Order Sequence (SS), is used to initiate the execution of the input-output program. To begin an Order Sequence program, the programmer includes an SS instruction in the main program sequence. When the Central Processor encounters the SS instruction, it selects and connects an In-Out Processor to the device addressed by the SS instruction. The address portion of the SS instruction contains the address of the *first* Order Sequence order which is to be executed by the In-Out Processor. Order Sequence programs are stored in Core Memory, as are main sequence programs. Thus, the SS instruction directs the In-Out Processor to the beginning of its Order Sequence program.

During the main program sequence, the In-Out Processor is granted memory access to retrieve the first Order Sequence order. Once the order has been transferred to the Processor, the Input-Output System proceeds independently from the Central Processor except for required data transfer between the Core Memory Unit. As soon as the first Order Sequence order has been completely processed, the In-Out Processor retrieves the next order from memory and processes it. The input-output operation continues until the Order Sequence is complete, at which time the In-Out Processor, and hence the device, are logically disconnected from the "central" computer.

SS AND ORDER SEQENCE ORDER WORD FORMATS

SS a,s,c START ORDER SEQUENCE



The formats for the Start Order Sequence instruction and the Order Sequence orders are effectively the same as the format for the Single Instruction Mode instruction words (see figure IV-3). SS and OSM orders are divided into the OP, *a, s,* and *c* portions like SIM instructions. The significance and usable magnitude of the various portions of the Order Sequence control words varies with the instruction or order, and is described under the individual operations themselves.

PROCESSOR OPERATION IN THE ORDER SEQUENCE MODE

When the Central Processing Unit encounters a Start Order Sequence (SS) instruction in a main program sequence, it attempts to select an In-Out Processor, as in the Single Instruction Mode of operation. A Start Order Sequence instruction specifies which device is to be selected, which Processor is to be used, and where the first order is located in the Order Sequence program. It also may exercise several options pertaining to the input-output operations to follow.

Once a Processor is selected and the appropriate input-output device is established as being available, the SS instruction is transferred, via the Main Transfer Bus, to the appropriate portions of the PIR register and the Order Sequence Registers (see Figure IV-2). The selected input-output device is then activated.

As soon as the SS instruction has been processed by the In-Out Processor, the first Order Sequence Order is retrieved from Core Memory. The address of the order is contained in the Order Sequence Register, which, like the Program Counter in the Central Processor, is advanced by ONE as each order is retrieved from memory unless a conditional transfer is indicated. Retrieval of the order takes place during the data retrieval portion of the basic machine cycle. However, since the Input-Output System is operating in the Order Sequence Mode, the retrieved order is transferred to the Control Section of the selected Processor.

Order Sequence Orders of the data-handling type specify the amount of information to be transferred and the locations to or from which it is to be transferred, much the same as with Single Instruction Mode instructions. When an Order Sequence order has been retrieved by an In-Out Processor, it is transferred into the appropriate portions of the Control Section of the Processor. It is then executed as an individual input-output operation.

Once an Order Sequence order has been completed (with the exception of certain terminal-type orders), the Order Sequence Register in the In-Out Processor specifies the address of the next Order Sequence order which is to be retrieved from Memory. Between the execution of the sequential orders, the Processor and the input-output device remain logically connected to the "central" computer. Thus, it is possible to execute a series of orders, which need not all be of the same type, and remain connected to the input-output device during the entire operation.

In addition to data-handling orders, the Order Sequence Mode includes provision for transfers within the Order Sequence Mode program and communication (including synchronization, if desired) with the main program in the Central Processing Unit.

Other Order Sequence Orders make it possible to write key words on magnetic tape, so that the tape may be searched for specific records. Order Sequence orders also enable the programmer to interrogate the

status of the Input-Output System at any time and, if desired, to interrupt the main program in order to carry out special operations.

INPUT-OUTPUT ORDER SEQUENCE ORDERS

The 9400 Order Sequence orders enable an In-Out Processor to transfer, re-arrange, and edit input-output data using its own order sequence.

An order sequence may contain any number of orders to be performed on one input-output device. If a new device is to be selected, a new order sequence must be initiated.

INTERPRETING SIGNS

In the Order Sequence Mode, the Interpret-Sign Mode is initiated by specifying Interpret-Sign as one of the options of the Start Order Sequence instruction. If Interpret Sign is specified the entire sequence will be in the Interpret Sign Mode and the signs of all data words will be interpreted during the sequence of orders.

INPUT-OUTPUT ALARMS

In the Order Sequence mode, the No Halt on Processor Error switch (NHP) is SET by specifying this as one of the options of the Start Order Sequence instruction.

If No Halt on Processor Error is specified, the entire sequence of orders will be performed by the In-Out Processor even though error controls IMO, ISE, TRE, and IPE may be SET during the sequence by their respective error conditions as described under SINGLE INSTRUCTION MODE.

The programmer may determine the specific error condition (if any) which exists, by including one of the special In-Out Processor sense orders in the sequence, immediately preceding the order terminating the sequence.

The End-of-Tape Alarm (ETA) operates in the same manner as in Single Instruction Mode.

PREPARATORY ORDERS

The preparatory orders initiate the Order-Sequence mode by designating which device is to be used and where the first instruction of the Order Sequence is located in memory. They also enable the program to specify an In-Out Processor.

SS is an instruction which is interpreted by the Central Processor to initiate a sequence of input-output orders in the Order-Sequence Mode. The sequence of orders to be executed starts with the Order Sequence order in memory location *a*.

The input-output device specified by *s* will be used with all orders in the sequence initiated by this SS instruction. The *s* bits of subsequent orders in this sequence serve as control functions.

Note that SS is an *instruction,* which is initiated by the Central Processor. Subsequent operations in the Order Sequence Mode are called *orders* because they are initiated and executed by the In-Out Processor.

Once initiated, the sequence is executed automatically by the selected In-Out Processor. No further attention by either the Central Processor or the operator is required until the sequence is specifically terminated and the In-Out Processor is disconnected. A sequence may be specifically terminated by one of the Order-Sequence orders or by a special End-Order-Sequence Order.

The various computer interrupts are usually enabled by setting the appropriate controls prior to the execution of an SS. However, the condition of these controls may change during the execution of a sequence.

SS also specifies several input-output control functions which will be performed concurrently with the sequence of instructions to be executed.

Extended SS Instruction

SS may be written as an extended instruction by adding any of the following control suffixes. All of these suffixes are optional but if chosen must be used in the sequence shown. Only one In-Out Processor may be selected for the entire Order Sequence. If a non-existent In-Out Processor is addressed or if no In-Out Processor is specified an improper order alarm will occur.

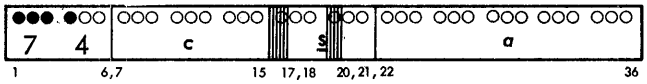
Control Symbols	*Control*	*Control Bits*
SS----		000 000 000 c
		Bit 7 9 13 15
S---	Interpret Sign	1-- 000 ---
-N--	No Halt on In-Out Processor error	-1- 000 ---
--E-	Erase magnetic tape with all write orders	--1 000 ---
---1 through ---4	Use In-Out Processor No. 1 through No. 4	--- 000 --1 --- 000 1--
---7	Use first available Processor	--- 000 111
Examples: SS1, SSN2, SS3, SSNE4, SSN7		

OUTPUT ORDERS

Output orders WRITE data on output devices such as magnetic tape, high speed printers, paper tape, punched cards, or typewriters. Each output order specifies how much data is to be written on the device

specified by the preceding SS instruction, and where the data is to be found or, usually, where the first word of a series of data words is located in memory.

GW (s) a,c GATHER WRITE ORDER



GW causes the input-output device designated by the previous SS to write c words (up to 511) from sequential memory locations beginning at a.

NON-INTERPRET SIGN

In this mode, the sign of $C(a)_{35\ldots}$ is ignored. The in-out processor divides each memory word into six 6-bit units.

INTERPRET SIGN

If SS specifies the Interpret-Sign mode, the In-Out Processor divides each memory word into seven 6-bit units. The sign is interpreted and translated as the first of the seven 6-bit units.

VARIABLE BLOCK LENGTHS

Blocks of different lengths may be generated by varying the number of words specified when writing a block. Thus, a 1000 word block may be written by a sequence of three GW's of 255 words each, and a fourth GW of 235 words. The fourth GW specifies that a Block-end (BLE) is written. If the next order is another GW, and the previous order had written a BLE, then BLS is written automatically as the first character in the next block. No other order may follow a Gather Write except another Gather Write or a Write Key Order Unless a BLE has been laid down.

WRITE FILES

It is frequently desirable to combine a variable number of blocks into larger record groups called files. The programmer need only specify that an End-of-File (EOF) character be written, in place of the BLE, by specifying a "1" in bit 21. The Write Key Order (WK) enables the programmer to make still another record grouping which is identified by a special key word.

GATHER WRITE

The Gather Write function enables information from multiple memory locations to be converged into blocks. Any number of GW orders may be used to write a block, and each GW specifies its own starting location in memory and how many words are to be written. The first GW in a series drops a Block Start (BLS) mark automatically.

EXTENDED GW ORDER

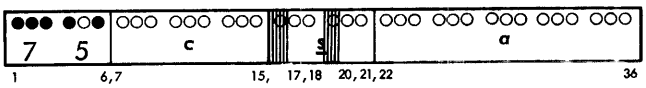
GW may be mnemonically represented as an extended order by adding one suffix for each of the following controls in the sequence shown:

Control Symbols	*Control*	*Control Bits*
		000 000
		s
		Bit 16 21
	Termination	
GW--		
W-	Write c words without termination marks.	000 000
B-	Write c words then write Block End (BLE) or Control Block End (EOC) mark*	0-- 010
F-	Write c words, then write End-of-File (EOF) mark Termination Action	0-- 001
-P	Proceed in sequence unless program interrupt occurs then disconnect	000 0--
-I	Interrupt the Central Processor and proceed	011 0--
-D	Interrupt the Central Processor and disconnect	010 0--
-C	Write a Control Block Start (BLS) C words, without termination marks *	000 100

*A Control Block End will be written if the first gather write of the sequence indicates a Control Block Start.

Examples: GWWP, GWBD, GWFD

WW(s) ac REWRITE ALPHANUMERIC ORDER



The WW order writes c (up to 511) words on the input-output device specified by s of the previous SOS instruction. WW must write the same number of words as were contained in the original block.

WW enables information to be rewritten in existing blocks. WW starts in the read mode and searches

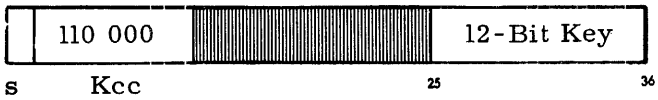
forward on magnetic tape until the first Block Start mark is detected. Thereafter, it performs the same as the GW order, including all control functions. If a BLS is not detected, the whole tape will be searched.

WO a, c WRITE OCTAL ORDER

The WO order is identical to the WOK instruction except that the input-output device is specified by *s* of the previous SOS instruction. See WOK instruction for details.

WK a WRITE KEY ORDER

WK causes the In-Out Processor to write the 12-bit key contained in $a_{25\text{-}36}$ on magnetic tape and transfers program control to the next order which must always be the order GW.



A Key Control Character (KCC) is automatically added to the key written on tape, as shown.

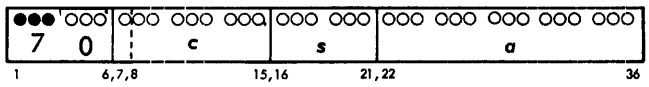
KCC on tape: 10110000. The two high order "1, 0" bits are parity and control, respectively.

The sign is written as the first of seven characters if the Interpret-Sign mode is specified by the previous SOS.

INPUT ORDERS

Input orders READ data from input devices. Each input order specifies how much data is to be read from the device selected by the preceding SOS instruction, and where the data is to be placed — usually, where the first of a series of data words is to be placed in memory.

SC (s,c) a,c SCATTER READ ORDER



The SC order reads from the device designated by the previous SS instruction. SC reads *c* (up to 255) words or blocks or to an End-of-File (EOF) mark. When reading words, if there are less than *c* words in a block, SC will cause subsequent blocks to be read until *c* words are read. EOF and ETA will always terminate a read order immediately. An ETA terminates the Read order when the BLE is Read.

A number of SC orders may be used to disperse variable numbers of words or blocks into many sets of sequential memory locations, each starting at location *a* of the respective SC order. Data cannot be read into addressable registers.

In this manner, Scatter Read enables the programmer to disperse input data, selectively, to a number of assigned memory locations. Similarly, words can be skipped, selectively, by reading in the Store-No-Words option (see Extended SC Order).

If the programmer wishes to read words of a block into various memory locations the termination action should be -- X. Whenever reading blocks or reading the last part of a block after giving SC--X the option SC-I, SC-D or SC-P must be used. Another Read order must always follow SC--X or a Search Key order.

NON-INTERPRET SIGN

In the Non-Interpret-Sign mode six 6-bit units are assembled and read into memory as a word. The sign bit of each word is left positive. The first 6-bit unit occupies the six high-order bits of a full memory word.

INTERPRET SIGN

In the Interpret-Sign mode the first of every 7 units is interpreted as the sign of the word when read into memory. The next six 6-bit units comprise the data. The second 6-bit unit makes up the six high-order bits of a full memory word.

Information is usually read in the same mode in which it was written. If an attempt is made to read data in the Interpret-Sign mode when the data was written in the Non-Interpret Sign mode, every seventh unit is interpreted as a sign. If a unit which is not a sign character is interpreted as a sign, the Interpret-Sign-Error alarm (ISE) will be SET and the 6-bit unit will be lost.

READ FILES

When an EOF mark is detected during an SC order, the EOF switch in the Central Processor is SET and the order is terminated.

If a "1" is placed in control bit 21, an EOF mark will interrupt the Central Processor. The Central Processor program control will be transferred to locations 1, 2, 3, or 4, corresponding to the In-Out Processor which initiated the SC order.

EXTENDED SC ORDER

SC may be written as an extended order by adding the following extended suffixes. One suffix from each of the first three categories must be specified in the sequence shown, the fourth category is optional:

Control Symbols	*Control*		*Control Bits*
		c	s
		Bit 7	16 21
SC----			
	Termination		
W---	Read Forward and terminate after c words	0	0-- ---
B---	Read forward and terminate after c blocks	1	0-- ---
S---	Read forward and terminate after c words or BLE, whichever is first	1	1-- ---
K---	Read forward and terminate after next KEY	0	1-- ---
	Word Disposition		
-A--	Store all words	-	--- 11-
-K--	Store key words	-	--- 10-
-D--	Store data words	-	--- 01-
-N--	Store no words	-	--- 00-
	Termination Action		
--I-	Interrupt the Central Processor when the SC order has terminated, and proceed to next order in Order Sequence. Tape will travel to BLE like normal RAN	-	-11 ---
--D-	Interrupt the CPU and disconnect the In-Out Processor	-	-10 ---
--P-	Go to BLE then proceed to next order in Order Sequence, unless program interrupt occurs then disconnect	-	-00 ---
--X-	Proceed immediately to next order in Order Sequence. Used when Reading Words of a block into various memory locations	-	-01 ---
	Optional Interrupt		
---F	Enable EOF interrupt	-	--- --1
	Examples: SCWAP, SCWAPF		

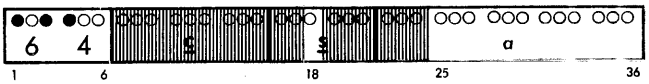
RR (c) a,c READ REVERSE ORDER

The RR order is identical to the RRV instruction except that the input-output device is specified by *s* of the previous SS instruction. See the RRV instruction for details.

DEVICE CONTROL ORDERS

The device control orders are magnetic tape instructions which initiate or control tape motion.

SK(s) a SEARCH KEY ORDER



An SK order initiates a search of a magnetic tape in the forward direction until a key corresponding to the *a* portion of the instruction word is detected, at which time control is transferred to the next order in the sequence — always an SC order. If a key is not detected, the tape is searched until the EOF or BLE specified by the order is detected. A No Key program interrupt will occur. Program control will be transferred to location 1, 2, 3 or 4 corresponding to the In-Out Processor which executed SK.

SK is an optional preparatory order for an SC order. Data to be read into memory is first located on the tape.

EXTENDED SK ORDER

SK must be written as an extended order by adding one of the following interrupt control suffixes to the basic mnemonic operation code:

Control Symbols	*Control*	*Control Bits*
		000
		s
SK-		Bit 18
B	Cause NKY interrupt if BLE is detected	0
F	Cause NKY interrupt if EOF is detected	1

Examples: SKB, SKF

SP(c) c SPACE ORDER

The SP order is identical to the SKP instruction except that the input-output device is specified by the previous SS instruction. See the SKP instruction for details.

BS(c) c BACKSPACE ORDER

The BS order is identical to the BSP instruction except that the input-output device is specified by the previous SS instruction. See the BSP instruction for details.

RW REWIND ORDER

The RW order is identical to the RWD instruction except that the input-output device is specified by the previous SS instruction. It also ends a sequence of input-output orders and disconnects the In-Out Processor.

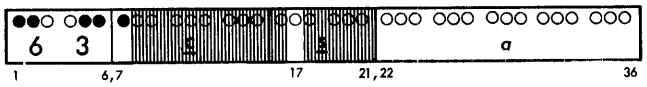
PROGRAM CONTROL ORDERS

The program control orders perform a variety of termination functions.

Program control orders are usually used to terminate a sequence of orders initiated by an SS instruction.

Program control orders all have the same octal operation code (63). They are distinguished from each other by code bits in the c and s portions, as shown under the descriptions of the individual orders.

PT(s,c) a I/O PROCESSOR UNCONDITIONAL TRANSFER ORDER



PT causes an unconditional transfer of In-Out Processor control to location *a* within the Order Sequence.

After the transfer the In-Out Processor continues to execute orders in sequence starting at memory location *a*.

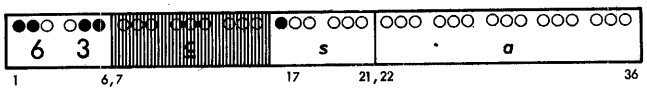
EXTENDED PT ORDER

PT may be written as an extended order by adding one of the following suffixes:

Control Symbols	*Control*	*Control Bits*
		000 000
		s
		Bit 16 21
PT-		
U	Cause an unconditional transfer of In-Out Processor control to location a within the order sequence	000 000
I	Cause an unconditional transfer of In-Out Processor control to location a within the order sequence and cause an End-of-Order Interrupt in the Central Processor	010 000

Examples: PTU, PTI

PS(s) a I/O PROCESSOR ERROR SENSE ORDER



PS conditionally transfers In-Out Processor control to the order sequence specified by *a*.

PS senses the In-Out Processor error controls specified by the control bits of this order. If the specified controls are SET, transfer occurs. If the specified error conditions do not exist the program continues in sequence. PS does not RESET the controls.

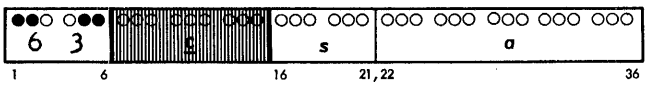
EXTENDED PS ORDER

PS may be written as an extended order by adding at least one of the following sense and transfer suffixes in the sequence shown. If the End-of-Order interrupt suffix is used only three of the sense and transfer suffixes may be used.

Control Symbols	*Control*	*Control Bits*
		000 000
		s
		Bit 16 21
PS----		
O	Transfer to location a if IMO is SET	1-1 ---
T	Transfer to location a if TRE is SET	1-- 1--
S	Transfer to location a if ISE is SET	1-- -1-
P	Transfer to location a if IPE is SET	1-- --1
A---	Transfer to a on any error	1-1 111

Examples: PSO, PSOP, PSOPI, PSOSPI, PSA interrupt

PR(s) a I/O PROCESSOR SENSE AND RESET ORDER

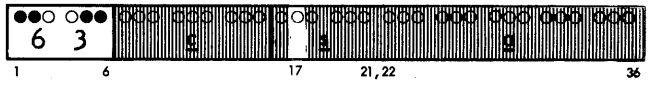


PR conditionally transfers In-Out Processor Control to the Order-Sequence order specified by *a*.

PR senses and RESETS the In-Out Processor error controls specified by the control bits of the order. If the specified controls are SET, transfer occurs. If the specified error conditions do not exist, the program continues in sequence.

PR is identical to PS including all control functions except that *bit position 16 always contains a zero,* whereas in PS, bit 16 always contains a one. For extended order options see the PS order.

ST (s) SEQUENCE TERMINATE ORDER



ST ends a sequence of input-output orders and disconnects the In-Out Processor.

EXTENDED ST ORDER

ST may be written as an extended order by adding one of the following control suffixes:

Control Symbols	*Control*	*Control Bits*
		000 000
		s
		Bit 16 21
ST-		
D	End sequence of Orders and disconnect	000 000
I	End sequence of Orders, cause an End-of-Order interrupt in the Central Processor and Disconnect	010 000

Examples: STD, STI

PROGRAM INTERRUPT

GENERAL DESCRIPTION

The Sylvania 9400 System has provision for interrupting the Central Processor Program Sequence when certain circumstances exist. All program interrupts are under programmer control except those which indicate an error condition such as "no key" being found within a block or file whichever is specified by the Read Key Instruction. Either type of interrupt is automatic and may occur at any time

during the execution of the main program. A special form of program interrupt, the Trapping Mode, is discussed under the CENTRAL PROCESSING UNIT.

A program interrupt can only occur when the Stop Program Interrupt Switch (SPI) is reset. As soon as the Central Processor program is interrupted, SPI is automatically set. The contents of the Program Counter (i.e., the address of the instruction which would normally be executed are placed in the B-register. An unconditional transfer instruction (TRU) is formed in the Instruction Register, and, according to where the program interrupt originates from (central processor or In-Out Processor 1, 2, 3, 4), an address is placed in the Program Counter. In this manner, control of the Central Processor is transferred to a specific memory location and the main program is interrupted. By immediately having SPI set, further interrupts are therefore prevented until SPI is reset at the end of the interrupt program. This is normally done by using a SNR instruction to transfer control back to the main program sequence.

PROGRAM INTERRUPT CONTROL SWITCHES

The switches, locations and characteristics associated with the various types of interrupts are shown in Tables IV-6 and IV-7. Each type of program interrupt is initiated by an activity switch. Activity Switches are automatically SET by the circumstances which cause the interrupt. Program interrupts are governed by either a *decision switch* which must be SET or by a bit being set in an instruction during the order sequence mode. This *decision* along with the appropriate activity switch will cause a particular type of program interupt. Thus, if the programmer wishes to permit a particular type of interrupt to occur, he SETS the appropriate decision switch (by a SNS instruction) at the beginning of his program. If during the execution of the program, the corresponding Activity Switch becomes SET, program interrupt will occur.

INTERRUPT PROGRAMS

A program interrupt transfers control to either location 0, 1, 2, 3 or 4 depending upon whether the interrupt originated from the central processor or Input Output Processor 1, 2, 3, 4. See Table IV-8. The programmer provides an unconditional transfer (TRU) instruction in locations 0, 1, 2, 3, and 4. This TRU will again transfer the program to another portion of memory where the various decision and activity switches or resultant switches are sensed by a SNR instruction. He will thus be able to set up his own priority within each group and quickly determine what conditions caused this interrupt. This condition being found will then transfer the program where a special interrupt program is stored. The interrupt program will normally resolve the circumstance which initiated the program interrupt. For example, if overflow or underflow caused interrupt, the interrupt program may perform special arithmetic operations to overcome the difficulty. Once the special interrupt program has been completed, control is normally returned to the main program sequence.

SIMULTANEOUS PROGRAM INTERRUPTS OR INTERRUPTS OCCURING WHEN SPI IS SET

A priority grouping scheme is established when there is more than one program interrupt occurring at the same instant or if there is more than one interrupt waiting to be processed. This priority scheme is not established according to the type of interrupt within each group because who is to say which interrupt is most important. This priority grouping scheme has been established according to the origin of the interrupt. Any central processor interrupt has the highest priority and the program transfers control to location whenever this type of interrupt exists. Interrupts occurring in the various In-Out Processors will be sampled in the order of the lowest number In-Out Processor first.

STOP PROGRAM INTERRUPT

In order to prevent a second program interrupt from occurring during the execution of an interrupt program, a Stop Program Interrupt Switch (SPI) is provided. Whenever program interrupt occurs, SPI is automatically SET. As long as SPI is SET, no further interrupts can occur. Thus, if the programmer wishes to allow additional interruption of the main program to occur, he must RESET SPI at the end of the interrupt program. This is normally done by using a SNR instruction to transfer control back to the main program sequence.

ACTIVITY AND RESULTANT SWITCH CONTROL

In addition to RESETTING SPI before returning to the main program, the programmer must RESET the activity or resultant switch which originally caused the interrupt. This is normally done early in the interrupt program. If the activity or the resultant switch is not RESET when control is transferred to the main program, program interrupt will immediately occur again and the Central Processor will "trap" to the interrupt program it just finished executing.

TYPES OF PROGRAM INTERRUPT

Following is a description of the various types of program interrupt (see Table IV-7).

1. Transfer Order Trapped — If the *Trapping Mode* decision switch (TRA) is SET and the Central

Processor decodes a transfer instruction [TS + (TU) $(m_{21})'$], the Transfer Order Trapped (TOT) switch is automatically SET. When TOT is set, Central Processor control is transferred to location 000. TOT must be RESET by a SNR instruction.

2. Real-Time Input Control Character Interrupt — When a special control character (see FUNCTIONAL CONTROL CHARACTERS) is read in to 9400 Systems with a Real-Time Channel, the Real-Time Input Control Character Interrupt activity switch (RINC) is SET automatically. If the Real-Time Channel decision switch (RCI) is also SET, the Central Processor program is interrupted and control is transferred to the central processor memory location 0. RINC must be RESET by a SNR instruction.

