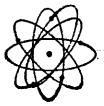
RADIO CORPORATION OF AMERICA | ELECTRONIC DATA PROCESSING | CAMDEN, N.J.



RCA 301 TRAINING MANUAL

RCA 301 EDP | ELECTRONIC DATA PROCESSING SYSTEMS A BROAD RANGE OF SPEEDS AND CAPABILITIES TO MATCH MANY USER REQUIREMENTS

RCA 301
TRAINING
MANUAL

Printed in U.S.A.
WA2-63

93-06-000

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

This edition includes 93-06-000 (Feb. 1963 Edition), and 93-06-000-1 (March 1963 Revision).

When ordering this manual, the following publication control number should be used:

93-06-000 (December 1963 Edition)

TABLE OF CONTENTS

CHAPTER	TITLE
I	History and Development
II	Development of an Application
III	What is a Computer?
IV	RCA 301 Equipment
V	The Binary Numbering System
VI	Data Layout
VII	High Speed Memory
VIII	Instruction Format and Computer Program Record
IX	Operating Logic
X	An Introduction to Flow Charting
XI	Introductory Instructions
XII	Card Instructions
XIII	Tape and File Terminating Test Instructions
XIV	Decision Making Instructions
XV	Data Handling Instructions
XVI	Printing
XVII	Iterative Coding
XVIII	Address Modification
XIX	Variable Connectors
XX	Logical Operations
XXI	Handling Variable Data
XXII	Repetition of Instructions
XXIII	Indirect Addressing
XXIV	Timing

TABLE OF CONTENTS (Cont'd)

CHAPTER	TITLE
XXV	Batching
XXVI	Simultaneity
XXVII	Editing
XXVIII	Program Controls
XXIX	Data Record File
XXX	Data Record File Mode
XXXI	Data Disc File
XXXII	Table Look-Up Techniques
XXXIII	Randomizing
XXXIV	Arithmetic Operations
XXXV	Console
XXXVI	RCA 301 Programming Standards
XXXVII	RCA 301 Supplied Routines
XXXVIII	Compatibility
XXXIX	Sorting and Merging
APPENDIX A	MICR Sorter Reader
APPENDIX B	Arithmetic Instructions
INDEX	

I — HISTORY AND DEVELOPMENT

Men have always been fascinated by mathematics but bored by the drudgery of it. In early days, the amount of computation required was relatively little and could be accomplished by piling stones in a heap or notching a stick. As numbers became more important in the advancement of civilization, numbering systems had to be developed. The development of the decimal system was the real beginning of experimentation in computational devices. The earliest of these devices was man's fingers, but unfortunately (or fortunately, as the case may be) he was relatively limited in this respect. If numbers can be represented by fingers, they can be represented by other things as well, and this thought lead first to the stones and notches we mentioned and finally to the construction of the earliest computer, the abacus. The abacus is nothing but beads strung in groups of ten. These beads can be moved easily and rapidly, and in the hands of skillful operators rival the speed of our modern mechanical calculators even today.

The beginning of modern machines dates back to the seventeenth century. Many of the famous mathematicians of that time (Pascal, Liebnitz, and Napier) invented mechanical devices to aid them in their work. Pascal's machine is particularly important because it introduced some of the basic principles and mechanisms which are still used in our machines. Pascal encountered the one difficulty which had been the stumbling stone up to that point, the fact that a machine which is to calculate must have the ability to recognize the need to "carry". That is, when a number is added mechanically, the carry (if any) must also be taken care of mechanically. Pascal's carry mechanism is significant as the beginning of automatic computing because it removes a function from the operator to the machine. Unfortunately, in an effort to make the invention completely free of operator control, he passed the technological boundary of his era, and therefore his invention was a failure, as were other adding machines for the next two centuries.

In 1885, William Burroughs produced the first saleable adding machine. The device caught the attention of the business world and the machines were improved so that they could both add and subtract.

Around the time of World War I, desk calculators were developed. These machines could perform arithmetics by keyboard control. All of these devices, however, required human operators who had to be trained. This introduced the possibility of human error.

In 1830, Charles Babbage attempted to build an automatic mechanical calculator. He failed again because of technical problems of construction. Babbage did, however, lay the design groundwork for our modern day computer. He divided his machine into three parts which he called the store, the mill, and the control. The purpose of the store was to hold all the data which would be used during the long computation. The mill worked on the data and the control was the automatic operator.

To aid in the computations of the 1890 census, Hollerith and Powers developed electrical contact reading which led to the initial punched card systems.

It wasn't until after World War I that the relay was developed. This was the technological advancement needed for the control functions that would free mechanical devices from the need of being operator controlled.

World War II brought about the impetus for development of our modern high speed electronic computers. The first computer was the Mark I developed at Harvard. In 1943, the ENIAC was built by the University of Pennsylvania. The ENIAC was the first all-electronic computer. This was the break through into the field of electronic computation and into the field of data processing.

II — DEVELOPMENT OF AN APPLICATION

Let us say that our ABC Bank has decided that its paper work is getting a strangle hold on the checking account part of business and that further mechanization is necessary. We ask for a proposal from a number of different computer (or data processor) manufacturers. A proposal is simply a written report on what their individual equipment will do for us and what will be required of us in terms of monthly rental, space requirements, time, personnel, etc. In order to prepare this report, each manufacturer's personnel must familiarize themselves with the problem. This requires that they learn:

1. What input will be fed into the computer. In our case this would include what information must be maintained for each account; what information will be brought in from each check or deposit; what additional input there might be, such as stop payments, holds, change of addresses, new accounts, etc.

2. What output will be required. Obviously one thing must be updated master information, but in addition, statements will be required periodically, summarizing reports will be needed, overdraft notices must be prepared to send to negligent account holders, etc.

3. What steps are currently being taken by the bank personnel to handle the work.

Having learned this, the analyst can determine what equipment will be needed to do the job in the time allowed and basically how the job will have to be handled.

Analysis
↓
Program
↓
Test Data

Once our ABC Bank has decided on a particular system, both the bank and the manufacturer supply personnel to begin the detailed work. The bank personnel must be trained to be able to handle the equipment. This training will be in programming and operating. The data developed by the analyst must be gone over in detail. The exact format of the input and output data must be determined. Specific problem statements must be prepared, and the programmers must determine the best way to have the equipment perform the required tasks. Having developed the procedure that the computer must follow, it is necessary to "code" or translate it into instructions in a form that the computer can understand. Once a program is written, it must be checked and rechecked, which will also include actually running it on the computer with test data to make sure that the results will be accurate. This must be done for every program or "run", before the actual equipment is delivered.

Once the equipment is installed and checked out, the conversion and systems check must be executed. This may mean that only a few programs are actually put into operation, with only the input data required for those programs converted to computer input; or it may mean that the entire system of programs is put into operation but only on a portion of the data.

Once the personnel in charge have assured themselves that things are running smoothly, the complete system can be established. Additional programs encompassing other jobs (such as savings accounts, special "club" accounts, etc.) are added as time allows. This may or may not require the addition of more equipment.

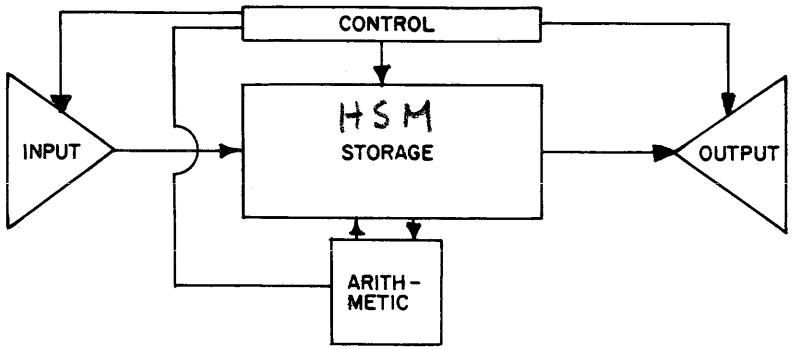
The same sort of procedure takes place at any installation, whether the application is demand deposit accounting which we covered here, customer billing, payroll, premium payments for an insurance company, subscription mailing, stock inventory, general accounting, or practically any job that requires a large volume of data processing.

III — WHAT IS A COMPUTER?

There are two basic types of computers; the analog computer which calculates by using physical analogs of the variables, and the digital computer which calculates by expressing all characters in a bi-state form. We are interested in digital computers.

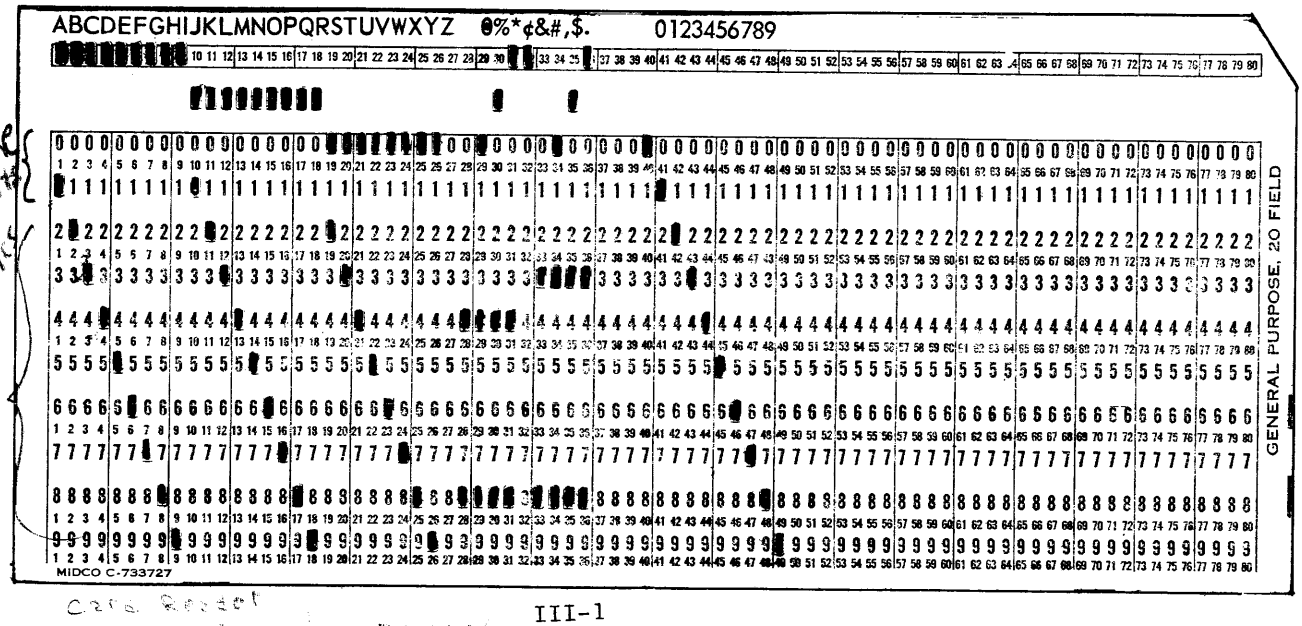
A computer is made up of five basic elements:

1) Input - i.e. Card Reader, tape Reader (Magnetic Tape, Paper Tape),
2) Output - i.e Card Punch, Tape (Magnetic & Paper), Printer
3) Storage - Main Memory
4) Arithmetic - Decisions, calculator, logic Functions (Comparision, etc)
5) Control - i.e tells Card Reader to read 1 card,



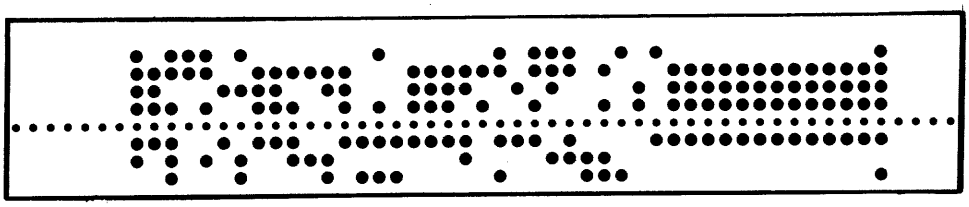
The purpose of the input devices is to feed data into the computer. There are numerous types of input, but we will concern ourselves with three basic types: cards, paper tape, and magnetic tape. All of these types of input are coded with a binary code. That is to say, it is only possible to represent data on cards and tape using holes or no holes, magnetic bits or no magnetic bits. Unique combinations of these holes and no holes, or bits and no bits represent the different alphabetic, numeric and special symbols.

In the case of card input, we will limit our discussion to 80 column EAM (Electric Accounting Machine) cards. An example card is shown below:



As you can see, the card has been "punched". That is to say, the card was run through a card punching device which works along the same lines as a typewriter, the difference being that the output are cards punched with the binary code of the character desired and, in this case, the character above it. This card can be read by a device called a card reader which can then transmit the data by impulses. In order to do this, the card passes over a roller, and a brush senses the holes or the absence of holes: i.e., when the brush is capable of touching the roller (hole must be present), it causes a current to flow.

Paper tape is another input device. It is also prepared on a special piece of equipment which causes the proper binary code to be placed in the paper tape. Paper tape is spoken of as having "levels". For example, a 7 level paper tape would refer to paper tape whose code for each character consists of 7 holes and no holes; 5 level tape has a code consisting of 5 holes and no holes per character. Information is separated on paper tape by the "gap" or physical lack of any punching except for a sprocket hole. A sprocket hole is punched along with each character (or in each character location). It is utilized by the reading device as well as the punching device to move the tape. A sample of punched paper tape (7 levels) is shown below:



Magnetic tape also utilizes a binary code; however, it is represented by magnetic bits. The tape itself is made of oxide coated plastic and comes in various widths and lengths depending upon the equipment on which it is to be used. This tape is processed in a unit called a "tape station" which contains, among other things, a read-write head. This device has the ability to place bits on the tape (write) by transmitting a current which it receives from the computer or other device. It also has the ability to sense the presence of these bits (read). If we can say that a + current places the bits, we can also say that a - current would erase them, thus allowing us to use the same tape over again at some later date. Information is separated on magnetic tapes by gaps, as in the case of paper tape. In addition, other sentinels must be utilized to indicate the beginning of the tape and the end of the tape. These differ between equipment.

One additional type of input is manual entry into the computer through the console, which is the operators' device for controlling the computer. This is done for "debugging" pruposes primarily. Debugging refers to proofing of a program in order to ascertain that it will do the job required of it accurately.

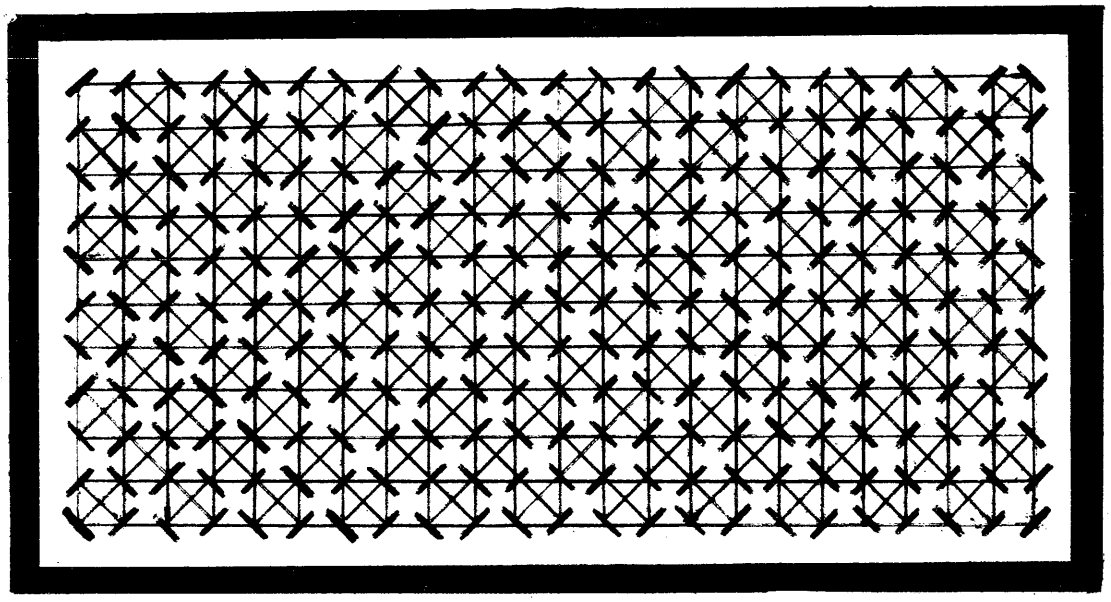
The output from a computer would be made up of the same sort of thing as the input; i.e., magnetic tape, paper tape and punch cards. The devices which produce the output would receive the information and prepare the media, either by punching cards, punching paper tape or placing bits on magnetic tape. The actual punching is done by triggering a solenoid driver which then hits a punch die that would place a hole in the card or tape.

Another very important type of output is printing, that is, preparing actual hard copy that can be read by human beings with ease. This is done by a printer which prepares documents at a rapid pace, by printing a complete line at a time. Many systems also have a high speed electric typewriter device, which produces written information one character at a time. This, obviously, is a much slower device than the printer.

Once the data is in the computer, there must be some place to store it. This unit is a storage device and is called "memory". Many types of memory devices have been used, such as mercury delay lines and electrostatic tubes, however, the magnetic core memories have proved a great improvement in the art. They allow for a larger capacity with regards to data, while requiring less physical area and power requirements. In addition, heat production is less. The internal speed (the time it takes to place a character, remove a character or transfer a character) is also considerably faster. A magnetic core memory is made of ferrite cores, which look something like tiny beads or like miniature doughnuts, and interlacing wires. Each core will represent one bit so that if a code were seven level, it would take 7 cores to represent a character. The wires have the ability to place bits in the cores (activate the cores) or de-activate them. They also have the ability to sense the condition of the cores.

} stack together

An illustration of one plane of these cores appears below:



Magnetize either way yes. No

Third wire called sense wire reads change no change

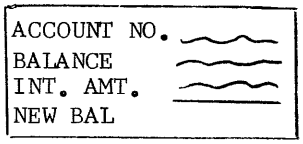
The memory "bank" is formed of a number of these planes.

It is also possible to store data in larger volumes on magnetic drums or discs. These have a much higher access time but cost less per character of storage then the internal computer memory. The information is stored on these devices so that the "bits" that make up a character follow one after another. Writing and reading devices attached to these units have the ability to place characters of information on the drum or disc or read information from the storage device.

The arithmetic unit of a computer performs any arithmetic functions required of it, and also handles such things as making logical decisions (is 13458 greater than 13576?).

The control unit of the computer "supervises" the work being done. It controls the input and output devices and the arithmetic unit and it accesses the storage devices. It knows what to do by following a "program". A program is simply a sequential series of steps to be taken by the computer control. The program may be wired by using a plugboard, or stored in memory. This latter type is the one in which we are interested.

To illustrate a program, let us take a sample problem and develop it for Brand X Computer. For many years the ABC Bank has had a number of girls whose job it was to calculate the interest on each savings account. They then entered on a file card the following information:



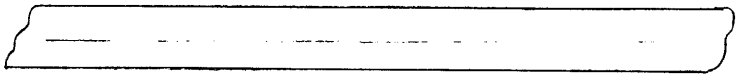
A second group of girls then took one card at a time, added the interest to the balance and placed the new balance on the card, then repeated the process for the next card. It is this job that we now want to do on a computer.

At some previous date, we have prepared a magnetic tape that has for each account:

1) a five digit account number
2) a 6 digit interest amount, loaded with insignificant zeros
3) a 6 digit balance amount, also loaded with insignificant zeros
4) Example: 43675000324134564

acct. int. bal.

The information for each account is separated on magnetic tape by a gap.



The steps we want to go through are:

1) Read in the information for one account. This will cause the tape to be moved past the read head so that we will end up in the gap after the first record and prior to the second record. When we initiate this read again, therefore, we will read in the data for the second record.

2) Add the interest to the balance. Now the balance figure will really be the new balance figure.

3) Write this information out to another tape. This means that we will end up with two tapes: the original and the new updated tape.

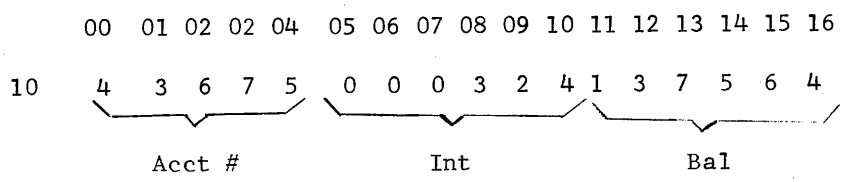
4) Go back and repeat the instructions over again. By going through these four steps, we can process every record on the tape.

In order to spell out these instructions to the computer, we must put them in a language that it can understand. This is called machine language or "absolute".

In addition, our computer has a magnetic core memory in which each character is addressable. A layout of a section of this memory is shown in the figure below:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
10																									
10	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
10	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
10	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99

The first thing we must do is to decide where we wish to place the data that will come in from tape. Having decided to place the first character at 1000, we can visualize how the data will fall when it is actually brought in (see figure). In addition, we must remember that this program that we are writing must be placed into memory after it has been written, so that the control logic can follow it step by step. For this reason, we decide to place the program in memory starting at location 1500 when we read it in.



Each instruction will be made up of 10 characters. The first character tells the computer what to do and is called an operation code, the second character might or might not contain an additional piece of information depending upon the instruction, the next four characters form the first address, and the last four characters form the second address.

The read instruction has an operation code of "R". The first address tells where to place the first character. The characters will then fall in sequentially until the gap is sensed. The second address is ignored, but we still have to tell the computer one additional piece of information and that is the tape station that holds the magnetic tape containing the data. For purposes of our problem, we will say that the tape is mounted on tape station 1. Our instruction therefore would be:

R 1 1000 0000 Read in the data placing the first character at 1000.
The data is on tape 1.

The add instruction has an operation code of "+". The addition will take place from right to left, so we must indicate the right hand end of each operand. We want to add to the balance, so we will mention this first. The second address will give the location of the right hand end of the interest. The computer must also know when to stop, and we'll specify the number of characters to be added in the information digit. Our instruction will be:

+ 6 1016 1010 Add the six characters of the interest to the six characters of the balance.

The write instruction has an operation code of W. It will be necessary to tell the computer the left hand end and the right hand end of the sector to be written and also the tape station to be written to:

W 2 1000 1016 Write out the data from 1000 to 1016 to tape unit 2.

The last instruction simply wants to indicate that the computer must execute the read instruction again. So far we have written three instructions and we have stated that the program will start in memory at 1500. A summary would look as follows:

location	instruction
1500	R 1 1000 0000
1510	+ 6 1016 1010
1520	W 2 1000 1016

The thing we must tell the computer is where to find the next instruction to be executed. That instruction is located at 1500 - 1509, the operation code at 1500. This transfer instruction therefore will have an operation code of 1, and the first address will be the address of the next instruction to be executed. Everything else will be ignored. The entire program will therefore appear as follows:

location	instruction
1500	R 1 1000 0000
1510	+ 6 1016 1010
1520	W 2 1000 1016
1530	1 0 1500 0000

Having written this program, we would have it prepared as input to the computer (either punching cards or paper tape) and would have the operator cause it to be read in from the media to memory so that the first character would fall at 1500. Then we would instruct the computer as to where it will find the first instruction, again 1500. Having mounted the input data tape on trunk 1, and a blank tape on trunk 2, we have only to hit the start button and the computer takes control. Remembering that the computer is not the "giant brain" of publicity but rather a "giant idiot" which must be told step by step what to do and also must be able to "jot down" everything it must remember, we must first mention that the computer has scratch pads called "registers". There will be a register to keep track of the address of the next instruction to be executed, a second to tell what to do, a third to hold the information character, a fourth to track the first address, and a fifth to track the second address. For simplicity sake, let's name these registers the Program register (since it tracks the program), the Operation Code register, the Information register, The First Address register, and the Second Address register. We have already stated that we manually placed in the Program register the address of the first instruction, 1500. By hitting the start button, the computer knows that it must read the 10 characters in memory between 1500 and 1509 in order to know what to do. This it does, placing the information in the appropriate registers, and increasing the program register by 10, in order to address the next instruction which it is to execute. The computer then has in the Operation Code register an "R", which tells it to read; in the Information register there is a 1, which indicates that the information to be read is on tape trunk 1, and that the computer must place this tape in motion. The First Address register holds the location to receive the first character (1000) and the data, as it is read by the tape station, is transmitted into memory at the location designated by the First register. As a character is placed in memory, the register is increased by one in order to address the location to receive the next character. The Second Register is not needed for the execution of this instruction.

When the gap (that separates the data records on tape) is sensed, the Read is completed and the computer refers to the Program register to find the location the next instruction to be executed. This time the Operation Code register will hold a +, which indicates addition; the Information register a 6, which tells how many characters are to be added; the First register 1016, which is the address of the least significant digit of the augend; the Second register 1010, which is the address of the LSD of the addend; and the Program register 1520, which is the address of the next instruction to be performed once the addition is completed. When the addition

begins, the computer knows which characters to add together, because they are addressed by the first and second Address registers. It knows where to place the sum, because the computer will simply wipe out the augend with the sum. As each set of characters is added together, the First and Second Address registers will decrease, in order to address the next set and the Information register will also decrease by one in order to be able to tell the computer when the entire field has been processed.

The third instruction will write the information out to tape. The computer knows that it is to write, since the Operation register contains a W, which was addressed by the Program register. The other registers contain the tape unit (the 2 in the Information register), and the left and right hand ends of the sector of memory which contains the data to be transcribed to tape (the left hand address in the First Address register, and the right hand address in the Second Address register). The Program register, at this point, will hold 1530, so that it will be able to pick up the next instruction. Tape trunk 2 will be activated and the first character will be picked up from memory and sent to the tape unit. The First Address register will increase by one, and the second character will be written out to tape. This process will continue until the character addressed by the Second Address register has been written to tape, at which time the computer will again look to the Program register to know what instruction is to be executed next.

This time the Operation Code register will tell the computer to change the contents of the Program register (which holds at this point 1540) by transferring the contents of the First Address register to the Program register. This is all the computer must do to execute this instruction, but note that when it again refers to the Program register to determine where the next instruction is located, it is told to pick up the instruction at 1500, which will begin to repeat the process again.

The one point which must be remembered is that the tape which contains the data to be processed (as well as the output tape) has moved as a result of reading in (or writing out) the first data record. When we repeat this read, therefore, the second data record will be brought into memory and subsequently processed and written out to the output tape. When all the data records have been processed, we will have two reels of information: the original information and a reel of new, updated information.

IV – RCA 301 EQUIPMENT

When we discussed elements of a computer, we broke the description into five parts. Now that we are about to discuss the elements of a particular data processor, the RCA 301, we will follow the same format.

INPUT:

One of the features of the RCA 301 is that any individual customer may design his own system with regard to input-output requirements. Data may be fed into memory by use of:

1) Cards
2) Paper Tape
3) Magnetic Tape
4) Data Record File
5) Data Disc File
6) Interrogating Typewriter

There are two models of Card Reading equipment in the RCA 301 system.

The Card Reader has the ability to read and translate 80 column EAM cards up to a rate of 600 cards per minute. The reading takes place column by column, the data in each column being translated and placed into memory as it is read. If translation is not desired, depression of the BCT (Bypass Card Translation) button will allow binary reading of a card into memory. Two characters will be placed in high speed memory for each of the 80 columns on the card. Thus, 160 high speed memory locations would be necessary. Accuracy is maintained by two reading stations which read the data twice and compare it by hole count. If a 600 card-per-minute rate is maintained, there is approximately 20 ms of free computing time between cards. The Card Reader can be instructed to maintain a 300 card-per-minute rate, resulting in 120 ms of free computing time between cards. Only one Card Reader can be attached to an RCA 301 system.

The Card Reader/Punch has the ability to read and translate 80-column EAM cards up to a rate of 800 cards per minute. The reading takes place row by row, the data in each column being translated to RCA 301 code and placed into memory as it is read. If translation is not desired, it is possible to obtain a representation of this card data by placing 12 information bits (two characters) in memory to represent each column. This is instruction controlled. Thus, 160 HSM locations would be needed for the read-in area. Accuracy is maintained by two reading stations which read the data twice and compare it by hole count. If an 800 card-per-minute rate is maintained, there is approximately 10 milliseconds of free computing time available per card cycle. When the read-release feature is programmed, an additional 21 milliseconds of free computing time is available per card cycle. The card read feed hopper holds 3000 cards. Three of the five 1000 card capacity output stackers are subject to program control. Only one Card Reader/Punch can be attached to a 301 system.

The paper tape reading equipment includes the Paper Tape Reader/Punch and the Paper Tape Reader.

The Paper Tape Reader/Punch is mounted on one base and can both read and punch at a rate of 100 characters per second. This unit will read and punch either 5 channel or 7 channel paper tape. Packing density is 10 characters per inch and tape speed is approximately 10 inches per second. The reader stops on a character and is positioned to read the next character.

The Paper Tape Reader is capable of reading either 500 or 1000 characters per second. It will accept 5, 6, or 7 level paper tape depending on a switch setting.

At the 500 character per second rate, the Reader stops on a character and is positioned to read the next character. When reading at the 1000 character per second rate, the Reader stops in a position to read a character at a maximum of 0.3 inch (three sprocket holes) from the last character read. Packing density is 10 characters per inch and the tape moves at either 50 or 100 inches per second.

Magnetic tape is read by a device called a tape station or tape deck. It is possible to have several different types of magnetic tape.

The first is the High Data Tape Group. The High Data Tape Group is composed of a cluster of six tape stations. Up to two clusters can be attached to a 301 system. Two 8" tape reels are mounted on each station; one a supply reel and the other a take up reel. Each reel accomodates 1230 feet of 1/2 inch wide oxide coated plastic tape of which 1200 feet are usable. Each tape station can be read in a forward or reverse direction or written to in a forward direction. Since packing density is 333.3 characters per inch and tape speed is 30" per second, the tape station reads and writes at a rate of 10,000 characters per second. Once a tape has been processed, it can be rewound at a rate of 90" per second. Information on these tapes is separated by gaps. These are approximately 0.34 inches in length. Only one tape station in each cluster can be addressed at one time since only one electronic unit services all six of the tape stations in the cluster.

Another type of tape unit which is available with an RCA 301 System is a high speed tape station. There are two models available - 33KC and 66KC.

The 33KC tape station records data at a density of 333 characters per inch and the tape speed is 100 inches per second. Gap size is a minimum of .45 inches. 33KC tape stations may be incorporated in an RCA 301 System through Tape Adapters, Dual Tape Channels or both. The Tape Adapter connects a single tape station to the Computer and an RCA 301 System can have a maximum of two. Two models of the Dual Tape Channel are available which will each operate up to six or up to twelve tape stations respectively. A maximum of fourteen 33KC tape stations may be attached to an RCA 301 System.

The 66KC tape station records data at a density of 667 characters per inch and the tape speed is 100 inches per second. Gap size is a minimum of .55 inches. 66KC tape stations may be incorporated in an RCA 301 System through Tape Adapters, Dual Tape Channels or both. The Tape Adapter connects a single tape station to the computer and and RCA 301 System can have a maximum of two. Two models of the Dual Tape Channel are available which will each operate up to six or up to twelve tape stations respectively. A maximum of fourteen 66KC tape stations may be attached to an RCA 301 System.

Both the 33KC and the 66KC tape stations use 3/4" oxide plastic tape. A tape reel contains 2400 feet of tape, 2300 feet of which are usable. All characters on these tapes are dually recorded (Each 7 bit character is recorded twice, side by side, across the width of the tape.) When a character is read, both sets of 7 bits are overlayed and the resultant character placed in memory. This procedure, used in addition to parity checking, greatly cuts down the number of computer stops due to the inability to read a character, without cutting down the accuracy of the information.

The Data Record File contains 128 magnetic coated records which can have information recorded on both sides (up to 4 1/2 million characters). The surface of every record has two bands each containing 10 cells of 900 characters each. Information is recorded in serial fashion in a spiral pattern around the record. Characters may be transferred from the Data Record File to HSM in blocks of from one to ten cells at a time. The rate of transfer is 2,500 characters per second. Up to six Data Record Files may be used in an RCA 301 system. Parity is checked on the information to be written on the record and on the address of the data to be selected.

