RCA 3301

REALCOM

TRAINING MANUAL

94-06-000

November 1964

The information contained herein
is subject to change without notice.
Revisions will be provided to advise
of such additions and/or corrections.

FOREWORD

Although some people may find this manual (particularly Section III) helpful in self-study, it is designed for use by experienced programmers in more or less formal training situations. Inexperienced programmers should consider a course in programming the RCA 301 before attempting to work with the RCA 3301 REALCOM System.

Of the three major sections in the text, only the third, which describes Assembly Language programming, is thought to be complete from a teaching point of view. Sections I and II, which contain descriptions of certain hardware elements and of the Operating System, respectively, depend heavily on other RCA documents. For further details of the REALCOM System itself the reader is referred to the 3301 System Reference Manual (94-16-000). A more comprehensive discussion of the Operating System is contained in the 3301 Operating System Manual (94-08-000).

Prepared by:

Education and Training
Electronic Data Processing
Radio Corporation of America
Camden 8, New Jersey

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

RCA 3301 SYSTEM

GENERAL INFORMATION

3301 SYSTEM - GENERAL INFORMATION

SECTION I - INTRODUCTION

The RCA 3301 system is a significant step in the evolution of the RCA 501/301 generation of computers. The two factors which have allowed achievement of this step are (1) the inclusion of an automatic interrupt system, (2) up to four modes of simultaneity for operation of Input/Output devices. These factors, incorporated with a high-speed main memory, and a Micro-Magnetic memory taking the place of conventional registers, fill the need for a high-speed, large capacity, economical, general purpose system with high throughput rates.

The 3301 incorporates an automatic interrupt system which enables immediate processing on two classes of interrupts; General or Real-Time. The hardware interrupt system is facilitated by use of controlling software. The software Operating System enables efficient handling of all types of interrupts, which may be generated by the necessity of Input/Output Control, machine errors, or Communications Devices.

Throughput operation is further enhanced by use of two standard Simultaneous Modes, which control all Input/Output operations. A third Simultaneous mode may be incorporated in a system to execute tape and random access operations. In addition, a Communications Mode Control may be utilized for Real-Time data flow.

SECTION II - 3301 SYSTEM COMPONENTS

A. Main Memory

The 3301 Main Memory is a decade-oriented, linear-select device. The term decade-oriented refers to the fact that any memory operation will access ten, seven-bit characters of information. As in the 301 system, where two characters were called a diad, the ten character word of the 3301 is called a decade.

The term linear-select refers to the method of electronically choosing the specified decade in memory. It indicates that only a single access wire passes through all 70 cores of a decade, as opposed to two wires per core in a coincident-current-selection memory. This allows neater and more compact packaging, as well as a higher level of reliability.

HSM (High Speed Memory) is physically arranged in a basic configuration of 40,000 locations, and may be field-modified in increments of 20,000 locations, with a maximum size of 160,000 characters. It is addressed by means of a decimal four-character address, whose scheme is compatible with 301 addressing.

B. Addressing

At first it may be difficult to comprehend how four decimal digits may represent 160,000 locations, since four decimal digits can only express quantities up to 9,999. Reviewing the 301 addressing scheme will show that in the 20K memory system, the 2^4 bit of the Most Significant Digit (MSD) was used to indicate a "carry bit". All addresses from 10,000 to 19,999 were represented by a 2^4 bit added to the MSD position.

When the 301 40K memory was designed, the 2^5 bit of the MSD of the address was mechanized to designate the additional 20,000 address locations. Thus, a binary count had been set up, utilizing the zone bits of the MSD of the address, to represent each segment of 10,000 locations. Chart 1 illustrates the relationship between the binary count and the groups of 10,000 locations.

ZONE BITS OF MSD		301 ADDRESS RANGE	
2^5	2^4	From	To
0	0	0000	9999
0	1	10000	19999
1	0	20000	29999
1	1	30000	39999

CHART 1

Note that the binary count of the zone bits is equal to the Most Significant Digit of the address range.

Since the zone bits of the MSD (C0) were completely utilized in expressing addresses up to 39,999, two more bits are necessary to express higher addresses up to 159,999. The zone bits of the second digit (C1) of the address is used to express these addresses, and follows the pattern of Chart #2.

ZONE BITS				3301 ADDRESS RANGE	
C1		C0			
2^5	2^4	2^5	2^4	From	To
0	0	0	0	0000	9999
0	0	0	1	10000	19999
0	0	1	0	20000	29999
0	0	1	1	30000	39999
0	1	0	0	40000	49999
0	1	0	1	50000	59999
0	1	1	0	60000	69999
0	1	1	1	70000	79999
⋮		⋮		⋮	⋮
1	1	1	1	150000	159999

CHART 2

Again, the zone bit binary count is equal to the number of the group of 10,000 locations, the address specifies.

As an example, the address of location 127,429 would be:

7U29

that of location 151,293 would be:

/S93

C. Core-Plane Configuration

The Main Memory has a destructive read-out system, with one core per bit. The cores are arranged in planes, constructed of 200 x 70 cores. Each plane is grouped into 200 - 10 character words, since a single access line is wired through 70 successive cores. Since each 3301 memory plane holds 2000 characters, the basic 40K 3301 memory must contain 20 planes. If memory is expanded, each 20K increment will add 10 planes. A full 160K memory will thus contain 80 planes.

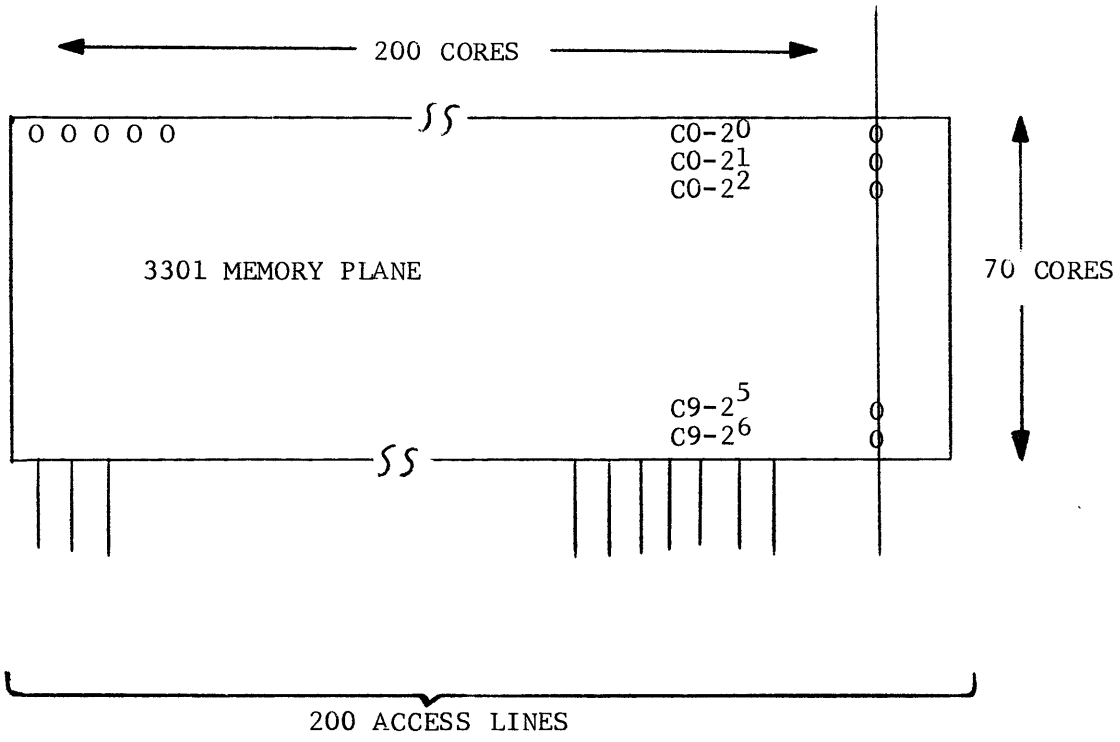


FIG. 1

These 80 planes are grouped into 4, 20 plane banks.

Fig. 1 is a drawing of a single 3301 memory plane, illustrating the 200 access lines that are required to enable each 10 - 7 bit character decade to be selected. When a current pulse is gated onto one access line, each of the 70 cores will be affected during a read or regenerate operation.

Note that each core along a single access line represents a bit position of each character of the decade. The characters in a decade are labelled C0 through C9, with their bit positions labelled as 2^0 through 2^6. The C0 character - 2^0 bit is at the top of the plane, and the C9 character - 2^6 bit is at the bottom of the plane. All bit positions of each character are grouped in that order. Each plane holds 2000 consecutive memory locations such as: 0000 to 9999, 2000 to 3999, &000 to A999, etc.

D. Address Decoding and Selection

In order to enable access of a specific HSM location, an address is placed in the Memory Address Register (MAR) at the beginning of a memory cycle. This address would have arrived via the four bus lines used for register communication in the 3301 system, after being transmitted (gated) from an address register.

The address is placed in the MAR to enable a decoding matrix to select one of the access lines in a plane; thus choosing a decade. The decoding matrix has the function of decoding the BCD and zone bit information in the address, to develop selecting signals for the desired access line. These selecting signals will allow read and write currents, generated by the HSM timing generator, to magnetically effect a decade in the core stack.

E. Memory Communication - Read/Regenerate

In order to allow communication between HSM and the Memory Register (MR) during the read portion of a memory cycle, an additional wire is wound through each core of each decade in a plane. It is called a sense winding, since it senses the magnetic state of each bit of the selected decade, and places the result, of either a one or zero bit, into the MR.

During the read portion of the memory cycle, a pulse is gated onto the selected decade access line, which tends to change the state of the 70 selected cores to the zero state. Any core which had been in the one state would generate a voltage pulse onto its sense line, during the change of state. Any core which held a zero bit would not change state, and therefore, would not generate a significant voltage pulse onto its sense line.

The voltage pulse on each sense line, whose core held a one bit, would be amplified, and would set a one bit into the proper stage of the MR. The MR had previously been reset to zero bits. Thus, the MR now contains the coding of the ten characters located in the selected decade of HSM.

Note, however, that the 70 cores of the selected decade are now all storing zero bits. The contents of the decade have been destroyed by reading. This is termed Destructive Read-Out. In order to retain

the decade content for future use, which is now stored in the MR, the MR content must be written back into memory during the last half of the memory cycle. This is called regeneration.

In order to regenerate the bit configuration of the decade into the core location, a reference is necessary to determine which bits of the decade are to contain ones, and which are to contain zeros. This can most easily be done by referencing the MR.

Since all the cores of the selected decade now contain zeros, it is necessary to write back only the one bits to specific cores. A one bit may be recorded in a core by generating a magnetic field; about the core, of a specific magnitude and polarity. As in the case of reading data from the core, the necessary magnetic field (to change it from the zero to the one state) may be generated by passing a current of sufficient magnitude along the access line. This would, however, change all cores of the decade to the one state.

A means is necessary to write one bits only to selected cores, referenced by the content of the MR.

All cores of a decade are serviced by only one access wire, but each core has its own sense wire. If the two may be used in a matrix system, one bits may be written to the selected cores.

Note that it was stated previously, that a magnetic field of a specific magnitude is necessary to cause a core to change its state. Only half the necessary value will have no effect on the core. If two adjacent wires (having currents passing through them generating only half the necessary magnetic field in each wire) cross each other, the total magnetic effect at their junction will be the summation of the magnetic field in each. This total field will be of sufficient magnitude to change a core to the one state.

Thus a "half current" pulse may be passed down the access wire and be felt by all cores of the selected decade. Another half current pulse may be passed down each Sense wire, servicing only each core which should contain a one bit. The net result is that the half current pulses will combine their magnetic fields about each core which will have a one bit regenerated to it, and all other cores serviced by either the sense lines or access lines will not be affected. Thus, one bits, referenced by the MR contents feeding the sense lines, are recorded in specific cores of the selected decade.

F. Main Memory Timing

It should be obvious from the previous discussion that the events concerned with memory operation must occur in a definite sequence. This sequence is developed by a logic block called the HSM timing generator.

The sequence of steps necessary are:

1. address memory decade
2. read selected decade into MR
3 regenerate decade

In order to mechanize these steps into the necessary detail of memory timing, we will find that the above three steps can be broken down into:

1. a. gate address from bus lines into MAR

   b. decode address and generate selections signals for desired decade access line

2. a. reset MR to all zero bits

   b. generate read pulse on access line

   c. place contents of sense lines into MR

3. a. generate write pulse along access line

   b. generate one bits from MR along sense lines

Due to certain physical properties of the cores and flip-flops used in the memory, each of the steps takes a certain amount of time. 1.5 microseconds is required to perform all the operational steps of memory, even though certain of the steps occur simultaneously.

Fig. 2 illustrates a time period of 1.5 microseconds, broken down into 214 nanosecond increments, and the memory timing relationship of each of these steps.

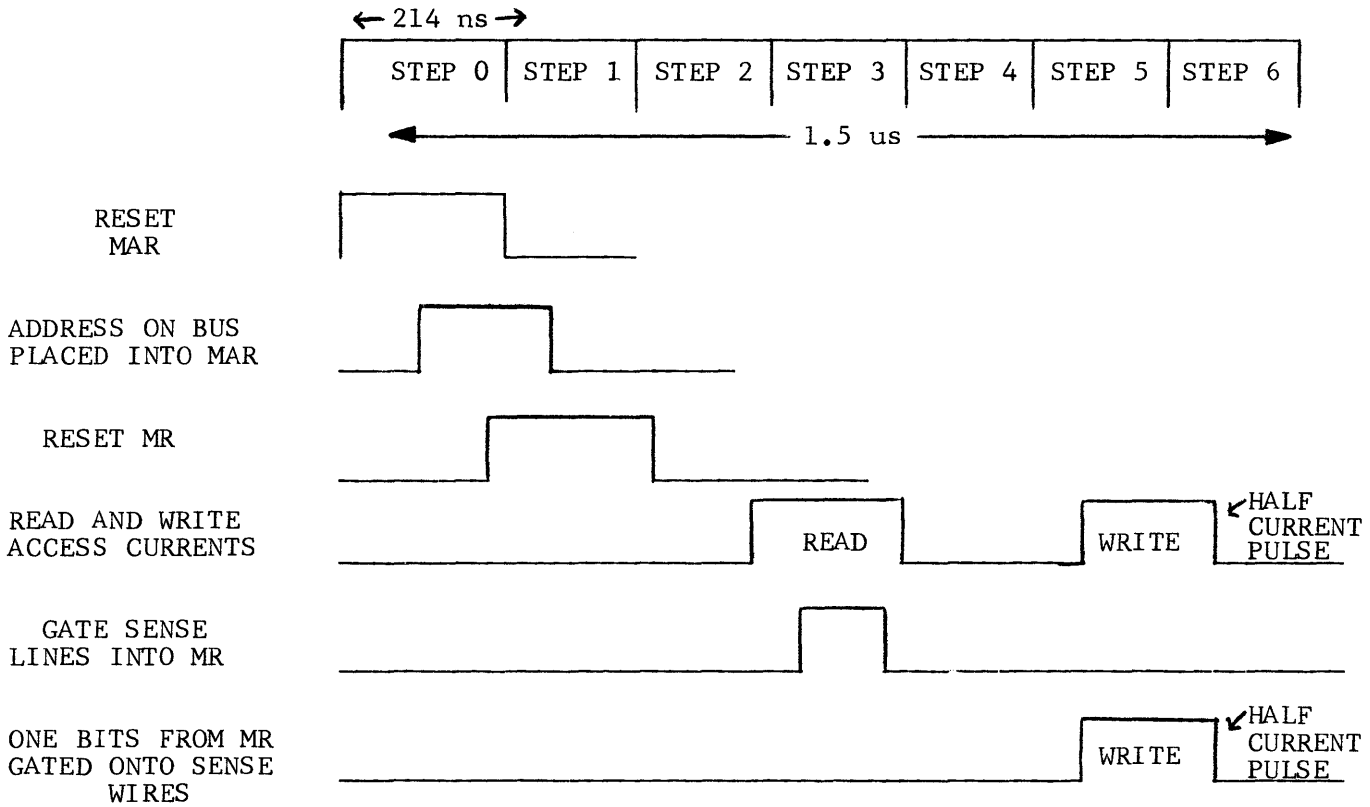


FIG. 2

HIGH SPEED MEMORY TIMING
(APPROXIMATE)

The first operation performed, at STEP 0, had not been discussed previously. The pulse shown is used to reset the previous contents of the MAR, before gating the new address into it. The MAR had initially contained the address from the previous memory cycle.

SECTION III - GENERAL TIMING INFORMATION

A. Instruction Timing

Since the 3301 Processor is part of the 501/301 family of systems, it too is a synchronous machine. That is, all instructions are executed utilizing three basic control-timing signals. These are based on the memory cycle timing and switching speeds of the logic circuitry.

The same three types of operation execution levels are utilized; namely, Time Pulses, Status levels, and Operation Code levels.

1. Operation Code Levels

An Operation Code Level is a signal which exists during the entire length of time an instruction is being executed. It is generated by logic which determines which Operation Code Character is in the Operation Code Register. Its function is to select all the logic blocks within the system that may be used to execute the instruction, and allow the proper linkage pathways between them to be constructed.

2. Status Levels

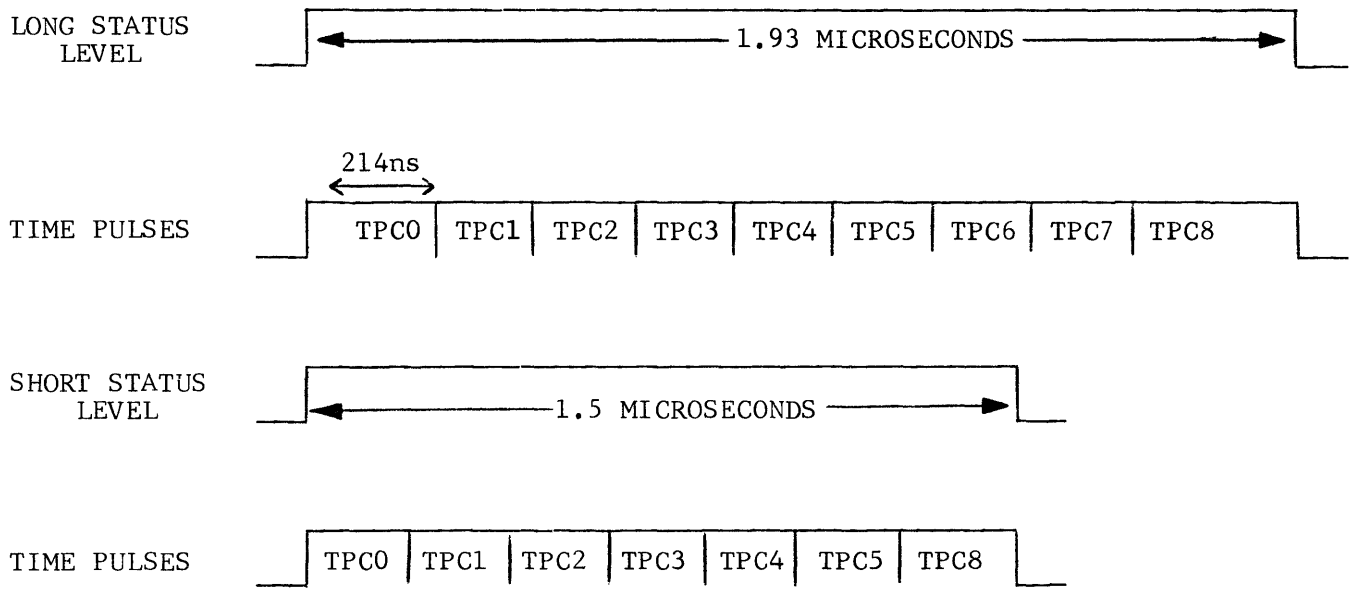
Any operation (or instruction) which is performed must be done in specific discrete steps, executed in a given sequence. In the 3301 processor, each instruction is executed by utilizing each logic block (such as a comparator or adder) at a specific time and in a specific sequence. The sequenced timing steps which are used to operate each logic block when executing an instruction are called Status Levels.

There are 33 basic status levels used in the 3301 system. Each one performs a certain basic operation, such as: reading a character from memory using the A address, adding two characters and writing the result in memory, taking a character from an Input/Output buffer and writing it to a memory location, etc. Using a certain combination of these status levels, executed in the proper sequence, allows an instruction to be performed.

Note that a status level generally must coincide with a memory cycle operation. Because of this fact, the basic 3301 status level duration time is the same as that of a memory cycle time. A basic status level exists for 1.5 microseconds, so that data may be read from, or written to memory, if necessary. Because certain status levels perform more transfers than others, they will require 1.93 microseconds duration time, since additional use of the Bus Lines is required.

3. Time Pulses

As each status level performs a single operation of an instruction, a timing pulse is necessary to execute each incremental step of a status level. Each time pulse is 214 nanoseconds in duration. A short status level of 1.5 microseconds duration consists of 7 time pulses and a long status level of 1.93 microseconds contains 9 time pulses.



STATUS LEVEL AND TIME PULSE

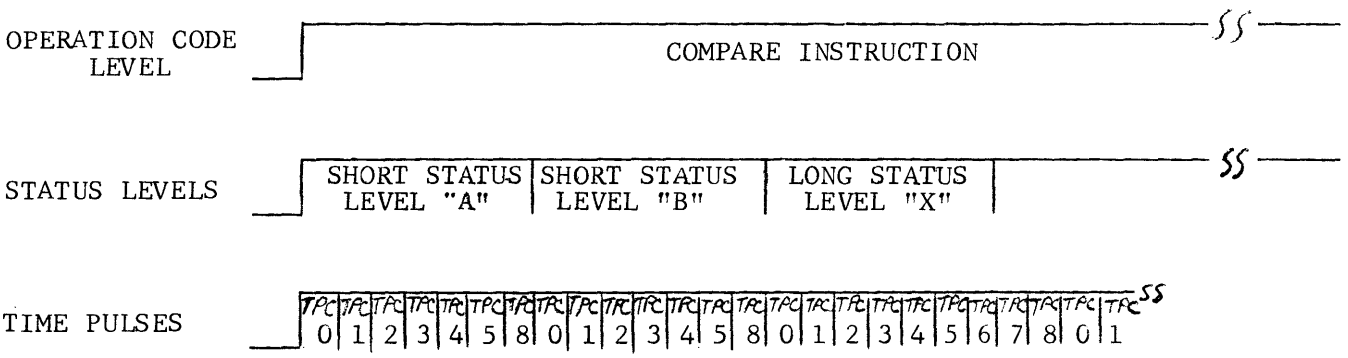
RELATIONSHIPS

FIG. 3

Fig. 3 illustrates Status Level and Time Pulse relationships. Note that the time pulses are labelled TPC0 through TPC8. The letters TPC stand for Time Pulse Clock, since a specific time pulse tells when a certain step should be performed. The numbering of the time pulses during the short status level follows the normal counting sequence until TPC5, since TPC8 is next. Effectively, to generate the short status level, TPC6 and TPC7 have been skipped over. This causes TPC8 to be generated immediately after TPC5.

Time pulses are generated in the machine by a logic block which is controlled by a fixed-frequency oscillator. They are generated continuously while power is applied to the machine.

4. Combined Timing



COMBINED TIMING RELATIONSHIPS

FIG. 4

COMBINED TIMING RELATIONSHIPS of Fig. 4 illustrates how TPC's and Status Levels (STL's) are utilized during an Operation Code Level to execute an instruction. The example given is that of a Compare instruction, and illustrates the possible status levels that would be needed to execute one comparison. Note that each necessary operation is listed.

Each STL (status level) is given a name: A, B, X. The name generally refers to the register address it uses for specifying a memory location. The first STL uses the A address, therefore it's called an A STL. Since the last STL performs a function which does not require an address, it is called an X STL.

Each TPC would be responsible for executing each minor step of a status level such as: addressing memory, gating a character from the MR to a Bus line, etc.

SECTION IV - SIMPLIFIED DATA FLOW

A. Basic Instruction Execution

The objective of this segment will be to explain the representative data flow concepts in the 3301 Processor, covering address and data transmission from register to register via the bus lines. Representative instructions will be covered in some detail, as well as the functions of Staticizing, utilizing Status Level and Time Pulse Clocking.

1. Execution Concepts

In order that any problem may be solved, whether by machine or human, its solution must be carried out in specific sequential steps. In a machine, the list of these steps to be taken is called a program, and each step is called an instruction. In order that a machine execute the program, it performs these instructions one at a time, in a definite sequence.

In order to execute each instruction, the machine must perform four basic steps:

a. Read - It must read the instruction from its memory.

b. Remember - The instruction must be "remembered" or stored while it is being executed..

c. Interpret - Before it can be executed, the machine must interpret the instruction to learn its parameter.

d. Execute - The interpreted instruction may now be executed correctly.

In the 3301 system, the functions of Reading and Remembering the instruction are performed by a process called Staticizing. The process of Staticizing consists of reading the instruction from its location in memory, and storing it in its operating registers so that it may be interpreted, and then executed.

B. Staticizing

C$\emptyset$	C1	C2	C3	C4	C5	C6	C7	C8	C9
OP CODE	N CHAR	A $A_\emptyset$	A A_1	A A_2	A A_3	B B_0	B B_1	B B_2	B B_3

3301 INSTRUCTION FORMAT

FIG. 5

Fig. 5 illustrates the 3301 instruction format. Note that it is compatible with 301 format. Each instruction is specified by 10 characters, which are always placed in a single decade of memory. That is, it must have its Operation Code Character located at an address ending in XXX$\emptyset$, and its B_3 character located at an address ending in XXX9.

There are 4 registers in the Control portion of the Processor, which are used to store the instruction while it is being executed. These are the OP, N, A, and B Registers. The OP and N Registers are single character registers, each constructed of 7 flip-flops.

The A and B Registers are not constructed by utilizing conventional flip-flops, in the 3301 system. Because of the various modes of simultaneity and the automatic interrupt system, multi-level storage for operating registers, and other machine conditions, are required. It is most economical to incorporate a small, rapid-access memory, in place of registers, to achieve this storage. This device is called the Micro-Magnetic Memory.

1. Micro-Magnetic Memory

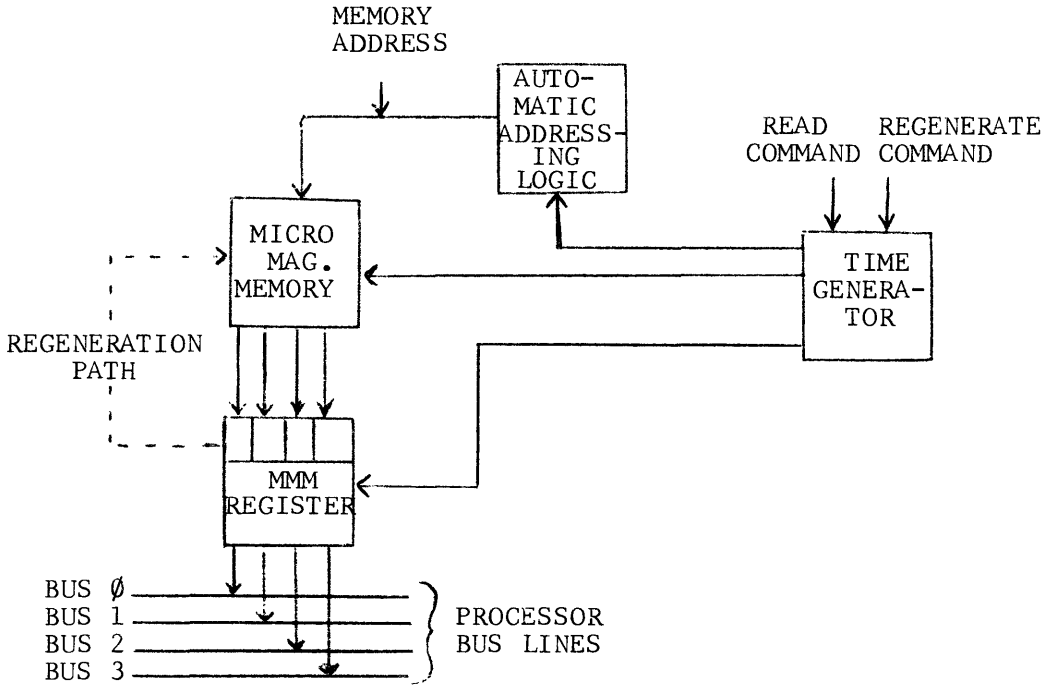
The MMM (Micro-Magnetic Memory) incorporated in the 3301 system is a word-oriented, octally addressed, core memory. Its format consists of 50 4-character words. A four-character word was chosen since each location generally is used to store a machine address.

Each word location of the MMM is given a standard name, relative to the address or machine condition which it will store. Fig. 6 illustrates the MMM layout, as well as the octal address necessary to specify any location.

MICRO MAGNETIC MEMORY LOCATIONS

8^0 / 8^1	1 (P)	2 (A)	3 (S)	4 (B)	5 (T)	6 (C)	7 (E)
0	P Register	A Register	S Register A-Add of Simo 1 Inst.	B Register	T Register B-Add of Simo 1 Inst.	C Register A-Add of Simo 2 Inst.	E Register B-Add of Simo 2 Inst.
1	← Storage of Machine Conditions During General Interrupt →						
	P-Reg	A-Reg	STA	B-Reg	STP	Machine Indicators	STPR
2	← Storage of Machine Conditions During Real Time Interrupt →						
	P-Reg	A-Reg	STA	B-Reg	STP	Machine Indicators	STPR
3	General Interrupt Routine Entry	← Storage of last Simo 1 Inst. →			← Multiply and Divide Storage →		
		A-Address	OP code & N char.	B-Address	MD 1	MD2	MD3
4	Multiply and Divide Storage MD4	← Storage of last Simo 2 Inst. →					
		A-Address	OP code & N char.	B-Address	Not Assigned	Not Assigned	Not Assigned
5		← Storage of last Simo 3 Inst. →			F Register	G Register	
	STP on STOP	A-Address	OP code & N char.	B-Address	A-Add of Simo 3 Inst.	B-Add of Simo 3 Inst.	Not Assigned
6	Not Assigned	Index Field #1	Increment Field #1	Index Field #2	Increment Field #2	Index Field #3	Increment Field #3
7	Real Time Interrupt Routine Entry	← DO NOT EXIST →					

FIG. 6



MICRO MAGNETIC MEMORY BLOCK DIAGRAM

FIG. 7

Fig. 7 shows a block diagram of the MMM. Note that it has all the components of the Main Memory; i.e., a corestack, a Memory Register, addressing logic, and a Timing Generator.

Since the MMM "Register" location to be used is dependent upon which sub-step (status level) of an instruction is being executed, the address of that location is automatically generated. For example, when storing the A address of an instruction during staticizing, the automatic addressing logic will generate the address of the "A" storage location. Referring to Fig. 6, the reader will note that this is location $02_{(8)}$.

The MMM is wired to perform either a read cycle or a write cycle at any given time. This is done in order to simulate register operation. Either a read cycle or a write cycle requires 214 nanoseconds to be executed; one processor time pulse. Again this simulates register operating speeds. Note the Read and Regenerate Command inputs to the Timing Generator. These are generated at the proper time during a status level in order to achieve the desired operation.

All communication with the corestack is via the MMM Register. It is a full, four character register to coincide with the word format of MMM. Note that all external communication is via the Processor bus lines.

2. Staticizing Data Flow

Because the 3301 instruction occupies a single decade location in HSM, only one memory cycle is required to access it during Staticizing. Once it has been read into the Main Memory Register (MR) it must then be placed in its appropriate operating registers.

The entire process of Staticizing thus requires only one status level, called P1 (it is called P1 because it utilizes the P register to address memory when accessing the desired instruction). The P1 status level is one of the long status levels which require 1.93 microseconds for execution.

The P1 status level requires this time because it must store both the A and B addresses of the instruction in the MMM. Only one address may be stored at a given time due to the four character format of MMM, and because only a single address may be transmitted via the processor bus lines at a given time.

Two MMM cycles are required to store each of the two addresses. This is a total time of 428ns + 428ns = 856ns. Since the instruction itself does not exist in MR until TPC3, 1 microsecond after the beginning of the status level, a minimum of 1.7 microseconds is requred to staticize an instruction. 1.93 microseconds or 9 time pulses is the figure actually chosen to ensure enough time for proper operation.

The following operating steps must occur during the P1 STL.

a. Access "P address" from MMM and address MAR.

b. Start Main Memory cycle and increment P + 10.

c. Place N character and OP code in N and NOR Registers after read-out occurs to MR.

d. Store "A address" in A location of MMM.

e. Store "B address" in B location of MMM.

f. Partially interpret the instruction in order to choose the necessary status flow for proper execution.

FIGURE 8 - STATICIZING DATA FLOW

P1 STATUS LEVEL	
TPCØ	P1 status level generates the address 01 for the "P location" of MMM, and a read cycle is performed. The P address is placed in the MMMR.
TPC1	The P address is transmitted via the bus lines from MMMR into the MAR to address the desired instruction. The Main Memory cycle begins.
TPC2	A MMM read cycle is performed to "reset" the contents of the P location so that the P address may be incremented by +10 in order to address the next instruction in sequence. The Bus Adder increments the address in MAR by +10.
TPC3	The contents of the Bus Adder are now transmitted over the bus lines into MMMR, and a MMM write cycle (with the P locations addressed) places the modified "P address" into its proper location. At this time, the decade from Main Memory has been placed into the MR.
TPC4	The OP code and N character are now gated from the MRØ and MR1 stages and into their appropriate registers. An MMM Read cycle is performed on the "A" location of MMM so that it may be "reset" to receive the new A address.
TPC5	The Interchange transmits the MR2, MR3, MR4, and MR5 characters onto the Bus lines. These four characters of the A address are then placed into MMMR while a write cycle is executed to the A location. Main Memory Regeneration occurs.
TPC6	Address the B location of MMM and generate a Read cycle to "reset" it for the new B address.
TPC7	Transmit the MR6, MR7, MR8, and MR9 characters onto the processor Bus lines, and place the four B address characters into MMMR.. A write cycle must be generated to write the B address into the B location of MMM.
TPC8	The Operation Code and the N character of the instruction are decoded to allow the next status level to be "selected". Since the next status level is the first of execution, it is called the FPL (First Processing Level).

C. Instruction Execution

Once an instruction has been staticized, it may then be executed. Instruction execution consists of performing the necessary sequential status levels, each of which perform a specific function. The total of these specific functions results in the completion of the instruction.

1. Instruction Data Flow

Each status level is able to perform an operation such as: transferring the content of one register to another via the bus lines, adding two characters and placing the result in a memory location, comparing two characters and setting an indicator to show the result, or transfer a character via the bus lines and Input/Output channels to an I/O device so that it may be written on the storage medium (such as tape).

In addition to performing each of these major functions, each status level performs several minor ones. These minor functions are performed to set up machine conditions for proper execution of the next status level. Some minor functions that are performed are: incrementing or decrementing the contents of an address register, counting down a register which determines a specific number of major operations to be executed, sensing for equality between two addresses, etc.

2. Add Instruction Data Flow

In order to better understand the "big picture" of status level data flow, we will consider an Add instruction, since the need and function of adding may be readily understood.

Consider the operands 42 and 37. For a human being to add them together, he must first group them in a special way.

```
 42  (Augend)
+37  (Addend)
 XX  (Sum)
```

Note that a name is given to the operands in each of these positions. The upper operand is called the Augend, etc.

The operands are grouped in this manner because it is far easier to add one character of one operand to one character of another, at one time, than it is to add each entire operand to the other.

Knowing this, a human would look at the rightmost digit of the Augend, and the rightmost digit of the Addend and say "two plus seven is nine". While he remembered the two digits, he searched in the addition table stored in his mind for the result. He then remembered the result, and wrote it in its proper position on his worksheet.

He then performs the same sequence of operations on the next two digits (read each digit, remember it, search for the result, and write the result onto the worksheet), and achieves the sum of 79. Note that he had to perform the same basic steps twice in order to

```
 42 (Augend)
+37 (Addend)
 79 (Sum)
```

develop the sum; one cycle of steps for each digit of each operand.

A computer performs the Add in much the same way as a human being. Just as the operands and the result were stored on his worksheet, the operands and result are stored in the computer's main memory, with each digit at a specific location. Because of this, the computer must perform the same basic steps of reading and remembering a digit from each operand, adding them together, and storing the resultant digit in its memory.

Each of these basic steps is performed by status levels.

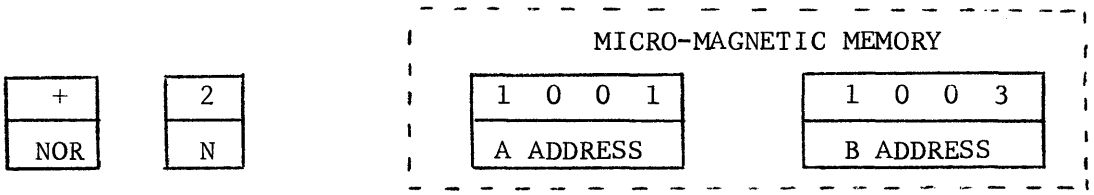
	00	01	02	03	04	05	06	07	08	09
05	+	2	1	0	0	1	1	0	0	3

	00	01	02	03	04	05	06	07	08	09
10	4	2	3	7	A	B	C	D	E	F

FIGURE 9

ADD INSTRUCTION AND PARAMETERS

Fig. 9 illustrates an Add instruction and the two operands, located in the computer's memory. After the instruction is staticized, it looks like the following to the computer. Once it is interpreted,



the computer knows it must Add two operands, each having two digits, the rightmost digit of the augend being located at 1001, and the rightmost digit of the addend located at 1003. Note that the instruction format has no third address to specify where the result is to be stored. Because of this fact, the result is stored in the same locations that the augend occupied and the augend is, of course, destroyed.

FIG. 10 - ADD DATA FLOW

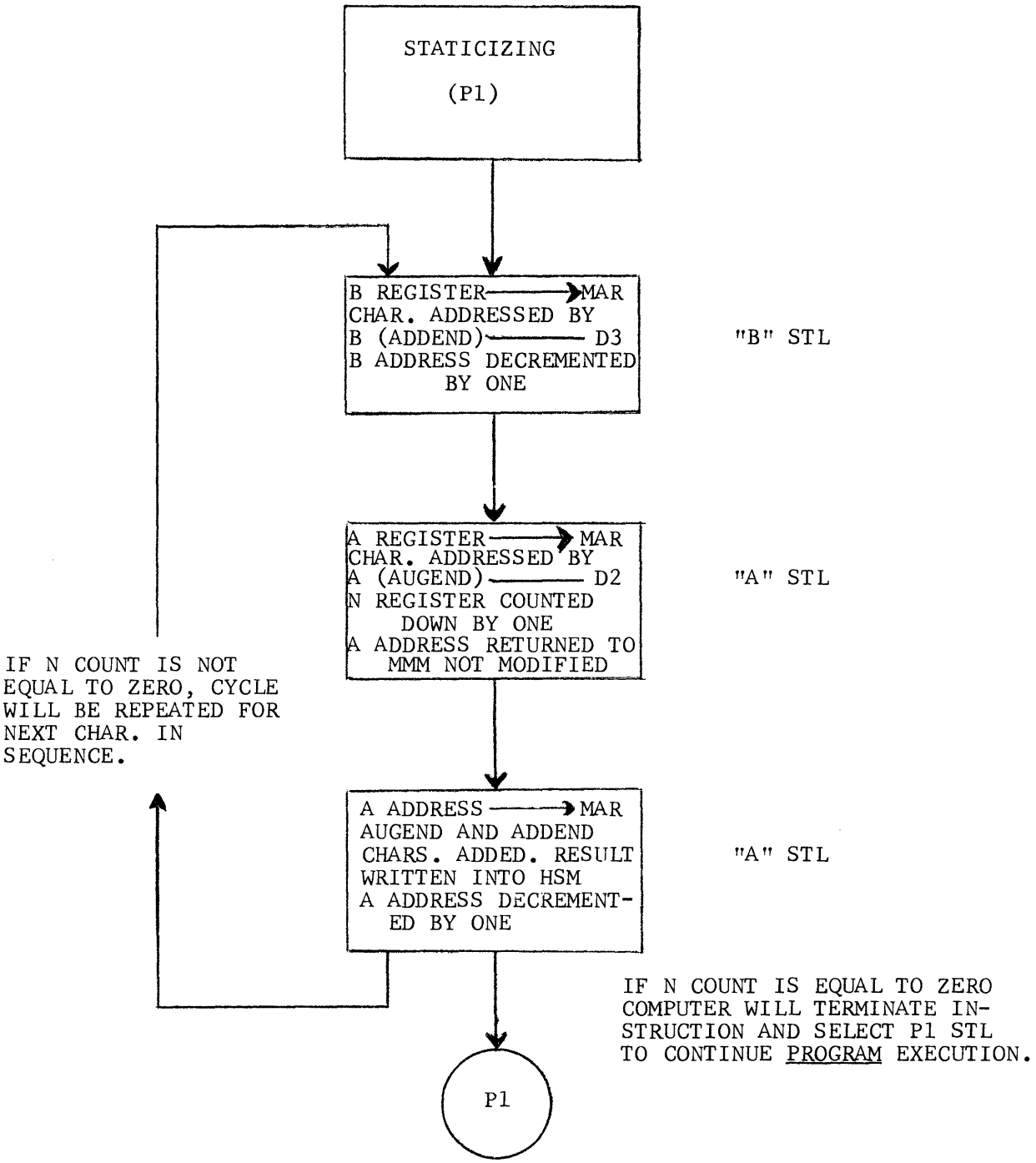


Fig. 10 is a drawing of the status levels necessary to execute a simple Add instruction. After staticizing, note that three status levels must be executed, and that these status levels perform the major operations mentioned previously.

The first status level after staticizing (always called the First Processing Level) uses the B address to access the rightmost digit of the addend and place it in the D3 register for storage. Note that, in addition, the contents of the B address are decremented by a quantity of one. This is done to prepare the address so that the next digit in sequence may be accessed if it is to be part of the operation. Fig. 11 illustrates the data flow in more detail. Note that it is called a "B" STL since it uses the B address to specify a memory location. The encircled numbers indicate the order of occurence of an operation.

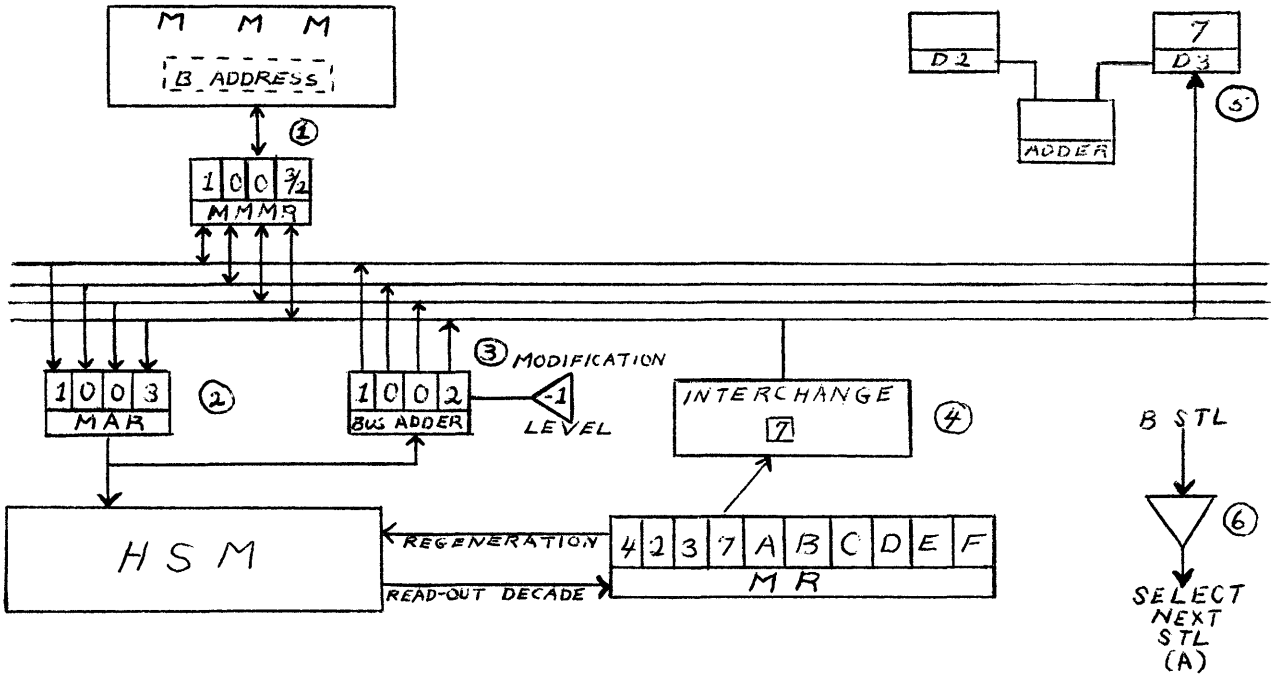


FIG. 11 - B STL DATA FLOW

Step #3 illustrates the modification of the B address by the Bus Adder. Note that this operation is performed before the desired character is placed in D3. This is done at the specified time because the desired decade is not read out of the HSM until sometime after the middle of the status level.

The bus lines are free until the decade is read out, so the hardware takes advantage of this fact in order to transfer the modified address from the Bus Adder, via the bus lines, into the MMMR for regeneration to the B address location. This is the usual procedure for any status level which is used to read a decade from memory without changing it.

Note that a very significant operation is performed at the end of the STL (step 6). The next status level is selected. This is the operation that allows the hardware to achieve its proper sequence of continuity when executing an instruction.

Fig. 12 is an illustration of the data flow through the 3301 block diagram for the A status level which was selected by the B type. Note that it uses the A address to access the necessary decade from HSM, but that no Bus Adder modification signal is generated.

Since the result must be placed in the same HSM locations occupied by the augend, the initial A address must be retained until the result is generated and written into that location, during the next status level. Though no change is to be made to the A address, the unmodified Bus Adder content is transmitted via the bus lines into MMMR, because the MMM has destructive read-out.

Once the address has been placed into its proper location in MMM, the chosen character of the decade is transmitted via the Interchange, and the bus lines into the D2 Register for storage. Note that D3 still contains the character from the ADDEND.

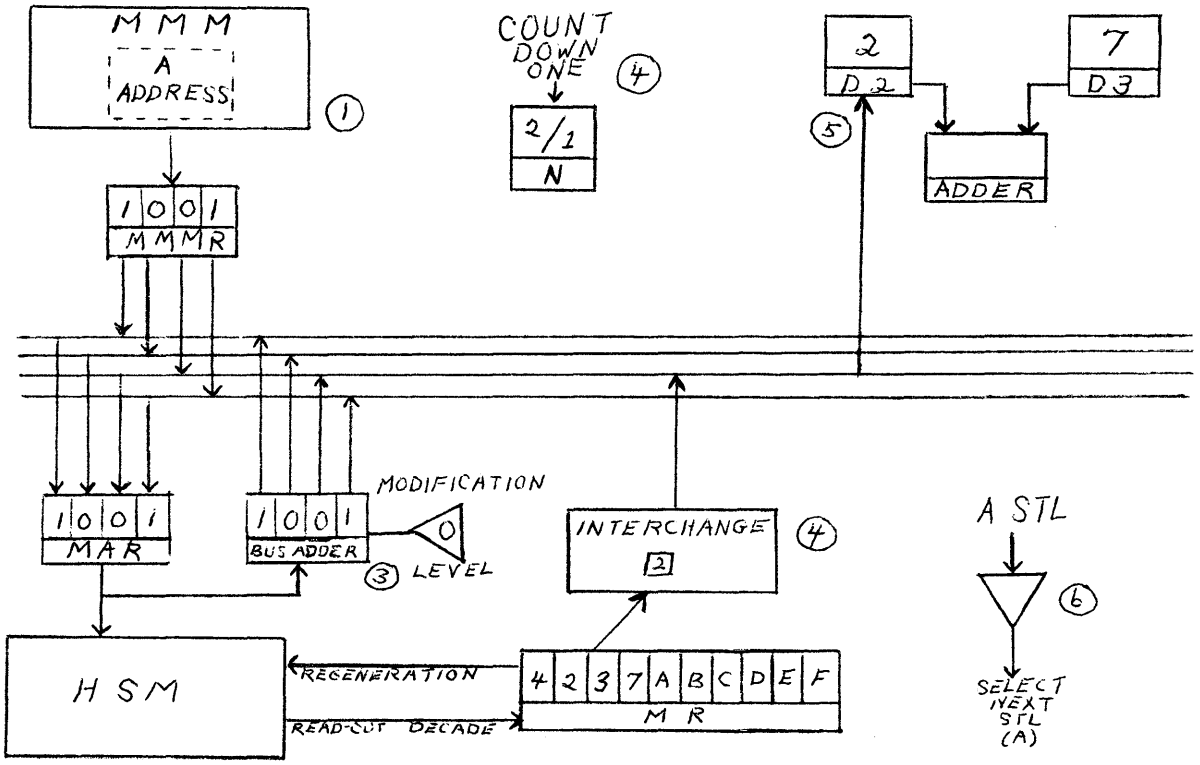


FIG. 12 - A STL DATA FLOW

The N Register content is counted down during step 4, and represents the fact that one digit from each operand is now ready to be added. When this indicator reaches a count of zero, the hardware will know that the addition of all digits of both operands has been completed, and the instruction may terminate.

Now that the D Register contains the two digits to be added, the result may be generated and placed in its proper memory location. This will be done during the third STL, which is selected just before the end of the <u>first</u> A STL (step 6).

The third status level will use the Adder, located near the D Register, to generate the result. It is a single-character decimal adder, which is wired to generate the result of the addition of any two decimal digits. In our example, because a 2 is in D2, and a 7 is in D3, the Adder will generate a result of 9.

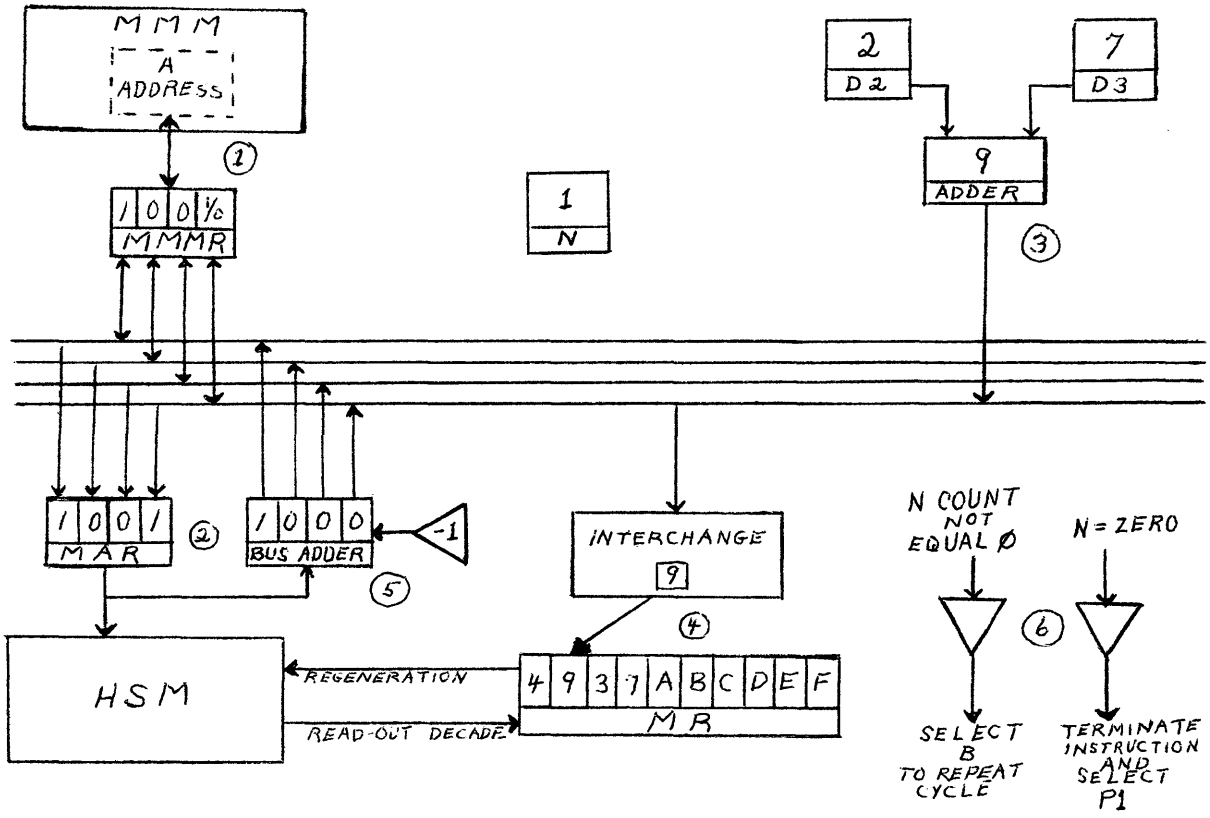


FIG. 13 - A STL DATA FLOW

Fig. 13 describes the data flow for the second A status level. Note that all the operations described in the previous paragraph are performed. The Adder generates a 9 as the result, and it is written to memory during the regeneration cycle of memory.

This process is performed by preventing the character 2 from being placed in the MR during the read-out portion of the memory cycle. The new character, 9, is then placed in its proper decade position of the MR (by the interchange) before regeneration occurs. This is the method by which any new information is written to memory. The character location to be changed is stopped from being placed in the MR during read-out, and the new character is placed in that position just before write-in (regeneration).

Note that this process occurs before the A location of MMM is modified by the Bus Adder content. The Bus Adder cannot modify a given address in MMM until the bus lines are free of any other operation; in the case of <u>any</u> status level <u>which writes</u> new information to memory, this modification of an address cannot occur until regeneration is performed.

Again, the last operation that is performed is to select the next status level in sequence. Since one character of each operand has been added, the computer must now determine whether any more characters exist in each operand, so that it may terminate the instruction or perform another cycle of three status levels.

This operation is performed by sensing the N count. Note that if the N count is not equal to zero, the computer realizes that it must perform the same cycle on the next character of each operand. Therefore, it will select a B status level in order to begin the cycle again.

If N is equal to zero, the computer knows that it has added all characters of each operand, and that it may terminate the instruction. An instruction is terminated simply by selecting a P1 status level, which will access the next instruction in sequence.

In our particular example, another cycle of three status levels would be performed in order to add the next (last) digit of each operand. Memory locations 1000 and 1001 would contain a 7 and a 9 respectively, when the instruction terminates, as the result of the operation.

Instruction execution for the majority of 3301 instructions will follow the same data flow patterns developed in this example, especially relating to use of bus lines, address updating, and memory-to-register communications.

SECTION V - 3301 SYSTEM INTERRUPT MECHANIZATION

A. Interrupt Evolution and Definition

Stored-program computing systems have always had a method by which a machine condition could cause a variation in program execution. Generally, this variation in program sequence was mechanized by utilizing a special instruction that sensed for an indicator, which was set by the machine condition.

This instruction held an address (or Program Count) of the next instruction to be executed, if the machine condition had set the sensable indicator. The Program (subroutine), located at the address the computer would jump to, would be designed to somehow handle the machine condition.

Usually, if the indicator had not been set when it was sensed for by the sense instruction, the sense instruction would simply terminate and not change the Program Count Register. This would allow the next instruction in normal sequence to be executed.

This combination of a sensable indicator, and an instruction to sense for it, gave the system the ability for hardware conditions to communicate with software operations. This communications facility enabled special exception conditions to be handled almost as soon as they occurred.

In addition to changing the Program Count Register in order to effect the jump in normal program sequence, the sense instruction had to store the address (or program count) of the next instruction that should have been performed. This was done to allow the "Exception Program" to return to the proper point (instruction) of the main program. This enabled its continuity once the exception had been handled.

This system, however, has its disadvantages. First, the machine condition cannot be sensed for efficiently unless the sense instruction is staticized often. This tends to elongate the timing of the main program, since there are usually many types of machine conditions to sense for. Therefore, many sense instructions must be included in any program.

This very fact makes it quite difficult to program a system of this type. The programmer must perform rigid timing summaries in order to efficiently place the sense instructions in effective locations of his program. He must avoid redundancy, and yet ensure that enough sensing is performed during small, seldom used loops. Timing becomes even more critical in loops where iteration is "data sensitive".

In a Real-time system, the task of programming becomes even more difficult. Real-time equipment generally sets an indicator to specify that input data is being received. The buffers which hold this data must be serviced, under program control, within certain time limits, else data may be lost due to buffer cycle timing. This causes the programmer to be very restrictive in his sensing operations for the Real-time indicator.

Where a large group of parallel communications lines are connected to a buffer interface, the complexity of the sensing instructions becomes very great. A large percentage of program time must be devoted to sensing, in order to properly service the buffers. In addition, the data transfer rates of the communications lines may vary, and the higher speed lines must be serviced more often. A priority system for buffer servicing must then be established.

Whenever any software operation must be repeated over and over again, the proper conclusion to be reached is that it should be performed by hardware, as an automatic function. Within economic limits, hardware is always designed to make the programming task more simple and versatile.

A hardware logic block may easily be mechanized to automate the function of the sense instruction.

Basically, this hardware operation will automatically sense for a machine indicator being set, and transfer control to a specific program point upon finding one set. In order to affect this transfer of control, as in any transfer of control, the content of P must be changed. In order to allow P to be changed to some pre-determined address, a register (storage location) must be added to contain the jump P address. This will simulate the transfer address of the sense instruction. In the 3301, this is done by designating locations of Micro-Magnetic-Memory to hold the address.

In a system which utilized sense instructions, to jump out of the normal program path for exception conditions, the jump was always performed after a previous instruction had terminated. This is obvious, since only one "compute" instruction may be executed at a time.

There is an added advantage given to the automatic hardware system if it does not sense the indicator and perform the jump until the instruction presently being executed has terminated. The advantage is that it may store all machine conditions of that instruction so that the conditions may be used for further processing when the computer returns to the main program. In the case of the 3301 system, when the jump occurs, the A and B addresses, standard Store #A, Store P of a Repeat, the PRI'S, Arithmetic indicators, etc. are stored in addition to the storage of the P address of the next instruction in sequence. These conditions are stored in appropriate locations of MMM.

Because this operation diverts Program Control from the main processing flow for some period of time, it is called an Automatic Interrupt System.

B. Specifications and Operation of 3301 Interrupt

Section A defined an Interrupt system as: a method by which a machine condition could affect an automatic variation in program execution sequence. This variation in program sequence enables exception conditions to be handled immediately by software and then provides for a return to the next instruction of the main program.

The Interrupt system of 3301 has been developed to a high state of the art. Since the 3301 system has both Real-time and General processing abilities, a two-level priority system has been established.

1. Interrupt Class and Priority

As Section A described, Real-time operations require great care in buffer service timing since data may be destroyed. General processing capabilites do not require this care, therefore, the 3301 has two classes of interrupts.

As you might imagine, the two classes of interrupt are:

a. Real-time

b. General

Real-time interrupts have a higher servicing priority over that of General interrupts. This means that if a General and a Real-time interrupt condition occurred simultaneously, the Real-time would be serviced before that of the General. This may be done since General conditions are usually not time-sensitive.

Each class of interrupt has several types of conditions which may cause interrupt. Fig. 14 lists the types of interrupt by class and the number assigned to their indicators.

The number assigned to the interrupt indicators is significant since it will be used to generate an address in determining which indicator had been set.

3301 INTERRUPT INDICATORS

REAL TIME	GENERAL	
01 Systems Error 02 CMC Service Request 03 Reserved for Future Enhancement 04 External Interrupt 05 Console Request	07 Arithmetic Error 08 Overflow 09 Off-Line Operation Complete 10 301 Compatibility 11 Busy or Inoperable 12 Simo 3 Terminated Abnormally	13 Simo 2 Terminated Abnormally 14 Simo 1 Terminated Abnormally 15 Simo 3 Terminated Normally 16 Simo 2 Terminated Normally 17 Simo 1 Terminated Normally 18 Program Test
06 Programmed Interrupt		

FIGURE 14

2. Basic Interrupt Operation

The basic method of interrupt operation in the 3301 system is similar to that described in section A. (Interrupt Evolution and Definition). During the execution of an instruction of the main program, running in the machine, one of the interrupt indicators may become set due to a machine condition of some type.

When the instruction terminates, instead of the machine selecting a P1 status level to enable staticizing the next instruction in sequence, special interrupt status flow is selected and generated.

The interrupt status flow will perform four operations necessary to the proper execution of the interrupt function. These four are:

a. Change the P address to that of the first instruction of the interrupt subroutine.

b. Store all machine conditions to allow proper return to the main program. Machine conditions which are stored are the final register contents of the interrupted instruction, the PRI settings, STPR, the content of the NR Register, etc.

c. Set the interlocks which prevent another interrupt of the same class or a lower priority level from interrupting the interrupt being processed.

d. Select a P1 status level to enable the interrupt routine to be executed.

When the interrupt condition has been processed by its appropriate subroutine, the last instruction of the subroutine will return the normal mode to the task of processing the interrupted main program. This is a special instruction called Return After Interrupt (RAI).

Basically, the RAI instruction will restore all machine conditions, remove the interlocks, and return the P Register content to that of the next main program instruction that would have been executed. Figure 15 (below) illustrates a flow diagram of the interrupt process.

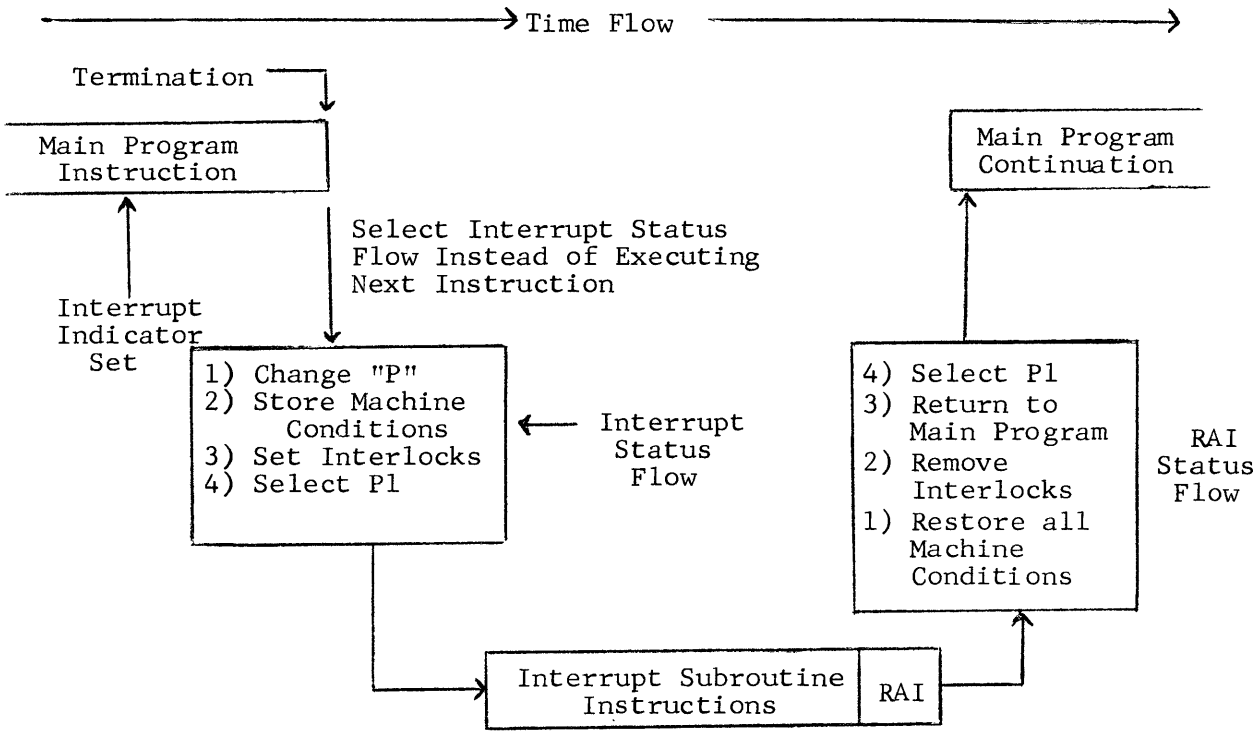


FIGURE 15

BASIC INTERRUPT PROCESS FLOW DIAGRAM

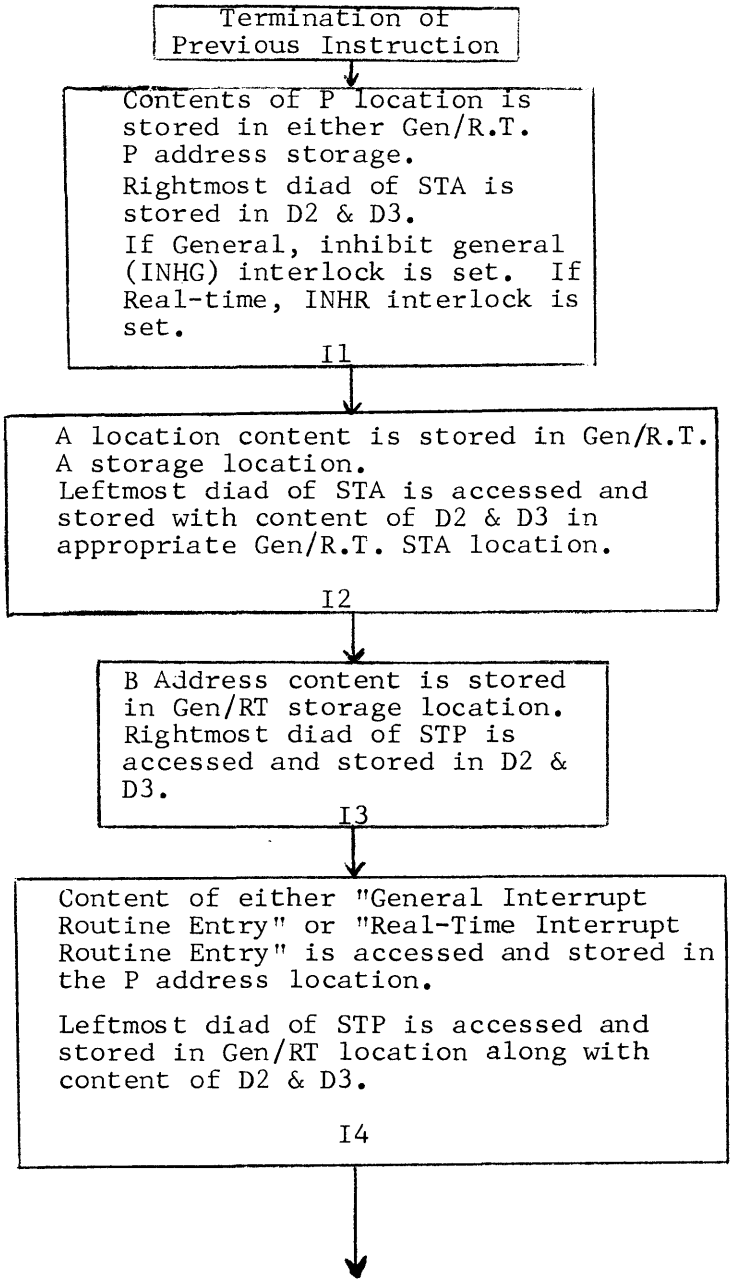
3. Detailed Interrupt Operation and Processing

Now that the "Big Picture" has been discussed, we may cover the details of interrupt processing. It is a hardware responsibility to determine when an interrupt indicator has been set, and effect the proper transfer to its appropriate subroutine, and a software responsibility to return to the main program at the desired exit point.