3. Overflow or Underflow — If the Alarm Program Interrupt decision switch (API) is SET and either the Overflow Alarm (OA) or the Underflow Alarm (UA) is SET, the program is interrupted and Central Processor control is transferred to memory location 000. OA an UA are considered as activity switches, and must be RESET by an SNR instruction. For overflow and underflow control, see Section III, CENTRAL PROCESSING UNIT.

4. Processor Busy — If all Input Output processors become busy or if the selected processor is busy the Processor Busy Activity switch (PRBA) is set automatically. If the Processor Busy decision switch (PRBI) is also set, and SPI is RESET the Central Processor program is interrupted and control is transferred to memory location "0". PRBA must be RESET by a SNR instruction if a Processor Busy Program Interrupt does occur. If SPI is set or if the decision switch is not set and the activity switch becomes SET, computer operation will "hang up" in Timing Function 8 until one of the processors are free in the regular order mode or until the selected processor is free in order sequence. No program interrupt will occur under these conditions. PRBA is automatically RESET when the processor or the selected processor becomes free.

5. Device Busy — If the device addressed is busy the Device Busy Activity Switch (DVBA) is set automatically. If the Device Busy decision switch (DVBI) is also set, the Central Processor program is interrupted and control is transferred to memory location "0". DVBA must be RESET by a SNR instruction if a Device Busy Program Interrupt occurs. If SPI is already set or if the decision switch is not set and the activity switch becomes set the actions described for Processor Busy occur.

6. End-of-Tape — If an End-of-Tape marker is encountered, the End-of-Tape activity switch (ETK) in the Processor to which the magnetic tape unit is connected is set. At the end of Write instruction the corresponding activity switch (ETA_n) in the Central Processor is set. When the End-of-Tape decision switch (ETI) is also SET, the program interrupt transfers Central Processor control to memory locations 001 through 004 depending upon which Processor received the End-of-Tape signal. A maximum of 4500 words may be written on the Magnetic Tape from the time the ETK switch is SET.

During a Read instruction a program interrupt will occur where ETI is set and the BLE after the End-of-Tape Warning is read.

7. Functional Control Character Interrupt—When a special control character (see FUNCTIONAL CONTROL CHARACTERS) is read in during an input operation, Functional Control Character activity switches (CFK) (WCK) is set in the Processor through which the control character is detected. This in turn sets the corresponding activity switch in the Central Processor (FCC_n). Coincidence of the Functional Control Character decision switch (FCI) and an FCC switch in the set condition causes the Central Processor program to be interrupted and control to be transferred to memory location 001 through 004 depending upon which Processor is connected to the Input device. The FCC switches must be reset by a SNR instruction.

8. End-of-File — If an End-of-File mark is encountered on magnetic tape during a read operation, the End-of-File activity switch (EFK_n) in the Processor to which the magnetic tape unit is attached is SET. This in turn sets the corresponding Activity Switch in the Central Processor (EOF_n). Coincidence of an EOF switch and the End-of-File decision switch (EFI) in the SET position causes program interrupt and Central Processor control is transferred to memory location 001 thru 004, depending upon which Processor received the End-of-File signal. The EOF switches must be RESET by a SNR instruction.

When an EOF is detected during a Scatter Read order the EFK switch in the Processor is SET and the order is terminated. This in "turn" sets the EOF switch in the Central Processor as described above. Coincidence of the EFK switch in the processor and the End-of-File decision control bit 21 being set will set the resultant processor switch EPI. The corresponding resultant switch in the Central Processor (EPR_n) is set when EPI is SET. This switch being set will interrupt the Central Processor. The Central Processor program control will be transferred to location 001 — 004 depending upon which Processor received the End-of-File signal. The EPI switches in the processor are automatically RESET following interrupt but the EPR switches must be RESET by a SNR instruction.

9. If an Order Sequence Program specified End-of-Order Interrupt the End-of-Order Resultant Switch (EOI) in the Processor which is executing the Order Sequence will automatically be SET when the order is completed. An End-of-Order Interrupt is initiated by placing a one in bit position 17 of a Scatter Read, Scatter Write or End Sequence Order. The corresponding resultant switch (EOR) in the Central Processor program is set when EOI is SET. The Central Processor program is interrupted and control is transferred to memory locations 001 through 004 depending upon which Processor initiated the interrupt. The EOI switches in the Processor are automatically RESET following interrupt but the EOR switches must be reset by a SNR instruction.

10. No-Key Interrupt — If, during the execution of the Order Sequence order Search Key (SK--), the key word being sought is not found either within a block or within a file depending upon whether bit 18 is not set to 1 or set to 1, the No-Key resultant switch (NKY_n) in the appropriate processor is SET. This in turn sets the No-Key Resultant in the Central Processor (NKR_n). The Central Processor program is always interrupted and control is transferred to memory locations 001 through 004 depending upon which Processor is involved. The NKY Switches in the Processor are automatically RESET following interrupt but the NKR switches must be RESET by a SNR.

RESETTING ACTIVITY SWITCHES

With the exception of the Trapping Mode Activity Switch (TOT), all program interrupt activity switches may be SET automatically regardless of the status of their decision switches. For example, in a program which is not concerned with program interrupt, a read operation may encounter an End-of-File mark on magnetic tape and cause an EOF switch to be SET. Although the switch will not cause interrupt at that time, the programmer may, later in the program, SET the EFI switch in preparation for future End-of-File conditions. If he does, without RESETTING EOF first, the main program will be interrupted immediately and the interrupt program will be executed at the wrong time. Thus, it is important that the programmer provide for the RESETTING of activity switches whenever any portion of his program or system of programs is to make use of the program interrupt feature.

FUNCTIONAL CONTROL CHARACTERS

The 9400 System is capable of generating special Functional Control Characters for output. It is also capable of accepting such characters as input. The primary use of the control characters is to "flag" incoming data. If the programmer wishes to carry out certain operations upon the receipt of specific pieces of incoming data, he may cause the data to be read to contain a control character or combination of control characters. Thus, when the control character is detected by the Input-Output System, program interrupt may be permitted (see PROGRAM INTERRUPT), causing automatic transfer of control to a special interrupt program.

A control character in the Central Processor is indistinguishable from a normal 9400 alphanumeric character. A control character gains its identity in the In-Out Processor during output. As the six-bit character leaves the Processor Buffer Register (BFR), it has attached to it a parity bit and a control bit. Normally, when the control bit is a ONE, the character is normal; when it is a ZERO, the 8-bit configuration becomes a control character. The status of the control character bit depends upon the mode in which the computer is operating.

To cause control characters to be output, the programmer must SET the Write Control Character switch (WCC) prior to initiating an output operation. WCC is sensable and may be RESET by Sense Instructions. Once WCC has been SET, the computer operates in the Control Character Mode and all characters output by following output instructions will be written as control characters.

For 9400 Systems with a Real-Time Channel, an equivalent control character switch, Real-Time Output Control Character (ROTC), must be SET before providing characters for output through the Real-Time Channel. ROTC, like WCC, may be SET and RESET by Sense Instructions.

During input operations, incoming data containing control characters may cause program interrupt. If the decision switches RCI and FCC are SET, detection of an incoming control character either in the Real-Time Channel or in any Processor will cause program interrupt and transfer of control to the appropriate memory location. In this manner, the programmer may use control characters to call the attention of his program to the arrival of important data.

Order	ISN Flip Flop	# of Characters Per Word	Word Structure	Bit 7 of RAN	Amount of Input or Output	Other Comments
WAN	1	7	A six bit character sign followed by six 6-bit characters of word with bits 31-36 recorded first.	Affects value of c only	Up to 511 words inclusive.	Writes initial block marks before beginning. Also terminating block marks when finished. If WEF flip flop is set, end of file marks replace usual end of block marks. Is inconsistent if zero words are specified.
	0	6	Six 6-bit characters recorded as above but ignoring sign.			
RAN	1	7	Same as WAN.	1	Up to 255 blocks inclusive.	Searches for beginning of block mark before reading. Coasts to end of block if k words < number of words in the block. If k words > number of words in block, end of block marks are ignored. If any end of file marks are interpreted, reading will cease immediately whether or not k words or blocks have been read.
				0	Up to 255 words inclusive.	
	0	6	Same as WAN	1	Up to 255 blocks inclusive.	
				0	Up to 255 words inclusive.	
RRV	1	7	Same as RAN except tape head moves in a backward direction. The characters are read into memory as if reading forward.	1	Same as RAN	Same as RAN except keep in mind tape moves in reverse direction.
				0	Same as RAN	
	0	6		1	Same as RAN	
				0	Same as RAN	
WWA	1	7	Same as WAN	Affects value of c only	Up to 511 words inclusive; however must be a total number of characters equal to the number of characters in the original block within one word length.	Used to replace a block of information on a tape. When executed, it searches for beginning block marks rather than drop the block marks as in the WAN order.
	0	6				
SKP	No effect	Non-transmitting	Non-transmitting	1	Skips up to 255 files inclusive.	One block is counted skipped for each set of end block marks or end file marks. One file is counted skipped for each set of end of file marks (end block marks in the latter case are ignored).
				0	Skips up to 255 blocks inclusive.	
BSP	No effect	Non-transmitting	Non-transmitting	1	Backspaces up to 255 files inclusive.	One block is counted backspaced for each set of start block marks. One file is counted backspaced for each set of end of file marks; the tape comes to rest at the beginning of the block containing the end of file marks.
				0	Skips up to 255 blocks inclusive.	
RWD	No effect	Non-transmitting	Non-transmitting	No effect	Rewinds tape unit j.	

Table IV-1. Operational Table for Magnetic Tape

Order	ISN Flip-Flop	No. of Characters Per Word	Word Structure	Bit 7 of RAN	Amount of Input or Output	Other Comments
WAN	1	7	Same as WAN on MAGNETIC TAPE	Affects value of c only	Up to 511 words inclusive.	Is inconsistent if zero words are specified.
	0	6				
RAN	1	7	Same as WAN on MAGNETIC TAPE	Affects value of c only	Up to 511 words inclusive or until STOP CODE is encountered.	If STOP CODE occurs in the middle of a word, there is an I-O alarm. It must occur after a full word, and it is not read into memory.
	0	6	Same as WAN on MAGNETIC TAPE	1	Until STOP CODE is encountered.	The STOP CODE may occur after any number of characters. If necessary, the low order bits of the last word will be filled with zeros and then the order will be terminated. The STOP CODE is read into memory as is any other character.
	0	1	One 6-bit character in low order position. Rest of word is 0.	0	Up to 255 characters.	
WOK	0 or 1	13	Sign digit followed by 12 octal digits. Each octal digit is converted to equivalent 6-bit configuration on tape.	Affects value of c only	Up to 511 words inclusive.	Is inconsistent if zero words are specified. No need for ISNff - The sign is automatically interpreted.
ROK	0 or 1	13	Each 6-bit configuration is converted to the sign digit followed by 12 octal digits to fill a word.	Affects value of c only	Up to 511 words inclusive, or until a STOP CODE is encountered.	Comments for WOK also apply here. Also will fail if a character other than "0" through "7" appears in the middle of a word. When such non-legimate characters appear between words they are called "comment characters" and are ignored.

Table IV-2. Operational Table for Paper Tape

Order	ISN Flip-Flop	No. Characters Per Word	Word Structure	Bit	Amount of Input or Output	Other Comments
WAN	1	7	2 words of memory per 1 line on the card. Sign of first word in positions 1 and 2 of card; magnitude of first word in positions 3-38 of card. Sign of second word in positions 39-44 of card; magnitude of second word in positions 45-80 of card.	Affects value of c only	Up to 511 words inclusive	Each card can contain 12 rows, 2 words each; ∴ 24 words. If the number of words to be punched does not constitute an integral number of cards, the last card will be punched with whatever data is available for it.
	0	6	2 1/3 words of memory per 1 line on the card. The first character of each row is stripped and placed in positions 1, 2.	Affects value of c only	Up to 511 words inclusive	Same as above except each card can contain 12 rows, 2 1/3 words each; ∴ 28 words.
RAN	1	7	Same as WAN above	A one is always forced into Bit 7	Up to 255 cards inclusive	Column 1 on card must = 0; columns 39-43 must = 11000; otherwise, there will be an error alarm.
	0	6	2 1/3 words of memory per 1 line on the card. The digits 1100 are added to the first character (positions 1, 2) of each row.	A one is always forced into Bit 7	Up to 255 cards inclusive	"Padding" is necessary in order to use this mode effectively. All blank rows are read as zeros into memory.

Table IV-3. Operational Table for Card Reader — Punch

Order	ISN Flip-Flop	No. Characters Per Word	Word Structure	Bit 7 of WAN or WOK	Amount of Output	Other Comments
WAN	1	7	Same as WAN on the Flexowriter	Affects value of c only	Up to 511 words inclusive, up to 120 characters inclusive per line.	In "automatic line feed" mode, there are one or two line feeds (specified beforehand) after each line printed.
	0	7	Same as WAN on the Flexowriter	Affects value of c only	Up to 511 words inclusive, up to 120 characters inclusive per line.	In "programmed line feed" mode, the first 6-bit character on any given line is not printed but specifies how many lines to be fed before printing the line. In the "verbatim printout mode", those characters which control the format of the printing are printed as unique symbols and their effect does not take place.
WOK	1 or 0	13	Same as WOK on the Flexowriter	Affects value of c only	Up to 511 words inclusive, up to 120 characters inclusive per line.	Can be used effectively in the "automatic line feed" mode only. Since a carriage return character is non-octal, it cannot limit the 120 characters per line maximum.

Table IV-4. Operational Table for Line Printer

Order	ISN Flip-Flop	No. of Characters Per Word	Word Structure	Bit 7 of WAN or WOK	Amount of Output	Other Comments
WAN	1	7	A sign character (printed as a zero or one) followed by six 6-bit characters of word (printed as alphanumeric characters) with bits 31-36 recorded first.	Affects value of k only	Up to 511 words inclusive.	Intermingled with the data to be printed must be those characters which control the format of printing (space, upper case, lower case, etc) in their desired positions. Every 127 characters (or less) there must be a carriage return character so that characters will not be superseded upon each other in the last print position.
	0	6	Same as above ignoring sign.	Affects value of c only	Up to 511 words inclusive.	Is inconsistent if zero words are specified.
WOK	1 or 0	13	A sign character (printed as a zero or one) followed by 12 3-bit characters of word (printed as octal characters).	Affects value of c only	Theoretically, up to 511 words inclusive. Practically, up to 9 words inclusive.	When using WOK, there is no way to specify a 6 bit carriage return character. Therefore if k were > 9, 9 x 13 = 117. The remaining k words will be superimposed on the last print position on the paper. This restriction may be overcome by combining use of WAN and WOK instructions.

Table IV-5. Operational Table for Flexowriter

m (Octal)	Code	Description	Can be Set by Program	Can be Reset by Program	Console Switchable Momentary	Console Switchable Lock
0001-0067	$\underline{IOD}_{1-4}$	In-Out Device	-	-	-	-
0070	$DVBI^{1-4}$	Device Busy Interrupt	x	x	-	-
0071	$DVBA^{1-4}$	Device Busy Alarm	x	x	-	-
0075-0072	$\underline{EOR}^{1-4}$	End Order Resultant	x	x	-	-
0076	PRBI	Processor Busy Interrupt	x	x	-	-
0077	PRBA	Processor Busy Alarm	x	x	-	-
0100	OA	Overflow Alarm	x	x	-	-
0101	ROP I	Real Time Output Program Interrupt	x	x	-	-
0102	ISN	Interpret Sign	x	x	-	x
0103	NHP	No Halt Control on In-Out Processor Error	x	x	-	x
0104	RPE	Real Time Parity Error	-	x	-	-
0105	ROBB	Real Time Output Busy Bit Signal to Kineplex	x	-	-	-
0106	ROR_{38}	Real Time Output Reg. Bit No. 38	x	x	-	-
0107			-	-	-	-
0110-0117	SFF_{1-8}	Sense Switch 1-8	x	x	-	x
0120-0127	SFF_{9-16}	Sense Switch 9-16	x	x	-	x
0130-0133	IOA^{1-4}	In-Out Alarm FF in I/O Processor 1-4	-	x	-	-
0134	UA	Underflow Alarm	x	x	-	-
0135	TPE	Tape Erase	x	x	-	x
0136	API	Alarm Program Interrupt	x	x	-	x
0137			-	-	-	-

Table IV-6. Program Interrupt Control Switches

m (Octal)	Code	Description	Can be Set by Program	Can be Reset by Program	Console Switchable Momentary	Console Switchable Lock
0140-0143	EPR$^{1-4}$	Enable Program Resultant	x	x	-	-
0144-0147	NKR$^{1-4}$	No Key Resultant	x	x	-	-
0150	FCI	Functional Control Character Interrupt	x	x	-	x
0151-0154	FCC$^{1-4}$	Read Functional Control Character IOP$^{1-4}$	x	x	-	-
0155	ROSN	Real Time Output Interpret Sign	x	x	-	-
0156	RISN	Real Time Input Interpret Sign	x	x	-	-
0157	TOT	Transfer Order Trapped	x	x	-	x
0160	SPI	Stop Program Interrupt	x	x	x	-
0161	ETI	End of Tape Interrupt (Allow End of Tape Signal to Interrupt)	x	x	-	-
0162	EFI	End of File Interrupt (Allow End of File Signal to Interrupt)	x	x	-	-
0163	RCI	Real Time Control Character Interrupt	x	x	-	-
0164	RINC	Real Time Control Character Input	-	x	-	-
0165	ROTC	Real Time Control Character Output	x	x	-	-
0166	WEF	Write End of File	x	x	-	-
0167	WCC	Write Control Character	x	x	-	-
0170-0173	ETA$^{1-4}$	End of Tape Alarm IOP$^{1-4}$	x	x	-	-
0174-0177	EOF$^{1-4}$	End of File Alarm IOP$^{1-4}$	x	x	-	-

Table IV-6. (Cont'd.) Program Interrupt Control Switches

Central Processor		In-Out Processor			Program Interrupt Conditions
Decision	Activity	Decision	Activity	Resultant	(SPI must be cleared)
TRA	TS + (TU) $(S_{21})'$	---	---	---	TOT
RCI	RINC	---	---	---	(RCI) (RINC)
API	OA + UA	---	---	---	API (OA + UA)
PRBI	PRBA set if all Processors are busy or if selected Processor is busy		All Processors Busy or if Selected processor is busy		(PRBI) (PRBA)
DVBI	DVBA				(DVBI) (DBVA)
ETI	ETA set by ETK	---	ETK	---	(ETI) (ETA)
FCI	FCC set by CFK & WCK	---	(CFK) (WCK)	---	(FCI) (FCC)
EFI	EOF set by EFK	(S_{21}) (SR)	EFK	EPI	(EFI) (EOF) + EPR set by EPI
---	---	(S_{17}) (Read or Write or ES)	END	EOI	EOR set by EOI
---	---	Search Key Order	Key not detected within a file or block depending upon bit 18		NKR set by NKY

Table IV-7. Program Interrupt Conditions

Central Processor Interrupts ----	Location 0 (Group 0)
Transfer Order Trapped -	TOT
Overflow or Underflow	API (OA + UA)
Real Time Input Control Character	RCI (RINC)
Processor Busy	PRBI (PRBA)
Device Busy	DVBI (DVBA)
Input Output Processor i Interrupt --	**Location i (i = 1, 2, 3, 4)**
Functional Control Characters	FCI $(FCC)^i$
End of Tape	ETI $(ETA)^i$
End of File	EFI $(EOF)^i$
End of File in Order Sequence	EPR^i
No Key in Order Sequency	NKR^i
End Order Interrupt in Order Sequence	EOR^i

Table IV-8. List of Program Interrupt Locations

SECTION V
INPUT-OUTPUT DEVICES

INTRODUCTION

The 9400 System may have up to 64 input-output devices attached to it. Each input-output device may be connected to any In-Out Processor, of which there may be a maximum of four. As many input-output devices may operate simultaneously as there are In-Out Processors, and in parallel with the Central Processor program.

Following is a generalized description of the major input-output devices included in the 9400 System repertoire. For details concerning a particular device, refer to the appropriate INPUT-OUTPUT EQUIPMENT MANUAL.

MAGNETIC TAPE UNIT

A magnetic tape unit consists basically of a *tape handler* which supports and moves a band of *magnetic tape* past a set of *read and write heads.* The magnetic tape is contained on reels of up to 3600 feet in length and is normally one inch in width. Information is recorded on magnetic tape in the form of magnetized "spots", each spot representing a single bit. See figure V-1. A single character, consisting of six data bits, a control bit and a parity bit, is written *across* the tape, as shown. Thus, a computer word on magnetic tape is made up of six (non-interpret sign) or seven (interpret sign) 8-bit characters written sequentially along the tape. Information is always recorded on magnetic tape in an integral number of words; it is not possible to write (or read) part of a word. Parity is computed when the tape is written and checked when the tape is read. A parity error during a read operation will SET the In-Out Parity Error Switch (IPE) and cause the In-Out Alarm (IOA) to be SET.

On magnetic tape, words are grouped into *blocks.* A block may consist of any number of words (when written in the Order Sequence Mode). Each block is preceded by a special Block-Start Marker (BLS) and followed by a Block-End Marker (BLE). In turn, any number of blocks of information may be grouped into a file. An End-of-File Marker (EOF) is written at the end of the last block in the file. End-of-File detection may be used to cause program interrupt (see under PROGRAM INTERRUPT, INPUT-OUTPUT SYSTEM).

In Single Instruction Mode write operations, BLS and BLE markers are written automatically, since only one block may be written for each instruction. In the Order Sequence Mode, the programmer has the option of writing BLE markers, so that he may write a block of any length. An EOF marker is written by SETTING the Write End-of-File switch (WEF prior to the write operation. In Single Instruction mode, WEF is SET by a SNS instruction; in Order Sequence Mode, it is SET by employing the appropriate suffix with the Order Sequence Order. In either case, it is RESET automatically following the operation. Detection of an EOF marker during a read operation automatically terminates the operation.

Each reel of magnetic tape has a special End-of-Tape marker (EOT) approximately three feet from the physical end of the tape. When an EOT marker is detected, the End-of-Tape Area activity switch (ETA_n) in the Processor to which the tape unit is connected, is SET. All Processor operations halt upon completion of the block currently being processed. If ETI is SET, program interrupt will occur. The only exception is during a *skip* operation, which will terminate upon detection of an EOF marker.

Following is a list of instructions and orders which may be used with magnetic tape; for specific details, see under the appropriate instruction or order:

Single Instruction Mode: WAN, WWA, RAN, RRV, SKP, BSP and RWD; Order Sequence Mode: GW, WW, WK, SC, RR, SK, SP, BS, RW and ES orders.

PAPER TAPE EQUIPMENT

The 9400 paper tape equipment includes a photoelectric tape reader and a Teletype punch. Paper tape consists of a narrow strip of paper containing rows of up to eight holes punched laterally across it (see figure V-2). The format is roughly equivalent to that for magnetic tape, in that each lateral row represents one 6-bit character. The two extra positions contain the control bit and parity bit, respectively. Parity is computed and punched automatically upon output and checked during input. In the event of a parity error the IPE activity switch is SET, and an In-Out Alarm (IOA) will result. A hole in the tape represents a binary ONE; the absence of a hole indicates a binary ZERO. The 9400 is also equipped to accept 5-hole paper tape, but in the non-interpret sign mode only.

It is possible to write out and read in an integral number of words on paper tape, as it is with magnetic tape. By using an alphanumeric read operation in

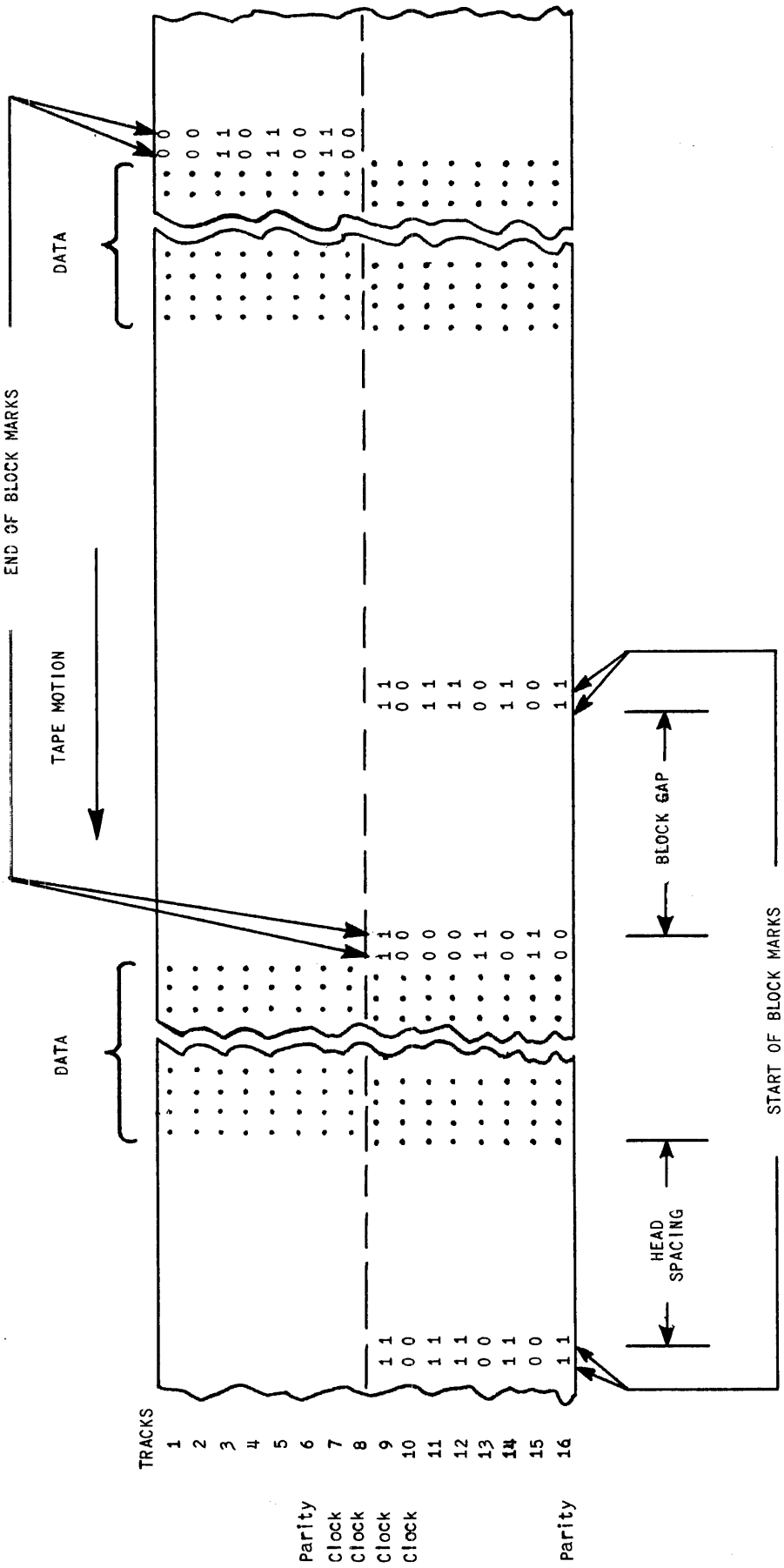


Figure V-1. Magnetic Tape Format

non-interpret sign mode, information may be read as individual characters. Paper tape contains no provision for BLS and BLE markers. A *stop code* (STP) may be used to terminate paper tape operations. Whenever a STP code is encountered in a read operation, the operation is terminated and the End-of-File activity switch (EOF_n) in the Processor to which the paper tape device is connected, is SET. If EFI is also SET, program interrupt will also occur.