The Data Disc File can consist of from one to four modules. Each module consists of six data discs with recording on both sides. There are 9 zones of 128 tracks on each disc surface, and each track contains 10 sectors of 160 characters. The total capacity of one module is 22,118,400 alphanumeric characters. One instruction can be used to transfer blocks of from one to ten sectors from the Data Disc File and HSM. When the strata concept of data organization is utilized, associated records of data stored within related tracks of each zone on each disc surface are accessible to a single file positioning. There are 108 related tracks per module for each strata.

Information is recorded bit-serially in each track. Each zone has one read-write head serving 128 tracks. All read-write heads in a file are moved simultaneously so that when an arm is positioned to a particular track in a particular zone, all arms are positioned to the corresponding track within their respective zones.

A Data Disc File can consist of from one to four modules. Two Files can be used in an RCA 301 System giving a maximum capacity of 176,947,200 characters in steps of 22,118,400 characters. If two Files exist in the system, both can be transferring data to or from the High Speed Memory at the same time, one in the Normal Mode and one in the Simultaneous Mode.

A parity check is performed on the information to be written on the disc and on the information read from the disc. A check is made to determine that the positioner is on the correct track of the disc. Conversion from bit-serial to character-serial operations are checked by a bit counter.

The Interrogating Typewriter, under Program Control, permits an operator to enter information via its' keyboard to the RCA 301, and to receive typed information from the computer. Programmed routines process the interrogations and information can be extracted from either magnetic tapes or storage or can be computed by subroutines. Data transmission speed is up to 10 characters per second.

On the output side of the picture we find:

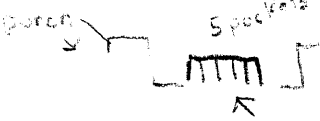
1) Cards
2) Paper Tape
3) Magnetic Tape
4) Data Record File
5) Data Disc File
6) Documents
7) Monitor Printer

Two models of Card Punching equipment are available in the RCA 301 system.

The Card Punch punches by row at a rate of 100 cards per minute. Accuracy is maintained by reading the card once it has been punched and comparing the hole count.

The Card Reader/Punch has the ability to punch cards either binarily or in Hollerith code as designated by the computer program. Cards are punched at a rate of up to 250 cards per minute. Accuracy is maintained by reading each card after it has been punched and performing a hole count check. There is a maximum of 22.5 milliseconds of computing time available per card cycle. When the punch-release feature is utilized, an additional 37 milliseconds of computing time is available. The punch feed hopper holds 1200 cards. Three of the five 1000 card capacity output stackers are subject to program control. Only one Card Reader/Punch can be attached to a 301 System.

The Paper Tape Reader/Punch and the Paper Tape Punch produce 5 or 7 level, even parity, punched paper tape at a rate of 100 characters per second. Packing density on tape is approximately 10 characters per inch and tape moves at approximately 10 inches per second. The punch stops in a position to punch a character a maximum of 0.3 inches (three sprocket holes) from the last character punched.



Magnetic Tape, the Data Record File and the Data Disc File have been previously discussed.

Output documents are prepared on the On-Line Printer. There are two models available in the RCA 301 System. The important differences between the two are 1) the number of characters that can be printed per line, and 2) the number of lines that can be printed per minute. One model prints a maximum of 120 characters per line. The printing rate is up to approximately 1000 lines per minute in the Synchronous Mode which permits the printing of 47 selected characters. In the Asynchronous Mode which permits the printing of 64 characters, the printing rate is up to approximately 800 lines per minute. The second model prints a maximum of 160 characters per line. The printing rate is up to approximately 1070 lines per minute in the Synchronous Mode and up to approximately 835 lines per minute in the Asynchronous Mode. The following characteristics are common to both printers:

1) The On-Line Printer is a transistorized device which prints data directly from the High Speed Memory to prepare output documents.

2) Data editing is accomplished by the computer via a stored program.

3) Paper advance is controlled by the computer program, either directly or through a tape loop in the Printer.

4) Ten characters are printed per horizontal inch and six lines per vertical inch.

5) Paper stock may be single or multiple sheet fanfold. One original, plus 5 carbon copies may be used. Hecto and multilith master stock may also be used.

In the Synchronous Mode, the On-Line Printer will print 26 letters of the English alphabet, the numerals (0 to 9), and the following 11 punctuation marks and symbols for a total of 47 available characters.

CR	credit	/	virgule		space
'	apostrophe	¤	lozenge	,	comma
*	asterisk	-	minus	.	period
&	ampersand	+	plus		

In the Asynchronous Mode, the On-Line Printer will print 26 letters of the English alphabet, the numerals (0 to 9), the above listed 11 punctuation marks and symbols, and the following 17 punctuation marks and symbols for a total of 64 available characters.

@	at the rate of	:	colon	$	dollar sign
%	percent	#	number	)	close parenthesis
"	quote or ditto	;	semicolon	[	open bracket
10	subscript$_{10}$	>	greater than	<	less than
(	open parenthesis	÷	divide	=	equal
]	close brackets	↑	up arrow		

A maximum of 2 On-Line Printers may be included in an RCA 301 System.

The Monitor Printer is a typewriter-like output device operated under program control. Characters may be printed at the rate of up to 10 characters per second. Ten characters are printed per inch. Paper may be single or multiple sheets and up to 17 inches wide. All the alphanumeric characters can be printed plus the following:

#	number	@	at the rate of	(	open parenthesis
)	close parenthesis	=	equal	:	colon
,	comma	%	percent	&	ampersand
EB	end block	;	semi-colon	EI	end information
$	dollar sign	"	quote or ditto	-	
.	period	•	ISS	+	plus
*	asterisk	ED	end data	EF	end information
/	virgule	-	minus		

A carriage return is effected by printing an RCA 301 "ED" or by setting tab stops.

Forces carriage return

STORAGE:

There are two basic types of storage in the RCA 301 System:

1) magnetic core memory
2) magnetic disc

The core memory has a minimum size of 10,000 locations. This may be increased to 20,000 or 40,000 character locations if desired. Each one of these character locations is individually addressable and the memory cycle (the time it takes to pull a character from memory and regenerate it) is 7 microseconds (7 millionths of a second).

Magnetic disc storage (Data Record File and Data Disc File) have been previously discussed.

ARITHMETIC:

The RCA 301 has the ability to do comparisons, allowing it to make decisions as to magnitude, and to add and subtract using a table look-up method which will be discussed in detail later in this manual. Logical instructions are also available. Multiplication and Division must be handled by program or RCA supplied subroutines.

V — THE BINARY NUMBERING SYSTEM

The Binary Numbering System can be thought of as the language by which the equipment in the RCA 301 System communicates with each other and processes the data internally.

A numbering system is simply an orderly system of marks, used for making quantitative measurements, which are controlled by a basic set of rules. The most familiar system is the decimal system, which uses the ten marks (or Arabic Numerals) 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

The first thing a person needs to know when working with a numbering system is its "base". This term refers to the number of marks used in a particular system. The decimal system operates on a base of 10; i.e., there are 10 different marks in the system. The binary system has a base of 2. That is to say, there are only two different marks in this system, 0 and 1.

Having learned the base of a system, the next step is to learn to count. The three rules of progression are:

1) Knowing the sequence of marks, the rightmost number is advanced to the next number in sequence. (From 0 to 1, from 1 to 2, from 2 to 3, etc.)

2) If the rightmost number is the last mark used in the system, it is changed to the first mark and then we move to the next column to the left and advance this number one mark in the system. (From 29 to 30, from 79 to 80, etc.)

3) If this column to the left also used the last mark in the system, it is returned to the first mark and we again move one column to the left and advance one mark. (499 to 500, 999 to 1000, etc.)

Since these rules apply to all numbering systems, we can proceed to count binarily:

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
etc.

Another concept which should be clarified before proceeding is the actual meaning of a number. Considering the decimal system, which has a base of 10, we could really say that the number 39 is 10 + 10 + 10 + 9. Another way of saying this is $3(10^1) + 9(10^0)$. (Remember that any number raised to the zero power is equal to 1). In the same way, the number 589 is in reality $5(10^2) + 8(10^1) + 9(10^0) = 5(100) + 8(10) + 9(1)$. The same logic holds true in connection with the

binary system, except that here the base is 2. For example, the binary number 101111 could be thought of as:

$$1(2^5) + 0(2^4) + 1(2^3) + 1(2^2) + 1(2^1) + 1(2^0).$$

Figuring this out decimally would actually give us the decimal equivalent of the binary number:

$$(101111)_2 = 32 + 0 + 8 + 4 + 2 + 1 = (47)_{10}$$

In summation, the meaning of a number, regardless of its base, is a succession of columns, where each column to the left is one power of the base higher.

BINARY CODE:

As has been stated previously the binary code is a system using only two marks. For convenience we use the Arabic Numerals 0 to 1 as the two marks. The "base" of this system is 2.

It was stated that this binary system is the language of the machines in the RCA 301 system. What does this actually mean? If we forget our convenience in using the Arabic Numerals 0 and 1 as the binary marks we can visualize the binary code internally in the machines as: no (0) a wire is not carrying an electrical pulse or yes (1) a wire is carrying an electrical pulse; yes (1) a gate is opened or no (0) a gate is closed; yes (1) a core is magnetized or no (0) a core is not magnetized. In effect any two way condition can be treated in binary code as a 0 or a 1.

Now because man does not think directly in binary, as he does in decimal, these are a few methods provided for his use. You cannot glance at the binary number $101101101_{(2)}$ and appreciate its relative magnitude, unless you can convert these binary marks to a decimal equivalent. Likewise the decimal number 789 is not readily converted mentally to its binary equivalent without the aid of a tool. There are various auxiliary rules to convert decimal to binary number and conversely.

a. Decimal to Binary:

Take the decimal number $329_{(10)}$ and convert it to its binary equivalent. The method is to successively divide by two and retain the remainders of each division. The successive list of remainders is the binary equivalent.

Example:		Remainders
	0 - 1	1
2	1 - 0	0
2	2 - 1	1
2	5 - 0	0
2	10 - 0	0
2	20 - 1	1
2	41 - 0	0
2	82 - 0	0
2	164 - 1	1
2	329	

Starting from the last remainder the binary equivalent of $329_{(10)} = 101001001_{(2)}$

b. Binary to Decimal:

This method of conversion merely follows the earlier discussion of the column values. In binary our base of two makes each column a value of two to a successively higher power.

Example:

$101001001_{(2)}$ from the previous example.

$2^8\ 2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
1 0 1 0 0 1 0 0 1

Only those columns with a one bit present are needed in the computation of the decimal equivalent.

$2^0 = 1 \times 1 = 1$
$2^3 = 8 \times 1 = 8$
$2^6 = 64 \times 1 = 64$
$2^8 = 256 \times 1 = 256$
329

c. Addition in Binary

There are only a few basic rules to remember in becoming proficient at addition of binary numbers. These rules are:

Rule 1. 0 + 0 = 0
Rule 2. 0 + 1 = 1
Rule 3. 1 + 1 = 10 (a 0 in the right column & a carry to the left.)
Rule 4. 1 + 1 + 1 = 11 (a 1 in the right column & a carry to the left.)

Example:

$1011011 = 91_{(10)}$
$0010001 = 17_{(10)}$
$1101100 = 108_{(10)}$

d. Subtraction in Binary:

Although there are several methods available for subtraction in binary the one presented here closely resembles the method used by a computer. The first requirement is an understanding of a method of "complementing" the subtrahend. This is done by changing all the 0's to 1's and all the 1's to 0's. If the subtrahend in a binary subtraction is 1001101 the "complement" of this number would be 0110010. These are the rules for subtraction of binary numbers.

1) Complement the subtrahend.
2) Add the subtrahend (complemented) to the minuend.
3) If there is a carry from the most significant digit, this indicates a positive number and the carry must be added to the least significant digit.
4) If there is no carry, this indicates a negative number and the answer must be complemented to obtain the correct result.

Example:

1) From 1011011 subtract 0110011

$1011011 = 91_{(10)}$
$0110011 = 51_{(10)}$
$40_{(10)}$

a) complement the subtrahend and the problem appears as follows:

1011011
1001100

b) adding these we obtain:

1 0100111

c) the carry indicates a positive answer and must be added to the result

0100111
1
0101000 = $(40)_{10}$

2) from 0011011 subtract 1001100

0011011 = 27 10
1001100 = 76 10
-49 10

a) complementing the subtrahend we have 0110011

b) adding we have

0011011
0110011
1001110

c) the lack of a carry indicates a negative result, so the answer must be complemented giving 0110001 $(49)_{10}$

With very few exceptions, we will not be concerned with the binary arithmetic that the computer is executing. This previous discussion, however, has given us a feeling of what is involved, which will prove useful as the course develops.

CLASS EXERCISE ON NUMBERING SYSTEMS

A. Convert the following decimal numbers to binary:

1. 24
2. 26
3. 47
4. 86
5. 104

B. Convert the following binary numbers to decimal:

1. 1011
2. 10100
3. 101101
4. 011100
5. 111111

C. Add the following binary numbers.

1. 101
   010

2. 0101
   1010

3. 11101
   11001

4. 11111
   10101

5. 10011
   01111

D. Subtract the following binary numbers:

1. 111
   001

2. 1001
   100

3. 11001
   00111

4. 0111
   1000

5. 11111
   10101

VI — DATA LAYOUT

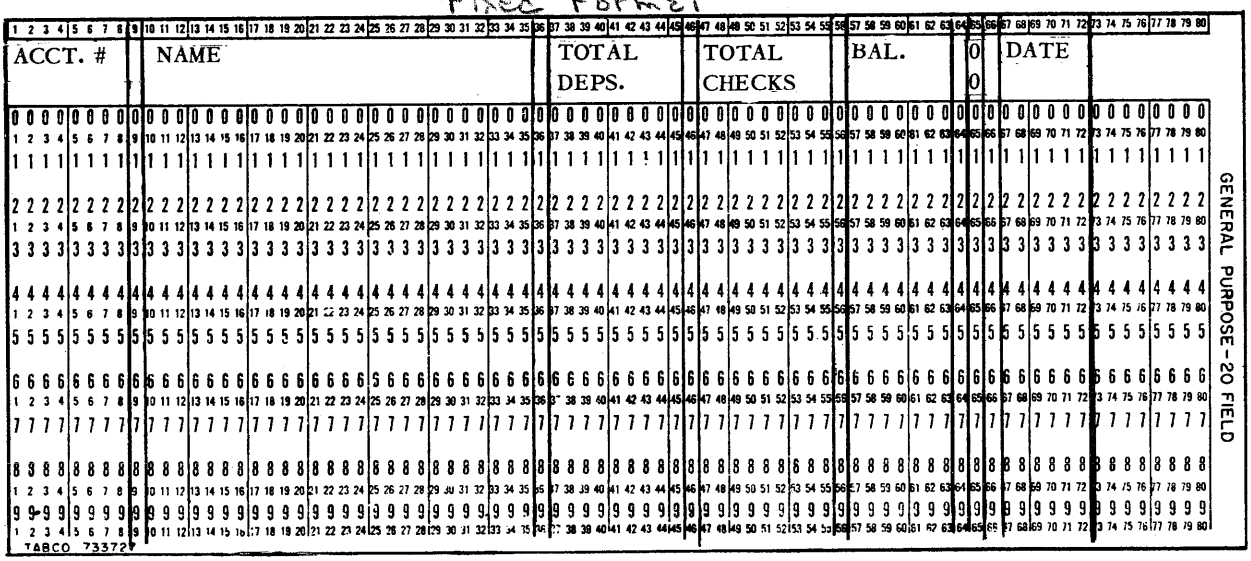
Data Layout is different depending on the various types of input-output media. In order to completely understand data layout it will be necessary to discuss each type.

CARDS:

As was mentioned in the introduction to computers, information is punched into these cards using a binary code. Each column represents one character, but since, in most cases, more than one character is needed to indicate some useful data, a number of adjacent columns will contain a "field" of information. For example, on one set of cards we find the following information:

Columns	Data
1-8	account number
10-35	name
37-45	total deposits
47-55	total checks
57-63	balance
65	one character code indicating overdraft if needed
67-72	date

Fixed Format



This same format will be followed by each card, and therefore the format is fixed. That is to say, 26 locations have been left for a name. If any one name should have fewer than 26 characters, it would be necessary to carry blank columns to fill out the field. The balance has a 7 character field, so insignificant zeros will be needed. For example, a balance of 64324 (read $643.24 although no dollar or decimal symbols are carried) would appear as 0064324. Although the length of each field is fixed, the different fields vary in size. For this reason, we say that the information is in "fixed-variable" format.

A card consists of 80 columns and 12 rows. We can further break down this description by mentioning that the rows are either numeric (rows 0-9) or zone (X or 11 and Y or 12) where 0 also acts as a zone. This type of set-up is necessary to allow all characters to be expressed. When discussing any decimal digit, a hole in the proper row would be sufficient. To express the alphabetic characters, however, we must combine holes. For example, the letters A-I are represented by the same 1-9 rows all of which have been overpunched (given a zone punch) with a "Y" or "12" punch (top row). Thus an "A" is a "Y" and a "1"; a "B" is a "Y" and a "2"; etc. J-R also utilize the numbers 1-9 but with a zone punch of "X" or "11" (second row). The remaining letters, S-Z, use the digits 2-9 with 0 acting as an overpunch. The punctuation and special symbols are also shown in this manner, some even having 3 punches (a period is represented by a Y, a 3 and an 8).

Looking at our example another though occurs. How would we show a negative balance? The card code indicates that a minus is shown by an "X" punch by itself. By overpunching this minus over the least significant digit of the balance we can then show a negative balance. For example, if our balance was a minus $5.92, it would appear on the card as 000059K, since a 2 punch in combination with an X punch would indicate a minus 2 for operational purposes.

In many cases, a number of punches may be placed in a column. These might not have a legitimate translation when referring to a code sheet, but may have a significant meaning to the particular installation in which it is used. An extreme example would be that one column were used to store the answer to 12 bi-state questions (Male or Female, Married or Single, etc.)

At this point, it seems advisable to define some frequently used terms:

bit: a bit is single binary digit expressed as either a "0" or a "1"

character: a character in the RCA 301 system is made up of 7 bits. These consist of 6 information bits and 1 parity bit. The 6 information bits are the unique binary configurations used to indicate the individual numerics, alphabetics, and special characters. The parity bit is used for accuracy purposes. In the RCA 301 machine code, we say that parity is "odd", that is the number of 1 bits in a character must be odd. If a bit is gained or lost, therefore, the parity will be wrong and this will cause the computer to register a parity alarm so that the character can be corrected. For example, the decimal number 0 is known to the computer as 000000. Since there are 6 zero bits in this character, parity must be a 1; the whole character would be 1000000. The letter A is 1010001. The 010001 are the information bits, the left hand 1 is the parity character. The bit positions are called:

ZONE		NUMERIC			
2^5	2^4	2^3	2^2	2^1	2^0

The RCA 301 code was developed to allow the easiest conversion of card code to machine code, in an attempt to keep cost and engineering hardware at a minimum.

item: or Field — Fixed No of Positions

one character seldom indicates enough data, so it is necessary to combine a number of characters to give one pertinent piece of information. An example would be a name, or another one might be a stock number. This may or may not be preceded by a special symbol to indicate the beginning. This symbol is used primarily when the data is variable, a term which will be discussed as a separate topic. In addition, an item could contain sub-items; for example an employee number might consist of a department number, a section number and a man number.

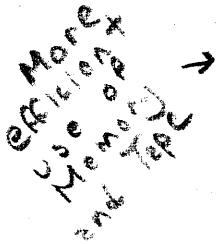
Character + Character Data = Item
Item + Item (Related) = record
record + record = File

record: a record consists of a number of related items. For example, all the data pertaining to John Doe's savings account would be one record; in addition, all the records referring to savings accounts would be one file. Another example would be all the records pertaining to the checking accounts. All records in a file must have the same format. A file is terminated by a single control symbol called an EF (End of File). When this is sensed, it means that all the records in the file have been processed.

variability: data may be presented in three possible formats:

1) fixed Reserve certain Positions
2) fixed-variable
3) variable

If data must be fixed, it means that the computer on which it will be processed has a fixed word length. For example, a computer with a 10 character word means that instead of being able to address each character, it is necessary to address 10 characters at a time. If data is fixed-variable, it means that the same item in each record must have a fixed length, but that different items may vary in length. The card has a fixed-variable format.



A variable system is utilized to eliminate the need to carry unnecessary characters (such as spaces or insignificant zeros) on tape, which would result in decreased input-output times. It allows the data to be carried at its significant length. For example, if a name is John Doe, that is all that is carried. A balance of 63457 would appear just like that, without any insignificant zeros.

The RCA 301 has the ability to handle data as fixed variable or as variable.

The following example indicates the difference. Note the amount of space that can be saved utilizing a variable format.

account number	8 characters maximum
name	25 characters maximum

fixed: (10 character word length)

0012345678JOE_DOE_____0012345679BOB_SMITH_____001

fixed variable:

12345678JOE_DOE_____12345679BOB_SMITH_____12345680JOHN_JONE

variable: (symbol * needed to differentiate between items)

*12345678*JOE_DOE*12345679*BOB_SMITH*12345680*JOHN_JONES*12345681*JACK_BROWN*123456

block: a block is any number of characters (minimum 3) which appear on tape between two gaps. A single record surrounded by gaps would therefore be a block. Two records placed end to end, with a gap preceding and a gap following, would be a block. An EF (which must be preceded and followed by a gap) is a legitimate one character block, as is an ED.

ED symbol: The ED is a control symbol which serves a function similar to the EF symbol. It indicates that the end of good data on this reel of tape has been reached, but that the file has not been completely processed.

For example, a file might take up two reels of tape. The first reel would have an ED as the last block; the last block on the second reel (appearing immediately after the last record) would be an EF. If there were no other data on the reel, the block after the EF would be an ED. It is also possible to have two files on one reel. The first file would be terminated by an EF, the second one by an EF followed by an ED to indicate the end of good data on the tape.

PAPER TAPE AND MAGNETIC TAPE:

Data on tape is also coded binarily; on paper tape, no holes represent the ones, holes the zeros; on magnetic tape, magnetic spots indicate the zeros, the lack of these spots shows ones. The information is separated by gaps. On paper tape this gap must be a minimum of three character locations; on magnetic tape it is approximately .34 inch.

Information on RCA 301 10kc Magnetic and paper tape is coded as the complement of the character in memory. For example, the letter "A" is represented in memory as $(1010001)_2$ but on tape it would be $(0101110)_2$. The reason behind this is two fold. To give the best possible accuracy check, information on tape should have even parity. This is because odd parity would permit a character to be written on tape with only one 1 bit, and if this 1 bit were lost, the character would become a gap. (On 33kc tapes, parity is odd but the dual recording eliminates the difficulty of losing a bit and not being able to read the character.) With even parity, however, it is very unlikely that the two 1 bits (which would be the minimum amount in any one character) would both be lost at the same time, so that even if one were lost, parity would be incorrect and the computer would halt on a parity error. However, if even parity is maintained, the decimal zero which has a code of $(000000)_2$ would itself be a gap. To prevent this from occurring, the entire character is complemented when being written to tape from memory or being read to memory from tape.

TOTALS AND WEIGHTED AVERAGES:

It was mentioned that every record in a file must have the same format. Therefore, by setting down the description of a file, we are in effect describing each record in the file. If the file is variable, we must know more than the maximum number of characters in each item; we must also know the average number of characters and the percentage of occurance of these items. By multiplying these two factors together, we have the weighted average number of characters in each item. This is important, primarily for timing purposes. Figure VI-I illustrates a DATA SHEET describing a variable file. Since it is necessary to carry the separating symbol for positional purposes, unless there is no data after it for the remainder of the record, it is necessary to add in these symbols. A file should normally be arranged by decreasing percentages of occurances, therefore, it may not be necessary to carry all of the separating control symbols. For this reason, we must determine how many are necessary and we do this by employing the probability of occurrence. An example would be best to illustrate this method:

ITEM NO.		% OF OCCURRENCE	ACCUMULATED % OF OCCURRENCE
1		100	
2		90	
3		75	
4		60	
5	(breakpoint item)	40 (40 + 59)	99
6		25 (25 + 34)	59
7		20 (20 + 14)	34
8		10 (10 + 4)	14
9		3 (3 + 1)	4
10		1	1
			211

Starting at the last item, figure the accumulated % of occurrence. Continue to do this until you reach the breakpoint item. This is the item whose accumulated % of occurrence is closest to, but less than 100%. Totaling the accumulated % of occurrences you will get a percentage of occurrence for the control characters, in this case 211%. This means that 2.11 control characters must be added to the 4 which will always appear (for the first four items) giving a grand total of 6.11 control symbols. Having developed the number of control symbols, add this amount to the sum of the weighted average of each item and you will have the total weighted average of characters in each record.

EXERCISE I

Organization of Data

1. Define the following:

   a) **Record -**

   b) Block -

   c) Item -

   d) Control Symbol -

   e) File -

2. The master inventory file is a three reel file. What controls symbols terminate each reel?

3. A record contains the following information:

	Number of Characters Max.	Avg.	Information
Stock Number	5	5	12345
Stock Description	20	10	GENERATOR
Balance on Hand	7	5	1000
Unit Cost	5	3	3275

E.D.P. DATA SHEET

	INPUT		OUTPUT	IDENTIFICATION	DATE
X	REFERENCE		INTERMEDIATE	STOCK MASTER FILE	60-10-6

SOURCE

USE: MASTER FILE

SCHEDULE

SEQUENCE: ASCENDING ORDER BY STOCK NO.

DISPOSITION

NO. OF MESSAGES: PEAK: / NORMAL:

DOCUMENT DESCRIPTION:

TYPE: / SIZE: WIDE X LONG / NO. OF COPIES

REMARKS

ITEM NO.	SUB. NO.	DESCRIPTION	NO. OF CHARS. MAX.	NO. OF CHARS. AVG.	% OCCURRANCE	WTD. AVG.
1		STOCK NUMBER	9	9	100	9
2		STOCK DESCRIPTION	25	14	100	14
3		BALANCE ON HAND	10	8	100	8
4		TOTAL ISSUED TO DATE	15	12	100	12
5		REORDER LEVEL	10	4	100	4
6		AMOUNT DUE IN	8	6	50	3
7		REJECTIONS PERCENTAGE	3	2	25	.50
8		OBSOLETE CODE	1	1	5	.05
		TOTAL CHARACTERS OF INFORMATION	81			50.55
		ADDED CONTROL CHARACTERS	8			6.15
		TOTAL RCA CHARACTERS	89			56.70

IE 24

a) Compose this information in a fixed length record. (Fixed word length 10 characters.)

b) Compose this information in a fixed-variable length record.

c) Compose this information in 301 variable.

4. Construct a data sheet for the following payroll file. There are approximately 10,000 messages.

Description	No. of Characters Max.	Avg.	% Use
Pay Period	2	2	100
Man Number	5	5	100
Tax Class	1	1	100
Year to Date Earnings	6	5	100
Year to Date Withholdings	6	4	100
Quarterly FICA	4	4	100
Gross Pay	5	4	100
Deductions			
Bonds	4	3	75
Combined Charities	3	2	50
Major Medical Insurance	3	2	35
Automobile Insurance	4	3	25

EXERCISE II

Create a data sheet for a payroll master file. The file is currently on ledger cards and contains the following information: (Maximum and average lengths have been determined). Assume you are setting up this tape file and may choose any sequence for each item.

	Maximum	Average
Name	20	12
Address	20	15
City	20	15
Telephone Number	7	7
Job Title	30	15
Job Classification Number	4	4
Employee Number	5	5
Department Number	2	2
YTD Municipal Wage Tax	5	4
Health Insurance	5	4
Automobile Insurance	5	4
Stock Ded.	4	4
YTD Gross	7	6
Quarterly FICA	5	4
YTD Withholding	6	5

The criteria for sorting will be department number. Within each department number, messages will be in order by employee number.

500 employees live in the city and must pay the Municipal Wage Tax; 2000 employees have health insurance deductions; 1500 also have automobile insurance; 1000 have Stock Deductions.

There are 10,000 employees.

VII — HIGH SPEED MEMORY

10K 20K 40K

Memory may contain either 10,000, 20,000, or 40,000 character locations. Since each character consists of 7 bits, each location must be made up of 7 cores, one core representing each bit. For this reason, we can say that memory is made up of 70,000, 140,000, or 280,000 magnetic cores. These are held in 14, 28, or 56 matrices of 50 x 100 cores.

Discussing the first 10,000 locations first, it is obvious that each location could be given an individual address, if we started counting at 0000 and continued to 9999. The problem of addressing does not exist, therefore, until we desire to address something in the second 10,000 character locations. To prevent having to add a fifth digit to the address, we simply take the most significant digit of the address and add a zone bit in 2^4. For example, if we wanted to address the location 10,000 character locations away from location 7854, we would simply take the 7 (the most significant digit), break it down to its binary code of 000111, and place a bit in the 2^4 position yielding 010111. This character is a G, so the address is G854. To address the third 10,000 locations, we take the MSD of the address and add a zone bit in 2^5 position. To address the fourth 10,000 locations, we add zone bits in 2^4 and 2^5 positions of the MSD of the address. The following table will simplify this process:

	2nd 10,000	3rd 10,000	4th 10,000
0 changes to	&	- (minus)	" (quotes)
1 changes to	A	J	/
2 changes to	B	K	S
3 changes to	C	L	T
4 changes to	D	M	U
5 changes to	E	N	V
6 changes to	F	O	W
7 changes to	G	P	X
8 changes to	H	Q	Y
9 changes to	I	R	Z

The term "diad" will be referred to throughout the text. A diad is two consecutive HSM locations with an even address for the first location and an odd address for the second.

For illustrative purposes, we will use our original problem of adding the interest to the balance and indicate how the Interest Account Masters would appear in High Speed Memory.

E.D.P. DATA SHEET

	INPUT		OUTPUT	IDENTIFICATION	DATE
	REFERENCE		INTERMEDIATE	INTEREST ACCOUNT MASTERS	

SOURCE	USE	SCHEDULE

SEQUENCE	DISPOSITION	NO. OF MESSAGES
		PEAK: / NORMAL:

DOCUMENT DESCRIPTION:

TYPE: / SIZE: WIDE X LONG / NO. OF COPIES

REMARKS

ITEM NO.	SUB. NO.	DESCRIPTION	NO. OF CHARS. MAX.	NO. OF CHARS. AVG.	% OCCURRANCE	WTD. AVG.
1		ACCOUNT NUMBER	5	5	100	5
2		INTEREST AMOUNT	6	6	100	6
3		BALANCE	6	6	100	6
		TOTAL CHARACTERS OF INFORMATION	17			17
		ADDED CONTROL CHARACTERS	-			-
		TOTAL RCA CHARACTERS	17			17

	00 01 02 03 04	05 06 07 08 09	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	
30	← ACCT# →	← INT →	← BAL →								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
31	PROGRAM →										
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
32											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
33											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
34											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
35											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
36											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
37											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
38											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
39											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 8-60

VIII – INSTRUCTION FORMAT AND COMPUTER PROGRAM RECORD

The RCA 301 is a two address computer. This means that the individual instructions allow for two addresses. Due to the construction of the instruction, it is possible to put in more than two pieces of information if necessary.

The format of the RCA 301 instruction is as follows:

1) a one character operation code that tells the computer what to do
2) a one character information bit, called "N", which gives various data depending upon the instruction
3) two four character addresses called A and B

In summary, we have:

OP	N	$A_0A_1A_2A_3$	$B_0B_1B_2B_3$
X	X	X X X X	X X X X

An example instruction would look as follows:

N 4 1515 1689

This tells the computer to transfer 4 characters starting at location 1515 to location 1689, moving right to left.

This instruction conventionally is placed in memory so that its operation code falls in a location whose address is a multiple of 10, for example 1750. N would then fall in 1751, the A address in 1752-1755 and the B address in 1756-1759.

To aid in the writing of programs, a standard form is available for coding. This form is called the RCA 301 COMPUTER PROGRAM RECORD. On it there is room for the program title, the coder's name, the date, remarks, and page notation. In the body of the form there are a number of columns, a few of which we will discuss now, the remainder to be mentioned later in the text when they become important to us.

The instruction itself is written in the columns marked OP, N, A and B. Note that each position is numbered 0-9, and this refers to the address of the HSM location which is to hold that particular character of the instruction.

The HSM LOCATION column should contain the address which is to receive the operation code of each instruction. Again note that the "0" has been pre-printed for programmer convenience.