Figure 15 shows that even though the interrupt indicator became set during the execution of an instruction, interrupt did not occur until the instruction had terminated, and the computer was about to staticize the next instruction in sequence. This is done simply by selecting the proper interrupt status flow when an instruction terminates, instead of selecting a P1 status level.

The last operation performed during the last status level of a "compute" instruction is to select a P1 status level. If any flip-flops of the interrupt register are set at this time, however, an I1 status level will be selected and the P1 inhibited. The I1 status level is the first status level of the interrupt entry status flow.

Six status levels comprise the interrupt status flow; I1 through I6. These six I status levels require a total execution time of 9.42 microseconds since five of them require 7 time pulses and the sixth requires 9. Figure 16 illustrates the specific functions performed by each of these status levels.



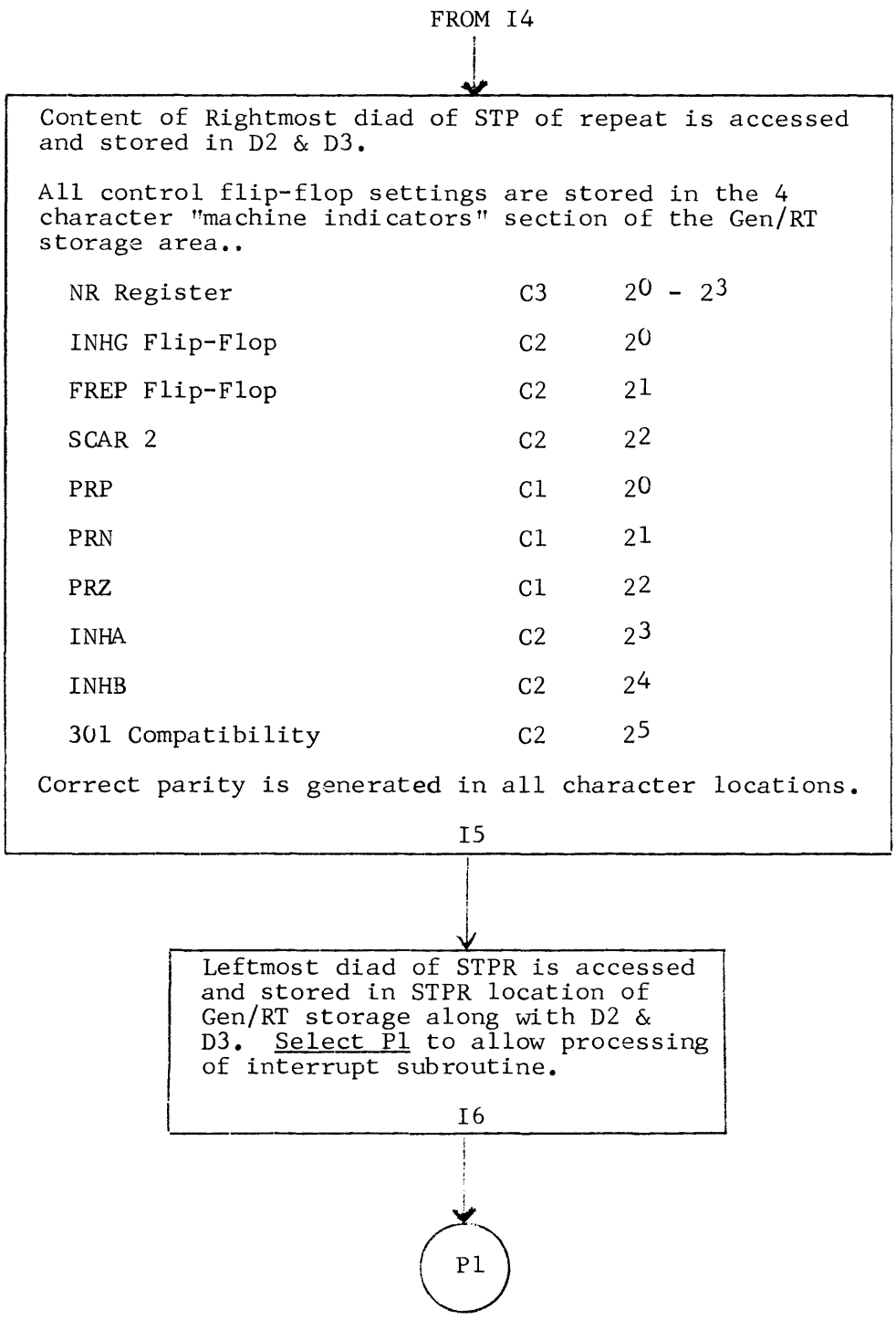


FIGURE 16

DETAILED INTERRUPT STATUS FLOW

Note that in each status level of Figure 16 the computer had to decide whether a Real-Time or General Interrupt had occurred, in order to store the machine conditions in the appropriate area of

Micro-Magnetic Memory. This decision also must be made in order to determine which of the "Entry" addresses will be placed in the P Register location.

The significant factor to note is that though 18 types of interrupt conditions exist, the hardware only determines which class of interrupt had occurred, in order to effect a transfer to the appropriate subroutine of the Executive Control System.

It will be the function of the ECS to determine which indicator (of a class) had been set before processing of the interrupt may occur. Once the specific type of indicator has been determined by the ECS, the specific cause of the interrupt condition must be determined, since several machine conditions may set the same interrupt indicator.

In other words, there are three levels of sensing to be performed in order to determine which machine condition generated the interrupt condition.

a. Hardware determines the class of interrupt, and transfers control to either the general or Real-time interrupt subroutines of ECS.

b. The ECS now has the responsibility of "scanning" the contents of the interrupt register to determine which indicator of that class was set. This is accomplished by performing a Scan Interrupt Instruction.

c. Once the specific indicator has been determined, sensing routines must be performed to determine what machine condition generated the interrupt. This is especially true in the case of I/O operations, and the FCP would have the function of performing the necessary sensing processes.

The first level of interrupt sensing (hardware) already has been described in the detailed interrupt status flow. The second level consists of the Scan Interrupt Instruction.

The Scan Interrupt Instruction (Pgs. VII-14, 3301 System Reference Manual) has the function of scanning the interrupt register to determine which indicator caused the interrupt. It scans the indicators sequentially, in multiples of six, and generates a special address constant for the first indicator that it discovers set. This address constant is stored in the standard STA location of HSM before the instruction terminates.

The indicators are scanned in the order of their numeric symbol (lowest to highest) and the numeric symbol of the first indicator found set, is used as the C1 and C2 digits of the address constant. If, for example, indicators 07 through 18 were scanned, and 11 (Busy or Inoperable) was found to be set, the address constant generated would be X110. The MSD (Most Significant Digit) of the address would be supplied by the program.

This address constant must now be used to transfer control to the specific routine which will process the interrupt condition. The simplest method of doing this is to set up a table of transfer instructions in HSM. One transfer instruction would be required for each interrupt condition (18), in order to TC to its associated processing routine. The address constant would be used as a transfer address to reach its appropriate TC instruction. This means that the TC instructions must occupy 180 locations of HSM, and the location of each TC instruction would be the address constant of its related indicator.

As an example, locations 4010 through 4180 would contain 18 transfer instructions. If indicator #12 was found set by the Scan Interrupt Instruction, it would store the address 4120 in STA. The next instruction in sequence (to the SIN) could be a Store P instruction, indirectly addressing STA, as its transfer address. Thus, the address constant could be used to allow the interrupt to be processed, with very little encoding or decoding necessary.

An option available to the SIN instruction is the ability to inhibit the scan of a certain indicator or group of indicators. This is done by setting up 18 bit positions in memory to present a one-for-one bit relationship to the 18 interrupt indicators. The 18 bit positions are set up in 3 character positions (six X three information bits for each character). This is why the interrupt register is scanned in groups of six. Each character is called an inhibit mask character.

Scanning occurs such that the 2^5 bit of the leftmost Inhibit Mask Character corresponds to the 01 indicator, and the 2^0 bit of the leftmost character corresponds to the 06 indicator. The 2^5 bit of the middle (2nd) Inhibit Mask Character relates to the 07 indicator, etc. Any one bit in the inhibit mask area will inhibit the detection of its associated, set, interrupt indicator. Scanning will continue on until an indicator is found set and not inhibited, or all indicators are scanned with none found set and not inhibited.

If no indicators are found set, the address constant remains as X000, and this, too, can lead to some specific subroutine.

A very important point is that the indicator which is discovered to be set is reset upon termination of the SIN instruction. If another interrupt should be generated during the interrupt processing, it will not be reset until a scan is performed for it.

Once the specific processing routine has been initiated, the third level of interrupt sensing may occur. Since several machine conditions may set the same indicator (especially in the case of I/O instructions), sensing instructions must be performed in order to determine which device, and specifically which machine condition, set the indicator. Normal I/O Sense instructions must be used for this function. This will be the responsibility of the FCP portion of the Operating system.

Once the three levels of sensing have been performed, and the specific machine conditon has been isolated, it may be processed by a specific routine (which may be user oriented).

After the interrupt processing routines have been completed, the return to the main program may be executed. As previously pointed out, this is accomplished by performing a special instruction called Return After Interrupt (RAI).

The RAI function is essentially an "N character" modification of the Control Interrupt Logic instruction. (Pgs. VII-7, 3301 System Reference Manual). It has the function of providing a return from interrupt processing to the next instruction of the main program. This is the next instruction, in sequence, that would have been executed if interrupt had not occurred.

The RAI instruction provides this return by accessing all machine conditions from their Real-time or General Interrupt Storage Areas of MMM, and restoring them to their proper registers, memory locations, and program indicators. It also removes the associated Interrupt class interlocks to enable the hardware to sense for any other interrupt condition which might exist. It then selects P1,

as its final operation, to allow return to the main program, since the P Register now contains the appropriate address.

The proper storage area of MMM which it will access in order to restore the proper conditions, is determined simply by sensing to determine which class of interrupt the computer had just processed. If a General interrupt had been processed, it would access the General Storage areas and place them in their appropriate normal mode storage locations, etc.

The RAI instruction performs its operations by executing six status levels in addition to its staticizing status level. These six are called X1 through X6. The first four require 9 time pulses, and the last two require only seven. Including staticizing, a total time of 12.63 microseconds is required for execution. The six status levels perform basically the same operations as did the I1 ——→ I6 status levels, except that they access their data from the appropriate storage areas of MMM and place their information into the normal mode registers and HSM storage areas. Any contents of STA, STP, STPR from interrupt processing will thus be destroyed.

The last status level, X6, selects a P1 to allow normal processing of the main program. Note that if another interrupt indicator had been set during processing of the initial interrupt, it would have been prevented from interrupting by the interlocks. Since the interlocks are removed by the RAI, interrupt may occur again as soon as the X6 attempts to select the P1. An I1 would be generated instead of the P1.

Other options available through N character modification of the Control Interrupt Logic instruction, provide for setting several of the interrupt indicators. In addition, interrupt <u>class</u> inhibits may be set to prevent an interrupt of that class from affecting the main program, or when performing debugging or testing procedures. In these cases, the indicator will be set by the interrupt condition, but interrupt status flow is not executed until the inhibits are removed. In fact, the inhibits are nothing more than the interlock flip-flops INHG and INHR, which are set by an interrupt condition.

4. <u>Multi-Level Interrupt Processing</u>

In an earlier segment of this write-up, the priority of the interrupt classes was described. Real-time interrupts have priority over that of general interrupts, and both interrupts have priority over normal processing.

Just as a General condition may interrupt normal mode processing, a Real-time condition may interrupt General interrupt processing. If a General interrupt is being processed, and a Real-time interrupt condition occurs, the Real-time condition will interrupt the General at the end of the instruction currently being executed. The Real-time condition will be processed, and the RAI will return program control to the General interrupt processing routine. Figure 17 illustrates a flowchart of this process.

Figure 17 shows normal mode processing interrupted by a General Interrupt condition. The normal mode machine conditions are stored in the General storage area of MMM, and General processing begins.

General processing is interrupted by a Real-time interrupt condition, and the General processing machine conditions are stored in the Real-time storage area of MMM. After Real-time processing is complete, the RAI instruction (R-T process) returns program control

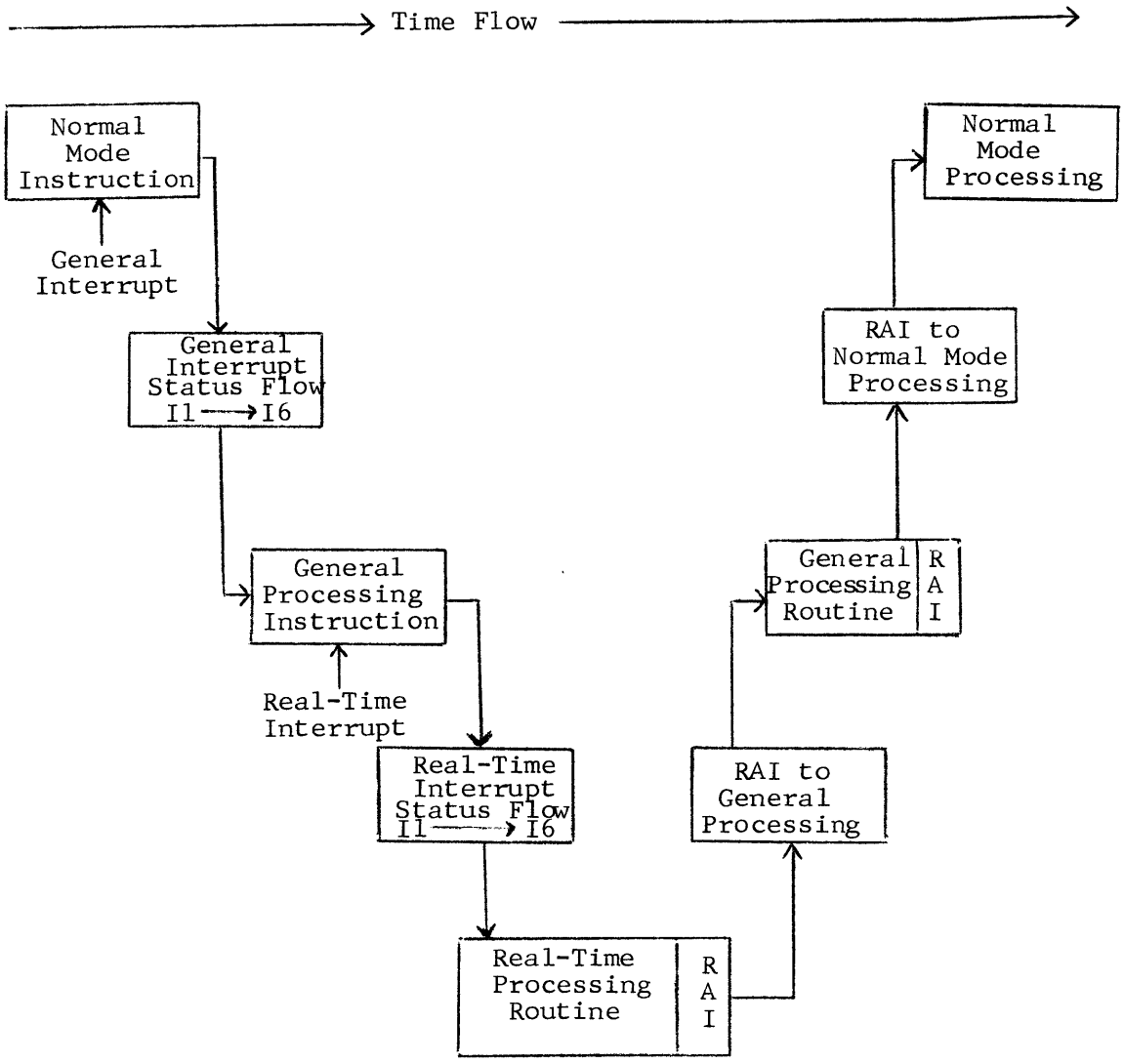


FIGURE 17

MULTI-LEVEL INTERRUPT PROCESSING

to the General process routine simply by accessing the General machine conditions from the R-T storage areas of MMM. Once the General process is complete, the RAI (General process) restores the normal mode machine conditions from the General storage areas of MMM.

The two storage areas of MMM are thus necessary to allow Multi-level interrupt processing to take place. Note that due to the interlocks involved, an interrupt of the same class, presently being processed, cannot immediately cause interruption.

SECTION VI - 3301 SYSTEM PERIPHERAL BASICS

A. BASIC PERIPHERAL CONTROLS AND DATA FLOW

A stored program computing system performs all processing operations upon data which is stored in its main memory. This data must be accessed from main memory and placed in some operational logic block in order to be processed. Of course, the instructions which direct the processing are also stored in main memory.

The size of main memory is usually limited by economic considerations, and all the data necessary to the solution of a problem cannot be stored at one time. HSM usually does not contain enough locations to hold both the entire data file, and the program, in order to solve a problem.

This means that the parameter data, as well as the program, must usually be segmented. It must be divided into logical portions, and placed in memory only when it is needed to solve a specific portion of the problem.

Once the result of each portion of the problem has been derived, it must be stored. It cannot be stored in main memory since it would then occupy area necessary to the solution of the next portion of the problem. In addition, a user will usually require access to the results.

The result is generally stored on some storage medium external to main memory; on a peripheral device such as a printer, or a tape station, etc.

In either case, devices are required which will act as an input storage or output storage medium. These are the peripheral devices which surround a processor, and whose function is basically to act as an extension of main memory; for storage, display, or input.

Data may be stored on many media, the two most popular media being paper (punched holes) and magnetically-sensitive surfaces (magnetic field variations). The devices which handle these storage media take many forms due to considerations of desired speed of data access versus economics. The most commonly used devices are magnetic tape stations, card readers and punches, paper tape readers and punches, disc file storage units, and magnetic card storage units.

A discussion of the simplest of these devices is in order to enable an understanding of the basic concepts inherent in all.

The simplest of these devices stores its data in the form of coded holes on a strip of paper tape -- the paper tape reader.

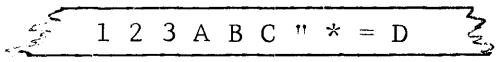


FIGURE 18

STOCK TICKER TAPE

Fig. 18 illustrates a strip of paper which has information printed onto it. One character is printed on it at a time by a device which operates similarly to a typewriter. As the information is being printed (one character at a time) the strip of paper must move from the right to the left, one character space at a time.

Each character occupies a definite position in the line of information on the tape, and this position is given a name. It may be called a "character frame". Because of the fact that each character frame is of

equal size, only a given number of characters may be placed, per inch, along the length of the tape.

The number of character frames per inch is an important measurement of tape recording, and is called the packing density of the tape. If ten character frames occupy one inch of tape, the packing density of that tape is ten characters per inch.

In order that a human being may make use of the information printed on the tape, he must read it. In reading it, he places the data contained on the tape into the storage areas of his mind. He may then make use of this data in solving some problem.

The process of "reading" printed information is a very complex function in a human being. It can be mechanized electronically, but is a very expensive process. It is far more simple and inexpensive to mechanize the reading of data, which exists in a form more similar to the language of the machine.

This can be accomplished by representing each printed symbol on tape by a coded symbol, more easily "understood" by the machine. Fig. 19 illustrates how a printed symbol may be replaced by a coded symbol, recorded by means of punched holes on the tape.

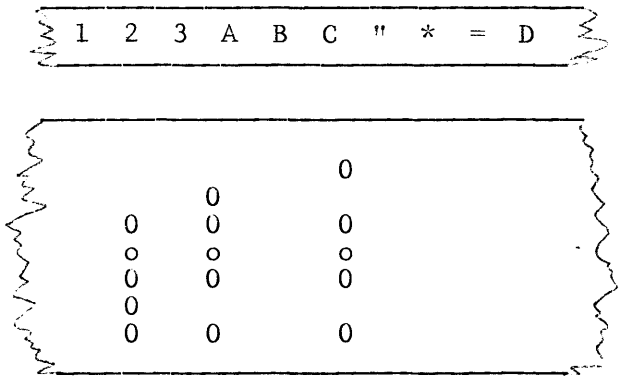


FIGURE 19

PRINTED AND CODED TAPE COMPARISON

In order that a computer may "read" the data encoded on the tape into its memory, some device is necessary to detect the presence or absence of holes in the paper tape. This device must be able to "translate" the configuration of holes in the tape into the electronically coded signals understandable to the computer.

Certain semi-conductor devices are able to sense the presence or absence of light, and are called "photo-diodes". A photo-diode is a semi-conductor element enclosed in a small tube, having a lens (to focus light) formed on one end. The two leads from the diode element exit at the opposite end. Fig. 20 is an illustration of a cut-away view of the photo-diode.

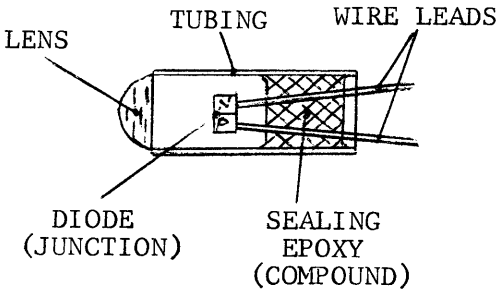


FIGURE 20

PHOTO-DIODE CONSTRUCTION

Essentially, the junction point of the two semi-conductor materials (N and P) in Fig. 20 has an electrical resistance (which tends to limit the flow of current in a circuit). The value of this resistance will vary when light is focused on the junction, and allow a greater amount of current to flow. If the varying current from the photo-diode is amplified and shaped by a special circuit, an output pulse can be produced when light is flashed on the photo-diode.

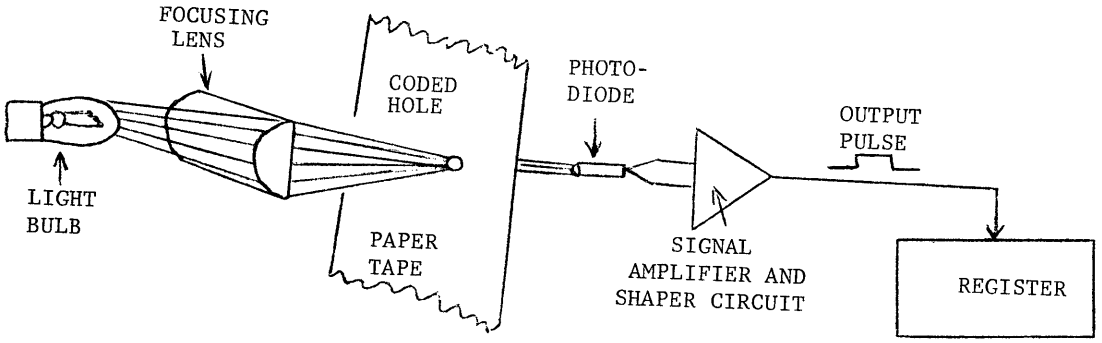


FIGURE 21

Figure 21 illustrates a basic circuit to enable a bit to be placed in a register, when a hole is sensed in paper tape.

This circuit can be expanded to include provisions for sensing the entire bit configuration of a coded character by incorporating seven photo-diodes, signal shapers, and register circuits; one for each bit (coded hole) of the character. Figure 22 illustrates the expansion of the reading station for the paper tape reader.

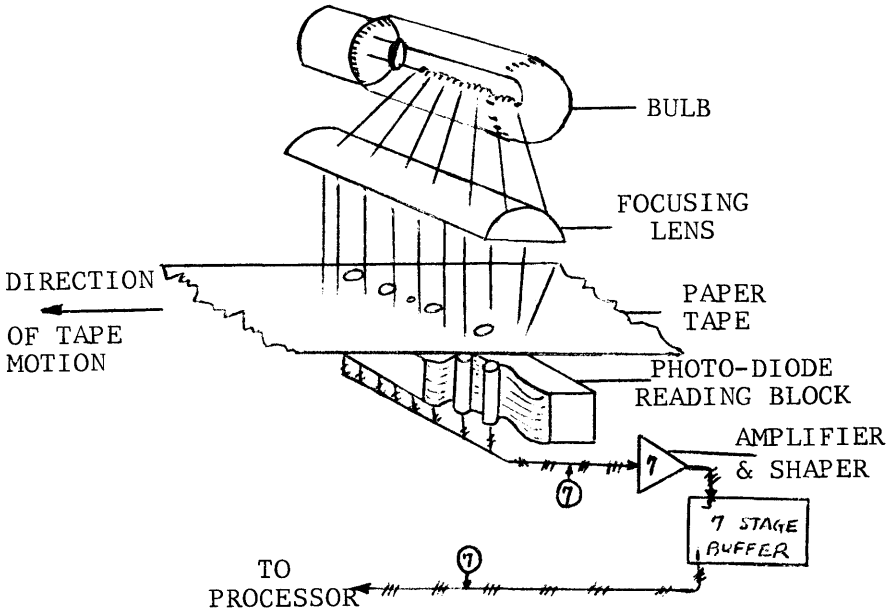


FIGURE 22

The entire purpose of mechanizing a scheme to enable "reading" coded information from paper tape is to enable the information to be placed (transferred) into the processor memory. In order to enable this function to be mechanized, the operation of memory must be considered in some detail.

In order to place (write) information into memory, it is necessary to perform a memory cycle. This memory cycle will consist, of course, of the three operations ADDRESS LOCATION, READ OUT, REGENERATE (write-in). These three functions are detailed in the section pertaining to memory operation. To write data into memory, then, the old character, occupying the location to receive the new character, must be destroyed during read out. This is accomplished by preventing it from being placed into the Memory Register during read-out.

In order to "write-in" the new character, it must be placed into the corresponding stage of the MR before regeneration occurs. Regeneration will then record the character in the specified location.

To enable a character (read from a storage medium) to be written into memory, requires that the character be transferred from the Buffer in the I/O device and into the MR during a memory cycle.

In order to synchronize the timing of the memory cycle and when the character (to be placed in memory) exists in its device's buffer, some form of synchronization control is required. All I/O devices, at the present state-of-the-art, read characters into their I/O buffers at a much slower rate than main memory can generate its cycles. To prevent main memory from generating a useless sequence of cycles during the time that no new character is to be stored, the most efficient method would be to allow the presence of the character in the I/O buffer to permit a memory cycle to be generated.

The character's presence in the I/O buffer must somehow be detected by the synchronizing control logic. There are usually two methods of accomplishing this function. One method commonly used is to record a special indicator in each character frame on tape, as well as the bit code of that character. This indicator may be read, along with the coded character, and used to develop a timing signal. The timing signal indicates to the main memory that a cycle should be generated.

Another method used is to detect the presence of any one of the coded bits of the character. This is accomplished simply by looking at all the bits of the I/O buffer, in parallel, and, if any stage is set, it is assumed that the character has been read.

In either case then, a signal is developed which specifies that a character has been read into the I/O buffer and the buffer requires servicing. For this reason, the signal developed from detecting the presence of a character is called a Buffer Service Request. This Buffer Service Request now has the function of telling the processor that a memory cycle may be generated.

In the case of a paper tape reader, the timing indicator which is recorded on tape is the Sprocket hole. It is read (along with the bits of the character frame) and stored in a single register of its own. The output of the register may then generate the Buffer Service Request (BSR). Figure 23 illustrates the generation of the BSR.

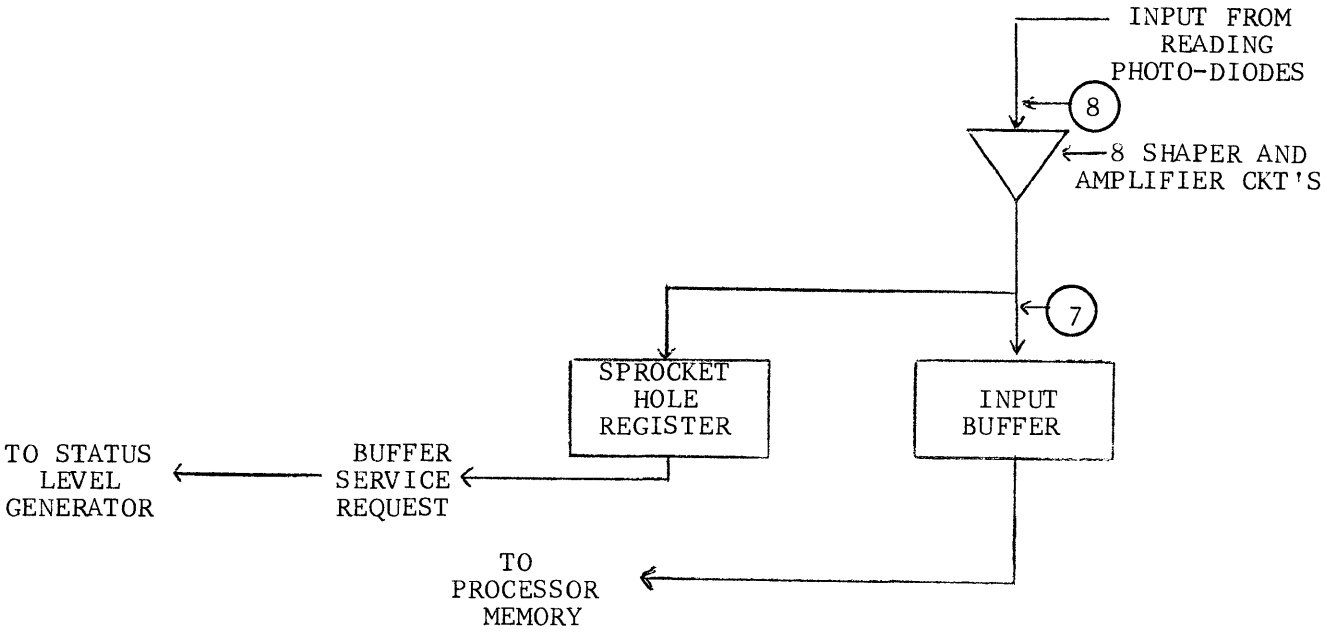


FIGURE 23

Normal "Compute" type instructions require a constant flow of status levels; one immediately following the other. In the 3301 system, these are generated by means of selecting the next status level to be generated by means of selecting the next status level to be generated during the last time pulse (TPC8) of the preceeding STL (status level). The presence of the next time pulse, TPC enables the next STL (the one that has been selected) to be generated.

Thus, this constant train of STL's may be interrupted, when desired, by the simple measure of allowing a previously selected status level to be generated when a BSR exists in conjunction with TPCØ. The 3301 STL generator does, in fact, have this provision. This enables a status level to be generated during the execution of an I/O instruction only when 9 characters have been recognized in the I/O device's buffer.

Figure 24 illustrates the synchronization and Data flow paths which exist in the 3301 system, as previously described.

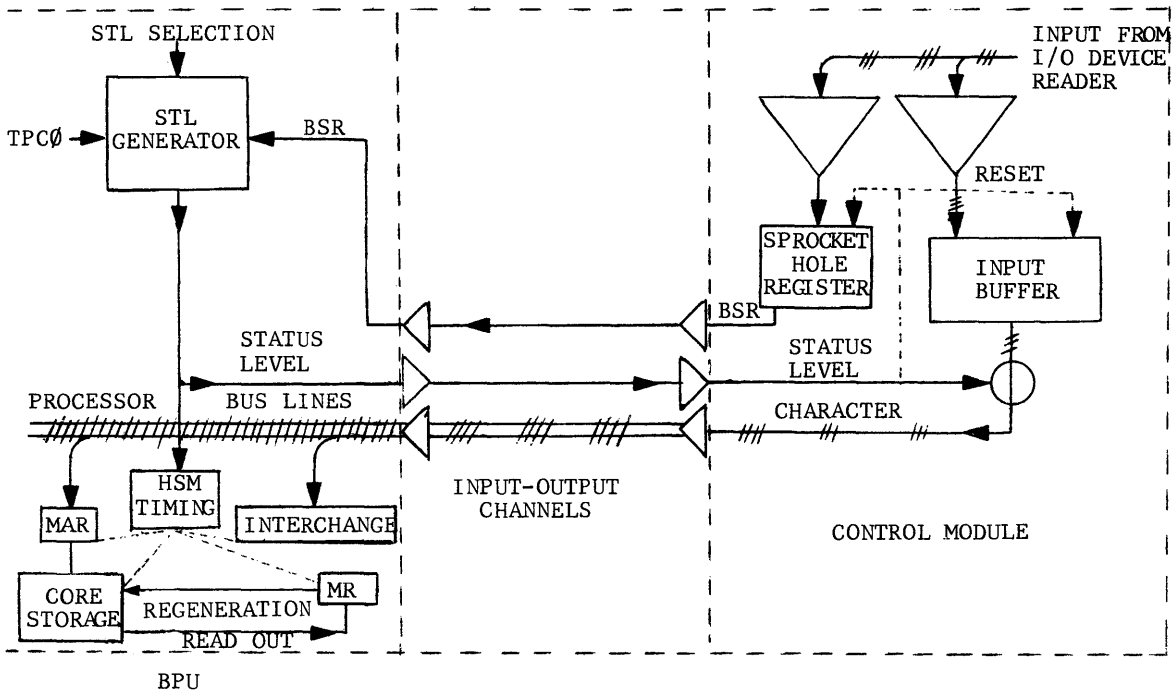


FIGURE 24

SYNCHRONIZATION OF DATA FLOW

The final sequence, then is that the staticizing of the instruction would select but not generate the status level required to transfer a character from the I/O buffer into memory. Once the first character had been read into the Input Buffer, its Sprocket Hole signal would generate a Buffer Service Request. The BSR would start to generate the Status Level at TPCØ and the STL would initiate the required memory cycle. The STL also would be sent to the I/O Control Module to transfer the contents of the Input Buffer onto the Input/Output Channels, onto the Processor Bus lines, and then into memory. During regeneration, the character would be placed in core storage from the MR. The last function of the Status Level would be to select the next STL in sequence and reset the Sprocket Hole Register and Input Buffer to prepare these to receive the next character in sequence.

Of course, the selected STL would not be generated until the next character had been read.

To give an example of the timing that would exist, if the data transfer rate of the paper tape reader was 1000 characters per second, this would indicate a time of 1 millisecond between characters being placed in the Input Buffer. Thus, only one STL would be generated every millisecond as Figure 25 illustrates.

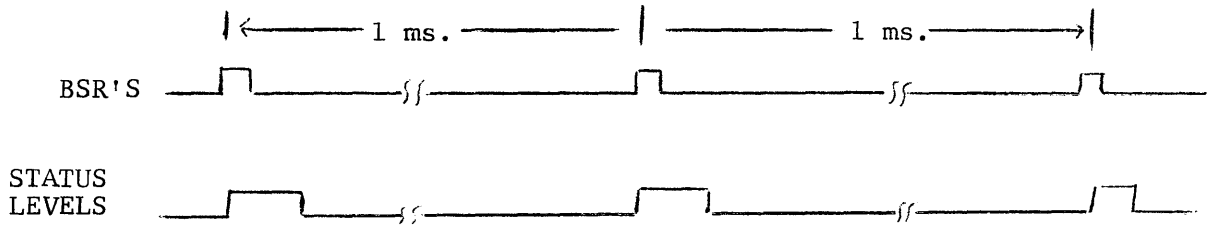


FIGURE 25

STATUS LEVEL AND BUFFER SERVICE REQUEST RELATIONSHIPS

Figure 24 also shows something which has not been previously described: the Input/Output Channels (I/O Channels). The I/O Channels exist in the 301/3301 systems for an engineering design reason, and have no relation to the number of I/O devices which may be connected to a Basic Processing Unit (BPU).

The design reason relates to the distortion of signals which occurs whenever pulses are transmitted over any great length of wire. The major function of the I/O Channels is simply to act as an extension of the Processor Bus lines from the main frame to the I/O control module.

Since the I/O device is usually situated at some distance from the main frame, the data and command signals transmitted between the two will receive a great deal of signal distortion between the transmission and receiving points. This distortion can lead to gross inaccuracies in data transfer.

Compensating circuits are placed at the two ends of the I/O Channels to correct for this distortion. Figure 26 shows the two types of compensation networks and what the pulses would look like at significant points along the data path.

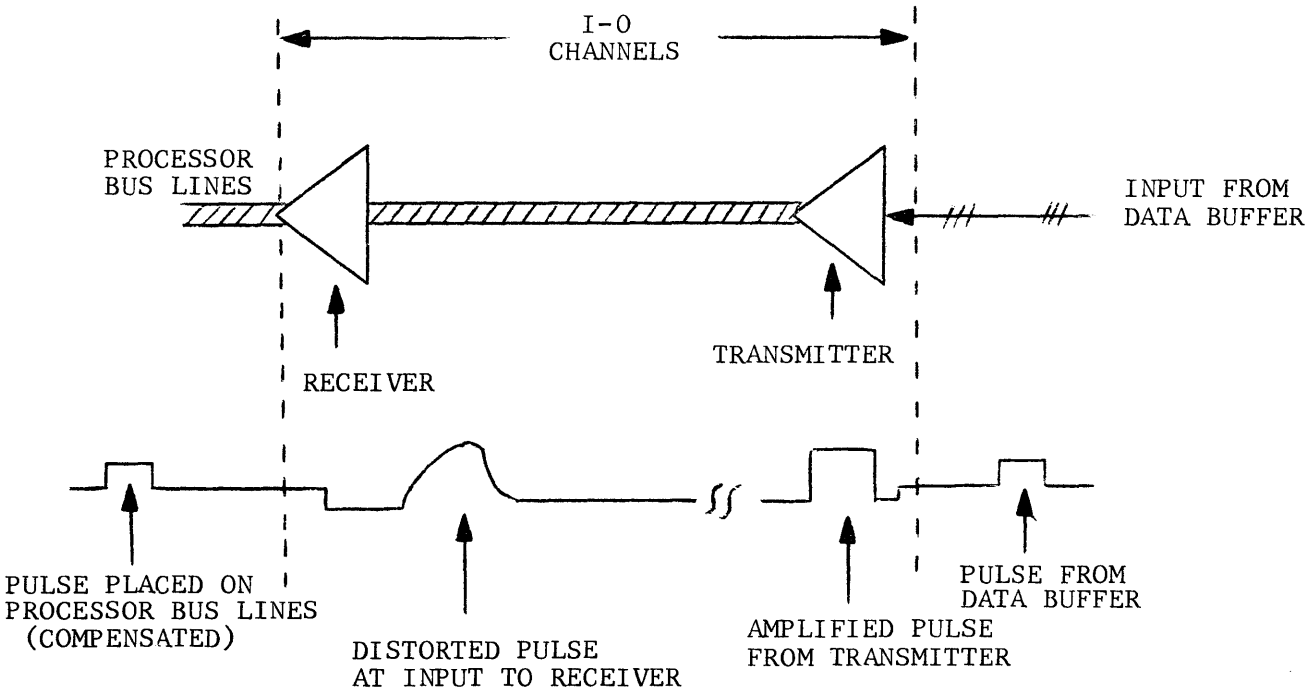


FIGURE 26

In summation then, the I/O channels actually perform two functions; (1) that of acting as an extension of the processor bus lines for the purpose of data and commands transmission, and (2) compensation of pulse distortion along the transmission paths to ensure greater accuracy.

It will be discovered that a typical I/O control module has two other functions including the major function we have considered. The three functions it performs are those of:

a. Synchronization of timing for Data Flow.

   (1) Includes code translations

b. Motion Control of the Storage Medium.

c. Accuracy Control of Data and Commands.

Motion Control relates to the necessary movement of the storage medium in order to transfer the appropriate block of data. Figure 27 illustrates the components that must exist to start tape moving On Demand and to stop it from moving when desired.

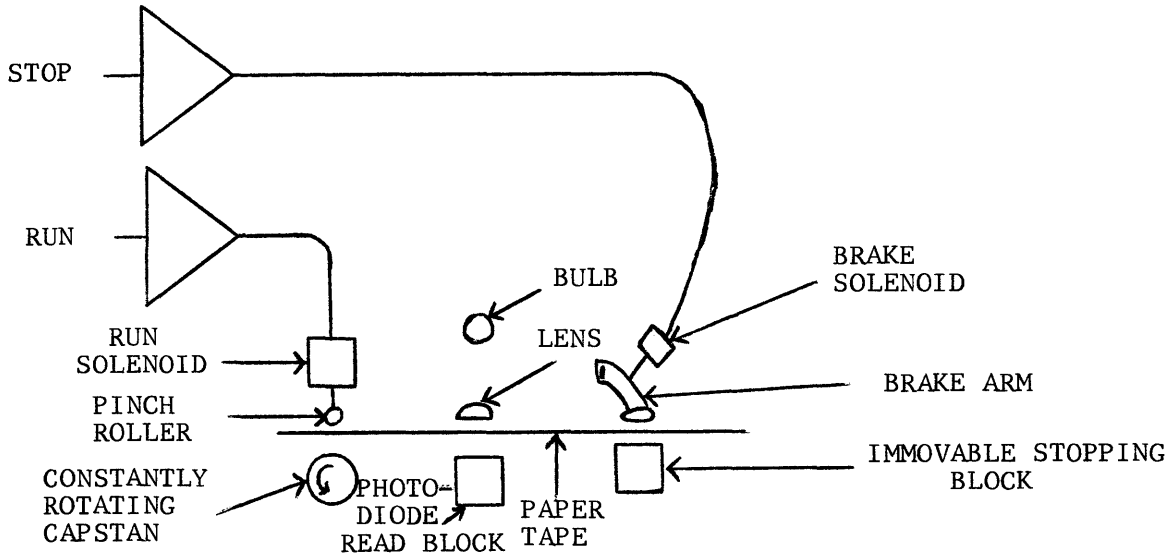


FIGURE 27

PAPER TAPE READER MOTION CONTROL

Illustrated in Figure 27 is the basic reading assembly of a paper tape reader surrounded by its start and stop mechanizations.

The paper tape movement assembly consists of a constantly rotating capstan which will be used to drive tape forward at its rated speed. Since the capstan is constantly rotating, all that is needed to impart motion from the capstan to the tape is to create friction between the two. This can be accomplished by use of an electrically-operated pinch roller. When the RUN command is transmitted to the Reader (by the initiation of the Read instruction) it will energize the pinch roller solenoid, forcing the tape against the capstan. The tape then will begin to move. Some time is required, however, to cause tape to accelerate from zero speed to its rated movement. This time is called Up-to Speed time.

Once all desired data has been transferred and a gap on tape has been detected, the tape motion must be stopped. This is accomplished by a Brake Arm assembly operated by a solenoid. A metal "Stop Block" is

positioned below the tape, while above the tape is a pivoted Brake Arm. If the Brake Arm is forced downward, the end of the arm will force the tape against the "Stop Block" and the friction created will cause the tape to stop. At the end of the Brake Arm is a rubber shoe, to prevent damage to the tape, and to create a higher coefficient of friction to stop the tape faster.

The control module, by recognizing the instruction initiation and the existence of a gap on tape, generates the desired RUN and STOP commands. Thus, it serves the function of providing Motion Control for the I/O device.

The final function to be discussed is that of Accuracy Control. This will consist primarily of performing parity checks on data in the Control Module's Input/Output Buffer in order to set a program sensible indicator. In the 3301 system, when a character exists in the Input Buffer with bad parity, the output of the parity generator is used to perform two significant operations.

The first operation is to set a Read Error (RE) indicator to be used by the program in error-correction procedures. The second operation is to prevent the "bad character" from being transmitted to memory. If the character were placed in the MR with improper parity, a Memory Register Parity Error would be generated (MRPE). Instead, the parity checker will place a special error character in the Input Buffer for regeneration into memory. Each "bad" character which is read will be replaced in memory with an $17_{(8)}$, a lower case e. All control modules in the 301/3301 perform these same functions; no matter what peripheral device they operate. (The 301, however, places an $57_{(8)}$ in memory as an error character.)

RCA 3301 OPERATING SYSTEM

RCA 3301 OPERATING SYSTEM

INTRODUCTION

A. This section of the manual contains an introduction and general description of the 3301 Operating System. It outlines the advantages inherent in a modular approach to systems programming. It includes a description of the Sequence, Segment, Process and Task, and serves as an introduction to PART 3 of this manual. For details of the Operating System, refer to "RCA 3301 REALCOM OPERATING SYSTEM" (94-08-000).

The RCA REALCOM Operating System is a total software package designed to permit effective utilization of the RCA 3301 Computer. It is important to understand the reasons which have led to the development of this comprehensive package. The advent of electronic computers is causing dynamic changes in our present day economy. Significant improvements in computer hardware have permitted the wide use of computers for collecting, processing and evaluating data. In the process of solving these business problems, however, new difficulties in implementation have arisen to plague management.

1. Programming

One of the most significant problems is the complete linkage of all functions of a program at assembly or compilation time. Because of this total linkage, changes in any particular function necessitate a re-programming effort and subsequent reassembly or recompilation. A change in computer configuration normally results in a substantial re-programming effort. In addition, a change in program function (i.e., a change from serial data processing to random data processing), requires that many changes be programmed and that complete reassembly or recompilation take place. It is significant to note that in most instances the major reason for reprogramming is that all linkage within the program was formally completed at assembly time.

With the development of present day programming languages certain inconsistencies become apparent at object time. It was difficult or often impossible to create linkage for program segments whose source language was different. A system which eliminates this linkage problem and permits the collection of segments at other than assembly or compilation time became highly desirable.

Programmers today are concerned with the need to schedule the use of core memory and input/output devices within their particular program. As a result, memory allocation and device assignment are buried within each individual program. To be effective, a complete software package should eliminate these considerations from the programming level. This ability to dynamically allocate memory and devices at object time provides a computer system with maximum flexibility and operating efficiency.

As a result of these present day inefficiencies, a large portion of every computer operation involves a maintenance function. This maintenance function, in many instances, is created solely by the inclusion, linkage and allocation within each program's coding.

2. Operations

The task of an operations group is becoming more difficult due to the complex uses being made of a computer. A lack of standardization in operating procedures requires that voluminous and detailed documentation be produced and maintained in order to operate each program. In addition to requiring substantial operator training, the need to refer to this material places a burden of efficient use of the computer.

The problem of controlling and maintaining adequate documentation at running time places an additional workload on the operator. He must

maintain detailed logs and reports concerning the operation of each program within the system.

The scheduling of computer operations in itself requires that the operator have detailed descriptions as to the sequence in which programs are to be run. As each program terminates, he is concerned with loading the next program in sequence and proceeding with the operation.

3. Management

The cost of these problems presents management with serious difficulties in initiating, maintaining or re-evaluating any computer application. The advantages of a particular change in function or computer configuration must be weighed against the cost of the reprogramming effort involved. In addition, substantial review of all program steps involved is required to determine the effects of a change in any particular function. A computer application whose functions can change dynamically is linked today with programs which themselves cannot change dynamically.

As a result of these factors, management is burdened with many daily problems and decisions created by today's programming and operations limitations. These requirements limit management in its prime duty of developing and evaluating new applications to be integrated into the computer system. The major cost to a firm with these conditions is the inability to effectively utilize the computer system in serving the present and future needs of the firm.

B. The RCA 3301 is based upon a design concept called "functional modularity." This concept permits growth, as applications require, by the incorporation of new units or "modules" that add not only speed and unit capacity but functional capabilities as well. This open-ended expansibility and growth design enables the RCA 3301 to present the right equipment configuration for any digital computer application.

The concept of functional modularity can also be extended to RCA 3301 user applications. All computer applications can be divided into individual units, each of which can be implemented independently, without burdening the programmer with the responsibility of fitting each unit into the overall problem solution. In other words, a user may program individual units of an application without the need to tie the units together at the coding level. This technique permits the application units to be considered as independent "building blocks" that can be grouped together to form larger blocks on the basis of user descriptions supplied to various RCA 3301 software packages. Consequently, it is possible to assign the coding responsibility to one job level while assigning the integration of the coding units to a higher job level.

Four levels of building blocks are provided in the 3301 modular programming system; i.e., sequence, segment, process, and task.

1. The Sequence

The basic building block of the REALCOM system (i.e., the most detailed level) is called a "sequence." Technically, a sequence is defined as the "unit of logical manipulation." Stated in other words, the sequence is the smallest logical unit of an application which can be considered independently of the other logical units of that application. Using more familiar terminology, a sequence may be thought of as a "subroutine"; however, since subroutines are not the only basic units required in an application, the more inclusive term is used. Within individual sequences, the user may provide not only programming logic (i.e., subroutines), but may also describe data files, working storage areas, and constants. To take a payroll application, for example, the routine that computes an employee's gross pay can be considered as one sequence whereas another routine that computes an employee's income tax deduction can be another sequence. The individual user can decide how large or how small his logical units should be. The choice largely depends on the amount of freedom desired in implementing an application.

2. The Segment

Sequences in the modular system are organized into operational units called segments. The segment therefore is the next higher level above the sequence and is defined as the "unit of loading and execution." The segment is usually interpreted as the portion of an application which fits into memory at any one time. In the RCA 3301 system, this concept is expanded to include a listing of all the logical units (e.g., data files as well as subroutines) required at any one time. A segment therefore can be considered as a user building block requiring a certain amount of hardware for execution purposes. The hardware in this case can include core memory, external (random access) memory, input/output devices, etc. and is assigned to the various sequences by the RCA 3301 Operating System. A segment is defined by the user through a "segment description" which provides:

a. A list of the sequences in the segment, and

b. A specification of the manner in which control is to be transferred between sequences.

The segment description thereby supplies the linkage from sequence to sequence as opposed to coding the linkage within the sequences themselves. In addition, the segment description provides the means by which the Operating System can make specific hardware assignments at execution time.

3. The Process

The next building block above the segment is called the process. From the user's viewpoint, the process affords a level for grouping together related segments. Since each segment represents a unit of execution, the process provides the means for describing the order in which segments are to be executed. This is accomplished by a "process description" which specifies:

a. The segments which comprise the process

b. The order in which segments are to be called during execution

c. The information to be passed from one segment to another

d. The manner in which linkage is accomplished between segments

A process can be considered as an operational unit since the RCA 3301 Operating System will load segments as specified in the process description without operator intervention. Technically, the process is defined as the "unit of scheduling and allocation" for the RCA 3301 Operating System. When a process is to be initiated, the Operating System determines whether or not sufficient hardware is available for executing the process. If so, tentative assignments are made and the process is started. Final hardware allocation is performed as each segment is loaded.

4. The Task

The highest level in the RCA 3301 building block system is called the task. This level provides the user the facility for listing a number of processes to be executed as an operational unit. These processes may be related to each other (in the case of a large application) or may be unrelated for the user who desires automatic execution of a series of independent operations. The task is defined as the "unit of external priority." In other words, this is the unit with which the operator and the Operating System are concerned. Regardless of the number of processes comprising a task, and the complexity of these processes, the only subjective operation required at the console is task

initiation. All subsequent actions required during the execution of the task (e.g., tape reel changes) are directed by the Operating System via messages on the Console Typewriter.

Note that the term "program" does not appear in the hierarchical structure previously described. In the RCA 3301 system, a program represents a unit of assembly; not a unit of execution. This distinction permits the user to separate the function of translating source coding from the function of machine code execution.

An outstanding feature of modular programming is the ease with which an RCA 3301 application can be modified. The serial processing application, for example, can be modified to an on-line, random access application by modification of only those sequences containing serial processing logic. In many cases, modifications or enhancements may be made by changing only the user supplied descriptions (i.e., regroup existing building blocks). Modularity in hardware can now be matched by an equally effective modularity in application design and implementation so that new equipment and new techniques can be employed by users without requiring a significant reprogramming effort.

Functional modularity is also the basis for the design and implementation of the RCA 3301 software. In the RCA 3301 Operating System, for example, each major function is designed and implemented as an independent building block. Upon initiation of the Operating System, the segments necessary for the performance of desired functions are automatically selected from the software library, thereby allowing the system to be tailored for each specific application.

Note: See the application example utilizing modular programming techniques on page II-6.

C. The nucleus of RCA 3301 software is the Operating System which co-ordinates and controls the execution of computer tasks. In performing this function, the Operating System assumes the responsibility for controlling the complete physical environment of the computer. The user, on the other hand, has the responsibility for supplying logical solutions to his tasks. This division of responsibility eliminates user concern regarding the physical characteristics of the computer, thereby reducing his total programming effort.

1. Objectives

The objectives of the RCA 3301 Operating System are to:

a. Free the user from concern regarding physical conditions in the computer (input/output operations, error recognition and handling, interrupts, etc.) so that he may concentrate upon problem solution logic.

b. Provide standard routines to perform functions required by all tasks utilizing the computer.

c. Optimize the use of high speed memory by:

1) loading only those segments actually required at a given time

2) permitting common usage of Operating System segments by one or more tasks operating concurrently

3) loading infrequently needed segments only as required rather than reserving memory for those functions on a permanent basis

d. Provide 3301/301 compatibility.

e. Minimize lost computer time due to excessive operator intervention.

f. Provide standardized operating procedures.

g. Minimize set-up time in an installation.

h. Simplify the user's testing procedures.

2. Components

The RCA 3301 Operating System is composed of a File Control Processor (FCP) and an Executive Control System (ECS). The File Control Processor performs functions related to input/output operations. The Executive Control System controls the execution of both user and software programming.

The sequences comprising the RCA 3301 Operating System are not made part of each user task; rather, they occupy a portion of software reserved memory at execution time and consequently are available to as many user tasks as are being executed concurrently. This feature minimizes program translation time by eliminating the need for including Operating System sequences with every user program. In addition, enhancements and/or modifications may be made to the Operating System without requiring regeneration of user coding.

D. Implementing the Application

A user preparing to code an application has a choice of three programming languages: RCA 3301 COBOL, RCA 3301 FORTRAN IV, and RCA 3301 Assembly. Using the basic building block approach the user may code in any one system or a combination of the three languages. This means that various sequences of an application could be coded in different languages without considering linkages within the user's coding. The required linkage is generated by the programming systems, in that they provide linkage between the user's coding and the Operating System.

It should be noted at this point that the basic building block (sequence) can be defined as either a data sequence or a core sequence. A data sequence basically defines the characteristics of a data file and requires an input/output device in addition to core storage. A core sequence is a sequence requiring core storage and may consist of instructions, working storage areas, constants, and/or tables. The core sequences therefore contain the blocks of coding logic which solve the user's problem.

Usually the programmer of a core sequence need not be concerned with the contents of other core sequences. The use of exit statements at each logical termination point provides the ability at assembly time to link the sequences to form a segment. This is accomplished by a "segment description " which provides information to the programming language regarding linkage to be generated at each sequence exit. The "segment description" also defines each sequence which is to be included within the segment.

In addition, at assembly or compilation time, the user must define the logical termination points within the segment. These exits will be utilized at the next level (Process) to generate linkage between segments. Therefore, as a result of assembly or compilation, the linkage between sequences is formalized and the output prepared to be linked with other segments to form a process.

The establishment of a process is basically the process of collecting the various segments which comprise the logical solution of the problem, defining the relationship between these segments, and providing the necessary information and control blocks for incorporation within a master library tape (MLT). This collection and generation process is accomplished by a computer run known as the Process/Task Generator. By means of process description parameters the user defines the segments required, the segment

linkage, and devices required by the various segments.

The process description parameters are prepared on cards or paper tape and introduced into the Process/Task Generator along with the Assembly Output tapes containing the required segments. The process generation is performed producing the process on an output tape (MLT). Once the process is generated, testing procedures can be initiated.

Once a process has been generated and tested, the user may desire to incorporate it with other processes to form a task. The task is generated by preparing a task description. This description provides the Process/Task Generator with a list of the processes to be included within the task, linkage between processes, and device identities shared by the processes. The output of the Task Generator phase of the Process/Task Generator is a tape (MLT) containing the control information and the processes necessary to execute the task.

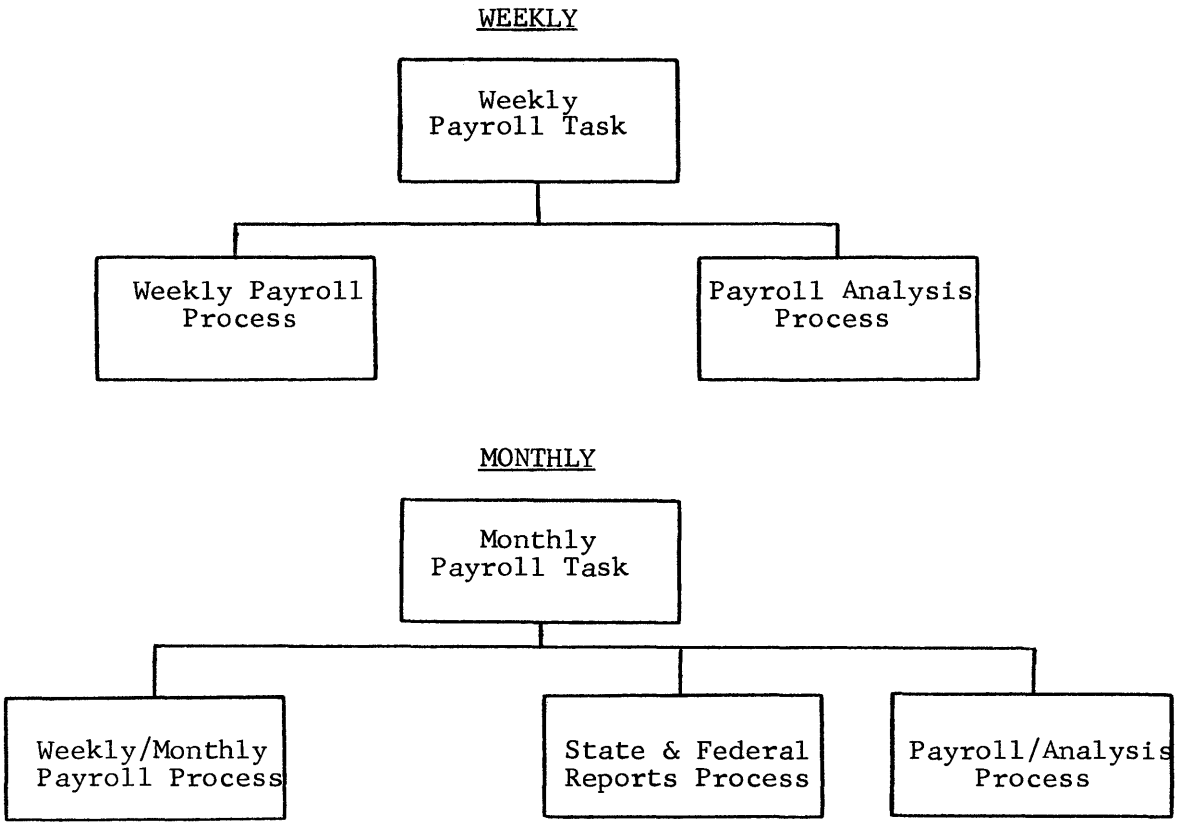
E. Operating System Usage

Reference should be made to the RCA 3301 REALCOM Operating System Manual (94-08-000) for additional descriptive material and programming information relative to the use of the individual components of the Operating System.

F. Application Example

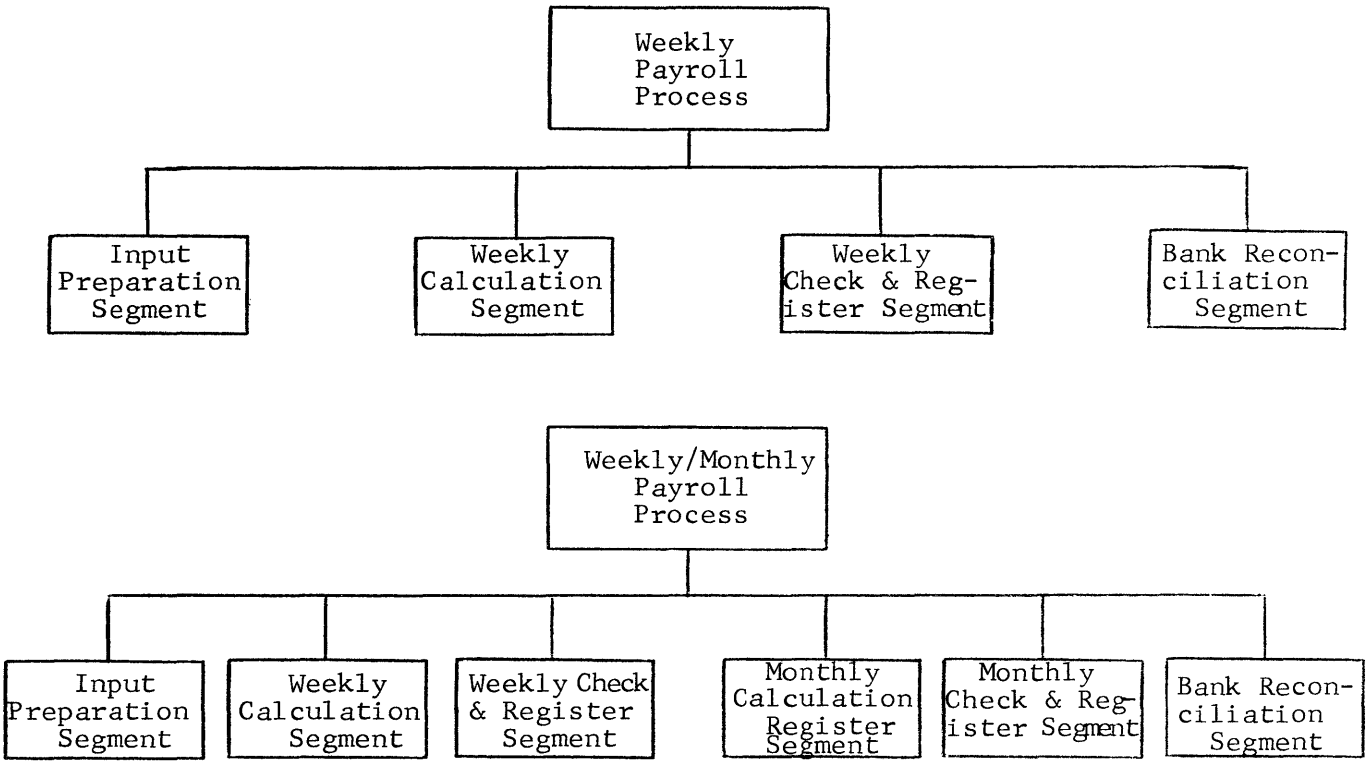
The key to a successful computer operation is a careful and detailed systems analysis prior to implementation. With the flexibility of a building block approach, the user is provided with the tools needed to most effectively perform his systems analysis. In order to briefly familiarize the user with some of the considerations that can be made, a sample analysis of a payroll application is presented below. The analysis is concerned with the techniques used to break an application down to its lowest level; i.e., the sequence. No attempt is made to present a completely accurate and detailed analysis of the particular application; the prime purpose being to describe the building block approach in familiar terms.

1. The basic operation to be accomplished is to prepare both a weekly and a monthly payroll. The monthly payroll is prepared at the same time as the last weekly payroll of that month. As analysis begins, a determination is made that two broad functions are required weekly, (1) payroll calculation and preparation and (2) payroll analysis reporting. An additional function, the preparation of federal and state tax reports, is required monthly. Two additional factors are also noted. The first is that payroll analysis reporting will not be affected by the addition of the monthly payroll while the second is that payroll calculation and preparation will be affected by the monthly payroll. The user now sees that two tasks will be required to accomplish the payroll application. These tasks are charted on the following page.



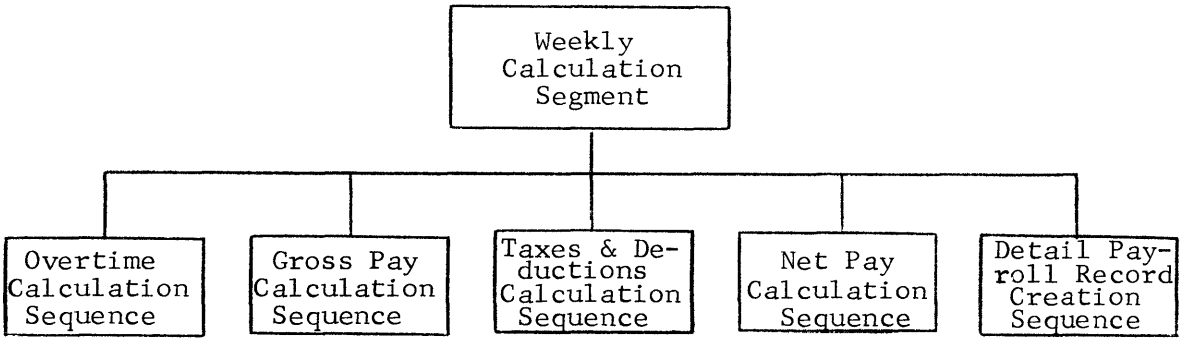
It is apparent that although two different tasks are required, a common process is present in both tasks. This process would be created once, and included in each task by means of a task description. The order in which processes are to be executed within the task would be specified in the task description, as well as the formalizing of the process to process linkage. The State and Federal Reports process would only be described in the task description of the Monthly Payroll Task. To further continue this analysis, the processes concerned with payroll calculation and preparation will be considered.

In analyzing the payroll calculation and preparation function six major areas are evident. Four areas, (1) input preparation, (2) bank statement reconciliation, (3) weekly payroll calculation and (4) weekly check and register preparation, are required in both processes. The Weekly-Monthly Payroll process will require two additional areas to (1) calculate the monthly payroll and (2) prepare monthly checks and registers. The users can now outline the segments which will be required in the payroll processes. These processes are charted on the following page.



The four segments which are present in both processes need only be programmed once. The process description will be used to include the programmed segments into the processes. Programming can proceed without concern for segment-segment linkage. This linkage will be defined via the process description.

In order to complete the analysis it is necessary to break a segment down into sequences. For this example the weekly calculation segment will be considered. In the analysis it is determined that five coding groupings are present in the segment. The segment can be charted as follows:



In addition to determining the coding sequences required, it is appropriate to consider file sequences. In this example, the user notes that the same input file will be required by the Weekly and the Monthly Calculation segments. He can require that this file sequence be described once and then be included in the segment descriptions of both segments. Each sequence in the Weekly Calculation Segment could be programmed independently. The sequences can be linked together at a later time by means of the segment description.

2. The applications breakdown presented above is a simple example of the building block approach. Using this as a basis, therefore, many of the advantages can be readily pointed out.

The first advantage is that programming effort is minimized. Once the application is completely analyzed all the sequences that must be programmed are known. Sequences can be programmed without concern for linkage or core and device allocation. These sequences can now be placed in one or more segments via a segment description. The segments which are assembled or compiled can be placed into one or more processes via a process description. The processes can again be placed into tasks via a task description. The various descriptions are the means by which linkage is formulated. Since additional programming is not needed when linkage requirements change, complete flexibility in creating segments, processes, and tasks is provided to the user.

To further point out this flexibility in linkage two examples can be considered. The first example is the situation in which the user has decided that input data would come into the Weekly-Monthly Payroll Process in sequence with weekly payroll followed by monthly payroll.