Octal input-output operations are permitted with the paper tape equipment. Octal input and output is always automatically carried out in the interpret sign mode. Each word in Memory is output in the octal mode as thirteen characters, including sign. Between octal words on paper tape, the programmer may insert non-octal (i.e. alphanumeric) comments, which are ignored when the tape is read in octal mode.

When a *control character* is detected during an alphanumeric read operation, the operation is immediately terminated.

Following is a list of instructions and orders which may be used with paper tape equipment; for specific details, see under the appropriate instruction or order:

Single Instruction Mode: WAN, RAN, WOK, ROK.

ELECTRIC TYPEWRITER

An electric typewriter is used with the 9400 System as an *output* device for producing hard copy on line. It also includes a paper tape punch and paper tape reader for generating and verifying paper tapes for input to the 9400 System through the photoelectric paper tape reader.

The electric typewriter accepts both alphanumeric and octal outputs. In addition to printing normal alphanumeric characters, it responds to *function* characters,* such as *carriage return, upper case, lower case,* and *tab*. See APPENDIX G, 9400 ALPHANUMERIC CODES. By proper use of the function characters together with the printable characters, the programmer may produce any desired output format.

Alphanumeric output to the electric typewriter may be in either interpret sign or non-interpret sign modes. Octal output is automatically in interpret sign mode, and thirteen characters are output for each computer word. No function characters may be output in the octal mode.

The following instructions and orders may be used with the electric typewriter; for specific details, see under the appropriate instruction or order:

Single Instruction Mode: WAN, WOK: Order sequence Mode: WA, WO, and the ES.

*Not to be confused with functional control characters of magnetic tape unit. They are handled the same as any other characters when reading but are used to control the electric typewriter or line printer operation when writing.

HIGH-SPEED LINE PRINTER

The high-speed line printer is an output device which prints information one line at a time, up to 120 characters per line, at a rate of 900 lines per minute. The output produced is hard copy on multiple-copy fan-fold paper. The format and mode of operation is controlled by plugboard, switching, and program control.

The 120 characters per line may be printed in any configuration spread over 132 character positions.

The line printer operates in three modes, according to a mode switch setting: Automatic Line Feed, Programmed Line Feed, and Verbatim Print-Out. In Automatic Line Feed, the printer automatically single or double-spaces following each line, according to the setting of a single-double space switch. In Programmed Line Feed, the number of lines to be spaced is determined by the binary value of the first character sent to the printer at the beginning of each line. In Verbatim Print-Out, the printer accepts and prints every character sent to it, regardless of whether or not the character normally has a control function.

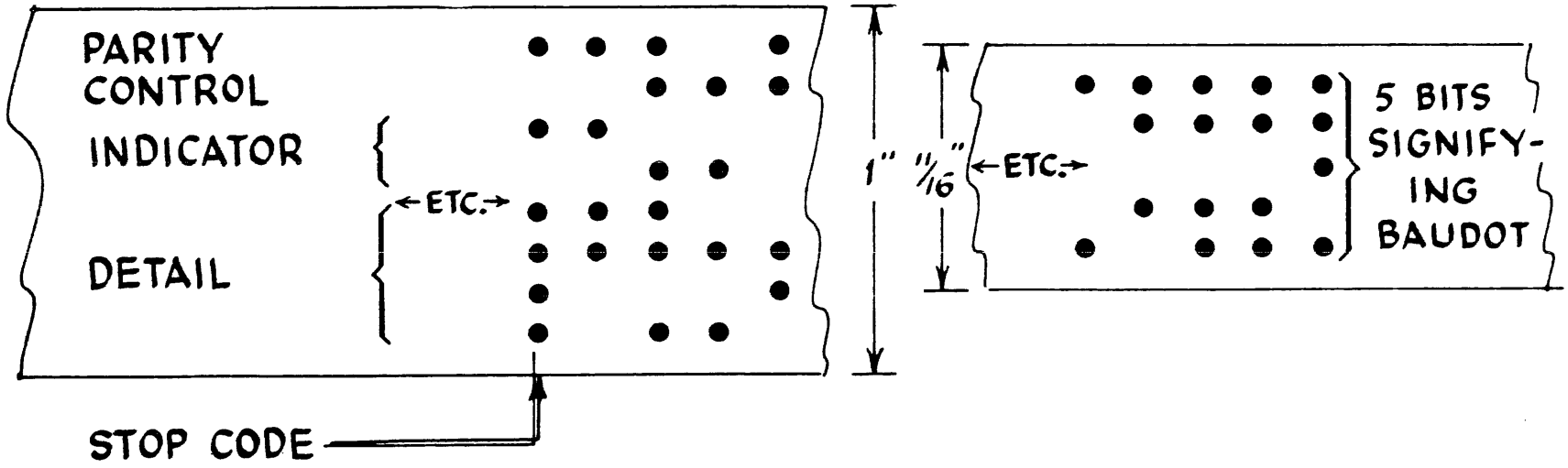
In modes other than Verbatim, the printer interprets the characters sent to it as one of two types: *function characters,* such as carriage return, tab, and stop code; and *printable characters,* such as those normally printed on the electric typewriter.

For alphanumeric output operations, the line printer may be used in either interpret sign or non-interpret sign mode. For octal output, interpret sign mode is automatically selected. Since function characters cannot be output in octal mode, octal output should normally be used for octal-type dumps in Automatic Line Feed Mode.

The line printer contains a buffer memory capable of storing 120 alphanumeric characters. Information sent to the printer is accumulated in the buffer until it is full, or until a function character terminates a line. As soon as a line is printed, and if the printer is still connected, the buffer accepts new data and prints the next line. It is possible to shorten a printed line through programming except in Verbatim Mode, as above, by including in the outgoing data the alphanumeric character for *carriage return* or *stop code* (see APPENDIX G, 9400 ALPHANUMERIC CHARACTER CODES). A carriage return automtically causes a line to be printed, regardless of its length. A stop code produces the same effect as a carriage return, excepting that after printing the current line the printer automatically feeds to the top of the next page and stops. Following a stop code, the printer halts and must be started by depressing the restart button on the buffer. The stop code is useful for calling the

PAPER TAPE, 8 CHANNEL AND 5 CHANNEL

DIFFERS FROM MAGNETIC TAPE BY CARRYING INFORMATION AS CHARACTERS ONLY. HAS NO BLOCKS, FILES OR ASSOCIATED GAPS. STOP CODE IS USED TO DIVIDE CHARACTERS INTO LARGER UNITS.



8 CHANNEL

ALPHANUMERIC CODED

PAPER TAPE

5 CHANNEL

BAUDOT* CODED

PAPER TAPE

*TELETYPE CODE

Figure V-2. Paper Tape Formats

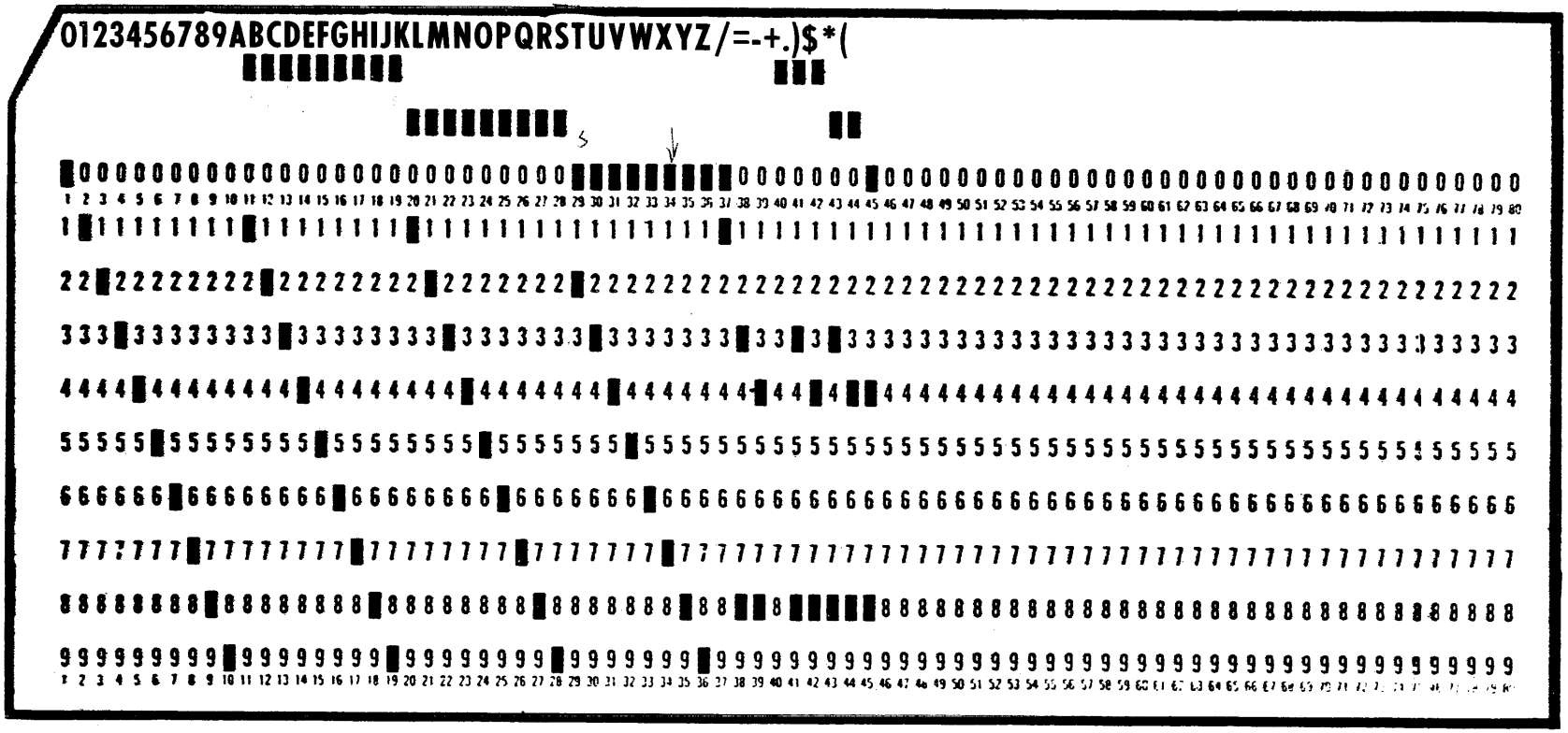


Figure V-3. Punch Card (Hollerith Code)

attention of the operator to the printer for special instructions.

External format control is provided through a *juxtaposition plugboard.* The juxtaposition plugboard is the link between the output of the buffer memory and the print hammers in the printer itself. The plugboard contains 120 patch cords which enable the operator to invert, space or reorder, in any fashion, the order in which the characters are printed for each line.

An additional plugboard, containing sixteen numbered plugs, allows the program to initiate *tabulate* operations. By selecting numbered plugs and arranging them in ascending order from left to right in the plugboard, the operator sets up a tab system that is directly analogous to that of a typewriter. If the data sent to the printer contains a tab function character, the printer automatically tabulates to the position in the printed line indicated by the numbered plug. A later tab function character in the same line will automatically cause the printer to tabulate to the next numbered location as specified by the tab plug.

The sixteen plug in the tab board is an end-of-line indicator. Normally, this space contains the plug numbered 121, i.e., one greater than the longest possible line. If the operator wishes to shorten the line, he may replace the last plug with a lower-numbered one, for example, 73. When this is done, the effective length of the printed line is shortened to 72 characters. Thus, if 72 characters are received by the printer buffer, or if a sixteenth tab function character is received, the line is automatically terminated and printed out.

The following instructions and orders may be used with the line printer; for details concerning them, see under the appropriate instruction or order:
Single Instruction Mode: WAN, WOK.

PUNCH CARD EQUIPMENT

Punch Card Equipment for the 9400 System includes an electromechanical card reader and punch. It accepts standard, 80-column punch cards, as shown in Figure V-3. The cards are read and punched 12-edge (upper long edge) first. They are processed in a card "cycle" consisting of twelve increments, one for each row on the card. In order to convert input and output information between card format and 9400 character format, a card buffer is provided between the card equipment and the 9400 System proper.

In the alphaneumeric read mode, each card is processed as a unit, e.g., the contents of an integral number of cards is read into memory. When the interpret sign mode is used, twenty-four words are read per card, two words per row, from the first 74 punch positions of each row. In this mode, punch positions 75-80 are not used. The card "image" produced in Memory consists of twenty-four sequential words, containing the information on the card as follows: Row 12 left (first word), and 12 right (second word), Row 11 left (third word), Row 11 right (fourth word), and so on. Where no information is punched in the card in columns 1 through 74, words are read into memory as positive ZEROS.

When the non-interpret sign mode is used, every punch position on the card is interpreted. The card is read as if four columns of information, containing $(1100)_2$, preceded the existing information in each row. The rows on the card, each row considered to consist of 84 punch positions, are read into 28 sequential locations in memory. This mode was designed specifically to read Hollerith characters from cards into a card image in memory. A conversion-from-Hollerith subroutine is then necessary to arrange the data into the desired program imput.

In the alphaneumeric write mode, information is processed by words. When the interpret sign mode is used, the number of words specified by the output operation are punched sequentially in cards in the same sequence as they are read, i.e., the first word in memory is punched in row 12 left, the second row in row 12 right, the third row in row 11 left, and so on. If the word count is larger than 24, the remaining words are punched in a second card. When the total word count is not divisible by 24, the remaining word spaces on the last card punched are left blank.

When the non-interpret sign mode is used in alphanumeric write operations, 2⅓ words are punched in each of the 12 rows on the card. The first four bits of the first character to appear in any row are ignored, and the remaining two bits of that character are placed in columns 1 and 2. This mode was designed specifically to write Hollerith characters on cards from a card image in memory. A conversion-to-Hollerith subroutine is first necessary to convert the desired data into a card image of Hollerith characters.

Following is a list of instructions and operations which may be used with punched card equipment; for specific details, see under the appropriate instruction or order. There is no provision for reading or punching cards in the octal mode.

Single Instruction Mode: WAN, RAN; Order Sequence Mode: WA, RA, and the ES orders.

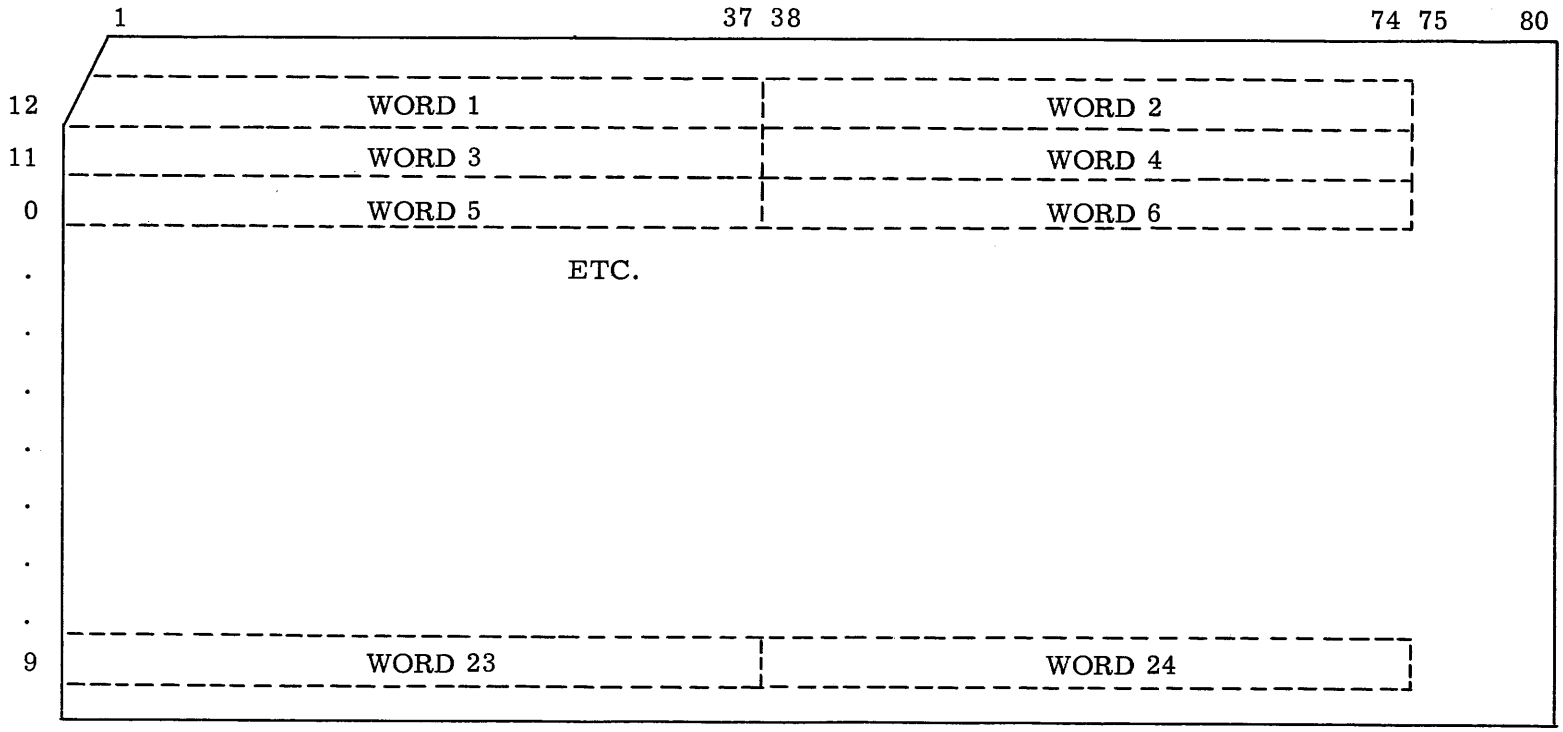


Figure V-4. Format of Data on ISN Card. This Data, Words 1-24, Will Appear in Successive Locations When Read into Memory.

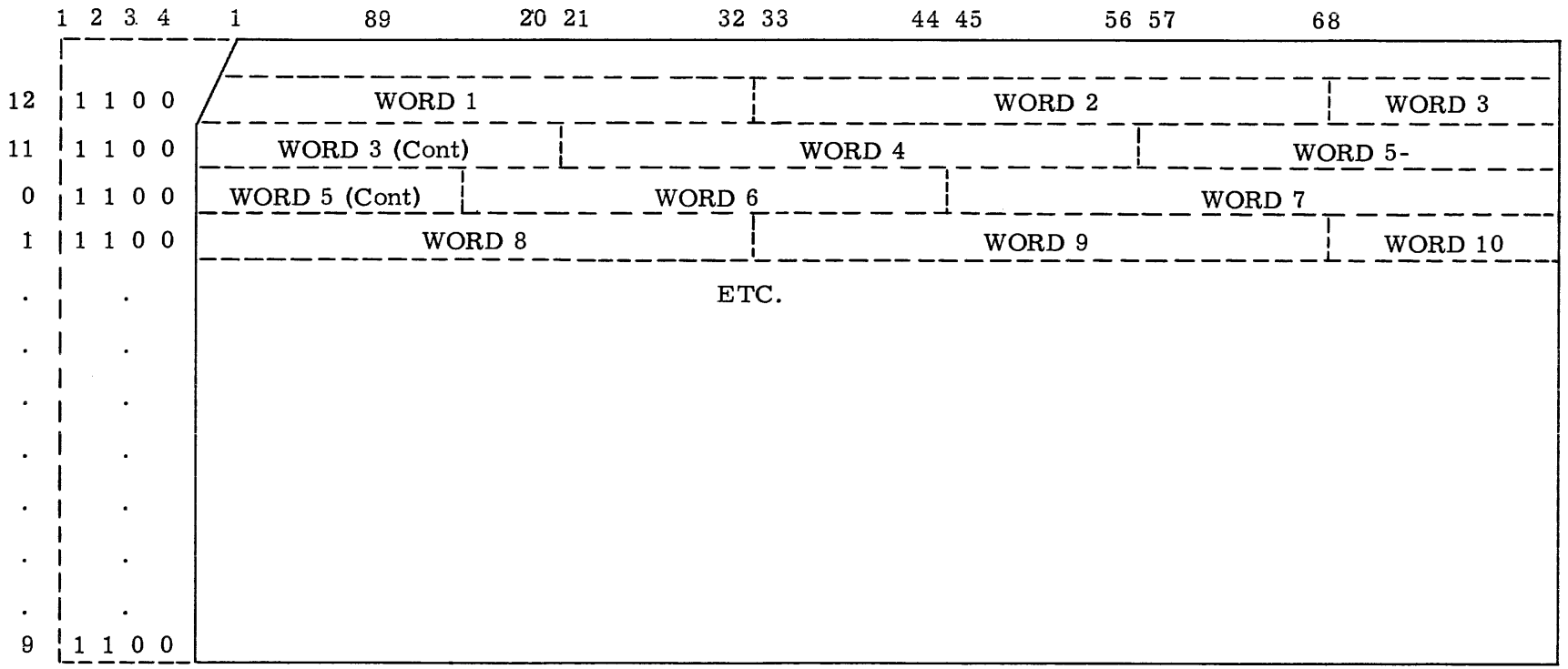


Figure V-5. Format of Data on NISN Card. This Data, Words 1-28, Will Appear in Successive Locations When Read into Memory.

SECTION VI

CONSOLE

GENERAL

The 9400 Console is designed to provide maximum access to the Central Processor for the operator and programmer. The lower half of the Console panel contains a set of switches which are operated manually and/or under program control. The upper half of the Console panel contains a set of indicator lights which exhibit the state of the electronic switches which store the binary digits within each word. The register indicator lights are placed in groups of three to facilitate binary-to-octal conversion.

Figure VI-1 is a general view of the System while Figure VI-2 is a detailed drawing of the Console switches and indicators.

FEATURES

The Console contains many features useful for a programmer testing his program, an operator monitoring production runs, or maintenance checking and analysis, as follows:

1. The contents of the Accumulator, Instruction Word Register, Program Counter, each of four Index Registers, and any one of seven other selectable registers can be displayed continuously.
2. The contents of each addressable register can be changed by setting the desired word and its address in switches and pressing a single switch.
3. The computer can be operated in special modes to aid in debugging and troubleshooting.
4. Initiating controls are used to read in programs from peripheral equipment or Console switches, and start a program at the location specified by the switches.
5. Programs can be checked out in segments by manually setting the sensable switches and using transfer and sense orders throughout the program to activate a program and type out important memory locations. After debugging, the sensable switches can be placed under program control so that the program will run without interruption.
6. Special switches are provided which allow the operator or programmer to exercise optional control over various computer operations.
7. A Halt switch on the Console enables the operator to stop the computer without destroying the contents of the active regisers, and thereby to restart under normal operation.

DESCRIPTION

The 9400 Console panel consists of two major parts, namely, the Console Switches and the Console Indicators.

CONSOLE SWITCHES

The Console Switches are described under the following categories: Mode Switches, Initiating Switches, Sense Switches, Special Switches, Register Switches, and Miscellaneous Switches.

MODE SWITCHES

The 9400 Computer can operate in any one of three modes, depending upon the following three switches.

Run (RUN) — This switch provides the gating levels necessary for operating the computer continuously under program control. Once the computer is started by depressing the Run switch, it can be stopped only by the program, the operator, or an error. The Run switch when operated releases the other two mode switches, described below.

One Instruction (ONC) — This switch provides the gating levels necessary to stop the computer after it has executed one instruction. This mode of operation is useful in program checking and computer maintenance.

Single Step (SIP) — This switch provides the gating levels necessary to inhibit the normal pulse distribution system and to allow the computer to stop after each timing function time during the execution of an instruction. This operating mode is useful for computer maintenance.

INITIATING SWITCHES

The initiating switches are used to start computer operations, to read in programs into the computer from punched paper tape, magnetic tape, and card readers, and also to permit manual read-in and read-out operations. These switches are described below.