The REMARKS column is for programmer use and should always be filled in to give an English explanation of what that particular instruction is doing.

The BOX NO. column refers to the number given to a box on the flow chart, (a picture statement of the problem discussed in another chapter) that pertains to this particular instruction.

The FROM INST. LOC. (from instruction location) box must contain the address(es) of any instruction(s) that transferred to this particular instruction. For example, in our initial interest problem, the last instruction transferred the program back to the Read instruction. The address of the transfer instruction would then be entered in the From Inst. Loc. line of the Read instruction.

Pretending for the moment that the instructions we discussed as example instructions were, in truth, RCA 301 instruction, the Program Record sheet would appear as follows:

TITLE INTEREST PROGRAM
CODER L. BAILEY
REMARKS PROGRAM RECORD EXAMPLE

DATE 10-6-60
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
1530	~~16~~	150 0	0	0	R	1	1	0	0	0	0	0	0	0		READ RECORD FROM 1	1
		1 0	0	0	+	6	1	0	1	6	1	0	1	0		ADD INTEREST TO BALANCE	2
		2 0	0	0	W	2	1	0	0	0	1	0	1	6		WRITE UP-DATED RECORD TO 2	3
		3 0	0	0	1	0	1	5	0	0	0	0	0	0		TRANSFER PROGRAM BACK TO READ	4
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

VIII-3

IE 1224 REV. 8-60

IX — OPERATING LOGIC

When the program has been written, prepared as input and read into memory, how does the computer operate on it? As in the case of human beings, the computer has many "memory devices" called registers. These help it to keep track of what it is doing. A few of the most important ones are discussed below:

The OP register is a one character register that holds the operation code of the instruction being executed.

The N register is a one character register that holds the N character of the instruction being executed.

The A register is a four character register that holds the A address of the instruction being executed.

The B register is a four character register that holds the B address of the instruction being executed.

The P register is a four character register that keeps track of the program, by holding the address of the next instruction to be executed.

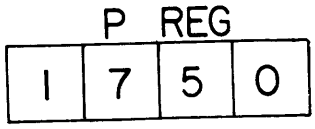
The Memory Addressing register is a four character register which addresses memory.

The Memory register is a two character register which holds the contents of the diad addressed by the MAR in order that they may be worked on.

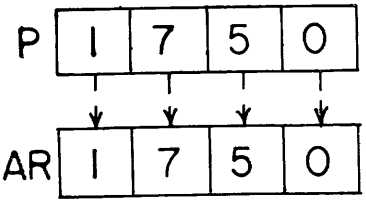
Bus Adder is a device which has the ability to increment or decrement the operating registers by 1 or 2.

Using our example (at location 1750 we have an instruction which reads N 4 1515 1689), let us investigate the operation.

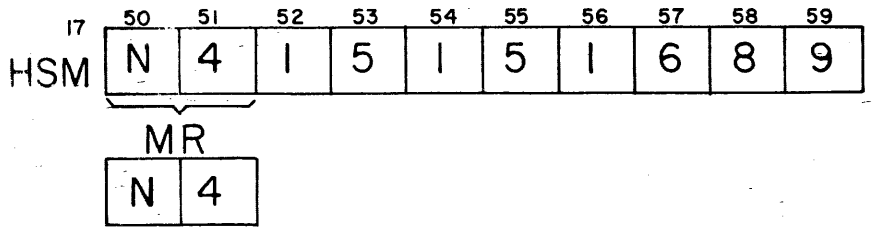
First of all, we must place the address of this instruction into the P register. We could do this from the console. Then, by hitting the start button on the console, the computer takes over.



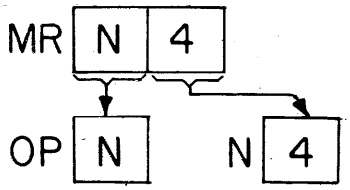
The contents of the P register fall into the MAR



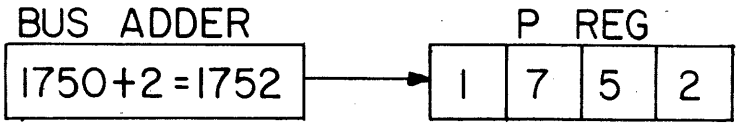
The diad 1750-1751 is pulled from memory and placed into the MR. At the same time the characters are regenerated so that they still appear at 1750 and 1751.



From the MR, these characters are placed in their respective registers; i.e., OP and N.

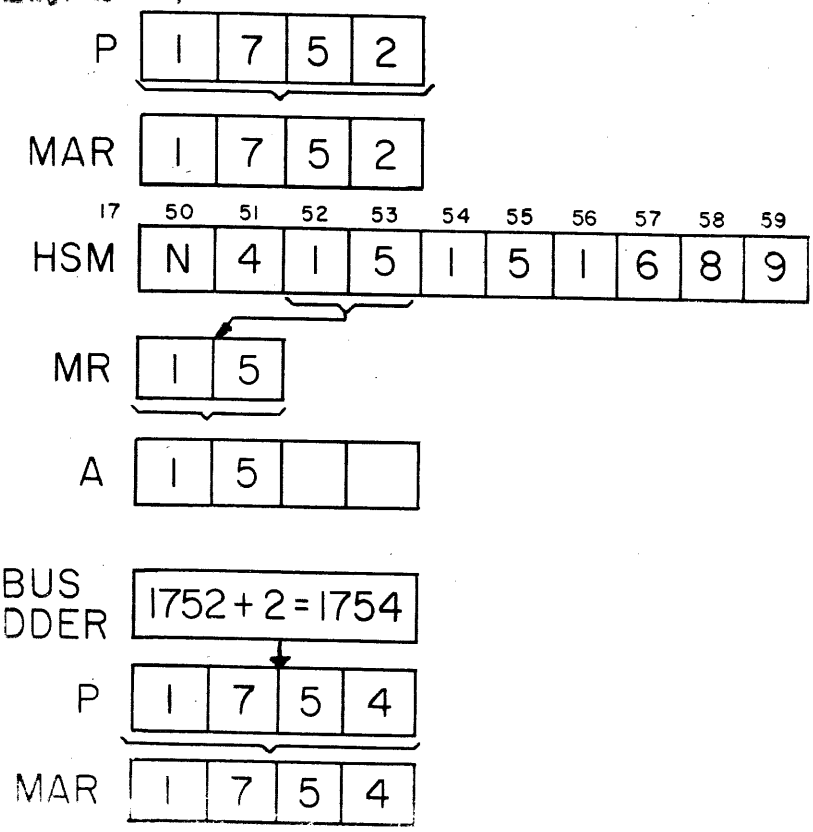


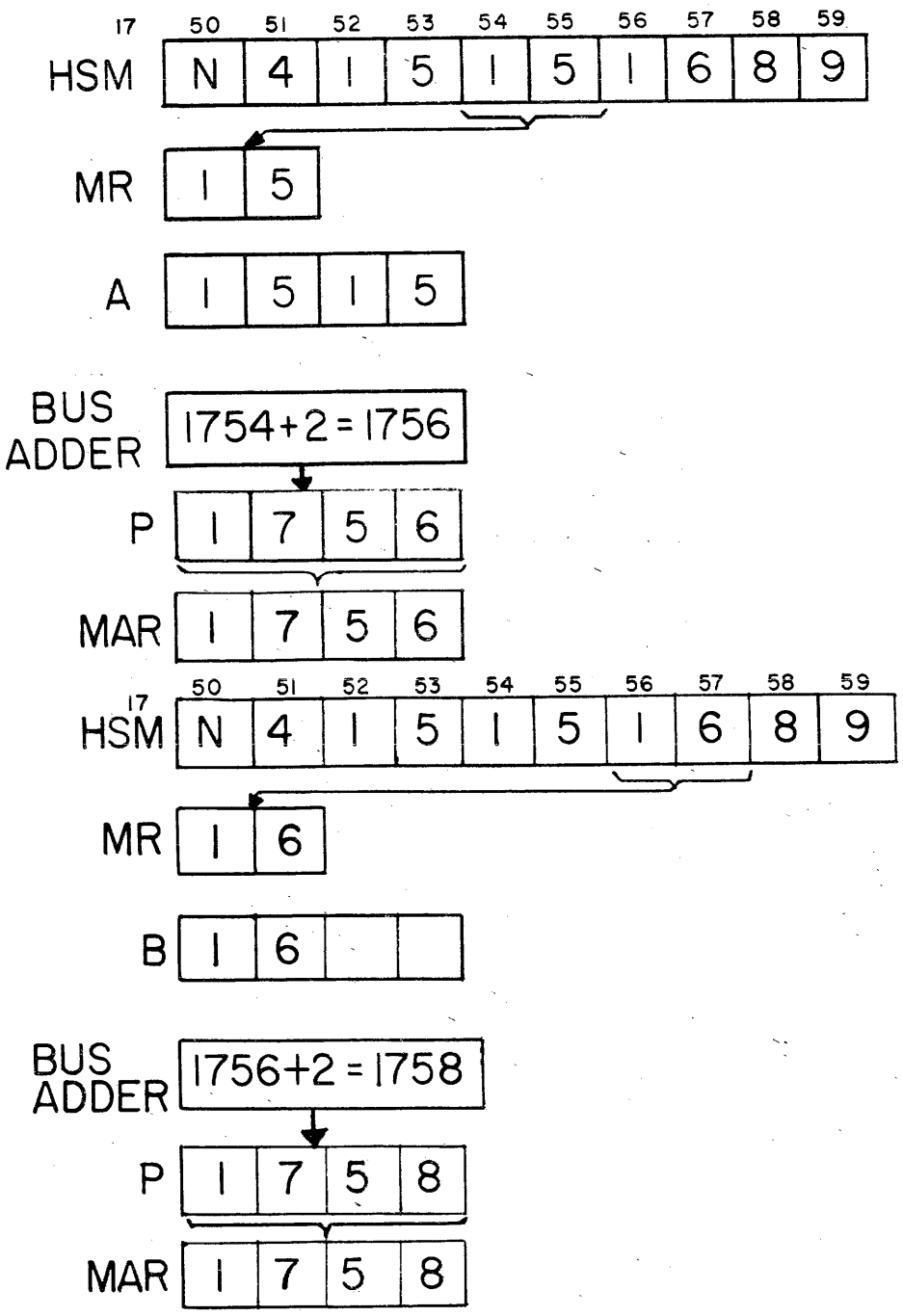
At the same time, the Bus Adder has added two to the P register contents.

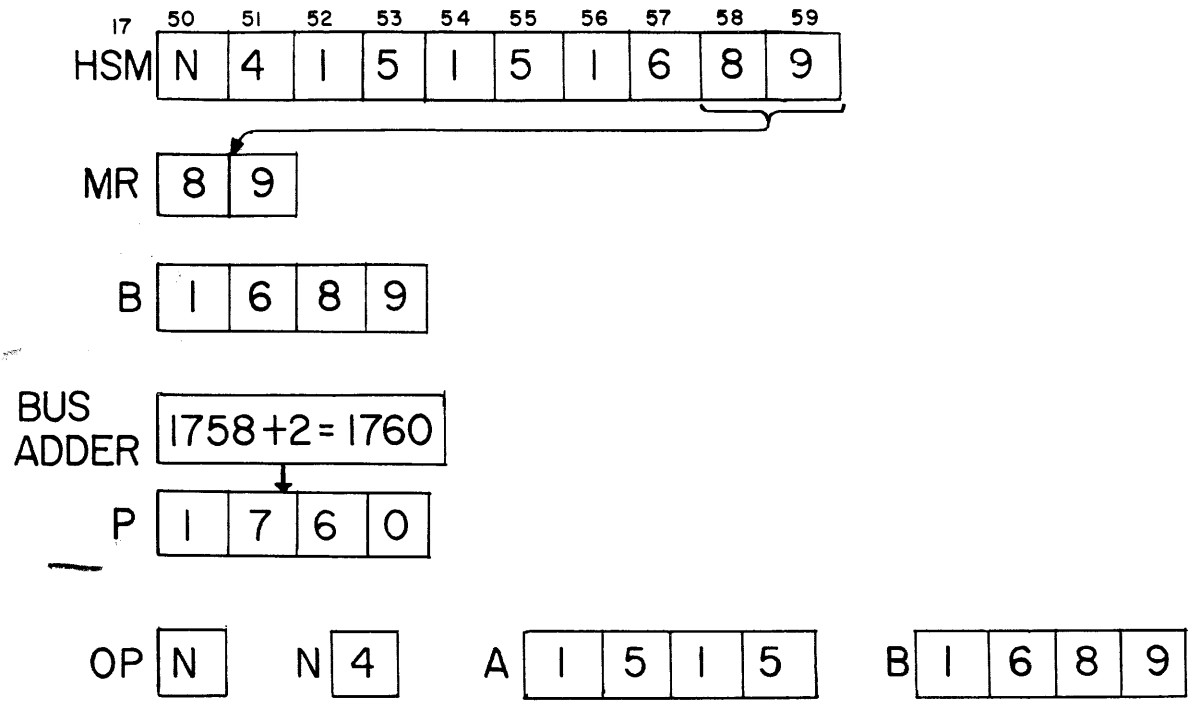


The process is repeated four more times, allowing the entire instruction to be staticized (to pull an instruction from memory and break it into its component parts, placing each in the appropriate register). The process itself takes one memory cycle (pulling the characters out and placing them in the registers). Since the process is executed five times, staticizing takes a total of 35 microseconds (35 millionths of a second).

5 cycles per each Instruction



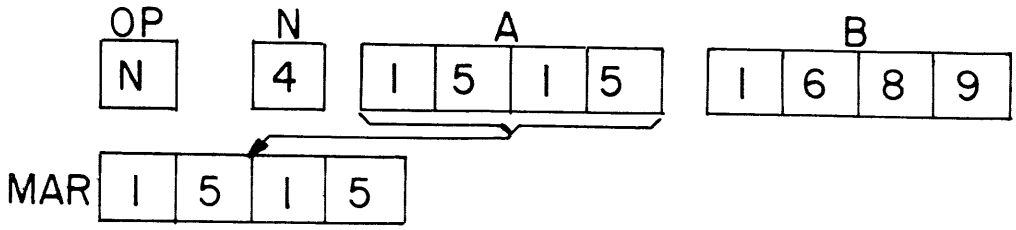

17
50 51 52 53 54 55 56 57 58 59
HSM N 4 1 5 1 5 1 6 8 9
MR 1 5
A 1 5 1 5
BUS ADDER 1754+2=1756
P 1 7 5 6
MAR 1 7 5 6
17
50 51 52 53 54 55 56 57 58 59
HSM N 4 1 5 1 5 1 6 8 9
MR 1 6
B 1 6
BUS ADDER 1756+2=1758
P 1 7 5 8
MAR 1 7 5 8

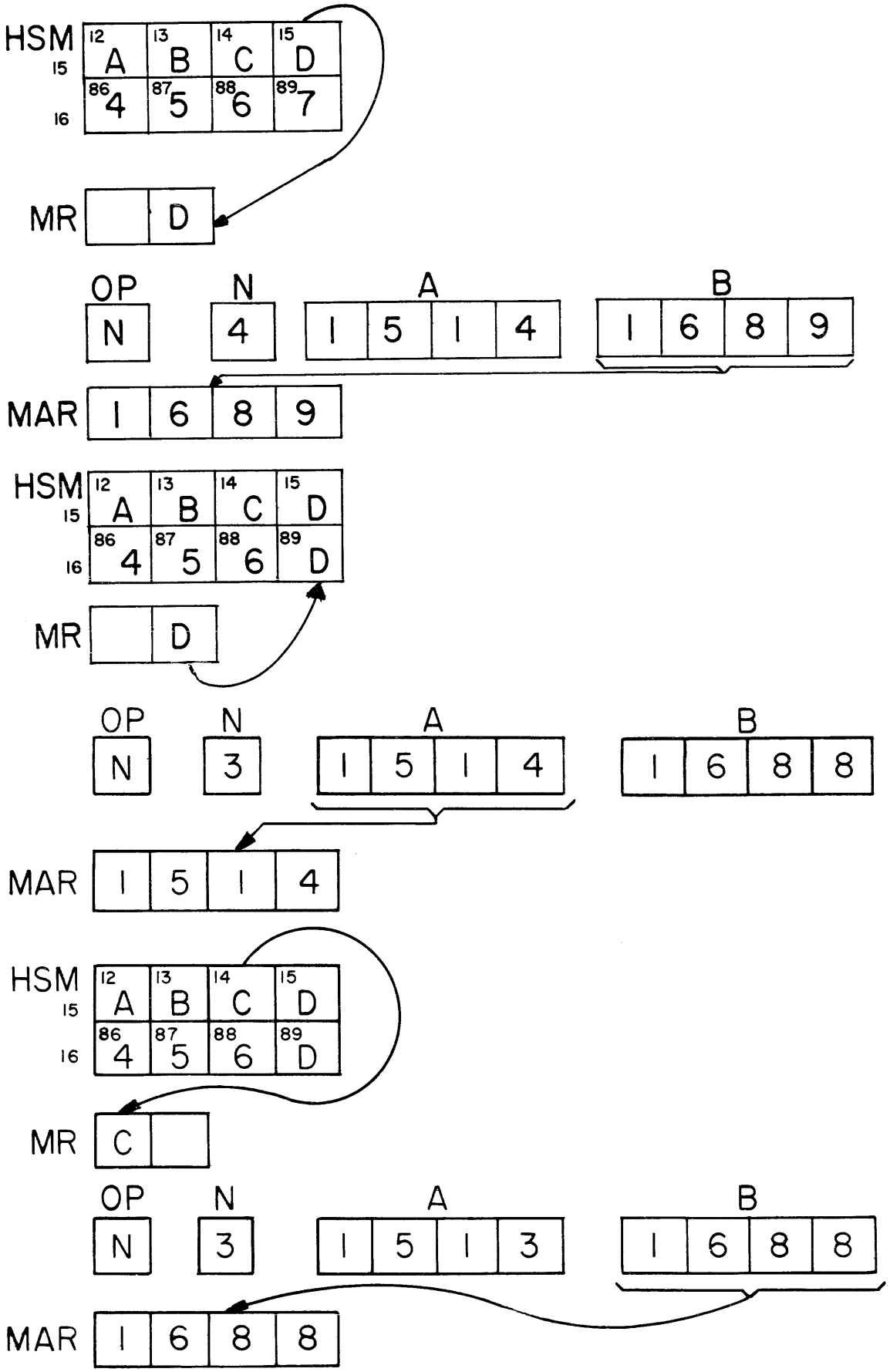


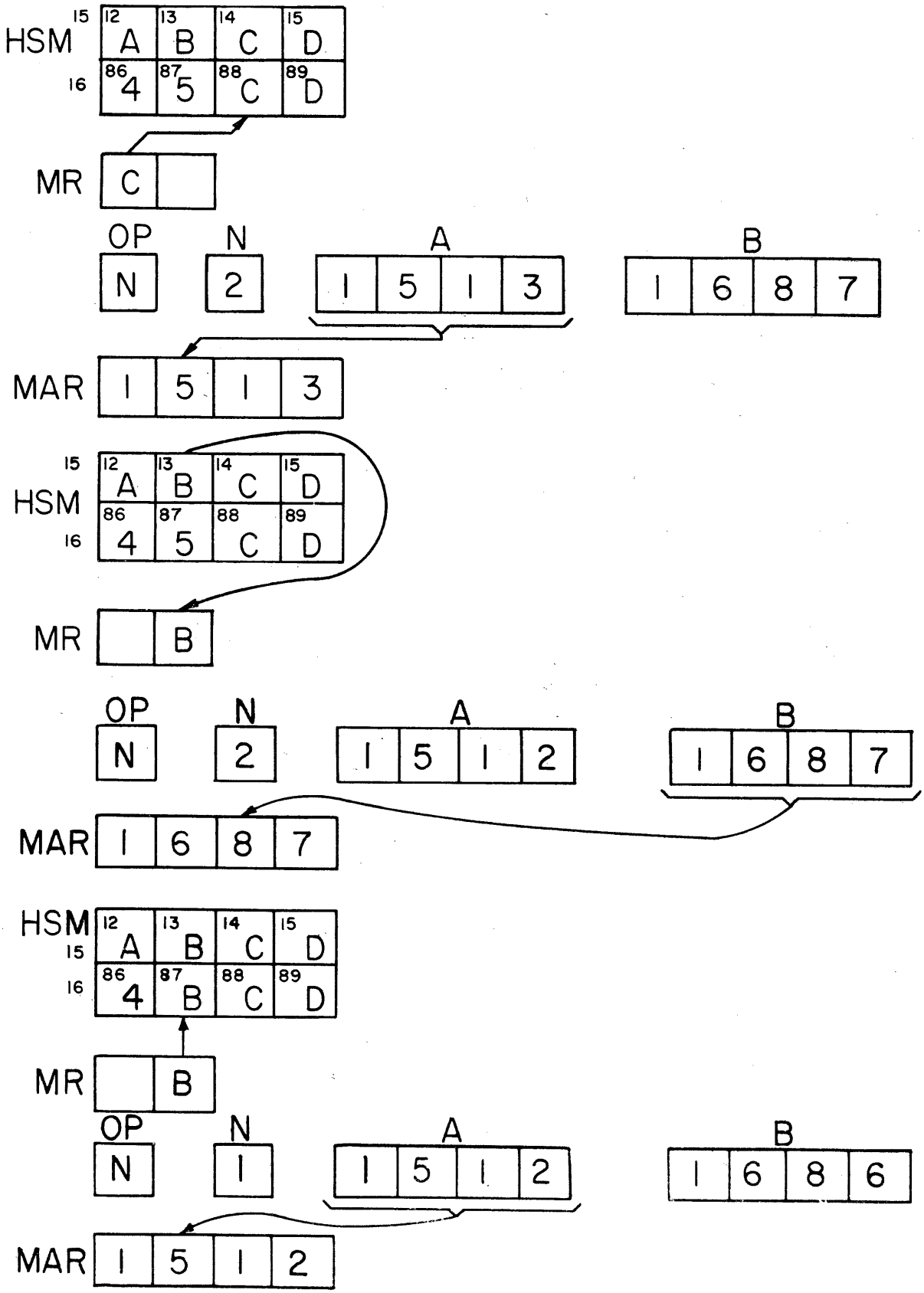
After staticizing, we find that the OP register holds our operation code (N), The "N" register holds a 4, the A register holds 1515, and the B register holds 1689. In addition, the P register holds the address of the next instruction to be executed (since this is a sequential machine), 1760. We are ready to execute this instruction.

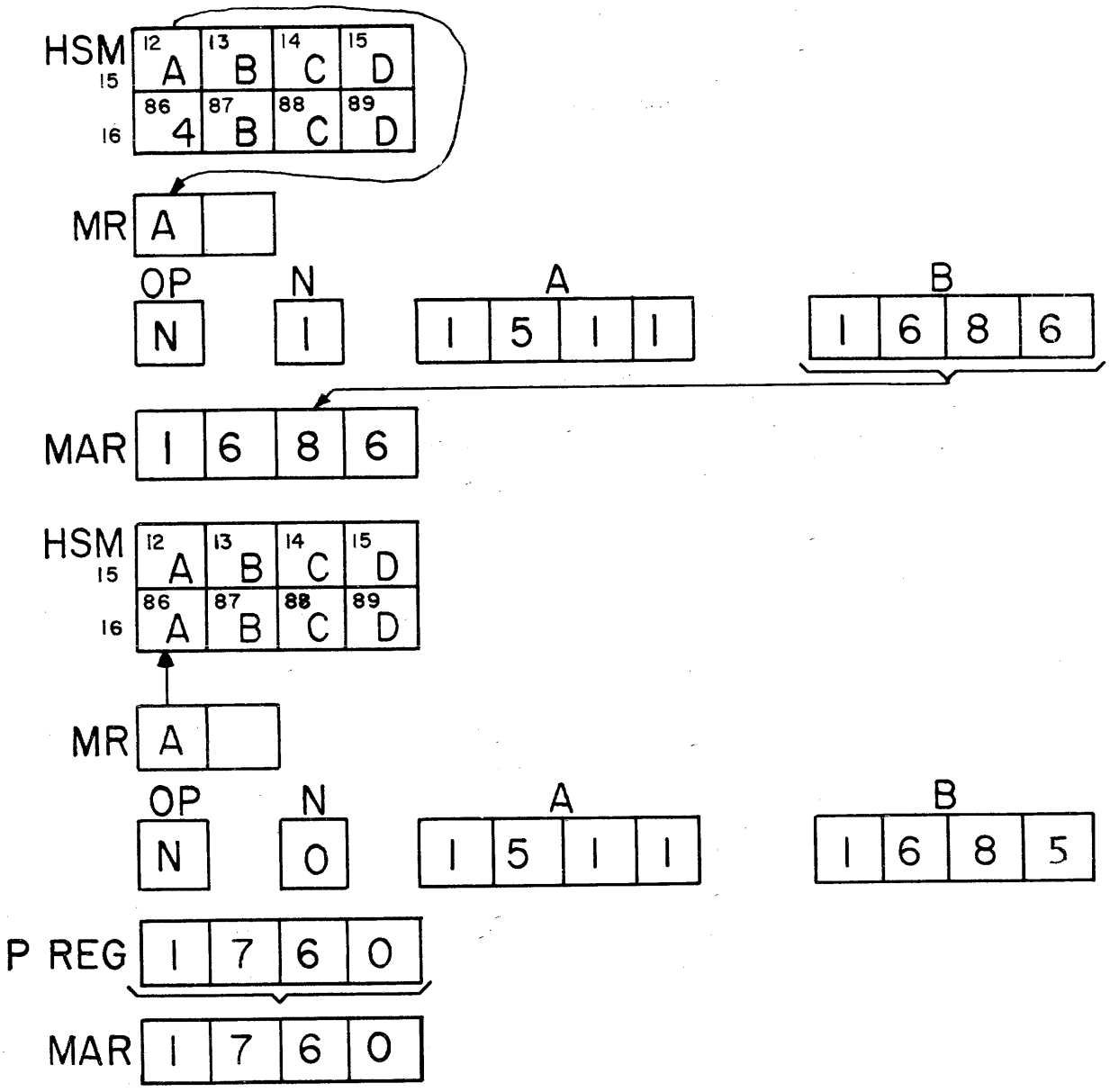
The N in the OP register tells the computer that we want to transfer data right to left. The content of the "N" register is not zero, so the instruction begins to execute. The A register contents are placed in the MAR and the character addressed is pulled out to the MR.

The Bus Adder decreases the contents of the A register by 1. The contents of the B register are placed in the MAR register and the character in the MR is placed in memory at this location. The contents of the B register are then decreased by 1, as is the content of the "N" register. Again, "N" does not equal 0, so the process is repeated. When N finally equals zero, the staticizing of the next instruction will begin.




HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 5
88 6
89 7
MR
D
OP
N
N
4
A
1 5 1 4
B
1 6 8 9
MAR
1 6 8 9
HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 5
88 6
89 D
MR
D
OP
N
N
3
A
1 5 1 4
B
1 6 8 8
MAR
1 5 1 4
HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 5
88 6
89 D
MR
C
OP
N
N
3
A
1 5 1 3
B
1 6 8 8
MAR
1 6 8 8


HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 5
88 C
89 D
MR
C
OP
N
N
2
A
1 5 1 3
B
1 6 8 7
MAR
1 5 1 3
HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 5
88 C
89 D
MR
B
OP
N
N
2
A
1 5 1 2
B
1 6 8 7
MAR
1 6 8 7
HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 B
88 C
89 D
MR
B
OP
N
N
1
A
1 5 1 2
B
1 6 8 6
MAR
1 5 1 2


HSM
15
16
12 A
13 B
14 C
15 D
86 4
87 B
88 C
89 D
MR
A
OP
N
N
1
A
1 5 1 1
B
1 6 8 6
MAR
1 6 8 6
HSM
15
16
12 A
13 B
14 C
15 D
86 A
87 B
88 C
89 D
MR
A
OP
N
N
0
A
1 5 1 1
B
1 6 8 5
P REG
1 7 6 0
MAR
1 7 6 0

X – AN INTRODUCTION TO FLOW CHARTING

A flow chart is a pictorial representation of the logical steps required to perform a task. Once completed, the problem need only be coded following the specifications of the flow chart, in order that the machine may understand its work.

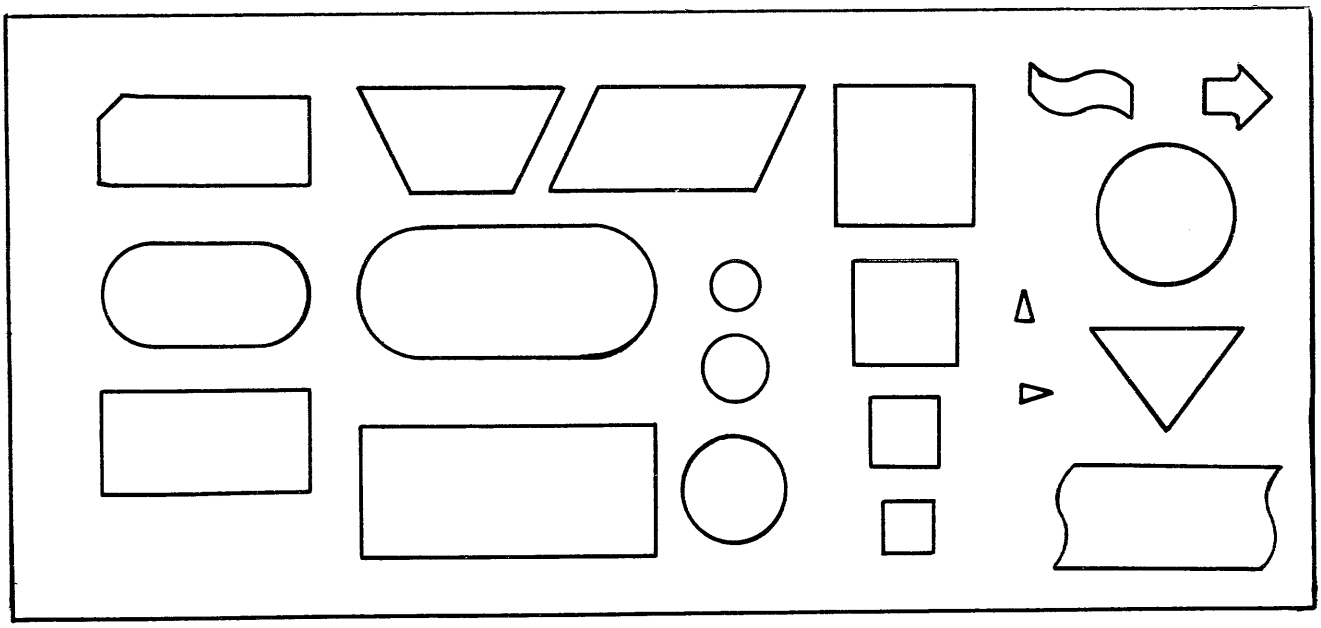
There are three types of flow charts.

The first is called a "systems flow chart". It is a diagrammatical representation of the sequential steps which must be performed in order to accomplish the complete data processing application. The analyst prepares this in order to determine what the input will be, what output is expected, and what types of data processing are necessary to derive the required output from the input.

The second type is a "functional flow chart". This is prepared by the programmer to show the general data processing steps which must be accomplished. The word "general" in this context means just the actual work to be accomplished with no reference to particular equipment. For example, a functional chart might contain a block called UPDATE BALANCE; in the corresponding detailed chart, this might require five or more computer steps to accomplish.

The remaining type is a "detailed process chart" often abbreviated to "detailed chart". The detail flow charting of a problem could actually be called the programming, since this is where the logical thinking is required. It is this chart that spells out the computer operations one by one. By utilizing this type of presentation of the problem, the programmer has the ability to think through the logical flow of the steps before having to concern himself with addresses, constants, names, and the like.

The following is a picture of the RCA template. The symbols on this template are the ones that are standard for all RCA charts.

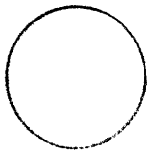


In order to be more familiar with the symbols, a discussion of each of the chart types of symbols follows:

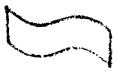
SYSTEM FLOW CHARTING SYMBOLS:

Since the major function of a systems flow chart is to show the input and output media, it is necessary to have symbols to show each type.