In this case he would have created a process description describing to the Operating System that the segments are to be executed as follows:

a. Input Preparation Segment

b. Weekly Calculation Segment

c. Weekly Check and Register Segment

d. Monthly Calculation Segment

e. Monthly Check and Register Segment

f. Bank Reconciliation Segment

At a point after production has begun, a request comes in asking that monthly reports be prepared first due to a higher priority. The user now prepares a new process description telling the Operating System that monthly segments are to be executed prior to the weekly segments, and changes the sort parameters to sort in a monthly-weekly sequence. The change is accomplished.

A second example is the situation where a user decides that a sort is required between two processes of a task. The task description is rewritten to include the sort, and the change is accomplished.

Another feature of the Operating System is the ability to link segments coded in different source languages into a process. It is possible, for example, that the Input Preparation Segment was compiled using the RCA 3301 COBOL Compiler while all other segments were coded using the RCA 3301 REALCOM Assembly System. No difficulty occurs when these segments are linked via a process description.

A third feature of the Operating System is the ability to assemble or compile and test a segment independently. Facilities are available to permit this individual segment testing. By this means, it is not necessary to complete all segments of a process before testing can take place.

In addition to the flexibility available when creating processes and tasks, programming changes are simpler within the Operating System. If in our examples a new employee stock purchase plan is instituted, the affected segments are readily changed. The new sequences which are required would be programmed, and the affected segments reassembled or recompiled. The new segment could be tested independently and, when

ready, included into the appropriate processes. There is no need to reassemble those segments not affected by this additional benefit. In addition, when programming the new logic, those sequences not affected within the segment are left exactly as they are and merely included in the new assembly or compilation.

When the various tasks are placed into operation, the effort required by the Operations Group is minimized. In order to run the weekly payroll, the operator need only notify the Operating System that the Weekly Payroll Task is to be initiated. At that point the Operating System would take control, and initiate and execute the task. The operator would be notified whenever external intervention is required (e.g., a need to mount a new tape, etc.).

RCA 3301 ASSEMBLY SYSTEM

SECTION I

INTRODUCTION TO THE RCA 3301 ASSEMBLY

The RCA 3301 Assembly language is oriented to the RCA 3301 computer hardware. It is a language that is enhanced by features that simplify the tasks of preparing, testing, correcting, and implementing programs.

The RCA 3301 Assembly System, used with the associated Operating System, frees the programmer of input/output functions of batching and unbatching records, simultaneity control, and the assignment of devices, to cite just a few examples.

The modular concept of programming, if used to its fullest extent, can free the programmer of such tedious tasks as linkage generation. Thus, with the proper design and definition, sequences may be utilized in more than one segment or process.

Automatic orientation of addresses is another sophistication of the RCA 3301 Assembler. Once a field has been defined with a symbolic name, use of this name will generate either the left or right-end address of the field as required by the instruction in use.

The Assembly System portion of this manual has been designed so that it may be used in two training situations. It may be used by students attending a formal training class as a text for classroom use and as a reading reference in preparation for classroom presentation of a subject. It may also be used as an explanatory reference text by persons with programming experience who are unable to attend a training class.

When used in a training course, it should be recognized that this portion of the training manual is oriented solely to the use of the 3301 Assembly System.

To the extent necessary in each training situation, the introductory part of the course should familiarize students with subjects that are fully documented in other RCA publications.

Examples of such subjects are:

History and Development of Computers
Computer Terminology
Elements and Functions of a Computer
Numbering Systems
Data Layout
Flow Charting

The Assembly System Codes presented include the controlling codes for the assembly operation, the codes used to generate proper file control, and the Operation Codes for instructions.

An attempt has been made to present these in a logical sequence such as the way in which a programmer might use them.

In using the Assembly System, for example, a programmer must first be familiar with the format requirements. Secondly, with knowledge of the system definition of input and output file formats, he would want to be able to allocate I/O file areas. This is the second general area covered.

In following through with this concept, the general topical areas are presented in the following sequence:

Format and Addressing Requirements
Sections I and II

Allocating and Defining Data Areas
Sections III and IV

Input and Output Control
Sections V and VI

Instructions
Section VII through Section XIV

Preparation of the Segment Description
Section XV

Assembly and Correction Parameters
Sections XVI and XVII

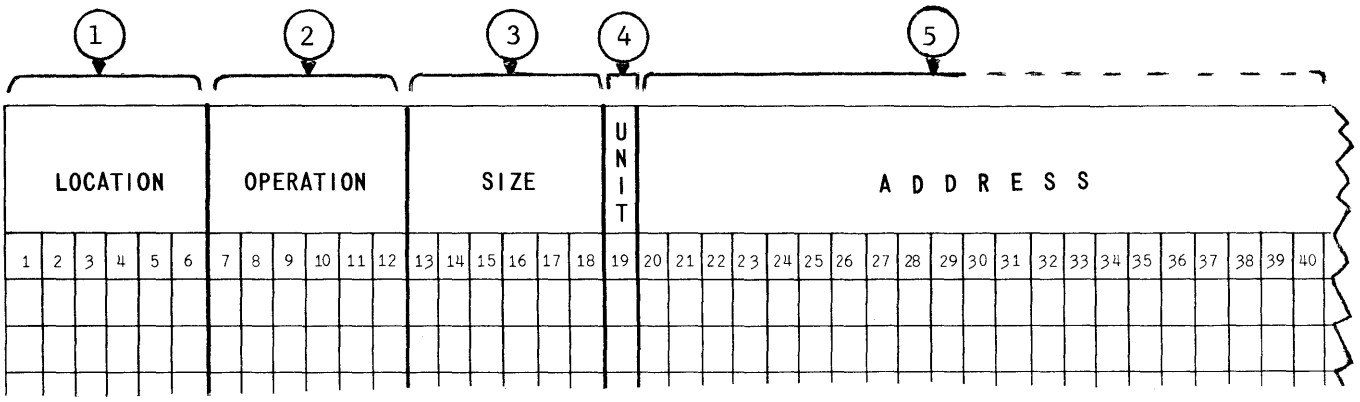
Sample Problem
Section XVIII

For each Assembly Code developed in the manual, the format requirements are explained, examples are furnished, and where deemed necessary, an example is furnished which associates the code being discussed with other related codes or options.

In the Format portion for each code encircled numbers are used across the top of an assembly form. These are used primarily as a quick method of pinpointing fields being referenced by explanatory remarks on lines which follow. They should also be helpful when used in formal training classes as a reference point for discussion.

At the option of the user of this manual, notes may be added in the space provided on the right.

The following is an example illustrating the Format Requirements portion of a code:

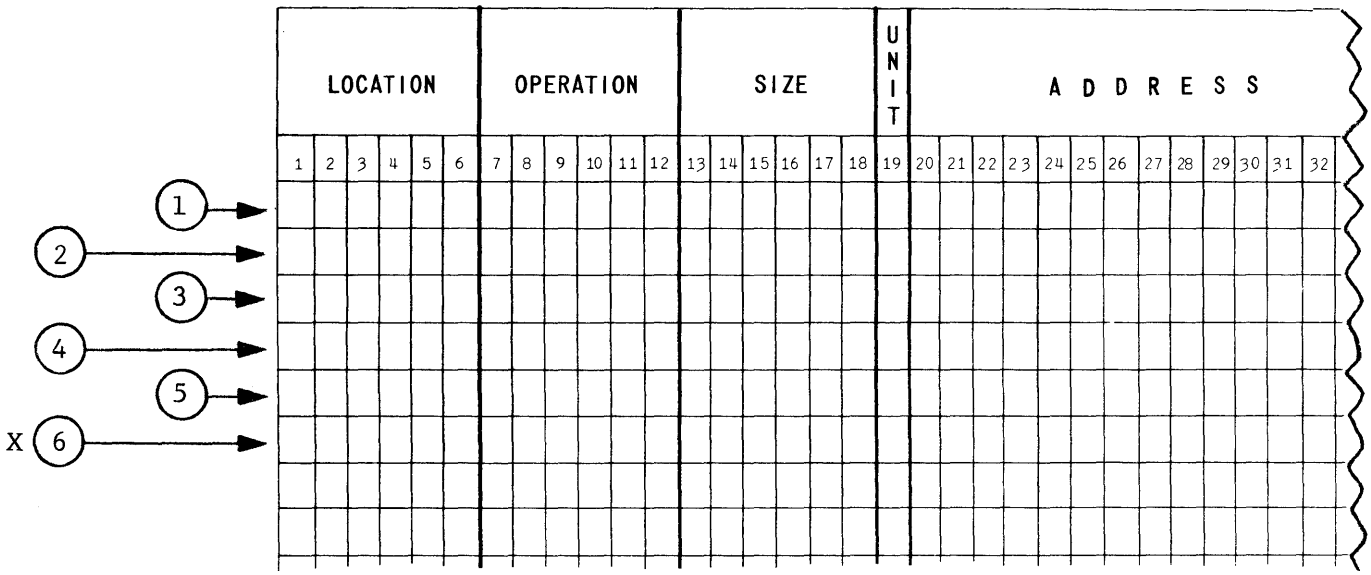


	FORMAT	NOTES
(1)	LOCATION *	
(2)	OPERATION *	
(3)	SIZE *	
(4)	UNIT *	
(5)	ADDRESS *	

*Explanatory remarks pertaining to this field for the code concerned.

In the "Examples" portion for each code, encircled numbers are used along the side of an Assembly form. The numbers are referenced by comments below the form as illustrated on the next page.

It should be noted that examples of incorrect entries preceded by an "X" are provided for illustrative purposes.



(1) Explanatory comments
Line 1

(2) Explanatory comments
Line 2

(3) Explanatory comments
Line 3

(4) etc.

(5) etc.

X (6) INCORRECT ENTRY
or
POSSIBLE ERROR ENTRY
Explanatory comments regarding error entry.

Examples of some of the other features of the RCA 3301 Assembly System that will be explained in detail in other portions of the manual include:

- Optional output listings
- Reassignment of Reference Keys
- Calling of Source Language Routines

GENERAL FORMAT REQUIREMENTS

On the following pages of this section, the general information required for preparing lines of assembly coding will be explained under each of the columnar headings: LOCATION, OPERATION, SIZE, UNIT, ADDRESS, IDENTification, and REFERENCE KEY. (A copy of the complete form is shown on the following page.)

For each of the columnar headings, the format requirements will be given and, in addition, examples of appropriate entries and examples of typical types of error entries will also be given for illustrative purposes.

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM _____

PROGRAMMER _____

PAGE NO. _____ OF _____

DATE _____

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
1 2 3 4 5 6	7 8 9 10 11 12	13 14 15 16 17 18	19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70	71 72 73	74 75 76 77 78 79 80

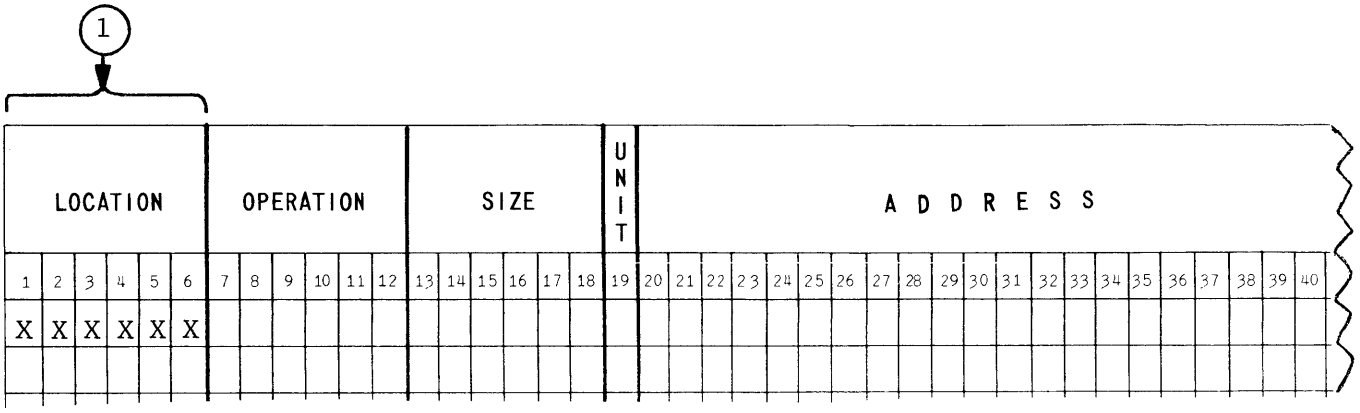
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

28-00-109

REMARKS:

LOCATION Field (Cols. 1 to 6)

FORMAT REQUIREMENTS



NOTES

LEFT-JUSTIFICATION

Each entry appearing in the LOCATION Field must be left-justified; i.e., spaces may not appear to the left or in the middle of any entry for this field.

LENGTH

May be from one to six characters in length.

NUMERIC and/or ALPHABETIC CHARACTERS

An entry (tag) in the LOCATION Field may include alphabetic (A to Z) characters, numeric (0 to 9) characters, or a combination of alphabetic and numeric characters. The use of other characters in this field is not permitted.

UNIQUENESS

Each tag must be unique within the sequence in which the entry appears. Note that an entry on this line may have a prefix character assigned. This character would be considered the first character of this entry even though not included on this line. (See ALOC controlling code - Section III.)

OMISSION OF ENTRY

It is not necessary or even desirable to assign an entry to every assembly line. Reference may be made to either a line of instruction coding or a data field entry by addressing the line relative to a line in which an entry appears.

LOCATION Field Format Requirements

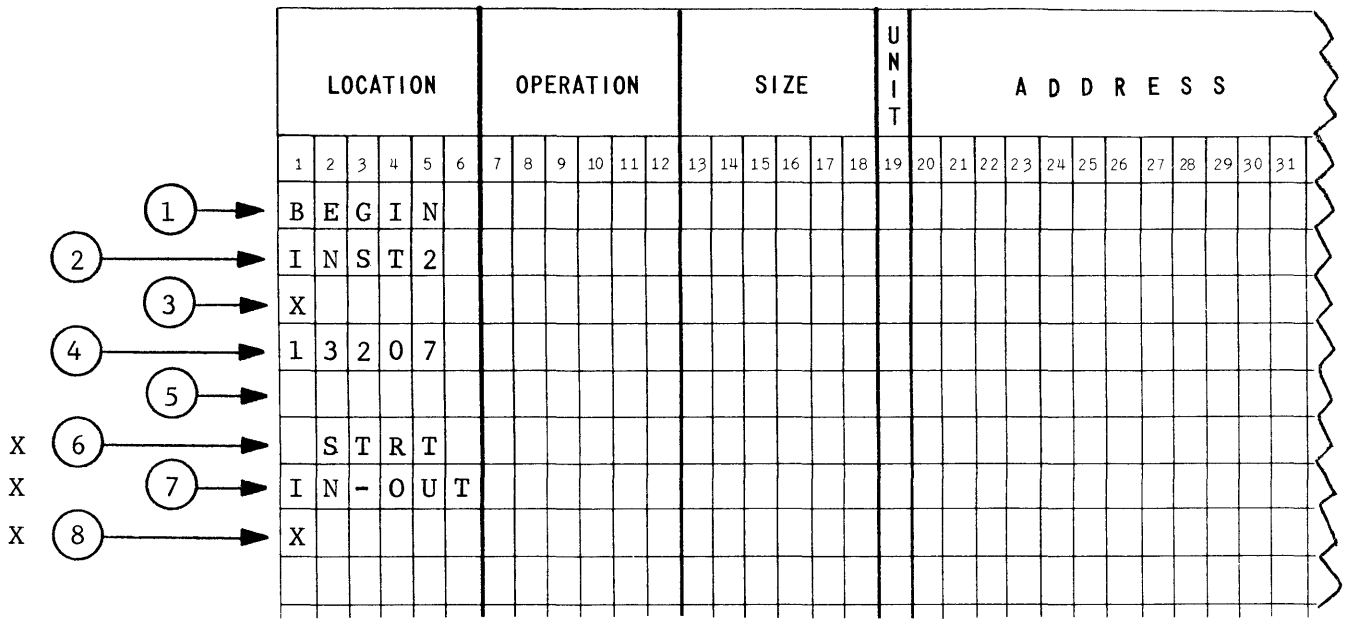
NOTES

ADDRESS ASSIGNMENT

The leftmost address of the data field or instruction generated by the assembly line will be assigned to the tag appearing in this field. This address will be considered the primary address. A secondary address of the right end of the field will also be assigned bv the Assembly System.

LOCATION Field Format Requirements

LOCATION FIELD EXAMPLES



NOTES

(1) thru (5) All of these are examples of valid entries, or the lack of an entry as on line (5).

X (6) Incorrect Entry
Tag is not left justified.

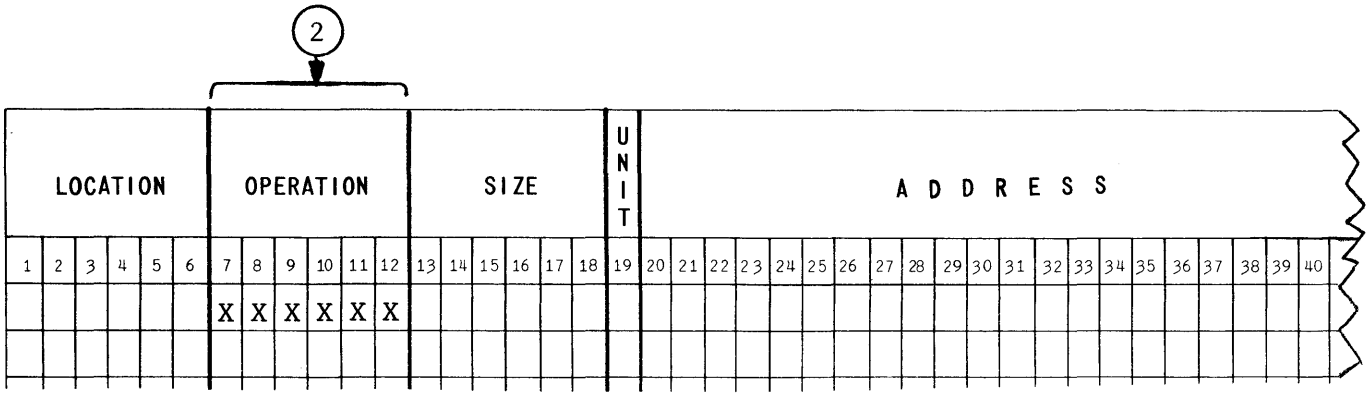
X (7) Incorrect Entry
Invalid character in tag.

X (8) Possible Error Entry
The same tag appears on line (3). Tag would not be unique if these examples were considered part of the same sequence and no prefix character was assigned. (ALOC controlling code, Section III of this manual.)

LOCATION Field Examples

OPERATION Field (Cols. 7 to 12)

FORMAT REQUIREMENTS



NOTES

LEFT-JUSTIFICATION

Each entry appearing in the OPERATION Field must be left-justified; i.e., spaces may not appear to the left or in the middle of any entry for this field.

LENGTH

May be from one to six characters in length.

CONTENT

May contain one of the following:

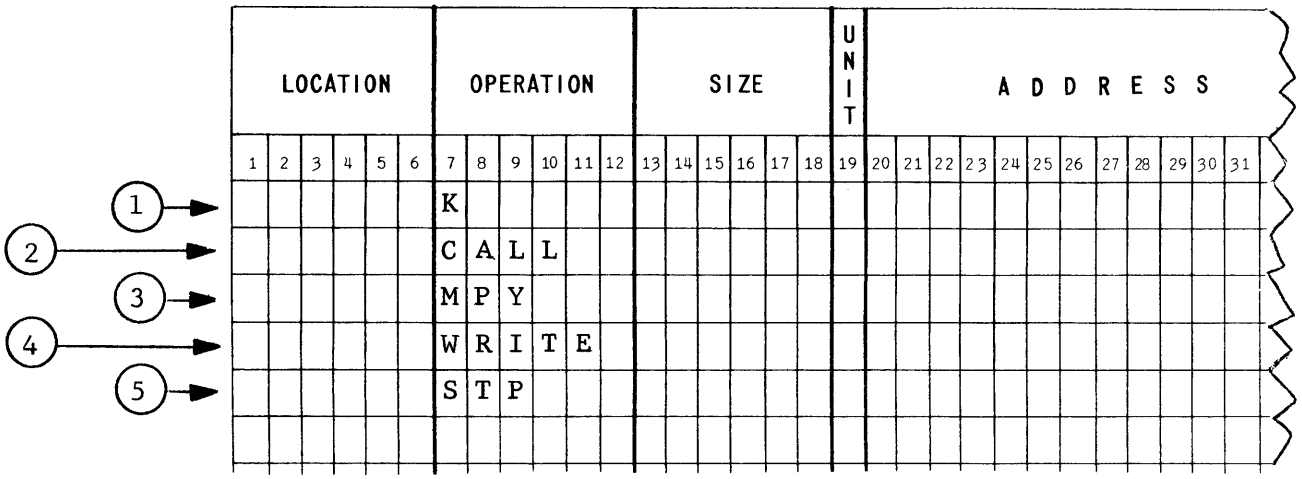
a. Machine Operation Code
b. Mnemonic or Extended Mnemonic Operation Code
c. An assembly or a file (FCP) controlling code
d. May be left blank (not used) on some specified assembly lines.

IMPORTANT

It should be noted that an entry is not assumed in the OPERATION Field. For a specific line, any required entry in this field must be made; i.e., the Assembler makes no assumption that an entry made on a preceding line is to be repeated.

OPERATION Field Format Requirements

OPERATION FIELD EXAMPLES



NOTES

(1) Valid entry of Machine Operation Code. (Locate absence of Symbol Left Instruction.)

(2) Valid entry of an Assembly Controlling code.

(3) Valid entry of Mnemonic Operation Code. (Multiply Instruction)

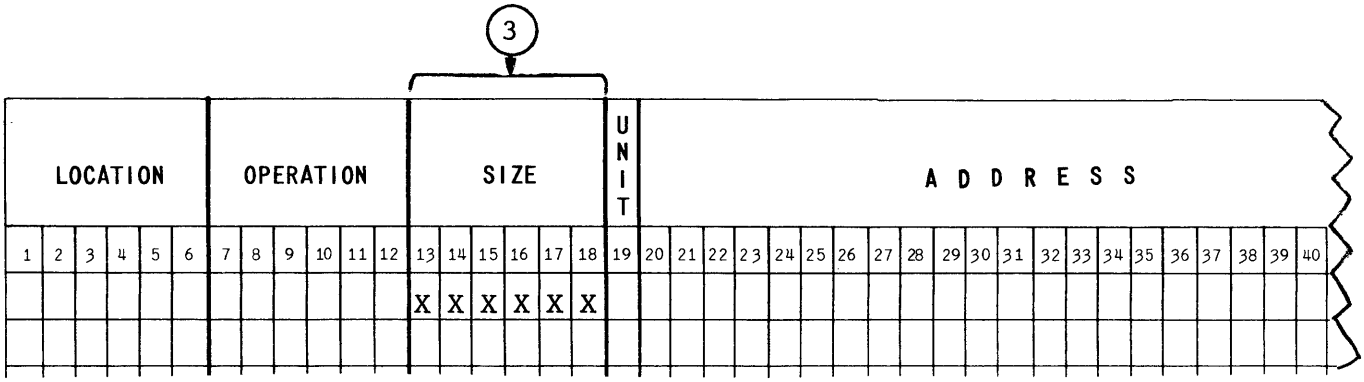
(4) Valid entry of File Controlling Code.

(5) Valid entry of Extended Mnemonic Operation Code. (Store P Register Instruction)

OPERATION Field Examples

SIZE Field (Cols. 13 to 18)

FORMAT REQUIREMENTS



NOTES

LEFT-JUSTIFICATION

Each entry appearing in the SIZE Field must be left-justified; i.e., spaces may not appear to the left or in the middle of any entry for this field.

LENGTH

May be from one to six characters in length.

CONTENT

An instruction may contain the value to be used in generating the second (N) character such as:

A decimal number or a symbol as required by the specific instruction or the data field it references.

For a controlling code:

A decimal number designating the size as required, or a program identification number.

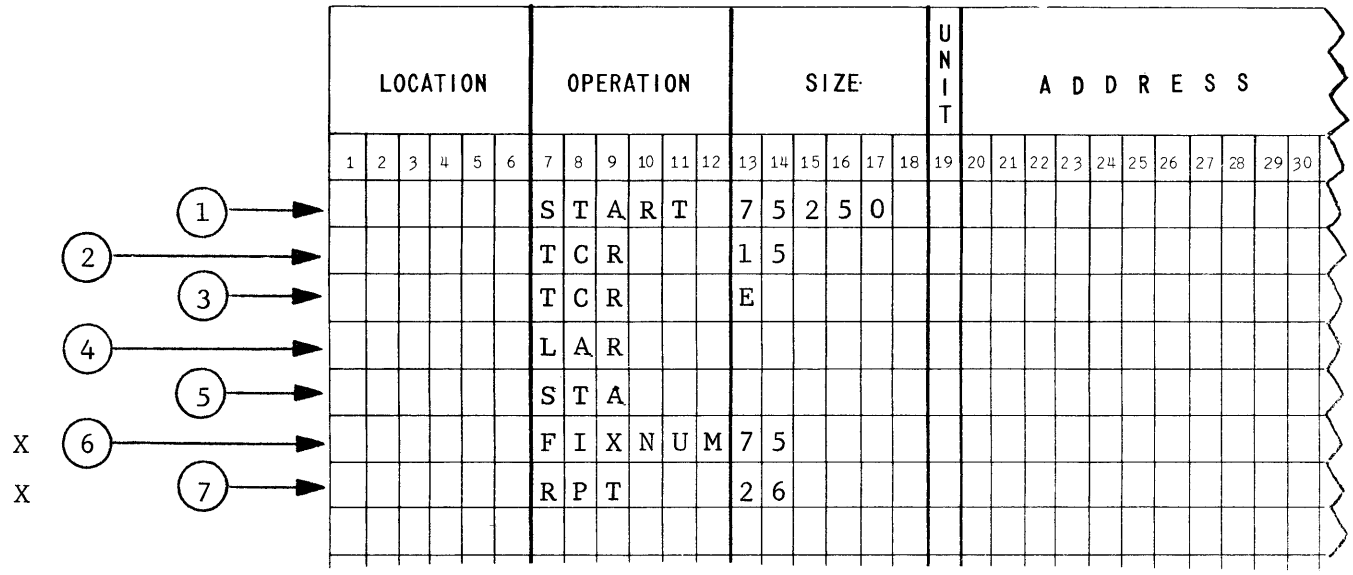
OMISSION OF ENTRY

For an instruction with an Extended Operation Code, the field may be left blank. For these codes, the N Character is automatically generated.

May also be left blank for controlling codes that do not require a SIZE Field entry.

SIZE Field Format Requirements

SIZE FIELD EXAMPLES



NOTES

(1) Valid entry of a Program Identification No. (START controlling code)

(2) Valid entry for a decimal count of 15.

(3) Valid entry for a symbolic character representing a decimal count of 15.

(4) Valid "Entry" assuming a space (left blank) is the character desired in the SIZE Field for this instruction.

(5) No entry required in the SIZE Field for an Extended Operation Code.

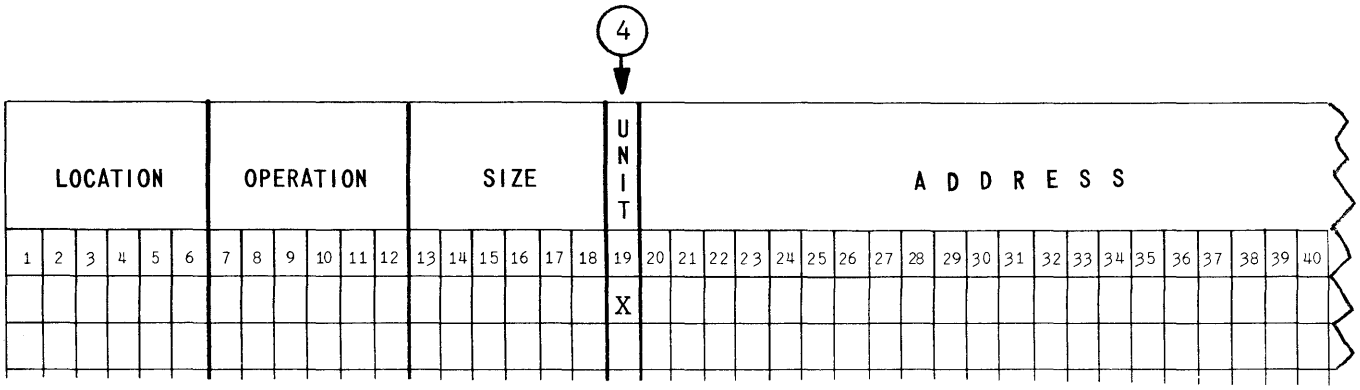
X (6) Error entry - Excessive SIZE entry for this controlling code (50 max.).

X (7) Error entry - Excessive SIZE entry for a Repeat Instruction (15 max.).

SIZE Field Examples

UNIT Field (Col. 19)

FORMAT REQUIREMENTS



NOTES

USE

An entry in the UNIT Field may be made to perform one of the following functions:

a. Specify how the allocation of memory is to be oriented.
b. Specify the type of Validation to be performed at the time of assembly.

CHARACTERS USED

The following characters may be used to specify the type of orientation desired for an applicable line: (See Section III, ALOC Controlling Code for a more detailed description.)

SPACE (Blank)
or

C	Character Orientation
D	Diad Orientation
W	Word (Decade) Orientation
H	Hundreds Orientation
T	Thousands Orientation

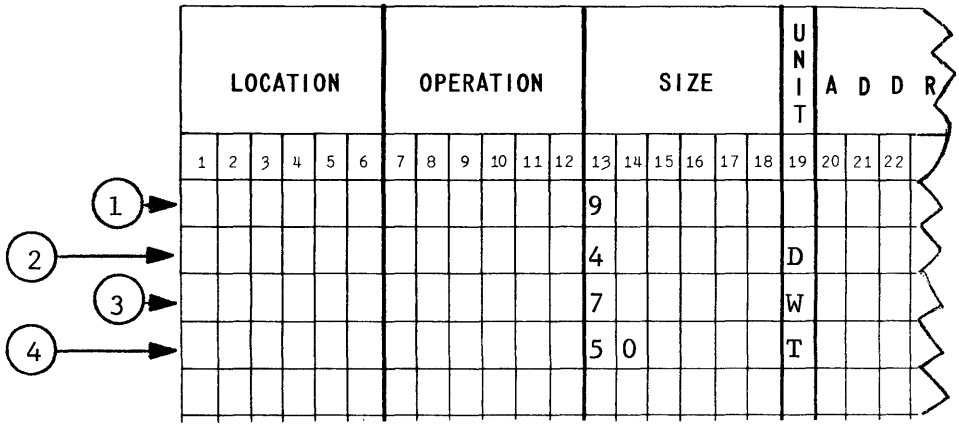
The following characters may be used to specify the type of Validation required on a DEFSEQ line: (See Section III, DEFSEQ Controlling Code for a more detailed description of the use of each of these characters.)

0 or Space	Segment Validation
1	Sequence Validation
2	Sequence Validation
4	Special use for Indexing

UNIT Field Format Requirements

UNIT FIELD EXAMPLES

The following example depicts how the UNIT Field may be used for desired orientation when the Assembler is allocating memory on the basis of the number of characters to be allocated for each line as specified in the SIZE Field.



ASSUMED HSM ALLOCATION

Left-End Address		Right-End Address
0000	-	0008
0010	-	0013
0020	-	0026
1000	-	1049

NOTES

(1) Character oriented (space in UNIT Field).

(1) Assuming as an example that memory locations 0000-0008 have been allocated for this entry then for subsequent lines the allocation of memory would be as follows:

(2) 0010-0013 Allocated

Diad orientation - First character position in next location with an even address (0010).

(3) 0020-0026 Allocated

Word (Decade) orientation - First character position in next location with an address ending in zero (0020).

(4) 1000-1049 Allocated

Thousands orientation - First character position in next location with an address ending in three zeros (1000).

IMPORTANT NOTE

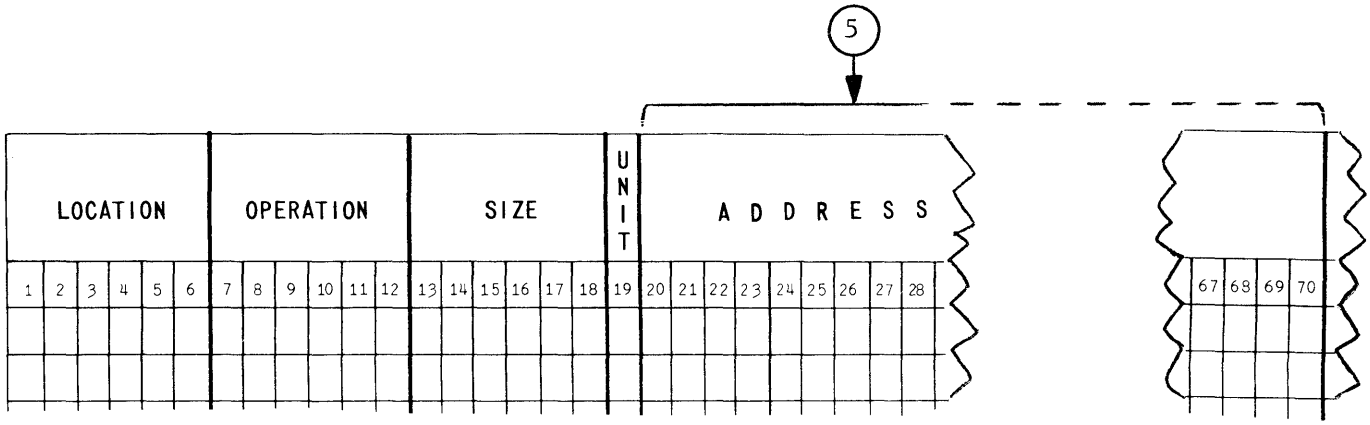
The above example is used only to illustrate how these orientation characters may affect the allocation of memory.

Unless care is taken in the use of this field, excessive memory wastage may take place as in this example between 0026 to 1000 (Lines (3) and (4)).

UNIT Field Examples

ADDRESS Field (Cols. 20 to 70)

FORMAT REQUIREMENTS



NOTES

PURPOSE OF ADDRESS FIELD

The ADDRESS Field is used for the following purposes:

1. To contain the addresses for an instruction.
2. To specify the value in memory to be allocated for data constants, working storage, or other program areas or information as required by specific controlling codes.
3. Users' comments desired on the output listing of the assembled program.

USE OF THE FIELD FOR INSTRUCTION ADDRESSING

The A or B Address may be either a machine or a symbolic address with the various applicable options desired as specified in the following section (Section II - Addressing and Validation).

A comma must be used to separate the A from the B Address and may not appear for any other purpose as a part of the A/B Address entry.

A space must follow the last character of the B Address and may not appear for any other purpose as a part of the A/B Address entry.

The entry consisting of the A Address, a comma, and the B Address must be left-justified in the ADDRESS Field.

Programmers' comments for output listing purposes may appear in any unused portion of the ADDRESS Field following the space used to terminate the A/B Address entry.

ADDRESS Field Format Requirements

NOTES

USE OF THE FIELD FOR VALUE TYPE ENTRIES

For the specific formats of these type of entries, reference should be made to pertinent controlling codes. The following are general rules:

1. Entries that give the value of a field (such as a constant) are left-justified and may consist of any RCA 3301 characters including the space character.

2. In these types of entries the value should be consistent with the value of the entry in the SIZE Field. Otherwise, either truncation or the generation of unwanted characters may result.

USE OF THE FIELD FOR OUTPUT LISTING REMARKS

The entire ADDRESS Field may be used for a comment to appear on the output listing with the use of the REMARK controlling code (See Section XIV - Assembly Parameter Controlling Codes).

In general, the ADDRESS Field may be used for comment on any line subject to the following restrictions:

1. The portion (columns) in which the comment is made is not required for other purposes.

2. At least one space must follow any required entry in the ADDRESS Field.

3. Care must be taken in left-justifying comments without use of the REMARK controlling code.

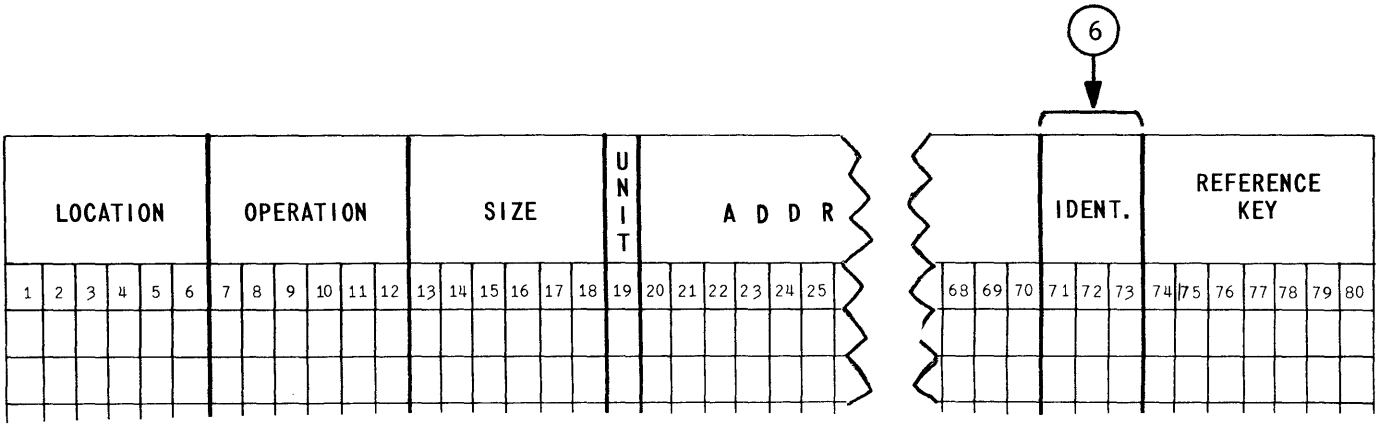
IMPORTANT NOTE

The above are general remarks concerning the use of the ADDRESS Field, and are for general information purposes. Specific use of this field is governed by the appropriate controlling code or instruction being used.

ADDRESS Field Format Requirements

IDENTIFICATION Field (Cols. 71 to 73)

FORMAT REQUIREMENTS



NOTES

USE

The IDENTification Field may be used at the option of the programmer for such purposes as:

1. To provide, for the line of coding, a reference to other documentation such as a flow chart number.

2. To provide a uniquely coded field for identification with a particular segment or process.

This field is for use on the Program Sheet and input data only. An entry appearing in this field will not appear on the output listing.

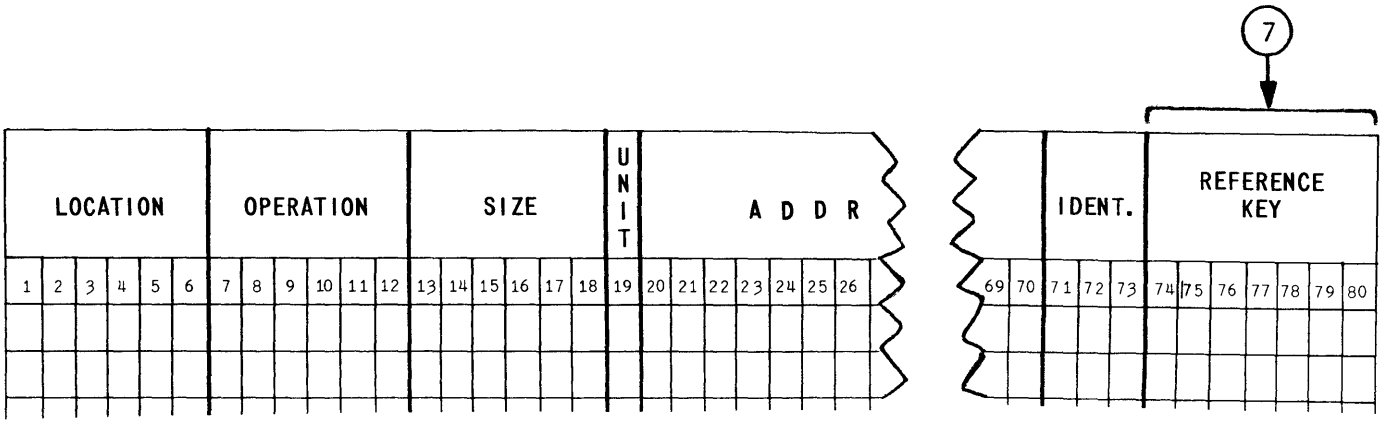
NOTE

This field may only contain 3 or a lesser number of characters and any 3301 character may be used.

IDENTIFICATION Field Format Requirements

REFERENCE KEY Field (Cols. 74 to 80)

FORMAT REQUIREMENTS



NOTES

NUMERIC VALUE

The value of this field should be all numeric characters.

USE

This REFERENCE KEY Field, if present, is used by the Assembly System to:

1. Verify and, if necessary, sort the input source data.

2. Provide a reference for the making of corrections. (See Section XV - Correction Procedures.)

FIELD NOT PRESENT

If this field is not present it will be assumed that:

1. Source statements are in sorted order.

2. Corrections are not desired.

OPTIONAL USE OF THIS FIELD BY ASSEMBLY SYSTEM

An option of the Assembly System provides for automatic renumbering of these fields. If this option is used, the lines will be renumbered in increments of 100 and the first line of sorted and/or corrected input will be assigned a value of 0000100.

REFERENCE KEY Field Format Requirements

NOTES

RIGHT JUSTIFICATION

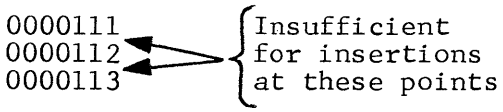
The numeric value assigned to the line in this field should be right-justified.

High-order insignificant zeros to the left of the most significant digit may be omitted if a sort is not desired for the source (Assembly) data.

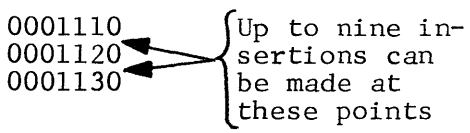
NOTE that in the assignment of values to the REFERENCE KEY Field, a sufficient increment value should be assigned to each successive entry to facilitate the insertion of additional lines of coding using the Correction Procedures.

Example:

Poor



Better



REFERENCE KEY Field Format Requirements

SECTION II

ADDRESSING AND VALIDATION

The purpose of this section is to outline the various methods that may be used in the RCA 3301 Assembly System for addressing and validation of addresses in the user's written program.

ADDRESSING

The ADDRESSING portion discusses the various options used in writing and, if necessary, in modifying addresses. The discussion and examples will be largely restricted to writing such addresses in the A and B Address portion of an instruction.

The options to be covered are:

1. Symbolic Addressing
2. Automatic Orientation of Symbolic Addresses
3. Machine Addressing and Mask Generation
4. Standard Location Addressing
5. Instruction Self-relative Addressing
6. Indirect Addressing
7. Address Modification by:
    a. Incrementing and Decrementing Addresses
    b. Using Index Fields
8. Address Qualification

SYMBOLIC ADDRESSING

A symbolic address is a label (or tag) used to specify a defined location within the context of a particular program.

In the previous section on format requirements, the first field (the LOCATION Field) of the Assembly Program Sheet was discussed.

It was stated that, if this field was used, two addresses will be generated by the Assembly System for the tag appearing in the LOCATION Field. The two addresses generated are the address for the right-hand-end and the address for the left-hand-end of the field generated for the tag.

As an aid in understanding how this is accomplished, the user should be aware that the Assembly System is using a Location Counter to generate an object (or machine language) program from a source (or assembly language) program.

This Location Counter might be considered as any other type of internal counter. It is given an initial value; for example, 0000 before any addresses are assigned for any symbolic tags.

Then, assuming that the user's input has been sorted and corrected, a part of the assembly operation is to assign values to each of the user's tags.

Certain types of lines of coding call for the Location Counter to be advanced to the next diad, word (decade), hundreds, or thousands location, rather than using the next character location.

Certain types of advancing the Location Counter may be done at the option of the user and certain types are performed automatically by the Assembler to conform with machine processing requirements.

The user may specify for certain types of storage that the beginning character position allocated be in the first position of a diad, word, etc. If, for example, the user specified that the beginning position of a diad be used, the Location Counter would be advanced if necessary to the next position that had an even address.

The Assembler, as an example, will automatically advance the Location Counter to the next word when an instruction is encountered as instructions must be word oriented.

AUTOMATIC ORIENTATION OF SYMBOLIC ADDRESSES

Two addresses are generated for each tag. The left-end address is the first character position assigned (after proper orientation mentioned above) and the second address is the last character position assigned. This last position address which is assigned is based on the size of the area required by the information as given on the assembly line of coding.

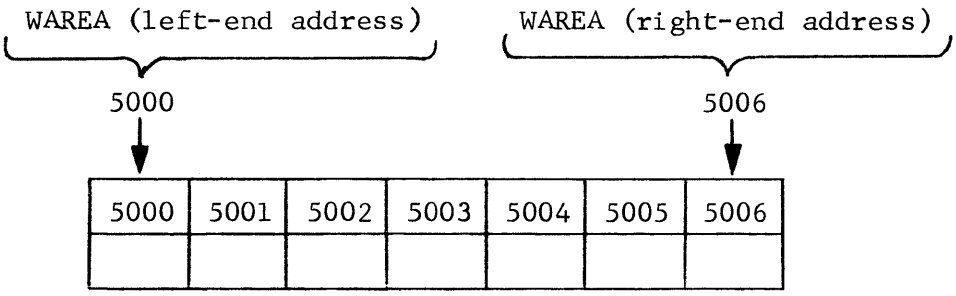
EXAMPLE

User defines a seven character area to be used as a working storage area.

In the LOCATION Field for the line describing this area the user has assigned the tag of WAREA.

Assuming that the Location Counter contained a machine address value of 5000, then the left-end address assigned would be 5000 for the first location of storage, and it would be incremented by a value of six for the next six characters of storage.

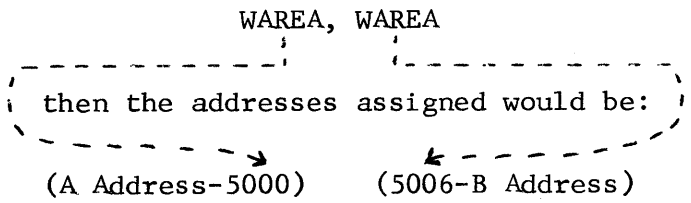
Thus, the right-end address assigned would be 5006 as illustrated below:



It should be noted that the left-end address might be considered the primary address to be assigned whenever the tag is used in a situation where it is not obvious that the right-end address is called for.

As an example, in a situation where a programmer desired to fill the above area (WAREA) with spaces, he might use one instruction. The A Address of the instruction requires the left-end address of a field and the B Address requires the right-end address.

This is an example of a situation in which it is obvious that the right-end address is needed. Thus, if the programmer wrote in the ADDRESS Field for this instruction as described:



If, however, the programmer wanted to fill only the first two locations (5000-5001) with spaces, he would have to use an incremented address for the B Address as described under Address Modification later in this section.

When in the Assembly process of assigning machine addresses, an instruction is encountered, the Location Counter is advanced to the next word; i.e., the next location having an address ending in a zero.

This is necessary because, as explained earlier in the Hardware portion of the manual, all instructions are word (decade) oriented. The Assembler recognizes a line of coding as an instruction by the entry in the OPERATION Field.

The example below illustrates the assignment of machine addresses for symbolic tags and the manner in which the Location Counter is advanced for a sequential series of Assembly lines:

EXAMPLE

In this example, assume that the Location Counter value is set to 5000 for the first assembly line shown.

ASSEMBLY LINE	CONTENTS OF LINE	TAG ASSIGNED	ORIENTATION REQUIRED	FOR THIS LINE, ADVANCE LOCATION COUNTER TO:	LEFT-END ADDRESS	RIGHT-END ADDRESS
First	A 6 char. field	WAREA	Character	5000	5000	5005
Second	A 4 char. field	CADDR	Diad	5006	5006	5009
Third	A 7 char. field	WHDR	Word	5010	5010	5016
Fourth	A 5 char. field	CSTR	Character	5017	5017	5021
Fifth	An instruction	START	Word*	5030	5030	5039
Sixth	An instruction	(none)	Word*	5040		
Seventh	An instruction	END	Word*	5050	5050	5059
Eighth	A 50 char. constant	WHDR	Hundreds**	5100	5100	5149

*Location Counter advanced by the Assembler.

**Note that it would have been a better utilization of memory to have written the eighth line as the first line. The positions from 5060 to 5099 inclusive have not been allocated for any stated purpose in this example. (See, also, the example in this manual in Section III for the ALOC Controlling Code.)

MACHINE ADDRESSING AND MASK GENERATION

As previously discussed in this manual, the modular concept of programming provides for efficiency in the writing of programs and the utilization of the memory required for the execution of programs.

To efficiently utilize memory at program running time therefore, all sequences are assembled relative to the lowest memory address of 0000. Then when the sequence is loaded at the program execution time, it can be "floated" to and referenced at the relative position it occupies.

Thus, the sequence may be floated and a float factor may be applied to every address used in the sequence except those referring to standard locations as discussed below. A more detailed discussion of this process appears in the portion of the manual on the Operating Procedures but the user should be aware of this concept and avoid the use of machine addresses to address other than the standard locations of memory.

Machine addressing should be used only for such functions as:

1. Addressing standard locations (see below - Standard Location Addressing)

2. Incrementing and Decrementing Addresses (see below - Address Modification)

3. Zero Value (Ignored) addresses

In order for the assembler to distinguish between a machine address and a symbolic numeric tag, the machine address must be prefixed by a dollar sign ($). Following the dollar sign, the user writes the machine address as an all numeric address.

Examples:

```
$0       - A zero address
$212     - The left-end address of STA (See Standard Location
           Addressing below)
$159000  - Machine address value Z"00
```

The user may generate a four character 'mask' by prefixing the four characters with an ampersand (&). This mask may be used in either the A or the B Address fields.

The mask, however, may not be modified by incrementing, decrementing, indirect addressing, or indexing as explained in other parts of this section.

Example:

&000" - A mask requiring that the least significant character position have 1 bits in the 2_4 and 2_5 bit positions and 0 bits in the 2_0 to the 2_3 bit positions.

STANDARD LOCATION ADDRESSING

There are three standard location fields that may be addressed with the dollar sign prefixing the symbolic name assigned to the field. These standard location fields are as follows:

HSM Address	Name of Field	Purpose of Field	Assembly Address
0212 - 0215	STA	Store final contents of the A Register after selected instructions	$STA
0216 - 0219	STP	Store the contents of the P Register following selected transfer of control (jump) instructions	$STP
0222 - 0225	STPR	Hold the contents of the P Register during the repetition of an instruction using the Repeat instruction	$STPR

Each of these names may be used; $STA, $STP, $STPR, in the ADDRESS Field and proper right-end or left-end addressing will be generated based on the requirements of the instruction used.

As an example in the use of $STA, the address generated by the Assembler would be 0215 if the right-end address was required, and 0212 would be generated if the left-end address was required.

NOTE:

In addition to the above standard locations certain other locations use the same type of notation.

$SYST for System Standard Locations
$PROC for Process Standard Locations
$USER for System Standard Locations (USER)

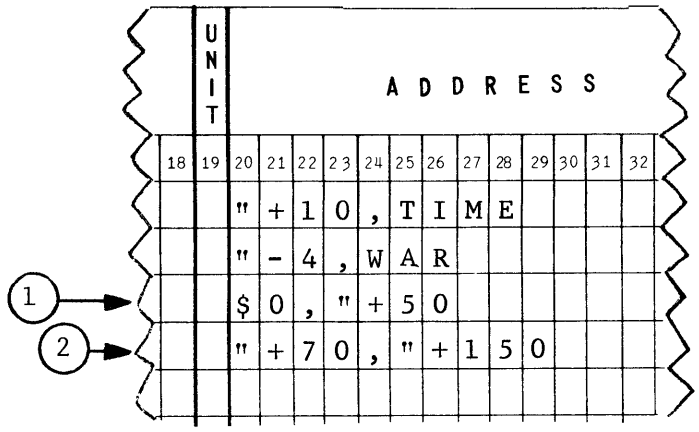
A complete description of the format of these fields appears in an Appendix in the Assembly Reference Manual.

INSTRUCTION SELF-RELATIVE ADDRESSING

In the ADDRESS Field of the instruction coding portion of a user's program, the Assembly System furnishes him with the ability to reference other locations in the sequence. He may do this with an address relative to the instruction in which the address appears.

The quote sign (") is used in this option to obtain the value of the left-end of the instruction in which the quote sign appears. Following the quote sign will be either a plus or minus sign and the value which is to be added to or subtracted from the value of the current instruction.

EXAMPLE



ASSUMED INSTRUCTION ADDRESS	ASSIGNED (SELF-RELATIVE) ADDRESS	
	A	B
0000	0010	--
0010	0006	--
0020	--	0070
0030	0100	0180

(1) (2) Caution should be used in self-relative addressing in these types of situations. The user should be aware of the condition that would be present if corrections in the form of insertions or deletions of lines of coding were made between these lines and the lines of coding referenced. The self-relative addresses on these lines would have to be revised. A better procedure would be to tag the referenced line and use the tag in place of these self-relative addresses. Then insertions and deletions would not affect proper addressing.

Note that the dollar sign ($) prefix is not necessary in self-relative addresses. The Assembler makes the assumption that the decimal value following the quote sign is an increment or a decrement.

INDIRECT ADDRESSING

Indirect addressing is used when four characters which are sequentially stored in two diads are desired as an address.

Indirect addressing is normally used when the programmer cannot address a desired field directly because its actual location varies with each record. This is usually the situation when programming variable sized records.

However, the programmer is aware that following an instruction which is designed for use with variable sized field that an address, relative to the first or last character of the field, will be placed in a known standard location as a function of the hardware.

The programmer can obtain this address by the location where it is stored. The hardware distinguishes an indirect address from a direct address by the presence of a 1 bit in the 2_4 bit position of the least significant character of the address.

The Assembly System recognizes an indirect address by the number sign (#) used as a suffix to the address.

The example below is an illustration of using the standard location of STA (0212-0215) in direct and indirect addressing:

EXAMPLE 1:

Assume the contents of STA are as follows at the time of execution of the assembled program for each of the following lines of assembly coding:

0212	0213	0214	0215
6	7	5	1

ADDRESS Field	Assembled A Address	Execution A Address	Type of Address
$STA,WAR	0215 *	0215 *	Direct
$STA#,WAR	021E	6751	Indirect

*0215 is furnished if the instruction address logic calls for the right-hand-end address.

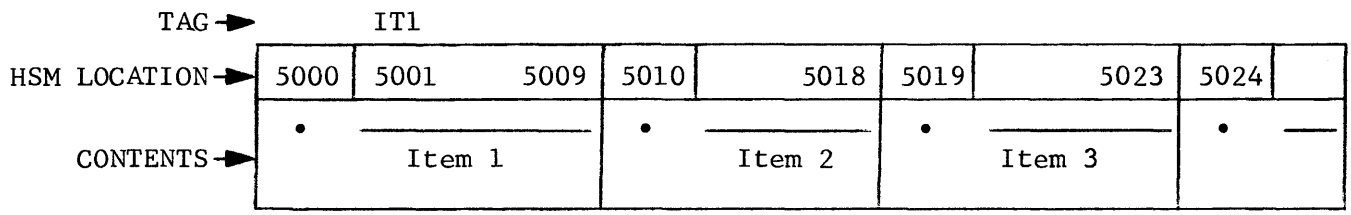
Some of the key points to be considered when using the Assembly option for indirect addressing are as follows:

1. An address to be indirectly addressed must be stored in two consecutive diads.
2. An indirect address is recognized by a number sign (#) following the address.
3. The rightmost diad of the two consecutive diads must be specified. Note that the Assembler will use the right-end address of a field assigned a symbolic tag, if this field is indirectly addressed. The user should assure that the field is diad oriented.

EXAMPLE 2:

This example is an illustration of how STA might be used in the ADDRESS Field of instructions.

Assume that a programmer is writing a portion of a sequence for processing records having variable sized fields. Assume also for this example that the record being currently processed occupies memory as indicated.



In the above example, it should be noted that each item is variable in size, and that this is the item size format of one sample record.

Each item in the record is preceded by a control character, in this case an Item Separator Symbol designated by •. Assuming that the left-end location of Item 1 is known, the programmer could access Item 2 following the instruction for handling Item 1 as follows:

	ADDRESS	ASSEMBLED A ADDRESS	CONTENTS OF STA ** AFTER EXECUTION
(1) →	IT1,WAR	5001	5011
(2) →	$STA#,WAR2	021E*	5020

*Executed address equals 5011 for this record.

**For this record as an example.

(1) This is the ADDRESS Field of an instruction which transfers Item 1 (IT1) to a Work Area (WAR). The instruction terminates after the first symbol is transferred. The final contents of the A Register are stored in STA.

(2) The programmer knows that following execution of line (1), STA will contain the address of the first significant character of Item 2. He uses STA as an indirect address to obtain the address of Item 2 to move it to Work Area 2 (WAR2). The programmer also has an option of placing the four-character contents of an address register into two consecutive diads. This option is explained in a later section - Register Manipulation Instructions. Thus, the user has the option of using the final contents of the A and/or the B Registers following any instruction as either a direct or an indirect address.

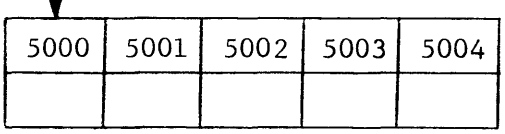
SYMBOLIC ADDRESS MODIFICATION - Incrementing and Decrementing

When the user has assigned a symbolic name (tag) to a field, an assembled address is assigned to that tag as explained earlier under SYMBOLIC ADDRESSING.

When the tag has been assigned in the LOCATION Field of the Assembly Program Sheet, the programmer may address any character position following or preceding the left-end address assigned to this tag. He has this ability through the use of an incremental (or decremental) address.

As an example, assume that a programmer had defined a five character field as WORK and the Assembler assigned the left-end or primary address of 5000 for WORK. He would thus have a field allocated as shown:

PRIMARY ADDRESS OF WORK



If the user wished to address the third position of the field (5002) he could use an address incremented by two locations as follows:

WORK+$2

When using a decimal number of memory locations as an increment or decrement the following key points should be remembered:

1. The dollar sign ($) prefix must be used before the decimal value.

2. The plus (+) or minus (-) sign must precede the prefixed dollar sign.

3. Whenever incremented or decremented addresses are specified, the value assigned to a symbolic tag is the left-end (or primary) address. A symbolic name may also be used for incrementing or decrementing. As an example, assume that the tags WORK and SCOPE have been assigned addresses 1000 and 2000, respectively. If the programmer then wrote an assembly address of WORK+SCOPE, the assembled address assigned would be 3000.

EXAMPLES

In the following examples of incremented and decremented addresses assume that the Assembler has assigned machine address values as indicated:

TAG	LEFT-END ADDRESS ASSIGNED
START	0100
WORK	1000

The Assembler would then assign addresses as shown below:

ADDRESS Field	ASSEMBLED A ADDRESS
START+$2,$0	0102
WORK-START,$0	0900
$STA+$1,$1	0213
$STP-$4,$0	0212
WORK+$25,$0	1025

Note that when a symbolic address is incremented or decremented by a machine value, that the symbolic address must be specified first.

ADDRESS MODIFICATION BY INDEXING

Indexing is a hardware function that automatically increments designated addresses by a value which is prestored in an Index Field.

An address to be incremented must contain a specified arrangement of 1 bits in the zone portion (2_4 and 2_5) bits of the third character of the machine address.

There are three Index Fields and any one of these may be designated in the address to be modified.

The programmer may specify indexing by the first, second, or third Index Field following the address with a colon (:) and the characters M1, M2, or M3, respectively.

For example, if the programmer had defined a field by the symbolic tag TABLE and wanted incrementation by the second Index Field, he would write:

TABLE:M2

To illustrate the manner in which indexing may be specified by the user and how it is implemented by the Assembly System the following example shows how the programmer writes an address to be indexed, the machine address as assembled, and the address that would be executed at object program running time.

EXAMPLE

Assume that the value of the address assigned the symbolic tag, and the values stored in each of the Index Fields are as follows:

TAG	ASSIGNED ADDRESS
READIN	5000

INDEX Field	1	2	3
Contents	0010	0150	0600

Assembly Address	Indexing Field	Zone Bits of 3rd Char. 2_5	2_4	Assembled Machine Address	Machine Address As Executed
READIN	None	0	0	5000	5000
READIN:M1	1	0	1	50&0	5010
READIN:M2	2	1	0	50-0	5150
READIN:M3	3	1	1	50"0	5600

The value in the Index Field is always added to the address as illustrated above. However, if the effect of subtraction from an address is desired, the Index Field may contain the complement of 160000 as the quantity. For example, assume that it was desired to decrement READIN (5000) by 2000 and using Index Field 3

CONTENTS OF INDEX FIELD 3
158000 (or Y"00 ACTUAL)

Assembly Address	Assembled Address	Execution Address
READIN:M3	50"0	3000

Each of the Index Fields is contained in Micromagnetic memory and associated with each Index Field is an Increment Field also in Micromagnetic memory. The contents of each Increment Field may be added to the contents of its associated Index Field at the option of the programmer.

This function of incrementing the Index Field is performed as an option of the execution of the Tally instruction. The details of the Tally instruction will be discussed in another section of this manual and only the manner in which Index Fields may be incremented will be illustrated in this section.

The incrementing function is designated by an arrangement of 1 and 0 bits in the second (N) character of a Tally instruction. The programmer specifies in the SIZE Field(s). This SIZE entry generates the N character of the machine instruction. Examples of the increment function specified in the SIZE Field are as follows:

Increment Index Field	SIZE Field Entry of Tally Inst.
1	I1
1 and 2	I1,2
1, 2, and 3	I1,2,3

The programmer has the ability to load values in both the Index and Increment Fields through the use of the Load Register instruction which is covered in a later section of this manual.

The programmer will find many uses for Indexing. An example of one way in which indexing may be used is the processing of fixed length records, a series of which are contiguously stored in memory.

In this example, assume that a programmer has five 80 character records stored in an area with a left-end address tagged READIN.

In processing each record he will use Index Field 2 as a modifier. Initially, for processing the first record, the Index and Increment Fields would be as follows:

INDEX 2	INCREMENT FIELD 2
0000	0080

Thus, all addresses in the processing path would be modified by 0000.

After processing the first record, he increments the Index Field 2 using the Tally instruction which also returns control of his program to the first processing step. The Index and Increment Fields are as follows for processing the second record.

INDEX 2	INCREMENT FIELD 2
0080	0080

Thus, in going through the processing path for the second record, every address is incremented by a value of 0080.

These steps would be repeated for the processing of the third, fourth, and fifth records. After processing the fifth record, however, the Tally instruction would not return to the first processing step. When another five records are in the area READIN, the Index Field would be set to 0000 and the processing steps outlined above would be repeated.

ADDRESS QUALIFICATION

The modular concept of programming, as discussed previously in this manual, provides many advantages in all phases of program planning, preparation, and implementation.

As an example of one efficiency in the area of program preparation, individual sequences may be prepared by different programmers, each programmer writing the type of sequence for which he is best qualified.

Unless there are adequate control standards governing the assignment and use of symbolic names (tags), two or more programmers might inadvertently use the same tag.

In such a situation, the programmer must qualify any reference to another sequence by the name of sequence followed by symbolic tag as in the following format:

SEQ1@TAG1

where TAG1 is the symbolic name in another sequence and SEQ1 is the name of the sequence in which the tag appears. The symbol "@" must be used following the sequence name.

This type of address qualification should be required for all out of sequence references where the possibility of duplicate tags exist.

Address qualification is not necessary when it is known that all tags are unique within the segment being assembled.

VALIDATION

Every symbolic tag which is used in the ADDRESS Field must be defined by appearing in the LOCATION Field. As explained in the beginning of this section under Symbolic Addressing, an entry in the LOCATION Field is assigned addresses. In addition, each entry in the LOCATION Field must be unique within a sequence, and may be written so as to be unique within the segment being assembled.

"Validation" is an operation of the Assembler that insures that all symbolic tags used in the ADDRESS Field have been so designed.

A one character entry may also be used to indicate whether or not the sequence requires loading at object time; i.e., a sequence that may consist only of working storage areas does not require loading at object time.

The codes and the specifications for each are as follows:

CODE	VALIDATION OF TAGS	LOADING AT OBJECT TIME
ZERO or SPACE	Each tag unique in the segment	Will be required
1	Each tag unique only within each sequence	Will be required
2	Each tag unique only within each sequence	Will not be required
4	Not to be performed*	Will not be required

*This is a special code used for assigning zero relative addresses to tags for use as increments in indexed addresses. An example of the use of this code is given in Appendix B, In-Place Processing.

SECTION III

ALLOCATION OF DATA AREAS AND CONSTANTS

The purpose of this section is to outline the manner in which data areas may be allocated and how constants may be specified.

This is normally one of the first steps in the writing of a program.

The controlling codes used in this section will be discussed by giving the format requirements and examples of the use of each code as well as progressive examples which show the combined use of two or more controlling codes.

As indicated earlier, this training document is intended to supplement rather than replace the Realcom Assembly Manual which should be used as a basic reference.

For each controlling code there will be first an outline of the format requirements and this will be followed by explanatory comments and an example.

The controlling codes to be covered and the general purpose of each is as follows:

DEFSEQ

This code notifies the Assembler that a new sequence is being started. Every line following this code will be included in the sequence until another DEFSEQ code or the END of the program is encountered.

ALOC

This code is used to notify the Assembler that the allocation of memory is required for subsequent lines. This code may be used for example to allocate data record, file read-in, and scratch pad work areas.

FIXCON

This code is used to set up alphabetic and alphanumeric constants of specified values. Each constant may be given a name (tag) which may be used as a later reference to this constant.

FIXNUM

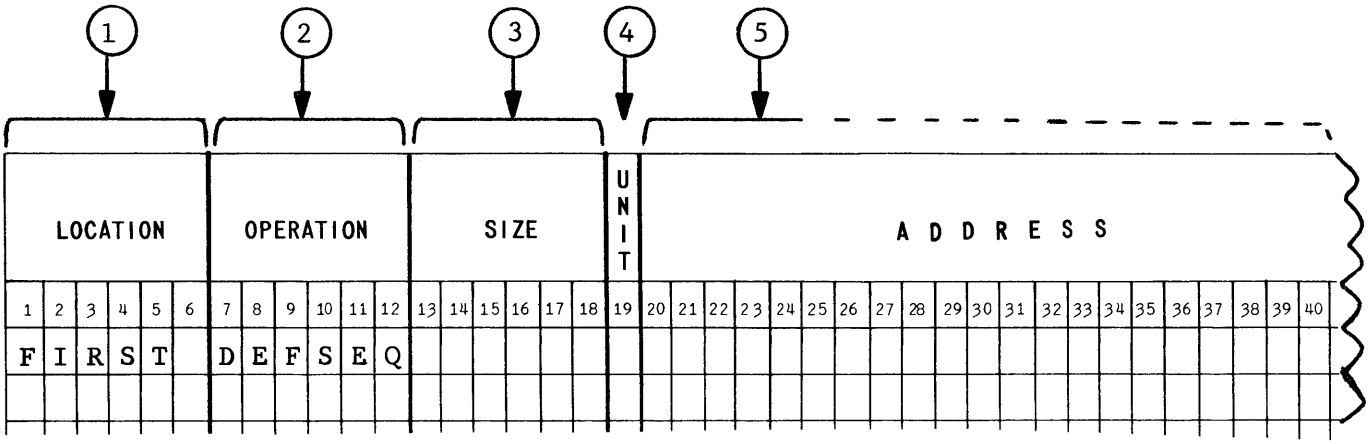
This code is very similar to the preceding code but may be used only for numeric fields consisting of the characters zero (0) through nine (9) and a sign which is included as a bit in the least significant digit.