Program Read-in (Load from CRD RDR (Card Reader) Load from MTU (Magnetic Tape Unit)

Figure VI-1. 9400 System, General View

Load from PTU (Paper Tape Unit) — Three switches are provided on the Console to read programs into the computer from an input device (using punched cards, magnetic tape, or punched paper tape) and to start the program. The switches are: Load from CRD RDR, Load from MTU, and Load from PTU. The input devices are, respectively, Card Reader, Magnetic Tape Unit and Paper Tape Unit. Either the RUN switch or the ONC switch must be previously depressed.

The first word on the tape or card is a "read" instruction which specifies the address of the in-out device containing the program to be read in, the number of words to be read in, and the memory location into which the program is to be read. The second word contains the address to which computer control is to be transferred upon completion of a program read-in operation.

Start at PC — This switch initiates an instruction sequence in which the location of the first instruction is specified by the contents of the Program Counter. Depressing the Start at PC switch following a central-processor halt (CP Halt) will cause the Central Processor to resume normal operation.

Start at ASR — This switch initiates an instruction sequence in which the location of the first instruction is specified by the contents of the Address Switch Register. There are 15 individual switches for a 15-bit binary address.

Manual Instruct — This switch initiates a sequence in which the instruction specified by the contents of the Word Switch Register is executed.

Read-In — This switch initiates a sequence which reads the contents of the Word Switch Register into the register or memory location specified by the contents of the Address Switch Register.

Read-Out — This switch initiates a sequence which causes the contents of the memory location specified by the contents of the Address Switch Register to be read into the Memory In-Out Register (MO). The contents of the Memory In-Out Register may then be displayed on the Bus Indicator Register by pressing the Register Selector MO switch.

SENSE SWITCHES

Sixteen 3-position sense switches are provided on the Console, and are used for general switching control. The three switch positions are: set, reset, and neutral. The sense switches can be set or reset either manually or under program control. In the neutral position, these switches operate under program control only, and can be interrogated by a SENSE order (i.e., SEN, SNS, or SNR). In the set or reset positions, the switches are manually controlled and cannot be altered by the program. The sense switches are not interlocked with the halt circuit and can, therefore, be set or reset at any time, even while the computer is operating. Two indicating lights are provided with each switch to display its state (set or reset).

Special Switches on Console

Seven 3-position special switches are provided on the Console. The three switch positions are: set, reset, and neutral. The special switches are similar in many ways to the sense switches, differing only in the specialized control functions performed. For example, these switches can be set or reset either manually or in neutral position. When in the neutral position the flip flops associated with these switches can be set or reset under program control. However, unlike the sense switches, the special switches are interlocked with the halt circuit and may be set or reset under manual control only when the computer is halted. Also, some of the special switches are spring-loaded (i.e., if the switch is manually set or reset, when the computer is halted, the switch will return to the neutral position immediately after being manually set or reset, leaving future control to the program).

Two indicating lights are provided with each switch to display its state (set or reset).

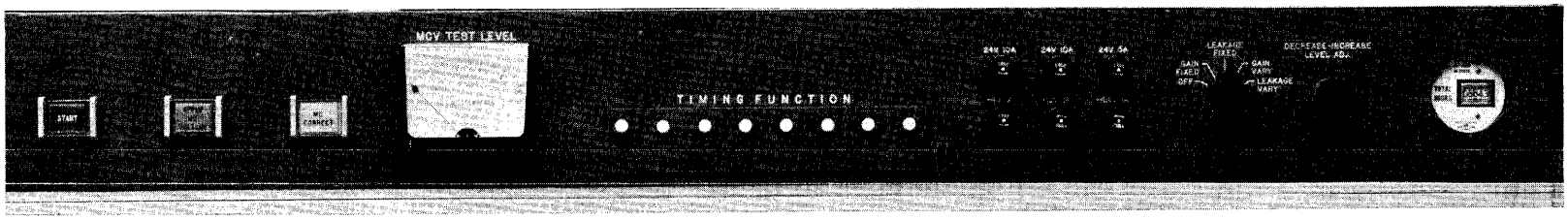
The special switches are listed below:

1. Interpret Sign (ISN).
2. No Halt on Processor Error (NHP).
3. Trapping Mode (TRA).
4. Alarm Program Interrupt (API).
5. Functional Control Character Interrupt (FCI).
6. Stop Program Interrupt (SPI).
7. Tape Erase (TPE): loaded in the set spring position.

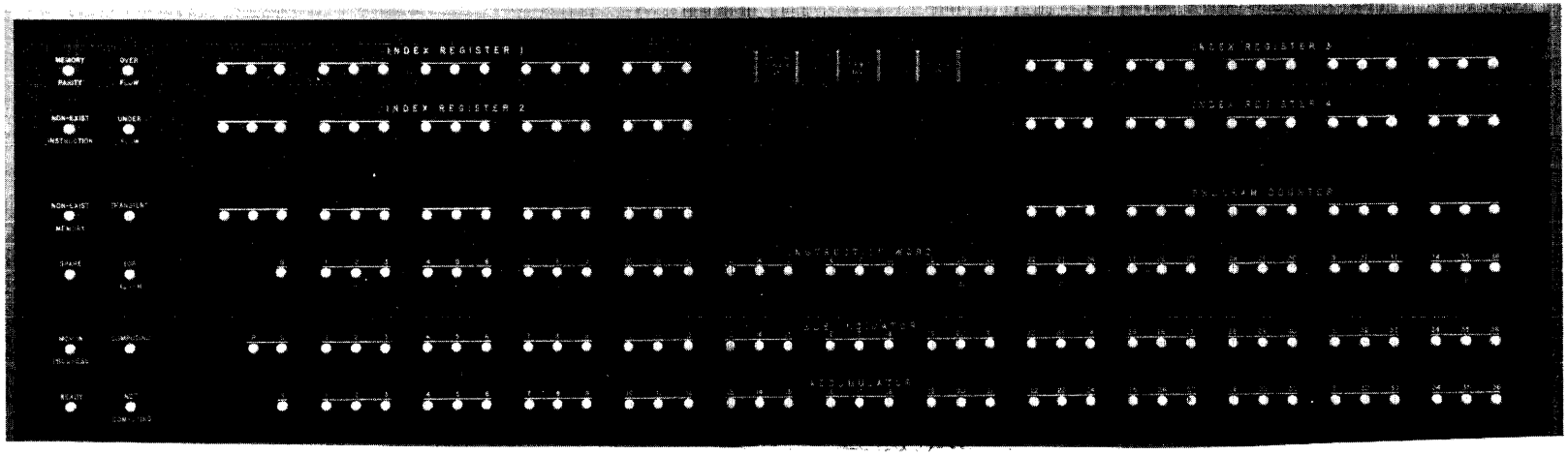
REGISTER SWITCHES

Two sets of register switches are provided on the Console, namely, the ASR switches and the WSR switches. These switches and their associated indicator lights are placed in groups of three to facilitate binary-to-octal conversion. They are described below.

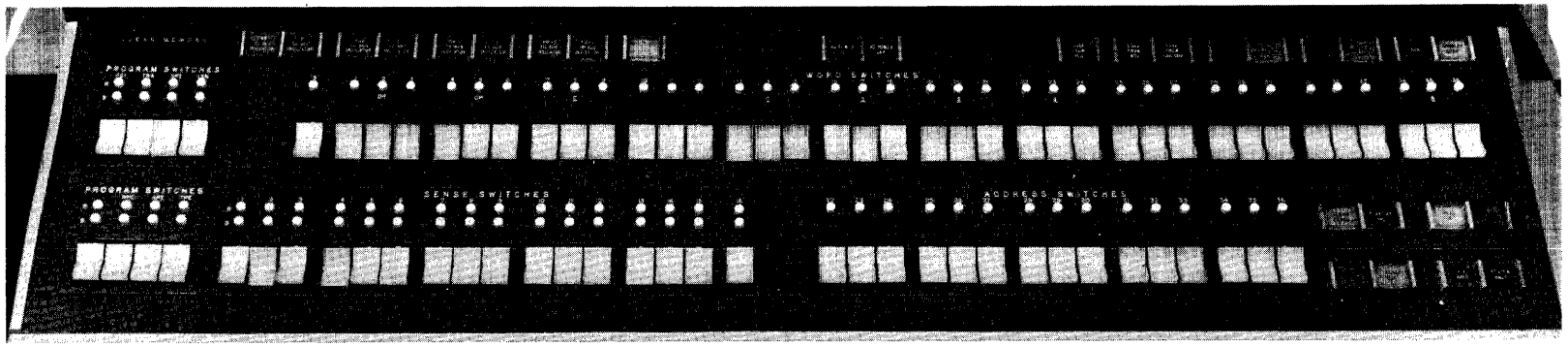
Address Switch Register (ASR) — The Address Switch Register, consisting of 15 switches provided on the Console, can be set manually to store at any address. A set of 15 indicators located immediately above the address switches is used to display the contents of the Address Switch Register. The ASR designates the address for Console operations during Read-In, Read-Out, and Start at ASR.



MCV PANEL



UPPER PANEL



LOWER PANEL

Figure VI-2. 9400 Console Switches and Indicators

Word Switch Register (WSR) — The Word Switch Register, consisting of 37 switches and associated display indicators, is provided on the Console. A word can be set up manually in the switches and can then be read into the register or memory location specified by the ASR. Alternatively, the instruction specified in the WSR can be executed by means of the manual initiating controls (Manual Instruct).

MISCELLANEOUS SWITCHES

These switches are used for various functions necessary for the proper operation of the computer, and are described below.

Register Selector — Eight register-selector switches are provided on the Console. These push-button switches are used to select and display the contents of various registers on the Bus Indicator Register which has 38 display lights. A special interlock circuit prevents the contents of the selected register from being read onto the main transfer bus while the computer is operating. The contents of any one of the following registers may be displayed by depressing the appropriate register-selector switch:

1. Memory In-Out Register
2. Program Counter Store
3. B-register
4. Q-register
5. Processor 1
6. Processor 2
7. Processor 3
8. Processor 4

A spare register-selector switch is also provided on the Console panel.

CP Halt — Pressing the Halt switch causes the Central Processor to suspend operation at the completion of the Central Processor instruction currently being executed. All input-output operations, however, are allowed to run to completion. As soon as the last in-out operation has been completed, the Central Processor goes into a halt (not computing) condition. Depressing the Start at PC switch, following a halt, will cause the Central Processor to resume normal operation.

System Halt — Pressing the System Halt switch causes the Central Processor to go into a Halt condition. At the same time, all in-out operations are terminated by clearing the In-Out Processor control registers. Because of the destruction of the in-out instructions, restarting the computer is not normally possible following a System Halt.

Clear Memory — The Clear Memory switch resets to ZERO the contents of all core memory locations, except the parity location which is set to ONE for odd parity. Normally, this is done before each new program is read into the computer.

Clear CP — The Clear Central Processor switch resets to ZERO all the CP switches (not the memory), except the Halt switch which is set, and the applicable Timing Function Generator switches which are also set (to insure that the computer is restarted at the proper point of the basic cycle, i.e., TF-4).

Clear Error — This switch resets all error alarms displayed on the indicator panel of the Console.

Halt on Transient — When this switch is depressed (in the ON condition), the Transient indicator on the Console will light and the computer will halt if an intolerable power transient is detected in the main power supply lines. An intolerable power transient would cause computer error. However, when this switch is in the OFF condition, the Transient indicator will light although the computer will not halt if an intolerable power transient is detected.

No Halt on Error — When this switch is depressed (in the ON condition), it inhibits error halts in the Central Processor. When it is in the OFF condition, any of the error alarms will cause the computer to stop. These alarms are:

1. Memory Parity Error
2. Overflow Alarm
3. Underflow Alarm
4. In-Out Processor Alarm
5. Nonexistent Memory
6. Nonexistent Instruction

Refer to the Alarm Indicators part of this section under Console Indicators.

AC Power On — This switch initiates the application of a-c power to the main computer.

AC Power Off — This switch electrically disconnects the main computer from its a-c power supply.

CONSOLE INDICATORS

The Console Indicators are described under the following categories: Register Indicators, Alarm Indicators, and Status Indicators.

REGISTER INDICATORS

The Register Indicators consist of various numbers of rows of indicator lights to display the contents of the associated registers. Also, one row of 15 indi-

cator lights are provided as spares. The register indicators are described below:

Index Registers — Four rows of 15 indicator lights are provided on the Console to display the contents of the 15-bit Index Registers 1 through 4.

Program Counter — One row of 15 indicator lights is provided on the Console to display the contents of the 15-bit Program Counter Register. The contents of the Program Counter indicate the memory address of the next instruction to be performed by the computer.

Bus Indicator — One row of 37 indicator lights (plus a 38th indicator light which indicates parity) is provided on the Console to display the contents of various registers depending on which Register Selector Switch is depressed. (See Console Switches, register Selector.) The bus indicator register circuits are connected directly to the main transfer bus, and therefore, indications of computer operations are possible since the bus indicators monitor every bus transfer.

Instruction Word — One row of 37 indicator lights is provided on the Console to display the contents of the Instruction Word Register. The Instruction Word Register is composed of the Address, X, G, and Instruction Registers (AR, X, G, and IR) which contain the last instruction executed.

Accumulator — One row of 37 indicator lights is provided on the Console to display the present contents of the Accumulator (A-register).

ALARM INDICATORS

Eight error alarm indicators are provided on the Console to indicate the state of the various error and alarm electronic switches in the computer. All of the error and alarm electronic switches can be reset by the Clear Error switch. Also, the No Halt on Error switch inhibits error halts in the Central Processor when an MPE, NXI, NXM, OA, UA, or IOA alarm is set (see Console Switches). The various alarm indicators on the Console are described below:

Memory Parity Error (MPE) — This indicator lights when incorrect parity has been detected upon testing parity in a memory read-out operation.

Nonexistent Instruction (NXI) — This indicator lights when an improper order code has been designated in an instruction.

Nonexistent Memory (NXM)—This indicator lights when a nonexistent memory location has been addressed.

Overflow (OA) — This indicator lights when an overflow condition has occurred. (See Overflow Chart, Section III, CENTRAL PROCESSOR UNIT).

Underflow (UA) — This indicator lights when an underflow conditions has occurred. (See Overflow Chart, Section III, CENTRAL PROCESSOR UNIT).

Transient — This indicator lights when an intolerable power transient is detected in the main power supply lines. It warns the operator that computer errors may have been caused by the transient. (See Console Switches, Halt on Transient.)

In-Out Processor Alarm (IOA) — This indicator lights when an In-Out Processor error has occurred (i.e., parity error, incorrect instruction, etc.).

DC Power Fail — This indicator lights when a d-c power failure occurs.

STATUS INDICATORS

The Status Indicators inform the operator of various computer states such as d-c power, peripheral equipment power, computation in progress, computer is stopped although ready to operate, and marginal checking operations.

DC Power On — This indicator lights approximately 35 seconds after the AC Power ON switch has been operated, indicating that d-c power has been applied to all units of the computer except the Core Memory.

Peripheral On — This indicator lights when power is applied to the peripheral equipment.

Ready — This indicator lights 25 seconds after the Clear Memory switch has been operated, indicating that d-c power has been applied sequentially to the Core Memory.

Computing — This indicator is lighted when the Central Processor is computing. It is connected to the primed output of the halt switch and interlocked with the memory Ready indicator circuit.

Not Computing — This indicator is lighted when the computer is halted although the memory is ready to operate. It is connected to the output of the halt switch and interlocked with the memory Ready indicator circuit.

MCV in Progress — This indicator is lighted when marginal check voltages are applied to the computer for marginal testing.

OPERATION

As previously mentioned, the 9400 Computer can operate in any one of three modes, namely, Run, One Instruction, and Single Step. The mode switches, when operated in conjunction with the initiating switches, initiate various program sequences.

PROCEDURES

Any one of 17 different program sequences can be initiated by these switches, as detailed below:

Run, Start at Address Switch Register (RUN STW)

The program sequence is started at the Address Switch Register by depressing the Start at ASR switch. Thereafter, the computer operation is under program control.

One Instruction, Start at Address Switch Register (ONC STW)

This operation is similar to RUN STW except that only one instruction is performed for each operation of the Start at ASR switch. Normally, this switch is used only to start the program sequence. The Start at PC switch is used thereafter.

Single Step, Start at Address Switch Register (SIP STW)

This operation is similar to RUN STW except that the computer executes the sequence one pulse at a time (i.e., one pulse each time that the Start at ASR switch is operated).

Run, Start at Program Counter (RUN COW)

The program sequence is started at the Program Counter Register by depressing the Start at PC switch. Thereafter, computer operation is under program control.

One Instruction, Start at Program Counter (ONC COW)

This operation is similar to RUN COW except that only one instruction is performed for each operation of the Start at PC switch.

Single Step, Start at PC (SIP COW)

This operation is similar to RUN COW except that the computer executes the sequence one pulse at a time.

Run, Program Read-In (RUN PRW)

This operational procedure is used to read programs into the computer. One of three switches on the Console (i.e., Load from PTR, Load from MTU, or Load from CRD RDR) initiates program read-in depending on whether the program is recorded on cards, punched paper tape, or magnetic tape. Also, the RUN switch must be previously depressed. The first word on the tape or card is a "read" instruction which specifies the address of the in-out device containing the program to be read in, the number of words to be read in, and the memory location into which the program is to be read. The second word contains the address to which computer control is to be transferred upon completion of program read-in operation.

One Instruction, Program Read-In (ONC PRW)

This operation is similar to RUN PRW except that the computer halts after the program has been read in.

Run, Manual Instruction (RUN MIW)

To perform this operation, the instruction in the Word Switch Register is transferred first to the Instruction Word Register and then is performed repeatedly. Repetition may be interrupted only by the operator or by an error.

One Instruction, Manual Instruction (ONC MIW)

This operation is similar to RUN MIW except that the instruction is performed only once.

Single Step, Manual Instruction (SIP MIW)

This operation is similar to RUN MIW except that the instruction is performed one pulse at a time. The first pulse is initiated by depressing the Manual Instruct switch. Succeeding pulses are initiated by depressing the Start at PC switch.

Run, Read-In (RUN RIW)

To perform this operation, the contents of the Word Switch Register are transferred to the addressable register or memory location specified by the contents of the Address Switch Register.

One Instruction, Read-In (ONC RIW)

This operation is similar to RUN RIW except that the operation is halted after the read-in instruction has been performed once.

Single Step, Read-In (SIP RIW)

This operation is similar to RUN RIW except that the gating levels cause the read-in operation to halt after one operation is completed.

Run, Read-Out (RUN ROW)

This operational procedure provides an indirect means of gaining access to any memory location. First, the contents of the Address Switch Register are transferred to the Memory Address Register and then a memory cycle is initiated. As a result, the contents of the addressed memory location are transferred to the Memory Output Register. The contents of the Memory

Output Register can be displayed on the Bus Indicator Register.

One Instruction, Read-Out (ONC ROW)

This operation is similar to RUN ROW except that the gating levels cause the read-out operation to halt after one operation has been completed.

Single Step, Read-Out (SIP ROW)

This operation is similar to RUN ROW except that the gating levels are such that the Read-Out switch is actuated to initiate the first pulse of the operation, and additional pulses of the operation are initiated by depressing the Start at PC switch.

SECTION VII

SYMBOLIC PROGRAMMING

INTRODUCTION

Symbolic programming consists essentially of writing computer programs using symbols which are not in the language of the computer. It is also normally considered to be *non-numeric* in nature, in that wherever practicable, alphabetic symbols are employed in place of numbers.

The primary purpose of symbolic programming is to relieve the programmer of the difficulties inherent in dealing with a numeric machine language, both in original program design and later program modification. Perhaps the most obvious advantage of symbolic programming is the use of alphabetic *names* for operation codes, such as ADD, SUB, MLY, and RANB, in place of the corresponding binary or octal codes. Further, by allowing the use of symbols instead of numbers in the variable field of an instruction, symbolic programming enables the design of a program without restricting it to any particular part of the computer memory and without the necessity of remembering and rechecking numerical addresses and codes. In addition, symbolic programs may be relocated in memory without requiring that the symbolic addresses be changed.

Symbolic programming is made possible through the use of an Assembly Program. The 9400 System makes use of the 9400 Symbolic Assembly Program, called 94AP. A detailed description of 94AP is provided in a separate manual. Following is a general picture of the function and operation of 94AP.

CONCEPT OF AN ASSEMBLY PROGRAM

When an assembly program is available, a programmer is allowed to write his program in symbolic notation instead of a numeric computer language. He is provided with a list of symbols and types of symbol which are acceptable to the assembly program, together with a list of the symbolic programming rules to which he must adhere in preparing his program. Equipped with the rules and the vocabulary for symbolic programming, the programmer may write his program.

It is the function of the assembly program to *translate* the symbolic program into actual machine language. Where the programmer makes use of alphabetic operation codes, the assembly program converts these codes into the binary equivalent of the operation to be executed. Where the programmer uses symbolic addresses for registers, switches and memory locations, the assembly program converts these addresses into their actual binary values. During the assembly process, the assembly program checks the symbolic program for consistency in notation and for conformity to certain program format requirements.

In addition to legitimate machine operation codes (such as ADD, WAN, SS, et cetera), the assembly program is capable of interpreting and acting upon a number of Pseudo-Operations. Pseudo-operations are instructions to the assembly program itself. They pertain primarily to data format and allocation of memory space.

9400 ASSEMBLY PROGRAM (94AP)

The 9400 Assembly Program is a general assembly program which converts symbolic programs into machine-coded binary instructions and data words. The medium for input and output may be chosen by the programmer, although punch cards are normally used. The exact operation of each of the various pseudo-operations which are acceptable to 94AP is defined in Appendix J.

94AP processes a symbolic program as a *list*. The assembly program is functionally divided into two sections. The first section consists of a *pass* over the input program, determining the actual machine addresses which are to be assigned to each symbolic reference. Each symbolic reference and its machine address equivalent are saved in a *symbol table* for later reference. During the first pass, the assembly program generates a copy of the input program in its symbolic form. The second section consists of a pass over the copy of the input program. Using the previously-generated symbol table, the assembly program forms the actual binary computer words which are to make up the machine-language program. At the same time, a binary (machine-language) output is provided on punch cards or magnetic tape. In addition, a listing is prepared containing a copy of the original symbolic program and its binary (printed in octal) equivalent. If the assembly program discovers inconsistencies such

as incorrect operation codes or symbolic references which are undefined, these errors are reported on the listing provided.

DEFINITIONS

Following are definitions of four terms used in reference to the 9400 Assembly Program:

1. Symbol: Any combination of not more than six alphanumeric characters, at least one of which is non-numeric.
2. Integer: With respect to instructions, any decimal integer less than 1,000,000.
3. Expression: Arithmetic combinations of symbols and/or integers, specifically in the variable field; the operations permitted are addition, subtraction, multiplication, and division.
4. Pseudo-Operation: Instructions to the assembly program, as described above.

SYMBOLIC CODING FORMS

Figure VII-1 is a standard 94AP Coding Form. Each operation or data location in a program may be given a symbolic address, under SYMBOLIC LOCATION. Symbolic instructions and orders are written under OP CODE; when a data word is specified, the OP CODE portion contains a pseudo-operation code, as required. The VARIABLE FIELD consists of combinations of symbols and numbers, depending upon the operation or the type of data. Under REMARKS, the programmer may insert comments which will have no effect on the program or its assembly other than to be printed out with the final listing. The IDENTIFICATION column is provided for punching identifying marks in the symbolic program card deck.

During the first pass of the assembly program, a counter is used to specify the absolute location of each word in the program. The location counter starts at the binary equivalent of the decimal location given in the variable field of the pseudo-operation ORG (origin) with which every symbolic program must begin. For each word in the program, the counter is increased by ONE. Simultaneously with the incrementing of the location counter, the symbol table is constructed. Each entry in the table defines a symbol used in the program in terms of an integer. Entries to the Symbol Table are made in two ways:

1. A symbol appears in the symbolic location field of a word and is assigned the value of the location counter, or
2. A symbol may be defined through the use of a pseudo-operation.

SYMBOLIC INSTRUCTIONS, ORDERS AND PSEUDO-OPERATIONS

The OP CODE portion for each symbolic instruction, order or pseudo-operation is represented by the mnemonic symbol assigned to it. The mnemonic code for each machine operation (as well as its octal equivalent) is given in Appendix I; the mnemonic codes for pseudo-operations are listed in Appendix K. The codes for addressable registers and sensable switches are contained in Appendices D and E respectively.

Figure VII-2 is a simple program written in symbolic notation acceptable to 94AP. Those operations which contain an asterisk (*) in the Remarks field are pseudo-operations; those which do not are legitimate machine operations. The asterisk is used as a means of identification for this program only and has no connection with or effect upon the assembly program.

Under OP CODE, the symbols for instructions and pseudo-operations are written. If it is desired to refer to a particular location, an *alphanumeric* location reference may be made under SYMBOLIC LOCATION. In the VARIABLE FIELD, the address portion (a), the index portion (i), and the modifier portion (m) are written from left to right in that order and separated by commas. If only the a portion is used, the i and m portions may be left out completely. If either the a portion or the i portion is not required, but a portion to the right of them is, a ZERO should be written in the unused portion of the variable field. For example, the ADD instruction in the program shown in figure VII-1 requires an m portion for overflow control, but no i portion. If the middle portion were left out, the assembly program would interpret the overflow control number 7 as referring to an index register.

The variable field is terminated by a *space;* whatever is written following the space is interpreted as remarks and has no effect on the assembly.

The program shown in figure VII-2 will add the contents of one memory location (ALPHA) to the contents of another memory location (BETA) and store the sum in a third memory location (ANSWER). The first operation in a symbolic program is always the pseudo-operation ORG (for origin). The variable field for ORG is normally a decimal number. The binary equivalent of the number is loaded into the 94AP location counter, and the program is assembled to start in the memory location specified by that number. Thus, in the program illustrated, symbolic location START is given the value 1000 in the symbol table, and the operation CLA will be built in memory location 1000. During the first pass, 94AP does not process the contents of the variable fields, with the exception of the ORG pseudo-operation. Operation codes are checked and the symbol table is built, in which ALPHA is assigned a value of 1004, BETA a

94 AP CODING FORM

PROJECT NO.	PROGRAMMER	LOCATION	DATE	PAGE ☐☐☐ OF _____	PROGRAM IDENTIFICATION ☐☐☐☐
PROGRAM DESCRIPTION					

PAGE	LINE	LOCATION	OPERATION	VARIABLE FIELD	REMARKS	PROGRAM IDENT.