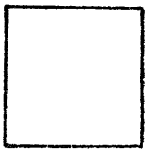
STORAGE MEDIA



Magnetic Tape

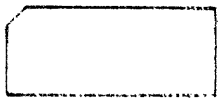


Paper Tape



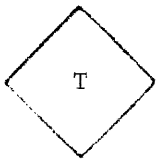
Random Storage

Documents

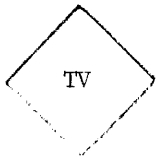


EAM Card

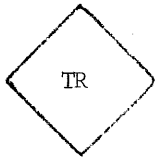
DATA PREPARATION



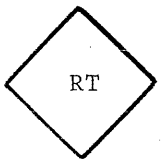
Tapewriter



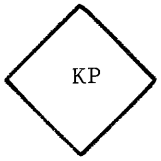
Tapewriter Verifier



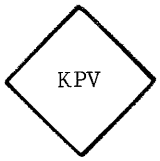
Tapewriter Reader



Reader Typer



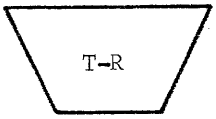
Key Punch



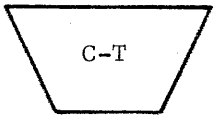
Key Punch Verifier

OFF LINE PERIPHERAL EQUIPMENT

Hi-Speed Transmission and Reception Unit

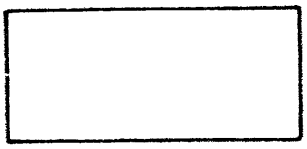


Transmission and Reception Unit



Code Translator

PROCESSING OPERATIONS



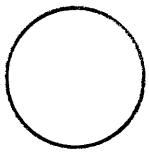
Computer Operation
(Sort, Merge, and Extract operations would be indicated within box.)

GENERAL RULE

Arrows should be used to clarify direction.

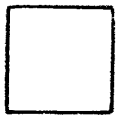
FUNCTIONAL AND DETAIL CHARTING SYMBOLS:

SYMBOLS AND SPECIFIC RULES



Tape Read or Tape Write (Magnetic and Paper Tape)
(Type of operation should be indicated within symbol.)

Card Read or Card Punch



Random Storage Read or Write

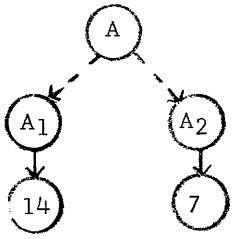
Document Read or Print

Internal Computer Operation
(Internal computer operation boxes may be drawn without separate vertical connecting lines provided there is only one point of entry.)

Decision

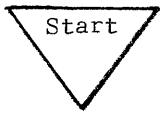
To and From Connector (Numerically designated)

1. The No. within "To" symbol is the destination box No.
2. The No. within "From" symbol is the box No. transferred from.
3. More than one "From" symbol can be used if one cannot indicate all "From" box numbers.
4. Connectors that refer to a different page should indicate the "referred to" page No.
5. "To" and "From" connectors are NOT numbered externally.

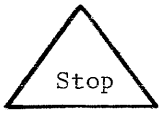


Variable Connectors (Alphabetically Designated)

1. Variable paths are designated with connector letter with subscripted number.
2. "To" path of variable paths are designated in identical manner as "To" connectors.
3. Variable connectors are externally numbered.



Start



Normal or Error Stop

1. PES # - Programmed ERROR STOP.
2. HALT # - Intermediate HALT.
3. EOR # - END OF RUN.

Closed Subroutine
(Designates entrance into, execution of, and exit from designated subroutine.)

Assertion Box

1. Defines purpose of any operation or decision box.
2. Indicates the box numbers that set variable connectors.
3. Defines purpose of variable connector settings.

GENERAL RULES

1. Lines

   A. Points of entry should be to the vertical line preceeding the box to be entered.

   B. Lines should be drawn in place of connectors whenever possible.

C. Lines must never cross-over.

D. Arrows are optional and used for clarification.

2. Other

A. External numbering of boxes should follow the actual coding sheets whenever possible.

B. Box sizes should be uniform but may be enlarged within reason.

C. Boxes encompassing a subroutine should be titled above the first box. The entire routine should be enclosed by a dotted line whenever possible.

D. Charting for subroutines not unique to the program need not be included.

E. Charts should be drawn to flow down and to the right whenever possible.

In addition to the template symbols, several written symbols are common. A few of these are:

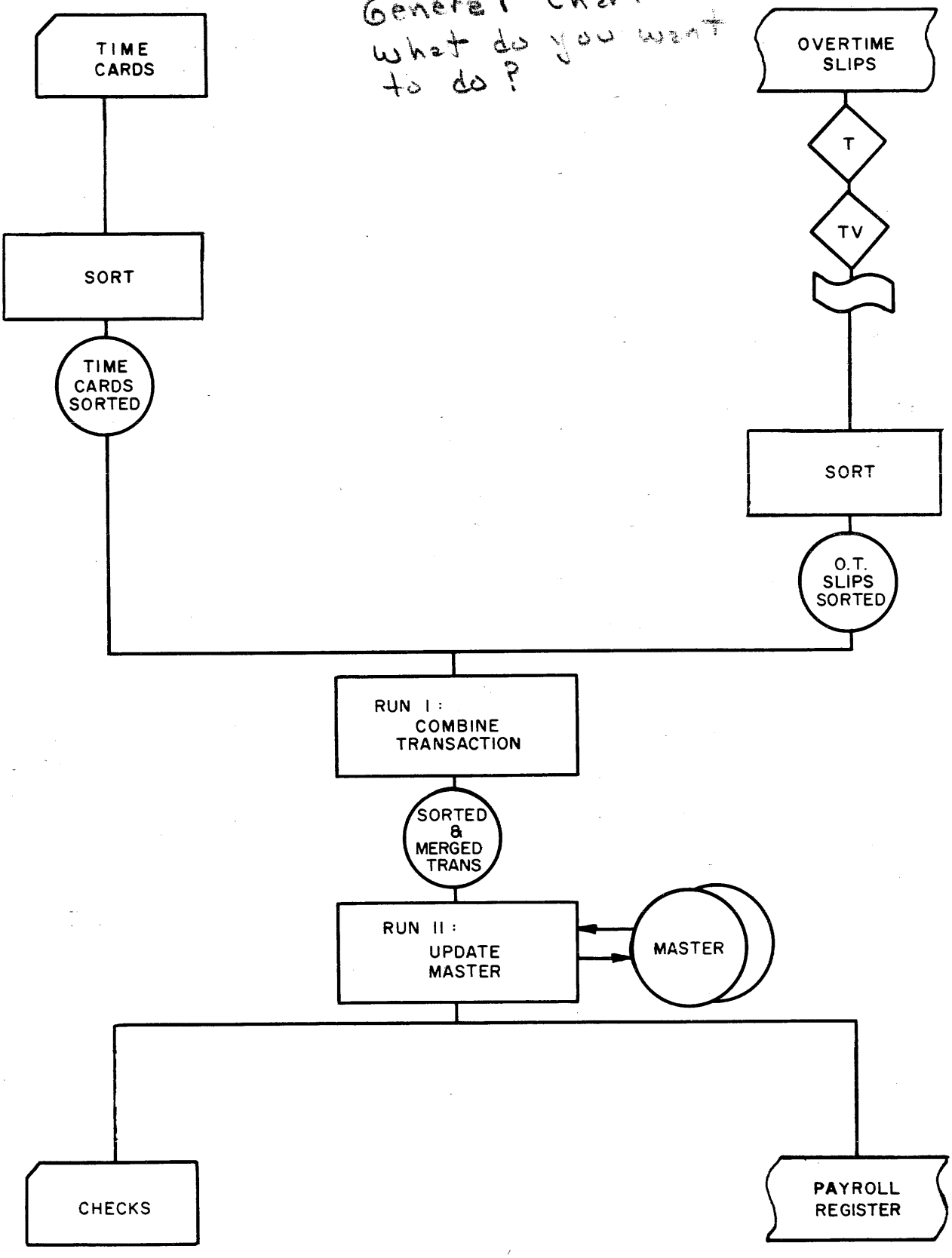
>	is greater than
<	is less than
=	equal
≠	not equal
:	is compared with
⟶	replaces
Σ	sum of
Ø	letter O
0	digit zero
Z	letter Z
.	period
	blank or space
⊖	minus

The following problem statement is used to exemplify what has been stated in this chapter. It shows the development from one chart to another, utilizing the symbols discussed.

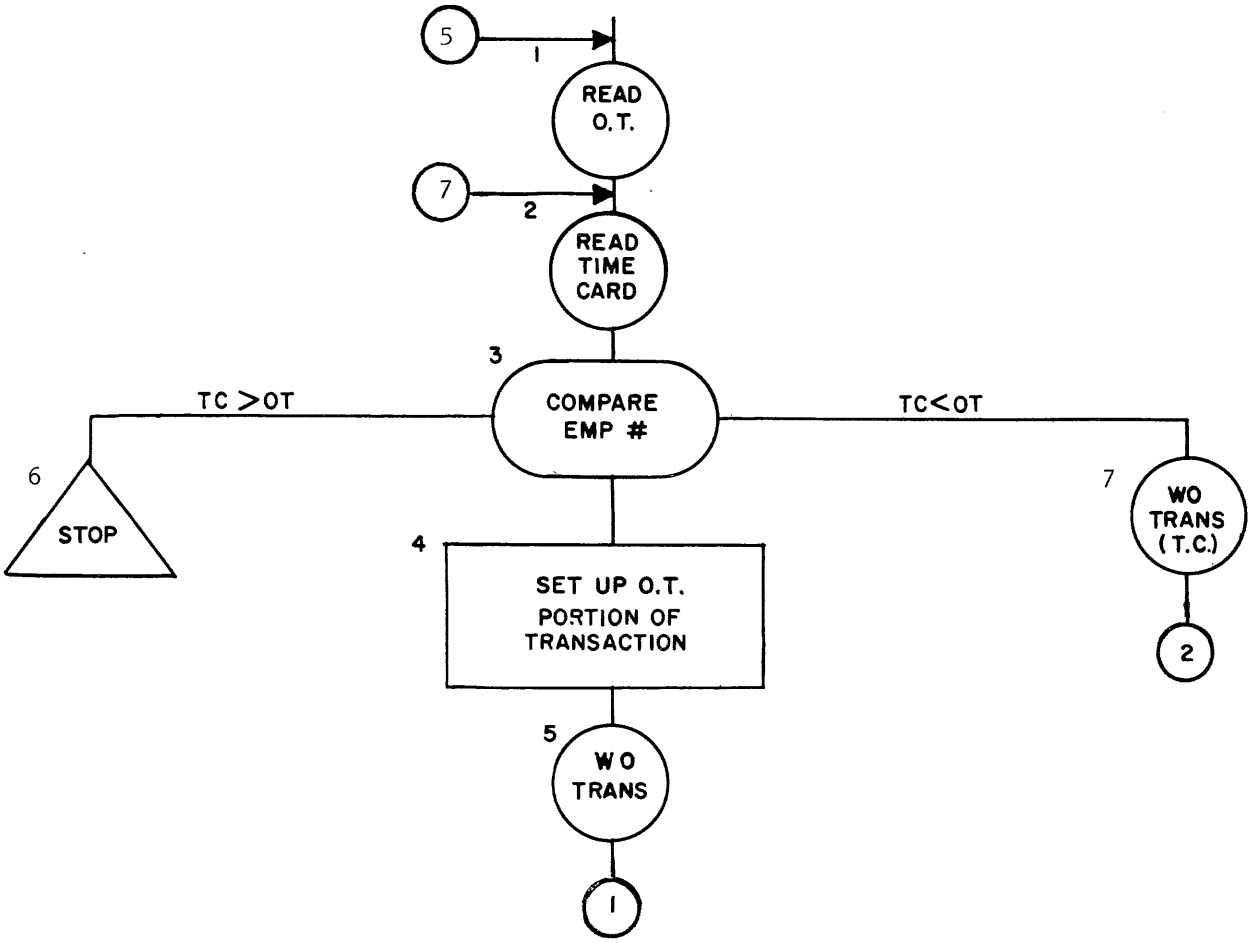
The problem is to prepare the payroll for 3000 employees paid on a weekly basis. Each employee has an identifying employee number and the master information for each employee is maintained on magnetic tape (1 employee per block) in ascending order by employee number. For any one week, approximately 2500 time cards must be processed to show the regular hours worked. In addition, there will be about 1000 overtime documents which must be processed. There will never be more than one card and/or 1 overtime slip per employee. (An employee cannot work overtime, without working regular hours.) The transactions are to be sorted and then a computer run must combine the two types of transactions, preparing the information which is to be posted against the master. This last updating is the second run. The output from this will be an updated master and a printed payroll register, and the checks prepared on card punch.

SYSTEMS FLOW CHART

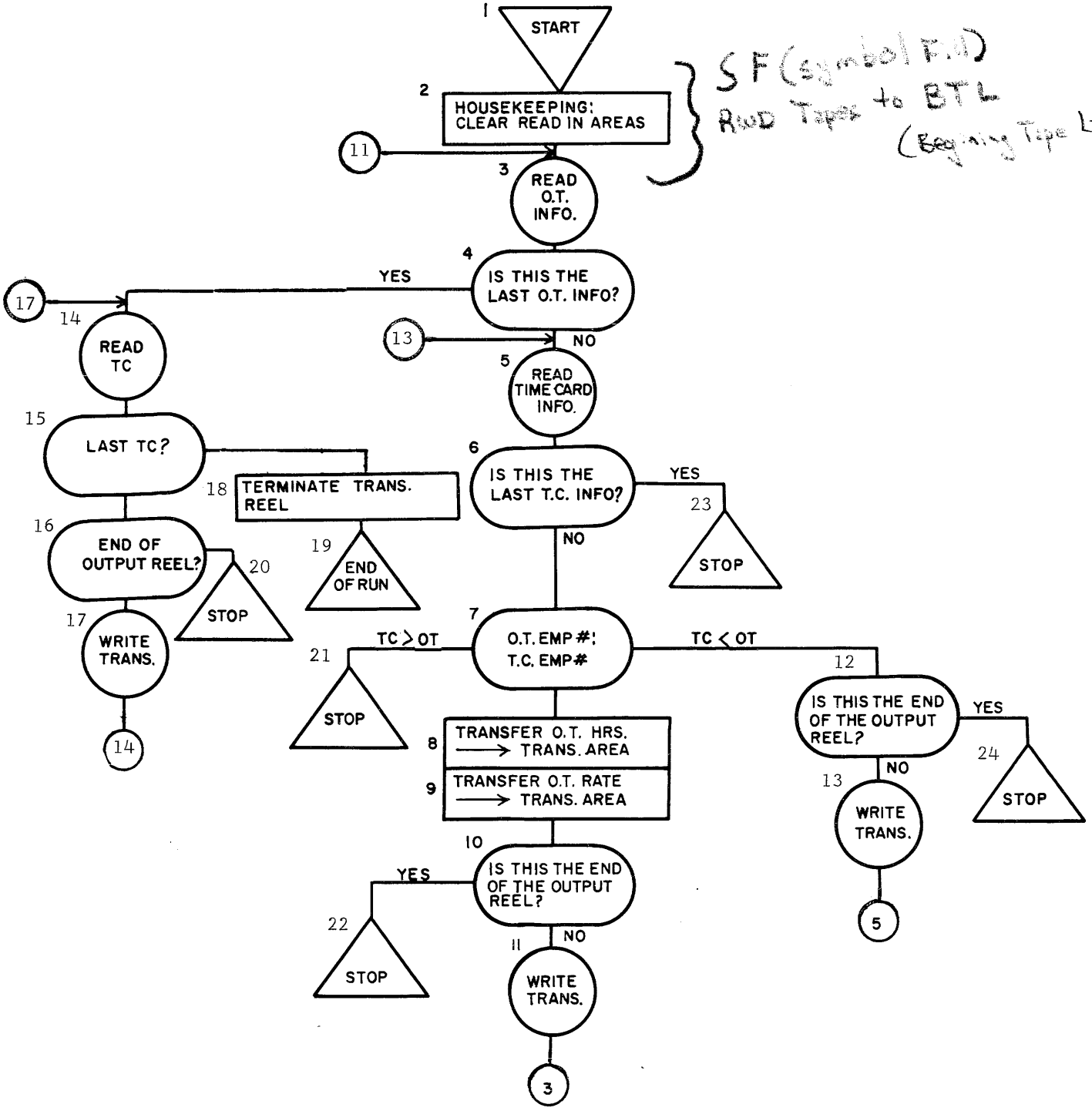
General Chart
what do you want to do?



RUN I FUNCTIONAL CHART

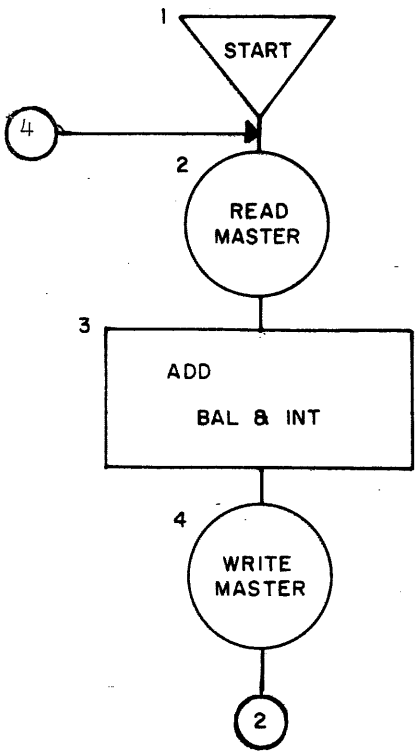


RUN I DETAILED CHART



SF (Symbol File) Read Tapes to BTL (Beginning Tape Label)

The detailed chart for our interest problem would appear as follows:



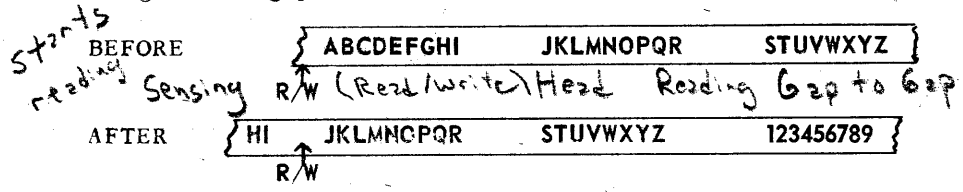
XI — INTRODUCTORY INSTRUCTIONS

TAPE READ FORWARD NORMAL, ADD, TAPE WRITE NORMAL, STORE REGISTER

Having discussed most of the preliminaries, it is now possible to investigate the specific format and operation of some of the instructions.

TAPE READ FORWARD NORMAL:

In order to be able to process any data, it is necessary first to bring the data into memory. One instruction that accomplishes this is the TAPE READ FORWARD NORMAL. The name of this instruction implies that it will read from tape moving in a forward direction and the instruction will be executed in the normal operating registers (OP, N, A, and B). As we have already discussed, when a read instruction is initiated, the tape moves, so that at the end of the read, the read-write head is sitting in the gap before the next block:



The operation code of the RFN is 4.

The "N" character gives the unit addressed.

Device	First Unit	Second Unit
Paper Tape Reader	8	
Hi-Data Tape Group	123456	ABCDEF
Tape Adapter:		
33KC	J	N
66KC	L	P
Dual Tape Channel (2x6)	123456	
Dual Tape Channel (2x12)	123456	ABCDEF
Interrogating Typewriter	U	

The A address gives the HSM location to receive the first character.

The B address gives the HSM location of the righthand end of the maximum read in area.

Example:

We have a file of records containing 10 characters each, which appear on tape as follows:

JOE_DOE_21	JIM_RAU_30	DON_RAY_07

The RFN instruction is:

4 3 1500 1509

The instruction is staticized and tape station 3 is activated. The tape begins to move and obtains the proper speed (30" per second). The first character is placed in the tape buffer and then transferred to the MR and then into memory at the address designated by the A register. The contents of the A and the B registers are compared. In our case, they do not match so the contents of the A register are increased by 1. The next character is read in and placed, the registers are compared. Again they do not agree, so the A register is increased by one and the process continues. After the tenth character has been placed in memory at location 1509, an A-B equality is found. [This causes an indicator to be set in order that the computer will be able to remember this condition.] The A register again is increased, but when the tape station attempts to bring in the next character from tape, it senses the inter-record gap. The sensing of this gap causes the instruction to terminate.

ADD:

Since addition is essential to any computation, a very important instruction is the ADD.

The OP code is a +.

"N" gives the number of characters in each operand, which implies that the operands must be of equal length. Obviously, if we were limited to the decimal numbers, the maximum length would be 9 characters. Since this would be too limited for our needs, there had to be developed a way to circumvent this problem. If we examine the construction of a character, we can see that we have ability to express all of the decimal digits from 0-9 with the right four bits:

2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

If we then take the remaining two bits, we can construct any number from 0-3:

2^5	2^4	
0	0	0
0	1	1
1	0	2
1	1	3

Combining the two, we can construct any number from 0-39. This plus 5 special symbols give us a field limit of 44. To simplify the process, a table has been developed, called the "N" table:

N Count	Symbol	N Count	Symbol	N Count	Symbol	N Count	Symbol
0	0	11	A	22	K	33	T
1	1	12	B	23	L	34	U
2	2	13	C	24	M	35	V
3	3	14	D	25	N	36	W
4	4	15	E	26	O	37	X
5	5	16	F	27	P	38	Y
6	6	17	G	28	Q	39	Z
7	7	18	H	29	R	40	EB
8	8	19	I	30	"	41	, (comma)
9	9	20	- (minus)	31	/	42	%
10	&	21	J	32	S	43	• (ISS)
						44	=

Right Most Position LSD of the Field you want to add to

The A address gives the HSM location of the least significant digit of the augend (augend plus addend equals sum). This is also the address of the least significant digit of the sum (the sum overlays the augend). Right Most Position LSD

The B address gives the HSM location of the least significant digit of the addend. SENDing Field

If an amount is negative, it will be indicated by placing a zone bit in the 2^5 position of the least significant digit. For example, a positive 5 would have a bit configuration of 000101, a negative 5 would have a configuration of 100101.

Addition is done in the RCA 301 by "table look-up". This simply means that there are two tables stored in memory, a sum table between 0000-0099 and a difference table between 0100-0199. The two digits that must be added form the right hand two digits of the address. If the signs of the two operands are alike, the lefthand two digits are 00, if the signs are unlike, the digits are 01.

If a carry occurs, one of the next two digits is increased by one.

If the signs are unlike, and the addend is larger than the augend, the entire sum must be complemented before the termination of the operation, in order to yield the proper result. For example, if we had the following problem the computer would handle it by the following steps:

1) ADD 23
   - 42

2) Using the difference table: * 81

3) *end around condition which requires complementation of 81, which yields 19

4) the answer is -19

Example:

Executing the following instruction, we will be able to follow through the steps of the computer.

+3 1999 1989

```
   86 87 88 89 90 91 92 93 94 95 96 97 98 99
19  A  1  8  3  B  C  D  E  F  G  H  4  6  4
```

Once the instruction is staticized, the least significant digits are examined. Since the signs are alike, the sum table will be utilized. In order to compose the address, another register must be introduced at this point. This is the D register (Data Register) which is composed of two one character registers, labeled D_2 and D_3 (due to the fact that they hold the right hand two characters of the address of the sum). The A address falls into the MAR and the character addressed is pulled from memory, placed first in the MR and then in D_2. The process is repeated for the B address. D_2 and D_3 would then contain, for our example, 43. Since we are using the Sum table, 00 will be added to this giving us an address of 0043. At that location in the table, there is a 7, which is the sum of 3 plus 4. This character will be placed into memory at the location given by the A register, namely 1999. The contents of the A, B, and N registers are decreased by one and N is again sensed for zero. Since it is not zero yet, the process is repeated. The address this time will be 0068, and at that location will be a 4. In addition, an indicator will be set which will cause a carry to be added to the next digits. This 4 is placed at 1998. The registers are again decreased, and N is sensed. Since it is not zero, the next address that will be constructed will be 0041. The carry must be added in, so the address will become 0042. At that address, we find a 6 which will be placed in memory at location 1987. Again the registers are decreased, but this time the "N" register indicates zero, which terminates the operation.

In addition, one more step is required before the operation is completed. This is the setting of one of three indicators known collectively as the PRI's (Previous Result Indicators). Individually, these are the PRP (previous result positive), PRN (previous result negative), and PRZ (previous result zero). This setting will indicate the result of the operation. In our case the result was positive (the sum was plus 647) so that the PRP would be set.

When there is a carry beyond the MSD of the sum, and the first Overflow Indicator has already been set in a Computer containing only one Overflow Indicator, the Computer stops on an alarm. When there is a carry beyond the MSD of the sum in a Computer with a second Overflow Indicator and this indicator has already been set, the Computer stops on an alarm.

The only allowable one zone bits in the operands of an addition for a processor with up to 20,000 characters are in the 2^4 bit position of the MSD and the 2^4 and 2^5 positions in the LSD. The only allowable one zone bits in the operands of an addition for a processor with 40,000 characters are the 2^4 and 2^5 bit positions in the MSD and the LSD.

In single character operands, however, a one bit in the 2^4 position of either operand causes an alarm stop.

TAPE WRITE NORMAL:

Once the data is updated, it will be necessary to prepare it as output. The TAPE WRITE NORMAL instruction allows us to write to magnetic tape utilizing the normal operating registers.

The OP code is an 8.

1-6 stations (Magnetic Tapes)

"N" again defines the tape unit (1-6, A-F, J or N, L or P, 9 for the Paper Tape Punch, 7 for the Monitor Printer, or U for the Interrogating Typewriter.

Tells Operator when jobs is finished

The A address gives the HSM location of the first character to be written, punched, or typed.

Writes one Character at time from Main Memory from Left to Right until A=B Memory

The B address gives the HSM location of the last character to be written, punched, or typed.

In Output E/ED is not sensed. Auto

Example:

We wish to write the following to magnetic tape: you must Write EF in your program to terminate end of write

	65	66	67	68	69	70	71	72	73	74	75
17	R	C	A	-	3	0	1	_	E	D	P

The instruction is

8 5 1765 1775

which says to write to tape on station 5 the information that falls in memory between 1765 and 1775.

The instruction is staticized and tape station 5 is activated. The contents of the A register are placed in the MAR and the character addressed is pulled out of memory and placed in the MR. From there it goes to the tape buffer and then to tape. The contents of the A and B registers are then compared. Since they will not be equal, the A register will be increased by one and the process will continue until A equals B. The A register will increase once more, so that it will end up with an address one to the right of the last character written, in our case 1776.

STORE REGISTER:

As we've already discovered, the P register keeps track of the program by holding the address of the next instruction to be executed. It follows, therefore, that if we could modify the P register, the instructions wouldn't necessarily have to be processed in sequential order. In order to do this, it must be possible to address the P register, but since registers are not part of HSM, we must have an instruction that will affect the registers, in this case the P register. This instruction is called STORE REGISTER. When we address the P register with a REG instruction, we not only have the ability to store the present contents of the P register, but also to place something new in that register.

In HSM there are some standard memory locations that are always used for the same thing. One of these is called STP (store P). It is used to store the contents of the P register before it is modified so that a temporary record can be kept of the location at which the change was made (STP will contain an address 10 greater than the instruction that made the change). The addresses of this location are 0216-0219. At this point, we will make a temporary rule that states that the contents of the P register must always be placed in STP before the P register is affected.

The OP code is a V.

N gives the register to be affected. At this point we are interested in the P register, so "N" must be a 1.

The A address gives the right hand end of the area to receive the contents of the P register, which we have stated to be 0219.

The B address is the address of the next instruction to be executed.

STA / These Areas reserved in Main Memory
STP / These Areas may be used

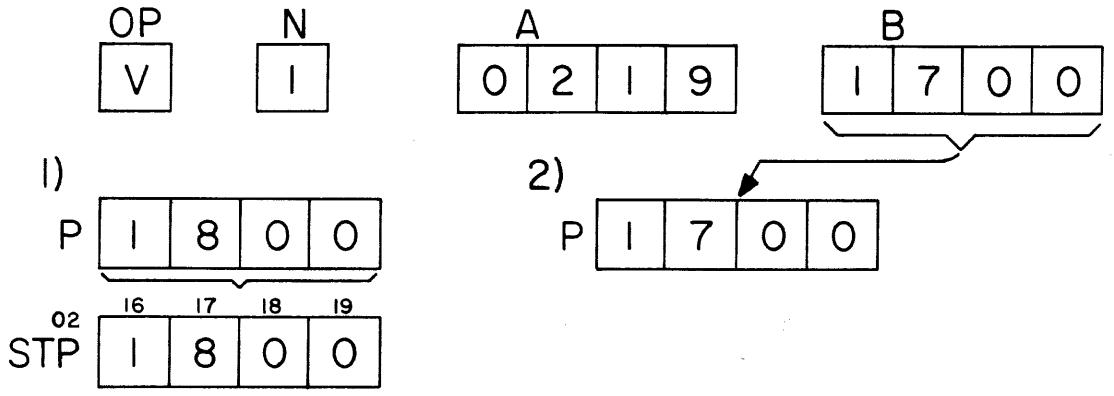
VO - may be used as XI-5 switch - Transfer of control

Example:

We have just written a program that consisted of 10 steps. We decided to place this program in memory starting at location 1700.

	OP	N	A	B
1700	X	X	XXXX	XXXX
1710	X	X	XXXX	XXXX
1720	X	X	XXXX	XXXX
1730	X	X	XXXX	XXXX
1740	X	X	XXXX	XXXX
1750	X	X	XXXX	XXXX
1760	X	X	XXXX	XXXX
1770	X	X	XXXX	XXXX
1780	X	X	XXXX	XXXX
1790	V	1	0219	1700

The last instruction in our program is a STORE REGISTER instruction. It will enable us to start doing the steps over again at 1700. Once the instruction has been staticized, the computer will recognize that it must affect the P register. The present contents of the P register (1800) will be placed in memory at STP (0216-0219) and 1700 will be placed in the P register. The computer will then start to staticize the next instruction, which will be the one located at 1700.



Having discussed these four instructions, Tape Read Forward Normal, Add, Tape Write Normal, and Store Register, it will be possible for us to write a program which will solve our interest problem. As you remember, we were to add the interest to the balance for each of the records.

Our first step would be to read in a record from tape 1 and place it into memory from 3000-3016, as depicted on the HSM Record (page XI-9). This instruction is:

4 1 3000 3016

The next step would be to add the interest to the balance, so that the sum becomes the new balance:

\+ 6 3016 3010

The next step would be to write the updated record out to tape on station 2:

8 2 3000 3016

The last step would be to start over again with the read. We have stipulated that the program will start at 3100. The Read will be at 3100, the Add at 3110, the Write at 3120 and the Register at 3130. Since we want to transfer back to the Read, our instruction will be:

```
V   1   0219   3100
```

To facilitate our writing of programs, we will use a 301 Computer Program Record sheet (page XI-10).

Following through on our program we would have it prepared as input, and read into the computer memory starting at location 3100. We would then set the P register to 3100 (by use of the computer console) and hit the start button. The following are the first two records we must process:

```
13452000256034189        23546000150013550
```

The first record is read into memory starting at 3000:

```
    00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
30   1  3  4  5  2  0  0  0  2  5  6  0  3  4  1  8  9
```

The next step would add the interest to the balance leaving:

```
    00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
30   1  3  4  5  2  0  0  0  2  5  6  0  3  4  4  4  5
```

The next step would write this new record out to a new tape:

```
13452000256034445
```

The Register instruction would put 3100 in the P register and the Read instruction would be executed again:

```
    00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
30   2  3  5  4  6  0  0  0  1  5  0  0  1  3  5  5  0
```

The add instruction would result in:

```
    00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
30   2  3  5  4  6  0  0  0  1  5  0  0  1  3  7  0  0
```

Writing this record out to tape we have:

```
13452000256034445        23546000150013700
```

The next instruction would transfer the program back to the read, etc.