ADRCON

This code is used to store an address in a location where it may be used as a constant.

DEFSEQ

Specifies the beginning of a sequence to the Assembler.



FORMAT NOTES

(1) LOCATION

The name (tag) of the sequence must be entered in this field. This name is necessary for the Segment Description which will be discussed later.

(2) OPERATION

DEFSEQ must appear in this field.

(3) SIZE

NOT USED (SIZE Field)

(4) UNIT

May contain one of the following characters to indicate type of validation for tags in the sequence.

Code 0 (Zero) or Space

Each tag should be unique within the segment.

Code 1

Each tag unique only within the current sequence. This code also indicates that loading of this sequence at object time will be necessary. Contains coding and/or constants.

DEFSEQ Format

FORMAT NOTES

Code 2

Same as Code 1 except that loading at object time will not be required. No coding or constants included.

Code 4

Used to obtain zero relative addressing for tags (to be used for index referenced addresses). Also a sequence that is not to be loaded.

(5) ADDRESS

USED ONLY IN file description sequence. See Section V. May be used otherwise for comments.

COMMENTS

(1) (2) The first line of a sequence must be a DEFSEQ entry and this sequence must be tagged in the LOCATION Field.

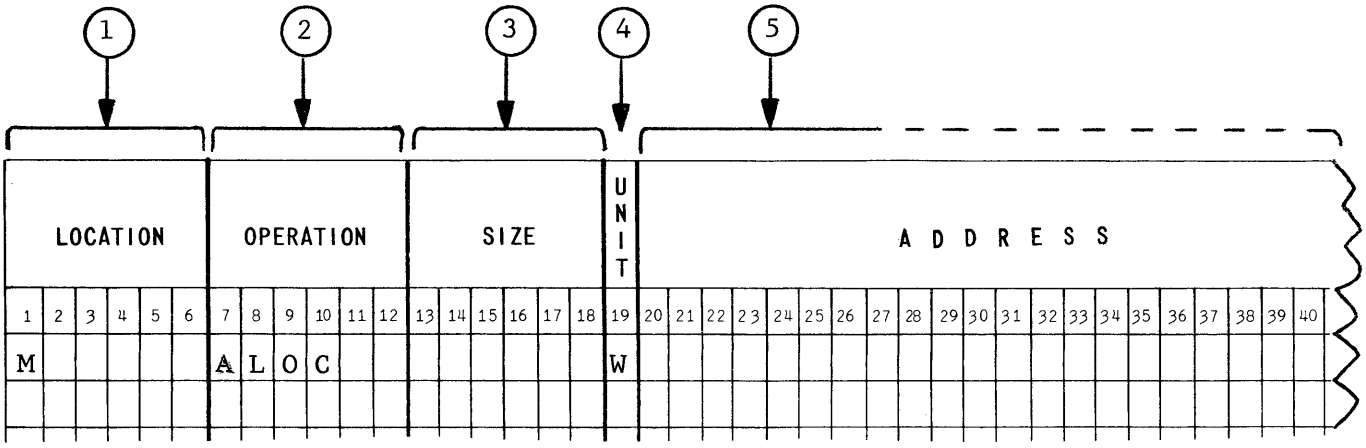
(4) Code 0 (or space) will check each tag in this sequence for uniqueness against all other tags in the segment.

(4) Code 1 and Code 2 checks each tag for uniqueness only within the current sequence. Thus, any out of sequence reference should be qualified by the name of the reference sequence.

DEFSEQ Format and Comments

ALOC

Used to allocate memory for data and work areas.



FORMAT NOTES

(1) LOCATION

This is an optional entry of one character only which is left-justified. This character, if used, will be considered a prefix to all tags that follow until the ALOC operation is terminated. (See Comments.)

(2) OPERATION

ALOC must appear.

(3) SIZE

NOT USED

(4) UNIT

Specifies how allocation of memory is to be oriented; i.e.,

C or Space - Use the next character location

D - Use the next diad (begin allocation in the next location having an even address)

W - Use the next decade (word) (Begin allocation in the next location with an address ending in zero - "XXX0")

H - Use the next hundred's location (Begin allocation in the next location with an address ending in two zeros - "XX00")

ALOC Format

FORMAT NOTES

T - Use the next thousand's location (Begin allocation in the next location with an address ending in three zeros - "X000")

(5) ADDRESS

NOT USED unless for the listing of programmer comments.

NOTE that when allocating areas for specified Input/Output functions a required type of orientation may be required for:

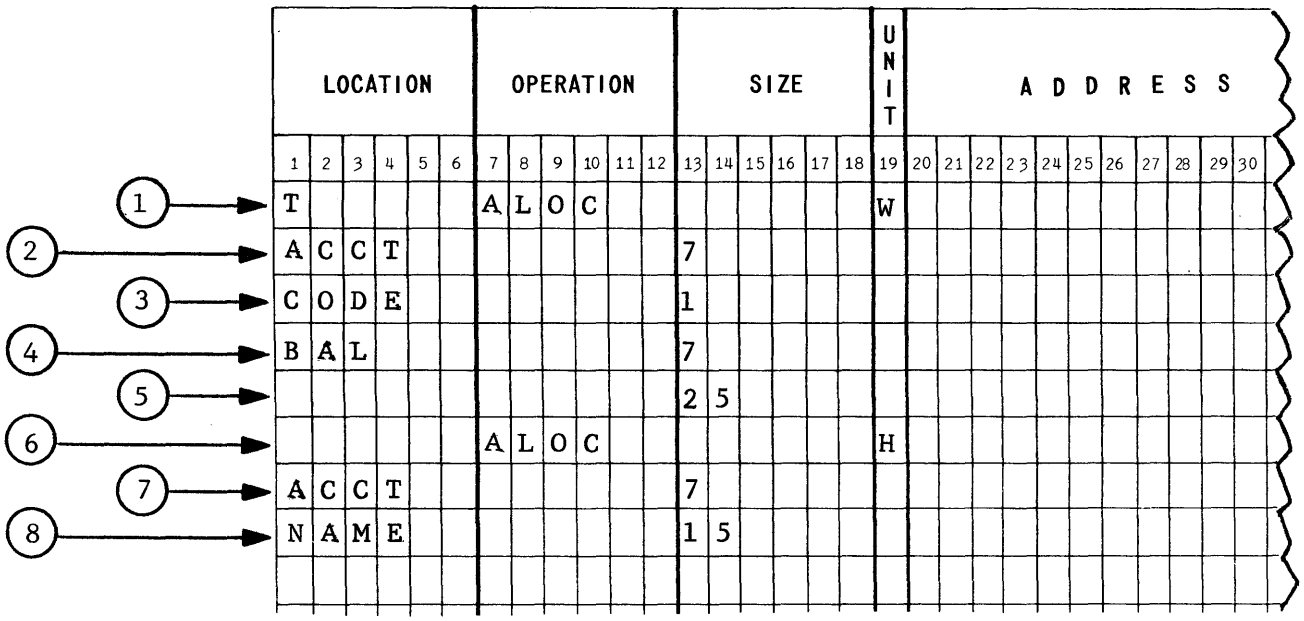
Printing - the area should be word oriented and 120 or 160 contiguous locations.

Card Punching - the area should be diad oriented and 80 contiguous locations.

ALOC
Format

ALOC (EXAMPLE)

This is an example of how a data area may be allocated.



COMMENTS

NOTES

(1) Prefix of T will be assigned to all tags on lines (2), (3), and (4). For example, later reference to the first entry (2) would be TACCT. (The symbolic tag T could also be used for the left-end address of this first field.)

(1) If a prefix is assigned in the ALOC entry it will prefix all tags that follow until the ALOC entry terminates. ALOC terminates by an entry in the OPERATION Field other than:

REMARK
RENAME
ADRCON
FIXCON
FIXNUM
FLTNUM

Thus, the prefix, if assigned, would prefix all tags used with these controlling codes which follow the ALOC code.

(1) Allocation of memory begins with the next decade, but: (See next page)

ALOC
Example
and
Comments

COMMENTS NOTES

(2) (3) (4) (5) All of these following entries are character oriented (space in UNIT Field).

(5) User is allocating 25 characters not referenced by a tag.

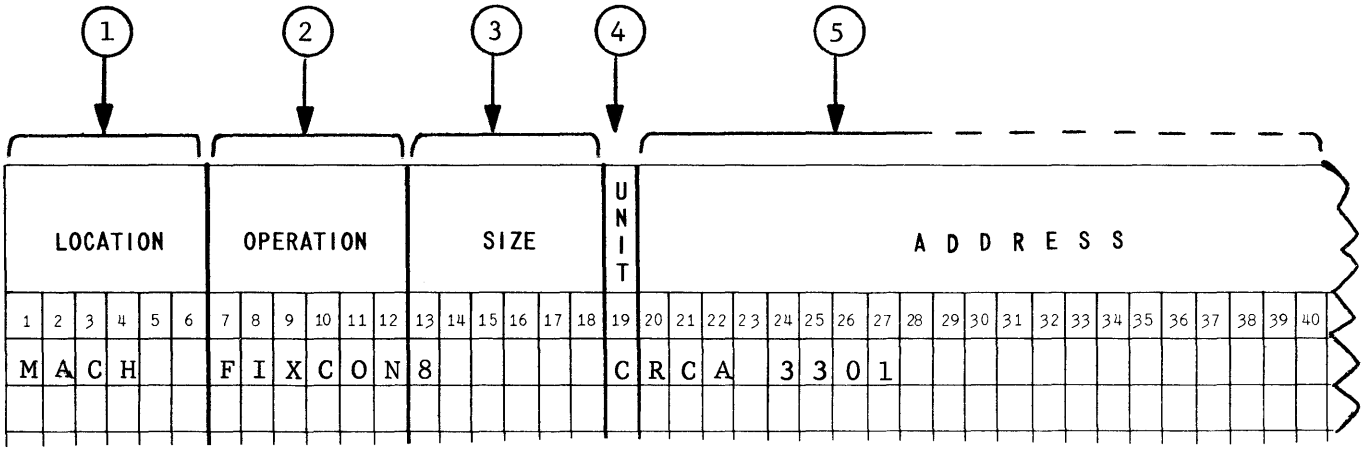
(6) Allocation will begin in the next hundreds location (address ending in XX00). Note that unless care is taken in allocating memory with the hundreds (H) and thousands (T) options, excessive wastage (unused areas) of memory may take place. For example, if allocation of memory for line (5) terminated with an address of XX01, then 98 unused positions would follow this entry (TBAL + $31).

(7) (8) Reference to these entries are as indicated in the LOCATION entry as there is no prefix assigned in the ALOC entry (Line 6). Note that the tags for lines (2) and (7) are unique; TACCT and ACCT respectively.

ALOC
Comments

FIXCON

Used to set up an alphabetic or alphanumeric constant.



FORMAT NOTES

(1) LOCATION

Constant is referenced by the tag which appears in this field (with ALOC prefix if applicable).

(2) OPERATION

FIXCON must appear in this field.

(3) SIZE

The size of the constant may not exceed 50 characters.

(4) UNIT

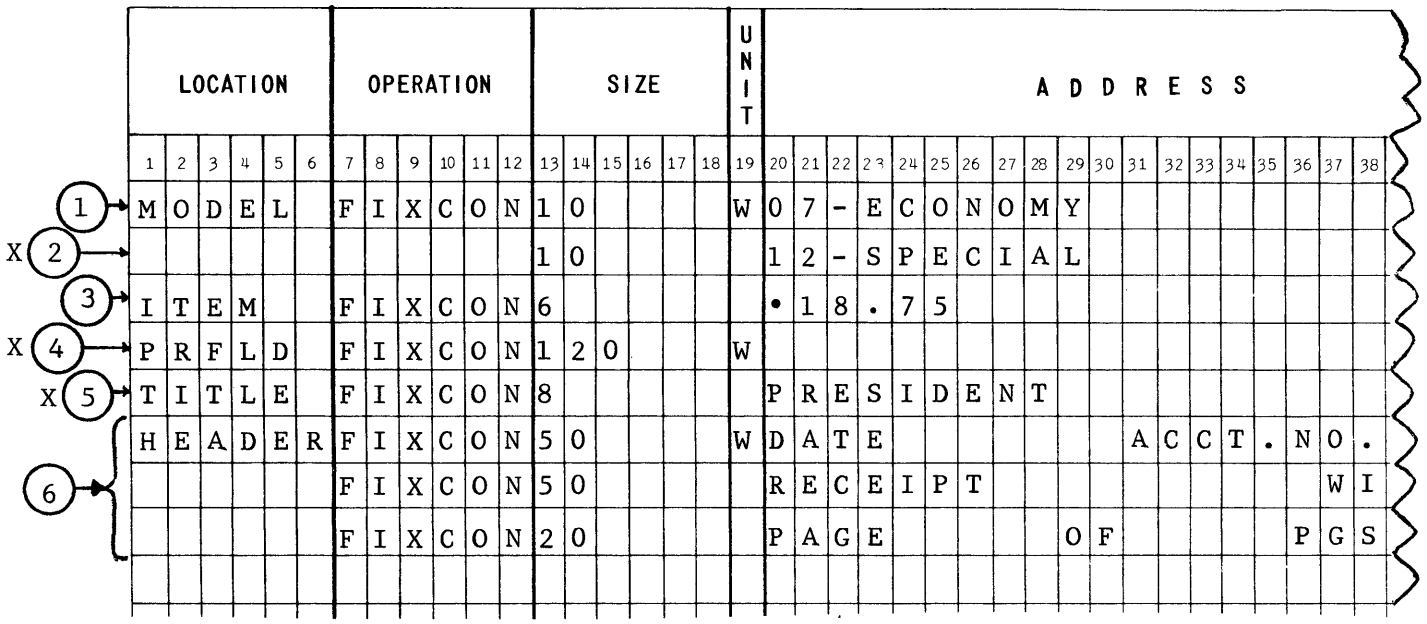
Constant is stored in the next available unit of memory as indicated in this field. If left blank, the constant will begin in the next available character location. (See ALOC entry for this field.)

(5) ADDRESS

The value of the constant is written in this field. The number of characters written must equal the SIZE Field (3).

FIXCON Format

FIXCON (EXAMPLE)



FORMAT

NOTES

(1) Word oriented constant.

X (2) Nothing allocated for this line because FIXCON is omitted (assuming ALOC code not previously used).

(3) This line would not be acceptable if input was from paper tape as the ISS (•) is used to specify separation of fields on paper tape input.

X (4) Error entry Maximum size per line for constant is 50 characters.

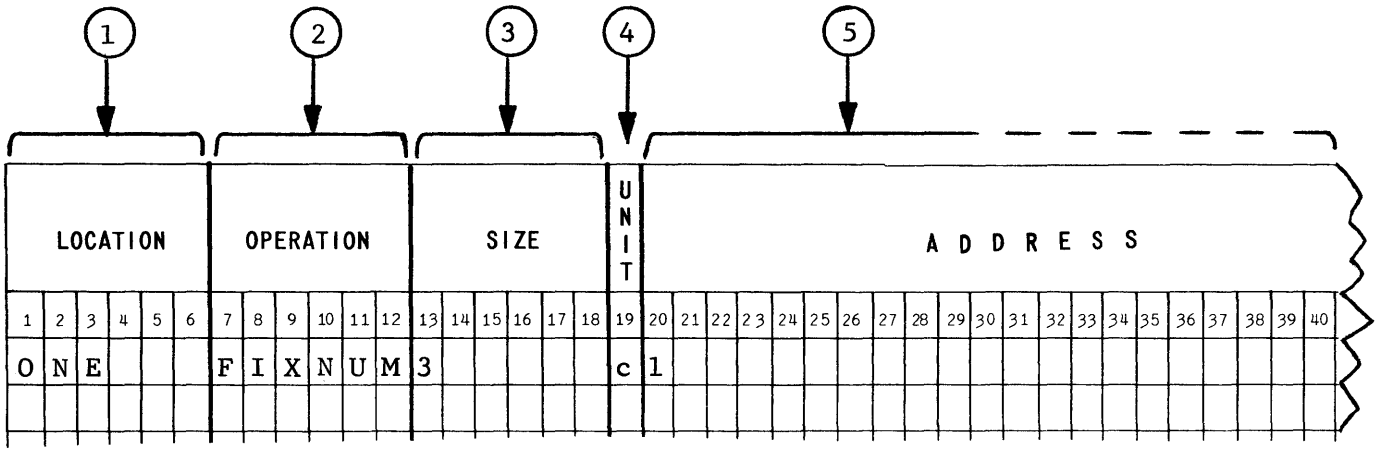
X (5) Possible error. SIZE Field does not agree with value in ADDRESS Field. Last character (T) will be truncated.

(6) This is an example of how a constant of over 50 characters may be written. NOTE, however, that to refer to the right-hand-end of the 120 character constant, a character relative address must be used such as HEADER+$119.

FIXCON
Examples

FIXNUM

Used to store a fixed numeric constant.



FORMAT NOTES

(1) LOCATION

Numeric Constant is referenced by the tag appearing in this field (with ALOC prefix if applicable).

(2) OPERATION

FIXNUM must appear in this field.

(3) SIZE

Size of the generated constant is specified in this field (Maximum of 50 per line).

(4) UNIT

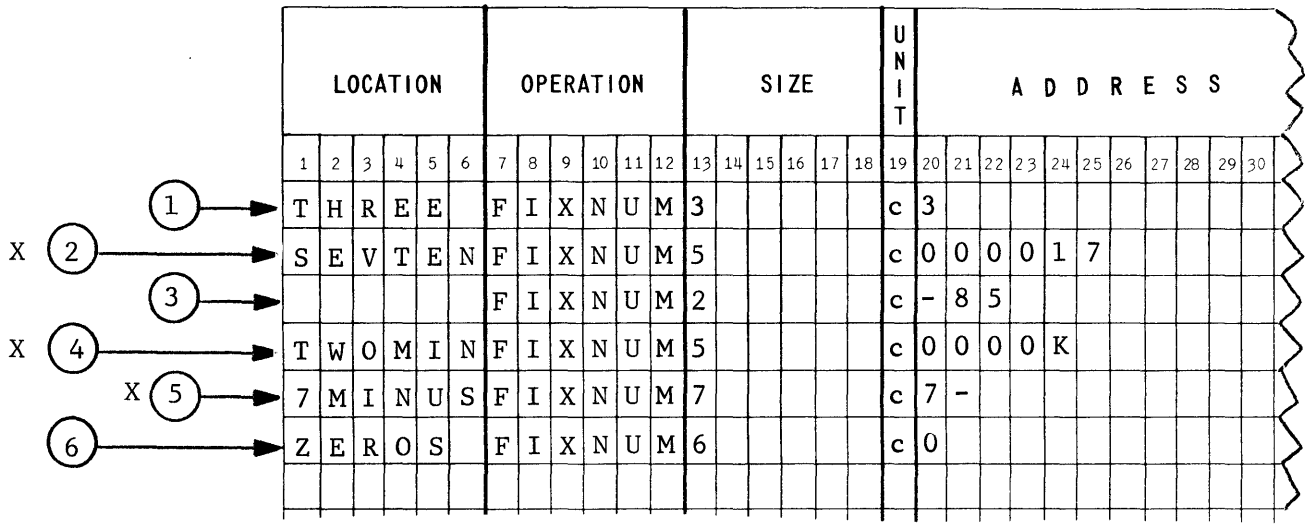
Constant is stored in the next available unit of memory as indicated in this field. (See ALOC entry for this field.)

(5) ADDRESS

The value of the constant is stored in this field. If the SIZE Field specifies a greater number of characters than the value in this field (as above) the stored constant will be zero filled (i.e., 001 above). If the value of the constant is minus, the minus sign must precede (be the first character) in this field.

FIXNUM Format

FIXNUM (EXAMPLES)



	LOCATION	OPERATION	SIZE	UNIT	ADDRESS
1	THREE	FIXNUM	3	c	3
X 2	SEVTEN	FIXNUM	5	c	00017
3		FIXNUM	2	c	-85
X 4	TWOMIN	FIXNUM	5	c	0000K
X 5	7MINUS	FIXNUM	7	c	7-
6	ZEROS	FIXNUM	6	c	0

FORMAT NOTES

(1) The stored constant will be 003 (zero filled).

X (2) The stored constant will be 00001 - Truncation of the 7 will occur because of SIZE entry.

(3) The stored constant will be 8N.

X (4) Error entry - Alpha character not permitted.

X (5) Error entry - Minus sign must precede entry.

(6) The stored constant will be 000000.

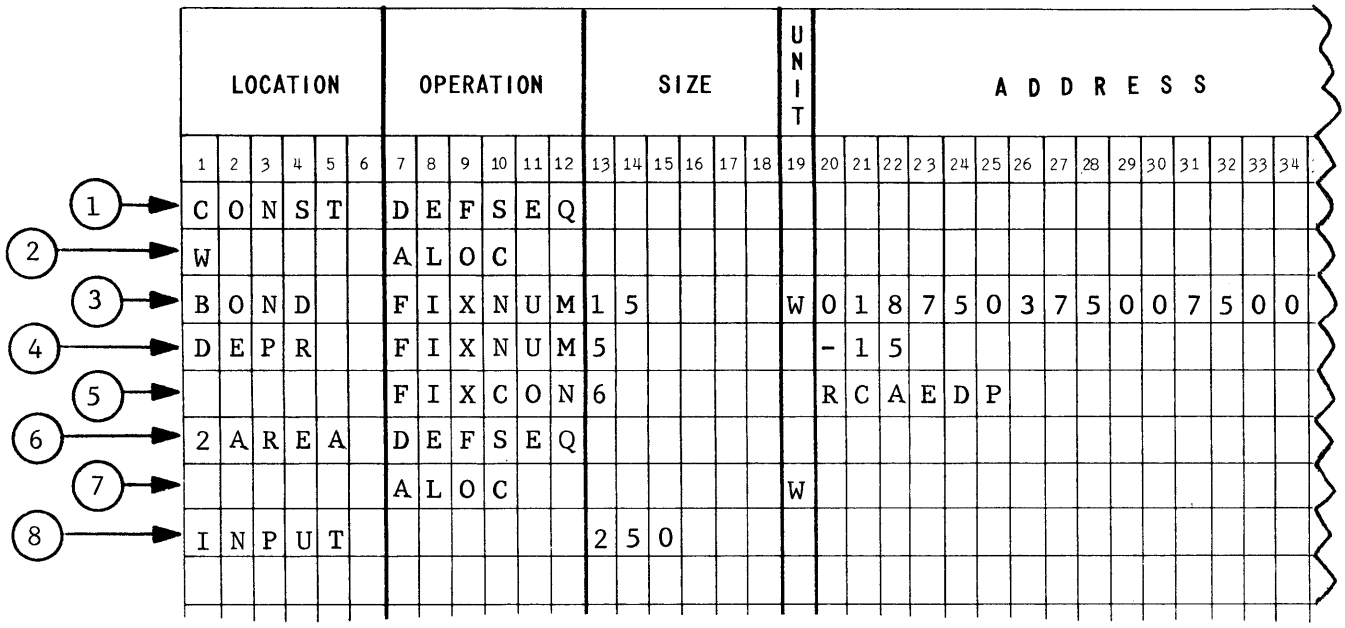
(At least one numeric character must be present in the ADDRESS field.)

FIXNUM
Examples

DEFSEQ
ALOC
FIXCOM
FIXNUM

} EXAMPLES

(SAMPLE CONSTANT SEQUENCE)



COMMENTS

NOTES

(1) A tag must appear in the LOCATION Field for each DEFSEQ line.

(2) The ALOC entry is used here only to furnish a prefix (W) to each tag that follows. The FIXNUM and FIXCON codes will allocate memory. Thus, this example could have omitted the ALOC line as follows:

```
CONST     DEFSEQ
WBOND     FIXNUM   etc.
WDEPR     FIXNUM   etc.
          FIXCON   etc.
```

(3) WBOND is word oriented.

(4) Even though the previous entry WBOND is word oriented it contains only 15 characters (1-1/2 words). Thus, WDEPR, which is character oriented, will use the last five character locations of the second word allocated for W.

(5) Constant of six characters is not tagged. Reference to the first character of this field would be WDEPR+$5.

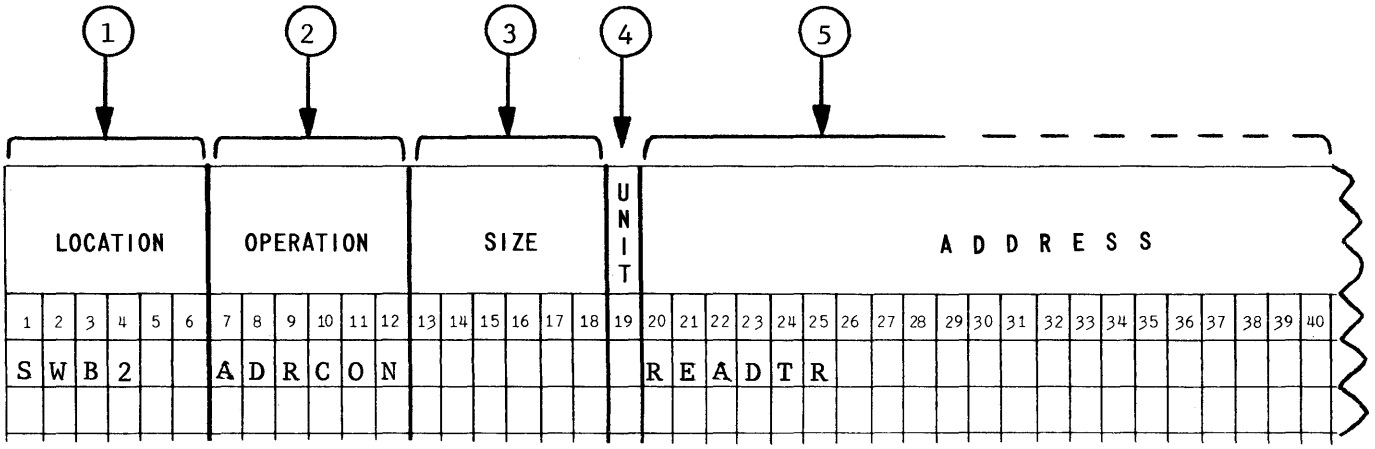
DEFSEQ
ALOC
FIXCON
FIXNUM
Examples

COMMENTS	NOTES
(6) (7) (8) A separate sequence in which a 250 character area is allocated and is word oriented.	

DEFSEQ
ALOC
FIXCON
FIXNUM
Comments

ADRCON

This code is used to generate an address as a constant.



FORMAT NOTES

(1) LOCATION

The Address Constant is referenced by the tag which appears in this field.

(2) OPERATION

ADRCON must appear in this field.

(3) SIZE

NOT USED

(4) UNIT

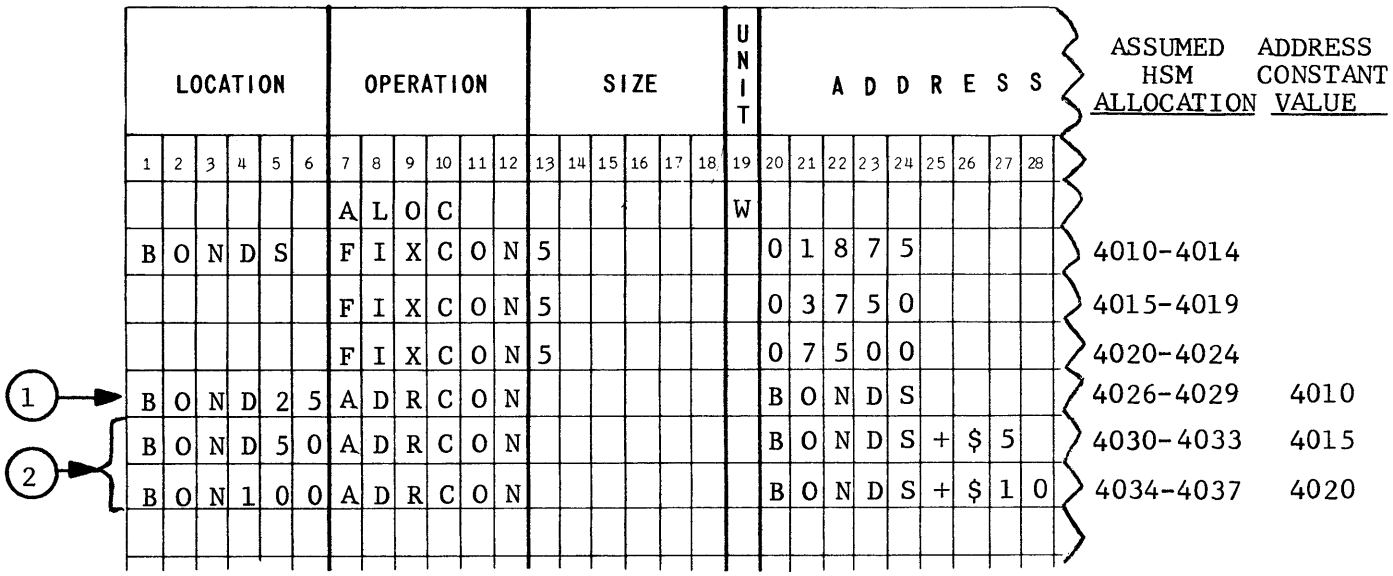
Specifies the type of orientation for the four character positions allocated as follows:

C	- Character Orientation
D (or space)	- Diad Orientation
W	- Decade Orientation
H	- Hundreds Orientation
T	- Thousands Orientation

(5) ADDRESS

A symbolic name or an actual machine address may appear in this field. For a pure symbolic name, the address stored as a constant will be the left-end address.

ADRCON Format



	LOCATION	OPERATION	SIZE	UNIT	ADDRESS	ASSUMED HSM ALLOCATION	ADDRESS CONSTANT VALUE
		ALOC		W			
	BONDS	FIXCON	5		01875	4010-4014	
		FIXCON	5		03750	4015-4019	
		FIXCON	5		07500	4020-4024	
(1) →	BOND25	ADRCON			BONDS	4026-4029	4010
(2) →	BOND50	ADRCON			BONDS+$5	4030-4033	4015
(2) →	BON100	ADRCON			BONDS+$10	4034-4037	4020

COMMENTS

NOTES

(1) Address value stored for a pure symbolic tag (BONDS) is the left-end address (4010).

(2) Address value stored for an incremented (or decremented) symbolic tag is the location referenced.

ADRCON
Examples

RCA 3301 ASSEMBLY SYSTEM

PRACTICE QUESTIONS

REFERENCE T & E MANUAL SECTIONS I-III

1. Indicate the addresses assigned to each of the following fields assuming the Location Counter was set in the beginning at 6000:

LOCATION	OPERATION	SIZE	UNIT		
W	ALOC		C		
ACCT		3	C	_____	_____
CODE		5	D	_____	_____
NAME		17	W	_____	_____
BAL		8	W	_____	_____
WORK		10	W	_____	_____

2. Using the fields in question 1 as a reference, would the following addresses in an instruction be valid in the ADDRESS Field? Explain.

BAL,WORK

3. Assume the following fields in Col. I have addresses assigned as in Col. II (as LHE assignment).

COL. I	COL. II
ABLE	2000
TIME	2425
CHECK	2650
BAKER	2730

What machine addresses would be assembled for each of the following?

ABLE+BAKER _____
TIME+CHECK _____
CHECK-TIME _____
CHECK+$50 _____
TIME# _____

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 1

GENERAL REQUIREMENT

Prepare a sequence to be used for setting up a file record area, read-in areas, and fixed numeric and alphanumeric constants.

FORMAT REQUIREMENT

1. The format of the record is as follows:

ITEM NAME	NO. OF CHARS.
ACCT NO	9
CLASS OF ACCT	2
NAME OF CUSTOMER	20
STREET ADDRESS	15
CITY STATE CODE	2
CREDIT CODE	2
AMOUNT DUE	7
ARREARS CODE	3
TOTAL NO OF CHARS	60

2. Assign a five character tag for each item and the first character of each tag must be an "M." This record area is decade oriented.

3. These records are batched by 5 records to a block on tape so allocation of a file read-in area must also be made in this sequence. The tag of this area is also a five character tag with the first character being an "F."

4. To allow for maximum simultaneity, allocate an alternate read-in area with the first character of a five character tag being an "A."

5. In addition to the record, file, and alternate areas, set up the following constants within this sequence as follows:

   a. A table of two character constants with the values 01, 04, 07, 10, and 24 respectively with the beginning of the table decade oriented. The tag of this table will be a five character tag, the first character of which will be an "M."

   b. Four constants of the following values:

      END OF RUN
      ACCOUNT NO
      ENTRY DATE
      CLOSE FILE

      Each tag will be a five character tag with the first character being an "M."

SECTION IV

RENAMING AND REDEFINING DATA AREAS

Section III was an explanation of how data areas and numeric and alphabetic constants may be allocated.

In this section we will discuss how such areas may be RENAMEd and REDEFined.

RENAME

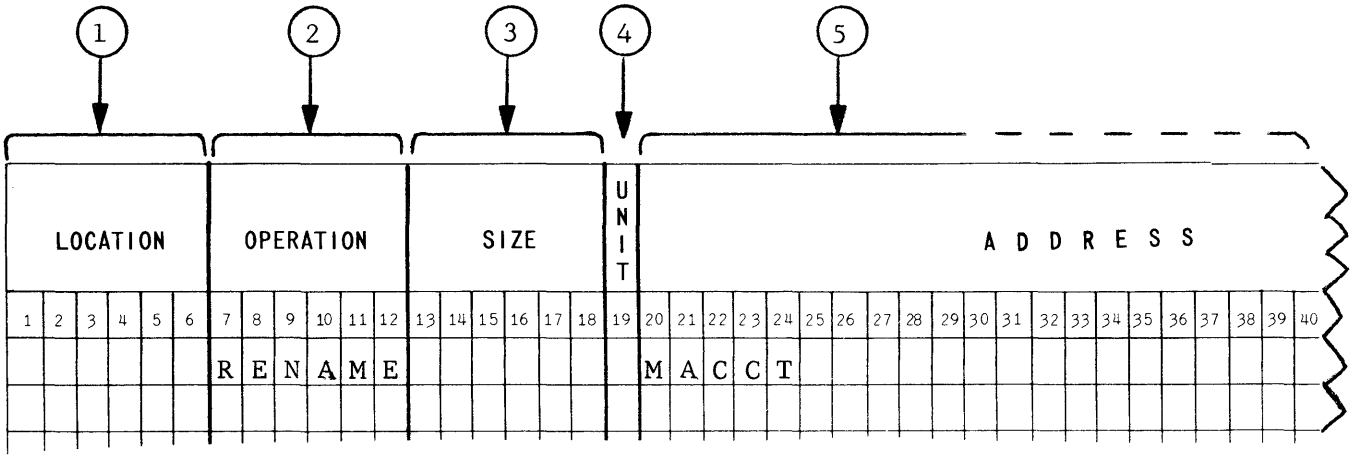
As the name of this controlling code implies, it enables the user to assign a different name (or names) to an area which has been previously named (tagged).

The word "previously" in the preceding sentence refers to the order in which statements are entered to the assembly process. In other words, the area to be renamed must have been described to the Assembler prior to using the RENAME option; i.e., the RENAME lines will have higher reference keys.

The format and examples of the RENAME option follow.

RENAME

To rename an area previously defined in the same sequence.



FORMAT NOTES

(1) LOCATION

Not used

(2) OPERATION

RENAME must appear in this field.

(3) SIZE

Not used on a RENAME line.

(4) UNIT

Not used on a RENAME line.

(5) ADDRESS

The name (tag) of an area previously defined in this same sequence. Tag relative (+ or -) addresses must not be used.

RENAME
Format

Example - Breaking down an item into sub-items

In this example, let us assume that a programmer in describing his record area has provided for a ten character account number as follows:

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	ASSUMED HSM ALLOCATION
1 2 3 4 5 6	7 8 9 10 11 12	13 14 15 16 17 18	19	20 21 22 23 24 25 26 27 28 29 30	
F I R S T	D E F S E Q				
M	A L O C		W		
A C C T		1 0	C		→ 0000-0009

This account number (MACCT) however consists of significant sub-items as follows:

1st CHAR. - STATION NO.

2nd and 3rd CHARS. - CYCLE NO.

4th CHAR. - BOOK NO.

5th thru 9th CHARS. - CUSTOMER NO.

10th CHAR. - CUSTOMER SUFFIX

The programmer may now RENAME the MACCT Field as shown in the example which follows:

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	ASSUMED HSM ALLOCATION BASED ON PRIMARY FIELD
1 2 3 4 5 6	7 8 9 10 11 12	13 14 15 16 17 18	19	20 21 22 23 24 25 26 27 28 29 30	
	R E N A M E			M A C C T	
S T A N		1	c		→ 0000-0000
C Y N O		2	c		→ 0001-0002
B K N O		1	c		→ 0003-0003
C U S N		5	c		→ 0004-0008
C U S X		1	c		→ 0009-0009
	A L O C				

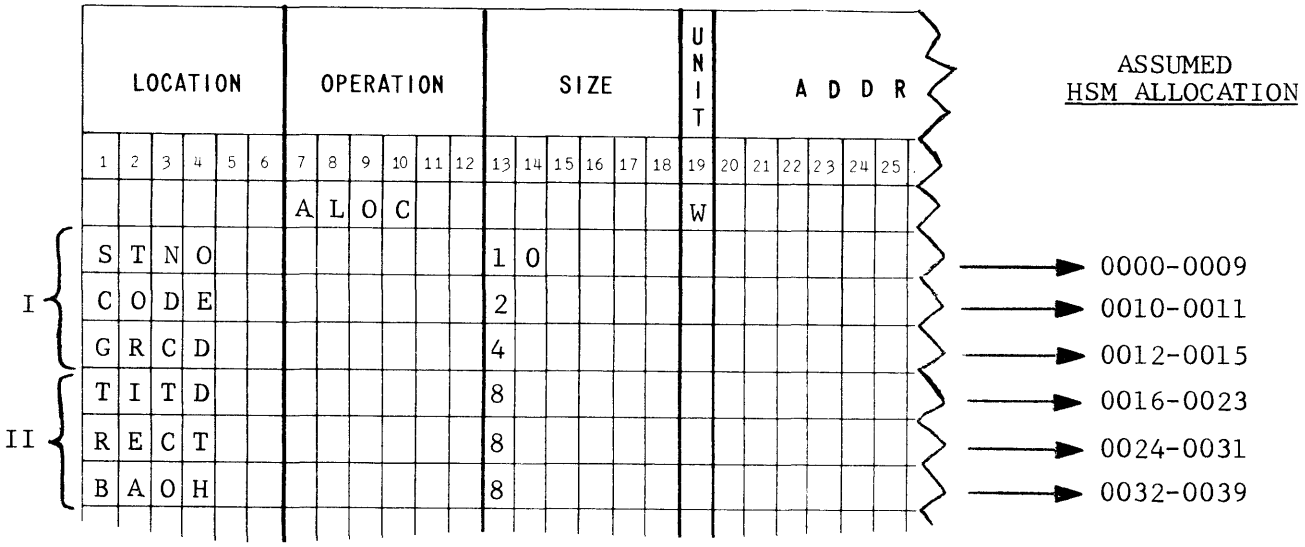
Assuming the RENAME option falls within the preceding ALOC area, the programmer now has the ability to use either the tag of the 10-character field (MACCT) or the tags of individual sub-items (MSTAN, MCYNO, MBKNO, MCUSN, and MCUSX).

RENAME Examples

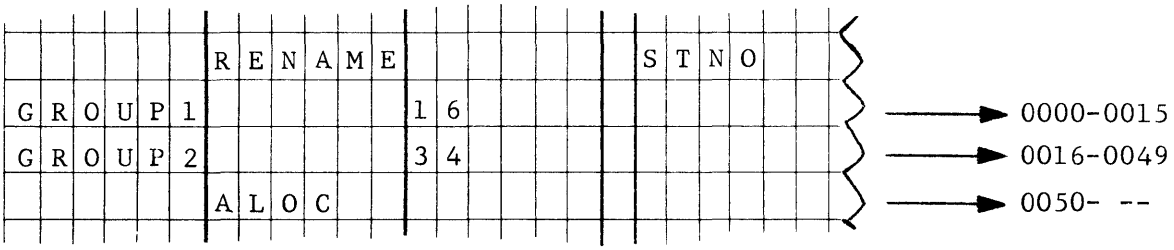
The RENAME option thus makes it unnecessary for example to use the tag MACCT+$3 to refer to the sub-item for the Book No. (MBKNO).

Example - Grouping of Items

In this example let us assume that the programmer had defined and allocated a forty character area as follows:



At a point in his program however he wants to handle the first three items (I above) as a unit, and the second three items as a unit also (II above). In addition, as a part of the second group he wants to add a 10 character work area. He might do this immediately after the last entry above and using the RENAME option as follows:



Note that the RENAME option used in this way must <u>immediately</u> follow the primary area. This is necessary because the Location Counter has been advanced an additional ten locations (to 0049) over its final setting for the primary field (0039).

Unless immediately following, the RENAMEd field would have included the first 10 characters of the field following the Primary field.

RENAME
Examples

Summary of Important Points

1. The RENAME controlling code must be preceded in the assembly process by the description of the area being renamed.

2. If the RENAMEd area falls within a ALOC entry with a prefix assigned, that prefix character will also be assigned to the RENAMEd Fields.

3. The RENAME option must be terminated by an entry in the OPERATION Field other than REMARK.

4. An incremented or decremented address (+ or -) cannot be used in the ADDRESS Field of a RENAME line.

5. The type of orientation (character, diad, decade, etc.) should be mutually consistent in the previously defined and the RENAMEd Fields.

RENAME
Summary

REDEF

The REDEFine controlling code is very similar in one of its functions to the RENAME controlling code. It may be used to assign a different set of names (tags) to a previously allocated area.

The primary purpose of the REDEFine code, however, is to utilize a previously allocated sector of memory for the storage of data in either a different format or with different assigned values.

An example of the use of REDEFine would be in a situation in which a user had a file in which records were in different formats. In this example, let us assume that a user had a transaction tape on which there were three types of transactions; receipts, payments, and new accounts with the following format for each type respectively:

Receipts		Payments		New Accts.	
Acct. No.	10	Acct. No.	10	Acct. No.	10
Code	2	Code	2	Code	2
Date	4	Date	4	Date	4
Amt. Rec'd	7	Amt. Paid	7	Customer Name	25
Total Chars.	23	Total Chars.	23	Street Address	27
				City State Code	3
				Type of Account	2
				Amount of Purchase	7
				Total Chars.	80

In this example the user will allocate a read-in area for the largest size record, 80 characters in this case, and then redefine the allocated area for the receipt and payment type of transactions. So that, in this example, following the read of the transaction tape, the user determines the type of transaction that is present in memory by use of the two-character code which is of a different value for each type of transaction.

An example of this use of the REDEFine code is furnished following the format of the REDEFine line.

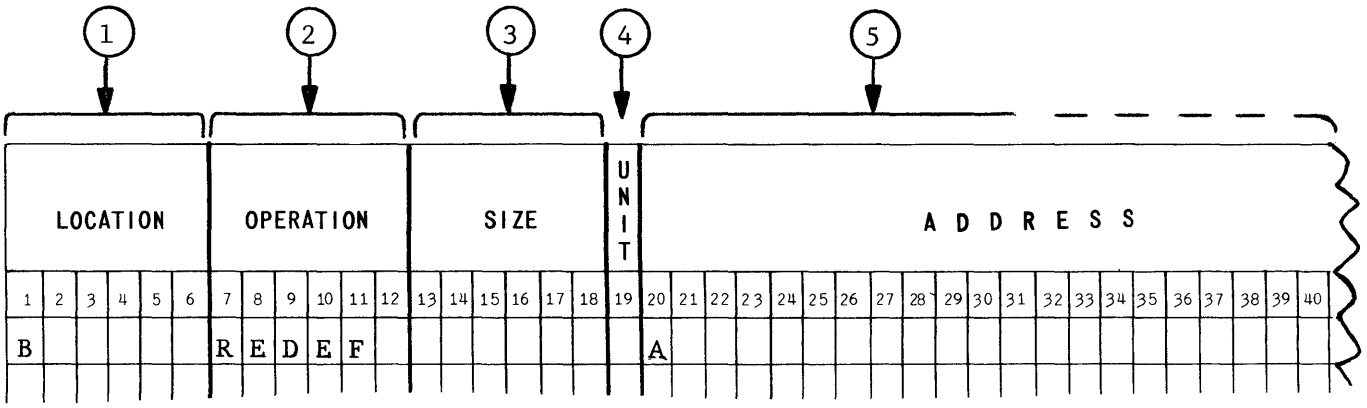
Another example of the use of this controlling code is the sharing of data work areas -- assuming that a programmer had need of two separate work areas. If these two areas did not have to exist concurrently he might share the use of the common area by allocating it for one work area and REDEFining it for the other work area.

Edit areas may be shared in the same manner. Assuming that a programmer will determine whether a record must be printed or punched as an error, he might share a common area in the same manner.

We might say then that the REDEFine option has three major functions; it may be used to define an area for different purposes; it may be used to give different values to a common area of memory; it may be used to save memory.

REDEF

Used to specify different values for an area of memory previously defined and allocated by an ALOC controlling code.



FORMAT NOTES

(1) LOCATION

This is an optional entry of one character. If used, it will be considered a prefix to the tags in the following lines in the REDEF area. The LOCATION Field entry may be the same character assigned on the ALOC line or a different character. (See General Comment below.)

(2) OPERATION

REDEF must appear.

(3) SIZE

Not used on the REDEF line.

(4) UNIT

Not used on the REDEF line.

(5) ADDRESS

Mandatory entry of one character that has been previously specified in the LOCATION Field of an ALOC line. Note that the REDEF option, if used, makes it mandatory that the previous ALOC line contain a prefix character assignment.

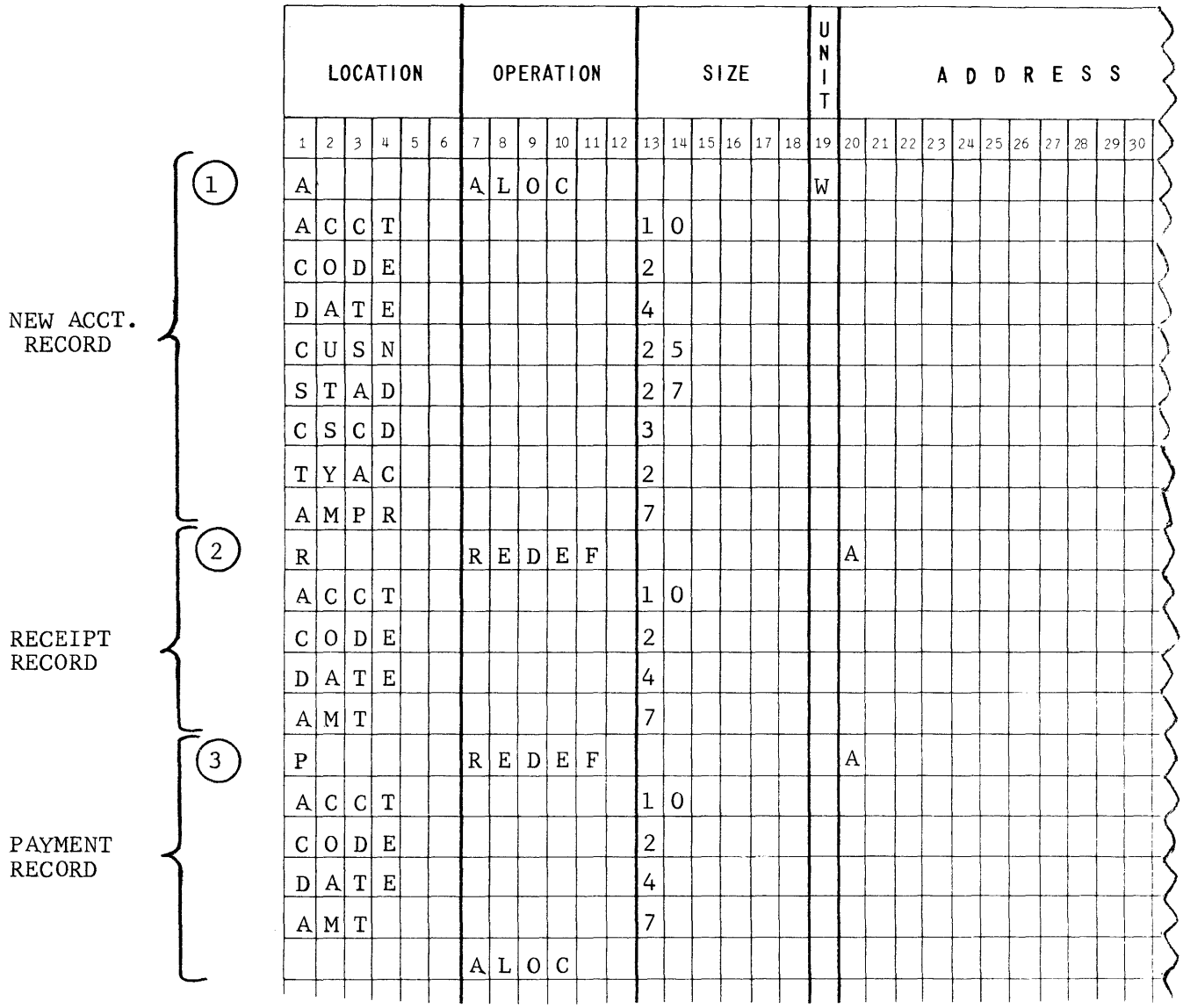
GENERAL COMMENT

It should be noted that the REDEF controlling code terminates the ALOC controlling code operation. If additional areas are to be allocated following the REDEFined area(s) another entry in the LOCATION Field would be required; i.e., ALOC, FIXCON, FIXNUM, INSTRUCTION, OPERATION CODE, etc.

REDEF
Format

REDEF

(Example based on the situation on page III-59 in which an input area is allocated and defined for different record formats.)



	LOCATION						OPERATION						SIZE						UNIT	ADDRESS												
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
(1)	A						A	L	O	C									W											
NEW ACCT. RECORD	A	C	C	T									1	0																
	C	O	D	E									2																	
	D	A	T	E									4																	
	C	U	S	N									2	5																
	S	T	A	D									2	7																
	C	S	C	D									3																	
	T	Y	A	C									2																	
	A	M	P	R									7																	
(2)	R						R	E	D	E	F									A									
RECEIPT RECORD	A	C	C	T									1	0																
	C	O	D	E									2																	
	D	A	T	E									4																	
	A	M	T										7																	
(3)	P						R	E	D	E	F									A									
	A	C	C	T									1	0																
PAYMENT RECORD	C	O	D	E									2																	
	D	A	T	E									4																	
	A	M	T										7																	
							A	L	O	C																				

COMMENTS NOTES

(1) A prefix must be given an ALOC line if the ALOC area is to be referenced by a REDEF controlling code.

(2) (3) The REDEF line must refer to the ALOC prefix character (A) in the ADDRESS Field. Note that the UNIT Field on each of these REDEF lines is left blank. Word orientation will take place, however, based on the ALOC UNIT Field entry of W.

REDEF
Example 1

Example of Combined Usage

This example is based on the assumption that the user is allocating an area, renaming a portion of the allocated area and then is redefining the area for a record of a different format.

	LOCATION	OPERATION	SIZE	UNIT	ADDR	ASSUMED HSM ALLOCATION
1	N	ALOC		W		0000-0053
2	ACNO		8			0000-0007
3	CODE		2			0008-0009
4	NAME		25			0010-0034
5	STAD		17			0035-0051
6	CSCD		2			0052-0053
7		RENAME			NACNO	
8	STAT		4			0000-0003
9	CUSN		4			0004-0007
10	M	REDEF			N	
11	ACNO		8			0000-0007
12	CODE		2			0008-0009
13	AMT		7			0010-0016
14	ONE	FIXCON	3	1		0054-0056

COMMENTS NOTES

(1) The allocated area is Word oriented.

(2) thru (6) and (8), (9)

The prefix character of the ALOC line (N) will be assigned to all tags on these lines; i.e., RENAME does not terminate the ALOC function.

(7) RENAME used here to obtain prefix of N to the subdivided first field (NACNO). Note that if the RENAME Field (lines (7), (8), and (9)) followed the REDEF area, no prefix would be assigned.

REDEF
Example 2

COMMENTS NOTES

(8) (9) These lines rename the NACNO Field so that it is subdivided into two four character fields, NSTAT and NCUSM.

(10) Note that the entry in the ADDRESS Field must refer to the previous ALOC tag (1 character).

(11) (12) (13) Prefix of M assigned to these tags.

(14) No prefix assigned to the tag (ONE) of this field as REDEF function is terminated by an entry in the LOCATION Field.

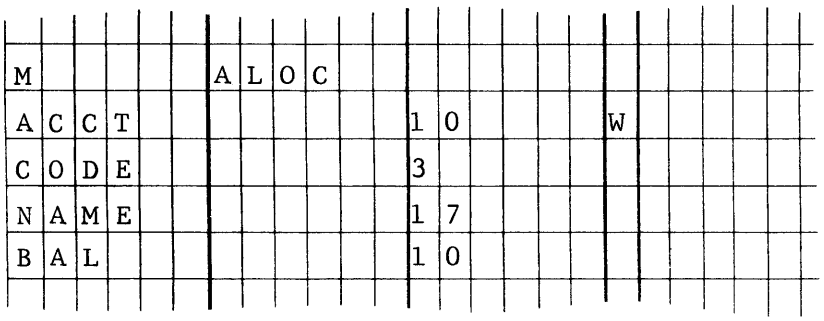
(14) Note that the HSM allocation (0054-0056) is set on the basis of its last setting, line (6).

REDEF
Example 2

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 1A

Assume the following is a portion of a sequence. It will be used as a reference for the questions that follow.



1. Rename the MACCT Field so that the first two and last two characters may be addressed with pure symbolic names.

2. Redefine M so that it may be used for an area having the following format:

ACCOUNT NO.	10
CODE	3
AMOUNT	7

3. Assuming the addresses assembled for MACCT were 0110-0119, what addresses would be assembled for MBAL?

SECTION V

PREPARATION OF THE FILE CONTROL PROCESSOR

FILE SEQUENCE

Thus far in the Sections in this Assembly System part of the Manual, we have discussed the format of the Assembly Program Sheet, and how to allocate, rename, and redefine data areas.

In this, and the following section, we will discuss topics related to input and output functions under control of the File Control Processor (FCP).

For the purposes of simplicity, only the primary level of FCP or file control will be discussed in this and the following section. The secondary level called device control will be discussed in an Appendix at the end of this part of the manual.

Also omitted from this section will be options that a programmer might use for processing in place by utilizing Index Fields. These methods are explained also in an Appendix at the end following a discussion of the use of Index Fields in a succeeding section.

The user should be aware that FCP will control an entire range of peripheral devices including magnetic tape, paper tape, card, and printing devices to name just a few. Also, the user should be aware that at execution time, interchangeability of selected devices may be made. As an example, a file may be written to a magnetic tape instead of the printer in certain situations.

The subject material in this and the section that follows will be developed on the basis of the solution of a small problem involving the duplication of records on an output file with a different batch size. The problem will be concerned with the allocation of data areas, preparing the FCP file sequence, and using the file controlling codes.

Before presenting the problem, however, certain basic FCP conventions and required formats must be explained with emphasis on the particular conventions and format requirements to be used in the problem.

RECORD FORMAT CONVENTIONS

There are four record formats and they are referenced by types A, B, C and D.

Type

A — Fixed Length or Variable Length Records, Character Oriented and Unbatched

May only be processed as unbatched. User must provide sufficient area for largest record in the file.

*B — Fixed Length Word Oriented

Requires least amount of time to process.
User provides INPUT (or OUTPUT) area and, if desired, a record area. All areas must be decade oriented.
Maximum record size is 4500 characters (450 words).
May be processed when in batched format.

*(Explanatory problem will use this format.)

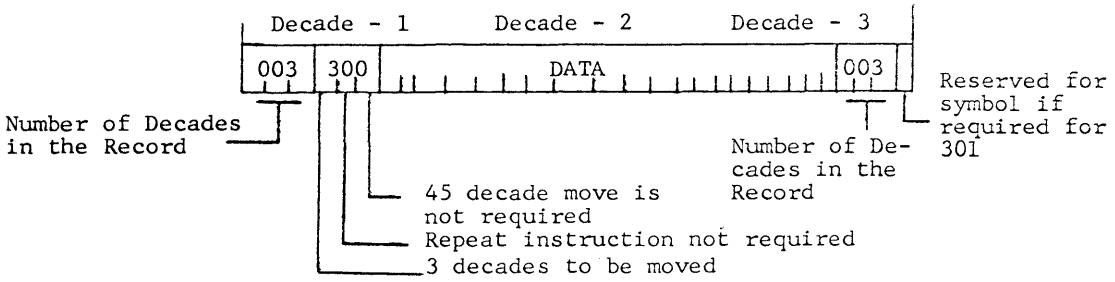
C Variable Length, Decade Oriented, Count in Record Controlled

Facilitates processing of variable sized records.
The first six characters are used for FCP processing control.
The last character position is reserved for a record symbol if used.
The three characters adjacent (to the left) of the last character are for FCP processing control when records are processed in a reverse direction.
User provides Input (or Output) area and if desired a record area equal in size to the largest record of the file. All areas must be decade oriented.
May be processed when in batched format.

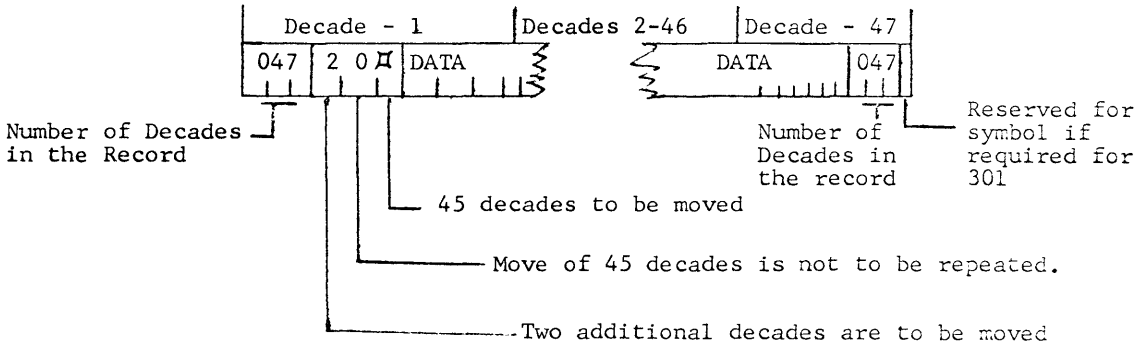
NOTE: FCP will generate in output records a decade count and control field characters only if, at the time of a WRITE (See Section VI) is issued, the first three characters are zero.

EXAMPLES OF RECORD TYPE C

Three decade record



Forty-seven decade record



D Variable Length, Character Oriented, Symbol Controlled

This type of record may be processed only in a forward direction.
A user specified symbol, which may not otherwise be used, is the last character of the record.
User provides Input (or Output) and a record area equal to the largest record of the file.

BATCHING

There are four types of batches and these also are referenced by the letters A, B, C and D.

Type

*A Fixed Number of Fixed Length Records

B Fixed Number of Variable Length Records

FCP handles both types of batches in essentially the same way.
For output, it constructs a batch based on a counter and when the required number of records have been accumulated, the batch is written to the output device.
The last block does not have to be a "full" batch.

C Variable Number of Fixed Length Records

FCP processes essentially the same as Batch Type A.
User specifies maximum number of records per batch and provides sufficient areas for Input (or Output) batch.
User may cause FCP to physically write a batch of a lesser than maximum number of records. (See RELS controlling code - Section VI.)

D Variable Number of Variable Size Records

FCP checks to see if each succeeding record will fit in a batch, the size of which is user specified.
If record will not fit, the batch is physically written, and the current record is used to begin construction of another batch.
Valuable option where wide variation in actual record sizes exists.

*Explanatory problem will use this format.

LABELS

For FCP to properly verify data files, labels may be used.

If standard labels are used, FCP will check such labels on input files and create appropriate labels on output files.

The user has three options in place of the use of standard labels. He may specify and use labels of his own design or he may use the standard label with added fields. He may also omit the use of labels for all or selected files.

In addition to the above options, the user may omit just the beginning or the ending label.

Regardless of the type of label in use, an E/I character must be the last character of the label.

For a non-standard (user-designed) label, the user must provide his own label creation and checking procedural routines. The first character of a non-standard label may not be an E/B character.

For labels which include the standard label fields plus label fields of his own design, FCP will pass control to the user for the creation and checking of such fields as he has specified. It will be the user's responsibility to overlay the terminal E/I at the end of the standard fields and place an E/I at the end of his label fields.

The format on tape is as follows:

Beginning Format

1. If a beginning label is used, it is the first block on tape followed in turn by an E/F character block and the first data block.

2. If a beginning label is omitted, the E/F character block is the first block followed by the first data block.

Ending Format

1. The ending label, if present, is always preceded by an E/F character block.

2. Following the ending label is an E/D character block or an E/F character block and an E/D character block depending upon whether the reel is an intermediate or final reel of a file respectively.

3. If the ending label is omitted an E/D character block follows the last data block on an intermediate reel. An E/F character block and an E/D character block follow the last data block of the last reel of a file.

*Standard Label Format

The beginning label has 29 characters in the following sequence of fields:

Field	No. of Chars.
E/B Char.	1
(Space)	1
File Ident. (User-Named)	8
(Space)	1
Reel-No	3
(Space)	1
Date Written	6
(Space)	1
Purge Date	6
E/I Char.	1
	29

The ending label has 19 characters in the following sequence of fields:

Field	No. of Chars.
E/B	1
(Space)	1
Block Count	7
(Space)	1
File Ident. (User-Named)	8
E/I Char.	1
	19

*(Explanatory problem will use this format.)

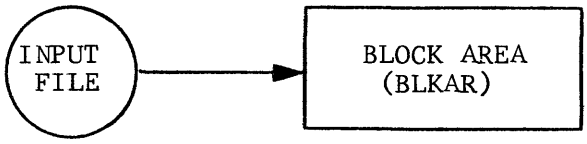
SIMULTANEITY

Simultaneity is the sharing of time of the 3301 Processor between I/O and other processing functions. The FCP will provide control of input/output functions and will attempt to provide the maximum saving of time through use of simultaneity.

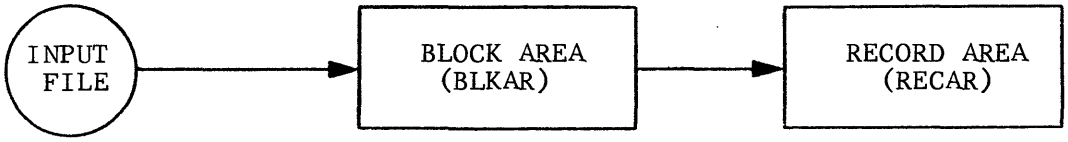
The user, however, should be aware that he must allow sufficient areas in memory to obtain the maximum benefit of this option.

For an easy to understand example, let us consider the processing of an input tape with single record blocks.

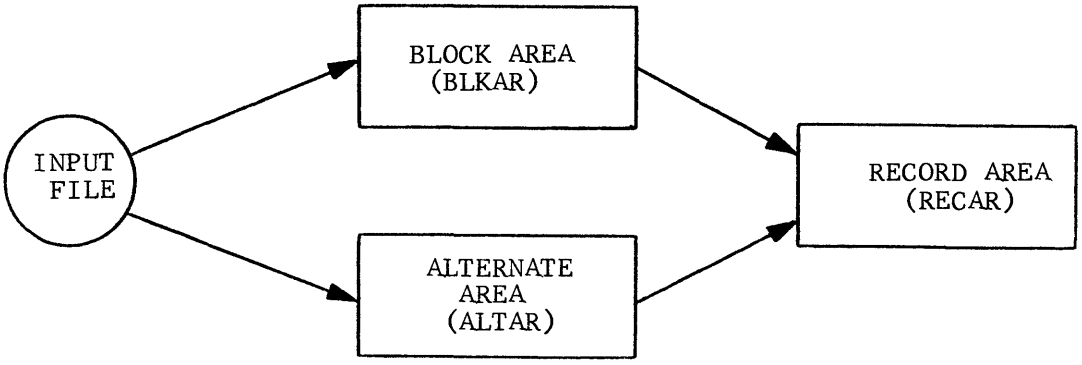
If the user provides only an input block area (BLKAR) for one record and this area is to be used also for processing, no simultaneity on this file will be obtained. FCP has no other area to read into while the record is being processed. This might be illustrated as follows:



If, however, the user provides FCP with a Record Area (RECAR) as well as an Input Block Area (BLKAR), he would obtain a degree of simultaneity. While processing the Record area, FCP could be reading into the Block area. This might be illustrated as in the following:



To provide for the maximum simultaneity, the user might provide still another (alternate) area (ALTAR). With this added area, FCP now can have two areas to read into alternately as each is moved to the Record area. This is illustrated below:



*(Explanatory problem will illustrate this option.)

While the above illustrates the allocation of areas for an input file, the same concepts apply to output files. For the maximum simultaneity FCP must have the areas within which to operate.

EXPLANATORY PROBLEM

With an understanding of some of the major functions of FCP as explained thus far in this section, let us now see how we can associate the allocation of areas previously discussed in the preceding sections of this manual with the requirements demanded by FCP.

To take a fairly simple problem involving only input and output processing, let us say that a bank, we will call it the First Savings Bank, has decided to change the batch size for a particular file. It has also been decided to write a short "run" to do this job.

The input file consists of 100 character fixed length records and the records are batched by eight on the input tape. The file Identification for label purposes is "DEMDEPFI".