Figure VII-1. Standard 94AP Coding Form

94 AP CODING FORM

PROJECT NO.	PROGRAMMER	LOCATION	DATE	PAGE	PROGRAM IDENTIFICATION
PROGRAM DESCRIPTION				OF	

PAGE	LINE	LOCATION	OPERATION	VARIABLE FIELD	REMARKS	PROGRAM IDENT.
			ORG	1000	*Start program in location 1000	
		START	CLA	ALPHA	Contents of loc. ALPHA → acc	
			ADD	BETA, 0, 7	Contents of loc. BETA added to C(acc) overflow is ignored	
			STR	ANSWER	Store Σ ALPHA, BETA in loc ANSWER	
			HLT		Central processor halt	
		ALPHA	DEC	150	*Place binary equivalent of decimal 150 in loc. ALPHA	
		BETA	DEC	110	*Place binary equivalent of decimal 110 in loc. BETA	
		ANSWER	BSS	1	*Reserve one word at loc. ANSWER	
			END	START	*Signal to 94AP that program is complete	

Figure VII-2. A Symbolic Program

value of 1005, and ANSWER a value of 1006. The pseudo-operation END indicates to the assembly program that the end of a functional unit, i.e. the program to be assembled, has been reached. END is not placed in the symbol table nor given an address, since it is not an actual part of the program. Its variable field portion (START) enables the assembly program to generate an unconditional transfer to the beginning of the program.

During the second pass, the assembly program builds the complete binary image of each operation between locations 1000 and 1006; the address portion of CLA ALPHA is converted to the binary equivalent of 1004, the address portion of ADD BETA,0,7 is converted to the binary equivalent of 1005,0,7 and so on. The numbers 0 and 7 are already pure numeric, in that they do not contain alphabetic characters, and are converted directly as such.

The DEC pseudo-operation causes the binary equivalents of 150 and 110 to be placed in locations ALPHA and BETA, respectively. The BSS (block start symbol) causes the 94AP location counter to be advanced by ONE, causing one memory location (as specified by the variable field) to be reserved for ANSWER.

During assembly, 94AP checks to see that the mnemonic codes for operations are correct and that the program is written in acceptable format. It also checks to see that the symbols used in the variable field are *defined*. If, for example, in the program illustrated, the programmer had referred to location BETA in the variable field of the ADD instruction, but had forgotten to define it by providing a data location called BETA, the assembly program would *flag* his error in the printed listing provided at the end of the assembly. The assembly program also comments in the listing upon incorrect operation codes (such as ADX instead of ADD) and inadmissable symbols and sequences of symbols.

PUNCH CARD FORMAT

Symbolic program decks consist of one card for each operation. Each line on a 94AP coding form requires one punch card to represent it. The format in which symbolic operations are punched is as follows:

1. SEQUENCE NUMBER: Columns 1 through 6 are optional; it provides the programmer with a means of specifying the ordering of his program deck.
2. SYMBOLIC LOCATION: Columns 7 through 12; if a symbol is to be assigned to a line of code, it is entered here.
3. OP CODE: The operation code starts in column 14 and may extend through column 19, depending upon the number of suffixes.
   The OP CODE field must be terminated before column 21 by a blank.
4. VARIABLE FIELD: The variable field starts at column 21 and may continue effectively to the end of the card. No blanks may be left between the portions of the variable field, since information punched to the right of the first blank is interpreted as remarks.

Arithmetic Operations in the Variable Field

The 9400 Assembly Program permits integer arithmetic within the variable field. It is possible to perform addition, subtraction, multiplication, and division of numbers and alphanumeric symbols. Addition and subtraction are indicated by conventional plus (+) and minus (−) signs. A symbolic instruction may be written as ADD A-B, for example. Multiplication is indicated by an asterisk (*), and division by a slash (/). The following rules have been established for the evaluation of arithmetic expressions in a variable field.

1. Each segment of the expression, where a segment is that portion of the expression from a plus or minus sign (or the beginning of the expression) to the next plus or minus sign (or the end of the expression), is separately evaluated from left to right with the consecutive multiplication and division being performed as specified.
2. As consecutive segments of the expression are evaluated, they are combined from left to right as indicated by the connective plus and minus signs.
3. All operations are 36-bit integer arithmetic calculations. In multiplication the low 36 bits of the product are retained; in division, the 36-bit quotient is retained. Remainders are ignored.

By way of illustration, the expression

A+200/15/6*15-B/C*D

is taken to have the meaning of

((A+(((200/15)/6)*15))-((B/C)*D))

where the parentheses denote the integral portion of the quotient described above.

SPECIAL SIGNIFICANCE OF ASTERISK (*) AND SLASH (/)

If an asterisk appears as the first character of the variable field, or follows immediately after a comma, plus sign, minus sign, slash, or two consecutive asterisks, the assembly program will substitute for the asterisk the current value of the 94AP location counter. For example, the operation TRU *-3, stored

in memory location 1006, is translated as meaning unconditional transfer to memory location 1006 *minus* 3, or 1003.

If an asterisk appears in column seven of a card in a symbolic deck, the assembly program treats the entire card as a remark. It will be reproduced in the output listing, but no other interpretation of the card will be made.

If a slash appears as the first character in the variable field or immediately after a connector, it will signal the assembly program to treat the following integer as an *octal* number. If the preceeding connector is an asterisk, its use must be that of multiplication; otherwise the slash will indicate division, as in the normal case.

ILLUSTRATIVE SYMBOLIC PROGRAMS

Figure VII-3 is a symbolic program for processing data read in from magnetic tape in Single Instruction Mode. The program reads 50 signed words, none of which contain all ZEROes, from magnetic tape unit 3. As the words are read into core memory, a count is kept of those which are *negative*.

The ORG (origin) pseudo-operation specifies that the program is to begin in memory starting at decimal location 00500; the symbol START will be assigned the value 00500, and the 94AP location counter will begin there.

The operation CLA ZERO clears the Accumulator. 94AP interprets the code ZERO as a non-existent addressable register, with the result that the Accumulator is loaded with all ZEROes. Once the Accumulator is cleared, the STR COUNT instruction causes memory location COUNT to be cleared also. The RPT-STR sequence following clears 50 consecutive memory locations starting at location DATA. For details concerning the REPEAT function, see Central Processor Instructions in Section III.

The SNS instruction SETS the Interpret Sign (ISN) switch. The assembly program translates the mnemonic ISN into the binary designation for the Interpret Sign switch. In Single Instruction Mode, the ISN switch must be SET before each input-output instruction where interpret sign mode operation is desired.

The input-output instruction RAN (Read Alphanumeric) causes an In-Out Processor to be selected and attached to magnetic tape unit 3. MT3 is interpreted and translated by the assembly program. As soon as the selected process has been initiated, the Central Processor program proceeds to the next instruction, in this case LXS. When a Processor is immediately available, the Central Processor program is not held up at all.

While the magnetic tape unit is preparing to read, LXS causes index register 1 to be cleared (e.g. loaded with ZEROes) and index register 2 to be loaded with the binary equivalent of decimal 50. Immediately following initialization of the two index registers, a check is made to see if a word has arrived in memory from magnetic tape. On the first time through the program, for example, Index Register 1 contains all ZEROes. Thus the instruction CLA DATA, 1 causes the contents of location DATA to be loaded into the Accumulator. If a word from magnetic tape has not yet arrived in location DATA, that location will contain all ZEROes, since the RPT-STR sequence earlier in the program cleared it out. Consequently, the TRZ (Transfer on Accumulator ZERO) instruction will *succeed* and control will be transferred to *-1, or back to the CLA DATA, 1 instruction. In this manner, the program is held in a loop until data arrives from magnetic tape. As soon as a word arrives, the TRZ instruction *fails* and control is passed on to the next instruction in sequence, namely TRN (Transfer on Accumulator Negative).

If all 50 of the words read from magnetic tape were positive, the TRN instruction would consistently fail and the program would always proceed directly to the TRX (Transfer on Index) instruction. Each time TRX is executed, the contents of Index Register 1 are incremented by ONE and the contents of Index Register 2 are decremented by ONE, except at the end of the Loop, when the contents of Index Register 2 have been counted down to ONE. Thus, the major loop in the program is between the TRX instruction and location LOOP. Each time CLA DATA, 1 is executed, a new word in memory is loaded into the Accumulator. Once TRX has counted the contents of Index Register 2 completely down, the Central Processor halts (HLT).

If the program encounters a negative word, the TRN instruction succeeds and control is transferred to location NEGA below. The ADB (Add Modifier) instruction causes the contents of location COUNT (originally ZERO) to be incremented by ONE each time a negative word is found. The Unconditional Transfer (TRU) instruction returns the program to the TRX instruction, and the loop continues.

Two sets of data locations are reserved for the program. DATA BSS 50 saves 50 locations for the incoming data. COUNT BSS 1 saves one location for the negative-word counter. The END operation signals the assembly program that the end of the symbolic program has been reached. ORG, BSS, and END are pseudo-operations and do not appear in the machine-language program.

Figure VII-4 is an Order Sequence Mode program involving both an input operation and error print-

94 AP CODING FORM

PROJECT NO.	PROGRAMMER	LOCATION	DATE	PAGE	PROGRAM IDENTIFICATION
				OF	

PROGRAM DESCRIPTION Read 50 signed non/zero words from MT3; keep a count of the negative words. Compute simultaneously with input in Single Instruction Mode.

PAGE	LINE	LOCATION	OPERATION	VARIABLE FIELD	REMARKS	PROGRAM IDENT.
			ORG	500	Start program at location 500	
		START	CLA	ZERO	Clear the accumulator	
			STR	COUNT	Clear location COUNT	
			RPT	49, 0, 1	Clear 50 consecutive memory locations,	
		*			starting at location DATA	
			STR	DATA		
			SNS	*+1, 0, ISN	SET Interpret Sign Mode switch	
			RAN	DATA, MT3, 50	Initiate alphanumeric read operation	
			LXS	0, 1, 50	Load IR2 with 50; IR1 with ZERO	
		LOOP	CLA	DATA, 1	Check to see if a word has arrived in	
					memory yet. If not, loop	
			TRZ	*-1		
			TRN	NEGA	Transfer if the word is negative	
			TRX	LOOP, 1, 1	Transfer on index	
			HLT		Central Processor halt	
		NEGA	ADB	COUNT, 0, 1	Increment contents of COUNT by ONE	
			TRU	NEGA-2	Transfer back to index	
		DATA	BSS	50	Save 50 locations for data	
		COUNT	BSS	1	Save one location for COUNT	
			END	START		

Figure VII-3. SIM Read Program

out. The program searches the first file on magnetic tape unit 1 for a *key* of decimal 5. When the key is located, the following 200 signed words are read into memory. Parity errors, if encountered during input, are not allowed to stop the operation. If the first file does not contain a key of 5, the message "no key on mt1" is printed out on the electric typewriter.

The program begins in memory location 00750. The first operation is a rewind instruction, which causes magnetic tape unit 1 to be rewound to the load point on the tape. The second operation is the Order Sequence Instruction SS, exercising several options.

The third S causes the ISN switch to be SET for all operations in the following Order Sequence. The suffix N causes the No Halt on Processor Error switch (NHP) to be SET, so that possible parity errors will not stop the operation. The integer 1 specifies that the Order Sequence program is to use Processor 1. The variable field of SSSN1 indicates that the Order Sequence program beginning in location READ and that magnetic tape unit 1 is to be the medium.

The instruction following SSSN1 is sense (SEN). The variable field causes the Central Processor to wait at the SEN instruction until the Order Sequence initiated by SSSN1 is completed, i.e. as long as MT1 is still connected. As soon as the sequence is complete, the Central Processor proceeds to the next instruction (HLT) and stops.

The first Order Sequence program begins at location READ. All orders in the sequence refer to magnetic tape unit 1. The first order, the Search Key, order (SKF), causes the Processor to search magnetic tape unit 1 for a key word containing the binary equivalent of a decimal 5 in its *a* portion. As soon as the key word is located, control is transferred to the next order in sequence, namely SCWDP. If the key word is *not* located by the time an End-of-File (EOF) condition is reached on magnetic tape unit 1, No-Key (NKY) program interrupt occurs, the Order Sequence is disrupted, and Central Processor control is transferred to memory location 00001. Interrupt may be caused either when a Block-End marker (BLE) is reached or an End-Of-File marker (EOF) is reached by making the SK__ order suffix either B (block) or F (file), respectively.

If the key word is located, the order SCWDP is initiated. The SC portion of the order specifies alphanumeric read-in. The option W indicates that words (as opposed to blocks) are to be read. The D restricts the type of words read in to data words (as opposed to key words). Non-data words are merely skipped over. The final suffix P causes the Order Sequence to proceed to the next order as soon as the read operation has been completed. In this manner, 200 data words are read into memory starting with location DATA.

The order STD is a variation of the End Sequence order (ES). STD causes the Order Sequence to be terminated and the device and the Processor to be logically disconnected from the central computer. In the program illustrated, if the key is found, the Central Processor program ultimately halts at memory location 00752 (decimal). The second Order Sequence program (starting at location PRINT) is not executed.

If the key word is not found by the time an End-Of-File is reached, program interrupt occurs. Processor 1 and magnetic tape unit 1 are logically disconnected from the central computer. Central Processor control is transferred to memory location 00001. The second ORG (origin) pseudo-operation in the symbolic program causes the 94AP location counter to begin counting at location 00001. The Unconditional Transfer instruction (TRU) is placed in location 00001, so that when program interrupt occurs, the transfer instruction will transfer control of the Central Processor to location READ minus 2, which is the WAN instruction for initiating the Order Sequence program for error printout.

The second Start Order Sequence instruction, SS2, specifies that Processor 2 be used with the electric typewriter (FLX). It indicates that the Order Sequence program begins in location PRINT.

The first Order Sequence order at location PRINT is WA (Write Alphanumeric). Four alphanumeric words, beginning in location OUTPUT, are to be written on the electric typewriter. The first word contains the alphanumeric codes for two functions; lower case (octal 02) and carriage return (octal 04), to initialize the typewriter. The pseudo-operation BCI (binary-coded information) converts the following three words into the binary equivalents of the alphanumeric text in the variable field. Thus, the WA order causes the sentence "no key on mt1" to be typed.

As soon as WAN has been executed, the electric typewriter is logically disconnected. The Central Processor halts at memory location 00754.

94 AP CODING FORM

PROJECT NO.	PROGRAMMER	LOCATION	DATE	PAGE	PROGRAM IDENTIFICATION
				OF	

PROGRAM DESCRIPTION Search first file on MT1 for a key of decimal 5; when the key is found, read the following 200 signed data words into memory. If no key is found, print "no key on mt1" on typewriter. Ignore input parity errors.

PAGE	LINE	LOCATION	OPERATION	VARIABLE FIELD	REMARKS	PROGRAM IDENT.
			ORG	750		
		BEGIN	RWD	MT1	Rewind magnetic tape 1	
			SSSN1	READ, MT1	Initiate an Order Sequence; use Processor #1	
			SEN	*, 0, MT1	Central Processor waits for end of sequence	
			HLT		Central Processor halt	
			SS2	PRINT FLX	Initiate an Order Sequence; use Processor #2	
			HLT		Central Processor halt	
		READ	SKF	5	Search for a key of decimal 5 on magnetic tape 1	
			SCWDP	DATA, 200	Read in 200 data words and proceed in sequence	
			STD		End Order Sequence and disconnect	
		PRINT	WA	OUTPUT, 4	Write four alphanumeric words on typewriter	
			STD		End Order Sequence and disconnect	
		OUTPUT	OCT	0204	Lower case and carriage return	
			BCI	3, NOΔKEYΔONΔMT1	Printout text	
		DATA	BSS	200	Reserve 200 locations for data words	
			ORG	20	New origin to NKY interrupt location	
			TRU	READ-2	Transfer to PRINT Order Sequence	
			END	BEGIN		

Figure VII-4. OSM Read and Write Program

APPENDIX B

LIST OF POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

APPENDIX A
NUMBERING SYSTEMS

Appendix A contains a general description of numbering systems together with a representation of integers and fractions in the decimal, octal, and binary systems. The methods involved in converting from one of these systems to another are also described. Finally, the basic arithmetic operations of binary, and various other numbering systems are included.

GENERAL

To understand the arithmetic operations in the 9400 Computer, a basic knowledge of numbering systems is essential. This section discusses the decimal, octal, binary, and various other numbering systems.

DECIMAL SYSTEM

Numbering systems make use of the concept of positional notation. These systems have a specified number of permitted symbols which may be used to indicate entries in the system. For example, the decimal system contains the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, totalling 10 digits. The number of symbols permitted in a system is called the base or radix of the system. Therefore the decimal system has a base or radix of ten. The magnitude represented by a number depends not only on the number itself, but also upon its position with respect to the point separating the interger (whole number) and the fractional part of the number. For example, the decimal number 3.14 can be rearranged as 4.13, 1.43, and 3.41, all having different values due to the position of the digits 1, 3, and 4 with respect to the decimal point.

OCTAL SYSTEM

The octal system differs from the decimal system in that the radix of an octal number is eight instead of ten. This system contains the symbols 0, 1, 2, 3, 4, 5, 6, and 7, making a total of eight digits. The point used to determine the magnitude of the number is now called the octal point. All other rules which apply to the decimal system apply also to the octal system.

BINARY SYSTEM

The binary system uses the radix of two. Therefore, this system contains a total of two symbols, 0 and 1. The point used to determine the magnitude of the number is now called the binary point. All other rules which apply to decimal and octal systems also apply to the binary system.

OTHER SYSTEMS

Athough the binary system is used predominantly in computers, any radix may be used to form a valid numbering system, for example:

Ternary system — 0, 1, 2	(3 digits)
Quinary system — 0, 1, 2, 3, 4	(5 digits)
Duodecimal system — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, t, e	(12 digits)

INTEGERS AND FRACTIONS

This section discusses the integers and fractions associated with the decimal, octal, and binary numbering systems.

DECIMAL INTEGERS

Decimal integers are represented by a sequence of digits to the left of the decimal point. Decimal integers have the radix ten. This means that each digit has ten times the value of the next less significant digit to the right.

For example, the decimal number 532 can be interpreted as follows:

$$532 = 500 + 30 + 2$$
$$532 = 5 \times 10^2 + 3 \times 10^1 + 2 \times 10^0$$
$$532 = (5)\ (100) + (3)\ (10) + (2)\ (1).$$

The least significant digit of a decimal integer represents units, the second significant digit represents tens, the third significant digit represents hundreds, etc.

DECIMAL FRACTIONS

Decimal fractions are represented by a sequence of digits to the right of the decimal point. The first digit to the right of the decimal point is the most significant digit of a decimal fraction. This digit represents increments of 10^{-1} (1/10). The second, third and nth digits to the right of the decimal point represent increments of 10^{-2} (1/100), 10^{-3} (1/1000), and 10^{-n} ($1/10^n$), respectively. Note that each digit is ten times smaller than the preceding digit to the left.

For example, the decimal number 0.608 can be interpreted as follows:

$$0.608 = + 0.600 + 0.000 + 0.008$$
$$0.608 = 6 \times 10^{-1} + 0 \times 10^{-2} + 8 \times 10^{-3}$$
$$0.608 = (6)(1/10) + (0)(1/100) + (8)(1/1000).$$

The most significant digit of a decimal fraction represents tenths, the second significant digit represents hundredths, the third significant digit represents thousands, etc.

Combination of both decimal integers and decimal fractions is illustrated below:

$243.269 = 200 + 40 + 3 + 0.200 + 0.060 + 0.009$

$243.269 = 2\times10^{2} + 4\times10^{1} + 3\times10^{0} + 2\times10^{-1} + 6\times10^{-2} + 9\times10^{-3}$

$243.269 = (2)(100) + (4)(10) + (3)(1) + (2)(1/10) + (6)(1/100) + (9)(1/1000).$

OCTAL INTEGERS

Octal integers are represented by a sequence of digits to the left of the octal point. Octal integers have the radix eight. This means that each digit has eight times the value of the next less significant digit to the right. The least significant digit of an octal integer represents units as in the decimal system. The second significant digit represents eights, the third significant digit represents sixty fours, etc.

For example, the octal integer 306 can be interpreted as follows:

$306_8 = 3\times8^{2} + 0\times8^{1} + 6\times8^{0} = 198.$

OCTAL FRACTIONS

Octal fractions are represented by a sequence of digits to the right of the octal point. The first digit to the right of the octal point is the most significant digit of an octal fraction. This digit represents increments of 8^{-1} (1/8). The second, third, and nth digits to the right of the octal point represent increments of 8^{-2} (1/64), 8^{-3} (1/512), and 8^{-n} ($1/8^{n}$), respectively. Note that each digit is eight times smaller than the preceding digit to the left.

For example, the octal fraction 0.360 can be interpreted as follows:

$0.360 = (3)\ (8^{-1}) + (6)\ (8^{-2}) + (0)\ (8^{-3}) =$ decimal 0.46875.

BINARY INTEGERS

A binary integer has a radix of two and is a whole number composed of a sequence of bits (binary digits). Each bit can be either a 1 or 0. The least significant bit represents increments of 2^{0} (units), and is located immediately to the left of the binary point. The second significant bit to the left of the binary point represents increments of 2^{1} (twos). The third, fourth, and nth bits to the left of the binary point represent increments of 2^{2} (fours), 2^{3} (eights), and 2^{n}, respectively. Note that each bit is twice the value of the preceding bit to the right. A typical six-bit binary number is shown below.

$101011 = (1\times2^{5}) + 0\times2^{4}) + 1\times2^{3}) + (0\times2^{2}) + (1\times2^{1}) + (1\times2^{0})$

$101011 = (1)(32) + (0)(16) + (1)(8) + (0)(4) + (1)(2) + (1)(1).$

BINARY FRACTIONS

Binary fractions are represented by a sequence of bits to the right of the binary point. The first bit to the right of the binary point is the most significant bit of a binary fraction. This bit represents increments of 2^{-1} (1/2). The second, third and nth bits to the right of the binary point represent increments of 2^{-2} (1/4,), 2^{-3} (1/8), and 2^{-n} ($1/2^{n}$), respectively. Note that each bit is half the value of the preceding bit to the left. A typical 6-bit binary fraction is shown below:

$0.101011 = (1\times2^{-1}) + (0\times2^{-2}) + (1\times2^{-3}) + (0\times2^{-4}) + (1\times2^{-5}) + (1\times2^{-6})$

$0.101011 =)(1)(1/2) + (0)(1/4) + (1)(1/8) + (0)(1/16) + (1)(1/32) + (1)(1/64).$

CONVERSION BETWEEN SYSTEMS

Included herein are methods of converting to or from any one of the three numbering systems (i.e., decimal, octal, or binary).

DECIMAL TO OCTAL CONVERSION OF INTEGERS

To convert a decimal integer to an octal integer, divide the decimal integer by the radix of the octal system (8) repeatedly and take the remainders in reverse order to get the octal integer (i.e., the first remainder is the *least* significant digit). Thus to convert 347_{10} to the octal system:

$$\begin{array}{rl} 8\,\big|\,347 & \\ 8\,\big|\,43 & \text{remainder is } 3 \\ 8\,\big|\,5 & \text{remainder is } 3 \\ 0 & \text{remainder is } 5 \end{array} \Bigg\} \; 533_8$$

Therefore,

$$347_{10} = 533_8$$

OCTAL TO DECIMAL CONVERSION OF INTEGERS

To convert an octal integer to a decimal integer it is necessary to convert all the digits of the octal integer to their decimal equivalent and add. Therefore, to convert 533_8 to the decimal system:

$$\begin{array}{rl} 5\times8^{2} = (5)(64) & = 320 \\ 3\times8^{1} = (3)(8) & = 24 \\ 3\times8^{0} = (3)(1) & = 3 \\ \hline & 347_{10} \end{array}$$

Therefore,

$$533_8 = 347_{10}$$

DECIMAL TO OCTAL CONVERSION OF FRACTIONS

To convert octal fractions to decimal fractions, conmultiply repeatedly by the radix of the octal number (8) and take out the resulting integral parts in forward order. That is, the first integral number is the *most* significant digit. Thus, to convert 0.468750_{10} to the octal system:

$$\left.\begin{array}{ll} 8 \times 0.468750 \pm = 3.750000, & \text{integral part is } 3 \\ 8 \times 0.75 \quad \pm = 6.00, & \text{integral part is } 6 \\ 8 \times 0.00 \quad \pm = 0.00, & \text{integral part is } 0 \end{array}\right\} 0.360_8$$

Therefore,

$0.468750_{10} = 0.360_8$.

OCTAL TO DECIMAL CONVERSION OF FRACTIONS

To convert octal fractions to decimal fractions, convert the octal digits to their equivalent decimal digits and add. Therefore, to convert 0.360_8 to the decimal system:

$$\begin{array}{l} 3 \times 8^{-1} = (3)(1/8) \quad = 0.375 \\ 6 \times 8^{-2} = (6)(1/64) \quad = 0.09375 \\ 0 \times 8^{-3} = (0)(1/512) = 0.00000 \\ \hline 0.46875_{10} \end{array}$$

Therefore,

$0.360_8 = 0.46875_{10}$

DECIMAL TO BINARY CONVERSION OF INTEGERS

To convert a decimal integer to a binary integer divide the decimal integer by the radix of the binary system (2) repeatedly and take the remainder in reverse order to get the equivalent binary integer. Again, as in converting to octal, the first remainder is the *least* significant bit. Thus, to convert 43_{10} to the binary system:

$$\begin{array}{lll} 2 & 43 & \\ 2 & 21 & \text{remainder is } 1 \\ 2 & 10 & \text{remainder is } 1 \\ 2 & 5 & \text{remainder is } 0 \\ 2 & 2 & \text{remainder is } 1 \\ 2 & 1 & \text{remainder is } 0 \\ & 0 & \text{remainder is } 1 \end{array} \Bigg\} 101011_2$$

Therefore,

$43_{10} = 101011_2$.