E.D.P. DATA SHEET

	INPUT		OUTPUT	IDENTIFICATION	DATE
	REFERENCE		INTERMEDIATE	INTEREST ACCOUNT MASTERS	

SOURCE	USE	SCHEDULE

SEQUENCE	DISPOSITION	NO. OF MESSAGES
		PEAK: / NORMAL:

DOCUMENT DESCRIPTION:

TYPE: / SIZE: WIDE X LONG / NO. OF COPIES

REMARKS

ITEM NO.	SUB. NO.	DESCRIPTION	NO. OF CHARS. MAX.	NO. OF CHARS. AVG.	% OCCURRANCE	WTD. AVG.
1		ACCOUNT NUMBER	5	5	100	5
2		INTEREST	6	6	100	6
3		BALANCE	6	6	100	6
		TOTAL CHARACTERS OF INFORMATION	17			17
		ADDED CONTROL CHARACTERS	-			-
		TOTAL RCA CHARACTERS	17			17

| | | |

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
3130	6	310 0	0	0	4	1	3	0	0	0	3	0	1	6	4	READ MASTER RECORD	1
		1 0	0	0	+	6	3	0	1	6	3	0	1	0		ADD BALANCE + INTEREST	2
		2 0	0	0	8	2	3	0	0	0	3	0	1	6		WRITE MASTER	3
		3 0	0	0	V	1	0	2	1	9	3	1	0	0		TRANSFER → READ (1)	4
		0	0	0													
		0	0	0													
	6	200 0	0	0	4	8	1	0	0	0	1	0	3	8		Read Input	
		1 0	0	0	W	8	3	0	0	0	2	0	2	0		Sense EF/ED	
		2 0	0	0	+	7	1	0	3	8	1	0	1	0		Add 1st Qrt	
		3 0	0	0	+	7	1	0	3	8	1	0	1	7		" " 2nd "	
		4 0	0	0	+	7	1	0	3	8	1	0	2	4		" 3rd "	
		5 0	0	0	+	7	1	0	3	8	1	0	3	1		" 4th "	
	6	2060	0	0	8	9	1	0	0	0	1	0	3	8		Write Output	
		7 0	0	0	V	1	0	2	1	9	2	0	0	0		Do Again	
		8 0	0	0	E	Ø	J	5	E								
		0	0	0													
	4	300 0	0	0	8	9	2	0	8	4	2	0	8	4		Write EF	
		0	0	0	8	9	2	0	8	3	2	0	8	3		Write E/D	
	6	0	0	0	8	7	2	0	8	0	2	0	8	3		Mon. for Message EOJ	
		0	0	0	•	9	9	9	9	9	9	9	9	9		Halt	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XI-10

IE 1224 REV. 8-60

FLOW CHARTING & CODING EXERCISE I

Flow chart and code the following routine: to develop savings totals for each of the message in the Savings File. The input data appears as follows:

acct no.	first dep	second dep	third dep	fourth dep	total
5	6	6	6	6	6

Assumptions:

1) The items are all fixed-variable.

2) Total is carried as 0's initially, all fields are loaded with insignificant zeros.

3) Accumulate total in total item.

4) Read in the information from tape deck 1.

5) Write out the entire record to tape deck 2.

FLOW CHARTING & CODING EXERCISE II

Flow chart and code the following routine to develop a total of the quarterly profits for each year of the company's existence. The information is to come in from paper tape in the format:

year	1st quarter	2nd quarter	3rd quarter	4th quarter	total
4	7	7	7	7	7

characters

Data Base

Punch the full record out to the paper tape.

Assumptions:

1) The items are all fixed-variable.

2) Total is carried as 0's initially, all items are loaded with insignificant zeros.

3) Accumulate total in total item.

4) Write the entire record.

We are reading a total of 39 characters into Memory

XII – CARD INSTRUCTIONS

Card Read Normal, Card Punch and Input-Output Control

So far, we have limited our input and output to magnetic tape, but it is quite feasible that it might be cards. For this reason, we must have the ability to read and punch cards.

Since there are two models of Card Reader and two models of Card Punch in the RCA 301 system, we shall discuss the card instructions for the Card Reader (600 CPM) and Card Punch (100 CPM) first and the Card Read Punch (800 CPM Read and 250 CPM Punch) second.

CARD READER UNIT

CARD READ NORMAL

This instruction reads information from punched cards in the Card Reader and places this data into HSM.

The OP code is 0 (zero).

N either determines the rate at which cards are to be read or is used in a card cycle ending routine.

N	RESULT
1	Reads single card (200 CPM) or ends continuous card reading
2	Used in continuous card ending routine
4	Used in continuous card reading cycle (600 CPM)
M	Used in alternate card reading cycle (300 CPM)
8	Ends alternate card reading cycle

The A address gives the HSM location to receive the first character read into memory.

The B address is zeros (0000).

Example:

```
0   1  2900  0000
```

Once this instruction is staticized, the Card Reader places a card in the read station. The first character is translated from card code to RCA 301 code and is transferred to the Card Reader buffer. From there it is placed in the MR and finally into memory at the location designated by the A register. The following 79 characters are placed in successive locations in HSM. The final character (column 80) is located at $(A)_i + 79$.

Automatic card translation does not take place if the BCT button on the Computer Console Panel is in the "on" position. Each card column is then split into two groups of six bits each, a parity bit is generated and added to each group, and the resultant two characters are transferred into HSM.

The character represented by rows 9 through 4 in column 1 (row 9 = 2^5, row 4 = 2^0) is read into HSM at the location specified by the A register. Successive characters representing rows 9 through 4 are also read into memory with column 80 being placed at $(A)_i + 79$. The character represented by column 1, rows 3 through Y (row 3 = 2^5, row Y = 2^0) is read into HSM at $(A)_i + 80$. Successive characters representing rows 3 through Y are likewise read into HSM with column 80 being placed at $(A)_i + 159$. The final A register setting is $(A)_i + 160$.

In order to maintain a 600 card per minute rate, a Card Read instruction in which the N character is 4 must be executed for each card read. The successive card read instructions must be staticized before 100 milliseconds have elapsed. Approximately 80 milliseconds are required to read one card leaving 20 milliseconds of compute time between cards.

Continuous card reading is initiated when N=4. Since only the first of the three cards in the card reader transport mechanism is read by this instruction, the continuous card reading cycle, must be terminated by two card read instructions in which the N characters are 2 and 1 respectively to avoid a "feed" error.

The card feed rate may be accomplished at 300 cards per minute by staticizing successive card read instructions every 200 milliseconds for each card read. The N character in the card read instructions must be M. Of this 200 millisecond card cycle, approximately 120 milliseconds are free for computing after the card has been read.

This Alternate Card Reading cycle must be terminated by a Card Read instruction in which the N=8.

When Card Read instructions are staticized in which the N character is 1, cards may be read at any rate up to 200 cards per minute. No terminating instruction is necessary since only one card has been fed into the Card Reader transport mechanism by this instruction.

CARD PUNCH NORMAL:

This instruction enables the Card Punch to punch 80-column cards from information contained in HSM.

The OP code is a 2.

"N" is a zero (0).

The A address gives the HSM address of the first character to be punched.

The B address gives the HSM address of the last character to be punched.

The information we wish to punch is located in memory from location 4500-4739. This indicates that we will be preparing three cards (80 x 3 = 240). The instruction is:

```
2   0  4500  4739
```

The instruction is staticized and the Card Punch is activated. A scanning process begins which enables the Card Punch to prepare one row of one card to be punched at a time. It continues punching a row at a time until the A and B registers are equal for the last row punched in a card. The punched cards are read to insure the accuracy of punching.

CARD READER/PUNCH UNIT

CARD READ NORMAL:

This instruction reads information from punched cards in the Card Reader/Punch and places this data into HSM

The OP code is 0 (zero).

N -

SYMBOL	RESULT
K	Binary read mode specified
1	Translate read mode specified

6 rows 160 columns

12 rows 80 columns

The A address gives the HSM location to receive the first character read from punched cards.

The B address is zeros (0000).

This instruction directs the Card Reader/Punch to place a card in the read station and operates on successive characters in the following cycle.

When the translate read mode has been specified (N=1) the character in column one of the punched card is automatically translated from RCA card code to RCA 301 code and is placed in HSM in the location specified by the A register. The following 79 characters are placed in successive locations in HSM. The final character (column 80) is located at $(A)_i + 79$.

When the binary read mode has been specified (N=K) card card column is split into two groups of six bits each, a parity bit is generated and added to each group and the resultant two characters are transferred into HSM.

The character represented by rows 9 through 4 in column 1 (row 9 = 2^5, row 4 = 2^0) is read into HSM at the location specified by the A register. Successive characters representing rows 9 through 4 are also read into memory with column 80 being placed at $(A)_i + 79$. The character represented by column 1, rows 3 through Y (row 3 = 2^5, row Y = 2^0) is read into HSM at $(A)_i + 80$. Successive characters representing rows 3 through Y are likewise read into HSM with column 80 being placed at $(A)_i + 159$. The final A register setting is $(A)_i + 160$.

A card read instruction must be staticized for each card read. In order to maintain an 800 card per minute rate, successive Card Read Instructions must be staticized before 75 milliseconds have elapsed. The 75 milliseconds card cycle is comprised of 21 milliseconds card feed access time, 44 milliseconds card read time, and 10 milliseconds time free for compute. When the read release option is exercised (via the Input Output Control Instruction), an additional 21 milliseconds of compute time is made available.

Trans reads 12

CARD PUNCH NORMAL:

This instruction enables the Card Reader/Punch to punch 80-column cards from information contained in HSM.

The OP code is 2

N -

N	CARD PUNCH MODE
&	Binary
0 (zero)	Translate

The A address gives the HSM address of the first character to be punched.

The B address gives the HSM address of the last character to be punched.

A start signal is sent to the card punch unit. The card punch punches the information from HSM between and including the locations addressed by the contents of the A and B registers, punching up to 80 columns to a card. When the A and B registers are equal, the last card is punched and the contents of the A register are incremented by one.

When the binary punch mode is specified (N = &), the A register denotes the memory location of the character to be punched in column 1, rows Y through 3. Each information bit results in a hole in the card (2^0 - row Y, 2^5 = row 3). Successive characters representing rows Y through 3 are also punched from HSM with column 80 being punched from $(A)_i + 79$. The character present at $(A)_i + 80$ is punched into column 1, rows 4 through 9 (2^0 = Row 4, 2^5 = Row 9). Successive characters representing rows 4 through 9 are also punched from HSM with column 80 being punched from $(A)_i + 159$.

The card punched is read concurrently with the punching of the succeeding card to insure the accuracy of punching.

INPUT-OUTPUT CONTROL:

This instruction performs the Read Release, Punch Release and Stacker Select functions for the Card Reader/Punch.

The OP code is ;.

N - (.

A Address - A_0, A_1, A_2 - 000 (zeros).
A_3 - designates function to be performed:

A_3	FUNCTION
	5 Pocket
1	Select Reader Stacker No. 1
2	Select Reader Stacker No. 2
3	Select Punch Stacker No. 4
4	Select Punch Stacker No. 8
5	Read Release
6	Punch Release
7	Read Release and Select Stacker No. 1
8	Read Release and Select Stacker No. 2
9	Punch Release and Select Stacker No. 4
sp.	Punch Release and Select Stacker No. 8

B Address - 0000 (zeros)

The A_3 character is sent to the control module. After the operation is initiated, the registers are available for use by the next instruction.

CARD EXERCISE I

B. F. Swift Company has a weekly run to determine the total gross pay. Transactions come in from cards in the following fixed format:

1-25	26-30	31-35	36-40
Name	Clock No.	Base Pay	Overtime
25	5	5	5

Compute total gross pay by adding overtime to base pay (columns 31-35 will be gross pay) and punch out name, clock no. and gross.

Flow chart and code.

XIII – TAPE AND FILE TERMINATING TEST INSTRUCTIONS

(REWIND, INPUT-OUTPUT SENSE, ED/EF SENSE, HALT)

The tape programs that we have written up to this time have been "loop" programs. That is to say, once they got started there would be no way to stop them until the tape physically ended. Obviously, this is not desirable, so we must have some way to be able to sense the end of tape, in order to be able to stop writing to it, and also be able to sense for the ED or EF on an input tape.

Each reel of tape will be supplied with three physical markers: The first one is called BTC (Beginning of Tape Contact). This is located near the physical beginning of the reel, and it is at this point that data should be started. For this reason, it is advisable to rewind each new tape placed on a tape station to BTC before beginning to record on it.

The second marker is near the end of the reel. It is called ETW (End of Tape Warning). This device will trigger an indicator which can be sensed by the programmer, thus allowing him to prevent writing up to the physical end of the tape (PET), which would cause the computer to stop in an error condition.

REWIND TO BTC:

This instruction causes a designated magnetic tape unit to rewind the tape reel mounted on it to BTC. Once it is initiated, the tape unit takes over and the computer is free to staticize the next instruction.

The OP code is ;.

"N" identifies the tape station (1-6, A-F, J or N, L or P).

The A and B addresses contain zeros (0000).

INPUT-OUTPUT SENSE:

With this instruction, the programmer is able to sense the ETW indicator to see if it has been set and then choose the proper one of two sequences of instructions.

The OP code is S.

N gives the device to be sensed (1-6, A-F, J or N, L or P).

The A address gives in A_0 a code that will ask the proper question. In this case, the question is "Has ETW been sensed?" and the code is a bit in the 2^2 position, which would give us a 4. $A_{1,2}$ and $_3$ are zeros.

The B address gives the address of the next instruction to be executed if the condition exists, in this case if ETW has been sensed.

Programming-wise, this implies that a test should be made for ETW before or after each write, so that when it is found, the transfer could be to a routine which would allow some steps to be taken to continue the writing on a new tape.

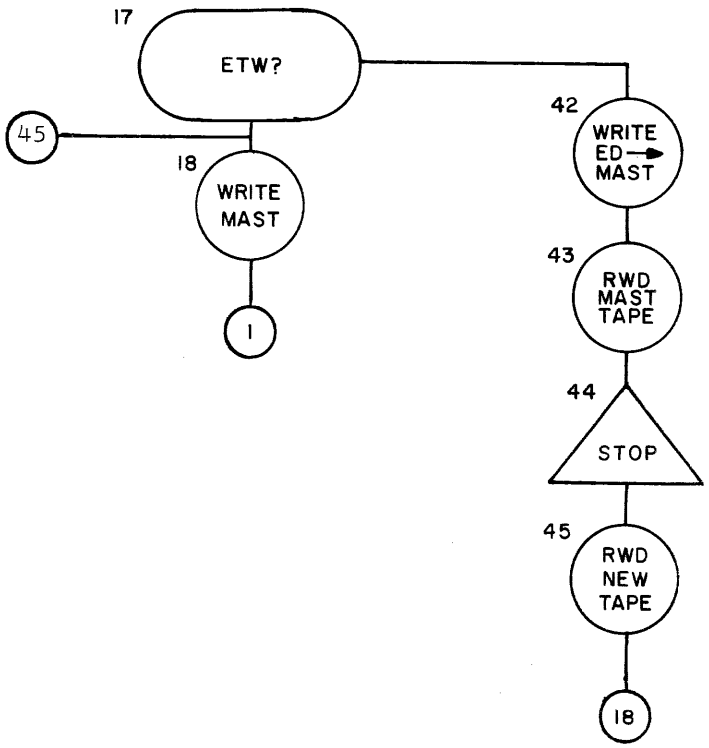
Example:

The following is only a section of a program:

```
1970   S  2   4000   2060      Sense Tape Station 2 for ETW
1980   8  2   1000   1099      Write out the record to Tape Station 2
1990   V  1   0219   1900      Transfer back to the first instruction

2060   8  2   2000   2000      Write an ED (an ED is found at 2000) to tape
2070   ;  2   0000   0000      Rewind tape 2
2080   ⊡  0   0000   0000      Halt (To mount new tape)
2090   ;  2   0000   0000      Rewind the new tape to BTC
2100   V  1   0219   1980      Transfer back to write of record
```

When the sense is staticized it will examine the ETW indicator. As long as this is not set (i.e., ETW has not been sensed by the tape unit), the P register will remain unaffected and the program will continue in sequential order, with the Write of the record occurring next. When ETW has been sensed, the indicator will be set and when the next Sense instruction is staticized, the contents of the P register will automatically go to STP (0216-0219) and then the contents of the B register will go to the P register. In this way the next instruction to be executed will be a write of an ED to the completed tape. As you remember, an ED symbol indicates the end of the good data on a reel of tape. The tape can then be rewound to BTC, so that the operator can remove it when the operation is completed.

The computer is instructed to Halt and wait for the operator to re-initiate the program. This allows him to remove the completed tape and mount a new tape. Then, by hitting the Start button, the next instruction will be to rewind this new tape to make sure that it is on BTC and then transfer back into the program to pick up where it had left off, with the write of the record. Flow charted, the program would look as follows:



HALT:

In our ETW routine, we mentioned one instruction we have not yet discussed, HALT. This stops the computer from staticizing the next instruction, causing the computer to cease operations. The OP code is ⊡ (period) written as ⊡ .

N, A, and B may contain numeric constants.

The P register remains unaffected.

CONDITIONAL TRANSFER OF CONTROL (ED/EF INDICATOR):

When we were writing to a tape we had to maintain a continual check for the end of the tape. When we sensed it, we placed an ED symbol that was to act as a sentinel for the end of data on the tape. It is only logical, therefore, that when we are reading data in from tape we should watch for this symbol, and for the EF symbol which would indicate that we had processed the last record in the file. In order to facilitate this, the tape stations are doing the actual sensing for these characters. When one or the other is found, an ED/EF indicator is set. It is the programmer's responsibility to continually sense this indicator. This is done by use of a CONDITIONAL TRANSFER OF CONTROL instruction.

The OP code is W.

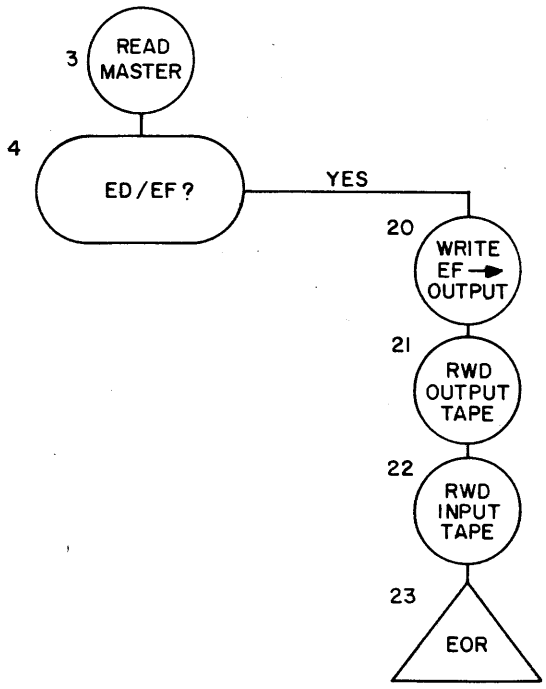
"N" indicates what is to be sensed. In this case we want to sense the EF/ED indicator, so the code would be an 8.

The A address gives the location of the next instruction to be executed if the EF/ED indicator is set.

The B address gives the location of the next instruction to be executed if the EF/ED indicator is not set.

Example:

The file we are dealing with is only one reel in length, so that we can assume that the control symbol we will find will be an EF. Once it is found we want to end the run, which means that we must put an EF on the output tapes, rewind all the tapes and stop. Charted, this particular portion of the program would look as follows:

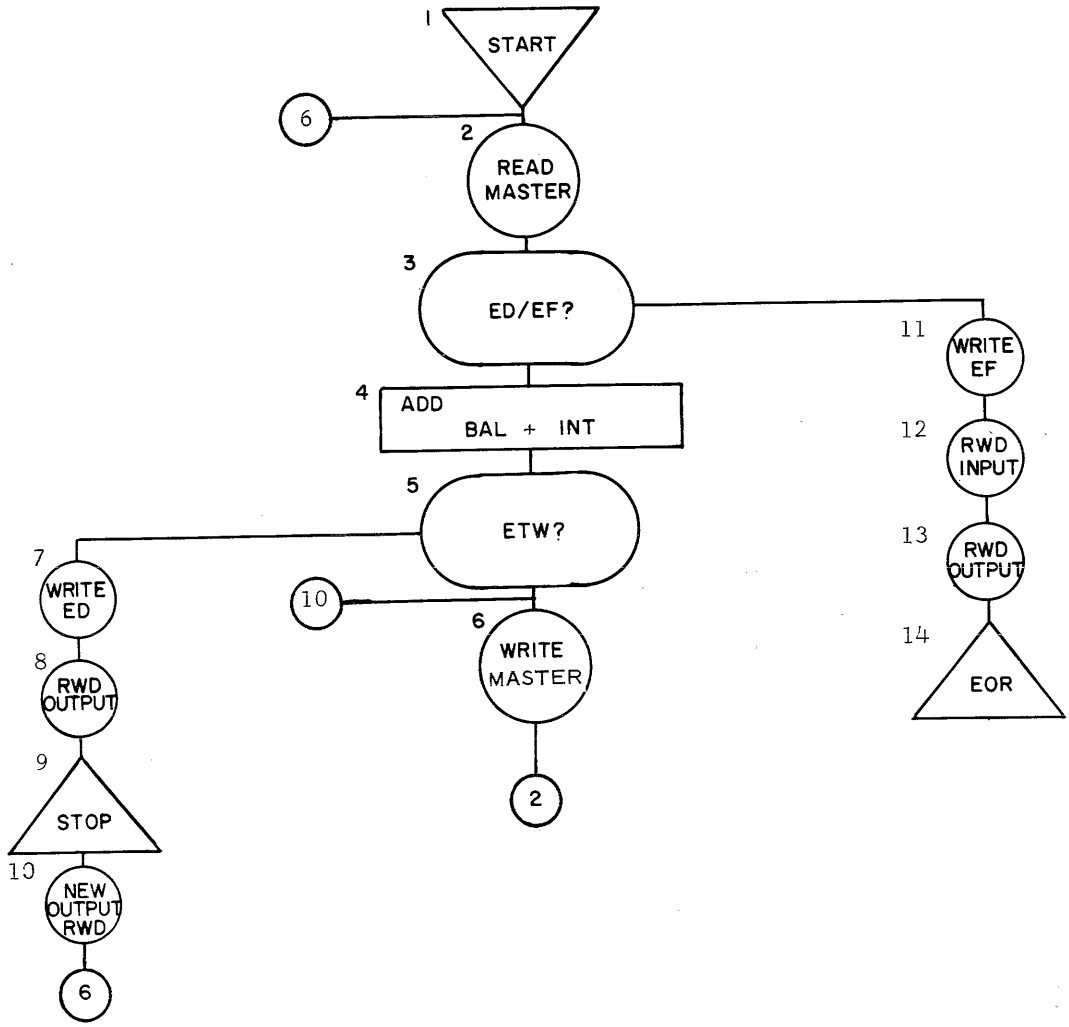


The coding would be:

```
3600  4  5  2500  2750    Read a record into memory from Tape 5
3610  W  8  3700  3620    CTC to sense EF/ED.

3700  8  3  3690  3690    Write an EF to the output tape (EF at 3690)
3710  ;  5  0000  0000    Rewind input tape
3720  ;  3  0000  0000    Rewind output tape
3730  .  0  0000  0000    Halt (End of Run)
```

After each read, the EF/ED indicator is sensed. As long as it is not set, the program will continue to the instruction addressed by the B register. This is accomplished because the current contents of the B register are placed in the P register. In our particular example, the P register will end up with the same address. Once the indicator is sensed, however, the program will transfer to 3700, which will cause the computer to place an EF on the end of the output reel in order that it may be used as input tomorrow, with a sentinel to indicate its end. The next two instructions cause the tapes being utilized by the program to start rewinding to BTC and then the computer stops at the end of the run.

In order to examine a complete program, let's take our interest problem and incorporate what we have just learned:



EXERCISE I - TAPE AND FILE TERMINATING TESTS

PROBLEM: Duplicate a file.

INPUT: One reel of tape containing records in the following format:

Branch	Acct No.	ID Code	Activity Date	Balance
3	8	1	6	10

PROCESSING:

1) Read the records into memory starting at 1000 from deck 1.

2) Check for ED/EF and ETW.

3) Write the message to an output tape on deck 2.

4) If EF is sensed, proceed to end of job routine.

5) End of procedure

   a) Write EF to output tape.
   b) Rewind both tapes.
   c) Stop

6) Start Program at 2000.

OUTPUT: A duplicate tape.

XIV – DECISION MAKING INSTRUCTIONS

(SECTOR COMPARE LEFT, CONDITIONAL TRANSFER OF CONTROL)

One of the most important functions of the computer is to make magnitude comparisons, i.e., is 3457 equal to, greater than, or less than 5674? In order to do this we utilize two instructions:

SECTOR COMPARE LEFT:

This instruction enables the program to compare two equal length sectors and indicate their magnitudinal relationships. The instruction compares the fields one character at a time working from left to right, so that the first time the characters are unequal the operation can terminate, having determined the relative magnitude.

The OP code is Y.

Maximum 44 works similar to Add

N is the number of characters in each operand (remember that the operands must be equal in length). Here again we make use of the "N" table shown on page XI-3.

The A address is the HSM location of the leftmost character of the first operand, which we may think of as the minuend (minuend minus subtrahend equals difference.)

The B address is the HSM location of the leftmost character of the second operand, which we may think of as the subtrahend.

Again, the Previous Result Indicators are utilized. As you remember, there are three of them, and only one can be set at a time. If the first operand is greater than the second operand, the PRP (Previous Result Positive) is set; if it is equal to the second operand, the PRZ (Previous Result Zero) is set; if it is less than the second operand, the PRN (Previous Result Negative) is set.

Example:

Compare two stock numbers shown below:

	45	46	47	48	49			34	35	36	37	38
34	3	3	5	6	3		23	3	3	6	3	2

The instruction is:

Y	5	3445	2334

The instruction is staticized and the contents of the HSM locations addressed by the A and B registers are transferred to the D register (Data Register). The A and B registers are increased by 1 and the 3 and the 3 are compared in the D registers. The N register is decreased by 1. They are equal so the next two characters are brought out and compared. Again they are equal and the comparison continues. When the 5 and the 6 are compared, they are found to be unequal. This will terminate the operation. The PRN is set (the second operand is greater than the first), and other than that nothing is affected. Both operands remain intact in the memory locations that they were in originally.

Having done the comparison, it is logical that we would wish to follow one of three paths, depending upon the result of the comparison. This is possible by utilizing the Conditional Transfer of Control which senses the PRI's.

CONDITIONAL TRANSFER OF CONTROL (PRI's)

Again this instruction enables us to branch our program depending upon a previous result, in this case the setting of the PRI's.

The OP code is W.

"N" this time will be a 1, indicating that the PRI's are to be examined.

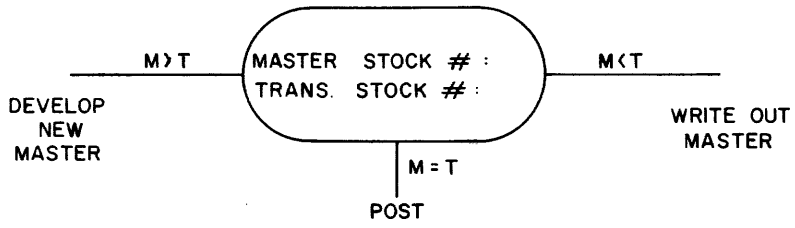
The A address gives the location of the next instruction to be executed if the PRP is set.

The B address gives the location of the next instruction if the PRN is set.

If the PRZ is set, the P register will not be affected, and therefore the next instruction in sequence will be executed.

Example:

We have just compared stock numbers. Let us assume that these were read in from two files, the first file containing master information to be updated by transaction information coming in on the second file. We are now comparing the Master stock number and the Transaction stock number (the master having been read into 3440 and the transaction into 2330 and assuming that the stock number is the first item in each record). If the master and the transaction have the same stock number, obviously we will want to post the information in the transaction against the master. If the master is greater than the transaction, the transaction refers to a non-existent stock master, which must then be developed. If the master is less than the transaction, we need a new master, so we must write out the present master and then bring in a new one. Briefly our chart looks like this:



Giving addresses at random to the two side paths, we'll say that the master preparation path starts at 6700 and that the write out of the old master path starts at 6850. Our series of instructions would then be:

```
6540   Y   5   3440   2330
6550   W   1   6700   6850

6560 begins the processing path
6700 begins the master preparation path
6850 begins the write out of the old master path
```

After the comparison instruction is done, the CTC is staticized, the computer then senses the PRI's and if the PRP is set, it places the current contents of the P register in STP and then places the contents of the A register into the P register. If the PRZ is set, the P register remains untouched and the program continues. If the PRN is set, as in our case, the contents of the P register are transferred to STP and then the contents of the B register are placed in the P register. Our particular example indicates that this master must be written out and a new master read in.

The following example will illustrate two comparisons, one for EF or ED (due to a multi-reel file) and the other a comparison of a constant (remains the same throughout the program) to the branch number.

DECISION MAKING EXAMPLE

PROBLEM: A bank keeps a master account file for all its branches on one tape file. As individual branches become large enough to handle their own updating, it is necessary to strip the master file of all the records pertaining to that particular branch. This is the purpose of your run.

INPUT: Bank Master Account File on magnetic tape deck 6.

Branch	Acct #	ID Code	Activity Date	Balance on Hand
3	8	1	6	7

PROCESSING:

1) Read Master File Record

2) Select all messages pertaining to a particular Branch indicated by card input and write these to a separate tape on deck 2.

3) Write all other messages of the file on deck 4.

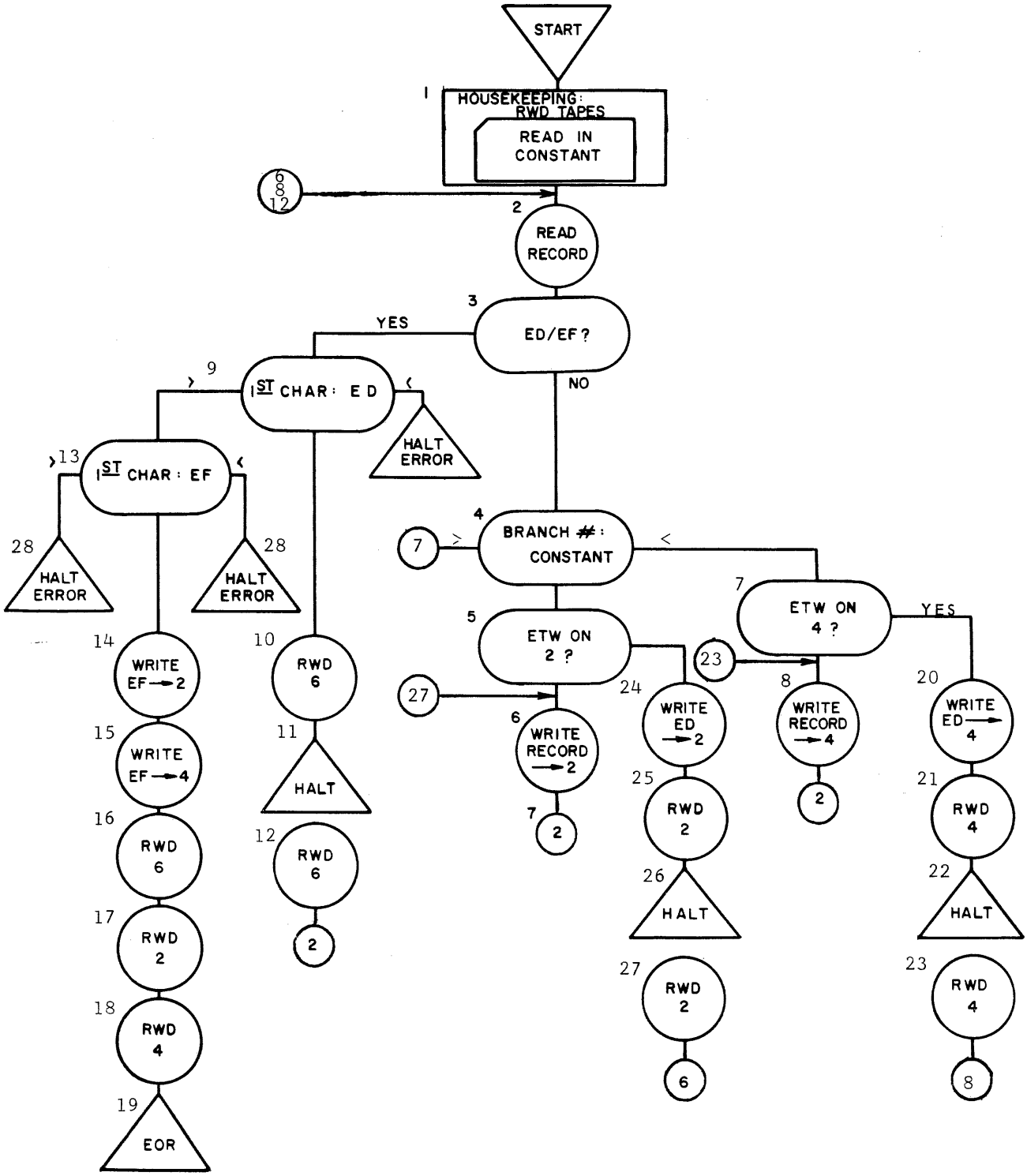
4) End of Job: a) Write EF to output tapes; b) Rewind tapes; c) Stop

OUTPUT:

1) A branch master tape containing all messages pertaining to the branch indicated on the card.