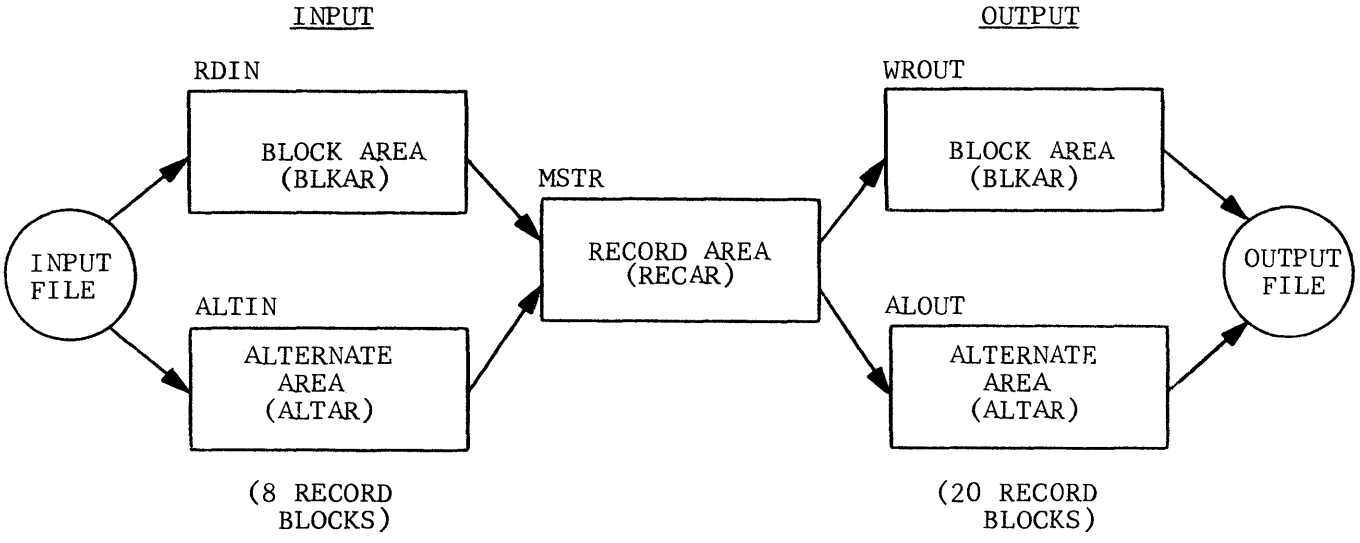
The output file is of the same record size and format but the batch size is to be twenty records. The file ID, for label purposes of course, will remain the same.

The record format, while not pertinent to the solution of this problem, must (in accordance with the standards of the First Savings Bank) be fully described for all input and output files. The record format is as follows:

Item	No. of Chars.
Acct. No.	10
Name	25
Street Address	20
City State Address	20
Total Deposits	9
Total Checks	9
Balance	7
Total Chars.	100

The record format for FCP purposes is Record Type B -- a Fixed Length, Word Oriented Record. The Batch Type is Type A -- a Fixed number of Fixed Length records on both input and output files.

In the solution of our problem we will provide for the maximum amount of simultaneity by providing input block and alternate areas for each file and a common record area for both files as illustrated below:



Before attempting to prepare the FCP File Sequence we must first allocate the memory for our input, output and record areas.

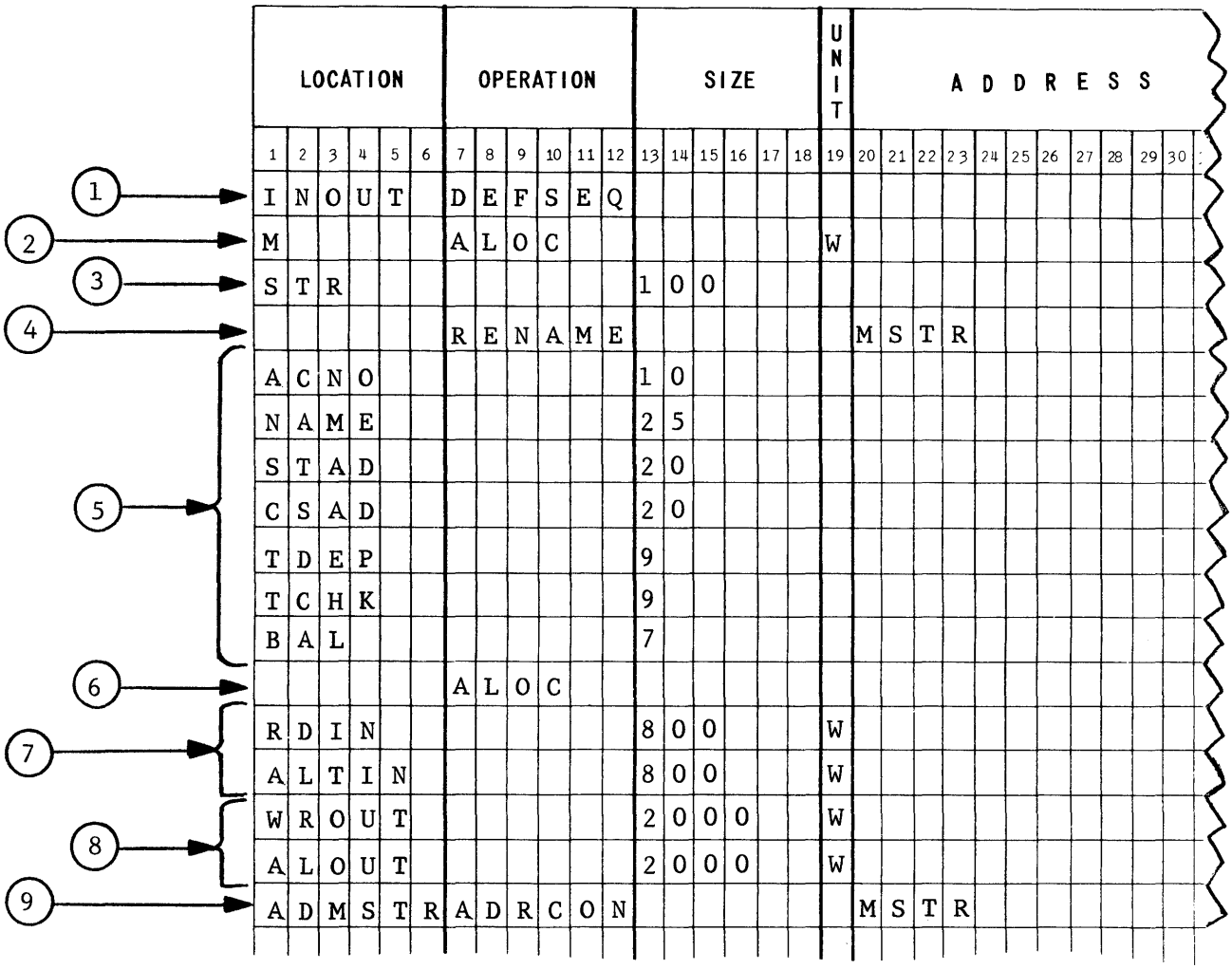
We will do this in a separate sequence to which we will give the symbolic name INOUT.

To each of the other areas we will give the symbolic names as indicated at the top of each of the boxes in the above illustration.

In addition, we will describe the Record Area item names using the RENAME controlling code and prefix all item names with an M.

EXAMPLE

Based on preceding problem.



COMMENTS NOTES

(1) Every DEFSEQ must be given a symbolic name.

(2) LOCATION entry provides prefix character of M.

(3) This line allocates the Record Area (referred to as RECAR in the FCP File Sequence).

COMMENTS NOTES

(4) RENAME code used to furnish item names for the MSTR record area.

(5) Used to describe record items and furnish symbolic tags.

(6) ALOC used to terminate use of prefix M on line (2)

(7) These provide for the input block and alternate areas.

(8) These provide for the output block and alternate areas.

Note that Word orientation (W) is provided in the UNITS Field, but would not be necessary as a 100 character word oriented area as provided above. It is not, however, an unnecessary precaution as the insertion of lines may destroy the desired positioning based on a preceding allocation.

(9) This entry is necessary for the output file descriptor sequence. An ADRCON is necessary for the OUTLHE entry requires a location where FCP can find the left-end address of the output record. (See the example for an output FCP File Sequence Page III-81).

PREPARATION OF FCP FILE SEQUENCE

Having allocated the necessary block, alternate, and record areas in the previous sequence and having a system knowledge of the input and output specification for labels, batch size, and record type, we can now prepare the FCP File Sequence for each file.

There must be a separate sequence for each file and this sequence will contain the information necessary for FCP to properly control the processing of this file.

For illustrative purposes, the complete format of the input and output file sequences will be shown. The symbolic names INFILE and OUTFILE will be given to the sequences for the input and output files respectively. The file descriptor sequences appear on pages III-80 and III-81 following the format requirements for the file descriptor sequence.

FORMAT REQUIREMENTS

The first line of the sequence, as in every sequence, is a DEFSEQ line. The following lines contain entries in the LOCATION Field as follows:

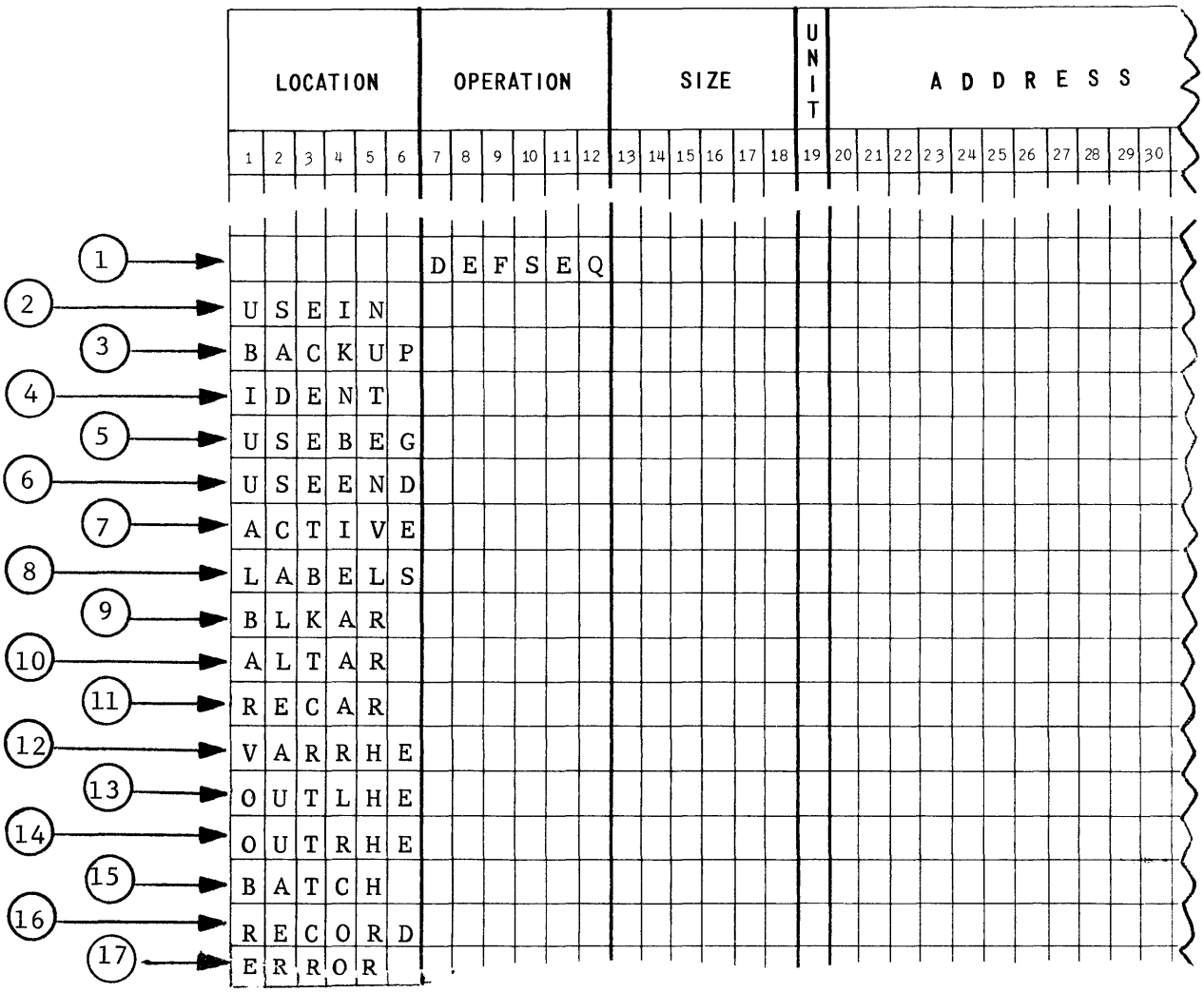
Device Region { USEIN / BACKUP

Label Region	IDENT USEBEG USEEND ACTIVE LABELS BLKAR
Data Region	ALTAR RECAR VARRHE OUTLHE OUTRHE BATCH RECORD ERROR

Each of the above entries in the LOCATION Field must appear even though the line may be for an option that is not desired.

An exception to the requirement for all of the entries would be when the user is exercising his own control over device(s) for a given file. In this case only the Device Region entries may be required. A full example of this exception appears in Appendix A - Device Control.

A brief description of the purpose and format for each of these entries follows.



FORMAT NOTES

1 DEFSEQ Line

The name assigned to the file must appear in the LOCATION Field.

In the ADDRESS Field of a file descriptor sequence the entries for Cols. 20 to 26 are required as indicated below:

COL.	REQUIRED ENTRY	REMARKS
20	F	File Sequence (Complete File Description generated)
	D	Device Sequence (Device Region generated) See Appendix on Device Control. *
	L	Device Sequence with FCP Label processing (Device and Label Regions generated) *
	P	Print File Sequence (Complete file description) *
	C	Card File Sequence (Complete file description) *
		* These include files to be written to or read from the respective devices directly or written to magnetic tape with control information for later use in transcribing to the device.
21	,	Comma required
22	Y	Interchangeability of device is permissible at object time
	N	Interchangeability of device is not permissible
23	,	Comma required
24	O	Optional Region type (Only permitted with types F, P, or C as indicated by Col.20.
	A	Always present.
25	,	Comma required only if the entry in Col. 26 which is optional actually appears.
26	W	This sequence to be WORD oriented
	H	HUNDREDS
	T	THOUSANDS

2 USEIN Line

ADDRESS Field may contain symbolic address of a routine the user desires to enter prior to physical initiation of an input/output instruction. In this routine, an FCP controlling code or a Repeat instruction may be used; however, the user must save the original setting of STPR and M1. At the completion of this routine,the user must return to FCP at the address originally stored in STPR.

3 BACKUP Line

Required entry if an output file for card punching. ADDRESS Field must contain symbolic address of an area to be used for error recovery. Must be a minimum area of 16 or 32 decades.

FORMAT NOTES

(4) IDENT Line

ADDRESS Field must contain the Label Identification Item if standard or combined labels are used as either beginning or ending labels.

(5) USEBEG Line

ADDRESS Field contains symbolic address of a routine to be used during label processing for the beginning label. At the end of the routine, the return to FCP is by an LDP instruction with addresses as follows:

A ADDRESS	B ADDRESS	
$STPR	$0	To continue normal processing
	$1	To reject a file or reel in this routine

This entry must be filled when the beginning label is combined or non-standard. No FCP controlling codes may be used in this routine.

(6) USEEND Line

ADDRESS Field contains symbolic address of a routine to be used during label processing for the ending label. At the end of the routine, the return to FCP is by an LDP instruction with addresses as follows:

A ADDRESS	B ADDRESS
$STPR	$0

(7) ACTIVE Line

ADDRESS Field for output file contains the number of days (three decimal numerics) to be used to create the purge (inactive) date. If entry is 000 or left blank, the current date at object time will be used for the purge date.

(8) LABELS Line

ADDRESS Field contains the type of label used for the beginning and ending labels. The beginning label type is indicated in Col. 20, a comma is entered in Col. 21, and the ending label type is indicated in Col. 22. The label type is indicated with one of the following characters in Cols. 20 and 22.

S - Standard Label
C - Combined Label
O - Labels Omitted
N - Non-Standard Labels

Entry must be present unless Device Control Processing without labels. (See Appendix A - Device Control.)

FORMAT NOTES

(9) BLKAR Line

ADDRESS Field must contain the symbolic name of the input or output area for the file. The area allocated must be large enough to accept the largest block of the file, including the labels. The area must be decade oriented and should be an integral number of decades in size.

(10) ALTAR Line

ADDRESS Field contains the name of an alternate input or output area if one is used for this file. If assigned, it must be the same size as the area allocated in the BLKAR line, and oriented in decades.

(11) RECAR Line

ADDRESS Field contains the address of an area used for processing a record. For other processing methods, the use of this entry is described in Appendix B - In-Place Processing.

(12) VARRHE Line

ADDRESS Field must contain an entry for an input file consisting of variable length records. The entry is the address of a four character location. FCP will use this location to store the address of the rightmost character of the record.

(13) OUTLHE Line

ADDRESS Field must contain the location in which FCP can find the leftmost address of each logical record prior to the issuance of a WRITE. The ADRCON controlling code may be used to initialize this field. See Example 2 of this Section. This entry is not required for an output file using address modifier processing (see Appendix B - In-Place Processing).

(14) OUTRHE Line

ADDRESS Field must contain, for a file containing variable length records, the location in which FCP can find the rightmost address of each logical output record. The user must store the address in this location before issuing a WRITE.

(15) BATCH Line

SIZE Field is used for the entry of the character used to terminate a batch. This entry, if present, is left-justified (i.e., Col. 13).

UNIT Field contains an F if there is a Fixed number of records or V if there is a Variable number of records per batch.

FORMAT NOTES

(15) BATCH Line (Cont'd)

ADDRESS Field contains a four character decimal field (Cols. 20 - 23) indicating the number of records per batch if the UNIT Field contains an F.

NOTE: For a file which contains one record per block, an F would appear in the UNIT Field and 0001 in the ADDRESS Field.

(16) RECORD Line

SIZE Field contains the record type (A, B, C, or D) in Col. 13. If the record type is D, a comma will be entered in Col. 14 and the "end record" symbol will be entered in Col. 15.

If the record type in the SIZE Field is B, a four character decimal number indicating the record size will appear in the ADDRESS Field (Cols. 20 - 23).

(17) ERROR Line

ADDRESS Field specifies the address of the user routine to which FCP transfers control on non-recoverable read or write errors. For input files FCP transfers control to this routine upon delivery of the logical record in which the error occurred. For output files the FCP transfers control to this routine upon attempting to write the physical block in which an error is contained. Any further handling of the error record on input or the error block on output must be provided by the user.

FCP FILE SEQUENCE (Example for Input File)

(This example is based on the preceding problem for the Input File.)

LOCATION	OPERATION	SIZE	UNIT	ADDRESS
1 2 3 4 5 6	7 8 9 10 11 12	13 14 15 16 17 18	19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
INFILE	DEFSEQ			F,N,S
USEIN				
BACKUP				
IDENT				DEMDEPFI
USEBEG				
USEEND				
ACTIVE				060
LABELS				S,S
BLKAR				RDIN
ALTAR				ALTIN
RECAR				MSTR
VARRHE				
OUTLHE				
OUTRHE				
BATCH			F	0008
RECORD		B		0100
ERROR				

FCP FILE SEQUENCE Example Input File

FCP FILE SEQUENCE (Example for Output File)

(This example is based on the preceding problem and is for the Output File.)

LOCATION	OPERATION	SIZE	UNIT	ADDRESS
1 2 3 4 5 6	7 8 9 10 11 12	13 14 15 16 17 18	19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
OUTFIL	DEFSEQ			F,N,S
USEIN				
BACKUP				
IDENT				DEMDEPFI
USEBEG				
USEEND				
ACTIVE				060
LABELS				S,S
BLKAR				WROUT
ALTAR				ALOUT
RECAR				
VARRHE				
OUTLHE				ADMSTR
OUTRHE				
BATCH			F	0020
RECORD		B		0100
ERROR				

FCP FILE SEQUENCE Example Output File

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 2

ALLOCATING FCP AREAS AND PREPARING THE FCP FILE SEQUENCE

SITUATION

You are programming a file maintenance job in which there is some internal checking to be done on certain fields of a record. It is necessary that you have an input and an output record area.

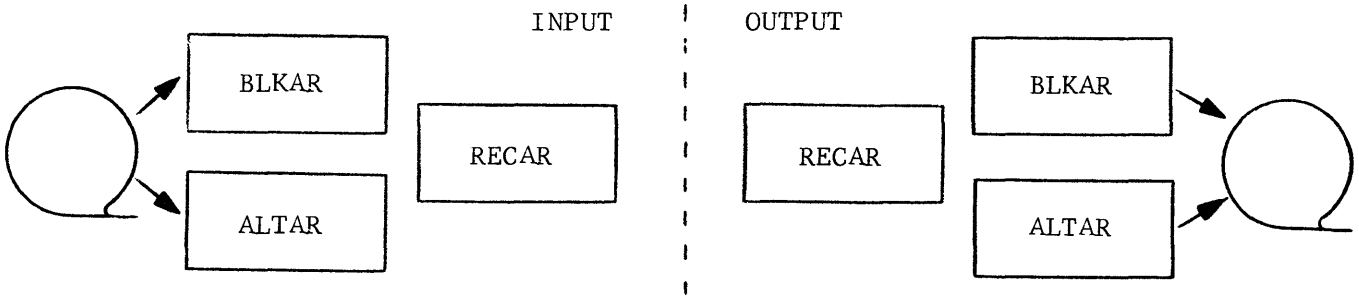
The record fields that you need to access in the record, both input and output, are as follows:

Stock Number	8
Charge Code	3
Usage Code	2
Reorder Level	7
Balance on Hand	7
(Unused fields)	23
	—

Total Record Size 50 characters

REQUIREMENT NO. 1

Write one sequence in which you allocate record, block, and alternate areas for both the input and output files to provide for maximum simultaneity; i.e., as follows:



Prepare your sequence with the following conditions:

a. Each record is fixed in length (50 characters).

b. Records will be batched on both input and output tapes and there are five records in a batch.

c. All areas are decade oriented. (The first character is in a location with an address ending in zero.)

d. Assign location tags with an "M" prefixing all input record items and with an "N" prefixing all output record items.

REQUIREMENT NO. 2

Prepare an FCP file sequence for both the input and the output file. Assign symbolic tags to each sequence and assign a label identification for the files. Assume standard beginning and ending labels. Assume an Active Time of 60 days.

SECTION VI

FILE CONTROLLING CODES

FILE CONTROL PROCESSOR FUNCTIONS

The File Control Processor, which is a part of the Operating System, has the function of controlling input/output operations. Through the use of Macros OPEN, CLOSE, READ, WRITE, and RELS, the FCP provides the following functions:

Batching

The FCP will operate on batched input and output files. For input files, the FCP will obtain sequential records from the batch and automatically read in a new batch when it is depleted. For output files, the FCP will include records in a batch and automatically write the batch when it is full.

Simultaneity

The FCP will schedule the use of the various simultaneous modes available in the 3301. The allocation of alternate input/output areas will allow the FCP to automatically read or write concurrently with other operations.

Queuing

In instances when all the simultaneous modes are busy, the FCP will maintain a backlog of I/O requests. The backlog list is serviced whenever an I/O instruction terminates.

Error Recovery

An abnormal I/O termination will cause a hardware interrupt and a transfer to the FCP recovery routine. The FCP provides various automatic error recovery procedures, such as re-reading the block that caused the abnormal termination.

Multiple Devices

The FCP will perform automatic tape swapping for files that require more than one reel of tape. This allows processing of one reel while the preceding reel is rewinding.

Multi-File Reels

The FCP allows the processing of reels that contain more than one file. Only one file may be OPEN on a reel at a given time.

Real Time Processing

Upon a Real Time interrupt, the FCP will collect the input message and transfer to the user's routine. Through the use of the FCP Macros, the user may process real time data as though it were just a logical record from an input device.

Device Interchangeability

Device interchangeability allows the user to switch the device type used to read or write a file without changing the program. For example, a file normally directed to the printer can be switched at object time to a magnetic tape. Controlling characters are automatically attached to each record so that a service routine can print the data from the tape in exactly the same format as intended.

FILE CONTROLLING CODES

OPEN

The OPEN Macro instructs the FCP to perform the housekeeping necessary to prepare an input or output file for processing. For magnetic tape, OPEN will process the label and position the tape for either reading (input files) or writing (output files) to the first data block.

CLOSE

The CLOSE Macro instructs the FCP to perform the housekeeping necessary to terminate the processing of an input or output file. For magnetic tape, CLOSE will write the last partially completed batch, process the label, and at the user's option, rewind the tape. End label procedures are not performed for input files when a CLOSE command is issued.

READ

The READ Macro instructs the FCP to obtain the next logical record from an input file. For "in-place processing", the READ Macro will cause the FCP to place the LHE address of the record (of a batched file or where alternate areas are used) in the address modifier (2 or 3) as indicated in the RECAR entry of the DATA sequence. For "record area processing," the READ Macro will cause the FCP to move the record from the input area to the record area. Control is transferred to a user specified address after the last record is processed.

WRITE

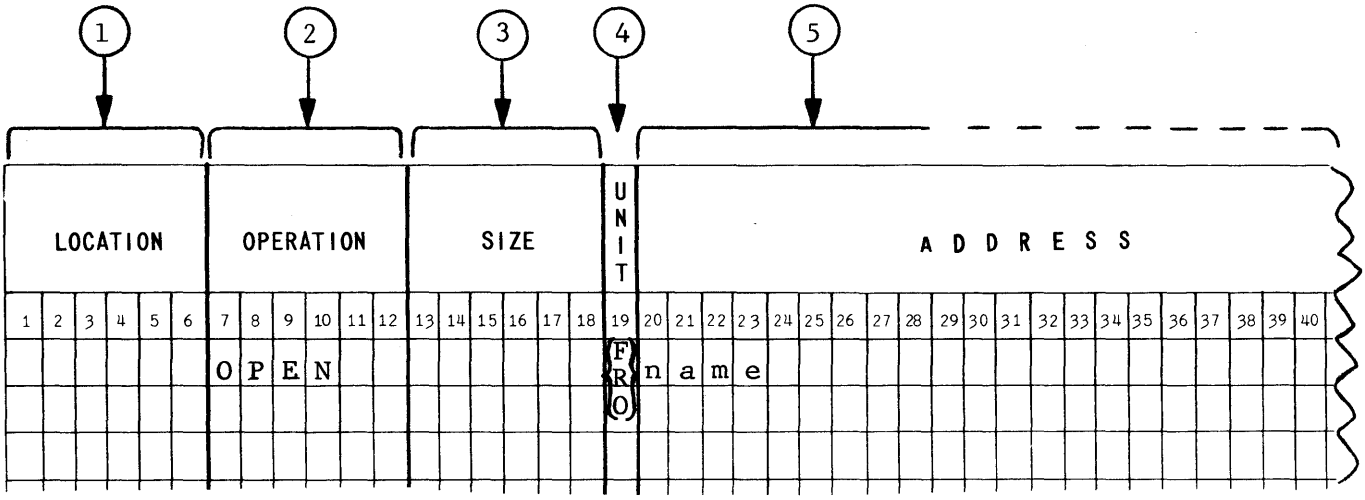
The WRITE Macro instructs the FCP to include a logical record in an output batch. For "in-place processing," WRITE will cause the FCP to place in the address modifier (2 or 3) the LHE address of the next record to be included in the batch. For record area processing, WRITE will cause the FCP to move the record into the output batch, and if the batch is completed, to physically write the batch.

RELS

The RELS Macro instructs the FCP to write the current output batch to an output file, or to bypass the remaining records in the current input batch. A READ issued after RELS will access the first record of the next batch. The remaining records in the released batch are no longer available to the programmer. For output files, RELS will cause the current output batch to be written unless a previous WRITE had filled the batch in which case RELS will be ignored.

OPEN

Initiates the processing of an input or output file.



FORMAT NOTES

(1) LOCATION

Symbolic name or left blank.

(2) OPERATION

OPEN

(3) SIZE

Blank

(4) UNIT

F = input processing in Forward direction.
R = input processing in Reverse direction.
O = output processing.

(5) ADDRESS

Name of the file sequence; the same name which appeared in the LOCATION Field of the DEFSEQ entry for this file.

COMMENTS

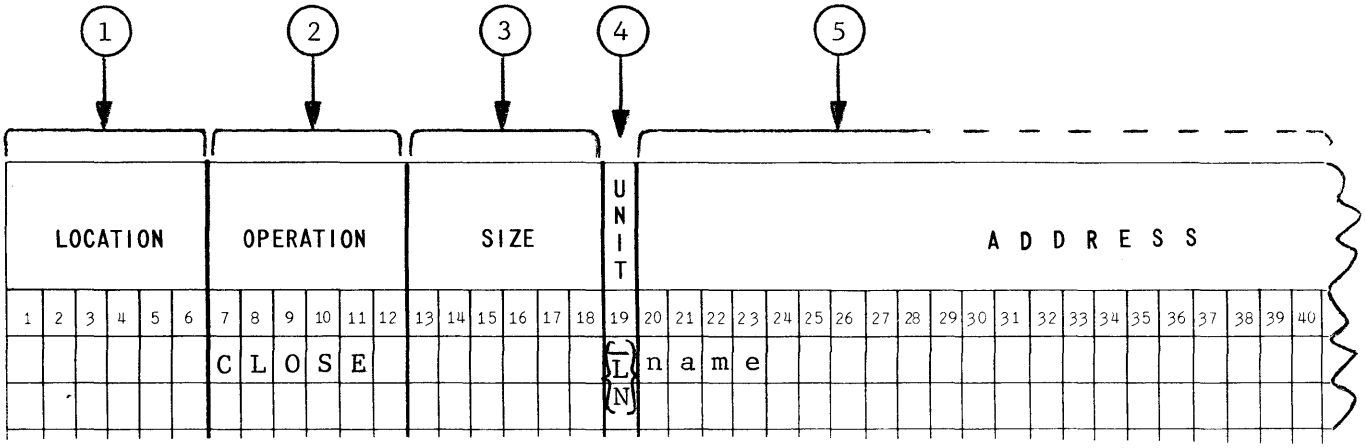
1. OPEN must be issued before any other File controlling codes are issued.

2. OPEN will cause the FCP to perform label processing and position the tape at the beginning of the file for forward processing or at the end of the file for reverse processing.

3. A file must be terminated by CLOSE prior to issuing another OPEN.

CLOSE

Terminates the processing of an input or output file.



FORMAT NOTES

(1) LOCATION

Symbolic name or left blank.

(2) OPERATION

CLOSE

(3) SIZE

Blank

(4) UNIT

_ (space) = Rewind to BTL.
L = Rewind to Load Point.
N = Do not Rewind.

(5) ADDRESS

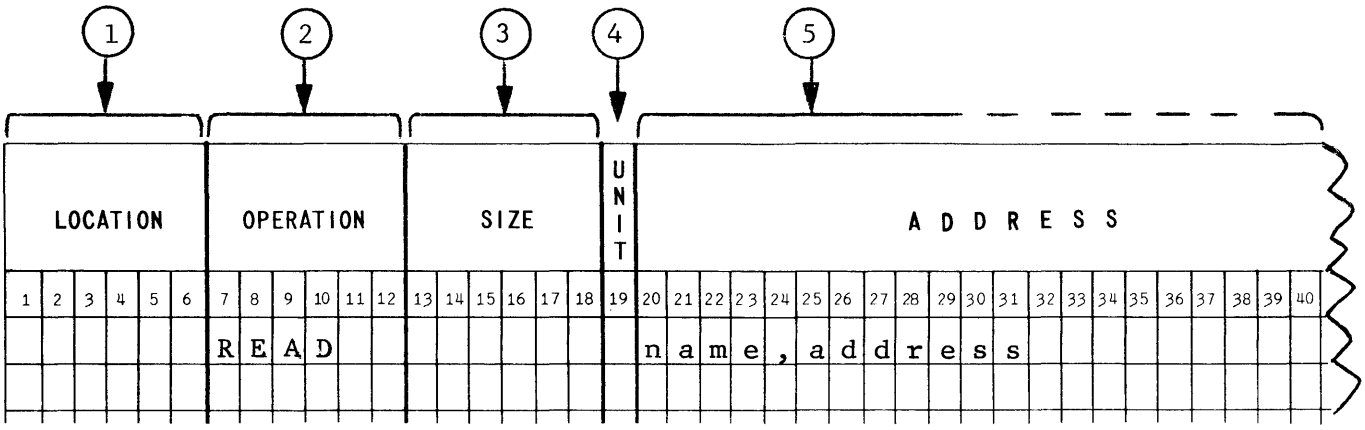
Name of the file sequence; the same name which appears in the LOCATION Field of the DEFSEQ entry for this file.

COMMENTS

1. A file must have been initiated with an OPEN before it can be terminated with a CLOSE.
2. A CLOSE issued to an input file will not cause end label processing.
3. A CLOSE issued to an output file will cause:
   a. Partially completed batches to be written.
   b. End label procedures.

READ

1. Obtains the next logical input record.
2. Transfers to end-file-jump-address after the last record is processed.



FORMAT NOTES

(1) LOCATION

Symbolic name or left blank.

(2) OPERATION

READ

(3) SIZE

Blank

(4) UNIT

Blank

(5) ADDRESS

name = Name of the file sequence. The same name which appeared in the LOCATION Field of the DEFSEQ entry for this file.

address = Address of the first instruction to be executed in the user's end-of-file routine.

COMMENTS

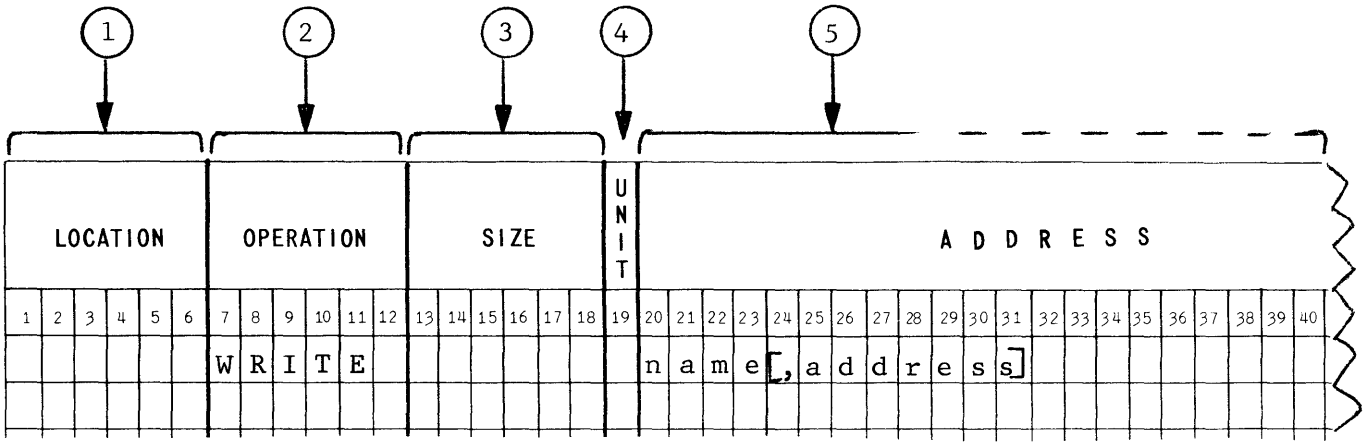
1. A file must have been OPENed before issuing a READ.
2. FCP, upon recognition of the end of an intermediate reel, will automatically perform:

   a. End label processing for the old reel.
   b. Tape swapping.
   c. Beginning label processing for the new reel.

3. End label processing will be performed prior to transferring to the user's end-of-file routine. CLOSE and OPEN must be issued before issuing any subsequent READ commands for this file.

WRITE

Instructs the FCP to include a logical record in the output batch.



FORMAT NOTES

(1) LOCATION

Symbolic name or left blank.

(2) OPERATION

WRITE

(3) SIZE

Blank

(4) UNIT

Blank

(5) ADDRESS

The first entry (name) is the name of the file descriptor sequence. This is the same name which appears in the LOCATION Field of the DEFSEQ entry for this file.

The second entry (address) is the address of a decade containing control information if this is a card or a print file.

The format of this decade is as follows:

For a printer control decade:

XXXXXXABCD

where:

X = unused characters

and:

A = 1, 2, or 4

where:

1 = paper advance via the C printer control character.
2 = advance paper via the tape loop.
4 = page change via the tape loop.

B = 0, 1, or 2

where:

0 = Asynchronous Mode printing.
1 = no printing.
2 = Synchronous Mode printing.

C = number of lines to advance if A = 1

where the maximum number of lines is 15. The numbers 10 to 15 must be specified in one character and are represented respectively by the following characters:

sp, #, @, (,), e.

D = 0, 1, 2, or 4

where:

0 = no high speed memory to buffer transfer.
1 = print 120 characters
2 = print 160 characters.
4 = transfer 64 contiguous Print Table characters to to the Print Table portion of the buffer.

For a card punch control decade:

XXXXXFXXXX

where:

X = unused characters

and:

F = Card Punch Mode. Insert zero for the translate mode or one for the binary mode.

COMMENTS

1. A file must have been OPENed before issuing a WRITE.

2. FCP, upon recognition of the end of an intermediate reel, will automatically perform:

   a. End label processing for the old reel.

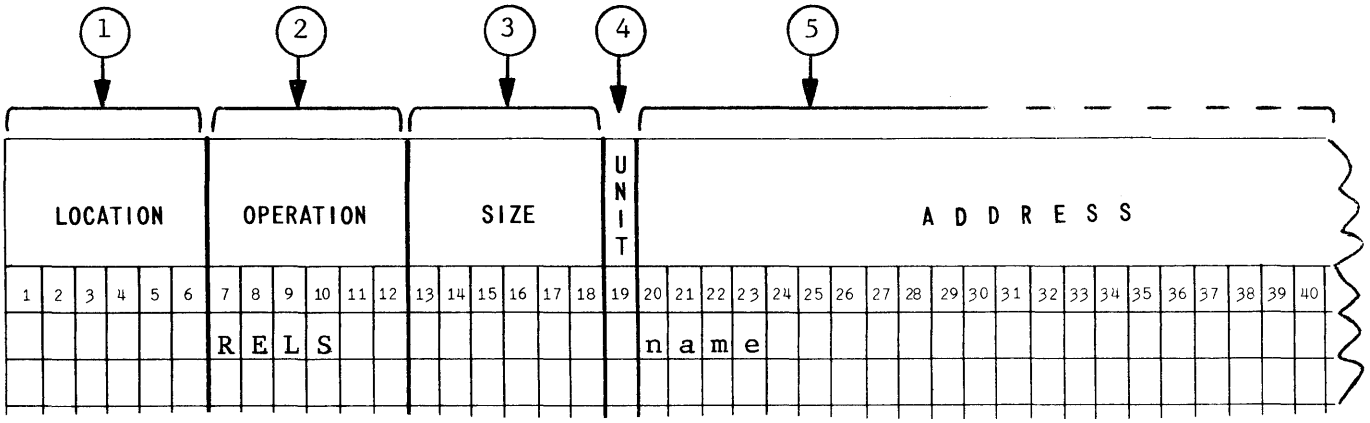
   b. Tape swapping.

   c. Beginning label processing for the new reel.

WRITE

RELS

1. Release currently active output batch to an output file.
2. Bypass remaining records in current input batch.



FORMAT NOTES

(1) LOCATION

Symbolic name or left blank.

(2) OPERATION

RELS

(3) SIZE

Blank

(4) UNIT

Blank

(5) ADDRESS

Name of the file sequence. The same name which appeared in the LOCATION Field of the DEFSEQ entry for this file.

COMMENTS

1. RELS issued to an input file causes the next READ command to access the first logical record of the next batch.
2. RELS issued to an output file causes the current batch to be written unless, due to a previous write, the batch contains no logical records.

RELS

EXAMPLE

The statements below are based on the "First National Bank" example in Section V.

LOCATION	OPERATION	SIZE	UNIT	ADDRESS
DUP	DEFSEQ			
	OPEN		F	INFILE
	OPEN		O	OUTFIL
BEGIN	READ			INFILE,END
	WRITE			OUTFIL
	UTC			BEGIN
END	CLOSE			INFILE
	CLOSE			OUTFIL
	EXIT	01		

The above sequence, along with the core sequence "INOUT" (page III-73) and the data sequences "INFILE" and "OUTFIL" (pages III-80 and 81), will duplicate the input file changing the batch size from 8 to 20 records per block.

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 6

A master tape contains fixed length records in batches of thirty. The format of each record is:

NAME	20	CHARACTERS
EMPLOYEE NUMBER	8	CHARACTERS
HOURLY PAY	4	CHARACTERS
TOTAL PAY-TO-DATE	6	CHARACTERS
TOTAL TAX PAID TO DATE	5	CHARACTERS

REQUIREMENTS

1. Reproduce the master tape, but write out batches of fifty records.
2. Prepare a separate tape in the output format as shown below:

NAME	20	CHARACTERS
(SPACES)	5	CHARACTERS
EMPLOYEE NO.	8	CHARACTERS
(SPACES)	5	CHARACTERS
TOTAL PAY TO DATE	6	CHARACTERS
(SPACES)	6	CHARACTERS

SECTION VII

INSTRUCTIONS FOR DATA TRANSFER OF FIXED LENGTH FIELDS

INSTRUCTIONS INCLUDED IN THIS SECTION

Transfer by Count Left
Transfer by Count Right
Transfer Decade by Count

TRANSFER BY COUNT Instructions

There are two instructions that may be used to transfer a fixed field of a given number of characters. The two instructions have the same format in the SIZE and ADDRESS Fields as follows:

	TRANSFER BY COUNT LEFT	TRANSFER BY COUNT RIGHT
OPERATION	TCL	TCR
SIZE	Number of characters to be transferred (0-45)	
A ADDRESS	Tag of the Sending Area	
B ADDRESS	Tag of the Receiving Area	

The instructions differ, however, in the way in which each is executed. The significance of the words Left and Right is that they specify the direction in which the actual transfer of characters take place.

Assembly of the Transfer by Count Left instruction generates left-end addresses for pure symbolic names in the A and B ADDRESS Fields. Assembly of the Transfer by Count Right instruction generates right-end addresses. When the instruction is executed, one character at a time is transferred as illustrated in the following example:

EXAMPLE

	Transfer by Count Left			Transfer by Count Right		
Assembly Instruction	TCL	4	DATA,WORK	TCR	4	DATA,WORK
Generated Instruction	M	4	0400 0600	N	4	0403 0603

	Transfer by Count Left				Transfer by Count Right			
TAG →	DATA				DATA			
Contents before and after execution	0400	0401	0402	0403	0400	0401	0402	0403
	0	1	2	5	0	1	2	5

	Transfer by Count Left				Transfer by Count Right			
TAG →	WORK				WORK			
Char. Transferred	0600	0601	0602	0603	0600	0601	0602	0603
First	0							5
Second	0	1					2	5
Third	0	1	2			1	2	5
Fourth	0	1	2	5	0	1	2	5

FINAL REGISTER CONTENTS

Transfer by Count Left	Transfer by Count Right
A_f = 0404	A_f = 0399
B_f = 0604	B_f = 0599

As the above example illustrates, both of the instructions produce the same result when the transfer is to another separate area in memory. However, if it is desired to transfer data in place, the proper instruction (Right or Left) must be used.

As an example, assume that it is desired to shift the following field

DATA			
6000	6001	6002	6003
0	1	2	5

one position to the right so that it will appear as follows:

6000	6001	6002	6003	6004
0	0	1	2	5

It should be obvious that the following instruction could not be used

```
TCL  4  DATA,DATA+$1
( M  4  6000 6001    )
```

because as the first character (0) is transferred, it replaces the second character to be transferred (1) and this process continues so that the resulting field after execution of the instruction would be:

6000	6001	6002	6003	6004
0	0	0	0	0

To obtain the result desired with a Transfer by Count instruction it must be written as follows:

```
TCR  4  DATA,DATA+$4

( N  4  6003 6004   )
```

It should be noted that in transferring data, the contents of the sending area remain undisturbed and that whatever existed in the receiving field prior to any data transfer instruction is replaced by the contents of the sending field.

TRANSFER DECADE BY COUNT Instruction

This instruction may be used to transfer an integral number of decades from one to another area of memory. It transfers ten characters at a time and is therefore ten times as fast as instructions which transfer by individual characters.

The instruction works from left to right in operation, thus, the left-end address is used in the A and B Addresses. The format of the instruction is as follows:

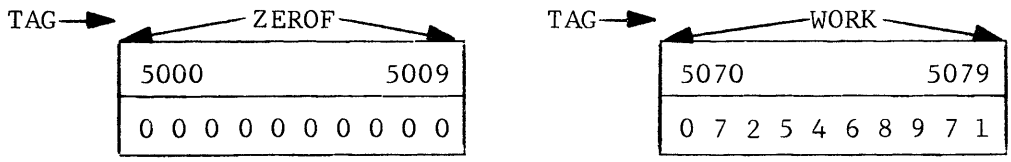
INSTRUCTION — TRANSFER DECADE BY COUNT

OPERATION	TDC
SIZE	Number of Decades to be transferred (0-45)
A ADDRESS	Tag of Sending Area
B ADDRESS	Tag of Receiving Area

The instruction should be used when transferring decade oriented fields such as records to another area of memory. It may also be used for filling decade oriented areas to a special character such as to a zero for numeric work areas and clearing printing areas to a space. Thus, the programmer might set up one decade filled with spaces (or zeros) and using the Repeat instruction fill 16 decades with spaces (or zeros). (See REPEAT Instruction -- Section XIII.)

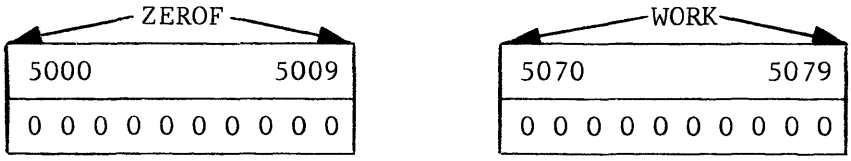
EXAMPLE 1 (Filling a Decade with zeros)

HSM Before Execution




```
Instruction              TDC  1  ZEROF,WORK
            Subscript    10   1  5000  5070
```

HSM After Execution



FINAL REGISTER CONTENTS

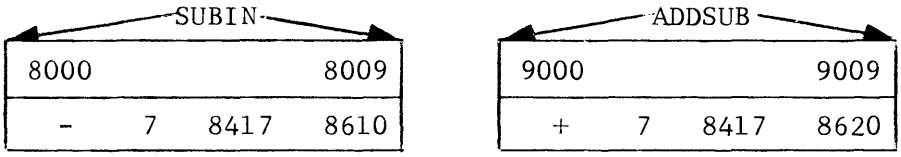
A_f = 5010

B_f = 5080

EXAMPLE 2

The Transfer Decade by Count instruction may also be used to transfer assembled instruction(s).

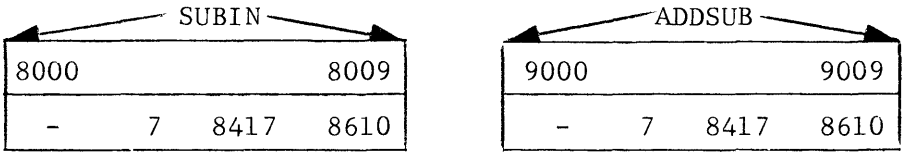
HSM Before Execution



Instruction

```
                TDC  1   SUBIN,ADDSUB
Subscript 10    1     8000 9000
```

HSM After Execution



FINAL REGISTER CONTENTS

A_f = 8010

B_f = 9010

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 3

SITUATION

A Master Record consists of the following items:

ACCT NO.	7
NAME	23
ST. ADDR.	17
CITY ST. ADDR.	15
TOTAL DEPOSITS	9
TOTAL CHECKS	9
BALANCE	10

An output record is to be constructed in the following format:

ACCT NO.	7
NAME	23
BAL	10

REQUIREMENT NO. 1

Write a sequence to allocate a record area for each of the above records (decade oriented).

REQUIREMENT NO. 2

Write a sequence to construct the output record.

SECTION VIII

COMPARE AND TRANSFER CONTROL INSTRUCTIONS

INSTRUCTIONS COVERED IN THIS SECTION

Compare Data
Conditional Transfer of Control
Unconditional Transfer of Control
EXIT Controlling Code

COMPARE DATA Instruction

This instruction is used to compare two fields of equal length. The two fields are not affected by this instruction but an indicator is set based on the relative value of the fields. This indicator, called the Previous Result Indicator, is set as follows:

PRP (Previous Result Positive)

The A addressed field is greater in value than the B addressed field.

PRZ (Previous Result Zero)

The A and B addressed fields are equal.

PRN (Previous Result Negative)

The A addressed field is lesser in value than the B addressed field.

Comparison is based on the relative binary values of each field. The instruction operates from left to right comparing one character at a time from each field. The instruction terminates when all characters (as determined by the SIZE Field) have been compared if both fields are equal in value. If the fields are unequal in value, the instruction terminates after the unequal character in each field has been compared. The final contents of the A and B Registers are one to the right of the last character compared.

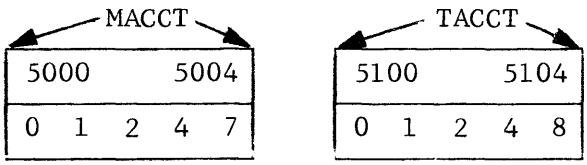
The format of the instruction is:

COMPARE DATA

OPERATION	CDT
SIZE	No. of characters in each field to be compared (0-45).
A ADDRESS	Tag of one field to be compared.
B ADDRESS	Tag of second field to be compared.

EXAMPLE 1

HSM Before and After Execution



Instruction

```
 CDT   5   MACCT,TACCT
( Y    5   5000  5100 )
```

FINAL REGISTER CONTENTS

A_f = 5005

B_f = 5105

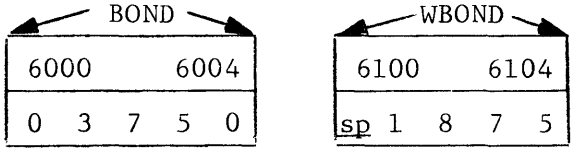
PRI'S

PRN is set.

Both fields should be of equal length and properly initialized. Alphabetic values, if compared, should be left-justified in the comparison fields and space filled in rightmost positions. If numeric fields are to be compared for their relative values, the significant characters should be right-justified in the comparison field and zero filled to the left. Otherwise, the PRI setting may not reflect the relative value of the numeric fields as in the following example:

EXAMPLE 2

HSM Before and After Execution



Instruction

```
 CDT   5   BOND,WBOND
  Y    5   6000 6100
```

FINAL REGISTER CONTENTS

A_f = 6001

B_f = 6101

PRI'S

PRN is set.*

*A space has a higher binary value than a zero.

CONDITIONAL TRANSFER OF CONTROL Instruction

This instruction tests an indicator and, based on the setting of the indicator, transfers control to a given location. The indicators it will sense are the PRI's, the Overflow Indicator and the Alteration Switches.

PRI Sensing

There are several instructions that set the PRI's; the CTC instruction may be used following each of these instructions to gain a transfer of control based on the setting.

The instruction may be written using the CTC Operation Code and a SIZE entry of 1 or with an Extended Operation Code of TPM as in the following format:

COND. TRANS. OF CONTROL

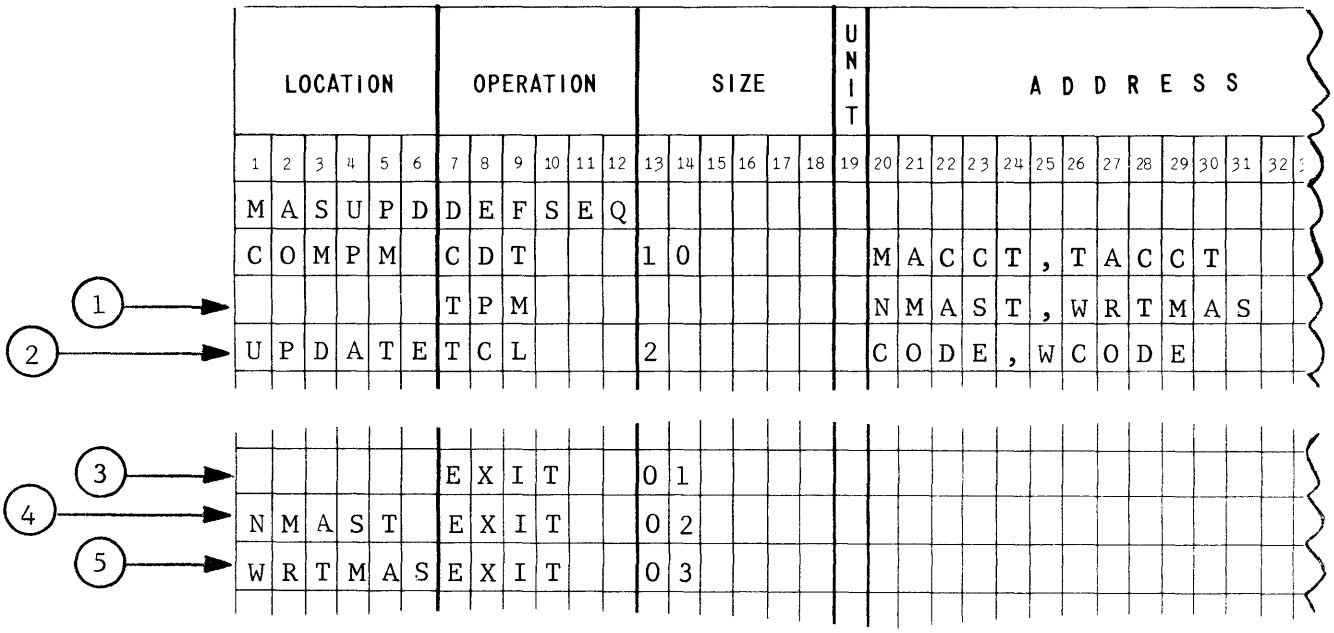
		EXTENDED OP. CODE
OPERATION	CTC	TPM
SIZE	1	(BLANK)
A ADDRESS	Location to transfer to if PRP is set.	
B ADDRESS	Location to transfer to if PRN is set.	

If PRZ is set, the next instruction in sequence will be executed.

EXAMPLE (Compare and CTC Instruction)

In this example assume that a master record and a transaction record are present in memory and the 10 character account numbers of each are to be compared. The tags MACCT and TACCT have been assigned to the account number fields of each, respectively. If MACCT and TACCT are equal, the master acct. will be updated. If MACCT is greater, transfer will be to a preparation of a new master sequence. If MACCT is less than TACCT, transfer will be to a write master sequence.

The following is an example:



NOTES

(1) Conditional Transfer of Control Instruction. (TPM Extended Operation Code) Transfer to NMAST if PRP is set (MACCT greater than TACCT). Transfer to WRTMAS if PRN is set (MACCT less than TACCT).

(2) First instruction of UPDATE routine.

(3) Last instruction of UPDATE routine EXIT from Sequence.*

(4) EXIT from Sequence for preparation of a new master record.*

(5) EXIT from Sequence for writing of the master record.*

*See EXIT controlling code at end of this section.

PRZ Sensing

The CTC instruction may also be used to generate a transfer of control if the PRI's are set to PRZ (Previous Result Zero). This instruction may be used following any of the instructions that set the PRI's. It tests, however, only the setting of PRZ and transfers control to a specified location if PRZ is set. If PRZ is not set, the next sequential instruction is executed. The format of this instruction with the option of using an Extended Operation Code is as follows:

TRANSFER ON ZERO

	EXTENDED OP CODE	
OPERATION	CTC	TRZ
SIZE	+	(BLANK)
A ADDRESS	Location to transfer to if PRZ is set.	
B ADDRESS	($0) Zero (Ignored) Address*	

*If PRZ is not set, the next instruction in sequence will be executed.

ALTERATION SWITCH Sensing

On the console, there are four Alteration Switches. Each of these may be set to an "On" or "Off" position. A CTC instruction may be used to test the status of a given switch and transfer control to one location if the switch is set (ON) or to another location if the switch is not set (OFF). The format of the instruction using the Extended Operation Codes are as follows:

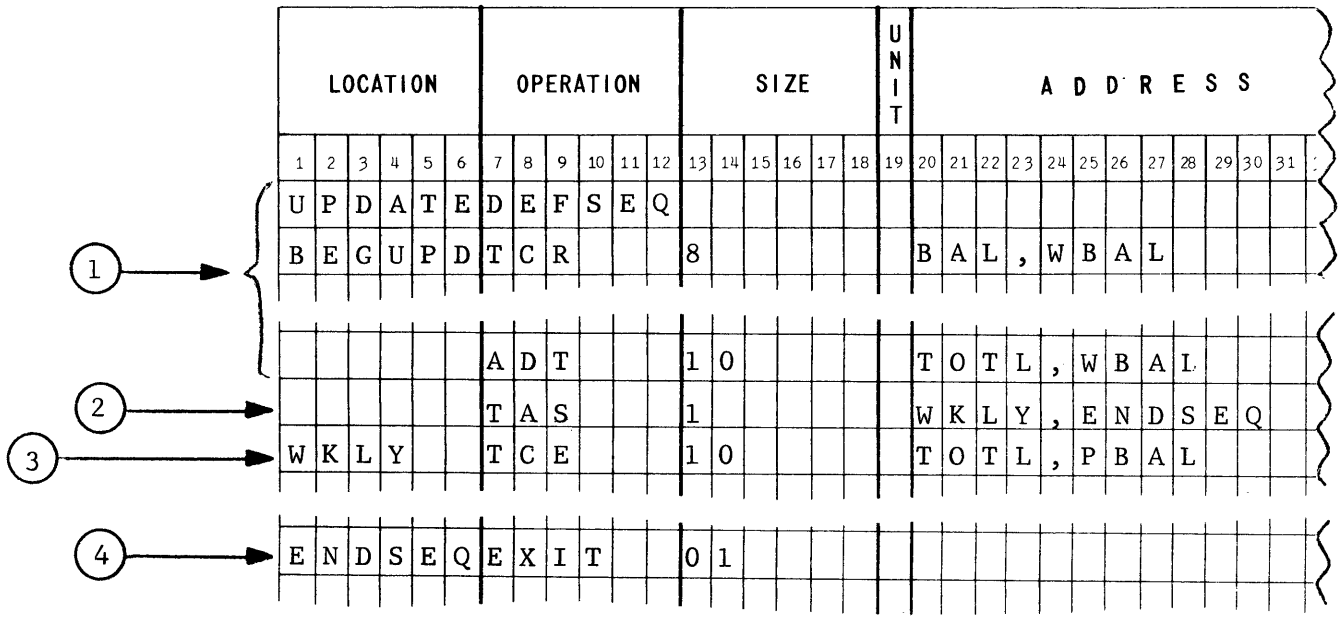
TEST ALTERATION SWITCH

	No. 1	No. 2	No. 3	No. 4
OPERATION	TAS	TAS	TAS	TAS
SIZE	1	2	3	4
A ADDRESS	Location to transfer to if the Designated Alteration Switch is set.			
B ADDRESS	Location to transfer to if the Designated Alteration Switch is not set.			

As the name implies, the Alteration Switches allow a program to be altered in its execution by a manual switch on the console.

As an example, assume that a program is written and is to be run (executed) on a daily cycle. Each day's totals are accumulated but only on the last day of the week are the accumulated totals to be written on a prepared report. A sequence may be written to include both the daily and weekly activity. When the weekly report is to be written, the operator sets Alteration Switch No. 1. An example of the sequence is shown on the following page.

EXAMPLE - Use of Alteration Switch (No. 1)



NOTES

(1) Daily Accumulation Routine

(2) Test Alteration Switch Instruction

Weekly (WKLY) routine is bypassed unless Alteration Switch No. 1 is set (ON).

(3) Weekly Preparation of Report routine

(4) Exit from Sequence

OVERFLOW Sensing

Whenever an overflow (carry) takes place in an Arithmetic instruction, a flip-flop, the Overflow Indicator, is set.* This indicator may be tested with a CTC instruction in the following format:

TEST OVERFLOW INDICATOR

	EXTENDED OP. CODE	
OPERATION	CTC	TOF
SIZE	2	(BLANK)
A ADDRESS	Tag of the instruction to which control is to be transferred if the Overflow Indicator is set.	
B ADDRESS	Tag of the instruction to which control is to be transferred if the Overflow Indicator is not set.	

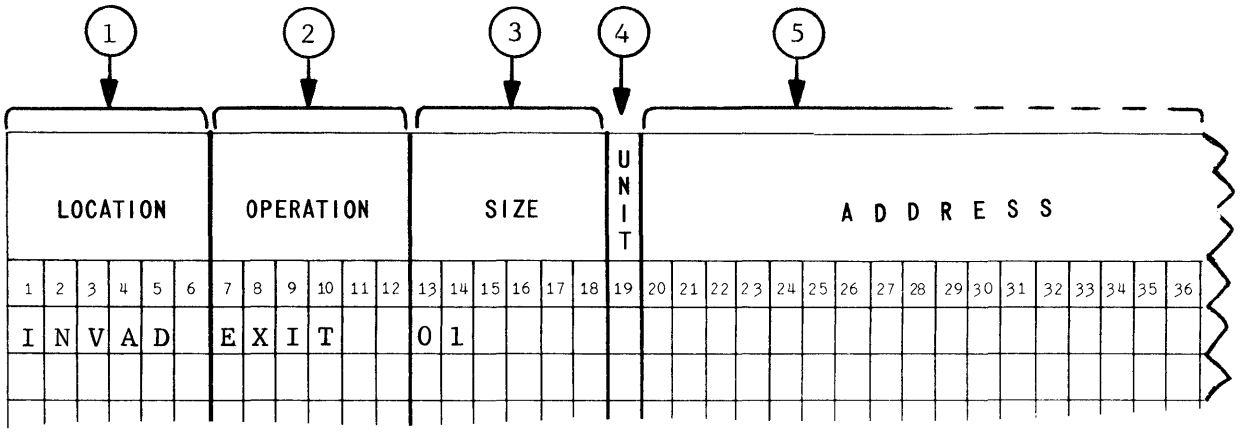
*It should be noted that when overflow occurs in data Arithmetic instructions, an interrupt occurs. This instruction might be used as a part of the interrupt routine. This instruction might also be used to test overflow in an address Arithmetic instruction (see Section XII of this manual).

EXIT Controlling Code

As shown in previous examples, all transfers out of the sequence are by way of an EXIT controlling code. As many as 99 EXIT lines may be present in a sequence. Each line is numbered in the SIZE Field consecutively beginning with 01.

Linkage to the desired location in another sequence is obtained by the Segment Description (see Section XV).

The format of the EXIT controlling code is:



FORMAT NOTES

(1) LOCATION

A symbolic name may be used in this field to reference the EXIT.

(2) OPERATION

EXIT must appear.

(3) SIZE

The EXIT sequence number must appear in this field for each EXIT line in the sequence. It is a two digital decimal number from 01 to 99.

(4) (5) The UNIT and ADDRESS Fields are not used for the EXIT line.

UNCONDITIONAL TRANSFER OF CONTROL Instruction

An Unconditional transfer of control instruction may be written using only the OPERATION and ADDRESS Fields as follows:

OPERATION	UTC
SIZE	Not used
ADDRESS	Tag of the Next Instruction to be executed

The address written in the ADDRESS Field however must be a direct address. No indirect addressing or indexing may be performed on this address.

EXAMPLE

Assume that the symbolic tag BEGN has a value of 4020

Assembly Instruction	UTC		BEGN
Generated Instruction	W	• 0000	4020

RCA 3301 ASSEMBLY SYSTEM

PRACTICAL EXERCISE NO. 4

SITUATION

A Master Inventory File consists of 100 character records. The stock number is the first 10 characters of each record. The stock number (in another sequence) has been assigned the symbolic tag MSTKNO.

REQUIREMENT

Write a sequence in which you will test this stock number and go to:

EXIT 01 if the Stock No. is less than 0000010000

EXIT 02 if the Stock No. is higher than 0000199999

EXIT 03 if the Stock No. is from 0000010000 - 0000199999 inclusive.

The sequence will include all necessary constants as well as the coding required.

SECTION IX

DATA ARITHMETIC INSTRUCTIONS

INSTRUCTIONS COVERED IN THIS SECTION

Add Data
Subtract Data
Multiply
Divide

ADD AND SUBTRACT DATA INSTRUCTIONS

These instructions may be used to perform decimal addition or subtraction on fixed length numeric fields. The result after execution of each of these instructions is found in the field specified by the A Address. Each of the instructions operates from right to left. Thus, right-end addresses will be generated for pure symbolic names appearing in the A and B Address fields. Each of the equal length fields may contain a maximum of 45 characters and should be long enough to accept the result without a carry (overflow) out of the most significant character position.

The sign of each field is carried as a zone bit of the least significant digit. A 2^5 bit of one in the LSD signifies a negative field. A zero in the 2^5 bit position signifies the field is positive.

The PRI's are set for each of the instructions based on the Result field.

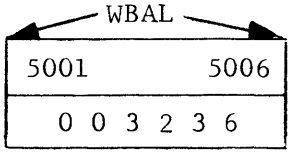
PRP is set if the result is positive.
PRN is set if the result is negative.
PRZ is set if the result is zero.

The format of the instructions are as follows:

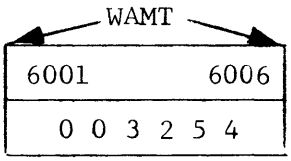
	ADD DATA	SUBTRACT DATA
OPERATION	ADT	SDT
SIZE	Number of Characters in each field (0-45)	
A ADDRESS	Tag of first field (This field will contain the result after execution of the instruction.)	
B ADDRESS	Tag of the second field	

EXAMPLES: (ADD DATA and SUBTRACT DATA Instructions)

HSM Before Execution



HSM Before and After Execution



	ADD DATA	SUBTRACT DATA
Instruction	ADT 6 WBAL,WAMT + 6 5006 6006	SDT 6 WBAL,WAMT - 6 5006 6006

HSM After Execution

WBAL	
5001	5006
0 0 6 4 9 0	

WBAL	
5001	5006
0 0 0 0 1 Q	

FINAL REGISTER CONTENTS

A_f = 5000	A_f = 5000
B_f = 6000	B_f = 6000
PRP is set.	PRN is set.

MULTIPLY INSTRUCTION

Multiplication is performed on two eight character numeric fields. The multiplicand is stored in the A addressed field and the multiplier is stored in the B addressed field before execution of the instruction. The product is stored in a sixteen character field which includes the 8 locations used to store the multiplier and an additional 8 character field to the right of the multiplier. Thus, after execution of the instruction, the multiplier is replaced by the eight leftmost characters of the product.

The eight digit field to the right of the multiplier (B addressed field) should be zero filled unless rounding or accumulation is desired; for whatever numeric value is present in this field will be added to the product.

The PRI's are set based on the result as in addition and subtraction.

The format of the MULTIPLY instruction is as follows:

MULTIPLY

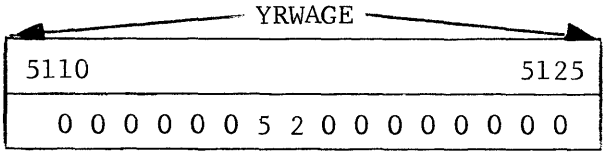
OPERATION	MPY
SIZE	(BLANK) No entry required
A ADDRESS	Tag of Multiplicand Field
B ADDRESS	Tag of Multiplier Field and field which will contain the 8 most significant digits of the product.