BINARY TO DECIMAL CONVERSION OF INTEGERS

To convert a binary integer to a decimal integer it is necessary to convert all the bits of the binary integer to their decimal equivalents and add. Therefore, to convert 101011_2 to the decimal system:

$$\begin{array}{l} 1 \times 2^5 = (1)(32) = 32 \\ 0 \times 2^4 = (0)(16) = 0 \\ 1 \times 2^3 = (1)(8) \quad = 8 \\ 0 \times 2^2 = (0)(4) \quad = 0 \\ 1 \times 2^1 = (1)(2) \quad = 2 \\ 1 \times 2^0 = (1)(1) \quad = 1 \\ \hline 43 \end{array}$$

Therefore,

$101011_2 = 43_{10}$.

DECIMAL TO BINARY CONVERSION OF FRACTIONS

To convert a decimal fraction to a binary fraction, multiply repeatedly by the radix of the binary number (2) and take out the resulting integral parts in forward order. Thus, to convert 0.375_{10} to the binary system:

$$\left.\begin{array}{l} 2 \times 0.375 = 0.75, \text{ integral part is } 0 \\ 2 \times 0.75 \ = 1.50, \text{ integral part is } 1 \\ 2 \times 0.50 \ = 1.00, \text{ integral part is } 1 \\ 2 \times 0.00 \ = 0.00, \text{ integral part is } 0 \end{array}\right\} 0.0110_2$$

Therefore,

$0.375_{10} = 0.0110_2$.

BINARY TO DECIMAL CONVERSION OF FRACTIONS

To convert a binary fraction to a decimal fraction, convert the binary bits to their equivalent decimal digits and add. Therefore, to convert 0.011_2 to the decimal system:

$$\begin{array}{l} 0 \times 2^{-1} = (0)(1/2) = 0.000 \\ 1 \times 2^{-2} = (1)(1/4) = 0.250 \\ 1 \times 2^{-3} = (1)(1/8) = 0.125 \\ \hline 0.375_{10} \end{array}$$

Therefore,

$0.011_2 = 0.375_{10}$.

BINARY TO OCTAL CONVERSION

In the octal system the radix is eight, and there are eight possible values that each digit of an octal number can have, namely, 0 to 7. A three-bit binary integer also has eight possible values. This permits the simple conversion between three-bit binary numbers and single octal digits, as shown in Table A-1.

Binary	*Octal*
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table A-1. Binary to Octal Conversion

The radix of the binary number system is two. Therefore, each three-bit binary group has eight times the weight of the next three-bit group to the right ($2^3 = 8$). This means that direct conversion is possible between any binary number and the equivalent octal number.

The first step is to divide the binary number into groups of three bits, working left from the binary point. If a group of two bits is left over, the next bit at the left is considered to be zero. A zero added to the left of a number has no effect on the value of the number. Similarly, if only one bit is left over, the next two bits to the left are considered to be zeros. Thus, the binary number "11101111" can be converted to an equivalent octal number in the following manner:

011	101	111	(binary)
3	5	7	(octal)

BINARY TO OCTAL TO DECIMAL CONVERSION

Table A-2 gives the binary to octal to decimal conversion for the decimal numbers 0 through 22.

BASIC ARITHMETIC OPERATIONS AS APPLIED TO BINARY NUMBERS

The 9400 Computer storage devices store binary information. Therefore, all arithmetic operations in the computer are performed using binary numbers. The octal numbering system is used only to represent the binary system. This section describes the addition, subtraction, multiplication, and division of binary numbers.

BINARY ADDITION

Binary numbers, both intergers and fractions, are added in much the same way as ordinary decimal numbers. The computer never adds more than two numbers during each single step. Therefore, more than two binary numbers are added together by successively adding each number to the preceding subtotal.

Binary	*Octal*	*Decimal*
Radix = 2	*Radix = 8*	*Radix = 10*
Digits Used 0 and 1	*Digits Used 0 to 7*	*Digits Used 0 to 9*
00000	0000	0
00001	0001	1
00010	0002	2
00011	0003	3
00100	0004	4
00101	0005	5
00110	0006	6
00111	0007	7
01000	0010	8
01001	0011	9
01010	0012	10
01011	0013	11
01100	0014	12
01101	0015	13
01110	0016	14
01111	0017	15
10000	0020	16
10001	0021	17
10010	0022	18
10011	0023	19
10100	0024	20
10101	0025	21
10110	0026	22

Table A-2. Binary to Octal to Decimal Conversion (Decimal 0 to 22)

When two binary numbers are added together, the addition of each column produces a sum bit (i.e., 1 or 0) which appears in the result under the column being added. The addition of each column can also produce a "carry" bit which is applied to the next more significant column to the left. However, a carry does not occur always.

Every possible combination for adding two binary numbers is shown below:

Addend	0	1	0	1
Augend	1	0	0	1
Sum	1	1	0	0
Carry				1

For example, the addition of two binary numbers is shown below:

Addend	101011
Augend	110011
Sum (without carry)	011000
Carry in*	100011
Sum	1011110

*The carry-in is formed by the carry-out of the previous column.

The addition of binary fractions is performed in exactly the same way as for binary integers.

DIRECT SUBTRACTION

When two binary numbers are subtracted, the subtraction of each column results in a difference bit (1 or 0) below the columns being subtracted. In some cases, it is necessary to borrow a 1 to complete a subtraction. Since each digit in the minuend is twice the value of the next less significant digit, it is possible to "borrow" to complete a subtraction (i.e., a 1 in the second least significant digit is equivalent to two 1's in the least significant digit). This procedure is shown below:

	1
Minuend after borrow	01111
Minuend before borrow	10000
Subtrahend	1001
Difference	00111

For example, the subtraction of the two binary integers is shown below:

Minuend	110011
Subtrahend	11101
Difference	10110

The subtraction of binary fractions is performed in exactly the same way as for binary integers.

SUBTRACTION BY COMPLEMENTS (END-AROUND CARRY)

Instead of using direct subtraction, it is possible to subtract by adding the complement of the subtrahend to the minuend and then making certain adjustments. There are two complement methods, namely, the no-end-around carry and the end-around carry. Since the end-around-carry method is employed in the 9400 Computer, it is the method described below.

In the decimal system, the end-around-carry method is also called the 9's complement method, or one less than the 10's complement of a decimal number. The 10's complement of a decimal number is the difference between the number and the next larger power of 10 greater than that number. For example, the 10's complement of 24 is 76 or 100 minus 24, and the 10's complement of 333 is 677 or 1000 minus 333. The 9's complement of 24 is 75 since its 10's complement is 76. Also, the 9's complement of 333 is 666 since its 10's complement is 667.

To subtract 23 from 236, the 9's complement of 23 is added to 236 and then a 1 is added to the result. The addition of a 1 to the result is termed end-around carry. That is,

```
  236
+ 976  (9's complement of 23)
 1212  (the 1 in the extra column indicates a
        positive number)
    1  end-around carry
 + 213
```

If the extra column 1 were not present, it would be necessary to 9's complement the result and prefix a minus sign. For example, to subtract 723 from 236:

```
  236
  276  (9's complement of 723)
  512  (no extra-column 1 in this result)
 -487  (9's complement of result with minus sign
        indicated)
```

The end-around-carry method when used in the binary system is called the 1's complement method, or one less than the 2's complement of a binary number. It is the difference between the binary number and the next larger power of 2 greater than that number. For example, the 2's complement of 1100 is 1000 minus 1100 or 0100, and the 2's complement of 10111 is 100000 minus 10111 or 01001. The 1's complement of 1100 is 1111 (or 1 less than 10000) minus 1100 or 0011. It may now be observed that the 1's complement of a binary number is obtained immediately by changing the 1's to 0's and the 0's to 1's. Therefore, the 1's complement of 1100 is 0011.

To subtract the binary number 01111001 from 11101001, the 1's complement of 01111001 is added to 11101001 plus 1 (end-around carry). That is,

```
 11101001
 10000110  (1's complement of 01111001)
101101111  (the 1 in the extra column indicates a
            positive number)
        1  end-around carry
+01110000
```

If the extra column 1 were not present, and therefore no end-around carry, it would be necessary to 1's complement the result and prefix a minus sign. For example, to subtract 01101110 from 0001111:

```
  00001111
+ 10010001  (1's complement of 01101110)
-----------
  10100000  (no extra-column 1 in this result)
-01011111  (1's complement of result with minus
           sign indicated)
```

DIRECT MULTIPLICATION

The rules for direct multiplication in the decimal system are also applicable in the binary system. Every possible combination for multiplying two numbers is shown below:

Multiplicand	1	0	1	0
Multiplier	1	0	0	1
Product	1	0	0	0

For example, the direct multiplication of two binary numbers is shown below:

```
    101101   Multiplicand
    100011   Multiplier
    ------
    101101
   101101
  101101
  ------------
  11000100111   Product
```

MULTIPLICATION BY REPEATED ADDITION

Instead of using direct multiplication, computers multiply by repeated binary additions. The 9400 Computer performs a binary multiplication in the following way. The multiplicand is added in the accumulator to zero if the first digit of the multiplier is a 1. If the first digit of the multiplier is a 0, no addition occurs. As each digit of the multiplier is considered, the accumulator is shifted to the right and then added to the multiplicand when the multiplier digit is 1. For example, to multiply by repeated additions the binary numbers 101101 (multiplicand) and 100011 (multiplier):

Accumulator		*Multiplier*
000000	Accumulator	100011
101101	Multiplicand	
101101	Addition of Multiplicand to Accumulator	
101101	Accumulator shifted one place to the right	100010
101101	Multiplicand	
10000111	Addition of Multiplicand to Accumulator	
10000111	Accumulator shifted four places to the right	100000
101101	Multiplicand	
11000100111	Addition of Multiplicand to Accumulator	000000

Thus, the product of 101101 and 100011 is equal to 11000100111, and is a positive number since both the multiplier and multiplicand are both positive.

DIRECT DIVISION

The rules for direct division in the decimal system are also applicable in the binary system. Direct binary division is simplified by having only two variables. Whenever the divisor is larger than the dividend, a 0 is placed in the quotient; otherwise, a 1 is placed in the quotient.

For example, to divide 10000101 (dividend) by 1110 (divisor):

```
                1001.1
          ------------
1110   10000101.0
         1110
         ----
          10101
          1110
          ----
           1110
           1110
           ----
           0000
```

DIVISION BY REPEATED SUBTRACTION

Instead of using direct division, computers divide by repeated subtractions. The 9400 Computer performs a binary subtraction in the following way. First, the dividend is examined to determine whether or not it is smaller than the divisor. If it is smaller, division is permissible (i.e., the quotient will be fractional and,

in the 9400 Computer, is represented by a 36-bit binary fraction). If division is permissible, the quotient is formed on a bit-by-bit basis, as described below.

The binary point which precedes the first bit of the dividend and the first bit of the divisor can be ignored. As a result, the dividend and divisor can be thought of as 36-bit binary integers rather than fractions. The divisor, which was a larger binary fraction than the dividend, is now a larger binary integer than the dividend. This insures that the quotient will be a fraction. Since the quotient is a fraction, the first bit to be determined is the bit immediately to the right of the binary point, since all bits to the left of the binary point are 0's. The bit is determined by multiplying the dividend by 010 (binary) or 2 (decimal). This is equivalent to shifting the dividend one bit to the left.

The multiplied (shifted) dividend is examined to see if it is now larger than the divisor. If it is larger, a 1 is put in the first bit position of the quotient and the divisor is subtracted from the "new" dividend. If the new dividend is not larger than the divisor, a 0 is placed in the first bit position of the quotient and the divisor is not subtracted. The difference (if a subtraction was performed) or the shifted dividend (if a subtraction was not performed) is then shifted one place to the left, and the same procedure used to determine the first bit of the quotient is followed again to determine the second bit of the quotient.

This procedure is continued for 36 times. The quotient will then contain 36 bits. The number that remains after the 36th (last) subtraction or nonsubtraction is the remainder of the division.

For example, to divide 0.011011 (dividend) by 0.101101 (divisor) using 6-bit numbers instead of 36-bit numbers for purposes of convenience, the following procedure is used. (Division is permissible since the dividend is smaller than the divisor).

0.011011	Dividend No. 1.
0.110110	Dividend shifted one place to the left (dividend No. 2).
Q = 0.1 — 0.1011101	Divisor smaller than dividend No. 2, therefore, subtraction is performed.
0.001001	Difference (dividend No. 3).
0.010010	Difference (dividend No. 3) shifted one place to the left (dividend No. 4).
Q = 0.10 — 0.101101	Divisor larger than dividend No. 4, therefore, no subtraction is performed.
0.100100	Difference (dividend No. 4) shifted one place to the left (dividend No. 5).
Q = 0.100 — 0.101101	Divisor larger than dividend No. 5, therefore, no subtraction is performed.
1.001000	Difference (dividend No. 5) shifted one place to the left (dividend No. 6).
Q = 0.1001 — 0.101101	Divisor smaller than dividend No. 6, therefore, subtraction is performed.
0.011011	Difference (dividend No. 7).
0.110110	Difference (dividend No. 7) shifted one place to the left (dividend No. 8).
Q = 0.10011 — 0.101101	Divisor smaller than dividend No. 8, therefore, subtraction is performed.
0.001001	Difference (dividend No. 9).
0.010010	Difference (dividend No. 9) shifted one place to the left (dividend No. 10).
Q = 0.100110 — 0.101101	Divisor larger than dividend No. 10, therefore, no subtraction is performed.

Q = Quotient = 0.101101

R = Remainder = dividend No. 10 = 0.010010

The quotient is a positive number since the sign of the dividend and the sign of the divisor are alike. If they were not alike, the quotient would have been negative. The sign of the remainder is the sign of the dividend and is positive in the above example.

APPENDIX C

LIST OF OCTAL-DECIMAL, DECIMAL-OCTAL CONVERSION

0000 to 0777 (Octal)	0000 to 0511 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal)	0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

2000	1024
to	to
2777	1535
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

3000	1536
to	to
3777	2047
(Octal)	(Decimal)

4000 to 4777 (Octal)	2048 to 2559 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal)	2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

6000	3072
to	to
6777	3583
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

7000	3584
to	to
7777	4095
(Octal)	(Decimal)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

APPENDIX D

LIST OF ADDRESSABLE REGISTERS

Code	*Address**	*Name*
	00000-77737	Memory Storage locations
ir1	77741	Index Register 1
ir2	77742	Index Register 2
ir3	77743	Index Register 3
ir4	77744	Index Register 4
acc	77750	Accumulator
qrg	77751	Q-Register
brg	77752	B-Register
pct	77753	Program Counter
pcs	77754	Program Counter Store
cio	77755**	Instruction register of In-Out Processor receiving the input order
afr	77756	Alarm Switch Register (composite of all alarm switches)
wsr	77760	Word Switch Register
rar	77761	Real-Time Address Register
ror	77762	Real-Time Output Register
pi1	77770	Instruction register of first In-Out Processor
pi2	77771	Instruction register of second In-Out Processor
pi3	77772	Instruction register of third In-Out Processor
pi4	77773	Instruction register of fourth In-Out Processor
	——***	Nonexistent register

*In octal.

**When used as the address of an input instruction, it tells the In-Out Processor to store the contents of its buffer register in its instruction word register.

***If a nonexistent register is addressed, it will be interpreted as a location which contains all 0's.

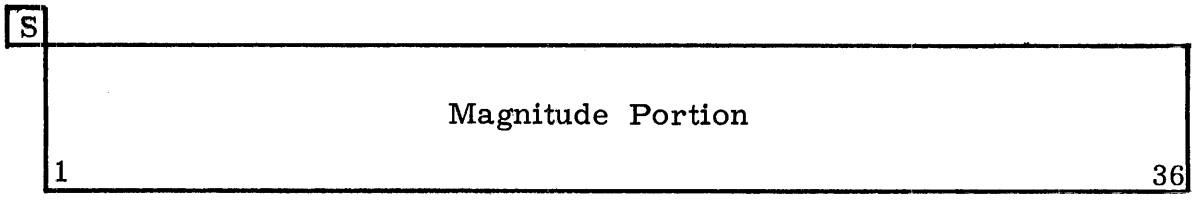
APPENDIX E

LIST OF PROGRAM INTERRUPT ACTIVITY SWITCHES AND LOCATIONS

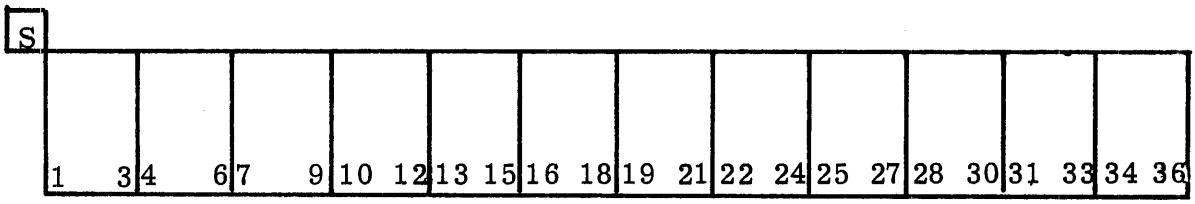
Central Processor Interrupts ----	Location 0 (Group 0)
Transfer Order Trapped -	TOT
Overflow or Underflow	API (OA + UA)
Real Rime Input Control Character	RCI (RINC)
Processor Busy	PRBI (PRBA)
Device Busy	DVBI (DVBA)
Input Output Processor i Interrupt --	**Location i (i = 1, 2, 3, 4)**
Functional Control Characters	FCI $(FCC)^i$
End of Tape	ETI $(ETA)^i$
End of File	EFI $(EOF)^i$
End of File in Order Sequence	EPR^i
No Key in Order Sequency	NKR^i
End Order Interrupt in Order Sequence	EOR^i

APPENDIX F

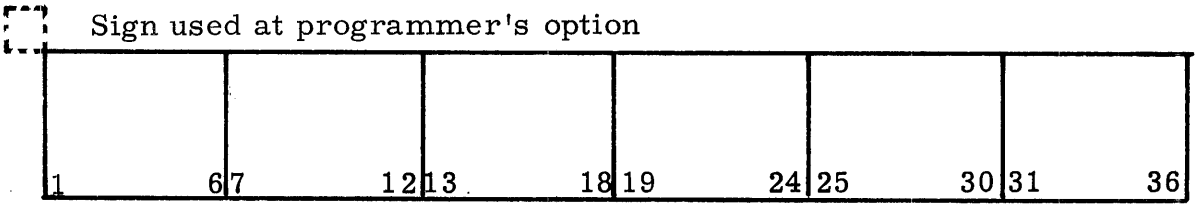
9400 WORD FORMATS



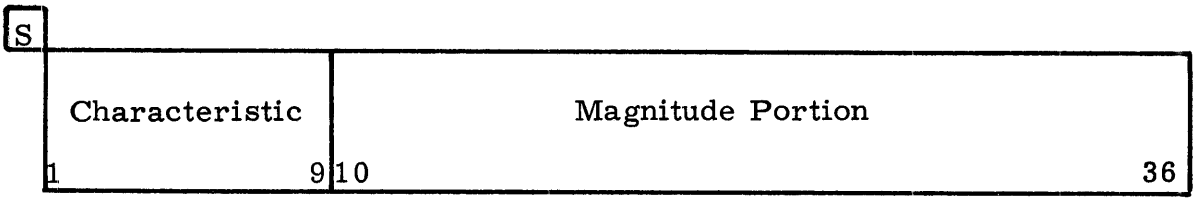
Numeric (Conventional) Data Word



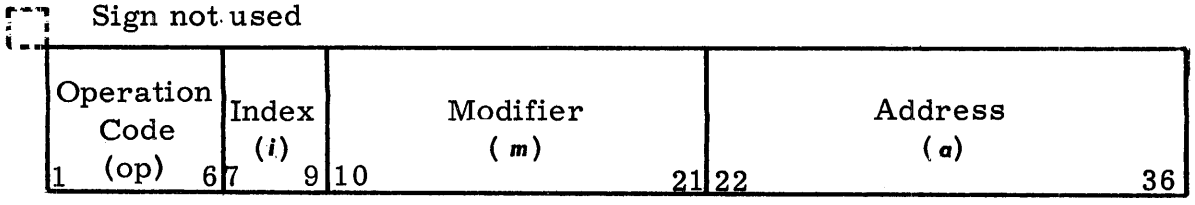
Octal Data Word



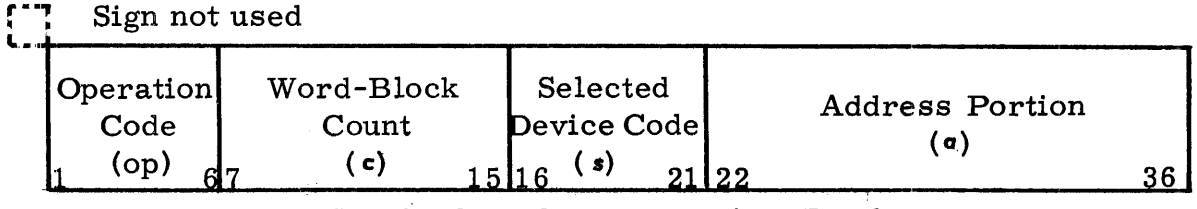
Alphanumeric Data Word



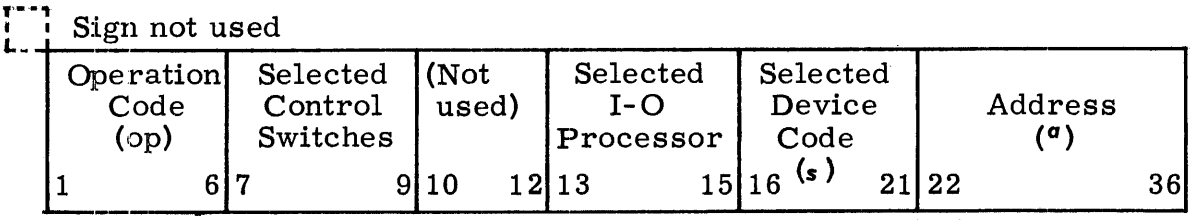
Floating-Point Data Word



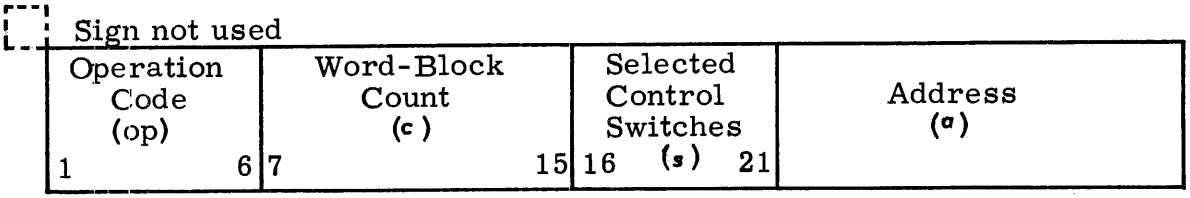
Central Processor Instruction Word



Standard In-Out Instruction Word



Order Sequence Mode Instruction Word (SS)



Order Sequence Mode Order Word

APPENDIX G

LIST OF ALPHANUMERIC CODES

CODE 543 210	OCTAL Interpretation	CHARACTER	CODE 543 210	OCTAL Interpretation	CHARACTER	CODE 543 210	OCTAL Interpretation	CHARACTER
000 000	00	*Master space ≈	010 101	25	P	101 010	52	"
000 001	01	*Upper Case ~	010 110	26	Q	101 011	53	:
000 010	02	*Lower Case ▼/▲	010 111	27	R	101 100	54	?
000 011	03	*Tab ◊	011 000	30	S	101 101	55	!
000 100	04	*Car. Ret. ⅄	011 001	31	T	101 110	56	,
000 101	05	*Space Δ	011 010	32	U	101 111	57	†
000 110	06	A	011 011	33	V	110 000	60	0
000 111	07	B	011 100	34	W	110 001	61	1
001 000	10	C	011 101	35	X	110 010	62	2
001 001	11	D	011 110	36	Y	110 011	63	3
001 010	12	E	011 111	37	Z	110 100	64	4
001 011	13	F	100 000	40	)	110 101	65	5
001 100	14	G	100 001	41	-	110 110	66	6
001 101	15	H	100 010	42	+	110 111	67	7
001 110	16	I	100 011	43	<	111 000	70	8
001 111	17	J	100 100	44	=	111 001	71	9
010 000	20	K	100 101	45	>	111 010	72	'
010 001	21	L	100 110	46	_	111 011	73	;
010 010	22	M	100 111	47	$	111 100	74	/
010 011	23	N	101 000	50	*	111 101	75	.
010 100	24	O	101 001	51	(	111 110	76	*Special □
						111 111	77	*Idle ←

*Special control characters for typewriter and line printer. Printed only by line printer on verbatim print-out mode.

For the typewriter: Symbols 40-56 are upper case; symbols 60-75 are lower case.

NOTE: The 6-bit alphanumeric codes number from 000 000 to 111 111 and correspond in sequence starting with typewriter control characters, alphabetic characters (A through Z), mathematical symbols, arabic numerals (0 through 9) punctuation marks, and special characters. All bit combinations are included and can be represented as printable characters.