2) A new master tape containing all other messages.

See Note Book


START
1
HOUSEKEEPING: RWD TAPES
READ IN CONSTANT
6
8
12
2
READ RECORD
3
ED/EF?
YES
NO
9
1ST CHAR: E D
HALT ERROR
13
1ST CHAR: EF
28
HALT ERROR
28
HALT ERROR
4
7
BRANCH #: CONSTANT
7
ETW ON 4 ?
YES
5
ETW ON 2 ?
23
14
WRITE EF→2
10
RWD 6
27
6
WRITE RECORD →2
7
2
24
WRITE ED →2
8
WRITE RECORD →4
2
20
WRITE ED →4
15
WRITE EF→4
11
HALT
25
RWD 2
21
RWD 4
16
RWD 6
12
RWD 6
26
HALT
22
HALT
17
RWD 2
2
27
RWD 2
23
RWD 4
18
RWD 4
6
8
19
EOR

	00 01 02 03 04	05 06 07 08 09	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	
20	BR # ACC T —— # ——→		←ID←A C T D A TE ←——		BAL ——→						
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
21	X X X										
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
22											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
23	PROG ——→										
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
24											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
25											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
26											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
27											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
28											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
29											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE DECISION MAKING EXERCISE

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	1	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		229 0	0	0									E/F	F/D			
	6	230 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD INPUT TAPE	1
		1 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD OUTPUT TAPE	1
		2 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD OUTPUT TAPE	1
		3 0	0	0	0	1	2	1	0	0	0	0	0	0		READ CARD WITH BRANCH CONSTANT	1
2430 2400 2490		4 0	0	0	4	6	2	0	0	0	2	0	2	4		READ IN A RECORD	2
		5 0	0	0	W	8	2	4	4	0	2	3	6	0		SENSE FOR ED/EF	3
2350	6	236 0	0	0	Y	3	2	0	0	0	2	1	0	0		COMPARE BRANCH # : BRANCH CONSTANT	4
		7 0	0	0	W	1	2	4	1	0	2	4	1	0		B # > BC → 8 B # < BC → 8	4
		8 0	0	0	S	2	4	0	0	0	2	6	3	0		ETW ON 2	5
2670		9 0	0	0	8	2	2	0	0	0	2	0	2	4		WRITE RECORD TO TAPE 2	6
		240 0	0	0	V	1	0	2	1	9	2	3	4	0		TRANSFER → 2	
2370		1 0	0	0	S	4	4	0	0	0	2	5	8	0		ETW ON 4	7
2620	6	242 0	0	0	8	4	2	0	0	0	2	0	2	4		WRITE RECORD TO TAPE 4	8
		3 0	0	0	V	1	0	2	1	9	2	3	4	0		TRANSFER 2	
2350		4 0	0	0	Y	1	2	0	0	0	2	2	9	9		COMPARE 1st CHAR READ IN : ED	9
		5 0	0	0	W	1	2	5	0	0	2	6	8	0		1st C > ED → 16. 1st C < ED → HALT (ERROR)	9
		6 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD OLD TAPE	10
		7 0	0	0	⊡	0	0	0	0	0	0	0	0	0		STOP TO ALLOW OPERATOR TO MOUNT NEW TAPE	11

XIV-6

IE 1224 REV. 8-60

TITLE DECISION MAKING EXERCISE

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	248 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD NEW TAPE	12
		9 0	0	0	V	1	0	2	1	9	2	3	4	0		TRANSFER → 2	
2450		250 0	0	0	Y	1	2	0	0	0	2	2	9	8		COMPARE 1st C : EF	13
		1 0	0	0	W	1	2	6	8	0	2	6	8	0		CTC ≠ EF → HALT (ERROR)	13
		2 0	0	0	8	2	2	2	9	8	2	2	9	8		WRITE EF TO 2	14
		3 0	0	0	8	4	2	2	9	8	2	2	9	8		WRITE EF TO 4	15
	6	254 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD 6	16
		5 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2	17
		6 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	18
		7 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT (EOR)	19
2410		8 0	0	0	8	4	2	2	9	9	2	2	9	9		WRITE ED → 4	20
		9 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	21
	6	260 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT TO MOUNT NEW TAPE	22
		1 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	23
		2 0	0	0	V	1	0	2	1	9	2	4	2	0		TRANSFER → 9	
2380		3 0	0	0	8	2	2	2	9	9	2	2	9	9		WRITE ED → 2	24
		4 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2	25
		5 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT TO MOUNT NEW TAPE	26
	3	266 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2	27
		7 0	0	0	V	1	0	2	1	9	2	3	9	0		TRANSFER → 6	
2450 2510		8 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT ERROR	28
		0	0	0													
		0	0	0													
		0	0	0													

DECISION MAKING EXERCISE I

PROBLEM: A company wishes to run a sales analysis on certain stock items. A work tape is to be prepared for this later run by extracting all messages pertaining to those items from the master inventory file.

INPUT:

1) Master Inventory File on tape deck 5.

   a) One reel

7	6	25	8
STK #	ACTIVITY DATE	DESCRIPTION	QTY ON HAND

PROCESSING:

1) Read a master.

2) Select all messages pertaining to stock numbers 0000100 through 0001111 and write these messages to magnetic tape on deck 3. Make these constants a part of the program.

3) Write all other messages to a separate magnetic tape on deck 1.

4) End of Job: write EF to output tapes, rewind tapes, stop.

OUTPUT:

1) A work tape containing all active messages

   a) One reel

2) A second tape containing all inactive messages

   a) One reel

REVIEW EXERCISE I

PROBLEM: Update all the messages in a master retirement fund file having the format:

ACCT #	BALANCE
5	7
	(loaded with insignificant zeros)

1-5	7-13
ACCT #	CURRENT DEPOSIT
5	7
	(loaded with insignificant zeros)

ASSUMPTIONS:

1) Only one reel of master information.

2) If ED or ETW sensed, stop.

3) If transaction account number is less than the master account number, the transaction is out of sort.

4) There is only one transaction per message.

5) The EF of the reference file will occur at the same time as a blank terminal card.

PROCESSING:

1) Read the transaction into memory at 1500 from card.

2) Read the master message into memory at 1500 from deck 3.

3) Check for ED/EF. If EF, go to end of run routine.

4) Compare the Master Account Number and the transaction account number:

   a) If M > T
      -1-Sort error; stop

   b) If M < T
      -1-Sense the end of tape deck 1
      -2-Write out the master message on deck 1
      -3-Go back to read in a new master

   c) If M = T continue processing

5) Add the current deposit to the balance.

6) Sense the end of tape on deck 1 and write the updated master to deck 1.

7) Go back and read in a new transaction and a new reference.

REQUIRED:

1) Functional and Detailed Charts

2) Coding

3) HSM layout

XV – DATA HANDLING INSTRUCTIONS

(SUBTRACT, TRANSFER DATA RIGHT, TRANSFER DATA LEFT, TRANSFER SYMBOL TO FILL)

Having learned most of the essential instructions, we will now investigate some more of the Data Handling instructions.

SUBTRACT: Right → Left

Obviously, it is as important to be able to subtract as to add. The subtract instruction works in a manner similar to the add instruction. Again it is table look up arithmetic on equal length operands, with the difference replacing the minuend (minuend minus subtrahend equals difference) and the PRI's being affected.

The OP code is ⊖ (minus).

→ As in Addition and Compare

N gives the number of characters in each operand (0-44).

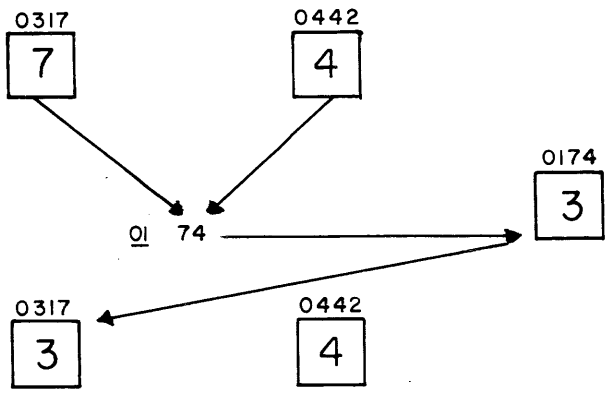
The A address gives the location of the least significant digit of the minuend, receiving Field and naturally, the difference.

The B address gives the location of the least significant digit of the subtrahend. Sending Field

All the rules are the same, with the exception that if the signs start out alike, the difference table is used; if they are initially unlike, the sum table is used.

Example:

⊖ 1 0317 0442 (assume signs are alike)



TRANSFER SYMBOL TO FILL: Left to Right

Suppose you should bring in a transaction that had a 7 digit amount and you were required to add this amount to a 10 digit field. Obviously, you must be able to place 3 insignificant zeros in front of this transaction amount. This could be accomplished with a TRANSFER SYMBOL TO FILL instruction which allows the programmer to fill any size sector of memory with any given character.

The OP code is a J.

N gives the character to be used.

The A address gives the left hand end address of the sector to be filled.

The B address gives the right hand end address of the sector to be filled.

Example:

In our problem stated above, we desire to load insignificant zeros. Let us say that the information is coming from cards, and that the format of the card is as follows:

Stock number	Amount
5	7
col 1-5	11-17

If we read this into memory starting at 8900, we would have:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
89	X	X	X	X	X	-	-	-	-	-	X	X	X	X	X	X	X

Since the location 8907-8909 are not used for data, we could fill these with zeros using the SF instruction:

J 0 8907 8909

Once the instruction is staticized, the character in N is placed in the location mentioned by the A register, A is increased by 1, and this process continues until the desired character is placed in the location mentioned by the B register. This would leave us with:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
89	X	X	X	X	X	-	-	0	0	0	X	X	X	X	X	X	X

which gives us a 10 character field to use as an addend.

TRANSFER DATA RIGHT:

Suppose we had the same problem, but the data was coming in from paper tape, so that when it fell in memory, there would be no locations free to place the zeros. In that case we would have to set up a "work area" which is simply a portion of memory utilized by the programmer as a "scratch pad". We could then fill the first three characters with zeros at the beginning of our program, and then (after reading in each new transaction) simply transfer the transaction amount from the original read in area to the work area. To do this, we will need an instruction that has the facility to transfer data from one location to another in memory. One such instruction is Transfer Data Right.

The OP code is N.

The "N" character indicates the number of characters in the sector to be transferred, and again we utilize the "N" table. The A address gives the right hand end location of the data to be transferred.

The B address gives the right hand end location of the area to receive the data.

Right → Left

Example:

Suppose the same data came into memory from paper tape and was placed in memory starting at 7500:

	00	01	02	03	04	05	06	07	08	09	10	11
75	1	2	3	4	5	3	7	4	2	9	0	0

STOCK # AMOUNT

We would then desire to transfer the amount to a work area which we have already loaded with three insignificant zeros. The work area is located at 1590-1599:

	90	91	92	93	94	95	96	97	98	99
15	0	0	0	3	7	4	2	9	0	0

Our transfer instruction would be:

N	7	7511	1599

The character addressed by A will be placed in the location designated by B. The A and B registers are then decreased by 1. N is decreased by one and sensed for 0. Since it is not zero the process will continue. When N does reach zero, the instruction will terminate. We will then have in memory:

	00	01	02	03	04	05	06	07	08	09	10	11
75	1	2	3	4	5	3	7	4	2	9	0	0

	90	91	92	93	94	95	96	97	98	99
15	0	0	0	3	7	4	2	9	0	0

As you can see, the transfer is not destructive, i.e., when the characters are picked up from their original locations, they are regenerated in those locations so that, at the termination of the instruction, they will appear in two places (the original location and the destination location).

TRANSFER DATA LEFT:

The same operation could have been accomplished moving the data from left to right instead of right to left.

The OP code is M.

"N" is still the number of characters to be transferred.

The A address gives the location of the leftmost character to be transferred.

The B address gives the location of the leftmost destination location.

The instruction:

M	7	7505	1593

would cause memory to end up in the same condition as the DR instruction did. The only difference is that the A and B registers would be increased by one after each transfer.

Left → Right

DATA HANDLING EXERCISE I:

PROBLEM: Perform a verifying run on Master Inventory File to assure ascending sequence of records.

INPUT: 1) Master Inventory File on deck 1.

a) One reel
b) Format

7	6	25	8
STK #	ACTIVITY DATE	DESCRIPTION	QTY ON HAND

PROCESSING: 1) Read a Master in

2) Sequence Check the criterion stock number. Assume that first record is in correct sequence and that multiple records concerning the same stock number are feasible.

3) Write Non-Sequence messages on an error tape, deck 3.

4) Write the properly ordered file to tape on deck 5.

5) When all messages have been processed, place EF on output tapes, rewind all tapes, and stop.

OUTPUT: 1) Master File in proper sequence on tape deck 5.

a) One reel

2) Out of sequence messages on error tape deck 3.

a) One reel

DATA HANDLING EXERCISE II

The problem is to update the master payroll information and then prepare a card to be punched out with weekly information.

INPUT: 1) Master Payroll Information (one reel)

EMP #	NAME	ST. ADDRESS	CITY-STATE ADDRESS	NET PER WEEK	TOTAL GROSS
5	25	18	20	6	7

2) Transaction information (cards)

	EMP #	GROSS PER WEEK
	5	6
cols:	1-5	6-11

PROCESSING: 1) Initially clear the punch area to spaces and a work area to zeros.

2) Read in a transaction.

3) Read in a master.

4) Compare the employee numbers. If the Master employee number is greater than the transaction employee number, assume a sort error and stop. If they are equal, continue to process. If the Transaction is greater than the Master, write out the master and bring in a new master. (Assume one transaction against a master.)

5) Processing consists of:

a) Updating the Total Gross item in the Master.
b) Transferring the following information so that a card can be produced with the given specifications:

EMP #	NAME	ST. ADD	C-S ADD	NET PER WEEK
1-5	7-31	33-50	52-71	75-80

c) Punch the card
d) Write out the master and bring in a new transaction and master

6) When all messages have been processed, place an EF on the output tape, rewind the tapes and stop.

REVIEW EXERCISE II: See Flow Chart

PROBLEM: To update the Master Checking Account Information of a Bank, and prepare an overdraft notification card if necessary.

INPUT: 1) Master Information (1 reel)

1089 - 92

ACCT #	NAME	ST. ADD	C-S ADD	TOTAL DEPOSITS	TOTAL CHECKS	BALANCE
7	25	18	19	9	9	6

(95 chars)

2) Transaction Information (cards)

	ACCT #	CODE	AMOUNT
	7	1	5
cols.	1-7	8	9-13

13

The code will either be a C or a D, standing for Check or Deposit.

Assume that there will never be more than one transaction per master.

PROCESSING: 1) The problem is to update the master by adding a deposit to both the total deposit amount and also to the balance, or adding a check to the total check amount and subtracting it from the balance.

2) After subtracting a check from the balance, the programmer should determine if the balance is negative (treat a zero balance as a plus). If so, an overdraft card should be prepared to be sent to the customer. The format of this card is:

ACCT #	NAME	ST. ADD	C-S ADD	BALANCE
1-7	9-33	35-52	54-72	74-79

XVI – PRINTING

PRINT AND PAPER ADVANCE

So far we have learned how to read from and write to magnetic tape, read and punch paper tape, and read and punch cards. The next output device is the printer.

The On-Line Printer prepares documents at a rate of 800 or 1000 lines per minute depending on the mode. A line may consist of 120 or 160 character locations.

The Printer sends a signal to the computer telling it what character on the print wheel is to be printed next. The computer uses this signal to develop an address in the print table which is stored in HSM. The character addressed in the print table is extracted and stored in the Print Register. The data to be printed is compared with the character stored in the Print Register until 120 or 160 consecutive memory locations have been checked. This checking for one character is known as a "scan". As each character is checked, a one bit is sent to the printer shift register for an equality, a zero bit is sent for each non-equality. At the completion of a scan, the shift register contains a one bit in every position on the line in which this character is to be printed. The one bits are then used to trigger the corresponding print hammers. The following is the Print Table and where it is stored in HSM.

Character	Table Location Memory Size			Character	Table Location Memory Size		
	10,000	20,000	40,000		10,000	20,000	40,000
- Minus	9900	I900	Z900	A Letter	9940	I940	Z940
+ Plus	9901	I901	Z901	B "	9941	I941	Z941
Space	9902	I902	Z902	C "	9942	I942	Z942
0 Zero	9903	I903	Z903	D "	9943	I943	Z943
1 One	9904	I904	Z904	E "	9944	I944	Z944
2 Two	9905	I905	Z905	F "	9945	I945	Z945
3 Three	9906	I906	Z906	G "	9946	I946	Z946
4 Four	9907	I907	Z907	H "	9947	I947	Z947
5 Five	9910	I910	Z910	I "	9950	I950	Z950
6 Six	9911	I911	Z911	J "	9951	I951	Z951
7 Seven	9912	I912	Z912	K "	9952	I952	Z952
8 Eight	9913	I913	Z913	L "	9953	I953	Z953
9 Nine	9914	I914	Z914	M "	9954	I954	Z954
, Comma	9915	I915	Z915	N "	9955	I955	Z955
. Period	9916	I916	Z916	O "	9956	I956	Z956
@ At	9917	I917	Z917	P "	9957	I957	Z957
% Percent	9920	I920	Z920	Q "	9960	I960	Z960
: Colon	9921	I921	Z921	R "	9961	I961	Z961
# Number	9922	I922	Z922	S "	9962	I962	Z962
$ Dollar Sign	9923	I923	Z923	T "	9963	I963	Z963
) Close Parenthesis	9924	I924	Z924	U "	9964	I964	Z964
" Quote	9925	I925	Z925	V "	9965	I965	Z965
$_{10}$ Sub 10	9926	I926	Z926	W "	9966	I966	Z966
(Open Parenthesis	9927	I927	Z927	X "	9967	I967	Z967
] Close Bracket	9930	I930	Z930	Y "	9970	I970	Z970
; Semicolon	9931	I931	Z931	Z "	9971	I971	Z971
> Greater Than	9932	I932	Z932	CR Credit Symbol	9972	I972	Z972
÷ Divide	9933	I933	Z933	' Apostrophe	9973	I973	Z973
↑ Arrow Up	9934	I934	Z934	* Asterisk	9974	I974	Z974
[Open Bracket	9935	I935	Z935	& Ampersand	9975	I975	Z975
< Less Than	9936	I936	Z936	/ Virgule	9976	I976	Z976
= Equal Sign	9937	I937	Z937	¤ Lozenge	9977	I977	Z977

* Four of the 64 code configurations will not be available as standard 301 codes but can be generated by computer programming. Also, some of the 301 symbols do not appear in this list so that other symbols may be chosen to represent the 301 symbols. It is advisable to prevent energizing the hammers for spaces by placing the code $(17)_8$ in location 9902, I902, or Z902. In fact, the code $(17)_8$ can be placed in any location in the table when it is desired to prevent printing of the associated character. For this reason, too, any $(17)_8$ codes appearing in the print area will never cause printing of any character.

The Synchronous Mode permits only 47 characters to be printed since a complete line of print and paper advance to the next print line is accomplished during each revolution of the print drum. The letter "A" is always the first character for which HSM is scanned, and the period is the last. The Asynchronous Mode permits all 64 characters on the print drum to be printed in one full revolution, and paper advance occurs in a portion of the next print drum revolution. Scanning can proceed with the next character in sequence on the print drum at the conclusion of the paper advance function.

Once the full line has been printed, it is necessary to advance the paper so that the next line will not be printed on top of this one. This is done in one of two ways:

1) the computer can count the number of lines to advance, or

2) the paper advance can be controlled by a tape loop.

The tape loop mentioned can be thought of as a reinforced piece of paper tape. It is divided into two parts or "channels", the first controlling the Page Change (21) type of paper advance and the second controlling the Vertical Tabulation (20). This tape is punched so that the holes correspond to the locations of the lines to be printed. In this manner, it corresponds to the vertical layout of several sheets. This tape is then spliced together to form a loop and is placed on the printer so that it corresponds with the blank paper.

Once a line is printed, and paper advance under control of the VT channel of the tape loop is requested, the paper will advance until it senses the first hole in the VT channel.

This occurs when the tape, which is passing between a photo-electric cell and a light, allows the light to hit the cell. This stops the paper, and the printer is ready to print the next line. The same sort of procedure would be followed if we requested control by the PC channel, the difference being that the sense would be made for a hole in the page change channel.

PRINT AND PAPER ADVANCE NORMAL:

This instruction allows the printing of one 120 or 160 character line on the On-Line Printer. It can also cause the positioning of paper for the next line of writing.

The OP code is B.

"N" supplies 2 types of information. The numeric portion indicates the number of lines to be advanced if the computer is to count the lines. This is limited to 0-14. The zone bit, 2^4, indicates which of two possible printers is to be used.

ASYNCHRONOUS MODE																SYNCHRONOUS
LINE #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
1st PRINTER	0	1	2	3	4	5	6	7	8	9	_	#	@	(	)	J
2nd PRINTER	&	A	B	C	D	E	F	G	H	I	+	□	;	:	,	/

The A address must be zeros (0000).

The B address contains the LHE of data to be printed excluding B_3.

B_0 - MSD of address
B_1 - if even, printing will occur. If odd, printing will not occur.
B_2 - always zero
B_3 - indicates type of paper advance.

B_3	TYPE OF PAPER ADVANCE
0	None
1	Paper Advance using N Register
2	Vertical Tab (using Tape Loop)
3	Page Change (using Tape Loop)

It is up to the programmer to edit the line for printing.

Example:

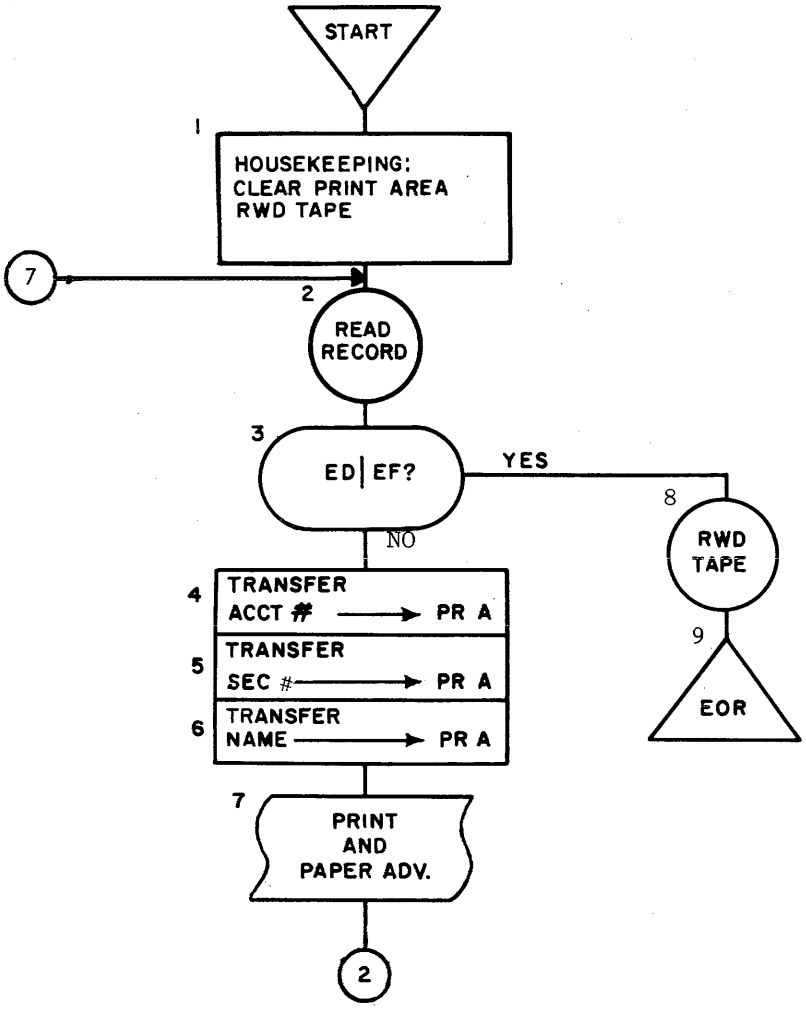
The problem is to print out a list of employee numbers, section numbers, and names. The information is coming in from magnetic tape mounted on tape station 3 (one reel). The input is in the format:

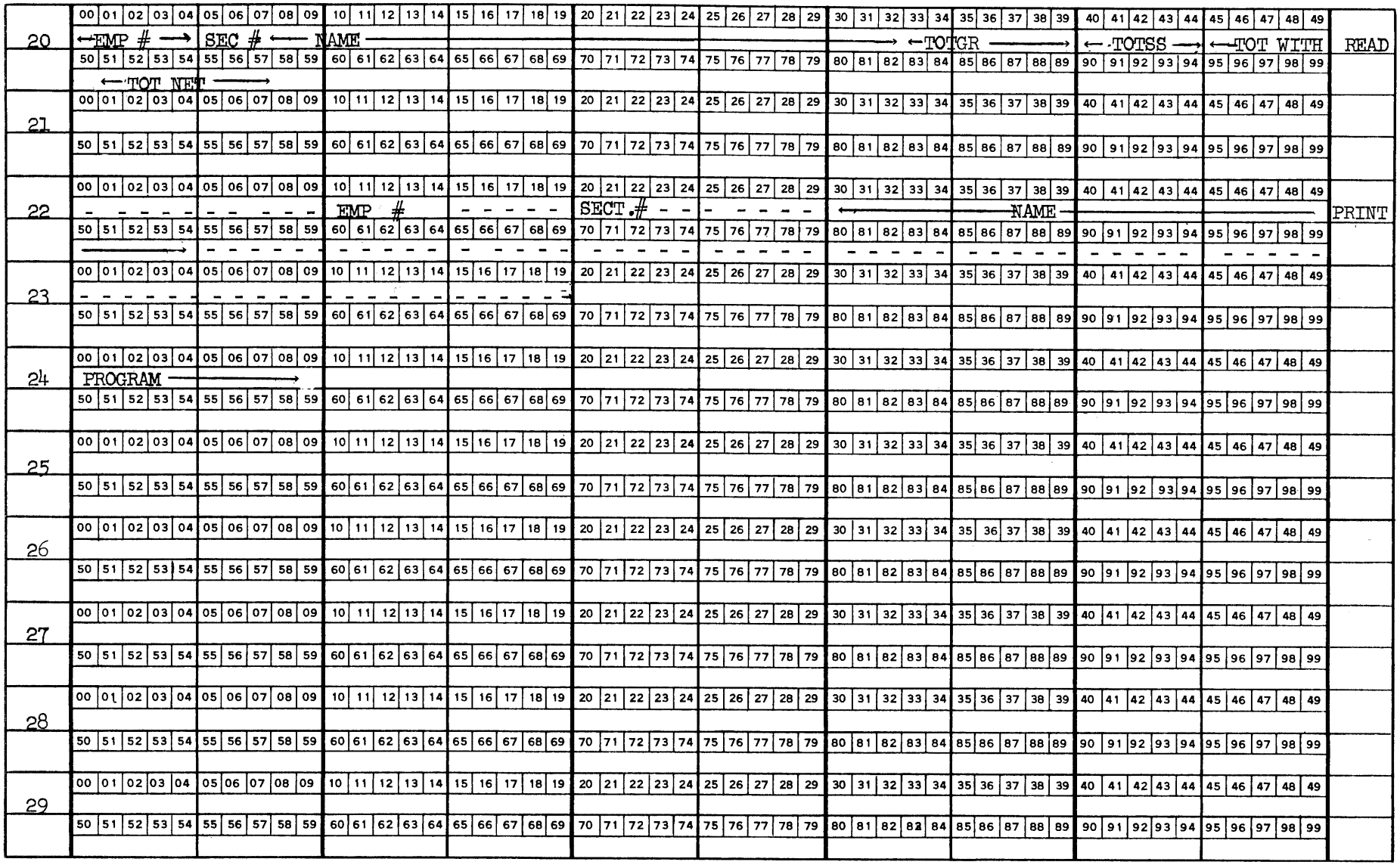
ACCT #	SEC #	NAME	TOTAL GROSS	TOTAL SS	TOTAL WITH.	TOTAL NET
5	3	25	7	5	6	7

The print format is to be:

	ACCOUNT #	SECTION #	NAME
Print Positions ——→	11-15	21-23	31-56

Assume that the VT channel of the tape loop has been punched to take care of all paper advance desired.


START
1
HOUSEKEEPING:
CLEAR PRINT AREA
RWD TAPE
7
2
READ
RECORD
3
ED | EF?
YES
8
RWD
TAPE
9
EOR
NO
4
TRANSFER
ACCT # → PR A
5
TRANSFER
SEC # → PR A
6
TRANSFER
NAME → PR A
7
PRINT
AND
PAPER ADV.
2



XVI-6

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE PRINT EXAMPLE

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	240 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	1
		1 0	0	0	J	⊟	2	2	0	0	2	3	1	9		CLEAR PRINT AREA TO SPACES	1
2480		2 0	0	0	4	3	2	0	0	0	2	0	5	7		READ IN ONE RECORD	2
		3 0	0	0	W	8	2	4	9	0	2	4	4	0		SENSE FOR ED/EF	3
		4 0	0	0	M	5	2	0	0	0	2	2	1	0		TRANS EMP # ⟶ PRINT AREA	4
2430		5 0	0	0	M	3	2	0	0	5	2	2	2	0		TRANS SEC # ⟶ PRINT AREA	5
	5	246 0	0	0	M	N	2	0	0	8	2	2	3	0		TRANS NAME ⟶ PRINT AREA	6
		7 0	0	0	B	0	0	0	0	0	2	2	0	2		PRINT AND PA BY VT	7
		8 0	0	0	V	1	0	2	1	9	2	4	2	0		TRANSFER ⟶ 2	
2430		9 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	8
		250 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT (EOR)	9
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XVI-7

IE 1224 REV. 8-60

PRINT EXERCISE:

PROBLEM: To prepare a list of names and addresses of each account in the file.

INPUT: A file containing account information of the format:

ACCOUNT #	NAME	ST. ADD	C-S ADD	TOTAL CREDITS	TOTAL DEBITS	BAL.
7	25	30	25	10	10	8

This file is multi-reel.

PROCEDURE:

1) Read in a record.

2) Print out the name on one line (starting at the first print position), the street address on a second line directly under it, and the city-state address on a third line (single space between the three lines and assume the VT channel is set up for the desired paper advance between the records).

3) Do this for every record in the file, allowing for a new reel when required.

(Repetitive)

XVII – ITERATIVE CODING

(TALLY)

A programmer often wishes to be able to repeat a portion of his program a given number of times and then go on to do something else. This is called "iterative coding".

One example of this might be a program which is to write out 100 records to a work tape and then stop. This means that, instead of a program transfer instruction at the end of the main path, we need a type of conditional transfer. One way it could be handled is to keep count of the number of writes which have been performed, by adding a 001 to a work area which was initially cleared to 000. Then we would have to continually sense this counter work area for 100.

Until it is found, we can continually go back to read in a new record and then write it out. When it is found, we can go to an end of run routine. These steps, however, would greatly increase our running time, which, of course, we are trying to keep at a minimum. For this reason, we have one instruction called TALLY that works in a reverse direction. That is, it starts out with the maximum number and decreases by one every time it is executed.

TALLY:

The OP code for this instruction is X.

The "N" character is zero.

Tally

The A address gives the HSM location of the diad containing the quantity to be tested. This is often referred to as a Tally Counter. The maximum value of the tally quantity is 99.