EXAMPLE 1 - MULTIPLY Instruction

HSM Before and After Execution

WAGE	
5003	5010
0 0 0 1 5 0 0 0	

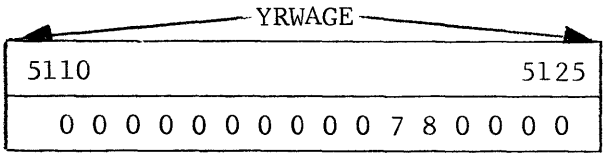
HSM Before Execution



Instruction

```
MPY     WAGE,YRWAGE+$7
 +   $  5010   5117
```

HSM After Execution



FINAL REGISTER CONTENTS

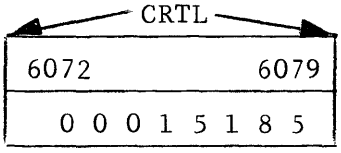
A_f = 5002

B_f = 5109

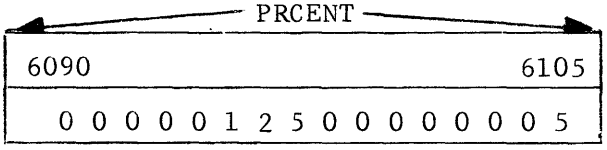
PRP is set.

EXAMPLE 2 - Multiply With Rounding

HSM Before and After Execution



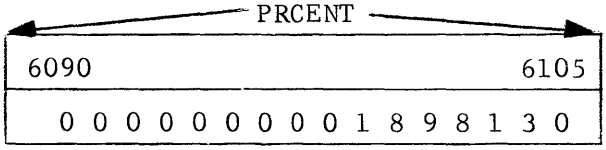
HSM Before Execution



Instruction

```
MPY     CRTL,PRCENT+$7
 +   $  6079   6097
```

HSM After Execution



FINAL REGISTER CONTENTS

A_f = 6071

B_f = 6089

PRP is set.

When PRN is set, both the eighth and sixteenth characters of the product will contain a negative sign (2^5 bit of 1) as illustrated in the following example.

EXAMPLE 3 (Multiplication of fields with unlike signs)

HSM Before and After Execution

TOTL

6013	6020
9 0 0 0 0 0 0 I	

HSM Before Execution

NUMBER

6021	6028	6036
0 0 0 0 0 0 0 2	0 0 0 0 0 0 0 0	

Instruction

```
MPY   TOTL,NUMBER
 +  $ 6020 6028
```

HSM After Execution

NUMBER

6021	6028	6036
0 0 0 0 0 0 0 J	8 0 0 0 0 0 1 K	

FINAL REGISTER CONTENTS

A_f = 6011

B_f = 6020

PRN is set.

DIVIDE INSTRUCTION

Division is performed on two eight-character numeric fields. The dividend is stored in the A addressed field and the dividend is replaced by the quotient after execution of the instruction. The remainder following execution will be stored in an eight character area immediately to the right of the quotient (A addressed field). This remainder field must be zero-filled prior to execution of the instruction.

The divisor is stored in the B addressed field and the B addressed field (divisor) must be greater in value than the A addressed field (dividend).

The format of the instruction is as follows:

DIVIDE

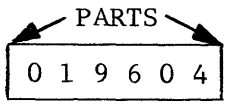
OPERATION	DVD
SIZE	(Blank) No entry required.
A ADDRESS	Tag of the dividend field and the field which will contain the quotient after execution of the instruction.
B ADDRESS	Tag of the divisor field.

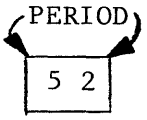
As mentioned above, the divisor must be greater in value than the dividend. A way of insuring this in most cases is to left-justify the divisor and right-justify the dividend in the B Address and A Address Fields, respectively.

As an example, assume that two fields have been defined and contain the values as indicated below:

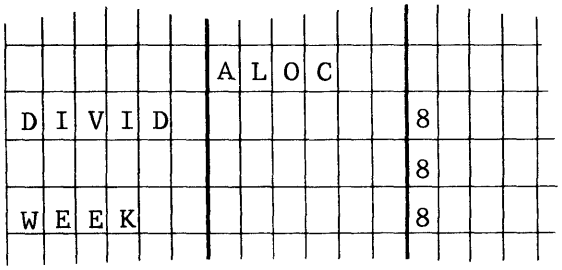
ALOC

PARTS	6
PERIOD	2





The user, to divide the first field by the second, has defined a dividend field and divisor field as follows:



	ASSUMED ALLOCATED AREA
Dividend (and Quotient)	9000 - 9007
Field	9008 - 9015
Divisor Field	9016 - 9023

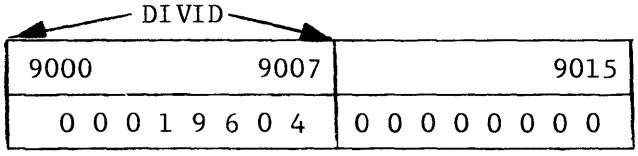
Before moving the fields to be divided, the Dividend and Divisor Areas would be zero-filled (Symbol Fill Sector instruction - Section X) and then use the following instructions prior to the DIVIDE instruction.

OPERATION	SIZE	ADDRESS	REMARKS
TCR	6	PARTS,DIVID	Right Justify in Dividend Field
TCL	2	PERIOD,WEEK	Left Justify in Divisor Field

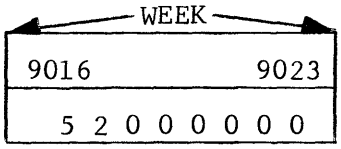
The fields of the Dividend (DIVID) and Divisor (WEEK) would thus appear as in Example 1 before execution of the instruction.

EXAMPLE 1

HSM Before Execution



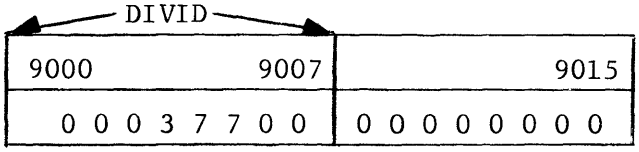
HSM Before and After Execution



Instruction

DVD	DIVID,WEEK	
+	9007	9023

HSM After Execution



FINAL REGISTER CONTENTS

A_f = 9007

B_f = 9014

PRP is set.

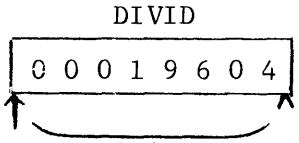
The location of the assumed decimal point in the quotient can be determined by use of a Scale Factor. The Scale Factor is the number of digits between the assumed (or problem) decimal point and the machine decimal point. The machine decimal point is assumed to be to the left of each 8 character field being divided.

The formula for determining the location of the assumed decimal point in the quotient is as follows:

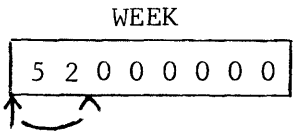
Quotient Scale Factor = Dividend Scale Factor - Divisor Scale Factor

Assuming in the preceding example that both fields were integers, then the machine and assumed decimal points and scale factors would be as follows:

Before Execution



DIVIDEND SCALE FACTOR 8

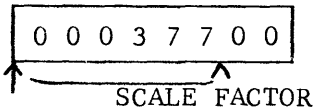


(MINUS)

DIVISOR SCALE FACTOR - 2

(EQUALS)

After Execution



QUOTIENT SCALE FACTOR 6

↑ Machine Decimal Point
^ Assumed Decimal Point

SECTION X

DATA EDITING INSTRUCTIONS

INSTRUCTIONS INCLUDED IN THIS SECTION

Symbol Fill Sector
Symbol Fill to Non-Zero Numeric
Float Dollar Sign to Non-Zero Numeric
Transfer by Count to Edit Field
Locate Absence of Symbol Left
Locate Absence of Symbol Right
Translate by Table

SYMBOL FILL SECTOR Instruction

This instruction may be used to place a specified character into all locations of a sector as defined by the A and B ADDRESS Fields. The character to be inserted in each location is written in Column 13 of the SIZE Field.

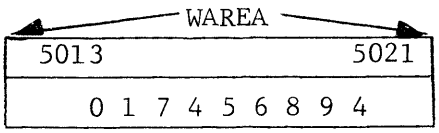
For pure symbolic names, the Assembler will assign a left-end address in the A ADDRESS Field and a right-end address for the B ADDRESS Field. All locations between and including the A and B addressed locations will be filled with the specified character.

The format of the SYMBOL FILL SECTOR instruction is as follows:

OPERATION	SFS
SIZE	Actual character to be placed in each location
A ADDRESS	Tag of the leftmost location of the sector to be filled
B ADDRESS	Tag of the rightmost location of the sector to be filled

EXAMPLE 1 (Filling a numeric work area with zeros)

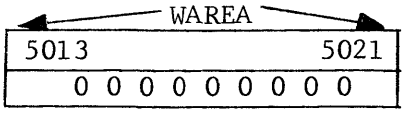
HSM Before Execution




```
Instruction        SFS   0   WAREA,WAREA
                    J    0   5013  5021
```

EXAMPLE 1 (continued)

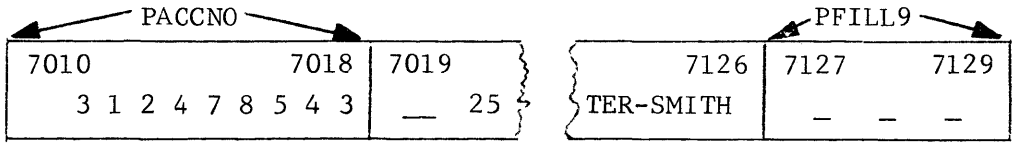
HSM After Execution



FINAL REGISTER CONTENTS $A_f = 5022 \quad B_f = 5021$

EXAMPLE 2 (Filling an area with spaces)

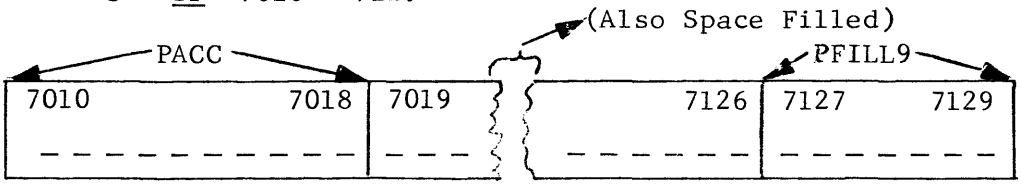
HSM Before Execution



Instruction

```
SFS      PACCNO,PFILL9
J   SP   7010   7129
```

HSM After Execution



FINAL REGISTER CONTENTS $A_f = 7130 \quad B_f = 7129$

SYMBOL FILL TO NON-ZERO NUMERIC Instruction

This instruction may be used for such functions as:

a. Suppression of high order zeros in a numeric field.
b. Insertion of a specified character (other than a dollar sign) in a field until a numeric character other than zero is found.
c. Determination of whether a non-zero numeric character is present in a field.

The instruction is executed from left to right. Each location, starting from the left, is examined and a specified character is inserted until either (1) a numeric character other than zero is found or (2) the right-end of the field is reached. The dollar sign is the only 3301 character that may not be specified for insertion. For pure symbolic tags, a left-end address will be assembled for the A ADDRESS and a right-end address will be assembled for the B ADDRESS. PRP is set if a numeric character other than zero is found. PRZ is set if a numeric character (1-9) is not found. STA is performed.

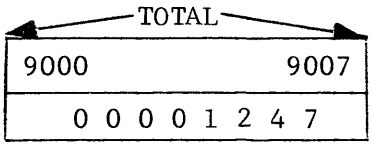
The format of the instruction is as follows:

SYMBOL FILL TO NON-ZERO NUMERIC

OPERATION	SFN
SIZE	Actual Symbol to be placed into each location. The Dollar Sign ($) may not be used.
A ADDRESS	Tag of the leftmost location to be examined.
B ADDRESS	Tag of the rightmost location to be examined.

EXAMPLE 1 (Zero Suppression)

HSM Before Execution

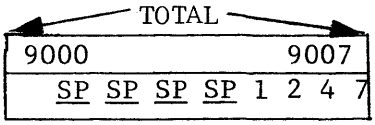


Instruction

SIZE Field left blank for a space character

```
SFN  (space)  TOTAL,TOTAL
,    (space)  9000  9007
```

HSM After Execution



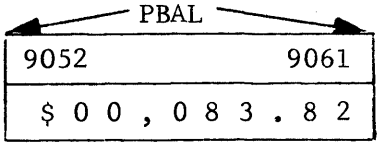
FINAL REGISTER CONTENTS

$A_f^* = 9004$ $B_f = 9007$

*STA is performed.
PRP is set.

EXAMPLE 2 ("Check Protect" function)

HSM Before Execution

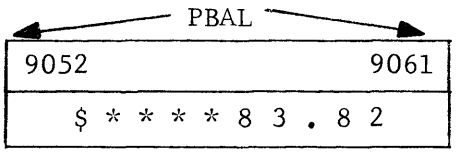


Instruction

```
SFN  *  PBAL+$1,PBAL
,    *  9053    9061
```

EXAMPLE 2 (continued)

HSM After Execution



FINAL REGISTER CONTENTS

$A_f^* = 9057 \quad B_f = 9061$

*STA is performed.
PRP is set.

FLOAT DOLLAR SIGN TO NON-ZERO NUMERIC Instruction

This instruction examines each location from the left in a specified field until a numeric character other than zero is found. When the non-zero numeric character is found, a dollar sign is placed in the location to the left of the non-zero numeric character. All other locations searched are space filled. If the first (leftmost) location of the field contains a numeric character (1-9) a dollar sign will be placed in the location to the left of the field (see Example 2). For pure symbolic tags a left-end address will be assembled for the A ADDRESS and right-end address will be assembled for the B ADDRESS.

STA is performed. PRP is set if a numeric character other than zero is found. PRZ is set if a numeric character (1-9) is not found.

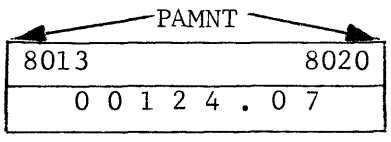
The format of the instruction is as follows:

FLOAT DOLLAR SIGN TO NON-ZERO NUMERIC

OPERATION	FDN
SIZE	(Blank) NO ENTRY REQUIRED
A ADDRESS	Tag of leftmost location to be examined.
B ADDRESS	Tag of rightmost location to be examined.

EXAMPLE 1

HSM Before Execution

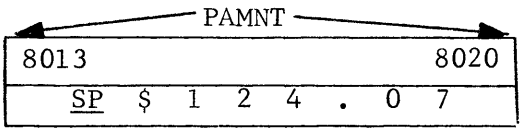


EXAMPLE 1 (continued)

Instruction

```
FDN     PAMNT,PAMNT
,  $ 8013  8020
```

HSM After Execution



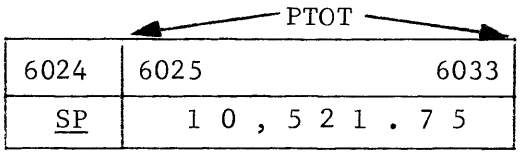
FINAL REGISTER CONTENTS

$A_f^* = 8014 \quad B_f = 8020$

*STA is performed.
PRP is set.

EXAMPLE 2

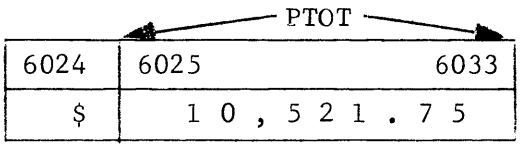
HSM Before Execution



Instruction

```
FDN     PTOT,PTOT
,  $ 6025 6033
```

HSM After Execution



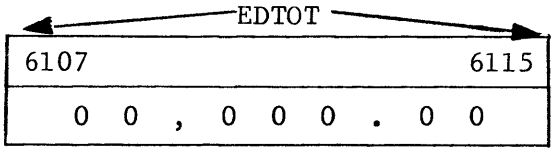
FINAL REGISTER CONTENTS

$A_f^* = 6024 \quad B_f = 6033$

*STA is performed.
PRP is set.

EXAMPLE 3

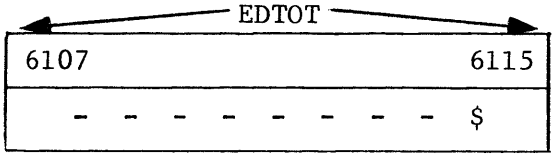
HSM Before Execution



Instruction

```
FDN     EDTOT,EDTOT
,  $ 6107  6115
```

HSM After Execution



FINAL REGISTER CONTENTS

$A_f^* = 6115 \quad B_f = 6115$

*STA is performed.
PRZ is set.

TRANSFER BY COUNT TO EDIT FIELD Instruction

This instruction transfers numeric data to an edit field. The edit field contains spaces in locations where the numeric characters are to be placed. An ISS (•) character in the edit field will be replaced by a space. Other characters appearing in the edit field will remain unchanged. The instruction is executed from right to left.

The first character transferred is tested for the presence of a negative sign (1 bit in 2^5 bit position); if present, PRN is set, otherwise, PRZ is set. Any 1 value zone bits present in this first and succeeding characters are removed as they are being transferred to the edit field. STA is performed.

This instruction is executed from right to left. Right-end addresses will be assembled for pure symbolic names appearing in the A and B ADDRESS Fields. The instruction terminates when the specified number of characters (SIZE entry) have been transferred to the edit field.

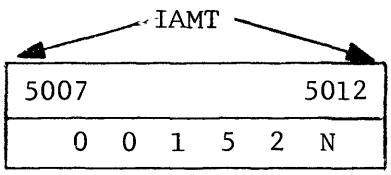
The format of the instruction is as follows:

TRANSFER BY COUNT TO EDIT FIELD

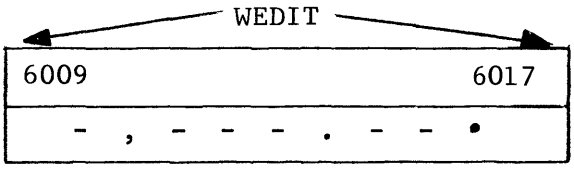
OPERATION	TCE
SIZE	Number of characters (0-45) to be transferred and edited.
A ADDRESS	Tag of the edit field (Receiving Area).
B ADDRESS	Tag of area containing the numeric data to be edited (Sending field).

EXAMPLE 1

HSM Before and After Execution



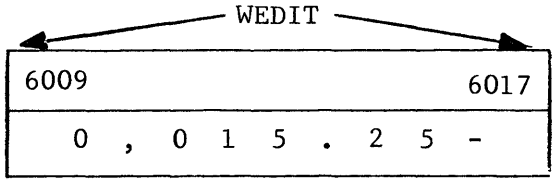
HSM Before Execution



Instruction

```
TCE   6   WEDIT,IAMT
÷     6   6017   5012
```

HSM After Execution



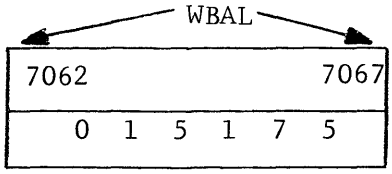
FINAL REGISTER CONTENTS

A_f^* = 6008 B_f = 5006

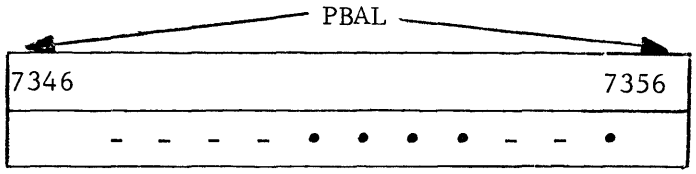
*STA is performed.
PRN is set.

EXAMPLE 2 (Example of splitting dollars and cents fields for use on a preprinted form.)

HSM Before and After Execution



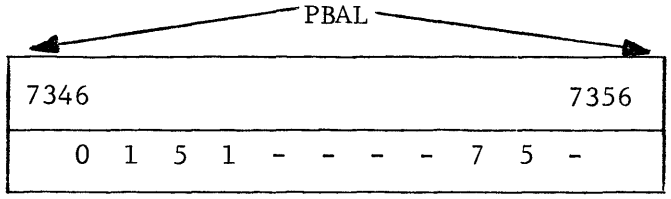
HSM Before Execution



Instruction TCE 6 PBAL,WBAL

÷ 6 7356 7067

HSM After Execution



FINAL REGISTER CONTENTS

A_f = 7345 B_f = 7061 *

*STA is performed
PRZ is set.

LOCATE ABSENCE OF SYMBOL Instructions

There are two instructions that may be used to search a field looking for the absence of a specified symbol. The Locate Absence of Symbol Left (LAL) instruction searches from left to right. The Locate Absence of Symbol Right (LAR) searches from right to left. Each instruction ends when a location is examined that either (1) contains other than the specified (SIZE entry) character or (2) is the last location to be examined.

The PRI's are set as follows:

PRN - The specified character is not present in the first location searched.

PRZ - The specified character is present in each location searched.

PRP - A non-specified character is found after the specified character has been found.

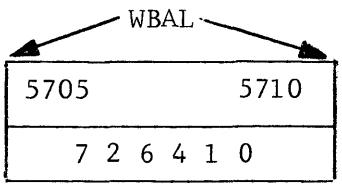
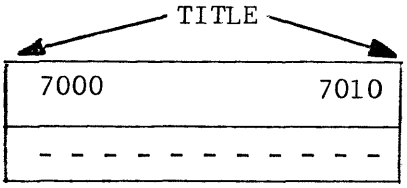
Assembly of each instruction will generate properly oriented addresses for pure symbolic names in the A and B ADDRESS Fields.

The format of each instruction is as follows:

	LOCATE ABSENCE OF SYMBOL LEFT	LOCATE ABSNECE OF SYMBOL RIGHT
OPERATION	LAL	LAR
SIZE	Specified Character	
A ADDRESS (Tag of field to be searched)	Leftmost Location	Rightmost Location
B ADDRESS (Tag of field to be searched)	Rightmost Location	Leftmost Location

The contents of memory (except STA) are not affected by this instruction. STA (0212-0215) will contain the address of the last location examined which contained the specified character. If the specified character was not present in the first location examined, the address in STA will be one location to the left or right in the LAL or LAR instruction respectively.

	LOCATE ABSENCE OF SYMBOL LEFT	LOCATE ABSENCE OF SYMBOL RIGHT
	EXAMPLE	EXAMPLE
HSM Before and After Execution	ACTNO: 5102 5107 / 0 0 0 1 2 3	NAME: 5120 5128 / S M I T H - - - -
Instruction	LAL 0 ACTNO,ACTNO K 0 5102 5107	LAR - NAME,NAME L - 5128 5120
FINAL REGISTER CONTENTS	* A_f = 5104 B_f = 5107	* A_f = 5125 B_f = 5120
	*STA is performed. PRP is set.	*STA is performed. PRP is set.

	LOCATE ABSENCE OF SYMBOL LEFT	LOCATE ABSENCE OF SYMBOL RIGHT
HSM Before and After Execution		
Instruction	LAL 0 WBAL,WBAL K 0 5705 5710	LAR - TITLE,TITLE L - 7010 7000
FINAL REGISTER CONTENTS	A_f^* = 5704 B_f = 5710 *STA is performed. PRN is set.	A_f^* = 7000 B_f = 7000 *STA is performed. PRZ is set.

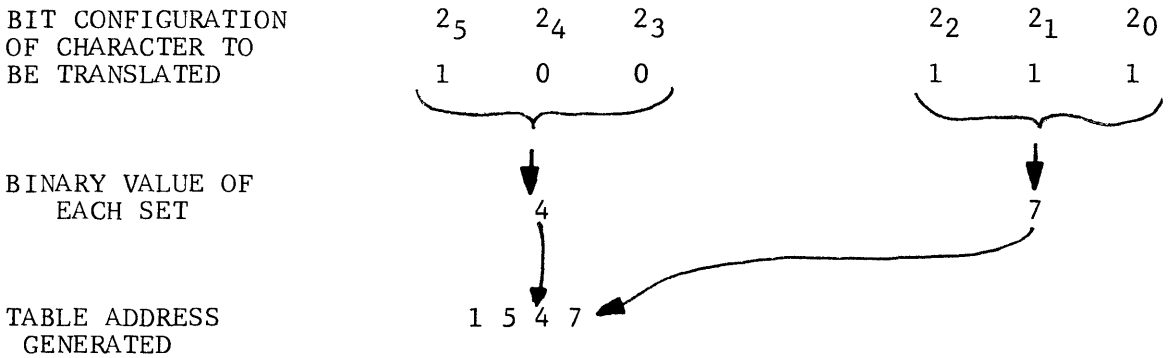
TRANSLATE BY TABLE Instruction

This instruction translates characters of a given code to characters of another code using a table which is present in memory. The information (2^0-2^5) bits of the character to be translated are used to generate the two least significant characters of the table address.

The table used for translation is stored in memory with the first character of the table hundreds (H) oriented.

Assuming at execution time the table was loaded at 1500 every table address in the translation process would have 15 as the first two characters. The second two characters of the table address would be generated by the bit configuration of the character being translated.

The information bits are split into two sets of three bits each and the binary value of each set of three bits are used to generate the last two characters of the table address as shown below.



Thus, in location 1547 would be stored the character desired for the translation result. This instruction may be used for functions such as:

a. Translation of data in a "foreign" code.

b. In conjunction with (a) above, substitution of characters for which there is no equivalent character in the translated code.

c. Validation of fields (see Sample problem).

d. Conversion of fields and random address generation.

Translation of data is done in place. The character to be translated is replaced by the character in the table location addressed. The instruction is executed from left to right. A left-end address, therefore, will be assembled for a pure symbolic name appearing in either the A ADDRESS Field (data to be translated) or the B ADDRESS Field (translation table).

The format of the instruction is as follows:

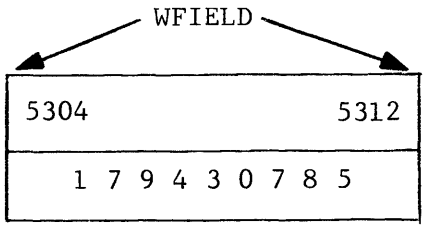
TRANSLATE BY TABLE

OPERATION	TBT
SIZE	Number of characters (0-45) to be translated.
A ADDRESS	Tag of the field containing the characters to be translated.
B ADDRESS	Tag of the translation table (hundreds oriented).

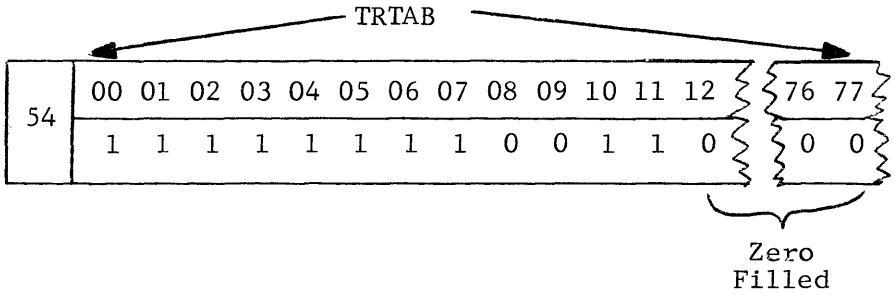
EXAMPLE (Coding and allocation of the table for this example is shown in Sample Problem for this manual.)

In this example the TBT instruction is used to validate a field that should consist of all numeric characters. A table constructed with the same character (1) in each location will have an address generated by a numeric character. In each of the other locations of the table, a different character (0) is stored. If the field to be translated is a valid (all numeric) field, it will consist of all characters with a value of one (1) after execution of the instruction. This field could then be tested by use of a LAL (or LAR) instruction followed by a CTC instruction. A setting of PRZ would indicate a valid field.

HSM Before Execution



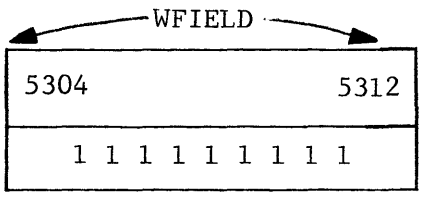
HSM Before and After Execution



Instruction

```
TBT 9 WFIELD,TRTAB
 A  9 5304   5400
```

HSM After Execution



It should be noted that the data originally appearing in this field has been replaced. If the original data must be preserved, translation of this type should be done in a work area.

FINAL REGISTER CONTENTS

A_f = 5313 B_f = 5400

REVIEW PROBLEM NO. 5.

The problem is to update a master file with a transaction file (See INPUT Formats). A transaction may be either a check (C-Code) or a deposit (D-Code).

The following actions are taken for each type of transaction:

Check - Add amount to Total Checks, Subtract amount from balance.

Deposit - Add amount to Total Deposits, add amount to Balance.

Assume the following conditions:

1. There will not be more than one transaction per master.

2. Each transaction will match a master record.

The output master record format will be the same as the input format.

For each record having a negative balance, write out a 120 character record as shown under the output format EDIT. The 11-character Balance field is to be edited in the following format, zero-suppressed and a floated dollar sign (---,---,--•).

INPUT

MASTER		TRANSACTION	
ACCT NO.	10	ACCT NO.	10
NAME	15	CODE	1
ST. ADDR.	17	C=CHECK	
CITY ST. ADDR.	15	D=DEPOSIT	
TOTAL DEPOSITS	8	AMOUNT	6
TOTAL CHECKS	7	SPACES	3
BALANCE	8		

OUTPUT

MASTER	EDIT	
SAME AS INPUT	ACCT NO.	10
	SPACES	5
	BALANCE	11
	SPACES	5
	NAME	15
	SPACES	74

SECTION XI

LOGICAL INSTRUCTIONS

INSTRUCTIONS COVERED IN THIS SECTION

Logical And
Logical Exclusive Or
Logical Inclusive Or

The Logical instructions are used to perform operations on the individual bits of character(s). The Logical instructions operate on the information bits (2^0 to 2^5). Proper parity (2^6 bit) is generated for each character in the result field based on the configuration of information bits. Each field may contain as many as 45 characters. The instructions operate from right to left on the same relatively positioned bits in each character. In the Logical instructions, the result after execution of each instruction replaces the original contents of the A addressed field.

The format of each of the Logical instructions is as follows:

	*LOGICAL AND	LOGICAL EXCLUSIVE OR	LOGICAL INCLUSIVE OR
OPERATION	LAN	LEO	LIO
SIZE	Number of characters in each field (0-45).		
A ADDRESS	Tag of first field and the field which will contain the result after execution of the instruction.		
B ADDRESS	Tag of the second field.		

*Logical And is the only Logical instruction that affects the PRIs. PRP is set if at least one of the information bits in the result is a 1 bit. PRN is set if all of the information bits are zeros in the result.

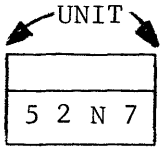
LOGICAL AND Instruction

The Logical And instruction may be used to test the presence of a 1 bit in any position of a field of one or more characters. The rule of Logical And is that a 1 bit in the same relative bit position of both fields will produce a 1 bit in that same position in the result. Any other combination will produce a zero bit.

PRP is set if any of the information bits in the result are 1 bits. Thus, by using a character with a 1 bit in any positions desired for testing, the Logical And instruction followed by a CTC instruction may be used.

EXAMPLE 1

In this example a four character field UNIT is to be tested for the presence of a 1 bit in the 2^5 position which may be present in any of the characters. Assume for example the field UNIT contains the following value.



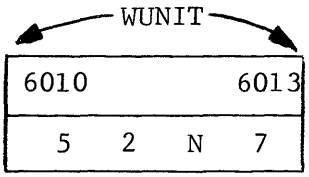
To test the field (UNIT) for the presence of a 1 bit in the 2^5 position in any character, we could allocate a field of four characters with a 1 bit in only the 2^5 position as follows:

LOCATION	OPERATION	SIZE	ADDRESS
MINUS	FIXCON	4	- - - -

So that the field UNIT will not be altered by the Logical And instruction, it would be moved to a work area (WUNIT). The fields would appear and be executed upon as follows (with assumed HSM allocations):

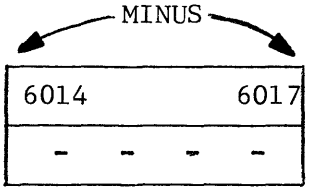
BIT CONFIGURATIONS
(Only Information Bits Shown)

HSM Before Execution



000101 000010 100101 000111

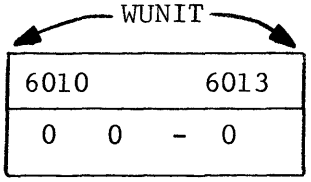
HSM Before and After Execution



100000 100000 100000 100000

Instruction

```
LAN  4  UNIT,MINUS
 T   4  6013  6017
```



000000 000000 100000 000000

FINAL REGISTER CONTENTS

A_f= 6009 B_f = 6013

PRP is Set

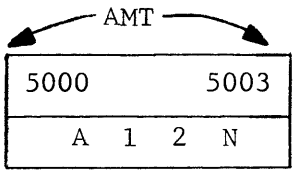
By following the Logical And instruction with a CTC instruction, the PRP condition would indicate a 1 bit in the 2^5 position was present in the field.

The Logical And instruction may also be used to extract 1 bits from any position in a field of one or more characters. This may be accomplished by using a masking character of 1 bits in each position except the positions(s) in which the 1 bit(s) are to be extracted.

The following is an example of masking zone bits (2^4 and 2^5) of 1 in a field of four characters.

EXAMPLE 2

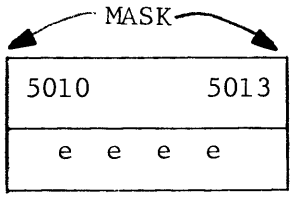
HSM Before Execution



BIT CONFIGURATIONS
(Only Information Bits Shown)

010001 000001 000010 100101

HSM Before and After Execution

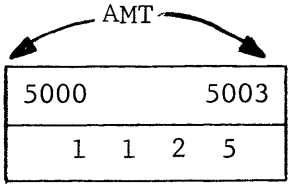


001111 001111 001111 001111

Instruction

```
LAN  4  AMT,MASK
T    4  5003 5013
```

HSM After Execution



000001 000001 000010 000101

FINAL REGISTER CONTENTS

$A_f = 4999$ $B_f = 5009$

PRP is Set

LOGICAL EXCLUSIVE OR Instruction

This instruction may be used to extract 1 bit(s) in specified bit position(s) of one or more characters. It must be known that a 1 bit is present if the LEO instruction is to be used. The rule of Logical Exclusive Or may be considered as the same as binary addition of two bits without a carry being generated, or as follows:

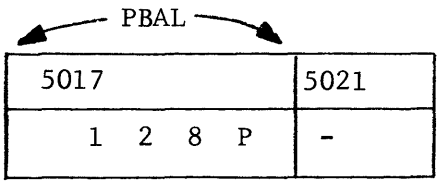
$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$0 + 1 = 1$$

$$1 + 1 = 0$$

As an illustration of the use of this instruction assume that a known negative field is to be printed and a minus sign has been placed adjacent to the field so that it appears as follows:

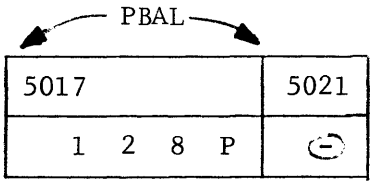


Before printing, it is desired to extract the 1 bit in the 2^5 position of the LSD of PBAL as the following example illustrates:

EXAMPLE 1

BIT CONFIGURATION
(Only Information Bits Shown)

HSM Before Execution

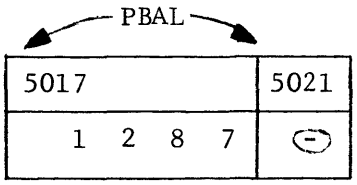


(P)	(-)
100111	100000

Instruction

```
LEO  1  PBAL, PBAL+$4
U    1  5020  5021
```

HSM After Execution



(7)	(-)
000111	100000

FINAL REGISTER CONTENTS $A_f = 5019$ $B_f = 5020$

LOGICAL INCLUSIVE OR Instruction

This instruction may be used to insert 1 bit(s) in any (information) bit position(s) of a character. The rule of LIO is that a 1 bit in the same relative position of either field will produce a 1 bit in the same position of the result, or as follows:

0 + 0 = 0

0 + 1 = 1

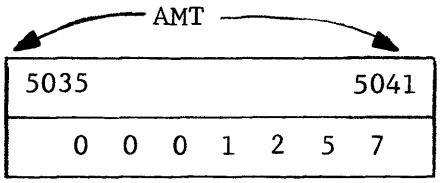
1 + 0 = 1

1 + 1 = 1

As an example of the use of this instruction assume that an amount field is to be either added to or subtracted from an accumulating field. A one character code field specifies whether addition or subtraction is to be performed. A zero (0) in the code field indicates the amount field is to be added to the accumulating field. A minus (-) indicates subtraction is to be performed.

The Code field could be tested for the presence of a zero or minus sign. A transfer of control to an Add or Subtract instruction could then be executed. However, if the codes have been validated, a Logical Inclusive Or instruction could be used as in each of the following examples with assumed values present in each field:

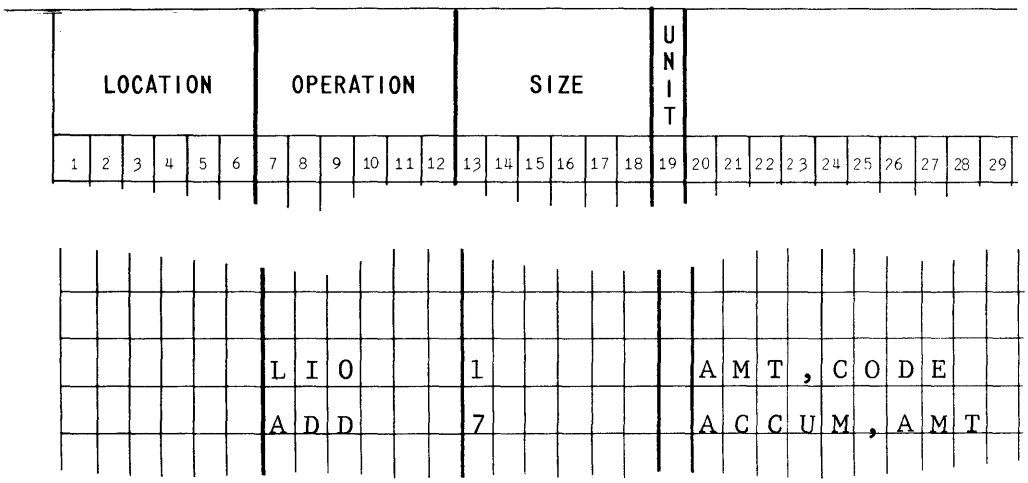
HSM Before Execution



	EXAMPLE 1	EXAMPLE 2
	CODE	CODE
Assumed Value of Code Field	5017: 0	5017: -
Instruction	LIO 1 AMT,CODE Q 1 5041 5017	LIO 1 AMT,CODE Q 1 5041 5017
HSM After Execution	AMT 5035 5041: 0 0 0 1 2 5 7	AMT 5035 5041: 0 0 0 1 2 5 P
FINAL REGISTER CONTENTS	A_f = 5040 B_f = 5016	A_f = 5040 B_f = 5016

The above example illustrates that the Code of zero does not alter the LSD of the AMT field as a zero consists of all zero bits. The minus character,

however, contains a 1 bit in the 2^5 position and when used as a modifying character with the LIO instruction will insert this 1 bit in the 2^5 position. In this example the AMT field has been made a negative field. After this use of the LIO instruction an ADD DATA instruction can be used as follows:



Thus, the amount (AMT) field is effectively added to or subtracted from the accumulating (ACCUM) field based on the presence or absence of the 2^5 bit of 1 in Code field.

RCA 3301 Assembly System

Practical Exercise No. 6

1. Show the result at the end of the following operations:

   a. $(100110)_2$ modified by $(000111)_2$ using a Logical And

   b. $(111000)_2$ $(111111)_2$ modified by $(000101)_2$ $(010011)_2$ using a Logical Inclusive Or.

   c. $(101010)_2$ modified by $(110011)_2$ using a Logical Exclusive Or.

2. What instruction(s) would you use to determine if the 2_3 bit of a character is a one or a zero? What would be the modifying character?

3. What instruction would you use to change 1 to a 2 the first time it was executed, 2 to a 1 the second time, 1 to a 2 the third time, and so forth? What would be the modifying character?

4. Suppose you wanted to insert a one bit into the 2^4 position of a character. What would be the instruction that you would use and what would be the modifying character?

5. If you wanted to extract a one bit, if present, from the 2^2 position of a character, what would be the instruction you would use? What would be the modifying character?

SECTION XII

VARIABLE FIELD TRANSFER AND

ADDRESS ARITHMETIC INSTRUCTIONS

INSTRUCTIONS COVERED IN THIS SECTION

Transfer by Symbol Left
Transfer by Symbol Right
Add Address
Subtract Address
Compare Address

TRANSFER BY SYMBOL Instructions

There are two instructions that may be used for the transfer of variable-sized fields. Both instructions have the same format in the SIZE and ADDRESS Fields as follows:

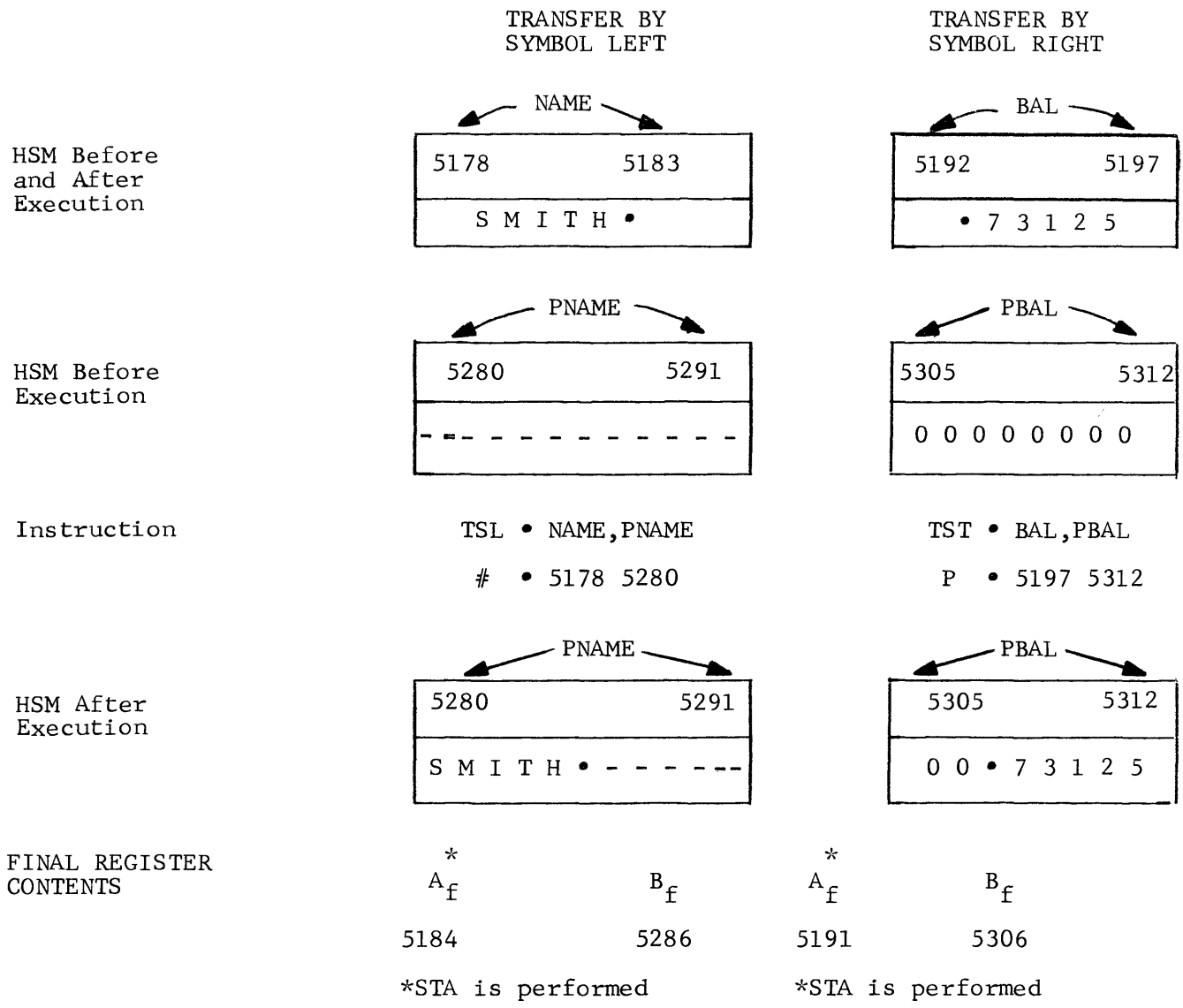
	TRANSFER BY SYMBOL LEFT	TRANSFER BY SYMBOL RIGHT
OPERATION	TSL	TSR
SIZE	Actual Character which, when transferred, will terminate execution of the instruction	
A ADDRESS	Tag of the Sending Area	
B ADDRESS	Tag of the Receiving Area	

The instructions are executed similarly to the Transfer by Count instructions. Instead of being terminated by the number of characters being transferred, each of these instructions is terminated upon the transfer of a specified (control) character. The actual character upon which to stop transfer is written in the SIZE Field.

Assembly of the instruction generates left-end addresses for pure symbolic names in the TSL ADDRESS Fields and right-end addresses for the TSR ADDRESS Fields.

STA is performed. The A Final Register setting is one location to the right (TSL) or left (TSR) of the last character (SIZE specified) transferred in the sending area. The B Final Register setting is in the same relative position in the receiving area.

EXAMPLE



FINAL REGISTER CONTENTS

	A_f*	B_f
TRANSFER BY SYMBOL LEFT	5184	5286
TRANSFER BY SYMBOL RIGHT	5191	5306

*STA is performed

*STA is performed

The following example, given for illustrative purposes, has symbolic names assigned to the sending fields.

In data records that consist of variable fields, however, symbolic tags are usually assigned only to the fixed length fields which are at the beginning of the record. Any field following the first variable size field will be in a different area dependent on the size of the first variable field.

As an example, if a record consisted of five items in the format as follows:

ITEM NO.	ITEM NAME	MAXIMUM NO. CHARS*		ASSUMED SAMPLE RECORD
1	STOCK NO.	9	FIXED LENGTH	•02314789
2	ITEM NAME	15	VARIABLE LENGTH	•PIN-02
3	BALANCE	8	VARIABLE LENGTH	•13
4	TOTAL ISSUES	9	VARIABLE LENGTH	•127
5	TOTAL RECEIPTS	9	VARIABLE LENGTH	•142

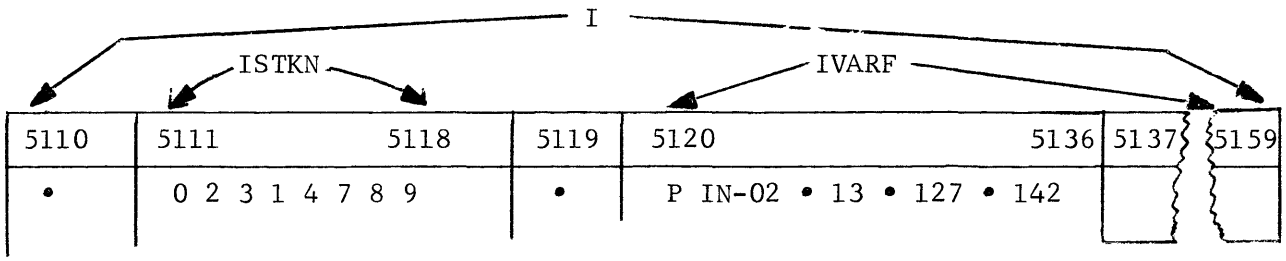
*including control symbol (•) preceding each field.

The length of the item name would determine the locations assigned to each of the following items (assuming the record is left justified in a record area).

Allocation of the area for storage of the record might be as follows:

LOCATION	OPERATION	SIZE	UNIT	REMARKS
I	ALOC		W	
		1		ISS of First Item
STKN		8		STOCK NO.
		1		ISS of Second Item
VARF		40		Variable Fields

and assuming an object time allocation of a record area from 5110 to 5159 the actual record would appear in memory as follows with symbolic names shown:



It can be seen that the only items that can be addressed by pure symbolic tags are the Stock No. (ISTKN) and the left-end address of the Item Name (IVARF) in an instruction which has a left-end address assembled.

The location of each of the other items will vary with each record dependent upon the relative sizes of the items.

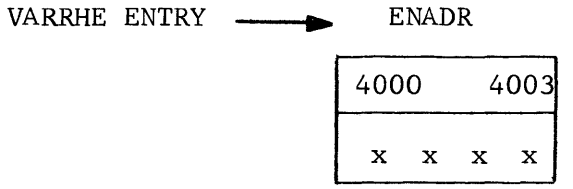
Thus, in the handling of variable records, a variable item is located by its relative position in the record. Thus, the fourth item will always follow the fourth control symbol (assuming <u>all</u> items are preceded by a control symbol).

When FCP has moved a record to a record processing area, it will store the address of the rightmost character of the record in a four character location specified by the programmer (See VARRHE, Section V of this manual). Thus the location of both the beginning and end items of a variable input record can be easily determined.

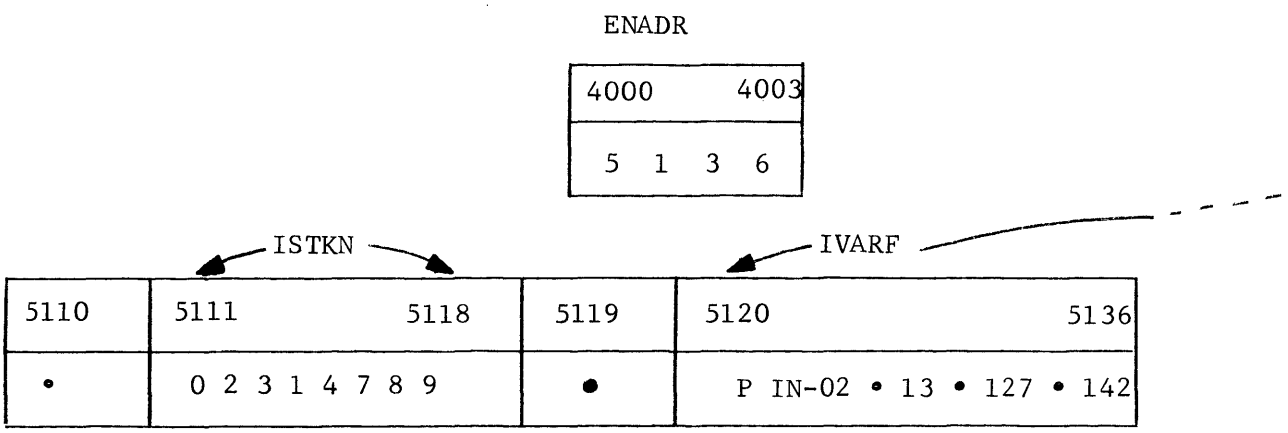
As an example of addressing variable fields, assume that the items of the record used in the preceding example are to be moved to the following fixed fields with the items justified in the fields as follows:

STOCK NO.	PSTKN	LEFT	6100-6107
ITEM NAME	PNAME	LEFT	6110-6124
BALANCE	PBAL	RIGHT	6130-6137
TOTAL ISSUES	PTISS	RIGHT	6140-6148
TOTAL RECEIPTS	PREC	RIGHT	6150-6158

Assume also the VARRHE entry for the file has been specified as follows with assumed HSM allocation:



Following the READ of the file, the record would be in the record area and ENADR would contain values as indicated



A section of the coding for transfer of the fields would be as follows:

ASSEMBLY CODING				INSTR. IN HSM AT OBJECT TIME	INSTR.AFTER STATICIZING
OPERATION	SIZE	UNIT	ADDR		
TCL	8		ISTKN,PSTKN	M 8 5111 6100	No Change
TSL	•		IVARF,PNAME	# • 5120 6110	No Change
TSR	•		ENADR#,PREC	P • 400C 6158	P • 5136 6158
TSR	•		$STA#,PTISS	P • 02IE 6148	P • 5132 6148
TSR	•		$STA#,PBAL	P • 02IE 6137	P • 5127 6137

ADDRESS ARITHMETIC Instructions

When necessary to modify an address by addition or subtraction, the ADD ADDRESS or SUBTRACT ADDRESS instruction should be used. These instructions give the correct address result in a four character field specified in the A ADDRESS For an address (10,000 and above) with zone bits in the first characters, these instructions will create proper results whereas the ADD (or SUBTRACT) DATA instruction would remove zone bits.

The format of each of the instructions is as follows:

	ADD ADDRESS	SUBTRACT ADDRESS
OPERATION	AAD	SAD
SIZE	(Blank) NO ENTRY REQUIRED	
A ADDRESS	TAG of first field and field which will contain the result after execution	
B ADDRESS	Tag of the second field	

The PRI's are set for each of the instructions as follows:

	ADD ADDRESS	SUBTRACT ADDRESS
PRP	SUM IS POSITIVE	DIFFERENCE IS POSITIVE
PRZ	SUM IS ZERO	DIFFERENCE IS ZERO
PRN	(NOT SET)	ADDRESS IN FIRST FIELD IS LESS THAN ADDRESS IN SECOND FIELD

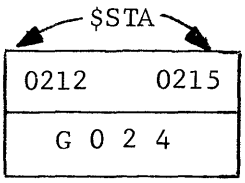
The Overflow Indicator is set by these instructions as follows:

ADD ADDRESS - When the result is a "wrap around" address with a value above 159,999.

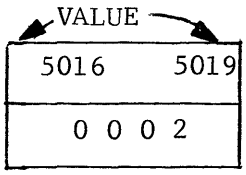
SUBTRACT ADDRESS - When the result is a "wrap around" address with a value "below" 0000.

EXAMPLE 1

HSM Before Execution

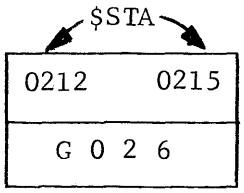


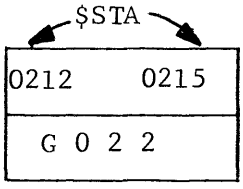
HSM Before and After Execution



Instruction	ADD ADDRESS	SUBTRACT ADDRESS
	AAD $STA,VALUE	SAD $STA,VALUE
	+ + 0215 5019	- + 0215 5019

HSM After Execution





FINAL REGISTER CONTENTS				
	A_f = 0211	B_f = 5015	A_f = 0211	A_f = 5015
	PRP is set		PRP is set	

COMPARE ADDRESS Instruction

This instruction is used to compare two addresses and set the PRI's based on the relative value of each address. This instruction does not alter either of the addresses.

This instruction, rather than the COMPARE DATA Instruction, should be used when addresses are to be compared. The configuration of zone bits in the first two characters of addresses may not give the true relative values of addresses if other than the COMPARE ADDRESS instruction is used (see example).

The format of the instruction is as follows:

COMPARE ADDRESS

OPERATION	CAD
SIZE	(Blank) No entry required
A ADDRESS	Tag of the field containing the first address
B ADDRESS	Tag of the field containing the second address

The PRI's are set as follows:

PRP - The first address is greater than the second address

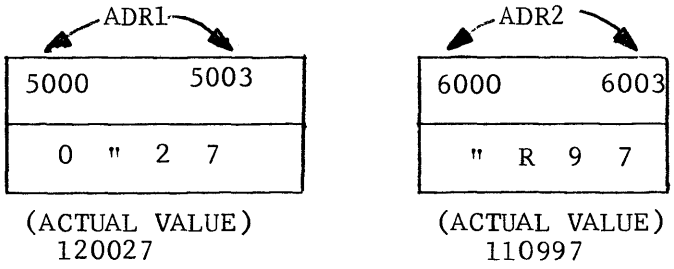
PRN - The first address is less than the second address

PRZ - The first and second addresses are equal in value

This instruction operates from right to left, thus, right-end addresses will be assembled for pure symbolic names appearing in the A and B ADDRESS Fields.

EXAMPLE

HSM Before and After Execution



(ACTUAL VALUE) 120027

(ACTUAL VALUE) 110997

Instruction

```
CAD      ADR1,ADR2
 -   .   5003 6003
```

FINAL REGISTER CONTENTS

A_f = 4999 B_f = 5999

PRP is set

RCA 3301 ASSEMBLY SYSTEM

Practical Exerise #7.

Write a program which will update a Master Insurance file by posting the received Premium Amounts to the appropriate Master File Records. There will be no more than one transaction per master record. An ISS (•) precedes each variable field.

Master File	Max. Chars.	
Policy Number	9	Fixed
Name	30	Variable
Street Address	30	Variable
City State Address	30	Variable
Total Premiums Paid to Date	6	Variable
Premium Due	6	Variable

Transaction		
Policy No.	9	Fixed
Premium Amount	5	Fixed

Write any unmatched transactions to an output error tape.

SECTION XIII

REPEAT AND TALLY INSTRUCTIONS

INSTRUCTIONS COVERED IN THIS SECTION

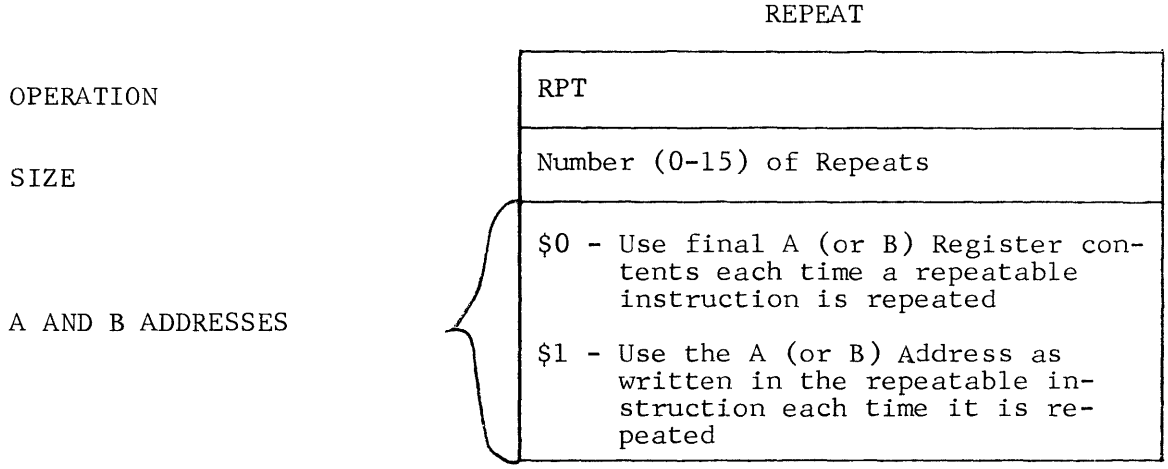
Repeat
Tally

REPEAT Instruction

In the 3301 complement, certain instructions are designated as "repeatable" instructions. Thus, if one of these instructions is written following a Repeat instruction, it is executed once and repeated a number of times (maximum of 15) as specified in the Repeat instruction.

The instruction to be repeated is executed the first time as written. When it is repeated (the number of times specified in the Repeat instruction), the programmer has two options. One of the options is the use of the A (or B) Address as written each time the instruction is repeated. The other option is the use of the final register setting when repeating the instruction.

The format of the Repeat instruction is as follows:



Any even or odd machine address may be substituted for the $0 or $1 addresses respectively. If a pure symbolic name is used in either the A or B ADDRESS Field, the left-end address will be assembled as the address.

EXAMPLE (Cumulative Addition)

(In this example the Repeat instruction is used instead of four separate ADD DATA instructions to accumulate the first four fields into the last field.)

HSM Before Execution

ITEMA	ITEMB	ITEMC	ITEMD	SUM
5103	5107	5111	5115	5119
0 0 1 2	0 0 4 0	0 9 6 3	0 1 2 7	0 0 0 0

Instructions

	ASSEMBLY INSTRUCTION			EXECUTION INSTRUCTION			
RPT	3	$1,$0		R	3	0001	0000
ADT	4	SUM,ITEMD		+	4	5119	5115

HSM After Execution

5103	5107	5111	5115	5119
0 0 1 2	0 0 4 0	0 9 6 3	0 1 2 7	1 1 4 2

FINAL REGISTER CONTENTS (ADT INSTRUCTION)

A_f = 5115 B_f = 5099

EXAMPLE (Location of a Variable Field)

In this example the Repeat instruction is used with a Transfer by Sybmol Left (TSL) instruction to locate a variable field. The field is to be transferred and right-justified in a fixed field (WAREA). The TSL instruction is done in place; i.e., each character is placed back in the same location.

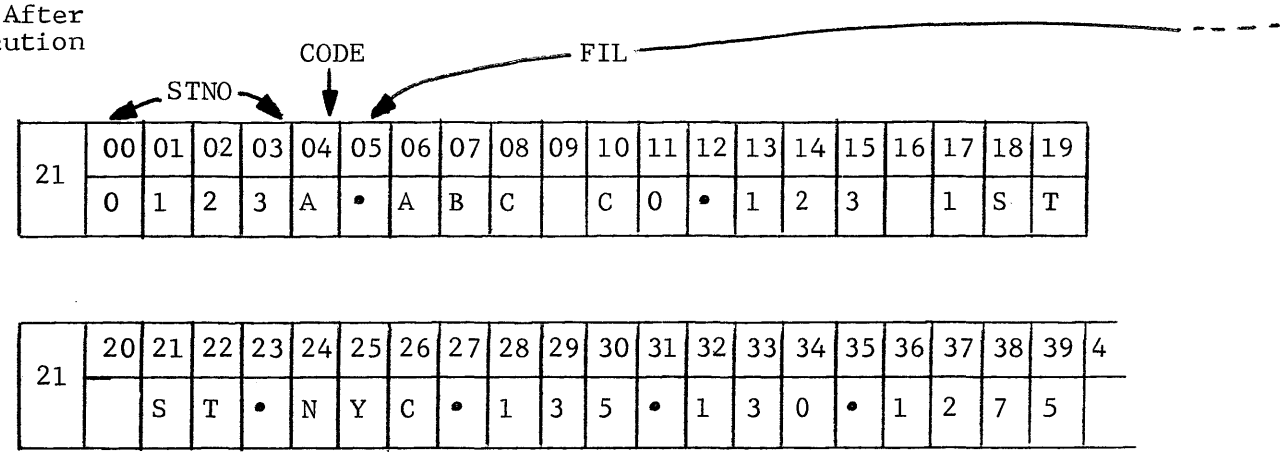
Assume the record is in the following format and that only the variable sized items are preceded by an Item Separator (•) Symbol.

STOCK NO.	4	FIXED SIZE
CODE	1	FIXED SIZE
MFGR. NAME	15	VARIABLE
STREET ADDR.	20	VARIABLE
CITY-STATE	20	VARIABLE
CURRENT ISSUES	8	VARIABLE
CURRENT RECEIPTS	8	VARIABLE

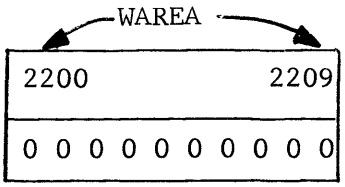
(ADDITIONAL ITEMS NOT SHOWN)

The instructions in the following example will locate the Current Receipts item and transfer it right-justified to the fixed field (WAREA). Assume a sample record for illustrative purposes as follows:

HSM Before and After Execution



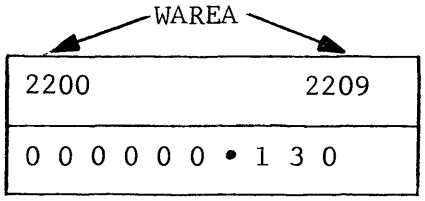
HSM Before Execution



Instructions

LOCATION	OPERATION	SIZE	UNIT	
LOC	RPT	4		$0,$2
	TSL	•		FIL+$1,FIL+$1
	SAD			$STA,LOC+$9
	TSR			$STA#,WAREA

HSM After Execution



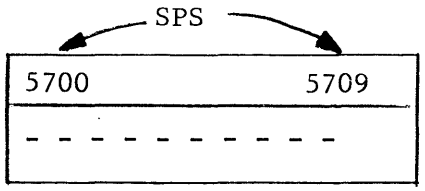
FINAL REGISTER CONTENTS (TSR INSTRUCTION)

A_f = 2130 B_f = 2205

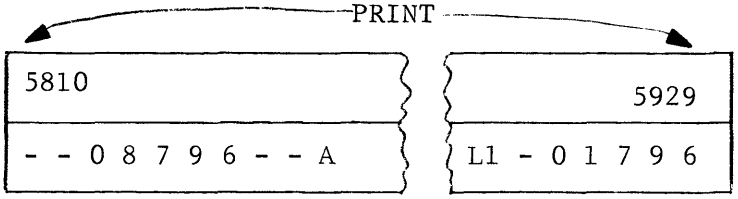
EXAMPLE

(Use of the Repeat instruction to Fill an area with a specified symbol using a Transfer Decade instruction.)

HSM Before and After Execution



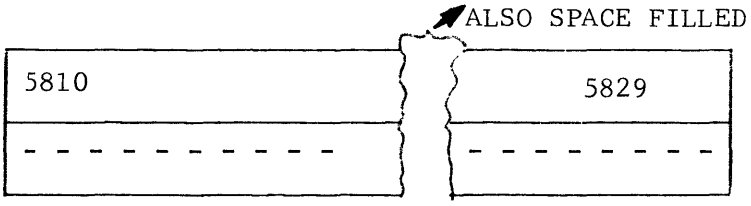
HSM Before Execution



Instruction

```
RPT    11    $1,$0
TDC     1    SPS,PRINT
```

HSM After Execution



FINAL REGISTER CONTENTS (TDC INSTRUCTION)

A_f = 5710 B_f = 5830

TALLY Instruction

The Tally instruction may be used to perform either or both of the following functions:

1. Loop through a given routine a specified number (99 max.) of times.
2. Increment any or all of the Index Fields.

The format for each of these options is as follows:

FORMAT 1 — TALLY (LOOPING function only)

OPERATION	TLY
SIZE	A zero (0) is written in Col. 13
A ADDRESS	Tag of the diad containing the Tally counter quantity
B ADDRESS	Tag of the location to which control is to be transferred if the Tally counter quantity is not 00.*

*When the Tally quantity is 00, the B Address is ignored and the next instruction in sequence after the Tally instruction is executed.

FORMAT 2 — TALLY (INDEX INCREMENTATION function only)

OPERATION	TLY
SIZE	Ix Ix,Y Ix,Y,Z Either of these entries where X,Y,Z represent any combination of 1,2,3 for incrementation of the first, second, and third Index Fields respectively.
UNIT	N (This character entered in Col.19)
A ADDRESS	$0 (Ignored Address)
B ADDRESS	$0 (Ignored Address)

FORMAT 3 — TALLY (Combination of Looping and Index Incrementation function)

OPERATION	TLY
SIZE	Ix Ix,Y (See SIZE entry of Format 2) Ix,Y,Z
UNIT	(Blank) or a Y in Col. 19 specifies the looping function is to be performed
A ADDRESS	Tag of the Diad containing the Tally counter quantity
B ADDRESS	Tag of the location to which control is to be transferred if the Tally counter quantity is not 00.