† Stop code

APPENDIX H

LIST OF 9400 INSTRUCTIONS AND ORDERS

A. NUMERICAL INDEX

Octal Code	Oper. Code		Operation	Page
	00	HLT	Halt	3-24
	01	RPT	Repeat	3-27
	02	LGM	Logical Multiply	3-27
	03	LGA	Logical Add	3-26
	04	LGN	Logical Negative	3-27
	05	SEN	Sense	3-23
	06	SNS	Sense and Set	3-23
	07	SNR	Sense and Reset	3-23
	10	CLA	Clear and Add	3-15
	11	CAM	Clear and Add Magnitude	3-15
	12	ADD	Add	3-16
	13	ADM	Add Magnitude	3-16
	14	CLS	Clear and Subtract	3-15
	15	CSM	Clear and Subtract Magnitude	3-15
	16	SUB	Subtract	3-16
	17	SBM	Subtract Magnitude	3-16
	20	MLY	Multiply	3-16
	21	MLR	Multiply and Round	3-16
	22	DVD	Divide	3-17
	23	DVL	Divide Long	3-17
	24	ADB	Add Modifier	3-25
	25	SBB	Subtract Modifier	3-25
	26	FLM	Floating Point Multiply	3-21
	27	FLD	Floating Point Divide	3-21
	30	SHL	Shift Left	3-17
	31	SLL	Shift Left Long	3-18
	32	SHR	Shift Right	3-17
	33	SRL	Shift Right Long	3-17
	34	CYS	Cycle Short	3-26
	35	CYL	Cycle Long	3-26
	36			
	37	NRM	Normalize	3-18
	40	TRU	Unconditional Transfer	3-22
	41	TRL	Load pcs and Transfer	3-23
	42	TRS	Transfer to pcs	3-23
	43	TRX	Transfer on Index	3-24
	44	TRP	Transfer on Positive Accumulator	3-22
	45	TRZ	Transfer on Zero Accumulator	3-22
	46	TRN	Transfer on Negative Accumulator	3-23
	47	TRC	Compare	3-24
	50	STR	Store	3-22
	51	LOD	Load	3-22
	52	MOV	Move	3-22
	53	LXS	Load Indices	3-24
	54	RPA	Replace Address	3-25
	55	MSK	Replace Through Mask	3-26
	56	FLA	Floating Point Add	3-20
	57	FAM	Floating Point Add Magnitude	3-20
	60	FLS	Floating Point Subtract	3-21
	61	FSM	Floating Point Subtract Magnitude	3-21
	62	SS	Start Order Sequence	4-9
	63	PR	In-Out Processor Sense and Reset Order	4-15
		PS	In-Out Processor Error Sense Order	4-15
		PT	In-Out Processor Unconditional Transfer Order	4-14
		ST	Sequence Terminate Order	4-15
	64	SK	Search Key Order	4-13
	65	WK	Write Key Order	4-12
	66	SKP	Skip	4-8
		SP	Space Order	4-14
	67	BSP	Backspace	4-8
		BS	Backspace Order	4-14
	70	RAN	Read Alphanumeric	4-7
		SC	Scatter Read Order	4-12
	71	RRV	Read Reverse	4-8
		RR	Read Reverse Order	4-13
	72	ROK	Read Octal Instruction	4-8
	74	WAN	Write Alphanumeric	4-6
		GW	Gather Write Order	4-11
	75	WWA	Rewrite Alphanumeric	4-7
	76	WOK	Write Octal	4-7
		WO	Write Octal Order	4-12
	77	RWD	Rewind	4-8
		RW	Rewind Order	4-14

B. ALPHABETIC INDEX

APPENDIX I

SUMMARY OF OPERATION CODES

A. FIXED POINT ARITHMETIC INSTRUCTIONS

Instruction	acc	qrg	brg	Time	Over-flow	Index-able	Notes
ADD 12	C(acc) + C(a) Sum		$acc_s = a_s$; C(a) --> brg	8 μs	P	Yes	
			$acc_s \neq a_s$ { $\lvert C(acc)\rvert \geq \lvert C(a)\rvert$; C(a)' --> brg; $brg_s = a_s$ $\lvert C(acc)\rvert < \lvert C(a)\rvert$; 0 --> brg				
ADM 13	C(acc) + |C(a)| Sum		acc_s = +; C(a) --> brg	8 μs	P	Yes	
			acc_s = - { $\lvert C(acc)\rvert \geq \lvert C(a)\rvert$; C(a)' --> brg; $brg_s = a_s$ $\lvert C(acc)\rvert < \lvert C(a)\rvert$; 0 --> brg				
CAM 11	+ |C(a)|			8 μs		Yes	
CLA 10	+ C(a)			8 μs		Yes	
CLS 14	- C(a)			8 μs		Yes	
CSM 15	- |C(a)|			8 μs		Yes	
DVD 22	Remainder	C(acc)/C(a) Quotient	C(a)	44 μs	P	Yes	(9 μs if overflow)
DVL 23	Remainder	C(acc, qrg)/C(a) Quotient	C(a)	44 μs	P	Yes	(9 μs if overflow)
MLR 21	C(acc) x C(a) High Order Rounded product	C(acc) x C(a) Low order Product	qrg_1 = 0; C(a) -> brg qrg_1 = 1; 0 -> brg	43 μs		Yes	
MLY 20	C(acc) x C(a) High order Product	C(acc) x C(a) Low order Product	C(a)	43 μs		Yes	
NRM 37	High order zeros shifted off; low order zeros inserted			$9 + \frac{n}{2}$ μs		Yes	(n = no. of shifts) If all "0's", n = 36 C(a) = n x 2^{-36}
SBM 17	C(acc) - |C(a)| Difference		acc_s = - ; C(a) --> brg	8 μs	P	Yes	
			acc_s = + { $\lvert C(acc)\rvert \geq \lvert C(a)\rvert$; C(a)' -> brg, $brg_s = a_s$ $\lvert C(acc)\rvert < \lvert C(a)\rvert$; 0 -> brg				
SHL 30	C(acc) Shifted left n = a mod 128			See Note	P	Yes	Sign bit not shifted n $\leq$ 9; 8 μs n > 9; $4 + \frac{n}{2}$ μs
SHR 32	C(acc) Shifted right n = a mod 128			See Note		Yes	Sign bit not shifted n $\leq$ 14; 8 μs n > 14; $1 + \frac{n}{2}$ μs
SLL 31	C(acc, qrg) Shifted left n = a mod 128	(See acc)		See Note	P	Yes	Sign bits not shifted n $\leq$ 9; 8 μs n > 9; $4 + \frac{n}{2}$ μs
SRL 33	C(acc, qrg) Shifted right n = a mod 128 places	(See acc)		See Note		Yes	Sign bits not shifted n $\leq$ 14; 8 μs n > 14; $\frac{n}{2} + 1$ μs
SUB 16	C(acc) - C(a) Difference		$acc_s = a_s$ { $\lvert C(acc)\rvert \geq \lvert C(a)\rvert$; C(a)' -> brg, $brg_s = a_s$ $\lvert C(acc)\rvert < \lvert C(a)\rvert$; 0 -> brg				
			$acc_s \neq a_s$; C(a) --> brg	8 μs	P	Yes	

B. FLOATING POINT ARITHMETIC INSTRUCTIONS

Instruction	acc	qrg	brg	Time	OA or UA **	Index-able	Notes
FAM 57	F(acc) +|F(a)| High order bits	F(acc) +|F(a)| Low order bits	0 —> brg	12 to 46 μs	P	Yes	Overflow is possible
FLA 56	F(acc) + F(a) High order bits	F(acc) + F(a) Low order bits	0 —> brg	12 to 46 μs	P	Yes	Overflow or underflow is possible
FLM 26	F(a) x F(acc) High order bits	F(a) x F(acc) Low order bits	F(a)	9 or 37 μs	P	Yes	Overlow or underflow is possible
FLS 60	F(acc) - F(a) High order bits	F(acc) - F(a) Low order bits	0 —> brg	12 to 46 μs	P	Yes	Overflow or underflow is possible
FSM 61	F(acc) -|F(a)| High order bits	F(acc) -|F(a)| Low order bits	0 —> brg	12 to 46 μs	P	Yes	Underflow is possible
FLD 27	F(acc) / F(a) Remainder	F(acc) / F(a) Quotient and Characteristic	F(a)	9 or 41 μs	P	Yes	Overflow or underflow is possible

* Trapping possible
** See Instruction Repertoire for detailed description
P = Possible
PND = Possible but not detected

Instruction	acc	qrg	brg	Time	Over-flow	Index-able	Notes
ADB 24	C(a) + m sum	C(acc)	a_{sn} = + ; m -> brg a_{sn} = - { |C(a)| ≥ m ; m -> brg, brg_{sn} = + |C(a)| < m ; 0 -> brg	13 μs	PND	No	After operation: C(iri) = C(a) + m and C(a) = C(a) + m
CYL 35	acc, qrg cycled left n = (a mod 128) places	(See acc)		See Note		Yes	Sign bits cycle n ≤ 14; 8 μs n > 14; $\frac{n}{2}$ + 1 μs
CYS 34	acc cycled left n = (a mod 128) places			See Note		Yes	Sign bit does not cycle n ≤ 14; 8 μs n > 14; $\frac{n}{2}$ + 1 μs
HLT 00				8 μs		No	Stops central processor
LGA 03	Logical Sum C(a) + C(acc)		C(a)	8 μs		Yes	Signs are logically added
LGM 02	Logical Product C(acc) x C(a)		C(a)	8 μs		Yes	Signs are logically multiplied
LGN 04	$C(a)'_{s,\ 1-37}$			8 μs		Yes	Sign complemented
LOD 51			C(a)	9 μs		Yes	C(a) -> Addressable Register, not memory location, specified by m
LXS 53				8 μs		No	C(iri) = a C(ir(i + 1)) = m
MOV 52		On RPT-MOV only a + (# of repeats - 1) (modifier of RPT incl)	C(a)	13 μs		No	C(a) -> i - m, any Addressable Register or memory location
MSK 55			C(acc)	11 μs		Yes	$C(qrg)_i$ = 1; $C(acc)_i$ -> $C(a)_i$ $C(qrg)_i$ = 0; $C(a)_i$ is unchanged For all i = sn, 1 - 37
RPA 54			C(acc)	11 μs		Yes	C(a): $C(a)_{sn,\ 1-21}$ $C(acc)_{22-36}$
RPT 01				8 μs		Yes	m -> ir4, a + C(iri) -> ir3 See instruction for detailed description
SBB 25	C(a) - m Difference	C(acc)	a_{sn} = + { |1C(a)|1 ≥ m; (m)' -> brg, brg_{sn} = + |1C(a)|1 < m; 0 -> brg a_{sn} = -; m -> brg	13 μs	PND	No	After operation: C(iri) = C(a) - m and C(a) = C(a) - m
SEN 05 *			TRA = 1; C(PCT) --> brg	8 μs		Yes	C(m) = 1; a --> PCT C(m) = 0; C(PCT) + 1 --> PCT
SNR 07 *			TRA = 1, C(PCT) --> brg	8 μs		Yes	C(m) = 1; 0 -> C(m), a --> PCT C(m) = 0; C(PCT) + 1 --> PCT
SNS 06 *			TRA = 1, C(PCT) --> brg	8 μs		Yes	C(m) = 0; i --> C(m), a --> PCT C(m) -1; C(PCT) + 1 --> PCT
STR 50				8 μs		Yes	C(acc) --> a
TRC 47		C(acc)	a	11 μs		Yes	C(acc) < C(a); C(PCT) + 1 --> PCT C(acc) > C(a); C(PCT) + 2 --> PCT C(acc) = C(a); C(PCT) + 3 --> PCT
TRL 41 *			TRA = 1, C(PCT) -->brg	8 μs		No	C(PCT) + 1 --> pcs a --> PCT m --> iri
TRN 46 *			TRA = 1, C(PCT) --> brg	8 μs		Yes	acc_s = + ; (C(PCT) + 1 --> PCT acc_s = ; a --> PCT
TRP 44 *			TRA = 1; C(PCT) --> brg	8 μs		Yes	acc_s = - ; C(PCT) + 1 --> PCT acc_s = + ; a --> PCT
TRS 42 *			TRA = 1; C(PCT) --> brg	8 μs		No	pcs --> PCT
TRU 40 *			m_{21} = 0, TRA = 1; C(PC) --> brg Otherwise; brg unchanged	8 μs		Yes	m_{21} = 0, TRA = 1; 2 --> PCT Otherwise; a --> PCT
TRX 43 *			TRA = 1, C(PCT) --> brg TRA = 0, a -> brg	11 μs		No	C(ir(i + 1)) > 1; a --> PCT { C(ir(i +1)) - 1 --> ir (i + 1) C(iri)+m --> iri C(ir(i - 1)) ≤ 1; { C(PCT) + 1 --> PCT 0 --> ir (i + 1) Time - μs
TRZ 45 *			TRA = 1, C(PCT) --> brg	8 μs		Yes	$C(acc)_{1-36}$ = 0; a --> PCT

* Trapping possible

P = Possible

PND = Possible but not detected

D. I/O SINGLE INSTRUCTIONS

Instruction	ISN	Word Format	Amount of Input-Output	Notes
BSP 67	No effect	Non-transmitting	Up to 255 blocks inclusive if $C(\underline{c})_7 = 0$ Up to 255 files inclusive if $C(\underline{c})_7 = 1$	Backspaces c blocks or files. Tape stops at the beginning of the block containing the EOF when backspacing files.
RAN 70	1	Sign bit plus six 6-bit characters occupying bits 1-36 of the memory word	Up to 255 words inclusive if $C(\underline{c})_7 = 0$	Reads c words or blocks. Terminates immediately if EOF is detected.
	0	Six 6-bit characters. Sign Position remains zero.	Up to 255 blocks inclusive if $C(\underline{c})_7 = 1$	
ROK 72	No effect	13 octal characters Sign + 12 data characters	Up to 511 words inclusive	NOT APPLICABLE TO MAGNETIC TAPE. Sign automatically interpreted.
RRV 71	1	Same as RAN Same as RAN	Same as RAN	Same as RAN except tape moves in reverse direction. Words stored in decreasing memory locations in reverse order. Results in same storage as RAN.
RWD 77	No effect	Non-transmitting		Rewinds tape unit s.
SKP 66	No effect	Non-transmitting	Up to 255 blocks inclusive if $C(\underline{c})_7 = 0$ Up to 255 files inclusive if $C(\underline{c})_7 = 1$	Skips c blocks or files.
SS 62	No effect	Non-transmitting		Initiates sequence of orders by I/O Processor. SETS ISN, if $C(\underline{c})_7 = 1$, SETS NHC if $C(\underline{c})_8 = 1$ SETS TPE if $C(\underline{c})_9 = 1$. Specifies I/O Processor to execute sequence.
WAN 74	1	Seven 6-bit characters Sign + six data characters Six 6-bit characters Sign not interpreted	Up to 511 words inclusive	Writes BLS, c words, BLE. If WEF is SET, writes EOF instead of BLE.
WOK 76	No effect	13 octal characters Sign + 12 data characters converted to sign bit and 36-bit memory word	Up to 511 words inclusive	NOT APPLICABLE TO MAGNETIC TAPE. Sign automatically interpreted.
WWA 75	1	Same as WAN Same as WAN	Up to 511 words inclusive	Must write only the same number of words plus or minus one as were in original block.

E. I/O ORDER SEQUENCE ORDERS

	ISN	Word Format	Amount of Input-Output	Notes
BS 67	No effect	Non-transmitting	Same as BSP Instruction	Same as BSP instruction.
GW 74	1	Seven 6-bit charac. ters Sign + 6 data bits	Up to 511 words inclusive	Writes BLS mark only if preceding order terminated with BLE or EOF. Writes BLE or EOF after $\underline{c}$ words if specified by the control bits in the order.
	0	Six 6-bit characters Sign not interpreted		
PR 63	No effect	Non-transmitting		Senses I/O Processor error controls. If SET transfers I/O Processor control to $\underline{a}$ within sequence.
PS 63	No effect	Non-transmitting		Senses I/O Processor error controls. If SET transfers I/O Processor control to $\underline{a}$ within sequence. PR RESETS error controls.
PT 68	No effect	Non-transmitting		Causes an unconditional transfer of I/O Processor control to location $\underline{a}$ within the sequence.
RO 71	No effect	13 octal characters the first of which occupies the sign position. Remaining 12 occupy binary bits 1-36 of memory.	Up to 511 words inclusive	NOT APPLICABLE TO MAGNETIC TAPE. Sign automatically interpreted.
RR 71	1	Same as RRV	Up to 255 words if $C(\underline{c})_7 = 0$	Same as RRV instruction.
	0	Same as RRV	Up to 255 blocks if $C(\underline{c})_7 = 1$	
RW 77	No effect	Non-transmitting		Rewinds magnetic tape specified by SS.
SC 70	1	Sign bit plus six 6-bit characters occupying bits 1-36 of the memory word	Up to 255 words if $C(\underline{c})_7$ and $C(\underline{s})_{16} = 0$ Up to 255 blocks if $C(\underline{c})_7 = 1$ and $C(\underline{s})_{16} = 0$	Reads c words or blocks. Options described under order
	0	Six 6-bit characters occupying bits 1-36 of the memory word	Words to next KEY if $C(\underline{c})_7 = 0$ and $C(\underline{s})_{16} = 0$	
		Sign position remains zero	Up to 255 words or to BLE which is first if $C(\underline{s})_7 = 0$ and $C(\underline{s}) = 1$	
SK 64	No effect	Non-transmitting		Searches magnetic tape for Key corresponding to $\underline{a}_{25-36}$ of the order. NKY interrupt if Key not found before BLE is detected if $C(\underline{s})_{18} = 0$, or before EOF is detected if $C(\underline{s})_{18} = 1$.
SP 66	No effect	Non-transmitting	Same as SKP instruction	Same as SKP instruction.
ST	No effect	Non-transmitting		Causes a disconnect of I/O Processor.
WK	1	Sign character followed by a six bit Key control character 3 non interpreted characters and 2 six bit characters making up a 12 bit key	One Key word	Key word specified by $\underline{a}_{25-36}$ of the WK order written on magnetic tape. Control transferred to next order - always GW.
	0	Same as above but sign not written		
WO76	No effect	13 octal characters sign + 12 data characters	Up to 511 words inclusive	NOT APPLICABLE TO MAGNETIC TAPE. Sign automatically interpreted.
WW	1	Same as GW	Same as GW	Same as GW but must write only the same number of words (plus or minus one) as were contained in original block.
	0	Same as GW		

APPENDIX J

LIST OF PSEUDO-OPERATIONS ACCEPTABLE TO THE 9400 PROGRAM (94AP)

GENERAL

In the following description, a pre-defined expression is an arithmetic expression in which all the symbols used have been previously defined.

ORIGIN SPECIFICATION (ORG)

In both passes the location counter is set to the value of the predefined expression appearing in the variable field. In the second pass it causes waiting output to be punched and a new loading origin set.

BLOCK STARTED BY SYMBOL (BSS)

The instruction BSS N reserves a block of storage extending from the current value of the location counter to its value plus N-1. N must be a predefined expression. If a symbolic location is present on the card it will be assigned the current value of the location counter, corresponding to the first word of the block reserved. During the second pass it will cause the waiting binary output to be punched. Finally the location counter is stepped by N.

BLOCK ENDED BY SYMBOL (BES)

This operation behaves the same as BSS except that the symbol associated with it is assigned to the next cell after the reserved block.

EQUAL (EQU)

The symbol appearing in the location field is assigned the value of the predefined expression appearing in the variable field. However, the symbol will be entered as an absolute quantity.

SYNONYM (SYN)

The same as EQU except that the symbol will be treated as a relocatable quantity.

HEADING (HED)

The first character in the variable field will be used as the heading character. Heading characters allow two or more program segments to contain the same symbols. The heading card supplies to the assembler a single character, any alphabetic or numeric character is permissible. Each symbol in the program following the HED pseudo-operation is prefixed by this character except when a special indication to cancel the prefixing operation is given or the symbol contains six characters. A new heading pseudo-operation will replace the heading character. The assembler normally has no heading character (i.e., master space). To restore the heading to this condition all one needs to do is to supply a HED card with a blank variable field.

It is sometimes necessary to cross-reference between the individual segments of the program. To accomplish this, such references must be written in the following way: let H be a heading character and K be the symbol in the block headed by H to which reference is to be made. To refer to K in a part of the program not headed by H but by, say J, write H$K in the variable field expression rather than K. The special character $ indicates to the assembler that K is to be prefixed by H instead of by the prefix J given on the last heading card.

To refer to a symbol in an unheaded segment of the program from a headed section simply write $K.

DEFINE (DEF)

This operation is similar to EQU except that the symbol is assigned the absolute value of the octal integer appearing in the variable field. Symbols defined by DEF may not be relocated.

WRITE SYMBOL TABLE (WST)

The operation WST M, N will cause the assembler, at the end of the first pass, to write the symbol table as the Nth table on tape M. (M is a pre-defined expression.)

A blank variable field will cause the symbol table to be punched on-line.

LIST SYMBOL TABLE (LST)

A PST card placed anywhere in the symbolic deck causes the symbol table to be printed at the end of the assembly listing.

FINAL SYMBOL TABLE (FST)

A FST card placed anywhere in the symbolic deck causes the symbol table to be punched at the end of the assembly. This table will include those symbols defined by the USO operation and the values assigned to the literal expressions. Otherwise this pseudo-op is the same as WST.

REPLACE SYMBOL TABLE (RST)

The operation RST M, N will cause the assembler, during the first pass, to replace the symbol table by the table stored in the Nth block on tape M.

If the variable field is blank it will cause the symbol table to read in through the on-line card reader.

APPEND SYMBOL TABLE (AST)

The operation AST M, N causes the assembler, during the first pass, to append the symbol table with the table stored in the Nth block on tape M. If the variable field is blank it will cause the additional symbol table to be read in through the on-line card reader.

DECIMAL DATA (DEC)

The decimal data in the variable field is converted to binary and stored in consecutive memory locations. Successive words of data on a card are separated by commas, and the data field is terminated by a blank. All punching to the right of this blank is treated as a remark. If a symbolic location is punched on the card it is assigned to the first word generated by the card.

Algebraic signs are indicated by a + or − preceding the quantity involved, + signs may be omitted, however.

Decimal numbers (with either sign) may be in one of the following three classes:

(1) An *interger,* which may range from 0 to 68719476735; the integer should not be followed by a decimal point.

Example: DEC 0, 4, −99, +68719476735

(2) A *fixed point fraction,* preceded by a decimal point, and consisting of up to eleven digits.

Example: DEC .25, +.875, −.4,
+.14285714286

(3) A *scaled fixed point number,* with an attached "binary scale factor" following the letter "B" immediately after the number. The numbers 12.875B-4 and .1102B3 are examples of scaled fixed point numbers with negative and positive binary scale factors respectively. A number with a negative binary scale factor (in this example, 12.875B-4) is scaled downward (scaled by 2^{-4}) so that the binary point in the converted number is considered to be to the right of bit 4 (in this example). A number with a positive binary scale factor (in this example, .1102B3) is scaled upward by a power of two specified by the number after the B, in this example scaled by $2^3 = 8$, resulting in a larger number (in this case equivalent to $8 \times .1102 = .8816$).

Example: DEC 12.875B-4, −.1102B3,
+.66667B-1, −3.1415926536B-2

Note that the binary scale factor must be in the proper range to scale the number so it will "fit" within a word. Improperly scaled numbers such as −12.5B-1 or +.333B4 will result in an assigned value of 0 and a "V" flag on the listing.

FLOATING DECIMAL DATA (FLT)

The decimal data in the variable field is converted to 9400 floating-point form. With the FLT operation a decimal exponent may be specified by the programmer. If out of range data (i.e., greater than 10^{76} and less than 10^{-76}) is called for, the assembler supplies a zero and an error indication in the listing.

The decimal exponent used is the number which immediately follows the letter E. If E is not used the exponent is assumed to be zero. If no decimal point (.) is used, it is assumed to be at the right-hand end.

Again the data words are separated by commas and the data field is terminated by a blank. If a symbol is associated with the card, it is assigned to the first word generated.

OCTAL DATA (OCT)

The octal integer data in the variable field is converted to binary and assigned to successive memory locations. The data words are separated by commas, and the data field is terminated by a blank. If a symbol is associated with the card, it is assigned to the first word generated.

DATA (DATA)

This operation is a combination of DEC and FLT. If a decimal point (.) or E or both occur in a number without B, the number is converted to floating point binary.

Example: 1.4, 1.3E-3,1.

If a B does occur in the number, it is converted to fixed point binary according to the rules for DEC.

Example: 1.8B-3, 201B, 8E1B-3.

If neither a decimal point (.), B, or E is encountered, the integer is converted to a binary interger with binary point at the right hand end.

If overflow (or underflow) occurs in the processing of any DEC, FLT, or DATA quantity, the error is flagged on the output listing with a "V" and a value of zero is outputted for the erroneous quantity.

BINARY CODED INFORMATION (BCI)

Whenever a card reading yyy BCI N, is encountered, the N Fieldata words following the comma are assigned to successive memory locations. The symbol location yyy is assigned to the first word generated. Note that the condition $N \leq 9$ must hold otherwise the card is ignored and flagged as in error.

BINARY CODE INFORMATION (BCZ)

This operation is the same as BCI except that all blank characters are replaced with masterspaces.

BINARY OUTPUT MODE (ABS) (FUL) (REL)

The form of binary output is determined by the programmer. The output which follows an ABS card

is in the standard absolute format. The output which follows a FUL card has up to 24 words per card and no control, loading, or hashsum information. The output following a REL card is in standard-relocatable format. Whenever one of these output operations is met the assembler punches the waiting output before changing to the new mode. If no form is specified the absolute will be used.

REMARKS (REM)

This operation causes the entire card to be treated as a remark. No words are generated. An asterisk in column 7 will provide the same feature.

LIBRARY CALL (LIB)

The library routine identified by the symbol in the location field is obtained from the library tape and inserted in the program. The identifying symbol is not entered in the symbol table, but any symbols appearing in the library routine are entered and properly defined.