The B address gives the location of the next instruction to be performed if the quantity being tested has not been exhausted. When the quantity has been exhausted, the next instruction in sequence will be executed.

Example I:

Our problem was to write the first 100 records of a file out to tape. Our input contains records, 200 characters in length, and it is mounted on tape station A. The output is to be on tape station C.

TITLE PRINT EXAMPLE 1
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		449 0	0	0	0	0	0	0	9	9	0	0	0	E/F		TALLY CTR EF	
	6	450 0	0	0	;	A	0	0	0	0	0	0	0	0		RWD A	1
		1 0	0	0	;	C	0	0	0	0	0	0	0	0		RWD C	1
4540		2 0	0	0	4	A	3	0	0	0	3	1	9	9		READ A RECORD	2
		3 0	0	0	8	C	3	0	0	0	3	1	9	9		WRITE RECORD	3
		4 0	0	0	X	0	4	4	9	5	4	5	2	0		TALLY 99 TIMES	4
		5 0	0	0	8	C	4	4	9	9	4	4	9	9		WRITE EF TO C	5
	3	456 0	0	0	;	A	0	0	0	0	0	0	0	0		RWD A	6
		7 0	0	0	;	C	0	0	0	0	0	0	0	0		RWD C	7
		8 0	0	0	.	0	0	0	0	0	0	0	0	0		HALT (EOR)	8
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

IE 1224 REV. 8-60

You will note that the tally quantity is 99. Since we have already executed the write one time before we came to the Tally, we desire to transfer back only 99 times.

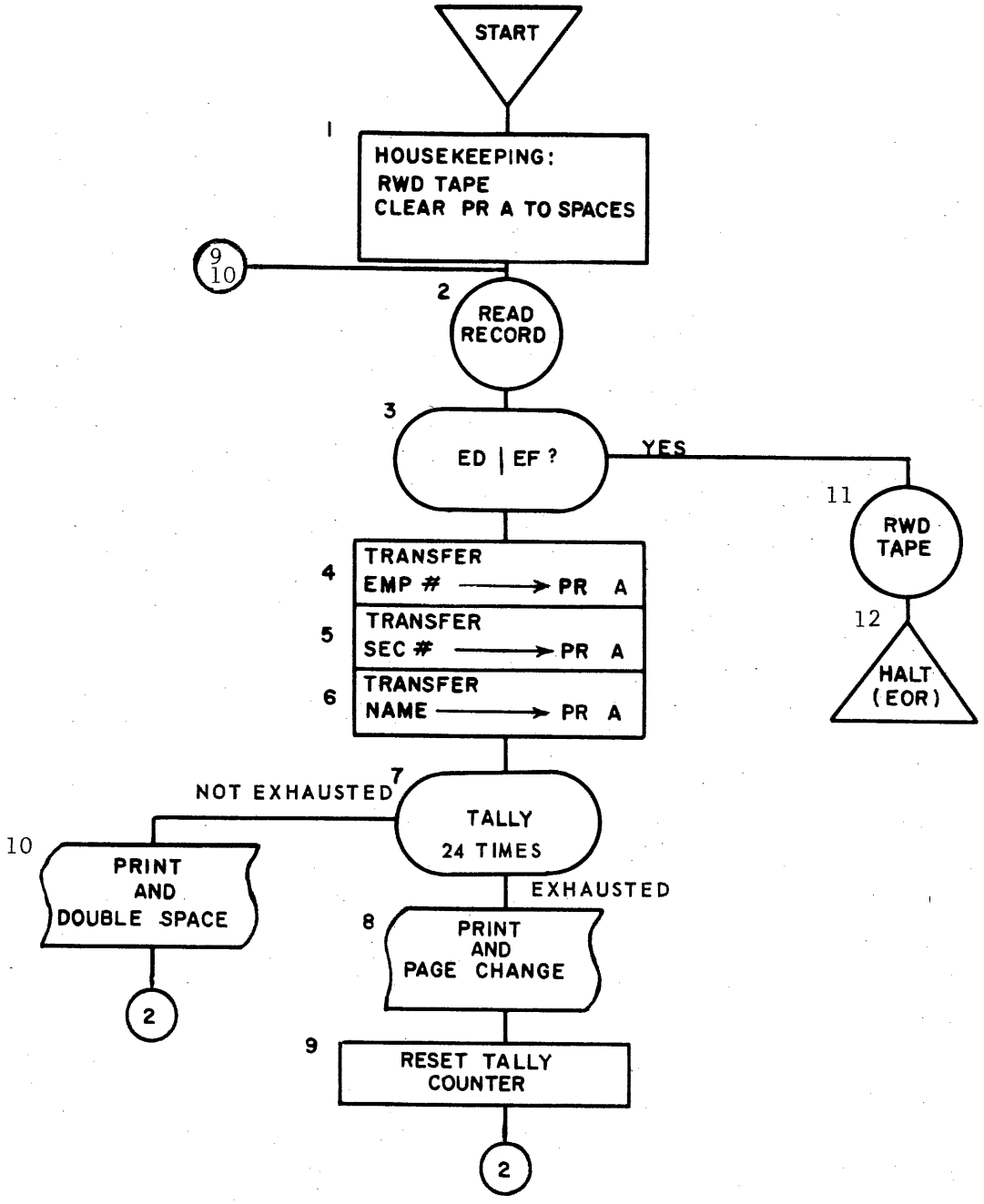
When the Tally instruction is executed, the contents of the tally counter addressed by A are sent to the D register. If the quantity is not 00, the contents of the P register are sent to STP and the contents of the B register are transferred to the P register. The contents of the D register are decreased by 1 and replaced at the address designated by A. If the D register does contain 00, the next instruction is staticized without affecting the P register.

Example 2:

A second example of the use of a Tally is connected with printing. Our problem (stated in the last chapter) was to print out each employee's number, section and name on one line.

We assumed that the VT channel of the tape loop had been punched to take care of any paper advance. Suppose, however, that this was not true, and we had to arrange to do this in our program. Our particular size of paper allows us to print the information concerning 25 employees on one page. Having accomplished this, we want to page change and then repeat the process. In other words, after printing 25 lines, we want to cause the paper advance control to be thrown to the PC channel, rather than having the computer cause the printer to double space. In order to do this, we have to be able to keep a count of the lines we have printed.

Again the Tally can be utilized. We can tally to the regular print routine 24 times. On the 25th, we will go to the print routine that executes a Page Change. Notice that we must reset the tally counter after printing out each group of 25 lines. This is necessary since the tally quantity is physically reduced each time the instruction is executed. If we did not set it back to 24, it would register 00, and from that point on we would print one line, page change, print another line, page change, etc.


START
1
HOUSEKEEPING:
RWD TAPE
CLEAR PR A TO SPACES
9
10
2
READ
RECORD
3
ED | EF ?
YES
11
RWD
TAPE
12
HALT
(EOR)
4
TRANSFER
EMP # → PR A
5
TRANSFER
SEC # → PR A
6
TRANSFER
NAME → PR A
7
NOT EXHAUSTED
TALLY
24 TIMES
10
PRINT
AND
DOUBLE SPACE
2
EXHAUSTED
8
PRINT
AND
PAGE CHANGE
9
RESET TALLY
COUNTER
2

TITLE PRINT EXERCISE 2

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	1	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		239 0	0	0	0	E/F	0	0	2	4	0	0	2	4		EF TALLY QTY TALLY CTR	
	6	240 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	1
		1 0	0	0	J	_	2	2	0	0	2	3	1	9		CLEAR PRINT AREA TO SPACES	1
2500 2520		2 0	0	0	4	3	2	0	0	0	2	0	5	7		READ A RECORD	2
		3 0	0	0	W	8	2	5	3	0	2	4	4	0		SENSE ED/EF	3
		4 0	0	0	M	5	2	0	0	0	2	2	1	0		TRANSFER EMP # → PRINT AREA	4
		5 0	0	0	M	3	2	0	0	5	2	2	2	0		TRANSFER SEC # → PRINT AREA	5
	6	246 0	0	0	M	N	2	0	0	8	2	2	3	0		TRANSFER NAME → PRINT AREA	6
		7 0	0	0	X	0	2	3	9	9	2	5	1	0		TALLY 24 TIMES	7
		8 0	0	0	B	0	0	0	0	0	2	2	0	3		PRINT WITH PC	8
		9 0	0	0	N	2	2	3	9	5	2	3	9	9		RESET TALLY CTR	9
		250 0	0	0	V	1	0	2	1	9	2	4	2	0		TRANSFER → 2	
2470		1 0	0	0	B	2	0	0	0	0	2	2	0	1		PRINT WITH DOUBLE SPACE	10
	6	252 0	0	0	V	1	0	2	1	9	2	4	2	0		TRANSFER → 2	
2430		3 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	11
		4 0	0	0	□	0	0	0	0	0	0	0	0	0		HALT (EOR)	12
		0	0	0													
		0	0	0													
		0	0	0													

ITERATIVE CODING EXERCISE I

PROBLEM: Search a file for the first ten messages containing a specified identification code.

INPUT: 1) Master Sales File mounted on tape E.

a) One reel
b) Format

CODE	SALESMAN'S NAME	REGION	BR. OFFICE	YRS. SERVICE	SALES TO DATE	QUOTA
2	25	8	8	2	9	9

PROCESSING: 1) Search the master file for the first ten messages which contain the identification code brought in from a card.

2) Write the selected messages to tape trunk 5

3) Write the other messages to tape trunk 1

4) When 10 selected messages have been written go to end of job procedure.

5) End of job procedure:

a) Write EF to output tapes.
b) Rewind all tapes.
c) Stop.

OUTPUT: 1) One reel containing all selected messages.

2) One reel containing all non-selected messages.

ITERATIVE CODING EXERCISE II

PROBLEM: Using the same problem mentioned in the chapter on printing, page XVI-8, single space between the lines of one address, triple space between addresses, and page change after printing 10 addresses.

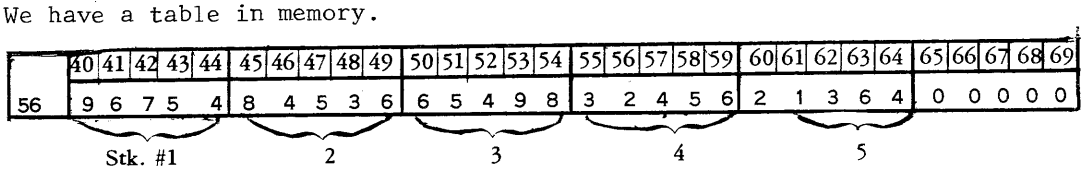
XVIII – ADDRESS MODIFICATION

(LOCATE SYMBOL LEFT)

In all of our programs to date the instructions we wrote remained intact throughout the running of the program. It is quite possible, however, that we might desire to change certain addresses within the program. This process is called "address modification".

One example of this is illustrated in the following program:

We have a table in memory.



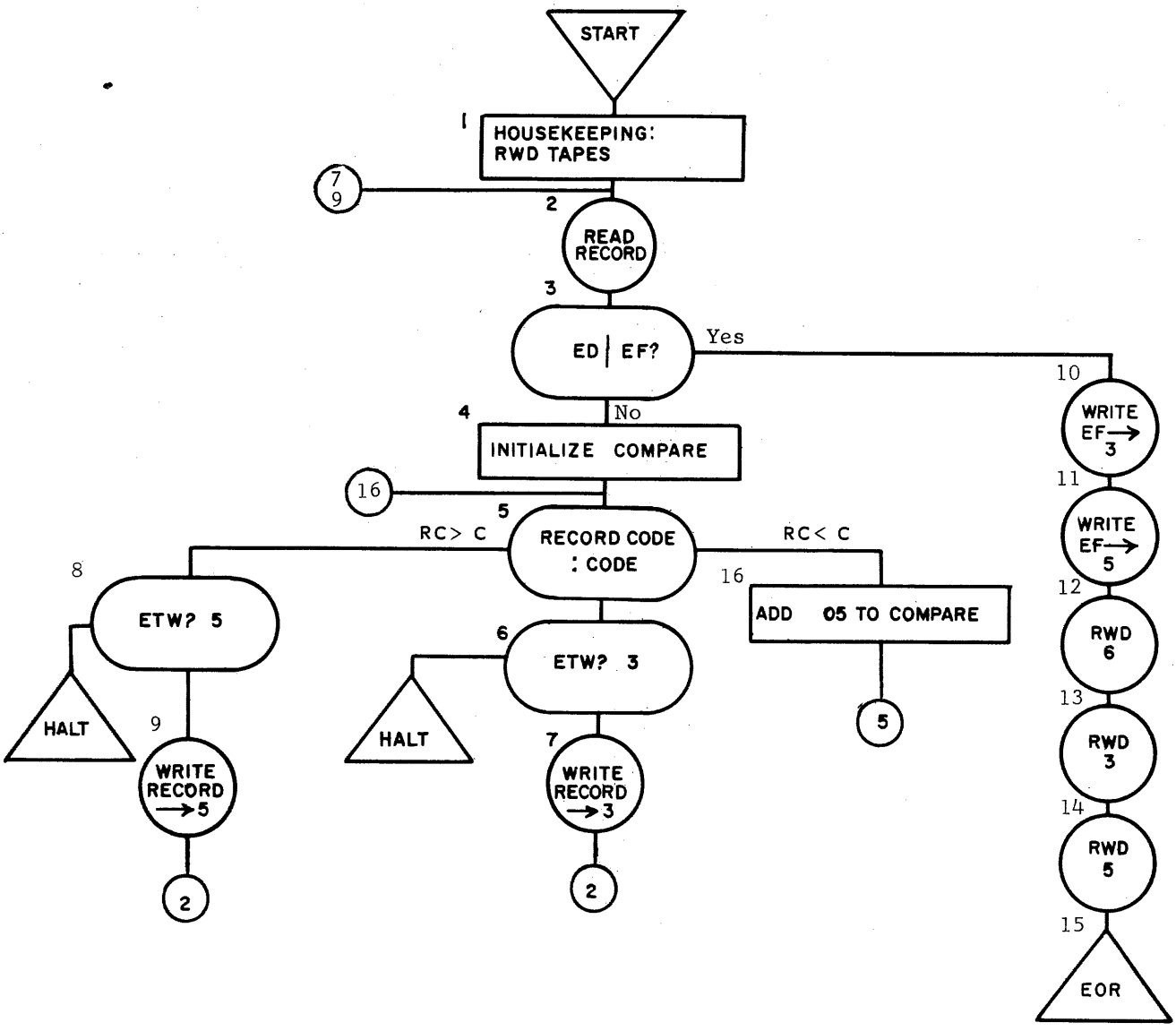
Our only job is to verify that all the records in our file have a stock code that agrees with one of the above. The zeros are placed at the end to terminate the loop. For example, if our first message had a stock code of 43257, it would be less than 96754, so we would compare it to 84536. Again it is less, so the next comparison would be to 65498. The next step would be to compare it to 32456. Since it is greater than 32456 but less than 65498, it must be an erroneous code. All codes will be greater than 00000, so a code of 10324 would be recognized as erroneous.

The problem is to keep the number of programming steps at a minimum. Since the MSD of each code is located exactly 5 positions away from the last MSD, wouldn't it be possible to start out comparing against the first code and if it is necessary to then compare against the second code, simply add 05 to the compare instruction? This loop could be maintained until the code matches, or until the record is proven to be illegitimate. We must remember, however, that if we are going to modify the address of the compare while processing one record, it will be necessary to re-initiate it before starting to process the next record. If we do not, we might have found a match on code 32456 on this record. When we bring in the next record, however, its code is 84536. If we did not reset the compare to start again at the beginning, this would register as a "greater than" and therefore be rejected, when, in fact, it is a perfectly valid record.

INPUT: 1 reel of master information on trunk 6:

STOCK CODE	STOCK NUMBER	BALANCE ON HAND	DUE IN
5	8	10	7

OUTPUT: 1) 1 reel of valid masters on trunk 3.
2) 1 reel of invalid masters on trunk 5.


START
1
HOUSEKEEPING:
RWD TAPES
7
9
2
READ
RECORD
3
ED | EF?
Yes
No
10
WRITE
EF→
3
4
INITIALIZE COMPARE
11
WRITE
EF→
5
16
5
RC> C
RECORD CODE
: CODE
RC< C
8
16
ETW? 5
ADD 05 TO COMPARE
12
RWD
6
6
ETW? 3
13
RWD
3
HALT
9
HALT
5
7
14
WRITE
RECORD
→5
WRITE
RECORD
→3
RWD
5
15
2
2
EOR

	00 01 02 03 04	05 06 07 08 09	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	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
51	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
52	STK CODE ←	STK # →		BAL →	DUE	IN →					
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
53	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
54	PROGRAM →										
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
55	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
56	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
57	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
58	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
59	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE ADDRESS MODIFICATION EXAMPLE 1

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	540 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD 6	1
		1 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	1
		2 0	0	0	;	5	0	0	0	0	0	0	0	0		RWD 5	1
5500 5530		3 0	0	0	4	6	5	2	0	0	5	2	2	9		READ RECORD	2
		4 0	0	0	W	8	5	5	4	0	5	4	5	0		ED/EF SENSE	3
		5 0	0	0	N	2	5	6	3	5	5	4	6	9		INITIALIZE COMPARE	4
5610	6	546 0	0	0	Y	5	5	2	0	0	5	6(	0	0)	4, 18	COMPARE RECORD : CODE	5
		7 0	0	0	W	1	5	5	1	0	5	6	0	0		R > C → 9 R < C → 18	5
		8 0	0	0	S	3	4	0	0	0	5	6	2	0		ETW ON 3	6
		9 0	0	0	8	3	5	2	0	0	5	2	2	9		WRITE RECORD → 3	7
		550 0	0	0	V	1	0	2	1	9	5	4	3	0		TRANSFER → 2	
5470		1 0	0	0	S	5	4	0	0	0	5	6	2	0		ETW ON 5	8
	6	552 0	0	0	8	5	5	2	0	0	5	2	2	9		WRITE RECORD → 5	9
		3 0	0	0	V	1	0	2	1	9	5	4	3	0		TRANSFER → 2	
5440		4 0	0	0	8	3	5	6	3	1	5	6	3	1		WRITE EF → 3	10
		5 0	0	0	8	5	5	6	3	1	5	6	3	1		WRITE EF → 5	11
		6 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	12
		7 0	0	0	;	5	0	0	0	0	0	0	0	0		RWD 5	13
	6	558 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD 6	14
		9 0	0	0	□	0	0	0	0	0	0	0	0	0		HALT EOR	15
5470		560 0	0	0	+	2	5	4	6	9	5	6	3	9		ADD 05 TO ADDRESS (5B)	16
		1 0	0	0	V	1	0	2	1	9	5	4	6	0		TRANSFER → 5	
5510 5480		2 0	0	0	□	0	0	0	0	0	0	0	0	0		ERROR HALT	
		3 0	0	0	□	E/F	0	0	4	0	0	0	0	5		CONSTANTS: EF; INIT ADD; INC. AMT	

IE 1224 REV. 8-60

Another example of address modification also ties back to printing. In the print examples covered up to this time, we have only been printing fixed alpha or alphanumeric information. Suppose, however, we wanted to print a balance. Since we are carrying it in a 7 digit (6 characters maximum, 1 space for overflow) fixed field on tape, a balance of $107.54 would appear as 0010754, and a balance of $6.98 would appear as 0000698. Once we place the balance in the print area, we must change all the insignificant zeros to spaces so that they will not appear on the printed copy. In order to do this we must discuss a new instruction called LOCATE SYMBOL LEFT.

LOCATE SYMBOL LEFT:

This instruction searches a sector of memory looking for the lack of a given symbol. For example, in our case, we would want to locate the rightmost insignificant zero in the balance area.

The OP code is K. Search or find symbol

N indicates the selected symbol.

The A address gives the left hand end of the area to be searched.

The B address gives the right hand end of the area to be searched.

If the balance item were placed in the print area between 2434-2440, our instruction would be:

K 0 2434 2440

Once the instruction is staticized, the contents of the HSM location specified by the A register is compared against the contents of the N register. If the character is like that in N, and A does not equal B, the A register is increased by one and the operation is repeated. The operation will terminate when either:

1) A equals B or
2) a symbol unlike the one in N is found. In this case, the A register is decreased by 1, so that the A register would be addressing the last symbol found (in our case, the last zero).

In addition, the PRI's will be affected as follows:

PRN is set when the first character searched is not equal to the contents of N.

PRZ is set when all characters searched are equal to the contents of N.

PRP is set if a non-symbol is found after a character equal to the contents of N has been found.

Now, if we have the ability to transfer the contents of the A register into the B address of a Transfer Symbol to Fill instruction, we would be able to replace the zeros with spaces. This is true since we know the symbol to fill would be a blank and that the left hand end of the field would be the left hand end of the balance or 2434. The only thing we didn't know originally was the right hand end of the sector to be cleared, and we have just found that. The problem, therefore, is to get to the contents of the A register.

The first instruction that occurs to us is the STORE REGISTER instruction since we already know that we have the ability to store the contents of the P register utilizing this instruction. This presents a difficulty, due to the fact that the REG instruction itself utilizes the A register, so that before we could remove the address we needed, we would destroy it. For this reason, the engineers supplied us with an automatic storage of the final contents of the A register in practically every case where we might need it.

Again we will be working with a standard high speed memory location. This time 0212-0215 are the locations, and they are called STA (Store A). At the termination of certain instructions, the final contents of the A register are placed automatically into STA. Once there, the programmer has the ability to transfer them wherever they may be needed. He must remember, however, that they are there only temporarily, since the next instruction that affects STA will change its contents.

The solution to our problem is now simple. All we must do is locate the right hand least significant zero, transfer the contents of STA to the B address of a SF instruction, and then fill the zeros with blanks. The routine would look as follows: (Assume one zero.)

```
3500   K   0   2434   2440   Locate MSD of balance
3510   N   4   0215   3529   Trans. STA to B address
3520   J   _   2434   (  )   Replace zeros with spaces
```

Using the two examples we mentioned before, let's follow the routine through:

	34	35	36	37	38	39	40
24	0	0	1	0	7	5	4

Locating the last zero would put 2435 into STA.

Transfer the contents of STA to the B address of the SF instruction so that it now reads:

J _ 2434 (2435)

Execute that instruction so that you have in memory:

24	34	35	36	37	38	39	40
	-	-	1	0	7	5	4

Starting out with:

	34	35	36	37	38	39	40
24	0	0	0	0	6	9	8

Locating the last zero would place in STA 2437.

Transferring this to the B address would give us an instruction that reads:

J _ 2434 2437.

Executing this instruction would yield:

	34	35	36	37	38	39	40
24	-	-	-	-	6	9	8

ADDRESS MODIFICATION EXERCISE I:

PROBLEMS: Post a set of totals developed in a previous run to a table of year-to-date totals maintained on magnetic tape.

INPUT:
1) Magnetic tape containing a table of 100 ten digit master totals
   a) One reel
   b) On tape 3

2) Magnetic tape containing a table of 100 ten digit transaction totals
   a) One reel
   b) On tape 5

PROCESSING:
1) Housekeeping
   a) Rewind all tapes

2) Read a master total
3) Read in transaction total
4) Add the 100 transaction totals to the 100 master totals. The first transaction total is to be added to the first master total, the second transaction total is to be added to the second master total, etc. The new sum will never exceed 10 digits.
5) Write the updated 10 digit total table to tape station 1.
6) End of job procedure:
   a) Write EF to output tape.
   b) Rewind all tapes.

OUTPUT:
1) Updated table on magnetic tape.
   a) One reel.

ADDRESS MODIFICATION EXERCISE II:

PROBLEM: Prepare and print order forms from the data coming in from a stock file:

INPUT: One reel of master stock information in the format:

STOCK #	MFG'S NAME	ST. ADD	C-S ADD	BALANCE	ORDER QTY	DUE IN
9	25	20	20	10	8	8

The numeric fields (balance, order quantity and due in) are loaded with insignificant zeros.

OUTPUT: Printed order forms in the format:

	STOCK #	MFG'S NAME	ST. ADD	C-S ADD.	ORDER QTY.
Print Pos.	1-9	20-44	50-69	75-94	113-120

Advance 5 lines between lines and page change after every 10 order forms.

ADDRESS MODIFICATION EXERCISE III:

PROBLEM: To edit a tape which is supposed to contain one of a possible ten stock numbers.

Station F = 6
A = 1
C = 3

INPUT:
1) An inventory file on one reel of magnetic tape, mounted on Station C. (100 character records; the stock number is the first 8 characters).
2) A table which is to be read into memory from a card with 10 eight digit codes, in ascending order.

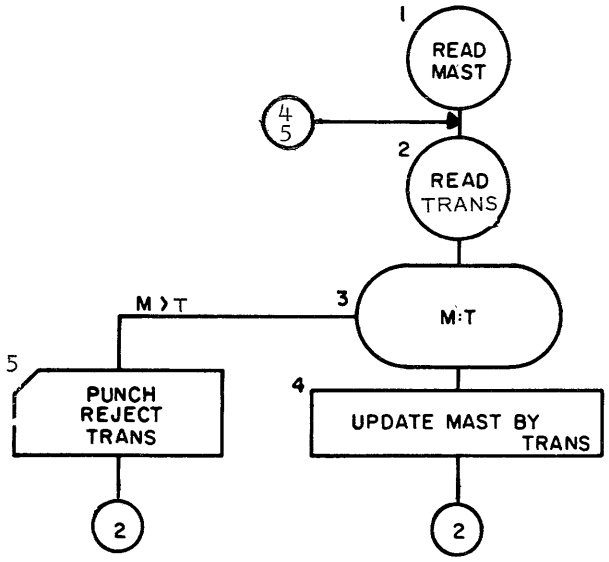
PROCESSING:
1) If the message doesn't contain a valid part number, write that message to the Station A. If the message does contain a valid part number write the message to tape F. 6
2) If an EF is detected write EF's, rewind tapes, and halt.
3) Use a tally to stop the program loop.

OUTPUT:
1) A tape of valid messages.
   a) One reel.
2) A tape of invalid messages.

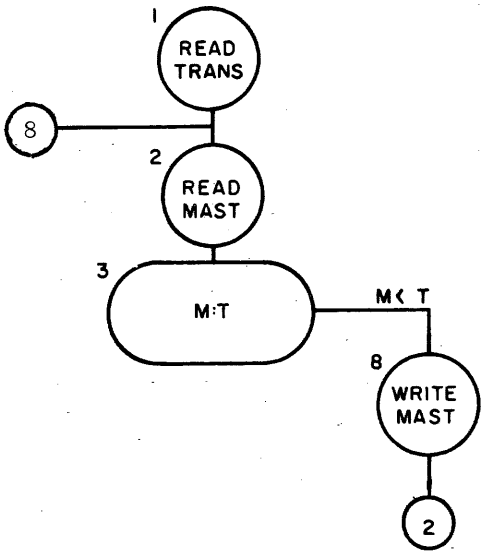
XIX – VARIABLE CONNECTORS

In our examples so far we have assumed that there would be no more than one transaction against any one master. This, of course, is not always a valid assumption. If we did have a problem with multiple transactions, another programming problem would be involved.

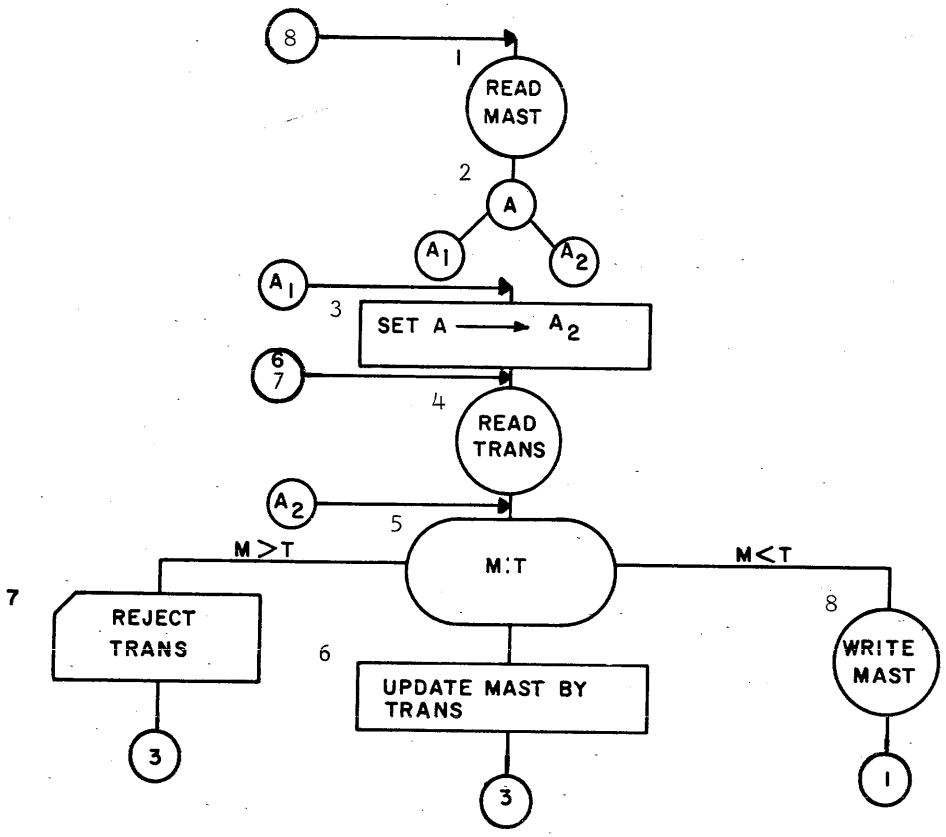
Initially we would have to bring in both a master and a transaction in order to start the processing. If the criteria of these two records matched, we would then update the master with the information contained in the transaction. At this point we would want to transfer back to read in another transaction, since we know that there may be multiple transactions against any one master. In addition, if the transaction record had a criteria that was less than the master criteria, we have been assuming that this was a sort error (or no master available) and have been stopping the computer. This is inefficient and a more common practice would be to put this error transaction out (for example, to a magnetic tape or to a punched card) and bring in a new transaction. If this was the entire program, we would have no problem, and the functional chart for it would appear as follows:



However, this does not take care of the case where the master criterion is less than the transaction criterion. This would indicate either that the master had no transactions against it and was simply to be written to the new master tape or that the master had one or more transactions against but these have been posted, and a transaction belonging to another master is now in memory. In either case, there is a good transaction in memory, and if we were to bring in another transaction over it we would, in effect, lose it. If we could consider just this portion of the program we could flow chart it as follows:



To have a complete program, however, we must be able to "overlay" the two flow charts. This we can accomplish by utilizing a "variable connector". This is nothing more than a program transfer instruction that is modified. In our example, we could simply place a Store Register instruction, affecting the P register, between the Read of the master and the Read of the transaction. Initially, the variable connector or "switch" will contain the address of the next instruction in sequence. This next instruction will modify the switch so that the next time it is executed it will cause the program to transfer to the compare of the criteria. From that point on, whenever we desire a master (but not a transaction) we will transfer to 1 (the Read of the Master) and the variable connector will skip us around the Read of the transaction to the comparison. Now we have the ability to bring in just a transaction, or just a master as the case may indicate. Functionally flow charted, our program would be as follows:



Example:

There is a master inventory file with 10,000 records. We have 8,000 issues sorted in ascending order by stock number to be posted against these masters. There may be multiple transactions against any one master. Both files are being maintained on magnetic tape.