EXAMPLE (LOOPING AND INDEX INCREMENTATION FUNCTION)

In this example assume that there are six 80 character records present in HSM. The records are contiguous in an area tagged (RECS) and each record is in the format:

ITEM	NO. OF CHARS.
ACCT NO.	7
BAL.	8
FILLER ITEMS	65

It is desired to move the first two items to a 120 character print area which has been tagged (PRINT). It is desired to print six Accounts on each print line with each of the six contiguous 20 character locations having the format:

ITEM	NO. OF CHARS.
ACCT NO.	7
SPACE	3
BAL.	8
SPACE	2

To accomplish this function the programmer has allocated Index 2 for incrementation of the record (RECS) area address and Index 3 for the Print (PRINT) area address. The Tally counter and Index Fields would be initialized each time before entering the looping routine as follows:

HSM and MMM Before Execution

MICROMAGNETIC MEMORY

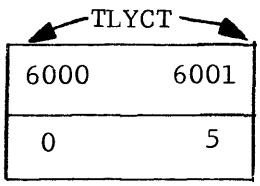
INDEX 2 | INCR. 2

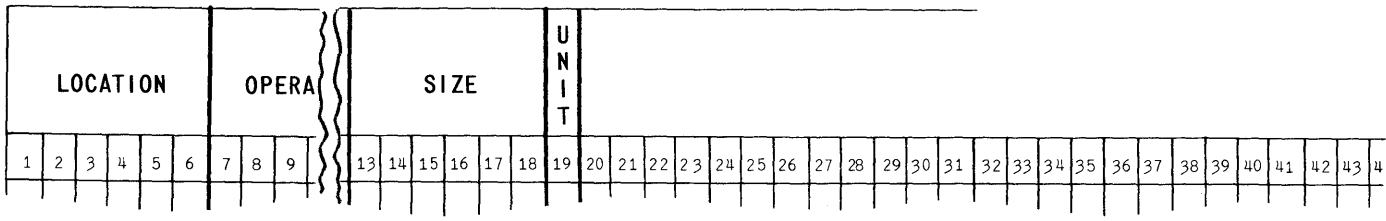
0 0 0 0 | 0 0 8 0

INDEX 3 | INCR. 3

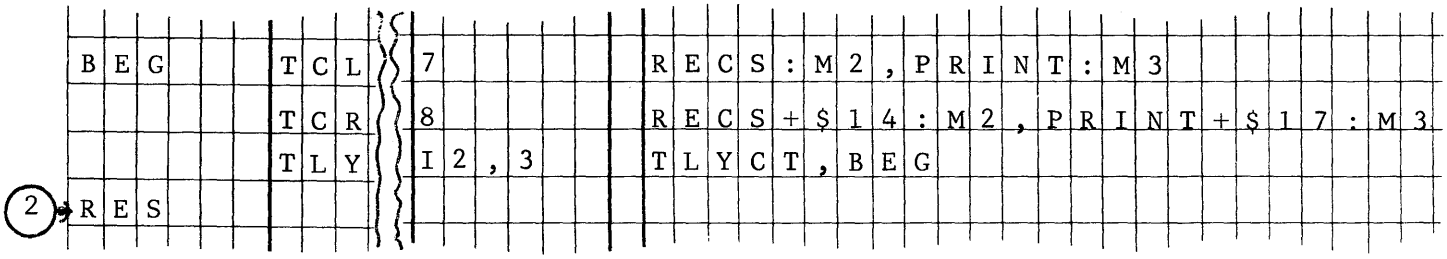
0 0 0 0 | 0 0 2 0

HSM





(1) → INITIALIZATION INSTRUCTIONS



HSM and MMM After Execution

MICROMAGNETIC MEMORY

IND. MOD. 2 | INCR. 2

0 0 8 0	0 0 8 0

IND. MOD. 3 | INCR. 3

0 0 2 0	0 0 2 0

HSM

TLYCT

6000	6001
0	4

(1) Initialization instructions might include setting Index Fields (see Section XIV) and Tally counter to initial values and also such functions as clearing the PRINT area if necessary.

(2) Instruction (RES) will be executed when the Tally instruction examines the Tally counter (TLCT) and finds a value of 00 present. Otherwise, the instruction BEG would be executed next.

SECTION XIV

REGISTER MANIPULATION INSTRUCTIONS

INSTRUCTIONS INCLUDED IN THIS SECTION

Load Register
Store Register

GENERAL

The Load Register and Store Register instructions, as the names imply, enable the programmer to address registers in Micromagnetic Memory for the purpose of either placing a value in a designated register from a four character location of HSM, or placing the contents of a register in a four character location of HSM.

The four character location of HSM used to load or store a register must be diad oriented; i.e., consist of two diads.

For the purposes of simplification only, the Extended Operation Codes are shown in the formats and examples of the instructions.

It should be noted that whenever the P Register contents are changed by either a Load or Store Register instruction, a transfer of control will take place.

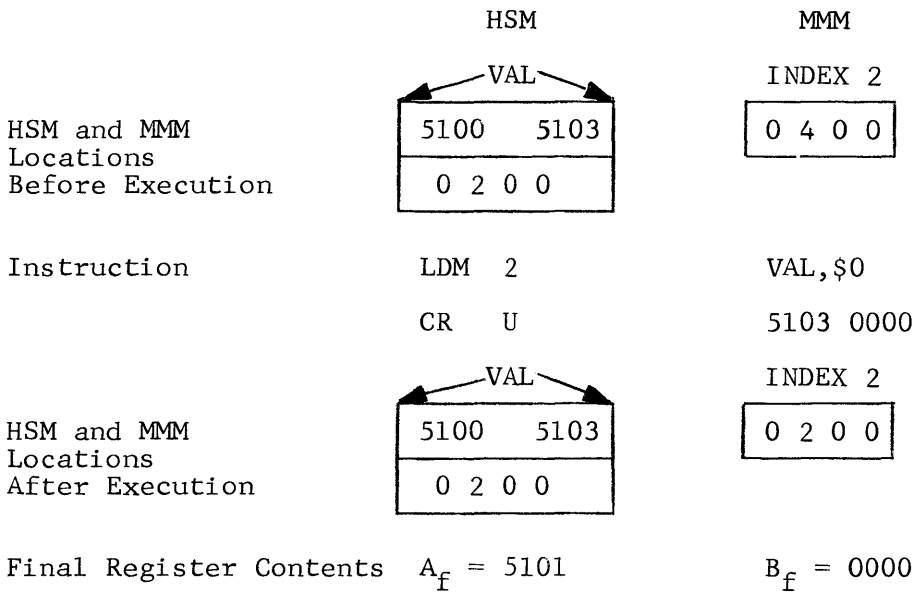
LOAD REGISTER Instruction

This instruction places the contents of a four-character (two diads) location of HSM into a designated register or Index Field of Micromagnetic Memory.

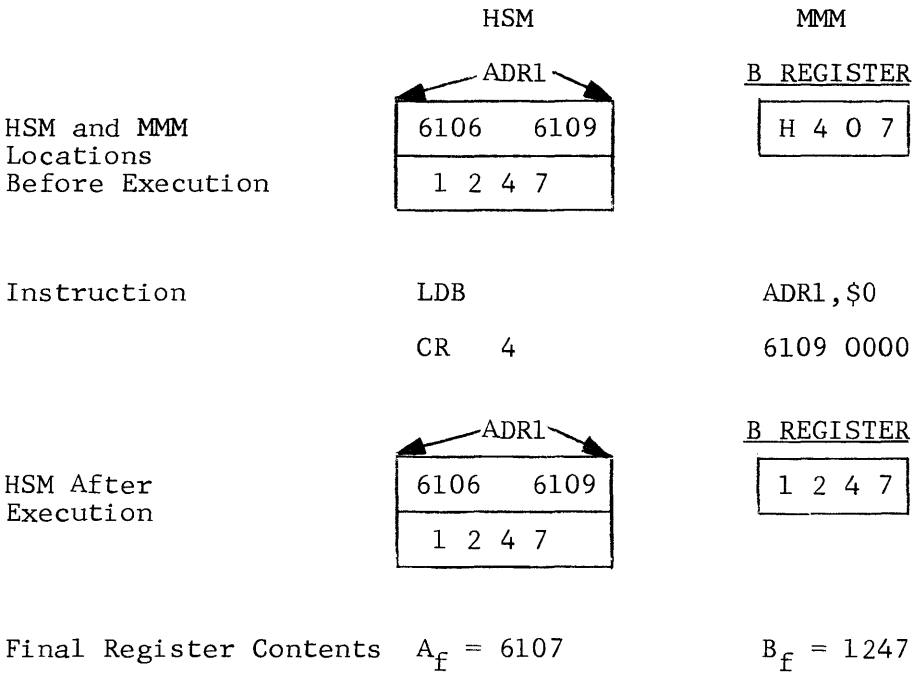
The format of the instruction is as shown below.

REGISTER OR INDEX FIELD TO BE LOADED →	REGISTERS							INDEX NO.			INDEX INCREMENT NO.		
	P	A	B	T	C	E	S	1	2	3	1	2	3
OPERATION	LDP	LDA	LDB	LDT	LDC	LDE	LDS	LDM	LDM	LDM	LDI	LDI	LDI
SIZE	(Blank) No Entry Required							1	2	3	1	2	3
A ADDRESS	Tag of the two-diad location whose contents are to be loaded into MMM.												
B ADDRESS	Zero ($0) This address is ignored if the B Register is being loaded. If other than the B Register is being loaded, this ADDRESS is placed in the B Register.												

EXAMPLE (Placing a value in an Index)



EXAMPLE (Loading the B Register)



STORE REGISTER Instruction

This instruction places the four-character contents of a Micromagnetic Memory location into two consecutive diads of High Speed Memory.

The format of the instruction is as shown below.

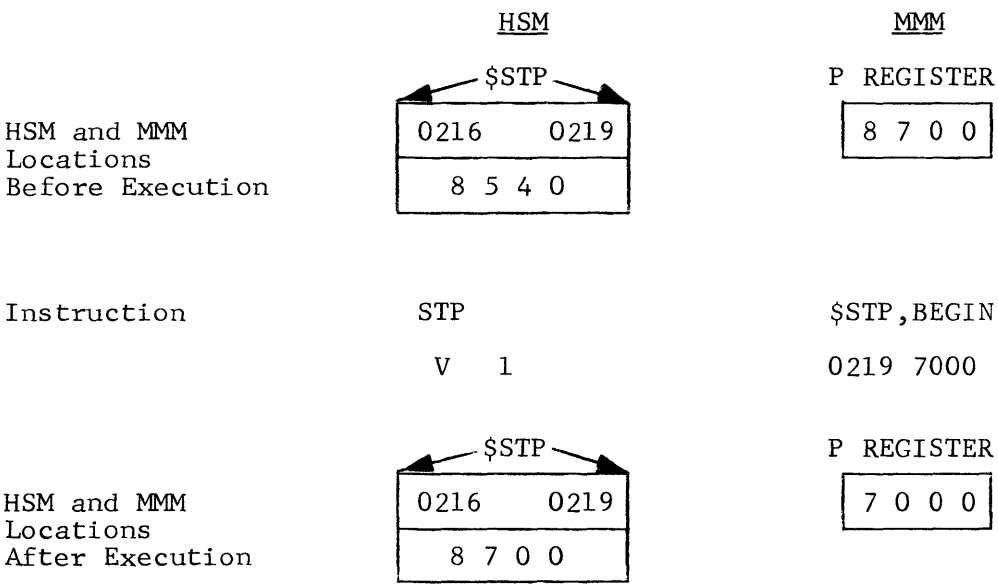
REGISTER OR INDEX FIELD TO BE STORED →	REGISTERS							INDEX NO.			INDEX INCREMENT NO.			None
	P	A	B	T	C	E	S	1	2	3	1	2	3	
OPERATION	STP	STA	STB	STT	STC	STE	STS	STM	STM	STM	STI	STI	STI	NOP
SIZE	(Blank) No entry required							1	2	3	1	2	3	(Blank)
A ADDRESS	Tag of the two diad location to receive the contents of the MMM location being stored. If the A Register is designated, the storage will be to STA (0212-0215) and any direct address (such as $0) must appear in this field.													
B ADDRESS	Tag of the next instruction to be executed if the P Register is being stored. If the P Register is not being stored, any direct address (such as $0) must be used.													

EXAMPLE (Storing the B Register)

	HSM	MMM
	CADR → 6106 6109	B REGISTER
HSM and MMM Locations Before Execution	1 2 5 8	4 9 5 6
Instruction	STB V 4	CADR,$0 6109 0000
	CADR → 6106 6109	B REGISTER
HSM and MMM Locations After Execution	4 9 5 6	4 9 5 6

Final Register Contents A_f = 6107 B_f = 4956

EXAMPLE (Storing the P Register)



Final Register Contents A_f = 0217 B_f = 7000

SECTION XV

PREPARATION OF THE SEGMENT DESCRIPTION

The purpose of the Segment Description is twofold. It informs the Assembler of all sequences which are contained in the segment and it may be used to establish linkages between the sequences. For example, where sequences have been written using the EXIT controlling code, linkage to another sequence is specified in the Segment Description using the SEQX controlling code.

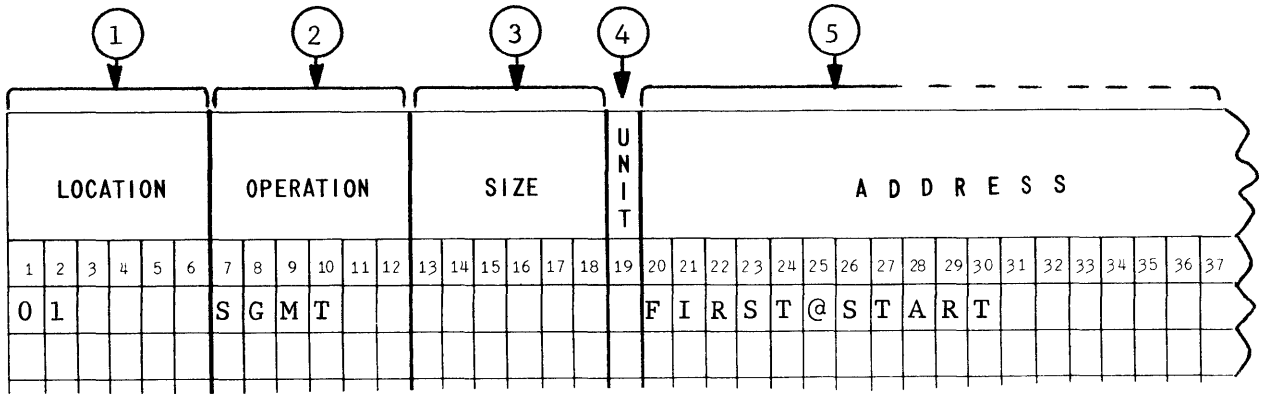
Actual preparation of the Segment Description might take place after all sequences have been written to be certain that the necessary linkages may be established.

However, the Segment Description may be written even before the individual sequences have been prepared. For example, a lead programmer might require that sequences be written in accordance with predetermined specifications and then assign individual programmers to the task of preparing sequences. An example outlining this method is given following the controlling code formats for the Segment Description.

The controlling codes and the format requirements for each are as follows:

SGMT Controlling Code

This controlling code informs the Assembler that a Segment Description is beginning. The SGMT line also assigns an Identification Number to the Segment and specifies where control is to be transferred after the segment is loaded. The format for the SGMT line is as follows:



FORMAT NOTES

(1) LOCATION

A Segment Identification Number must be entered in this field and must be left-justified. The number assigned each SGMT must be unique and in ascending order from 1 to 99 for each process.

(2) OPERATION

SGMT must appear in this field.

(3) SIZE

FORMAT NOTES

Not used for SGMT line

(4) UNIT

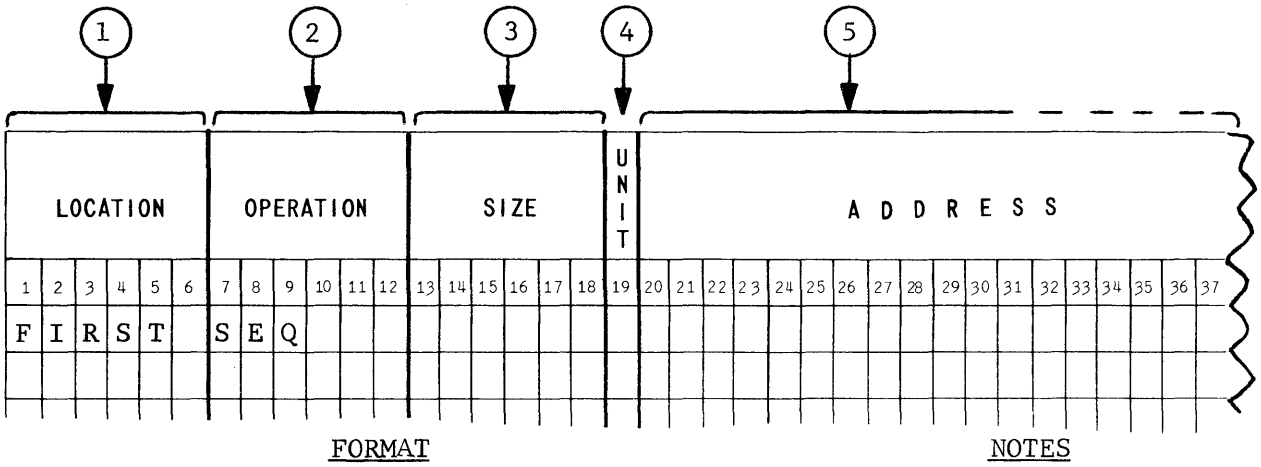
Not used for SGMT line

(5) ADDRESS

This field is used for designating the location to which control is to be transferred when the segment is loaded at object time. The entry must be in the format as shown above where FIRST represents the name of the sequence used for entrance at object time and where START represents the location within the FIRST sequence where entrance is to be made. The entry in the ADDRESS Field must be a direct symbolic address that is not modified by incrementing, decrementing, or indexing.

SEQ Controlling Code

This controlling code is used in the Segment Description to list all of the sequences that are included in the segment. Thus, there must be a SEQ line in the Segment Description for every DEFSEQ line that will appear in the segment.



(1) LOCATION

The name of the Sequence is entered in this field. Note that this entry must be the same as for the entry in this field on the DEFSEQ line for the corresponding sequence.

(2) OPERATION

SEQ must appear in this field.

Segment Description

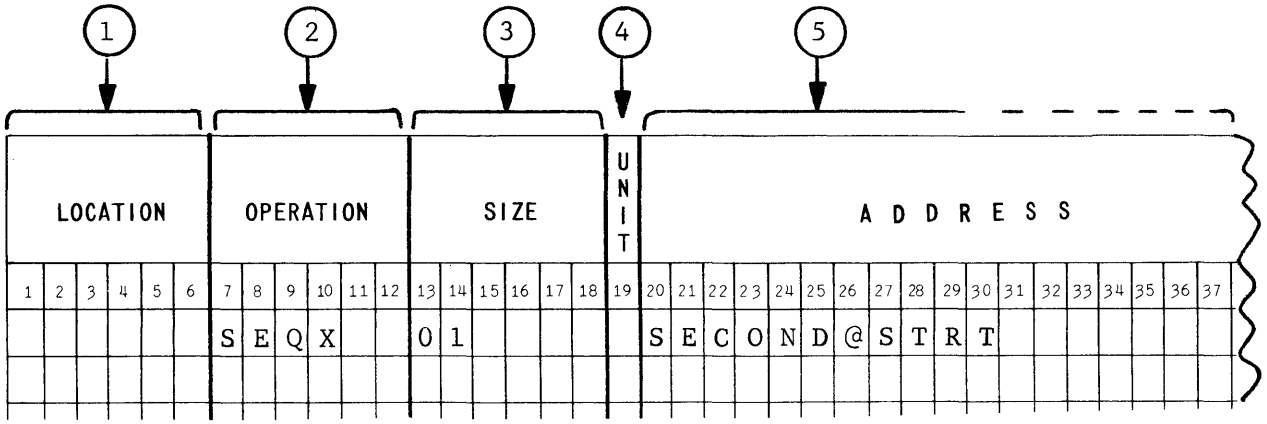
FORMAT NOTES

(3) (4) (5) SIZE UNIT ADDRESS

These fields may be left blank for the SEQ line.

SEQX Controlling Code

This controlling code is used to establish the linkage for a sequence in which the EXIT controlling code has been used. The linkage established may be to another sequence within the segment or may be a logical termination point of the segment.



FORMAT NOTES

(1) LOCATION

Not used for the SEQX line.

(2) OPERATION

SEQX must appear.

(3) SIZE

The two character decimal number appearing in this field must be the same number as assigned to the EXIT line in the corresponding sequence.

(4) UNIT

Not used for the SEQX line.

(5) ADDRESS

This field is used for designating the location to which control is to be transferred. If control is to be transferred to another sequence within the same segment the entry would be in the format as shown above where SECOND represents the name of the sequence to which control is to be transferred and STRT represents the location within the SECOND sequence where entrance is to be made.

Segment Description

FORMAT NOTES

(5) ADDRESS (Cont'd)

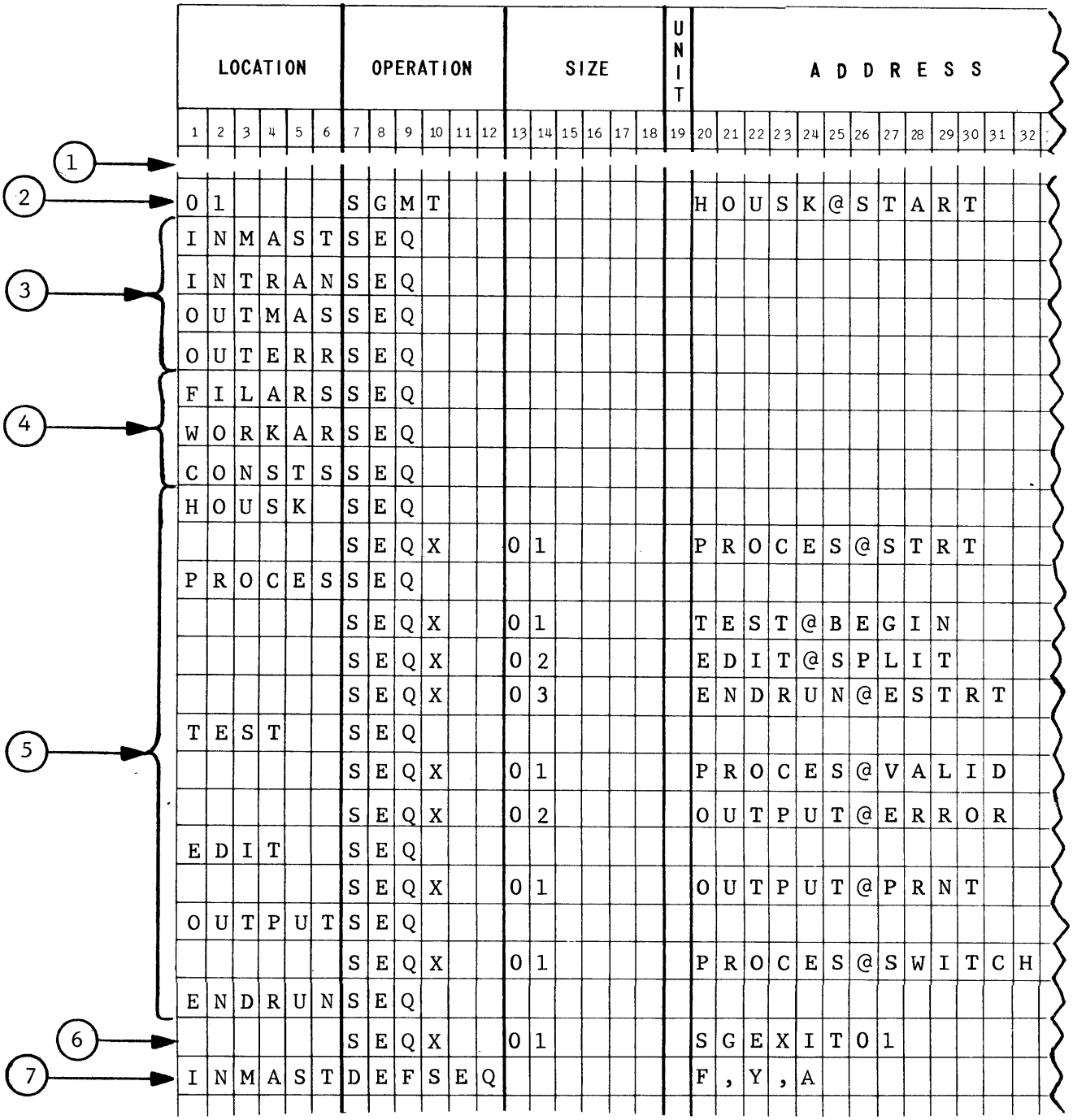
If control is to be transferred to another segment, the entry appearing in the ADDRESS Field would be SGEXITnn, where nn represents a decimal number identifying an exit from the segment. This is illustrated in the following example.

Note that the EXIT controlling code must be used in at least one of the sequences where a transfer of control out of the segment is necessary.

Segment Description

SEGMENT DESCRIPTION (Example No. 1)

The following is an example of a Segment Description for a segment in which there are four files being processed; an input master file, an input transaction file, an output master file, and an output error file. In addition there are sequences for allocating file areas, work areas, and constants. The sequences containing instruction coding are divided into Housekeeping, Process, Test, Edit, Output, and Endrun sequences and the exit from the segment is through the Endrun Sequence.



etc.

Segment Description Example No. 1

FORMAT

NOTES

(1) Preceding the Segment Description would be the START controlling code, and (if used) the NAME controlling code. These codes are described in the following section.

(2) SGMT line must have a number assigned in the LOCATION Field and an entrance location for a transfer of control at object time when the segment is loaded.

(3) These are SEQ lines for the FCP File Descriptor Sequences.

(4) These are SEQ lines for sequences in which I/O areas, work areas, and constants have been allocated.

(5) These are all sequences containing instructions.

NOTE that all of the sequences must have an EXIT controlling code line if SEQX is used in the Segment Description.

(6) Note that ENDRUN sequence must contain the EXIT controlling code and the related SEQX line specifies an exit (SGEXIT01) from the segment.

(7) Assembly lines defining sequences follow the segment description. (Assuming a one segment program)

Segment Description Example No. 1

SEGMENT DESCRIPTION (Example No. 2)

This example illustrates how a Segment Description may be prepared in advance of the actual programming of sequences. It illustrates, in addition, the advantages inherent in modular type programming. Some of the advantages are the ability to speed up the programming job by outlining certain specifications for programming and assigning sequences to individual programmers.

As an example, assume that a lead programmer (or data processing manager) had a validation type of program to write in a relatively short period of time.

To use a simple example for illustrative purposes, assume that specific fields in a record must be validated.

Each record in the file has the format and symbolic tags assigned as follows:

ITEM	NO. OF CHARS.	SYMBOLIC TAG
ACCOUNT NUMBER	8	AACNO
ACCOUNT CODE	2	ACODE
CITY CODE	2	ACITY
AREA CODE	3	AAREA
BALANCE	7	ABAL

The type of validation that must be performed for each of the items is as follows:

ACCOUNT NO

The second and third digits must be "04"

ACCOUNT CODE

The first digit of the code must be a numeric value of 3 or less

CITY CODE

The code must have a value of 12 or higher

AREA CODE

The first digit of the area code must be 0

BALANCE

The balance must be an all numeric field

The lead programmer delegates the writing of sequences to individual programmers and specifies how each sequence is to be written. All processing sequences require moving the portion of the record to be validated to a work area within the sequence. The sequences to be written are as follows:

SEQUENCE NAME	PURPOSE OF SEQUENCE
INREC	FCP Descriptor Sequence for input records
OUTVAL	FCP Descriptor Sequence for output validated records

Segment Description Example No. 2

SEQUENCE NAME	PURPOSE OF SEQUENCE
OUTINV	FCP Descriptor Sequence for output invalid records
HOUSK	Housekeeping Sequence
RECALC	Sequence in which input/output areas are allocated and includes input and output routines
ACNVAL	Sequence in which the Acct. No. is validated
CODVAL	Sequence in which the Acct. Code is validated
CITVAL	Sequence in which the City Code field is validated
ARVAL	Sequence in which the Area Code field is validated
BALVAL	Sequence in which the Balance field is validated
ENDRUN	Sequence for end-of-run routines

SPECIFICATIONS

SEQ. NAME	PREFIX FOR ALL SYMBOLIC TAGS WITHIN THE SEQUENCE	ENTRY LOCATION	EXIT FROM SEQUENCE 01	EXIT FROM SEQUENCE 02
INREC	(N.A.)	(N.A.)	(N.A.)	(N.A.)
OUTVAL	(N.A.)	(N.A.)	(N.A.)	(N.A.)
OUTINV	(N.A.)	(N.A.)	(N.A.)	(N.A.)
HOUSK	H	START[1]	To Next SEQ.	(N.A.)
RECALC	A	INPUT[2]	To Next SEQ.	(N.A.)
		VALID[3]	(Internal linkage to AINPUT	
		INVAL[4]	after output of record)	
ACNVAL	B	STRT	Valid Field	Invalid Field
CODVAL	C	STRT	Valid Field	Invalid Field
CITVAL	D	STRT	Valid Field	Invalid Field
ARVAL	E	STRT	Valid Field	Invalid Field
BALVAL	F	STRT	Valid Field	Invalid Field
ENDRUN	G	STRT	Segment Exit	(N.A.)

Notes

1. Initial Entry Location
2. Entry location for the read of a record
3. Entry for output of a valid record
4. Entry for output of an invalid record

With these specifications the lead programmer could write the segment description as each programmer writes his sequence.

The entire programming of this problem will be included as the SAMPLE PROBLEM in this Assembly portion of the manual but only the Segment Description is included on the following page.

Segment Description Example No. 2

The segment description would be as follows:

LOCATION	OPERATION	SIZE	UNIT	ADDRESS
01	SGMT			HOUSK@HSTART
INREC	SEQ			
OUTVAL	SEQ			
OUTINV	SEQ			
HOUSK	SEQ			
	SEQX	01		RECALC@AINPUT
RECALC	SEQ			
	SEQX	01		ACNVAL@BSTRT
ACNVAL	SEQ			
	SEQX	01		CODVAL@CSTRT
	SEQX	02		RECALC@AINVAL
CODVAL	SEQ			
	SEQX	01		CITVAL@DSTRT
	SEQX	02		RECALC@AINVAL
CITVAL	SEQ			
	SEQX	01		ARVAL@ESTRT
	SEQX	02		RECALC@AINVAL
ARVAL	SEQ			
	SEQX	01		BALVAL@FSTRT
	SEQX	02		RECALC@AINVAL
BALVAL	SEQ			
	SEQX	01		RECALC@VALID
	SEQX	02		RECALC@AINVAL
ENDRUN	SEQ			
	SEQX	01		SGEXIT01

Segment Description Example No. 2

SECTION XVI

ASSEMBLY OPERATION CONTROLLING CODES

The purpose of this section is to describe the use of the Assembly controlling codes that either define the limits of the Assembly Operation or are used for output listing purposes.

The codes to be explained in this section and the general function of each is as follows:

START

This code informs the Assembler to begin assembly of a program and assign a given Program Identification Number.

NAME

This is an optional code that may be used to give an identifying name to the program for output listing purposes.

REMARK

This is an optional code that may be used to give descriptive information on the output listing. It does not generate any object coding but is valuable for documentation purposes.

CALL

This code may be used at a point in the program where it is desired to insert a source language routine which has been previously prepared.

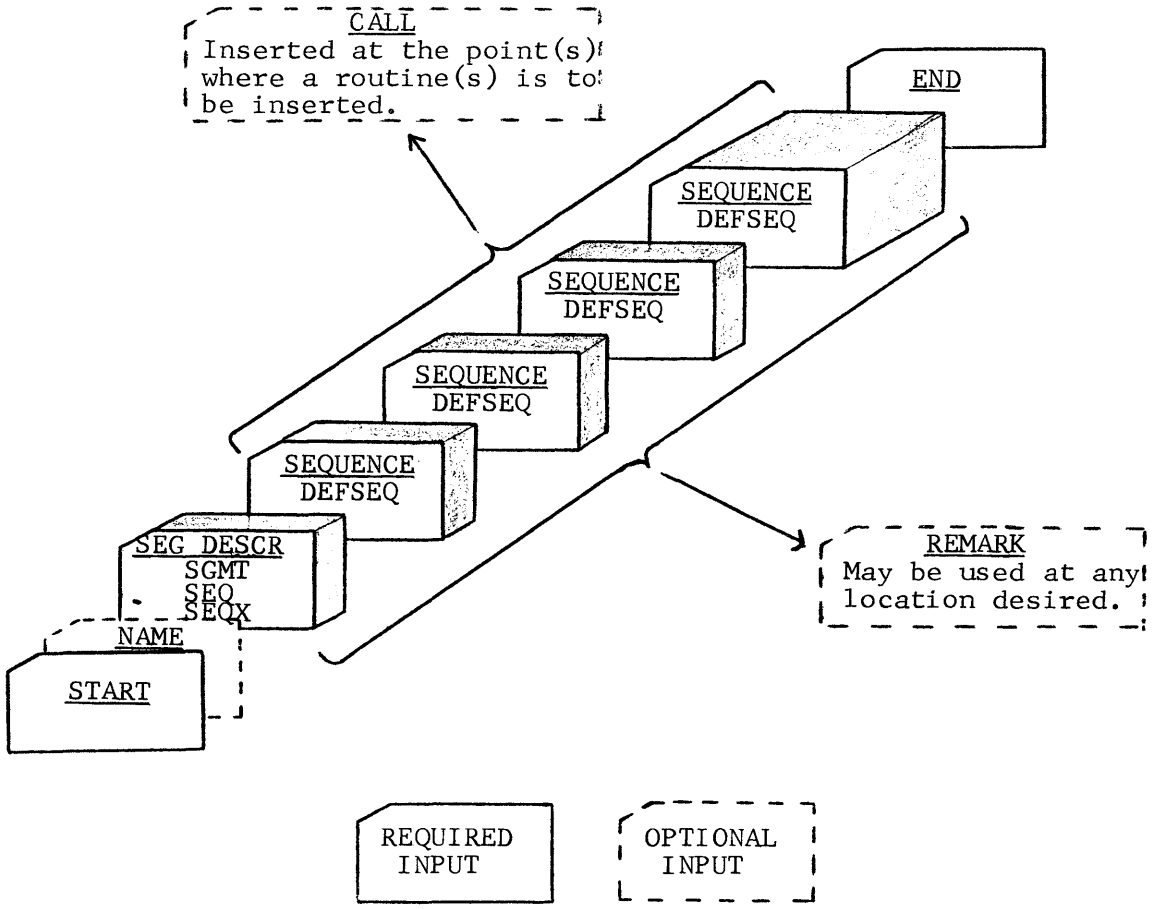
END

The END controlling code is used to inform the Assembler that the end of input for a program has been reached.

Input to the Assembly Operation requires that a START statement be the first statement and an END statement be the last statement introduced to the Assembly Operation.

The source language statements may be in any order only if (1) every source statement contains a numeric filled reference key that can be used for sorting the input and (2) the reference keys are in ascending sequence to conform with the diagram on the following page. (A one segment assembly is assumed.)

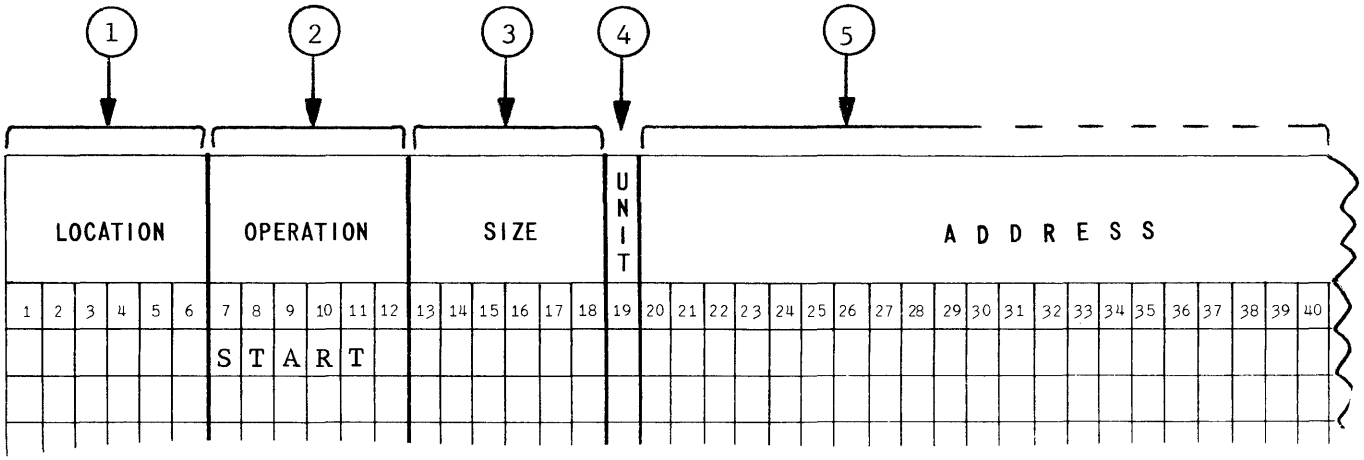
If reference keys are not present, the input cannot be sorted and the input will be assumed in the order as shown.



NOTE

If two or more segments are assembled at one time, the order of input is the same as shown above except that the Segment Descriptions are grouped together and must precede the individual sequences. The sequences may be in any order following the Segment Descriptions.

START Controlling Code (FORMAT)



(2) OPERATION

START must appear in this field.

(3) SIZE

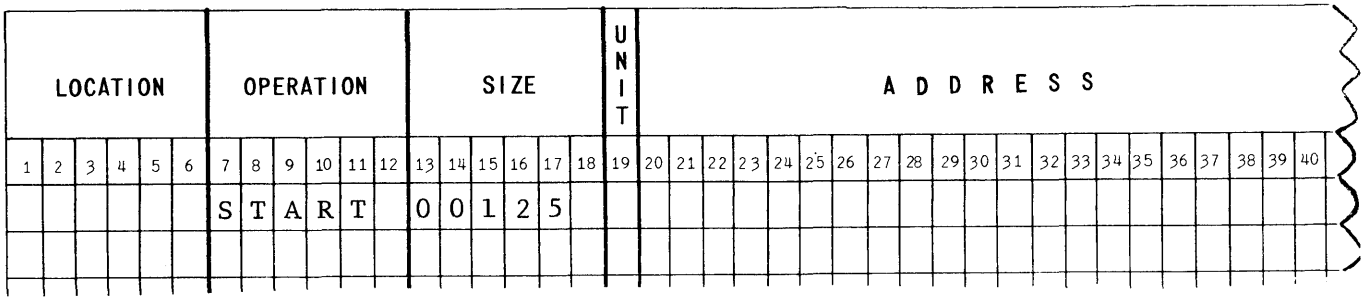
A Program Identification Number must be entered (left-justified) in this field. This must be a unique five digit decimal number for each program.

(5) ADDRESS

Not used.

EXAMPLE

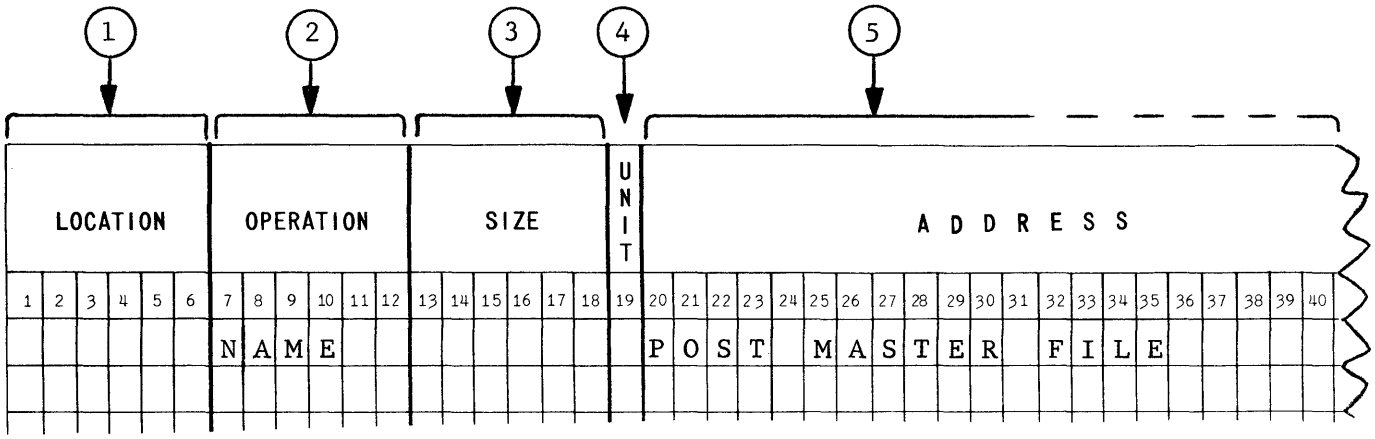
The following is an example of a START line for a program with an Identification Number of 00125.



START
Format

NAME Controlling Code (FORMAT)

The NAME controlling code may be used if it is desired to furnish a name to the program on output listings.



FORMAT NOTES

(2) OPERATION

NAME must appear if this line is used.

(5) ADDRESS

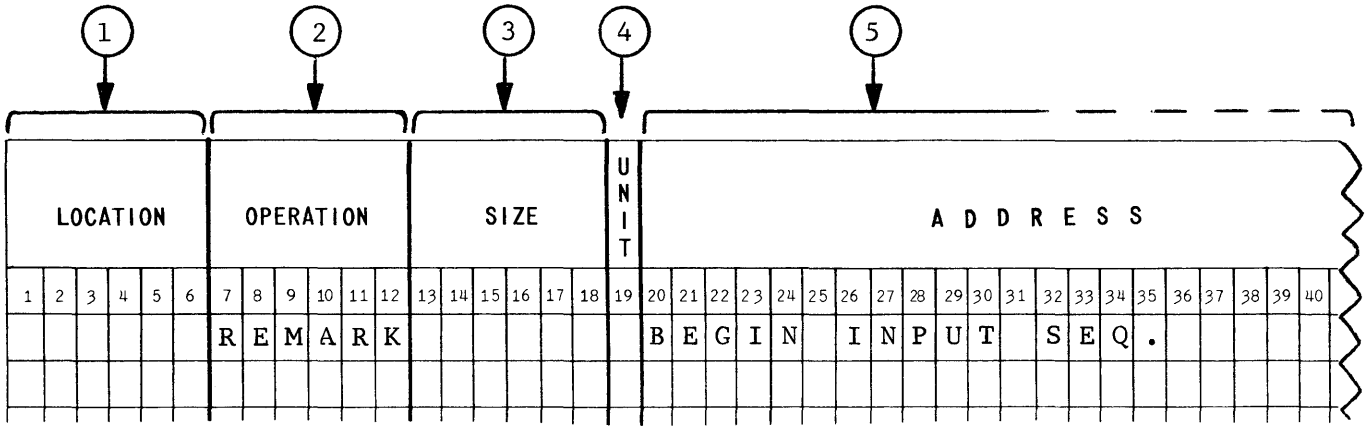
A left-justified entry of 30 characters maximum length will provide a program name on the output listings.

NOTE that if this statement is used it must follow the START statement. If not used, the Program Identification Number of the START line will be used to identify the program on the output listings.

NAME
Format

REMARK Controlling Code (FORMAT)

The REMARK code may be used at any place where a programmer's comments are desired on the output listing. This code does not affect the object program.



FORMAT NOTES

(2) OPERATION

REMARK must appear for each Remark line.

(5) ADDRESS

This field will contain a comment that will appear on the output listing. Any 3301 character may be used except the ISS (•) which may not appear if the input is from paper tape.

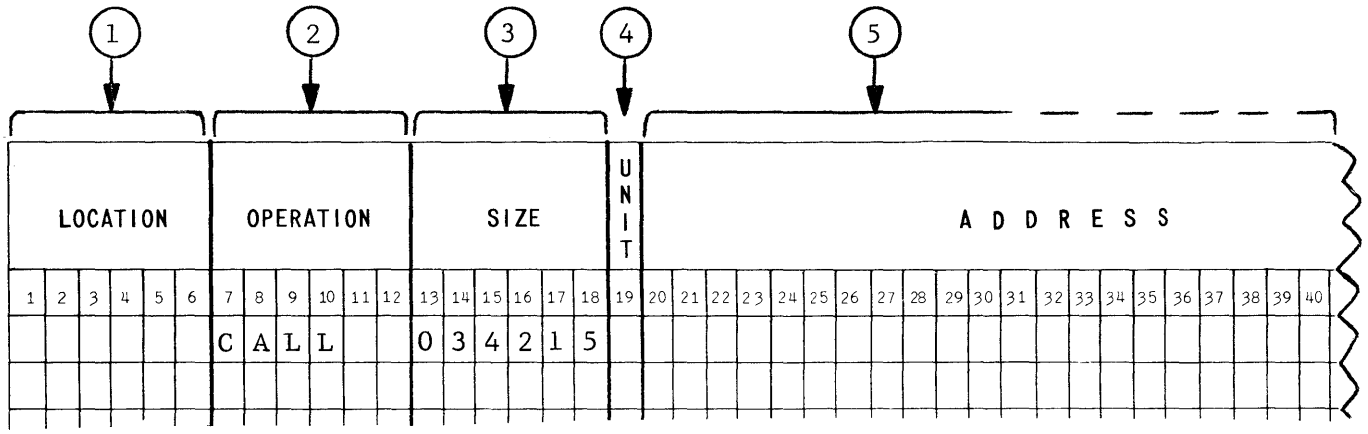
Note that the REMARK code may be used as often as required and appear at any point after the NAME line and before the END line.

Note also that remarks may be made in unused portions of the ADDRESS Field of other assembly lines.

REMARK Format

CALL Controlling Code (FORMAT)

The CALL code may be used where it is desired to enter a source language routine as a part of the assembled program. The routine is identified by a six character decimal name and is stored on a source language library tape.



FORMAT NOTES

(2) OPERATION

CALL must appear as shown above.

(3) SIZE

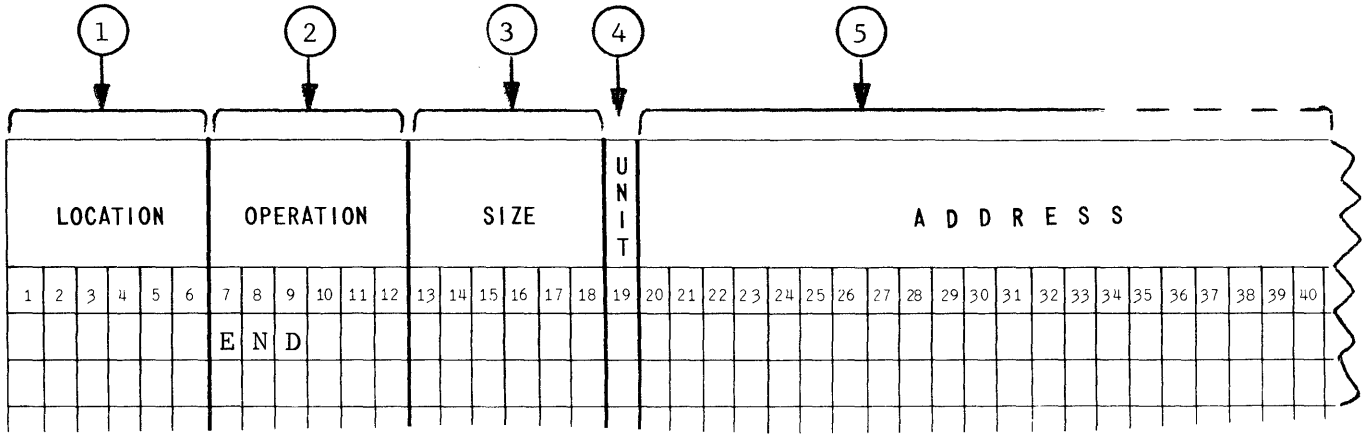
The six character routine identifying name must be decimal numeric.

The routine which is "CALLed" into the program is contained on a magnetic tape and is in source (assembly) language. The routine is entered into the program at the point where the "CALL" line is encountered.

CALL
Format

END Controlling Code (FORMAT)

The END controlling code specifies the last line of a program and is the required entry to terminate input to the Assembly Operation.



FORMAT NOTES

(2) OPERATION

END is the only required entry for this line.

END
Format

SECTION XVII

CORRECTION PROCEDURES

Corrections may be made to source input statements only if Reference Keys have been assigned to the source input.

The corrections should be entered in ascending sequence by Reference Keys.

The three correction controlling codes are:

STARTC

This code notifies the Assembler that the statements that follow are corrections to be applied to a designated program.

DELETE

This code is used to delete one or a continuous series of statements in a program.

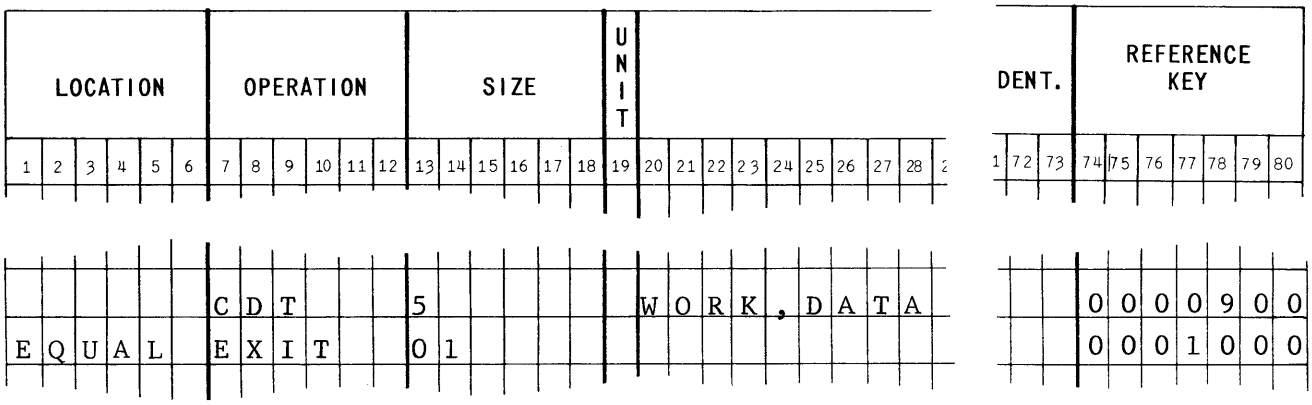
ENDC

This code specifies the end of the corrections to the program as designated in the STARTC statement.

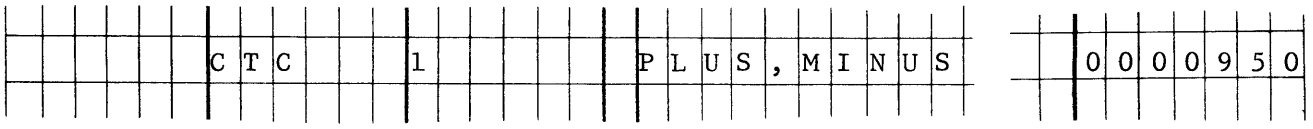
Insertions and/or replacements may also be made to the source input.

To insert a statement, it is prepared in the desired format and assigned a Reference Key value which will place it in the location desired.

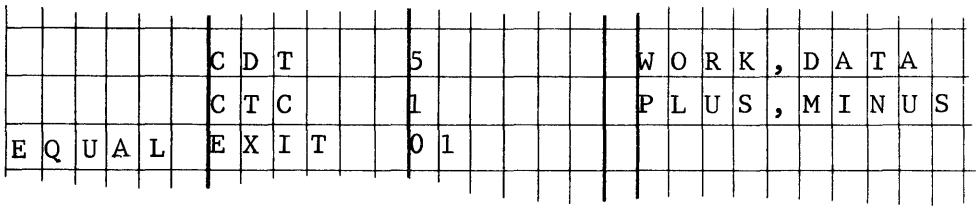
As an example, assume the source program statements appear as follows before corrections have been applied:

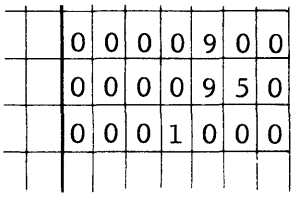


And a statement such as the following appeared in the corrections:



Then the corrected output would be as follows:

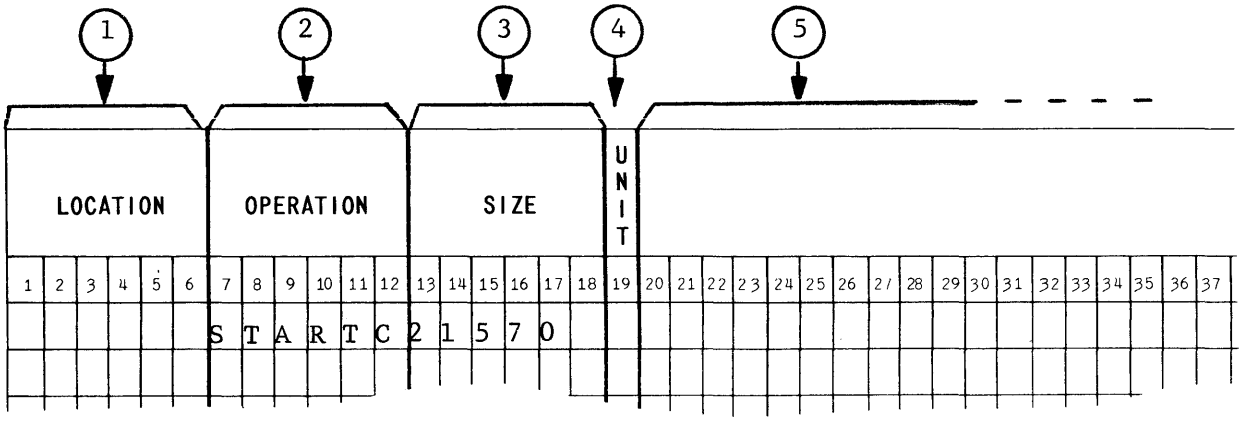




To replace a statement, it is only necessary to write a statement in the corrected form desired and assign the same Reference Key. As corrections are applied, statements in the source program are replaced by statements in the Corrections having identical Reference Keys.

STARTC (Format)

(This code signifies to the Assembler that the statements that follow are correction statements to be applied to the designated program.)



FORMAT NOTES

(1) LOCATION

Not used.

(2) OPERATION

STARTC must appear in this field.

(3) SIZE

The Identification Number of the program to be corrected is in this field. This entry will be identical to the Program Identification Number on the START statement of the program being corrected.

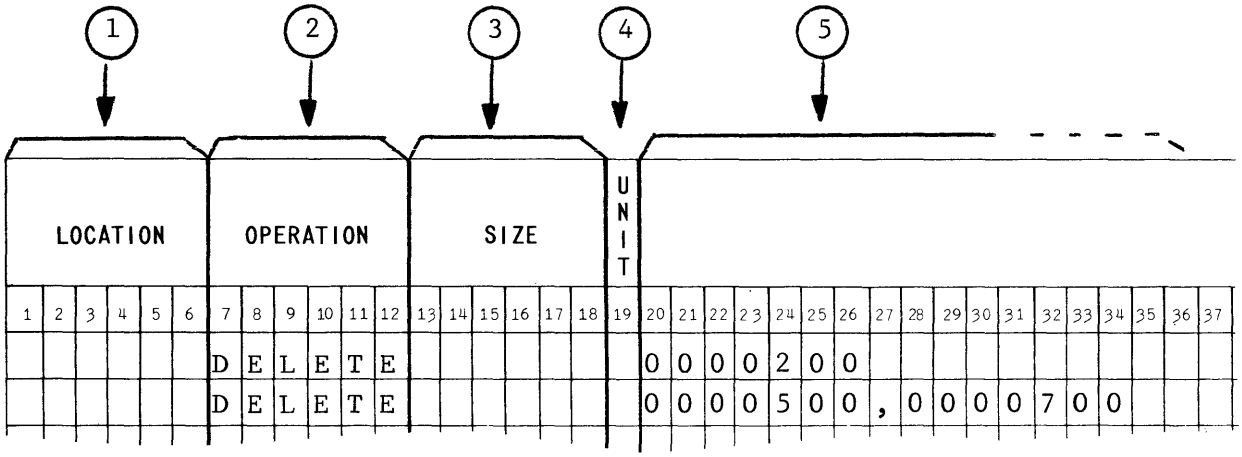
(4) (5) UNIT and ADDRESS

Not used.

STARTC Format

DELETE (Format)

(This code is used to delete one or a series of continuous statements of the source program.)



FORMAT NOTES

(1) LOCATION

Not used.

(2) OPERATION

DELETE must appear in this field

(3) (4) SIZE and UNIT

Not used.

(5) ADDRESS

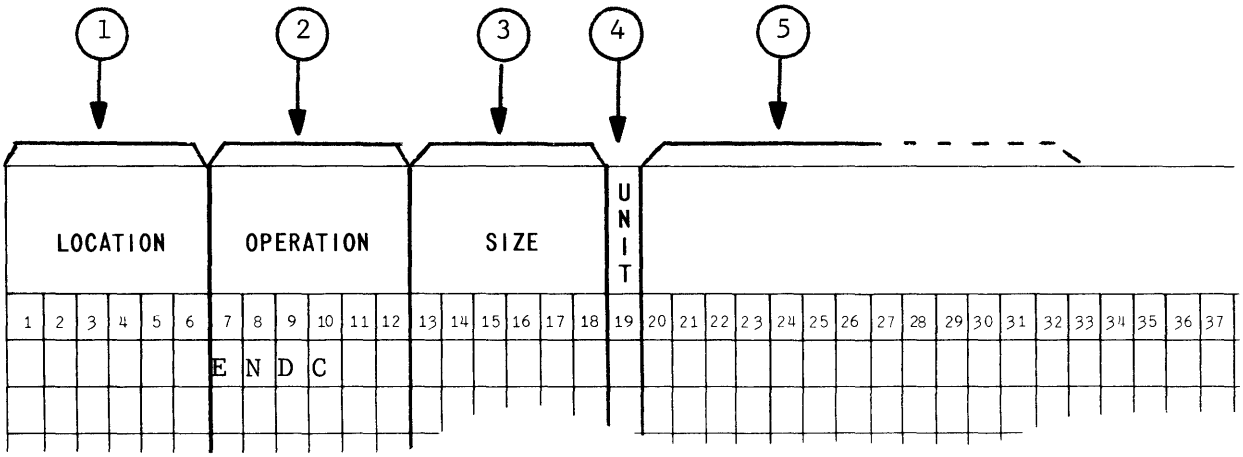
The ADDRESS Field contains the Reference Key(s) of the statement(s) to be deleted from the source program. One entry as above (0000200) will delete the source statement having the same Reference Key.

An ADDRESS Field having two entries, as on the second line above, will cause the deletion of all source statements having Reference Keys between and including those shown.

DELETE Format

ENDC (Format)

(This code signifies the end of the corrections within the program specified by the STARTC statement)



FORMAT NOTES

(2) OPERATION

ENDC must appear. This is the only field that is used for this statement.

(1) LOCATION
(3) SIZE
(4) UNIT
(5) ADDRESS

NOT USED

ENDC Format

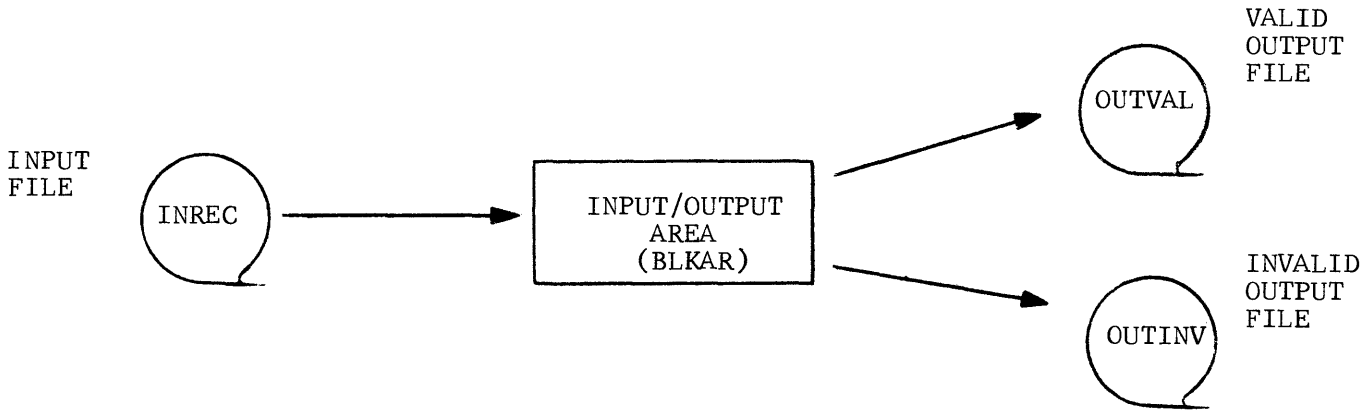
SECTION XVIII

SAMPLE PROBLEM

This Sample Problem is based on Example 2 of Section XV, Preparation of the Segment Description; an explanation of the problem appears there.

This problem has been simplified and is for illustration of the modular concept and the organization and structure of input for the assembly operation.

The input and output tapes are unbatched and processing is done in a single block area as illustrated.



RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	U N I T	ADDRESS	IDENT.	REFERENCE KEY
	START	00001				00001000
	NAME			SAMPLE SIMPLE VALIDATION RUN		00002000
01	SGMT			HOUSK@HSTART THE SEGMENT DESCRIPTION		00003000
INREC	SEQ			INPUT SEQUENCE		00004000
OUTVALSEQ				OUTPUT VALID RECORD SEQUENCE		00005000
OUTINVSEQ				OUTPUT INVALID RECORD SEQUENCE		00006000
HOUSK	SEQ			HOUSEKEEPING SEQUENCE		00007000
	SEQX	01		RECALC@AINPUT		00008000
RECALC	SEQ			INPUT/OUTPUT SEQUENCE		00009000
	SEQX	01		ACNVAL@BSTRT		00010000
	SEQX	02		ENDRUN@GSTRT		00011000

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
ACNVAL	SEQ			VALIDATE ACCT.NO.SEQUENCE		000120000
	SEQX	01		CODVAL@CSTRT		000130000
	SEQX	02		RECALC@AINVAL		000140000
CODVAL	SEQ			VALIDATE ACCT.CODE SEQUENCE		000150000
	SEQX	01		C1TVAL@DSTRT		000160000
	SEQX	02		RECALC@AINVAL		000170000
C1TVAL	SEQ			VALIDATE CITY CODE SEQUENCE		000180000
	SEQX	01		ARVAL@ESTRT		000190000
	SEQX	02		RECALC@AINVAL		000200000
ARVAL	SEQ			VALIDATE AREA CODE SEQUENCE		000210000
	SEQX	01		BALVAL@FSTRT		000220000
	SEQX	02		RECALC@AINVAL		000230000
BALUAL	SEQ			VALIDATE BALANCE SEQUENCE		000240000
	SEQX	01		RECALC@AVALID		000250000

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
	SEQX	02		RECALC@AINVAL		000260
ENDRUN	SEQ			END OF RUN SEQUENCE		000270
	SEQX	01		SGEXIT01		000280
INREC	DEFSEQ			F,N,A INPUT FCP DESCRIPTOR SEQUENCE		000290
USEIN						000300
BACKUP						000310
IDENT				MASTFILE		000320
USEBEG						000330
USEEND						000340
ACTIVE						000350
LABELS				S,S		000360
BLKAR				A		000370
ALTAR						000380
RECAR						000390
VARRHE						000400
OUTLHE						000410
OUTRHE						000420
BATCH			F	0001		000430
RECORD		A				000440
ERROR						000445
OUTVAL	DEFSEQ			F,N,A OUTPUT (VALID) FCP DESCR. SEQUENCE		000450
USEIN						000460

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

PAGE NO. 4 OF 13

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
BACKUP						000047 00
IDENT				MASTFILE		000048 00
USEBEG						000049 00
USEEND						000050 00
ACTIVE				030		000051 00
LABELS				S,S		000052 00
BLKAR				A		000053 00
ALTAR						000054 00
RECAR						000055 00
VARRHE						000056 00
OUTLHE				ADRLHE		000057 00
OUTRHE						000058 00
BATCH				F0001		000059 00
RECORD		A				000060 00
ERROR						000060 50
OUTINV	DEFSEQ			F,N,A OUTPUT (INVAL) FCP DESCR. SEQUENCE		000061 00
USEIN						000062 00
BACKUP						000063 00
IDENT				ERRORFIL		000064 00
USEBEG						000065 00
USEEND						000066 00
ACTIVE				030		000067 00

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
LABELS				S,S		0006800
BLKAR				A		0006900
ALTAR						0007000
RECAR						0007100
VARRHE						0007200
OUTLHE				ADRLHE		0007300
OUTRHE						0007400
BATCH				F0001		0007500
RECORD		A				0007600
ERROR						0007650
HOUSK	DEFSEQ					0007700
	REMARK			BEGIN HOUSEKEEPING SEQUENCE		0007800

III-196

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
HSTART	OPEN		FINREC	OPEN INPUT FILE		00079000
	OPEN		OOUTVAL	OPEN OUTPUT VALID FILE		00080000
	OPEN		OOUTINV	OPEN OUTPUT INVALID FILE		00081000
	EXIT	01		EXIT FROM HOUSEKEEPING		00082000
RECALC	DEFSEQ					00083000
	REMARK			BEGIN I/O & AREA ALLOC. SEQUENCE		00084000

REMARKS:

RCA 3301

PROGRAM SAMPLE PROGRAM

PROGRAMMER

REALCOM ASSEMBLY
PROGRAM SHEET

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
A	ALOC			INPUT AREA		000850
ACNO		8		ACCOUNT NUMBER		000860
CODE		2		ACCOUNT CODE		000870
CITY		2		CITY CODE		000880
AREA		3		AREA CODE		000890
ABAL		7		BALANCE		000900
	ALOC					000905
ADRLHE	ADRCON			AACNO STORED OUTPUT LHE ADDRESS		000910
AINPUT	READ			INREC,EOF		000920
	EXIT	01		EXIT AFTER INPUT		000930
AVALID	WRITE			OUTVAL WRITE VALID RECORD		000940
	UTC			AINPUT		000950
AINVAL	WRITE			OUTINV WRITE INVALID RECORD		000960
	UTC			AINPUT		000970
EOF	EXIT	02		END OF FILE EXIT		000980
	REMARK			BEGIN ACCT.NO. VALIDATION SEQUENCE		000990

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
ACNVALD	DEFSEQ					0010000
B	ALOC					0010100
WACT		2				0010200
FOUR	FIXNUM	2		4		0010300
BSTRT	TCL	2		AACNO+$1,BWACT 2ND & 3RD CHARS. TO WORK AREA		0010400
	CDT	2		BWACT,BFOUR COMPARE TO 04		0010500
	CTC	1		BINVAL,BINVAL A&B EQUAL INVALID ACCT. NO.		0010600
BVAL	EXIT	01		EXIT FOR VALID ACCT.NO.		0010700
BINVAL	EXIT	02		EXIT FOR INVALID ACCT.NO.		0010800

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
CODVAL	DEFSEQ			BEGIN ACCT. CODE VALID. SEQUENCE		001090
C	ALOC					001100
WACOD		1				001110
VALUE	FIXNUM	1		3		001120
CSTRT	TCL	1		ACODE,CWACOD 1ST CHAR. TO WORK AREA		001130
	CDT	1		CWACOD,CVALUE COMPARE TO 3		001140
	CTC	1		CINVAL,CINVAL A&B EQUAL INVALID ACCT.CODE		001150
CVAL	EXIT	01		EXIT FOR VALID ACCT.CODE		001160
CINVAL	EXIT	02		EXIT FOR INVALID ACCT.CODE		001170

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
CITVAL	DEFSEQ			BEGIN CITY CODE VALID. SEQUENCE		001180
D	ALOC					001190
WACIT		2				001200
TWELV	FIXNUM	2		12		001210
DSTRT	TCL	2		ACITY,DWACIT	CITY CODE TO WORK AREA	001220
	CDT	2		DWACIT,DTWELV	COMPARE TO 12	001230
	CTC	1		DINVAL,DVAL	A ADDR. EQUALS INVALID CODE	001240
DVAL	EXIT	01			EXIT FOR VALID CITY CODE	001250
DINVAL	EXIT	02			EXIT FOR INVALID CITY CODE	001260

28-00-109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
ARVAL	DEFSEQ			BEGIN AREA CODE VALID. SEQUENCE		00127000
E	ALOC					00128000
WAREA		1				00129000
ZERAR	FIXNUM	1		0		00130000
ESTRT	TCL	1		AAREA,EWAREA 1ST CHAR. TO WORK AREA		00131000
	CDT	1		EWAREA,EZERAR COMPARE TO ZERO		00132000
	CTC	1		EINVAL,EINVAL A&B EQUAL INVALID AREA CODE		00133000
EVAL	EXIT	01		EXIT FOR VALID AREA CODE		00134000
EINVAL	EXIT	02		EXIT FOR INVALID AREA CODE		00135000

28.00.109

REMARKS:

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
BALVAL	DEFSEQ			BEGIN BALANCE VALIDATION SEQUENCE		001360
F	ALOC					001370
TABLE	FIXNUM	8	H	11111111 TABLE FOR TRANSLATION		001380
	FIXNUM	2		00		001390
	FIXNUM	2		11		001400
	FIXNUM	50		00000000000000000000000000000000000000000000000000		001410
	FIXNUM	16		0000000000000000		001420
WBAL		7				001430
FSTRT	TCL	7		AABAL,FWBAL BALANCE TO WORK AREA		001440
	TBT	7		FWBAL,FTABLE TRANSLATE WORK AREA		001450
	LAL	1		FWBAL,FWBAL VALID BAL FIELD EQUALS ONES		001460
	CTC	1		FINVAL,FINVAL A&B EQUAL IVALID BAL. FIELD		001470
FVAL	EXIT	01		EXIT FOR VALID BALANCE FIELD		001480
FINVAL	EXIT	02		EXIT FOR INVALID BALANCE FIELD		001490

REMARKS:

28-00-109

RCA 3301

REALCOM ASSEMBLY

PROGRAM SHEET

PROGRAM SAMPLE PROGRAM

PROGRAMMER

DATE

LOCATION	OPERATION	SIZE	UNIT	ADDRESS	IDENT.	REFERENCE KEY
ENDRUN	DEFSEQ			BEGIN END OF FILE SEQUENCE		001500
GSTRT	CLOSE			INREC CLOSE INPUT		001510
	CLOSE			OUTVAL CLOSE OUTPUT VALID		001520
	CLOSE			OUTINV CLOSE OUTPUT INVALID		001530
	EXIT	01				001540
	END					001550

28-00-109

REMARKS:

APPENDIX A

DEVICE CONTROLLING CODES

DEVICE CONTROL (Except Console Typewriter)

The Device Controlling Codes provide the user with the ability to control devices by issuing commands calling for such physical operations as "reading," "writing," and "rewinding."