The first set of information on the library tapes is an ordered list of the subroutines which are on the tape. The assembler keeps track of the position of the library tape and treats the list of subroutines as a table of contents. As subsequent requests are met the library tape is backspaced or skipped to obtain the proper routine. The tape unit that the library tape is mounted on is specified by the variable field of the LIB card. If this field is blank the system standard drive is used. The table of contents of only one tape is retained in memory. In order to minimize tape researching time, it is recommended that the library routines be taken from one library tape at a time and in the order that they are on the tape.

Library routines may be called in two ways. Calls on library routines are indicated by the use of a left parenthesis followed by the call word e.g., TRL (SQRT. In the case where no LIB SQRT is used but only a TRL (SQRT, the assembler will stack the square root routine object coding at the end of used memory, in essentially the same way that literals are stacked. However, if a LIB SQRT is written, then the square root routine will be punched directly in line. In both cases the method of referencing the square root routine is the same and only the position of the object coding is different.

PRINT LIBRARY ROUTINE (PLR)

Normally the routines copied from the library tape are not included in the printed listing. If a PLR card is in the symbolic deck, all library routines will be included in the output listing.

TRANSFER CARD (TCD)

A TCD card causes the assembler to punch the waiting output and also punch a transfer card to the location called for by the variable field of the TCD card. A TCD card does not terminate the assembly.

END OF PROGRAM (END)

There are two uses of an END card in the assembly process. One is to mark the end of a Macro defining skeleton and the other is to signal the assembler that the end of the object program has been reached. The assembler can tell by the preceeding input which action is called for by the END card. The end-of-object-program-mode causes the waiting binary output to be punched, a transfer card to the location specified by the expression in the variable field to be punched, and the assembly terminated. The end of defining a skeleton mode causes the assembler to return to its normal processing.

FINISH (FIN)

A FIN card indicates the end of a group of stacked assemblies.

EJECT A LISTING PAGE (EJECT)

This pseudo-op affects the listing output only. It causes the printing of a page to cease and start again at the top of a new page.

SPACE A LINE (SPACE)

This pseudo-op affects the listing only. The variable field contains a predefined field specifying the number of lines to be shipped. A blank variable field will be considered a unit (1) skip.

LITERAL ORIGIN CARD (LOC)

The variable field of the LOC pseudo-op specifies the nominal origin for the literals. The variable field may be any predefined expression. Provision has been made for 1000_{10} literals, after which the old table is punched and a new table is formed.

General instructions for composing literals:

format is = (VALUE, where (VALUE) is a decimal interger;

or = /(VALUE), where (VALUE) is an octal interger.

The decimal integer is converted to fixed or floating point (see DATA-OP).

Example: LOC /100
CLA =127

will generate the instruction word:
10 00000 00100

and location 100_8 will contain the data word:
000000000177

The literal table is punched when another LOC card is encountered, or at the end of the assembly.

UNDEFINED SYMBOL ORIGIN (USO)

During the second pass all subsequent undefined symbols will be assigned consecutive values starting at the value of the expression in the variable field of the USO card.

VARIABLE FIELD DEFINITION (VFD)

This operation causes the variable length fields specified in the variable field of the card to be strung together to form 37 bit logical words. The Fields may consist of octal, decimal, or Fieldata information. The subfields may span across computer words however, no subfield may be longer than 37 bits. The subfields and their descriptions are punched without blanks across the variable field of the card. The first blank column signals the end of the data field. The various subfields are separated by commas and the data and description part of the subfield are separated by a slash.

The subfield starts with a descriptor which tells the length of the subfield in bits. If the descriptor is prefixed by an "O" or an "F" it means that the data is in octal or Fieldata form. The absence of a prefix means that the data is an unsigned decimal integer or an expression. The descriptor is terminated by a slash and the data portion of the subfield follows. The data portion of the subfield is terminated by a comma (or a blank for the last subfield). If the data portion consists of Fieldata characters the subfield must be an integral of six bits long and enough characters must be in the data field to fill the subfield. Octal and decimal information in a data field need not exactly fit the subfield. If the integer in data field is to be unsigned decimal (i.e., no special prefix) a symbolic reference (expression) may be used. Again the subfield will be computed mod.2^X where X equals the field length. Headed symbols are also allowed. If a VFD card does not fill the last word, the generated bits occupy their allotted positions in the *high order* part of the word. VFD 1/1 will generate 1000——0.

MACRO PSEUDO-OPERATIONS

Many programs have within themselves some body of coding performing a certain function, which is repeated in numerous places in the main program. In this type of coding, the only essential difference between the coding appearing at each of these places is the change of field names within the body of code. That is, each body of coding performs the same logical task, but the input and output symbolic names are different. Without the macro concept these routines are written by the programmer in each place that he wishes the coding to appear. He may also try to write the routine as a closed subroutine, but in many cases, this technique is cumbersome since the necessary initialization must be performed at object time. The macro approach is a simple answer to this problem.

Use of Macros

Oftentimes in a program one uses the same series of instructions over and over (e.g., matrix manipulation, calling sequence etc.) and it is a desirable feature of an assembly program to be able to generate these fixed sequences with a minimum of coding. This is done by the use of macros.

Macro Definitions: A macro definition is composed of a macro statement and a macro skeleton. The macro statement defines those fields which are parameters to the macro while the macro skeleton indicates the position within each specific macro use where the parameters are substituted.

Example:

```
BC MACRO  A, B      macro statement  ┐
    INS     A   }   macro skeleton   │ macro definition
    INS     B   }                    │
    END             end statement    ┘
```

Macro Definitions must be done before any coding of the specific macros.

The preceding skeleton, extremely simplified, can be used to demonstrate the salient feature of the macro definition.

Macro Statement: BC is the title of the new op-code which is to generate the desired sequence for each specific macro. The variable field is used to define the parameters within the skeleton. The macro statement defines those parameters which are substitutable in each specific macro op.

Skeleton: This defines the set of words which are to be generated by the new op code defined in the macro statement. Symbolic location titles, op codes, or variable field items may be parameter defined.

End Statement: This signals to the assembler that definition for this macro has been completed. It therefore signals a return to normal processing procedure.

Skeleton Format: A. Character(s) in the symbolic location field will be treated in any one of three ways.

1. The symbol name may also occur in the parameter list in which case it will be assigned its associated parameter name and equivalence and inserted in the symbol table upon the encountering of the macro.
2. If the symbol name does not also occur in the parameter list, it is to be assumed that this line is to be referred to from *within* the macro.

Name and equivalence are stored in a symbol list associated with the macro (this symbol *cannot* be addressed from outside the macro).

3. In conjunction with an EQU or SYN card, both symbol name and equivalence can be parameterized. However the variable field of this skeleton line must be a reference to one parameter value and may *not* contain any variable field connector (+, −, /, *, $ etc.).

Restrictions

A. After any skeleton line which has a variable field value to be computed from the parameter list, and of which the op code has a direct influence on the location counter (e.g., BSS N (where N is in parameter list)), no further symbolic — location — definition of line position is possible.

B. An op-code may be parameter-defined also, in which case the parameter definition must be single-valued. Macros may occur within macros, in which case the main parameter list defines all interior items (including the parameter lists for any interior macros).

C. Variable field characters and/or symbols may refer to parameter values or defined symbols, connected by any legitimate variable field connector. ((+),(−),(*),(/),(,),(b))

Note 1: In using the programmer-defined macro, the parameter list may overflow into as many cards as are necessary to insert the desired number of parameters. Each card but the last must terminate with a comma — blank (,b) sequence and each card but the first must contain the op-code ETC.

Note 2: At present 32_{10} parameters per macro may be specified in the list, 100_{10} macro definitions may be processed.

EXAMPLE OF MACRO CODING

```
CSQ     MACRO     B,C,D
  A     CLA       B,1
        STR       C,1
        TRX       A,1,1
        TRU       *+D+1        or (C+D)
  C     BSS       D
        END
```

Coding could be

```
Starts at loc 100   LXS   0,1,100
                    CSQ   COMMON, MTRX1,100
                    CLA   MTRX1
```

This would be equivalent to:

```
100         LXS     0,1,100
101     A   CLA     COMMON,1
102         STR     MTRX1,1
103         TRX     A,1,1
104         TRU     *+101
105     C   BSS     100
251         CLA     MTRX1
```

Note that in the above example the value of the symbol C within the macro is being used to define the value of the symbol MTRX1 for the main program.

That is — a value has not been assigned to the symbolic location MTRX1 from the main program but this value is determined only from the macro, e.g., MTRX1 is assigned the value 105.

APPENDIX K

LIST OF OPERATION CODES CURRENTLY PROCESSED BY 94AP

Within This Appendix, The Field Definitions Are As Follows

Field Explanation

1) Effect

GW(S) = Generate Word(s)

2) Field Interpretation

A = Address Field
C = Count Field
I = Index Field
M = Modifier Field
O = Octal Integer
S = Device Field
( ) = May Be Coded, But Not Required

Effect	Title	Field Interpretation	Purpose Or Octal Equivalent
GW	BS	S,C	670000000000
	DS	A,I,M	050000000000
	ES	A,S,C	630000000000
	GW	A,S,C	740000000000
	PR	A,(I),M	630004000000
	PS	A,(I),M	634000000000
	PT	A,(I),M	634000000000
	RW		770000000000
	SK	00...00	640000000000
	SP	S,C	660000000000
	SS	A,S,C	620000000000
	ST	(A),(I),(M)	630000000000
	WK	00...00	650000000000
	WO	A,C	760000000000
	WW	A,S,C	750000000000
	PRA	A	630001700000
	PRO	A	630001000000
	PRP	A	630000100000
	PRS	A	630000200000
	PRT	A	630000400000
	PSA	A	630005700000
	PSO	A	630005000000
	PSP	A	630004100000
	PSS	A	630004200000
	PST	A	630004400000
	PTI	A	634002000000
	PTU	A	634000000000
	SKB	00..00	640000000000
	SKF	00..00	640001000000
	SSE	A,S	621000000000
	SSN	A,S	622000000000
	SSS	A,S	624000000000
	SS1	A,S	620010000000
	SS2	A,S	620020000000
	SS3	A,S	620030000000
	SS4	A,S	620040000000
	STD		630000000000
	STI		630000000000
	GWBD	A,C	740002100000
	GWBI	A,C	740003100000
	GWBP	A,C	740000100000

Effect	Title	Field Interpretation	Purpose Or Octal Equivalent
GW	GWFD	A,C	740002200000
	GWFI	A,C	740003200000
	GWFP	A,C	740000200000
	GWWD	A,C	740002000000
	GWWI	A,C	740003000000
	GWWP	A,C	740000000000
	PRAI	A	630003700000
	PROI	A	630003000000
	PROP	A	630001100000
	PROS	A	630001200000
	PROT	A	630001400000
	PRPI	A	630002100000
	PRSI	A	630002200000
	PRSP	A	630000300000
	PRTI	A	630002400000
	PRTP	A	630000500000
	PRTS	A	630000600000
	PSAI	A	630007700000
	PSOI	A	630007000000
	PSOP	A	630005100000
	PSOS	A	630005200000
	PSOT	A	630005400000
	PSPI	A	630006100000
	PSSI	A	630006200000
	PSSP	A	630004300000
	PSTI	A	630006400000
	PSTP	A	630004500000
	PSTS	A	630004600000
	SSE1	A,S	621010000000
	SSE2	A,S	621020000000
	SSE3	A,S	621030000000
	SSE4	A,S	621040000000
	SSNE	A,S	623000000000
	SSN1	A,S	622010000000
	SSN2	A,S	622020000000
	SSN3	A,S	622030000000
	SSN4	A,S	622040000000
	SSSE	A,S	625000000000
	SSSN	A,S	626000000000
	SSS1	A,S	624010000000
	SSS2	A,S	624020000000
	SSS3	A,S	624030000000
	SSS4	A,S	624040000000
	WWBD	A,C	750002100000
	WWBI	A,C	750003100000
	WWBP	A,C	750000100000
	WWFD	A,C	750002200000
	WWFI	A,C	750003200000
	WWFP	A,C	750000200000
	WWWD	A,C	750002000000
	WWWI	A,C	750003000000
	WWWP	A,C	750000000000
	PROPI	A	630003100000
	PROSI	A	630003200000
	PROSP	A	630001300000
	PROTI	A	630003400000
	PROTP	A	630001500000
	PROTS	A	630001600000
	PRSPI	A	630002300000
	PRTPI	A	630002500000
	PRTSI	A	630002600000
	PRTSP	A	630000700000
	PSOPI	A	630007100000
	PSOSI	A	630007200000
	PSOSP	A	630005300000
	PSOTI	A	630007400000

Effect	Title	Field Interpretation	Purpose or Octal Equivalent
GW	PSOTP	A	630005500000
	PSOTS	A	630005600000
	PSSPI	A	630006300000
	PSTPI	A	630006500000
	PSTSI	A	630006600000
	PSTSP	A	630004700000
	SSNE1	A,S	623010000000
	SSNE2	A,S	623020000000
	SSNE3	A,S	623030000000
	SSNE4	A,S	623040000000
	SSSE1	A,S	625010000000
	SSSE2	A,S	625020000000
	SSSE3	A,S	625030000000
	SSSE4	A,S	625040000000
	SSSNE	A,S	627000000000
	SSSN1	A,S	626010000000
	SSSN2	A,S	626020000000
	SSSN3	A,S	626030000000
	SSSN4	A,S	626040000000
	PROSPI	A	630003300000
	PROTPI	A	630003500000
	PROTSI	A	630003600000
	PRTSPI	A	630002700000
	PSOSPI	A	630007300000
	PSOTPI	A	630007500000
	PSOTSI	A	630007600000
	PSTSPI	A	630006700000
	SSSNE1	A,S	627010000000
	SSSNE2	A,S	627020000000
	SSSNE3	A,S	627030000000
	SSSNE4	A,S	627040000000
	SCKNPF	A,C	700004100000
	SCKNP	A,C	700004000000
	SCKNXF	A,C	700005100000
	SCKNX	A,C	700005000000
	SCKNDF	A,C	700006100000
	SCKND	A,C	700006000000
	SCKNIF	A,C	700007100000
	SCKNI	A,C	700007000000
	SCKDPF	A,C	700004300000
	SCKDP	A,C	700004200000
	SCKDXF	A,C	700005300000
	SCKDX	A,C	700005200000
	SCKDDF	A,C	700006300000
	SCKDD	A,C	700006200000
	SCKDIF	A,C	700007300000
	SCKDI	A,C	700007200000
	SCKKPF	A,C	700004500000
	SCKKP	A,C	700004400000
	SCKKXF	A,C	700005500000
	SCKKX	A,C	700005400000
	SCKKDF	A,C	700006500000
	SCKKD	A,C	700006400000
	SCKKIF	A,C	700007500000
	SCKKI	A,C	700007400000
	SCKAPF	A,C	700004700000
	SCKAP	A,C	700004600000
	SCKAXF	A,C	700005700000
	SCKAX	A,C	700005600000
	SCKADF	A,C	700006700000
	SCKAD	A,C	700006600000
	SCKAIF	A,C	700007700000
	SCKAI	A,C	700007600000
	SCSNPF	A,C	704004100000
	SCSNP	A,C	704004000000
	SCSNXF	A,C	704005100000

Effect	Title	Field Interpretation	Purpose Or Octal Equivalent
GW	SCSNX	A,C	704005000000
	SCSNDF	A,C	704006100000
	SCSND	A,C	704006000000
	SCSNIF	A,C	704007100000
	SCSNI	A,C	704007000000
	SCSDPF	A,C	704004300000
	SCSDP	A,C	704004200000
	SCSDXF	A,C	704005300000
	SCSDX	A,C	704005200000
	SCSDDF	A,C	704006300000
	SCSDD	A,C	704006200000
	SCSDIF	A,C	704007300000
	SCSDI	A,C	704007200000
	SCSDPF	A,C	704004500000
	SCSKP	A,C	704004400000
	SCSKXF	A,C	704005500000
	SCSKX	A,C	704005400000
	SCSKDF	A,C	704006500000
	SCSKD	A,C	704006400000
	SCSKIF	A,C	704007500000
	SCSKI	A,C	704007400000
	SCSAPF	A,C	704004700000
	SCSAP	A,C	704004600000
	SCSAXF	A,C	704005700000
	SCSAX	A,C	704005600000
	SCSADF	A,C	704006700000
	SCSAD	A,C	704006600000
	SCSAIF	A,C	704007700000
	SCSAI	A,C	704007600000
	SCWNPF	A,C	700000100000
	SCWNP	A,C	700000000000
	SCWNXF	A,C	700001100000
	SCWNX	A,C	700001000000
	SCWNDF	A,C	700002100000
	SCWND	A,C	700002000000
	SCWNIF	A,C	700003100000
	SCWNI	A,C	700003000000
	SCWDPF	A,C	700000300000
	SCWDP	A,C	700000200000
	SCWDXF	A,C	700001300000
	SCWDX	A,C	700001200000
	SCWDDF	A,C	700002300000
	SCWDD	A,C	700002200000
	SCWDIF	A,C	700003300000
	SCWDI	A,C	700003200000
	SCWKPF	A,C	700000500000
	SCWKP	A,C	700000400000
	SCWKXF	A,C	700001500000
	SCWKX	A,C	700001400000
	SCWKDF	A,C	700002500000
	SCWKD	A,C	700002400000
	SCWKIF	A,C	700003500000
	SCWKI	A,C	700003400000
	SCWAPF	A,C	700000700000
	SCWAP	A,C	700000600000
	SCWAXF	A,C	700001700000
	SCWAX	A,C	700001600000
	SCWADF	A,C	700002700000
	SCWAD	A,C	700002600000
	SCWAIF	A,C	700003700000
	SCWAI	A,C	700003600000
	SCBNPF	A,C	704000100000
	SCBNP	A,C	704000000000
	SCBNXF	A,C	704001100000
	SCBNX	A,C	704001000000
	SCBNDF	A,C	704002100000

Effect	Title	Field Interpretation	Purpose Or Octal Equivalent
GW	SCBND	A,C	704002000000
	SCBNIF	A,C	704003100000
	SCBNI	A,C	704003000000
	SCBDPF	A,C	704000300000
	SCBDP	A,C	704000200000
	SCBDXF	A,C	704001300000
	SCBDX	A,C	704001200000
	SCBDDF	A,C	704002300000
	SCBDD	A,C	704002200000
	SCBDIF	A,C	704003300000
	SCBDI	A,C	704003200000
	SCBKPF	A,C	704000500000
	SCBKP	A,C	704000400000
	SCBKXF	A,C	704001500000
	SCBKX	A,C	704001400000
	SCBDXF	A,C	704001300000
	SCBDX	A,C	704001200000
	SCBDDF	A,C	704002300000
	SCBDD	A,C	704002200000
	SCBDIF	A,C	704003300000
	SCBDI	A,C	704003200000
	SCBKPF	A,C	704000500000
	SCBKP	A,C	704000400000
	SCBKXF	A,C	704001500000
	SCBKX	A,C	704001400000
	SCBKDF	A,C	704002500000
	SCBKD	A,C	704002400000
	SCBKIF	A,C	704003500000
	SCBKI	A,C	704003400000
	SCBAPF	A,C	704000700000
	SCBAP	A,C	704000600000
	SCBAXF	A,C	704001700000
	SCBAX	A,C	704001600000
	SCBADF	A,C	704002700000
	SCBAD	A,C	704002600000
	SCBAIF	A,C	704003700000
	SCBAI	A,C	704003600000
	HLT	(A),(I),(M)	000000
	ABD	A,(I),M	240000000000
	ADD	A,(I),M	120000000000
	ADM	A,(I),M	130000000000
	CAM	A,(I)	110000000000
	CLA	A,(I)	100000000000
	CLS	A,(I)	140000000000
	CLZ		140000077740
	CSM	A,(I)	150000000000
	CSZ		150000077740
	CYL	A,(I)	350000000000
	CYS	A,(I)	340000000000
	DVD	A,(I),M	220000000000
	DVL	A,(I),M	230000000000
	FAM	A,(I),M	570000000000
	FLA	A,(I),M	560000000000
	FLD	A,(I),M	270000000000
	FLM	A,(I),M	260000000000
	FLS	A,(I),M	600000000000
	FSM	A,(I),M	610000000000
	LDQ	A,(I)	510775100000
	LGA	A,(I)	030000000000
	LGM	A,(I)	020000000000
	LGN	A,(I)	040000000000
	LOD	A,(I),M	510774000000
	LXS	A,I,M	530000000000
	MLR	A,(I)	210000000000
	MLY	A,(I)	200000000000
	MOV	A,I-M	520000000000

Effect	Title	Field Interpretation	Purpose Or Octal Equivalent
GW	MSK	A,(I)	550000000000
	MVQ	I-M	520000077751
	MVZ	I-M	520000077740
	NRM	A,(I)	370000000000
	RPA	A,(I)	540000000000
	RPT	A,(I),M	010000000000
	SBB	A,(I),M	250000000000
	SBM	A,(I),M	170000000000
	SEN	A,(I),M	050000000000
	SHL	A,(I),M	300000000000
	SHR	A,(I)	320000000000
	SLL	A,(I),M	310000000000
	SNR	A,(I),M	070000000000
	SNS	A,(I),M	060000000000
	SRL	A,(I)	330000000000
	STR	A,(I)	500000000000
	SUB	A,(I),M	160000000000
	TRC	A,(I)	470000000000
	TRI	A,(I)	510775300000
	TRL	A,(I),(M)	410000000000
	TRN	A,(I)	460000000000
	TRP	A,(I)	440000000000
	TRS		420000000000
	TRU	A,(I),(M)	400000000000
	TRX	A,I,M	430000000000
	TRZ	A,(I)	450000000000
	BSP	S,C	670000000000
	BSPB	S,C	670000000000
	BSPF	S,C	674000000000
	RAN	A,S,C,	700000000000
	RANW	A,S,C,	700000000000
	RANB	A,S,C,	704000000000
	ROK	A,S,C,	720000000000
	RRV	A,S,C,	710000000000
	RRVW	A,S,C,	710000000000
	RRVB	A,S,C	714000000000
	RWD	S	770000000000
	WAN	A,S,C	740000000000
	WOK	A,S,C	760000000000
	WWA	A,S,C	750000000000
	SKP	S,C	660000000000
	SKPB	S,C	660000000000
	SKPF	S,C	664000000000

Within This Appendix, The Field Definitions Are As Follows

Field Explanation

1) Effect

CO = Card Output
EST = Enter Symbol In Table
GWS = Generate Words
HST = Head Symbols
LO = Listing Output
LC = Listing Control
LIC = Library Control
MAD = Macro Definition
ORG = Set Origin Control
STI = Symbol Table Input
STO - Symbol Table Output

2) Field Interpretation

A = Alphanumeric Field
D = Decimal Integer
F = Literal Fieldata Character
M = Tape Designator
N = Count
O = Octal Integer

Effect	Title	Field Interpretation	
CO	ABS		Absolute Bin Card Format
STI	AST	M,N	Symbol Table Append
GWS	BCI	N,FF...FFF	Enter Fieldata W Space Fill
GWS	BCZ	N,FFF...FFF	Same With Bin Zero Fill
GWS	BES	N	Enter Zero Block
GWS	BSS	N	Same
GWS	DATA	DD...DD(F)D	Enter Decimaldata
GWS	DEC	DD,...,DD(F)	Enter Decimaldata
EST	DEF	00...00	Octal Value For Symbol
LC	EJECT		Advance Listing To Next Page
LO	END	A	End Macro Or Assembly
EST	EQU	A	Equate Nonrelocatable Fields
LO	FIN		Terminate Stacked Assemblies
GWS	FLT	DD,...,DD(F)D	Floating Point Data
STO	FST	M,N	Write Final Symbol Table
CO	FUL		Full Bin Card Format
HST	HED	F	Heading Character
LIC	LIB	A	Enter A Library Routine
ORG	LOC	A	Origin For Literal Table
STO	LST		List Symbol Table
MAD	MACRO	A,A,A,...A	Define Macro Skeleton
GWS	OCT	000,...,000	Enter Octal Numbers
ORG	ORG	A	Origin For Program
LO	PLR		List All Library Routines
CO	REL		Relocatable Bin Card Format
LO	REM		Write Remark
STI	RST	M,N	Replace Symbol Table
LC	SPACE	N	Advance Listing N Lines
EST	SYN	A	Equate Relocatable Fields
CO	TCD	A	Punch Transfer Card
ORG	USO	A	Origin For Undefined Symbols
GWS	VFD	(F)N/D(O)(F)...	Variable Field Definition
STO	WST	M,N	Write Symbol Table

Within This Appendix, The Field Definitions Are As Follows

Field Explanation

1) Effect

SIM = Symbolic Item Macro
IOM = Input And/Or Output Macro

2) Field Interpretation

A = Symbolic Item Reference Field
B = Symbolic Item Index Field
C = Symbolic Item Bit Location Field
D = Input Or Output File Location Field
F = Input Or Output File Designator Field
K = A Set Of File Definition Fields

Effect	Title	Field Interpretation	
IOM	AHEAD	F,D	Skip Over Logical Records
IOM	BACK	F,D	Reverse Direction And Skip Record
IOM	CHANGE	F	Terminate A Reel Of A File
IOM	CLOSE	F	Terminate Processing Of A File
SIM	CLPL	A,B,C	Same
SIM	CLPR	A,B,C	Clear And Add A Symbolic Item
IOM	FILDEF	K	Define In A File
IOM	FLEX	F,D	Output Lo Vol Record
IOM	OPEN	F	Start Processing A File
IOM	READ	F,D	Get Next Record
IOM	READIN	F,D	Get Next Record
SIM	STOR	A,B,C	Store A Symbolic Item
SIM	STOS	A,B,C	Store A Symbolic Item
IOM	WRITE	F,D	Output A Record
IOM	WRTFRM	F,D	Output A Record