FCP will retain control, however, over the scheduling of devices, control modules, and Simultaneous modes.

The actual command to a device (such as for reading, writing and rewinding) must be preceded by an ISSUE line of assembly coding.

Before another command may be given to the same device a FREEDV line of assembly coding must be given. The FREEDV line directs FCP to retain control until the previously ISSUEd command to the specified device has been completed.

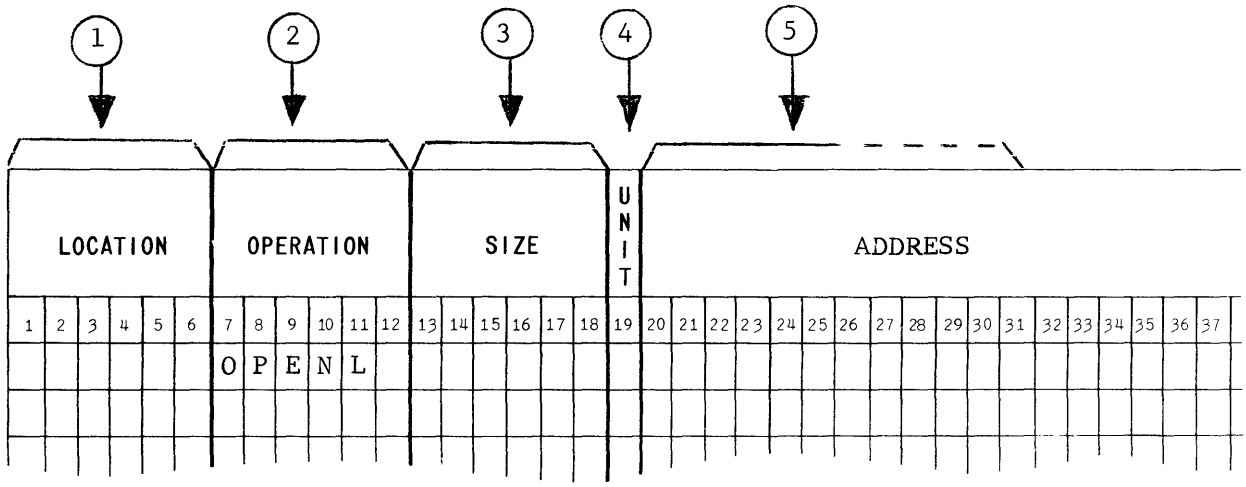
The lines of coding that are required, therefore, are an ISSUE line, the command line, and a FREEDV line.

Other lines that may appear are the TESTDV line and the SWAPDV line. The TESTDV line is used to test the physical status of a device. The SWAPDV line is used to switch devices for a magnetic tape file.

The user must prepare a device sequence with the Device Region entries required. (See Example 1.)

If the user, in addition to Device Control, wants FCP to perform its standard label functions, he must prepare a sequence with both the Device and Label Regions present. With this option he may use the Device Controlling Codes OPENL (Open Label) and CLOSEL (Close Label) for beginning and ending label functions respectively.

DEVICE CONTROL (OPENL Line)



FORMAT

NOTES

(1) LOCATION

A symbolic tag may appear in this field if desired.

(2) OPERATION

OPENL must appear.

(3) SIZE

Not used.

(4) UNIT

Used to describe the usage of the device where:

F = Forward input processing.
R = Reverse input processing.
O = Output processing.

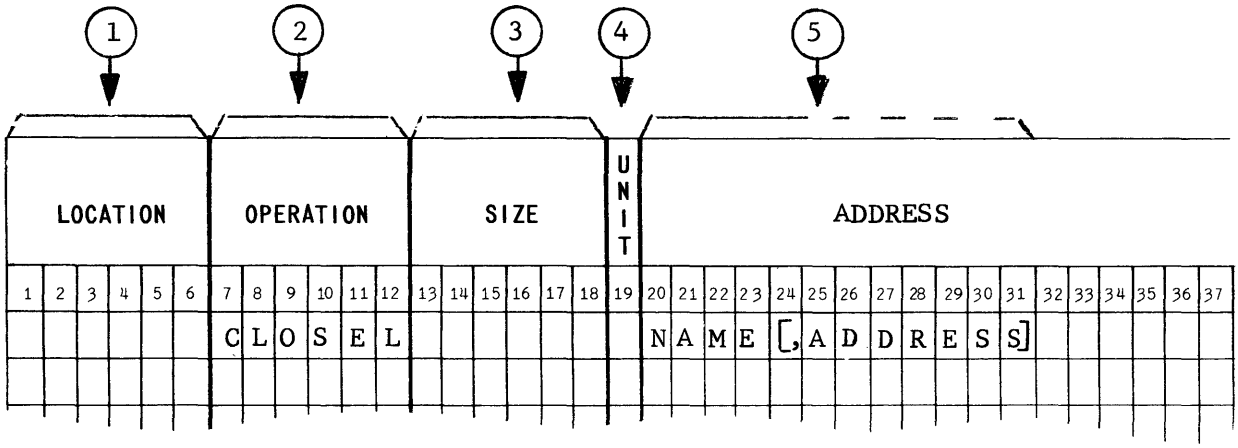
(5) ADDRESS

Name of the device sequence to be opened.

NOTE: The OPENL function is analagous to the OPEN function of file processing.

OPENL Format

DEVICE CONTROL (CLOSEL Line)



FORMAT NOTES

(1) LOCATION

A symbolic tag may appear in this field if desired.

(2) OPERATION

CLOSEL must appear.

(3) SIZE

Not used.

(4) UNIT

For an Output file:

I = Intermediate reel to be closed.
F = Final reel to be closed.

(5) ADDRESS

First entry is the name of the device sequence to be closed.

Second optional entry may specify the end of file control address for an input file.

CLOSEL Format

NOTE:

INPUT

A CLOSEL, issued to a device sequence which issued an OPENL signifying input processing, provides the ability to specify end of file control, as in the READ macro for file processing. FCP will perform end label processing as defined in the LABELS entry of the users sequence.

If FCP sensed the end of file sentinel, it will not rewind the device, will not swap devices, and will transfer control to the specified location.

If end of file is not sensed, FCP will rewind the current reel, swap devices, perform beginning label checking for the new reel, and position the tape prior to first data block.

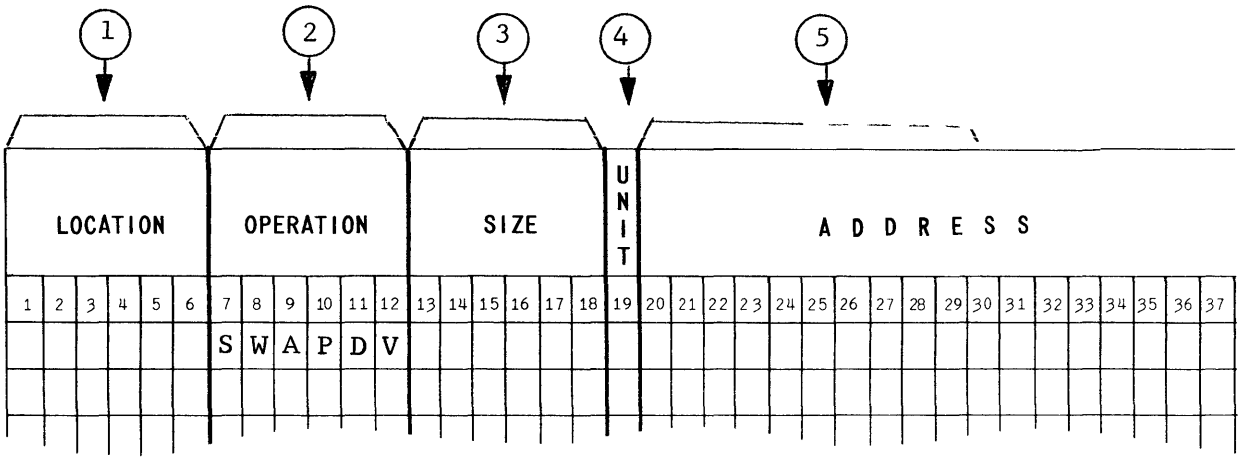
OUTPUT

A CLOSEL, issued to a sequence which issued an OPENL signifying output processing, provides the ability to indicate to the FCP whether to create an end of reel or an end of file indication in the sentinel format.

If end of reel is signified, FCP will perform end label processing as specified in the LABELS entry of the sequence, create end of reel sentinel format, rewind the device, swap devices, and perform beginning label processing on the new reel. Rerun procedures will be performed if indicated in the device assignment for this sequence.

If end of file is signified, FCP will perform end label processing as specified in the LABELS entry of the sequence, create end of file sentinel format, and return control to the user. Note that the device is neither rewound nor swapped. If the user desires the first device to be the current device in the region, a SWAPDV must be issued.

DEVICE CONTROL (SWAPDV Line)



FORMAT NOTES

(1) LOCATION

A symbolic tag may appear in this field if desired.

(2) OPERATION

SWAPDV must appear.

(3) SIZE

Not used.

(4) UNIT

Not used.

(5) ADDRESS

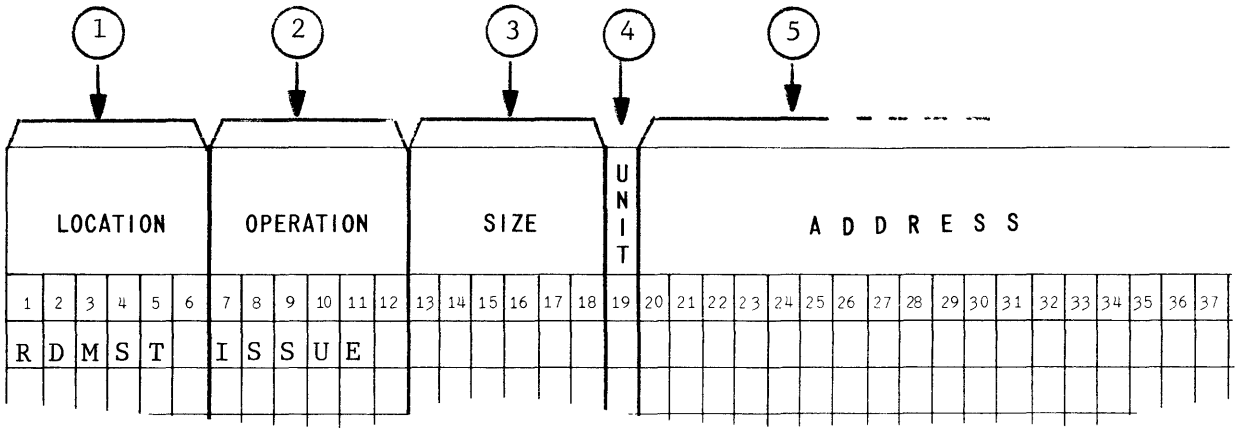
Name of the device sequence for which a device swap is to be performed.

NOTE: For users processing multi-reel files.

FCP makes the next device the current one to be processed. Reverts back to initial device when the multiple devices assigned have been exhausted.

SWAPDV Format

DEVICE CONTROL (ISSUE line)



FORMAT NOTES

(1) LOCATION

A symbolic tag may appear in this field if desired.

(2) OPERATION

ISSUE must appear.

(3) SIZE

See following page.

(4) UNIT

Not used.

SIZE

This field is used for control information only when device interchangeability has been specified in the Device Sequence.

The characters that should appear in each column are as follows:

COL.	CHAR.	
13	0	Not defined at present
14	0	Not defined at present
15	0	Not defined at present
16	Device Code	If a device type code (see table below) is placed in this location and another device is assigned, the following

ISSUE Format

SIZE (Cont'd)

COL.	CHAR.	
		command line will not be executed and control will be transferred to the instruction following the command line.
16	0	If a zero is entered, the following command line will always be performed.
17	1	This character is used to specify the appending of printer control information to the right end of output data when magnetic tape is substituted for the printer. The program must allocate the additional four character area following the output area if the option is used and if the maximum record exceeds 116 (or 156) characters. This control information is for the Tape To Printer service routine.
17	0	No function performed.
18	0	Card punch translate mode
18	1	Card punch binary mode

DEVICE CODES

CODE	DEVICE	CODE	DEVICE
1	581 Magnetic tape station	E	Paper Tape Punch
2	582 Magnetic tape station	F	Interrogating Typewriter
6	681 Magnetic tape station	G	Data Disc File
8	3485 Magnetic tape station	H	3488
#	Any Magnetic tape station	J	Communication Control
A	Printer	K	CMC
B	Card Reader	L	DXC
C	Card Punch	Z	Console Typewriter
D	Paper Tape Reader	Ø	Any Device

(5) ADDRESS

This field will contain the name of the Device Sequence as the first entry. This first entry may be $0 if the sequence name has been placed in INDEX 1. The second entry in this field is an optional four character entry and must appear (separated from the first entry by a comma) if the following command line may address the printer as one of the possible devices. (See Note below.)

This four character entry, following a comma, is written as follows:

ISSUE Format

1st CHAR.

1 Advance paper number of lines specified by 3rd character

2 Advance paper by tape loop

4 Page change by tape loop

2nd CHAR.

0 Asynchronous Mode Printing

1 No printing

2 Synchronous Mode Printing

3rd CHAR.

If the 1st character is 1, this specifies the number of lines to advance paper. The numbers 10 to 15 must be expressed as SP, #, @, (,), e, respectively.

4th CHAR.

0 No high speed memory to buffer transfer

1 Print 120 characters

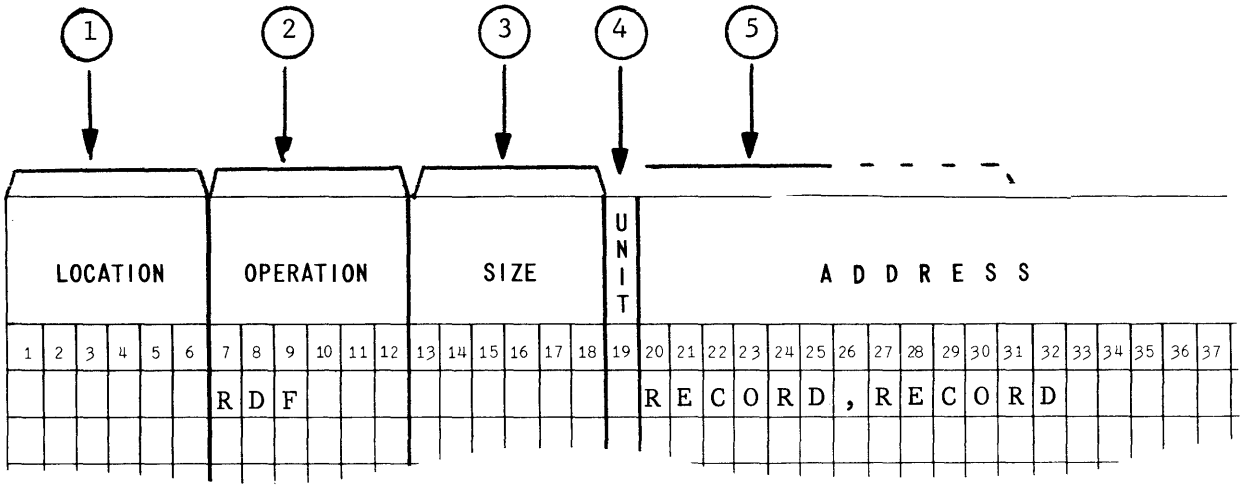
2 Print 160 characters

4 Transfer 64 contiguous print table characters to the print table portion of the buffer.

NOTE: The second entry (printer control information) is used when interchangeability is specified in the Device Sequence and the printer is one of the possible devices. If device interchangeability is not specified and the printer is specified as the device to be designated, this control information will appear as the second entry of the command line.

DEVICE CONTROL (Command line)

(This line of coding must follow the ISSUE of command line.)



FORMAT NOTES

(1) LOCATION

A symbolic name may be entered in this if desired.

(2) OPERATION

One of the following entries may appear:

RDF	(Read Device Forward)
RDR	(Read Device Reverse)
WRT	(Write to Device)
RWD	(Rewind Magnetic Tape)
ERS	(Erase Magnetic Tape)
CDV	(Control Device)

(3) (4) SIZE AND UNIT

Not used.

(5) ADDRESS

A and B addresses are required in the ADDRESS field of the command line for the following specified devices. (See NOTE for entries where interchangeability of devices may be specified.)

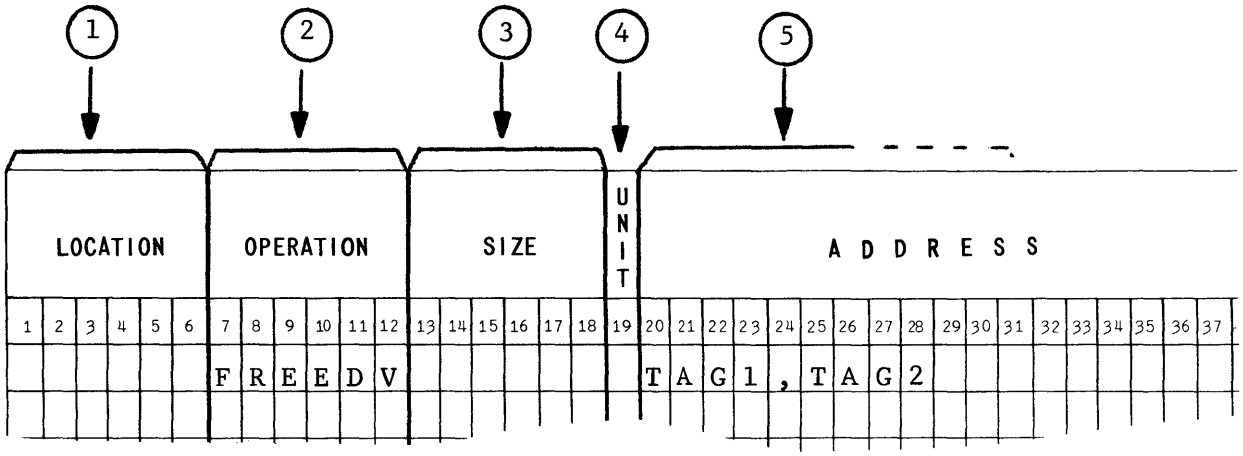
DEV. CONTROL COMMAND LINE Format

DEVICE	A ADDRESS	
Magnetic Tape (RDF-WRT)	Tag of I/0 Area (The left-end and right-end addresses will be assigned and assembled for the A and B addresses respectively.)	Tag of I/0 Area
Magnetic Tape (RDR)	Tag of I/0 Area (The right-end and left-end addresses will be assembled for the A and B addresses respectively.)	Tag of I/0 Area
Magnetic Tape (RWD)	$0	$1 (for Rewind to BTC) $2 (for Rewind to Load-681 only) $4 (for Rewind of 1 gap)
Card Reader (RDF)	Tag of Input Area (The input area must be diad oriented. A left-end address is assembled.)	$0000
Card Punch (WRT)	Tag of Output Area The output area must be diad oriened. A left-end address is assembled.)	$0 (for translate mode) $1 (for binary mode)
Paper Tape Punch (WRT)	Tag of Output Area (Left-end address assembled.)	Tag of Output Area (Right-end address assembled.)
Printer (WRT)	Tag of Output Area Area must be diad oriented and left-end address will be assembled.	&abcd (where abed represent the printer control information as explained for the ISSUE line)

NOTE: Where interchangeability of devices is specified, the A and B address entries are written as specified for magnetic tape (RDF-WRT). Necessary control information for printer or card punch output are entered in the ISSUE lines in the 2nd ADDRESS and SIZE entries respectively.

DEV. CONTROL COMMAND LINE Format

DEVICE CONTROL (FREEDV Line)



FORMAT NOTES

(1) LOCATION

A symbolic name may appear in this field if desired.

(2) OPERATION

FREEDV must appear.

(3) (4) SIZE and UNIT

Not used.

(5) ADDRESS

This field will contain the name of the Device Sequence as the first entry (TAG 1). This first entry may be $0 if the sequence name has been placed in INDEX 1.

The second entry (TAG 2) is the location to which control will be transferred if an abnormal or non-recoverable termination occurs such as:

a. E/F or E/D

b. A/B equality and no gap.

c. Unrecoverable read parity.

d. Designated device not present.

(For determining the reason of the termination see TESTDV code.)

If this second entry is $0 the address of the instruction following the FREEDV line will be generated.

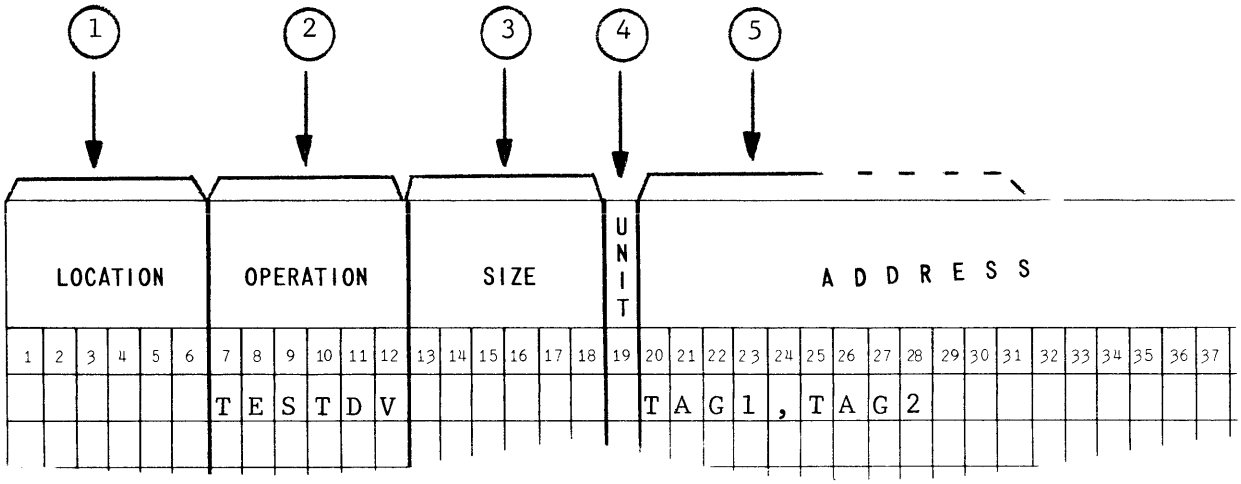
FREEDV Format

NOTE: Following the FREEDV code the address of the location to the right of the last character read will be stored in MASTER + $16 to MASTER + $19. (Where master = device sequence name).

(For reading in a reverse direction the address stored would be that of the location one to the left of the last character read.)

FREEDV
Format

DEVICE CONTROL (TESTDV line)



FORMAT NOTES

(1) LOCATION

A symbolic name may be entered in this field if desired.

(2) OPERATION

TESTDV must appear.

(3) SIZE

This entry specifies the test(s) to be performed. A table of the codes used is on a following page.

(4) UNIT

Not used.

(5) ADDRESS

The name of the Device Sequence is the first entry (TAG 1) in this field. This first entry may be $0 if the sequence name has been placed and is present in INDEX 1.

The second entry (TAG 2) is the location to which control will be transferred if any of the tested conditions are present.

TESTDV
Format

SIZE

This field specifies which tests are to be performed. The tests are performed based on a 1 or 0 bit present in a character. Only the character is shown for performing a specific test.

COL.	CHAR.	FUNCTION
13	%	Used to test more than one condition. Additional tests performed at the location specified in the second entry of the ADDRESS field.
14	-	Has the ISSUED command terminated normally?
14	&	Has the ISSUED command terminated abnormally?
14	4	Has the ISSUED command been initiated?
14	2	Has the ISSUED command been queued?
15 } 16 }		See following table for proper column and character for specific test(s).
17	Device Code	If a Device Code (see SIZE entry of ISSUE format) is specified in this location the TESTDV function will be performed only if that device has been assigned.
17	0	TESTDV will always be performed.
18	1	Ignore if a CDV or RWD command.
18	2	Ignore if a RDF command.
18	4	Ignore if a RDR command.
18	8	Ignore if a WRT command.
18	&	Ignore if a ERS command.

TESTDV Format

TABLE OF TESTS

COLS. 15-16 of SIZE field

COL.	CHAR.	Test or Command
		MAGNETIC TAPE
15	1	Is the device inoperable?
15	2	Is the tape in motion?
15	4	Has ETW been sensed?
15	8	Is the tape at BTC?
15	&	Is the tape moving in reverse?
15	-	Is splice detected?
16	1	Is there a Parity Error on read or write (PE)?
16	2	Is there a Magnetic Tape Alarm (MTA)?
16	4	Has A-B equality and no gap been sensed?
16	8	Is the E/F-E/D indicator set?
		CARD READER
15	1	Is the device inoperable?
15	2	Is the device operating (busy)?
15	4	Is there a Photo-Diode Failure (PDF)?
15	8	Is there a Multi-Punch Error (MPE)?
16	8	Is the E/F indicator set?
		CARD PUNCH
15	1	Is the device inoperable?
15	2	Is the device operating (busy)?
15	4	Is there a Punch Compare Error (PCE)?
15	8	Is there a Parity Error (PE)?
		PAPER TAPE READER
15	1	Is the device inoperable?
15	2	Is the device operating (busy)?
15	8	Is there a Parity Error on a read (PE)?
16	4	Has A-B equality and no gap been sensed?
16	8	Is the E/F-E/D Indicator set?
		PAPER TAPE PUNCH
15	1	Is the device inoperable?
15	2	Is the device operating (busy)?
15	8	Is there a Parity Error on a write (PE)?

Cont'd on next page.

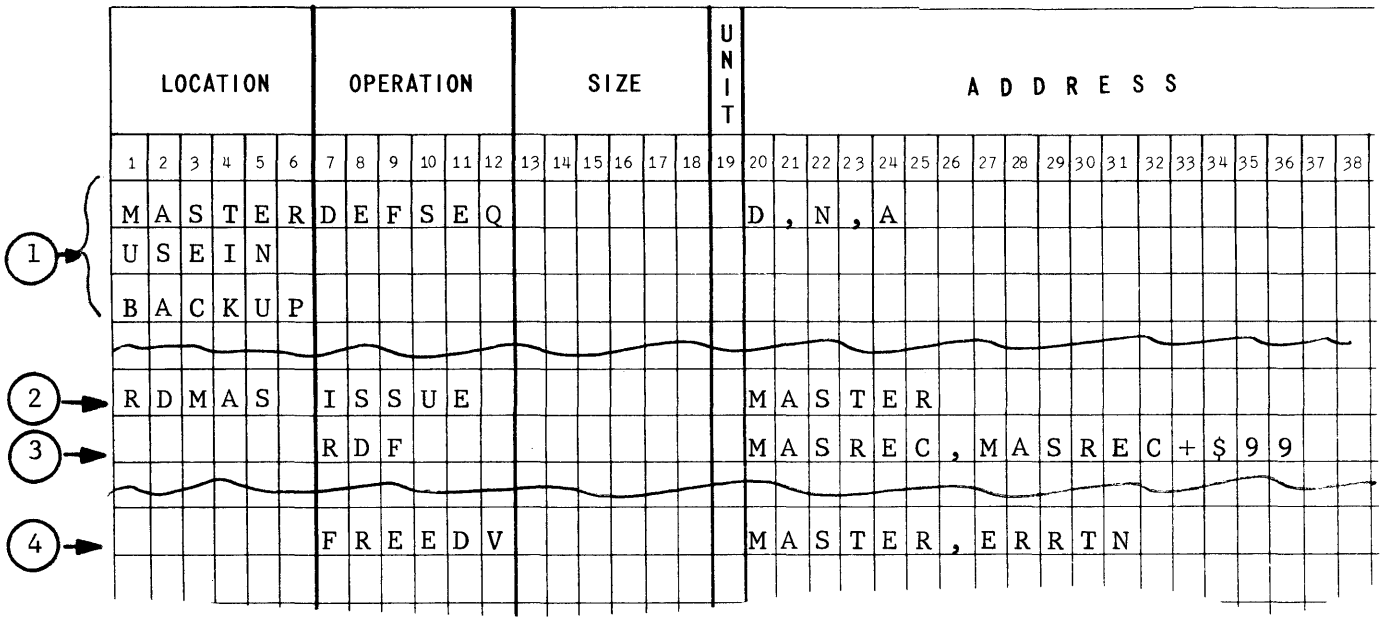
TESTDV
Format

COL.	CHAR.	Test or Command
		ON LINE PRINTER
15	1	Is the device inoperable?
15	2	Is the device operating (busy)?
15	4	Is there a low paper supply?
15	8	Is there a Parity Error (PE)?
15	&	Is the paper advancing?

TESTDV
Format

DEVICE CONTROL EXAMPLE 1

(EXAMPLE OF FORWARD READING OF MAGNETIC TAPE OR PAPER TAPE)



(1) This is the Device Sequence Note that only the USEIN and BACKUP lines are necessary.

(2) The ADDRESS entry contains the symbolic name of the Device Sequence.

(3) The input area MASREC must have been allocated by the programmer.

(4) The FREEDV command directs FCP to retain control until the previously ISSUED command has been completed.

The first ADDRESS entry (MASTER) is the Device Sequence name. The second ADDRESS entry (ERRTN) is the location to which control will be transferred if any of the following conditions are present:

a. E/F or E/D

b. A-B equality and no gap on tape.

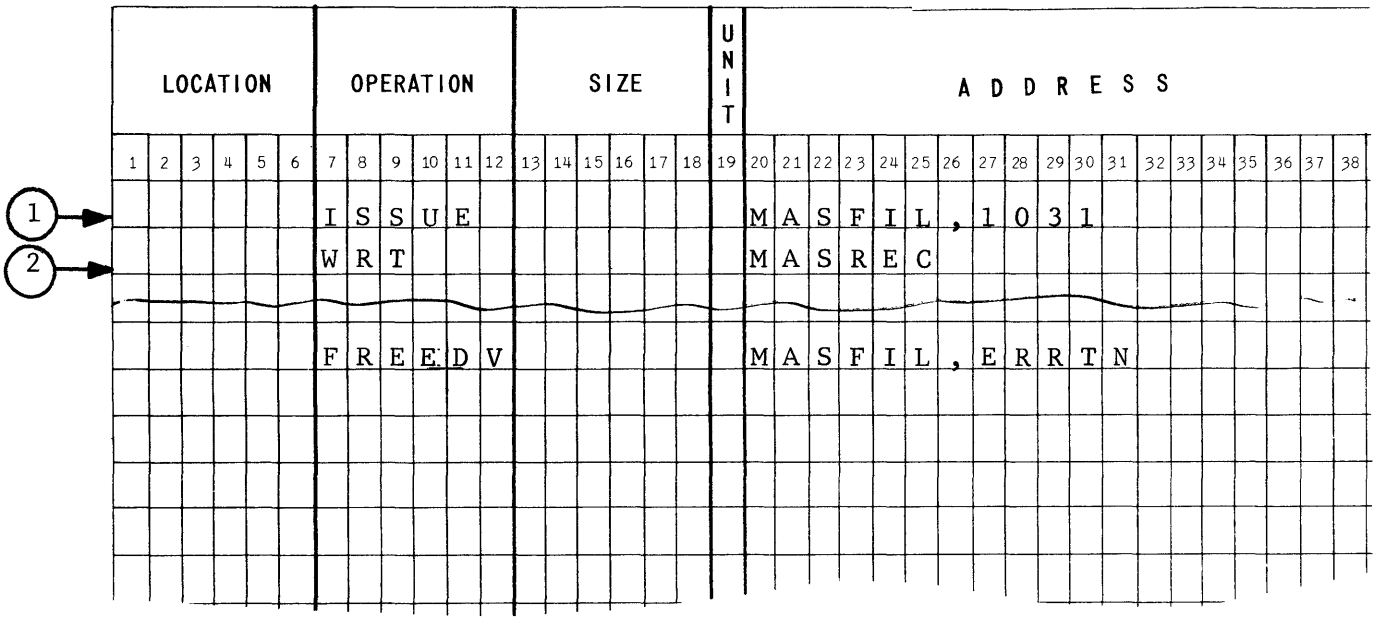
c. Unrecoverable read parity.

(3) (4) Symbolic names may appear in the LOCATION field for these lines if desired.

DEVICE CONTROL Example 1

DEVICE CONTROL EXAMPLE 2

(Example of writing to a device that may include the printer as one of the interchangeable devices.)



(1) First ADDRESS entry (MASFIL) is the name of the Device Sequence. Second ADDRESS entry is printer control information for printing a 120 character line in the Asynchronous mode and advancing the paper three lines.

(2) First ADDRESS entry is the tag of a 12 (or 16) decade area allocated by the programmer and will contain information to be written.

DEVICE CONTROL Example 2

SPECIAL DEVICE CONTROLLING CODES

These controlling codes TYPE and TYPERD used for the console typewriter do not require an associated sequence description.

The two codes and their functions are:

TYPE

This code provides the ability to type a message on the console typewriter

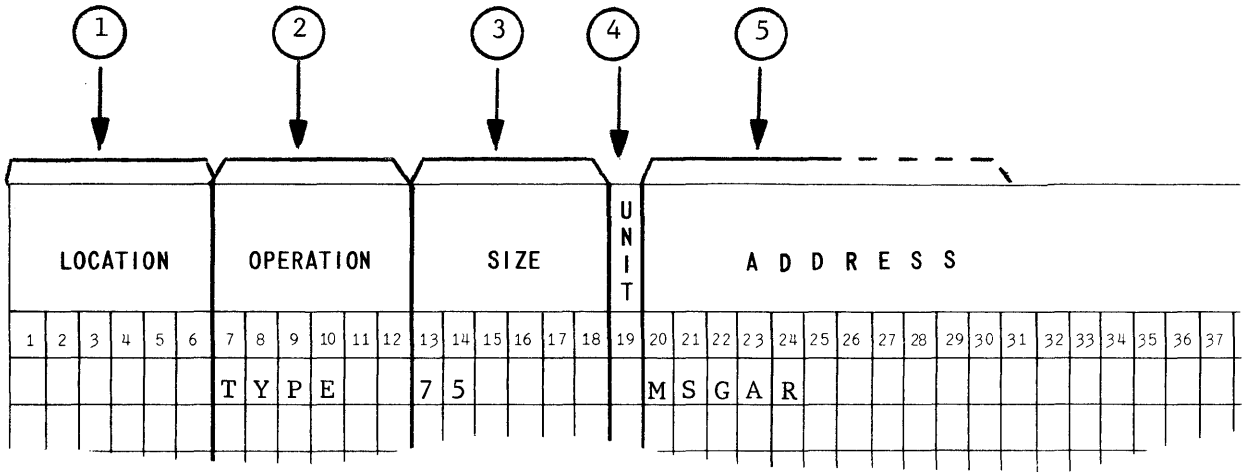
TYPERD

This code provides the ability to type a message on the console typewriter and receive a return message.

These are special codes and do not require the use of either the ISSUE or FREEDV codes.

TYPE (Format)

(The TYPE controlling code provides the ability to type a message on the console typewriter. The message area is immediately available to the user.)



FORMAT NOTES

LOCATION

A symbolic name may appear in this field if desired.

OPERATION

TYPE must appear in this field

SIZE

The decimal number of characters to be TYPEd is written in this field. The range of characters to be TYPEd may be between 01 and 79 if the UNIT field is left blank. If a letter I is entered in the UNIT field the range may be from 01 to 85.

UNIT

If the UNIT field is blank, a six character area at the beginning of the TYPEd message will be allocated for the Task Identification label followed by a space. If an I is entered in the UNIT field, this prefix function will be inhibited and the message may be 85 instead of 79 characters in length.

TYPE Format

ADDRESS

The name (tag) of the message area (left-end address) is written in the ADDRESS field. This may be an address that is modified and/or indirectly addressed.

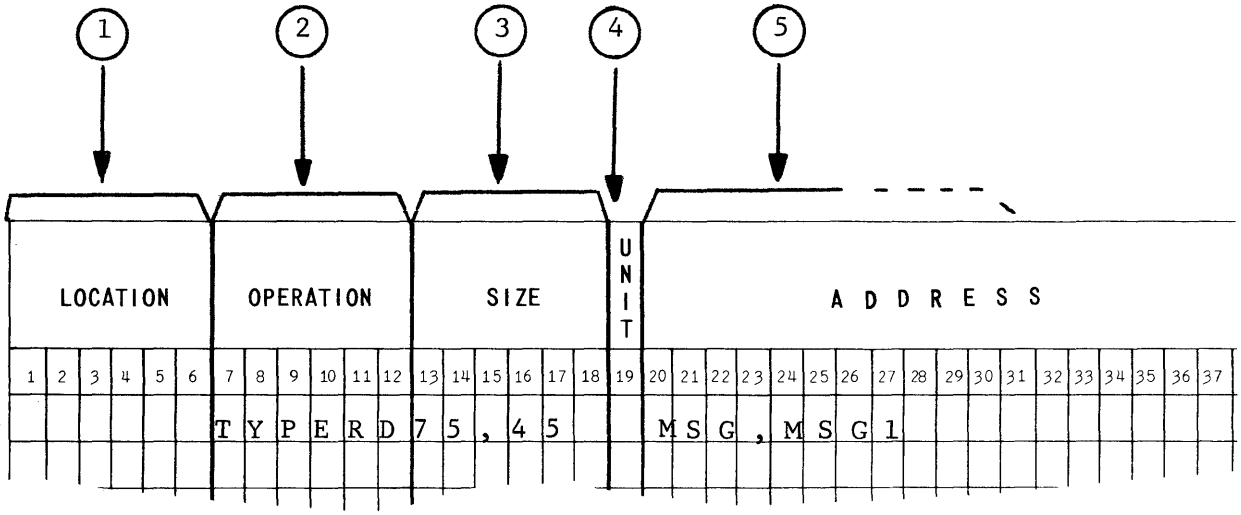
NOTE

The use of the TYPE device code destroys the contents of STA and STP.

TYPE
Format

TYPERD (Format)

(The TYPERD controlling code provides the ability to type a message on the console typewriter and then receive a return message from the console.)



FORMAT NOTES

LOCATION

A symbolic name may appear in this field if desired.

OPERATION

TYPERD must appear in this field.

SIZE

The decimal number of characters to be TYPEd is written as the first entry in this field. A comma follows this entry and the number of characters of the input messages is the second entry of this field. The output message size (first entry) may range from 01 to 79 if the UNIT field is left blank or from 01 to 85 if the letter I is written in the UNIT field. The input message size (second entry) should be one greater than the actual input message desired.

TYPERD
Format

UNIT

If the UNIT field is blank, a six character area at the beginning of the TYPEd message will be allocated for the Task Identification label followed by a space. If an I is entered in the UNIT field, this prefix function will be inhibited and the message may be 85 instead of 79 characters in length.

ADDRESS

The name (tag) of the output messages is written as the first entry followed by a comma.
The name (tag) of the return message area is written as the second entry.
Either of these addresses may be modified and/or indirect addresses.
The terminal address plus one character will be stored in the four character area following the read area; therefore, four additional characters must be allocated following the input area.

NOTE

The use of the TYPERD device code destroys the contents of STA and STP.

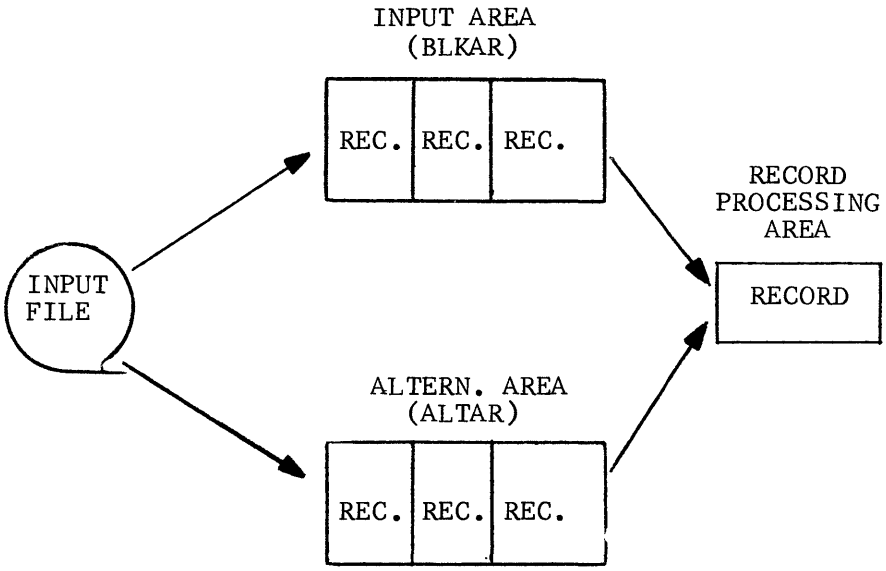
TYPERD
Format

APPENDIX B

IN-PLACE PROCESSING

The type of batch processing discussed in previous sections of this manual has required the allocation of a record processing area in addition to the input or output areas.

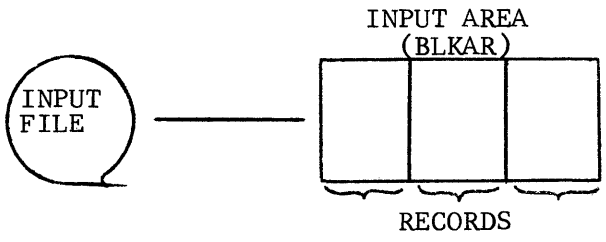
An illustration of the area allocation for processing of this type would be as illustrated below for an input file which is batched:



This method of processing has advantages. It is easily understood and easily programmed. The programmer can address items within the record area by their symbolic names without indexing. With the provision of an alternate area, he can obtain maximum simultaneity without being concerned with which input area is providing the record.

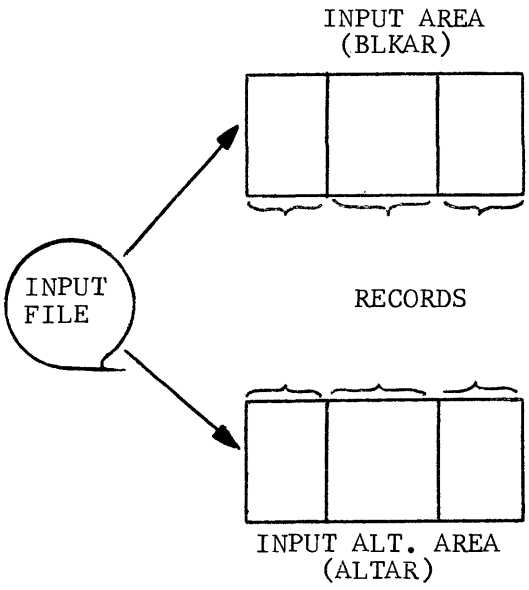
However, in some programming situations there are disadvantages to processing in this manner. The programmer may not have sufficient memory to allocate for either an alternate area or a record processing area. Also in some situations he might want to save the time required to move each record to a processing area.

In such a situation, the programmer may decide to process each record in the input area (BLKAR) as in the following illustrations:



Each Record processed in place in the Block Area (BLKAR)

The above illustrates the use of the block area only for processing. The programmer might also use an alternate area. This would provide for simultaneity and would still allow for in-place processing as in the following illustration:



Each record processed in place in the Block and Alternate Areas.

When doing in place processing, the programmer must furnish FCP with use of the second or third INDEX Fields if either or both of the following conditions are present:*

1. The file is in batched record format.

2. An alternate file area has been allocated.

Example of in-place processing in a Block Area (BLKAR)

As an example of in-place processing of an input file, assume that a file is batched by five fixed length, 100 character records and each record will be processed in place in the input area.

The programmer allocates a 500 character area, decade oriented, as the input block area and gives the symbolic tag INPAR to this field.

For the in-place processing function, the pertinent lines of the file descriptor sequence for this file are as follows. (See Section V for complete format of FCP File Sequence.)

---

*It should be noted that providing FCP with the use of the INDEX Field (M2 or M3) as a modifier does not preclude its use by the programmer for other functions.

LOCATION						OPERATION						SIZE						UNIT	ADDRESS												
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
I	N	F	I	L	E	D	E	F	S	E	Q								F	,	N	,	A									

INFILE is the name assigned to this file.

B	L	K	A	R															I	N	P	A	R									
A	L	T	A	R																													

INPAR is the input block area assigned for this file and no alternate area is assigned.

R	E	C	A	R														2															

The RECAR line specifies in the UNIT field that Index 2 has been assigned.

B	A	T	C	H														F	0	0	0	5											
R	E	C	O	R	D							B							0	1	0	0											

The BATCH line specifies a fixed (F) number of records (0005) to the batch. The RECORD type is B and there are one hundred (0100) characters in each record.

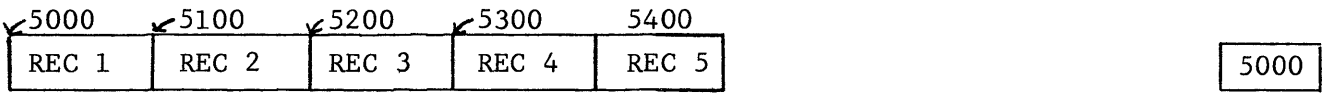
FCP, furnished the above information, can perform its input function and furnish the address of the current record being processed in INDEX Field 2.

As an example, assume that at object time, the area allocated for INPAR is HSM 5000 to 5499. When the first READ controlling code of the program is encountered, the input area is filled with five records and INDEX Field 2 contains the address of the first record as illustrated below:

First READ

Input Area INPAR

INDEX 2
After READ



At the time the next READ is encountered the INDEX Field 2 is incremented by 0100 (the number of characters on the RECORD line) as illustrated below:

Second READ

INDEX 2
After READ

5100

This incrementing function of INDEX 2 continues with each encounter of a READ controlling code until the input area is exhausted. This is controlled by the number of records (5) on the BATCH line. When the area is exhausted, FCP performs a physical read, refilling the input area and resets INDEX 2 to 5000.

Thus, the programmer is aware that following every read the left-end address of the record being currently processed is in INDEX 2.

He might use this field as a modifier to each address in the record processing path. As an example, assume that each record consists of items in the following format:

Item	No. of Characters
ACCOUNT NO.	7
CODE	3
BALANCE	8
FILLER ITEMS	82
TOTAL CHARACTERS	100

To address the Account No., for example, he might write in the ADDRESS field of an instruction a zero relative indexed address as follows:

$0:M2

and to address the Code item a similar type address each as:

$7:M2

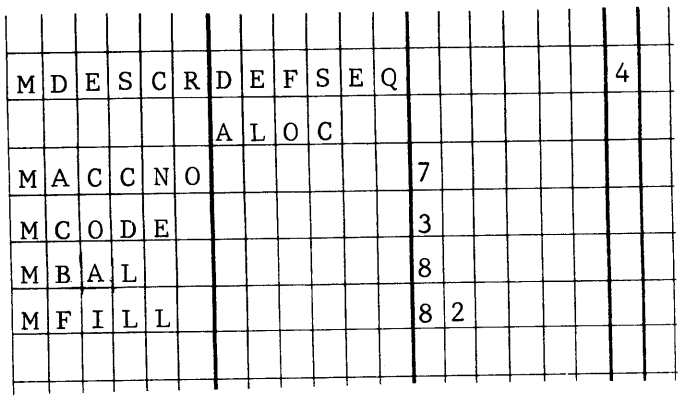
However, the 3301 Assembly System provides an option for this type of addressing function which allows the use of symbolic names in place of pure zero relative addresses.

As an option at assembly time, a sequence may be assembled using a code of 4 in the UNIT field (See VALIDATION - Section III of this manual).

This code assembles a zero relative address for each symbolic name in the sequence and further specifies that the sequence is not to be loaded at object time.

The programmer could write a sequence and obtain addresses for this example as follows:

CODE 4 VALIDATION*



ASSEMBLED ADDRESSES

0000 - 0006
0007 - 0009
0010 - 0017
0018 - 0099

*The only codes other than ALOC that may be used in this type of sequence are REDEF, RENMANE and REMARK.

Having thus defined the sequence, the programmer might write addresses as follows:

After first READ

	INDEX 2	EXECUTED ADDRESS*
MACCNO:M2	5000	5000
MCODE:M2		5007
MBAL:M2		5010
MFILL:M2		5018

*Assuming generated left-end address.

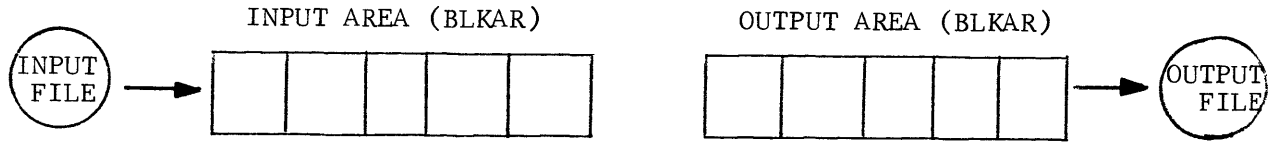
After second READ

	INDEX 2	EXECUTED ADDRESS*
	5100	
MACCNO:M2		5100
MCODE:M2		5107
MBAL:M2		5110
MFILL:M2		5118

*Assuming generated left-end address.

Another example of in-place processing may be used to conserve a separate "record" area. This method allows processing of the record in the output area. The following is an illustration using only the input and output block areas. (Alternate--

ALTAR--areas may also be used in this method but are not shown for simplification purposes.)



In this example each successive READ moves an input record to the output area and is processed in the output area. When a WRITE is given for the output file the record is included in the output batch.

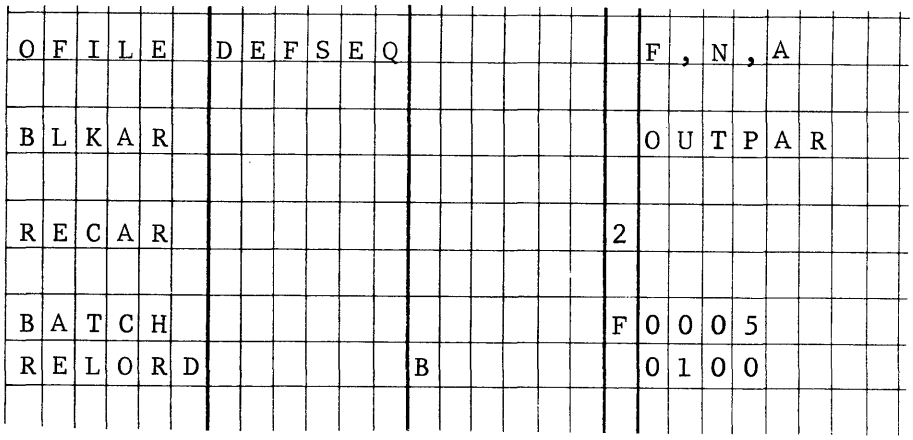
For this example assume again that there are five 100 character records per input and output batch and that the programmer has allocated an input area and output area as INPAR and OUTPAR respectively.

The lines pertinent to the in-place processing function for both the input and output file are as illustrated below.

INPUT FILE

LOCATION						OPERATION						SIZE						UNIT											
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27			
I	F	I	L	E		D	E	F	S	E	Q								F	,	N	,	A						
B	L	K	A	R															I	N	P	A	R						
R	E	C	A	R															:	M	2								
B	A	T	C	H														F	0	0	0	5							
R	E	C	O	R	D							B							0	1	0	0							

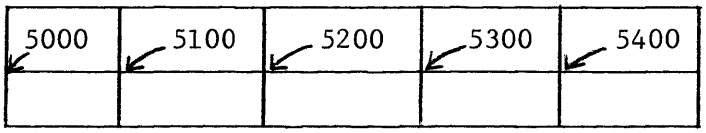
OUTPUT FILE



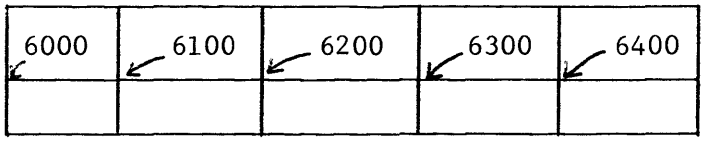
It should be noticed that INDEX Field 2 is used as a reference on the RECAR line for both the input and output files. Thus, the FCP Controlling code READ for the input file not only fills the input area when necessary but also transfers a record to the location in the output area. The output file however is controlling the INDEX Field. INDEX 2 is not incremented until a WRITE FCP controlling code is given.

To illustrate, assume the following allocations are made at object time.

INPAR



OUTPAR



PROGRAMMED OPERATION	INDEX 2 CONTENTS AFTER FCP CODE	FUNCTION(S) PERFORMED
(FIRST) READ	6000	*Input area (INPAR) filled. First record moved to 6000 (INDEX 2)
(PROCESSING STEPS)		Processed in output area. All addresses modified by :M2
(FIRST) WRITE	6100	Record included in output batch by incrementation of INDEX 2
(NEXT) READ	6100	*Next record moved from input area to 6100 (INDEX 2). Note that INDEX 2 is not affected by a READ.
(Processing Steps)		
(NEXT) WRITE	6200	Record included in output batch by incrementation of INDEX 2.

*FCP is using INDEX 2 only to move the current record to the output area. It is using internal counters (number of records per batch) and the number of characters per record to maintain position control within the input area.

The examples given have been simplified for illustrative purposes. It should be realized that these same methods may be used with records of a different format type, with alternate areas provided, and with batch sizes that are of different size on input and output files.

APPENDIX C

STUDENT PROJECT

STATEMENT OF THE PROBLEM

This problem is for training purposes only and in no sense is it intended to be a realistic approach to a problem.

The Standard Gas Company (a public utility) includes the merchandising application as a part of its data processing operation. Three types of merchandise are sold on a time payment contractual basis. These are gas ranges, water heaters, and conversion (gas) burners.

For this problem, the assumption is that only three models of each item may be contracted for and a table of these items by type of stock is on page C-9.

The problem requires you to write sequences for two runs in the merchandising application; a posting run, and a billing message calculation and preparation run. A systems chart for each of these runs is on the following page.

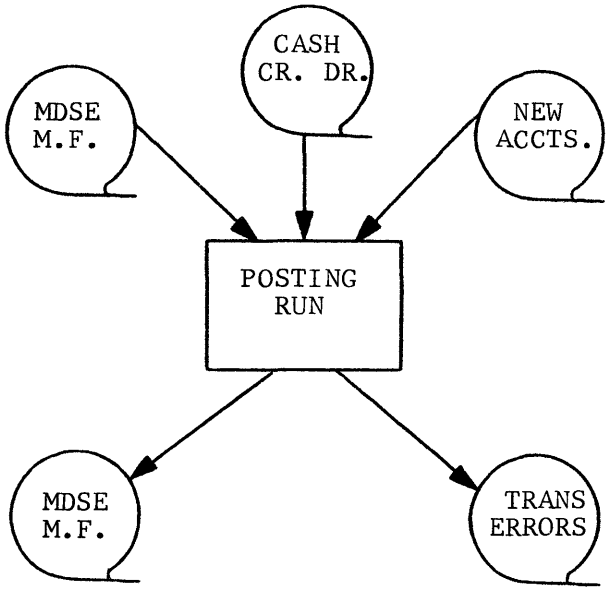
The posting run is simplified to include only two general types of transactions. One transaction file consists of cash payments and revenue adjustments. These are posted to the master file and accumulated in credit fields. See format and processing action on page C-7. A second transaction file consists of new accounts. It will be necessary to construct a master file record for each new account. See format and processing action on page C-5.

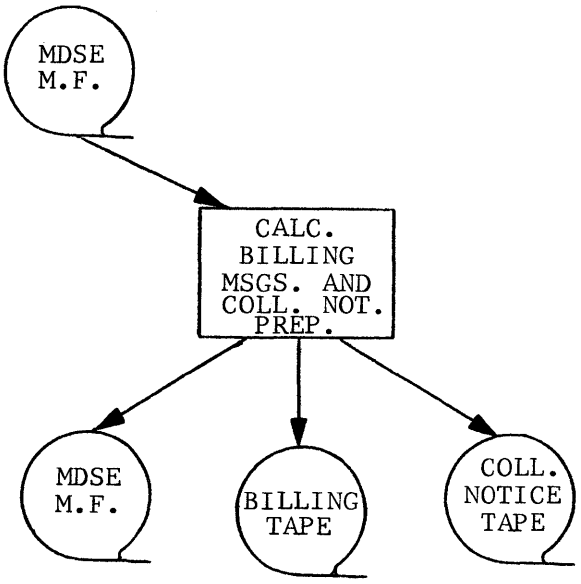
In the billing run, you will prepare a billing message for each record on the master file and a collection notice message for each account in arrears after the bill is calculated.

You will be required to prepare sequences for each of these process runs as outlined in detail under Requirements on page C-3.

MERCHANDISE APPLICATION

PARTIAL SYSTEMS CHART





REQUIREMENT 1

Prepare the FCP file sequences.

REQUIREMENT 2

Prepare a data sequence to describe the record areas for all files in the Posting Run.

REQUIREMENT 3

Prepare the PROCESSING sequences for Posting Run. Sequence of preparation to be given in class.

REQUIREMENT 4

Prepare PROCESSING sequences for Calculating Bill and preparation of billing and Collection Notice Tape. Sequence of preparation to be given in class.

DATA SHEET

MERCHANDISE MASTER (TAPE)

FIXED FIELD FORMAT

UNBATCHED

MERCHANDISE ACCT NO	10	
UTILITY ACCT NO	10	
CUSTOMER NAME	30	
STREET ADDRESS	20	
CITY STATE ADDRESS	18	
BAD CHECKS	2	
COLLECTION NOTICES SENT	2	
MERCHANDISE STOCK NO	4	
TYPE OF STOCK	3	
DESCRIPTION	15	
SALES ORDER NO	5	
DATE OF CONTRACT (YR MO DA)	6	
TOTAL MOS CONTRACT PERIOD	2	
NO MOS REMAINING CONTRACT PERIOD	2	
ALLOWANCE	7	
TYPE ALLOWANCE	1	
DEPOSIT	7	
FINANCE CHARGE	7	
TIME BALANCE (TOTAL)	7	
MONTHLY PAYMENT	7	
FINAL PAYMENT	7	
TOTAL DUE	7	
AMOUNT CURRENTLY DUE	7	
30 DAY ARREARS	7	
60 DAY ARREARS	7	
90 DAY ARREARS	7	
OVER 90 DAY ARREARS	7	
1st CREDIT CODE (CASH PAYMENT)	3	Related Items
DATE (MO DA)	4	
CREDIT AMOUNT	7	
2nd CREDIT CODE (MISC CREDITS)	3	Related Items
DATE (MO DA)	4	
CREDIT AMOUNT	7	

DATA SHEET

NEW ACCOUNT TRANSACTION (TAPE)

FIXED FIELD FORMAT

UNBATCHED

MERCHANDISE ACCT NO	10
TRANSACTION CODE	3
UTILITY ACCT NO	10
CUSTOMER NAME	30
STREET ADDRESS	20
CITY STATE ADDRESS	18
MERCHANDISE STOCK NO	4
TYPE OF STOCK	3
SALES ORDER NO	5
DATE OF CONTRACT	6
ALLOWANCE	7
TYPE ALLOWANCE	1
DEPOSIT	7

CODE	TITLE	ACTION
010	NEW ACCOUNT	1. Construct New Master and insert on Master File. 2. Create following fields on New Master. All other alpha and numeric fields space and zero filled respectively. TOTAL MOS CONTRACT PERIOD Calculate based on Time Balance Amt. LESS THAN $100.00 12 MOS $100.00 - 199.99 18 MOS $200.00 - 299.99 24 MOS $300.00 AND OVER 36 MOS FINANCE CHARGE 6% per year on Cash Price less Deposit and allowance. EXAMPLE: $300.00 Cash Price less Deposit & Allowance 54.00 FINANCE CHARGE TIME BALANCE (Cash Price less Deposit and Allowance plus finance charge) MONTHLY PAYMENT TIME Balance ÷ NO MOS (Nearest full dollars) FINAL PAYMENT (Monthly Payment X (Total Months Contract Period -1) minus Time Balance TOTAL DUE Amt Currently Due plus Arrears AMOUNT CURRENTLY DUE This month's payment ARREARS FIELDS Calculated and updated in Billing Run except for New Accts and Bad Check Transactions

DATA SHEET

CASH RECEIPTS, CREDIT AND DEBIT

TRANSACTIONS

MERCHANDISE ACCT NO	10
CODE	3
DATE (MO DA)	4
AMOUNT	7

FIXED FIELD FORMAT

BATCHED BY 10

CODE	TITLE	ACTION
110	CASH PAYMENT	1. Add amount to first Credit Code Amount field. 2. Date to 1st Credit Code date field.
120 to 150 incl.	MISC. CREDITS	1. Add amount to 2nd Credit Code Amount field. 2. Date to 2nd Credit Code date field. 3. Code to 2nd Credit Code field.
160	BAD CHECK	1. Add amount to Total Due. 2. Add amount to last open arrears field. 3. Increase Bad Check Code by 1.
161	BAD CHECK (DEPOSIT)	1. Add Amount to Total Due (plus 10% of AMOUNT as a bad check charge). 2. Add amount to last open arrears field. 3. Increase Bad Check Code by 1.

DATA SHEET
BILLING MESSAGES
FIXED FIELD FORMAT
VARIABLE NO OF FIELDS
BATCHED BY FIVE

MDS ACCT NO	10
CUST NAME	30
ST ADDR	20
CITY STATE ADDR	18
TOTAL DUE	7
AMOUNT DUE	7
30 ARREARS VARIABLE	7
60 ARREARS VARIABLE	7
90 ARREARS VARIABLE	7
OVER 90 ARREARS VARIABLE	7
E/I	

DATA SHEET
COLLECTION NOTICE MESSAGES
FIXED FIELD FORMAT
BATCHED BY FIVE

MDSE ACCT NO	10
CUST NAME	30
ST ADDR	20
CITY STATE ADDR	18
COLLECT NOTICES	2
TYPE STOCK	3
MDSE STOCK NO	4
DESCRIPTION	15
SALES ORDER NO	5
TOTAL DUE	7
AMT CURR DUE	7
ARREARS TOTAL	7

STOCK AND PRICE

TABLE

TYPE OF STOCK	CLASS
0XX	GAS RANGE
1XX	WATER HEATER
5XX	CONV. BURNER

TYPE OF STOCK	CASH PRICE	DESCRIPTION
011	150.00	#75 ECON
015	195.00	#84 AUTO
016	220.00	#93 AUTO
100	75.00	#25 ECON
110	170.00	#30 AUTO
116	275.00	#40 MAGIC MIND
501	250.00	#17 ECON
503	325.00	#41 MEDIUM
505	1093.00	#85 INDUSTRIAL

POSTING RUN PROCESSING ERROR CONDITIONS

1. Same Merch. Acct. No. on Master and New Acct. Record

   Write new account record to error tape with a "9" as the third digit of the transaction code.

2. New Acct. Record with Erroneous Trans. Code

   Same as above except with an "8" as third digit of the transaction code.

3. No Matching Master for Cash Record

   Write transaction to error tape with "7" as third digit of transaction code.

4. Cash Record with Erroneous Trans. Code

   Same as No. 2 above.

RCA 3301 REALCOM Training Manual

Dear Reader:

Have you any comments or suggestions concerning this publication? Are there any omissions or deletions which you feel would enhance its usefulness....are there any errors that should be corrected? If so, we would appreciate it if you would use this form to tell us about it.

CORRECTIONS AND OMISSIONS (Please specify page numbers):

FOLD FOLD

CUT ALONG LINE

ADDITIONS AND DELETIONS (Please specify page numbers):

FOLD FOLD

Name _____

Title _____

Company _____

Address _____

NO POSTAGE REQUIRED IF MAILED IN U.S.A.

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 16
CAMDEN, NEW JERSEY

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY—

RADIO CORPORATION OF AMERICA
ELECTRONIC DATA PROCESSING
CAMDEN, NEW JERSEY 08101

ATTN: Manager, Education and Training
Cherry Hill
Bldg. 204-2

CUT ALONG LINE

FOLD

FOLD