File Descriptions:

Master:

Stock Number	Balance on Hand (BOH)
7	10

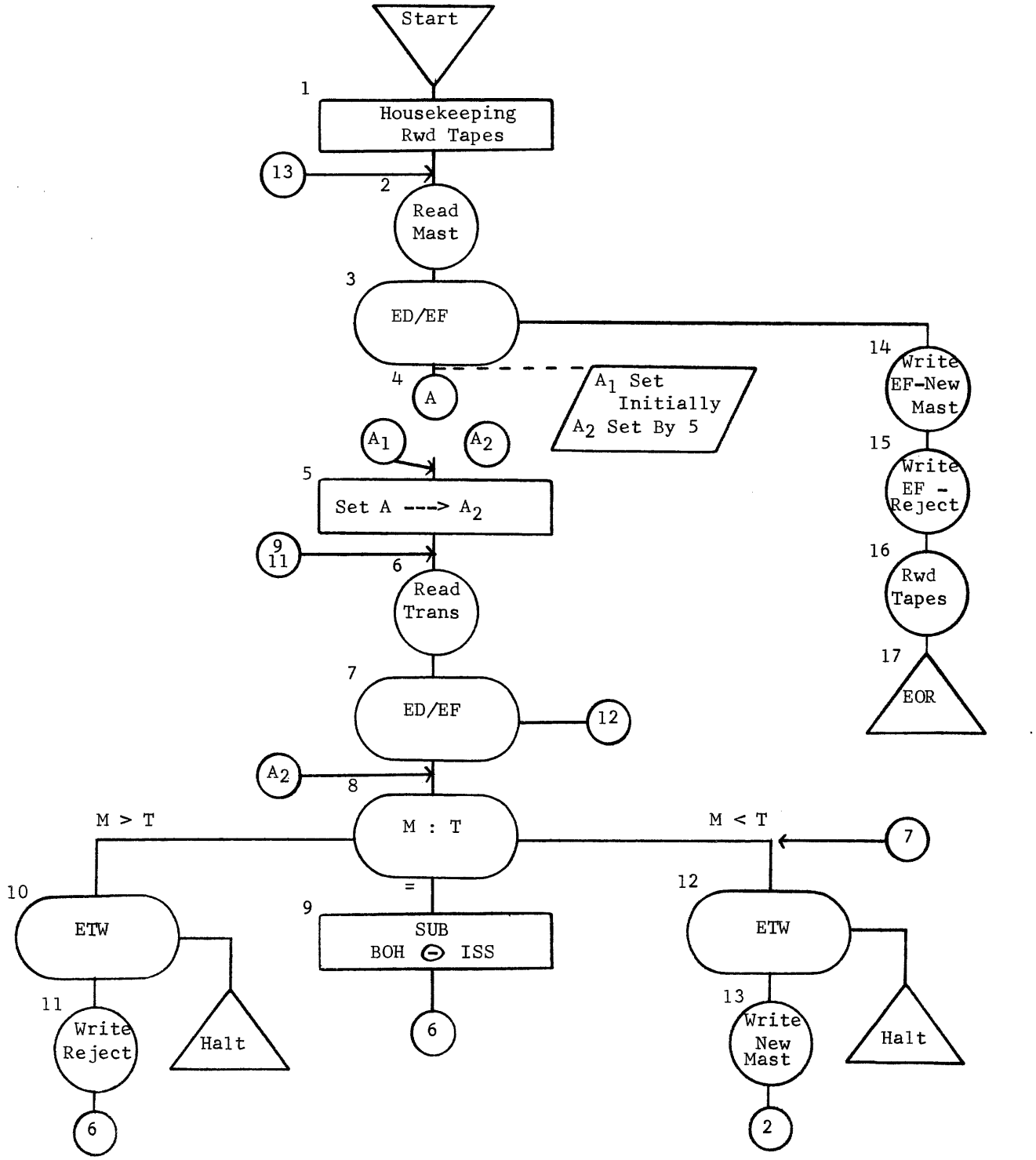
Transaction:

Stock Number	Issued Quantity
7	10

We are to write any out of sort transactions to an error tape.

Assume that the EF on the Master tape will occur at the same time as the EF on the Transaction tape.

Flow charted and coded the problem would appear as follows:



| | | |

TITLE VARIABLE CONNECTOR EXAMPLE I

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	520 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1 MAST	1
		1 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2 TRANS	1
		2 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3 NEW MAST	1
		3 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4 REJECT	1
5390		4 0	0	0	4	1	5	0	0	0	5	0	1	6		READ MAST	2
		5 0	0	0	W	8	5	4	0	0	5	2	6	0		ED/EF	3
5250	6	526 0	0	0	V	1	0	2	1	9	(5	2	7	0)	4	A1 → 4 A2 → 7	4
		7 0	0	0	N	4	5	4	8	5	5	2	6	9		SET A → A2	5
5360 5330		8 0	0	0	4	2	5	1	0	0	5	1	1	6		READ TRANS	6
		9 0	0	0	W	8	5	3	7	0	5	3	0	0		ED/EF	7
5260		530 0	0	0	Y	7	5	0	0	0	5	1	0	0		COMPARE MASTER STK #: TRANS STK #	8
		1 0	0	0	W	1	5	3	4	0	5	3	7	0		IF M > T → 10 M < T → 13	8
	6	532 0	0	0	⊖	&	5	0	1	6	5	1	1	6		SUB B.O.H. ⊖ ISSUE QTY	9
		3 0	0	0	V	1	0	2	1	9	5	2	8	0		TRANS → 5	
5310		4 0	0	0	S	4	4	0	0	0	5	4	7	0		ETW ON 4	10
		5 0	0	0	8	4	5	1	0	0	5	1	1	6		WRITE REJECT	11
		6 0	0	0	V	1	0	2	1	9	5	2	8	0		TRANS → 5	
5310 5290		7 0	0	0	S	3	4	0	0	0	5	4	7	0		ETW ON 3	12
	6	538 0	0	0	8	3	5	0	0	0	5	0	1	6		WRITE NEW MAST	13
		9 0	0	0	V	1	0	2	1	9	5	2	4	0		TRANS → 2	
5250		540 0	0	0	8	3	5	4	8	9	5	4	8	9		WRITE EF → NEW MAST	14
		1 0	0	0	8	4	5	4	8	9	5	4	8	9		WRITE EF → REJECT	15
		2 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1	16
		3 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2	16

XIX-9

IE 1224 REV. 8-60

TITLE VRRIABLE CONNECTOR EXAMPLE I

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	5	544 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	16
		5 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	16
		6 0	0	0	⊡	0	0	0	0	0	0	0	0	0		EOR	17
5340		7 0	0	0	⊡	0	0	0	0	0	0	0	0	0		ERROR HALT	20
		8 0	0	0	⊡	0	5	3	0	0	0	0	0	E/F		CONSTANTS A_2 EF	
		9 0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

As you can see, setting a switch implies transferring the address of the instruction we want to transfer to into the program transfer instruction. In our example, we utilized a Store Register instruction as our variable connector. By initially having the Store Register transfer to 5270, we will be able to read in both a master and a transaction. The instruction at 5270, however, places 5300 in the B address of the Store Register, and from that point on when we read a master we will skip over the read of the transaction to the compare located at 5300.

We could have saved an additional instruction by using the B address of the CTC executed as an EF/ED test. The B address of this instruction tells the computer where to go if the ED/EF has not been sensed. By initially having this address the Transfer Data Right instruction and then modify it to address the Sector Compare Left instruction we would get the same effect that is obtained in the program that we coded.

The example that we just investigated incorporates a "one way" switch. That is to say, we changed its setting from A1 to A2 and never affected it again. Many programs incorporate variable connectors that are changing two (or more) way connectors. For example, suppose we had a payroll problem to program. Against each master there can be no more than one transaction, however there may be masters without transactions and there may be transactions without masters. We will assume that all of these latter cases indicate new employees for which master must be set up. Examining the problem, there are three possible requirements:

1) we have updated a master and need both a new transaction and new master or

2) we have a master without a transaction and need just a new master, as there is a good transaction in memory or

3) we have a transaction without a master and need a new transaction as there is a good master in memory at this point.

By placing a variable connector between the two reads we give ourselves the necessary flexibility.

Since we don't know what path we have executed at any one time, we must make sure that the variable connector is in the proper condition. For example, if we have just updated a master we will want the switch to allow us to read in both a transaction and a master. For this reason we must make sure the switch is set to A1. It might already be set to A1, which means that we are wasting an instruction, but we are safeguarding our program in case it is not. If we have just set up a new master (the transaction did not have a master) we will want to read in another transaction but not a new master (since we have a good one in memory). To do this we will set switch A to A2, which skips us over the read of the master. If we have just written out a master then did not have a transaction, we simply want a new master and since we will transfer the program back to step 2, we are not interested in the condition of the switch. Completely flow charted and coded (ignoring ED/EF, and ETW routines) the problem appears as follows:

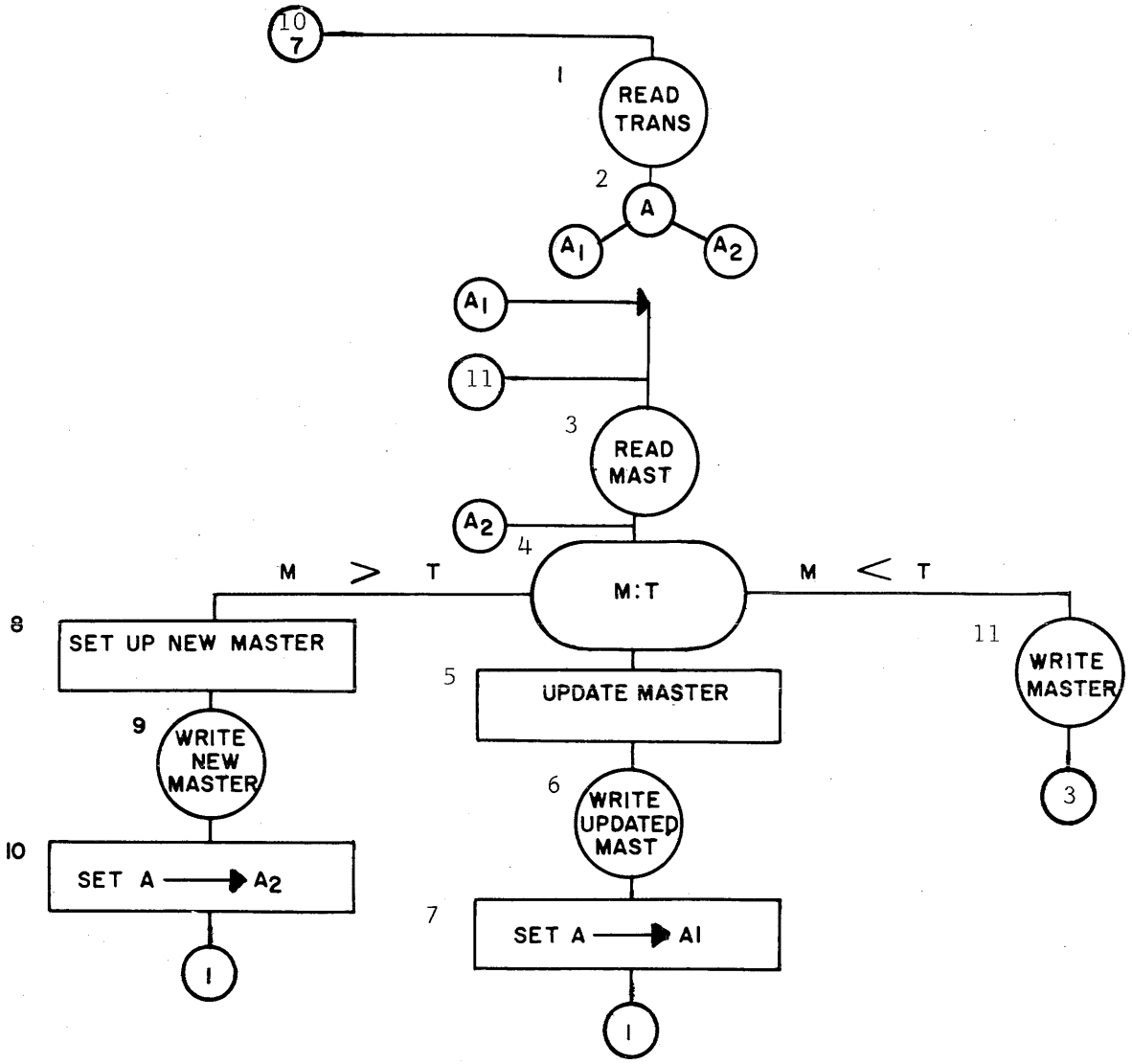
Data Description:

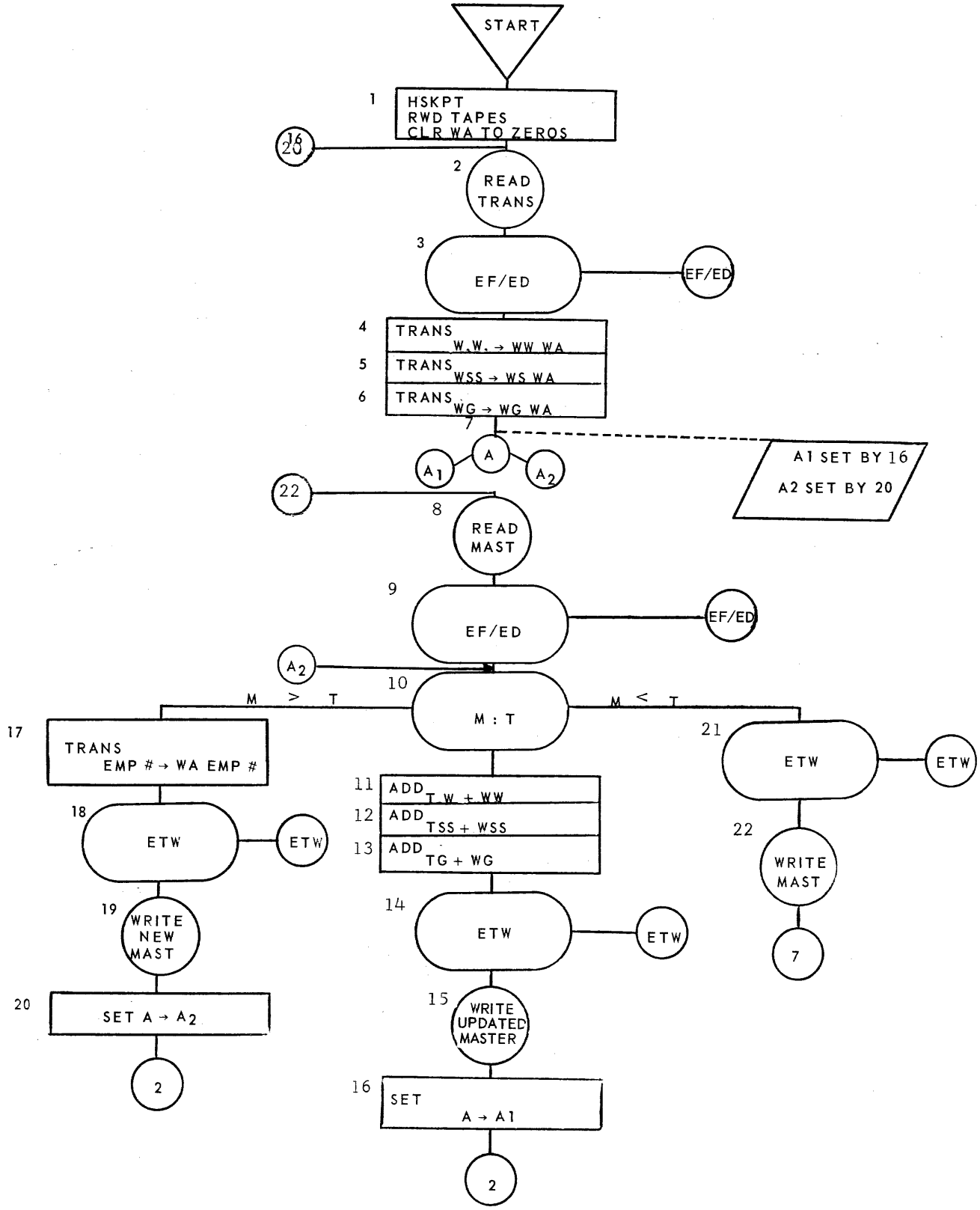
Master

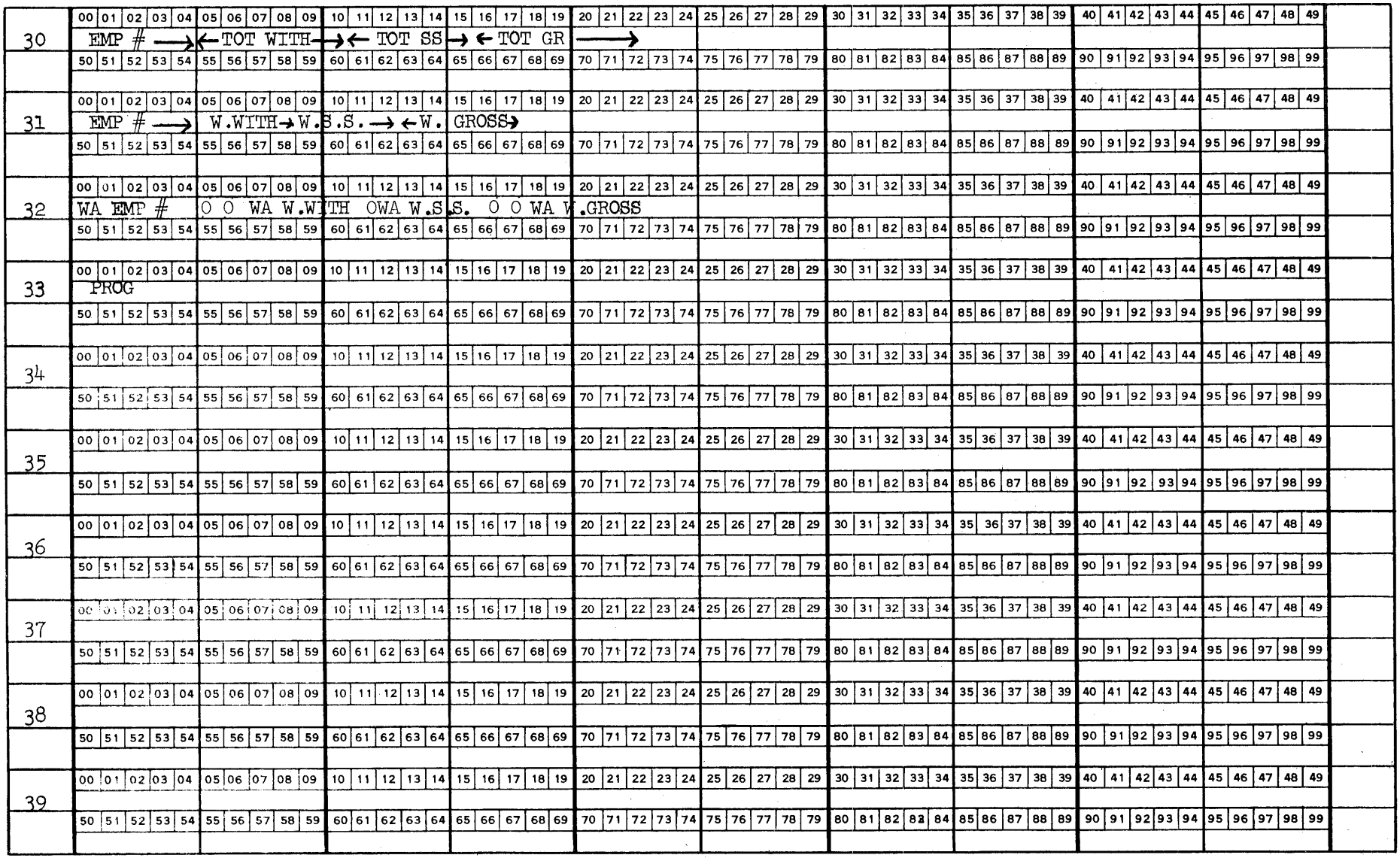
Employee Number	Total Withholding	Total S.S.	Total Gross
5	6	5	7

Transaction

Employee Number	Weekly Withholding	Weekly S.S.	Weekly Gross
5	4	4	5


10
7
1
READ TRANS
2
A
A1
A2
A1
11
3
READ MAST
A2
4
M > T
M:T
M < T
8
SET UP NEW MASTER
9
WRITE NEW MASTER
10
SET A → A2
1
5
UPDATE MASTER
6
WRITE UPDATED MAST
7
SET A → A1
1
11
WRITE MASTER
3


START
1
HSKPT
RWD TAPES
CLR WA TO ZEROS
20
16
2
READ
TRANS
3
EF/ED
EF/ED
4
TRANS
W.W. → WW WA
5
TRANS
WSS → WS WA
6
TRANS
WG → WG WA
7
A
A1
A2
A1 SET BY 16
A2 SET BY 20
22
8
READ
MAST
9
EF/ED
EF/ED
A2
10
M : T
M > T
M < T
17
TRANS
EMP # → WA EMP #
18
ETW
ETW
19
WRITE
NEW
MAST
20
SET A → A2
2
11
ADD
T W + WW
12
ADD
TSS + WSS
13
ADD
TG + WG
14
ETW
ETW
15
WRITE
UPDATED
MASTER
16
SET
A → A1
2
21
ETW
ETW
22
WRITE
MAST
7



TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1120 - 20M 3-66

TITLE VARIABLE CONNECTOR EXAMPLE 2

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	330 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD MAST	1
		1 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD TRANS	1
		2 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD NEW MAST	1
		3 0	0	0	J	0	3	2	0	0	3	2	2	2		FILL WORK AREAS	1
3550 3500		4 0	0	0	4	2	3	1	0	0	3	1	1	7		READ TRANS	2
		5 0	0	0	W	8	E	F	E	D	3	3	6	0		ED/EF	3
	6	336 0	0	0	M	4	3	1	0	5	3	2	0	7		TRANS WEEKLY WITH → WA	4
		7 0	0	0	M	4	3	1	0	9	3	2	1	2		TRANS WEEKLY S.S. → WA	5
		8 0	0	0	M	5	3	1	1	3	3	2	1	8		TRANS WEEKLY GR → WA	6
		9 0	0	0	V	1	0	2	1	9	(3	4	0	0)	15,20	SWITCH A A1 —> 7 A2 —> 9	7
3580 3390		340 0	0	0	4	1	3	0	0	0	3	0	2	2		READ MAST	8
		1 0	0	0	W	1	E	F	E	D	3	4	2	0		EF/ED	9
3390	6	342 0	0	0	Y	5	3	0	0	0	3	1	0	0		COMPARE EMP #	10
		3 0	0	0	W	1	3	5	1	0	3	5	6	0		M > T → 17 M < T → 22	10
		4 0	0	0	+	6	3	0	1	0	3	2	1	0		ADD TOT WITH + W. WTTH	11
		5 0	0	0	+	5	3	0	1	5	3	2	1	5		ADD TOT S.S. + WEEKLY S.S.	12
		6 0	0	0	+	7	3	0	2	2	3	2	2	2		ADD TOT GROSS + W. GROSS	13
		7 0	0	0	S	3	4	0	0	0	E	T	W			ETW	14
	6	348 0	0	0	8	3	3	0	0	0	3	0	2	2		WRITE UPDATED MAST	15
		9 0	0	0	N	4	3	5	9	5	3	3	9	9		SET A → A1	16
		350 0	0	0	V	1	0	2	1	9	3	3	4	0		TRANS → 2	
3430		1 0	0	0	M	5	3	1	0	0	3	2	0	0		TRANS EMP # → WA	17
		2 0	0	0	S	3	4	0	0	0	E	T	W			ETW	18
		3 0	0	0	8	3	3	2	0	0	3	2	2	2		WRITE NEW MASTER	19

TITLE VARIABLE CONNECTOR EXAMPLE 2

CODER

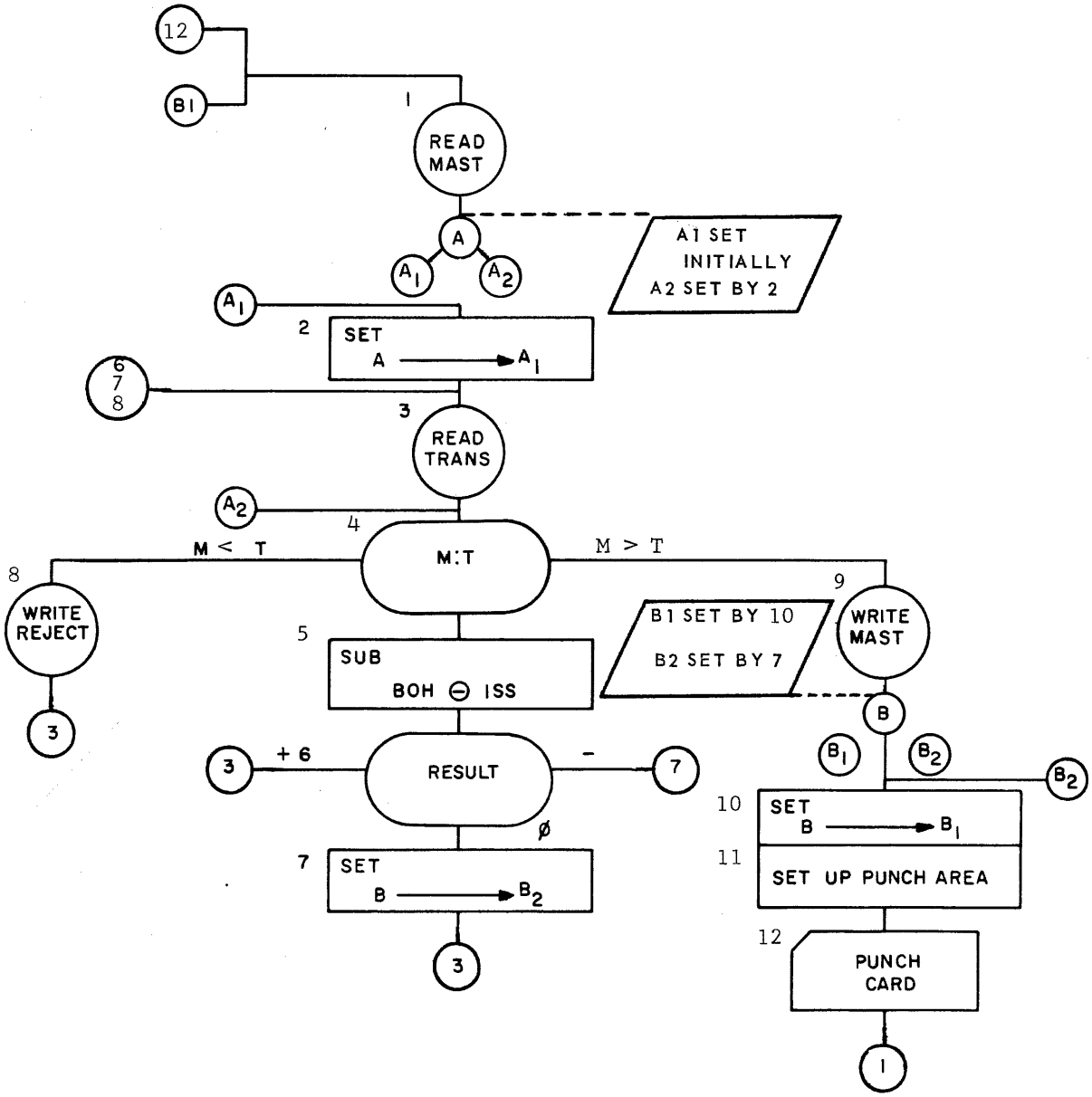
REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	354 0	0	0	N	4	3	5	9	9	3	3	9	9		SET A → A2	20
		5 0	0	0	V	1	0	2	1	9	3	3	4	0		TRANS → 2	
3430		6 0	0	0	S	3	4	0	0	0		ETW				ETW	21
		7 0	0	0	8	3	3	0	0	0	3	0	2	2		WRITE MASTER	22
		8 0	0	0	V	1	0	2	1	9	3	4	0	0		TRANS → 7	
		9 0	0	0	–	–	3	4	0	0	3	4	2	0		CONSTANTS A_1 A_2	
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

Another example can be developed using our original inventory problem, by adding another step to punch out a card if the BOH of a particular stock number reaches zero or goes negative. We could determine this condition easily by sensing the PRI's after subtracting the issues. If the balance is zero or negative the PRZ or PRN will be set. Upon sensing this we could affect another variable connector, this time the transfer that normally would take us back to read in a new master record. We could change this so that it takes us to a punch routine and then back to the read of the master. Since the negative result pertains to only this one record, however, it is advisable to set the switch back to its initial setting (B1) immediately upon entering the B2 path. This will prevent us from punching negative cards for stock accounts which are still positive. Functionally flow charted, the program would look as follows:



It is possible to form a switch using only 1 character and two instructions, if the Skip option of the Store Register instruction or the Conditional Transfer of Control instruction is used. If an N or 0 appears in either of these instructions, it will cause the instruction to be staticized but not executed. This means that the P register will not be effected.

Suppose, for example, we have a switch which is to transfer to 3190 if set to A1, and to 4850 if set to A2. We could code the switch as follows:

```
3500  V  (0)  0219  3190
3510  V   1   0219  4850
```

Now, if the switch is to be set to A1, we must place a "1" in the N character of the Store P Register instruction at 3500. This will cause the program to jump to 3190. If the switch is to be set to A2, however, we must place a "0" in the N character of that instruction. Then, when the instruction is staticized, it will not be executed and the transfer instruction at 3510 will be staticized in sequence and executed, causing the program to jump to 4850.

Variable Connector Exercise I:

A bank has 10,000 demand deposit (checking) accounts against which 8,000 transactions are to be posted. There may be multiple transactions on any one master. There are two types of transactions, withdrawals (checks) and deposits. The type of transaction is determined by examining the transaction code which will be either a W (withdrawal) or a D (deposit). There are approximately 1,000 deposits and 7,000 withdrawals. The transactions have been sorted in ascending order by account number and then by transaction code.

Data Description:

Master: (1 reel, tape unit 5)

Account Number	$ amount of Total Deposits to Date	$ amount of Total Withdrawals to Date	Balance
7	7	7	6

Transaction: (1 reel, tape unit 3)

Account Number	Transaction Code	Amount
7	1	7 (at least 1 insignificant zero in every amount)

Assumptions:

1. Any out of sort transactions are to be preceeded with the code letter "A" and placed on an error tape, unit 1.

2. Any transaction with an erroneous transaction code are to be preceeded with the code letter "B" and placed on the same error tape.

3. The new master file is to be constructed on tape unit 4.

4. Do not chart or code ED/EF or ETW routines, simply indicate a transfer to them.

Variable Connector Exercise II:

Using the same banking problem stated in variable Connector Exercise I, allow for the punching of an overdraft card if the balance goes negative. The overdraft card is to be in the format:

Col 1-7	59-65	67-73	Col 75-80
Acct. #	Total Deposits	Total With-	Balance
Flow chart only.		drawals	

Review Exercise III:

The purpose of this problem is to obtain a total of salaries to date by:

1) section number

2) department number

The payroll file is maintained so that the employees of a particular section are together and all the sections of a given department are together. Since the employee number consists of 10 digits:

XX	XXX	XXXXX
department number	section number	employee number

the entire file is in ascending order by employee number. Starting with the first record in the file, accumulate the total gross by section and by department. When the section number changes print out the total on the on-line printer. Continue accumulating for a department total, but also start a new section accumulation. Print out the total for every section and every department in this fashion. Page change after every department total. Do not print insignificant zeros.

Data Description:

Master (1 reel, tape A)

Employee Number	Total Withholding	Total Social Security	Total Gross
10	6	5	7

Print Layout:

Department Number (print positions 1-2)
Section Number (print positions 3-5) total (print positions 10-18)

Department Number Dept. total (print position 20-28)

Example:

```
12
  234     345263422
  453     098765435
12                  444028857

23
  654     002746378
  987     098745324
  989     286543647
```

Assumptions:

1. Assume that the file is in correct order, halt on any out of sort.
2. Assume no department will take more than one page.
3. Simply transfer to ED/EF and ETW routines, but do not chart or code them.
4. Assume maximum accumulation size of 9 digits.

Flow chart in Notebook

XX – LOGICAL OPERATIONS

(LOGICAL AND, LOGICAL OR, EXCLUSIVE OR)

Most program steps require that the computer work on the information bits of the character as a whole, however there are some cases that require that we operate on each individual bit as a separate piece of information.

For example, we already know that a minus sign appears in the 2^5 position of the least significant digit of an amount. Suppose we are interested in printing out a list of all the customers that have over paid their charge accounts. We are carrying this as a negative amount (balance due = total purchases - total payments). Our first step would be to examine 2^5 position of the least significant digit. If this contains a 1 bit, the amount is negative; if it contains a 0 bit, the amount is positive. To examine only 1 bit of an entire character, it is necessary to "mask" out the other 5 bits. This is possible with a LOGICAL AND instruction.

LOGICAL AND;

This instruction enables us to extract or "mask" out individual bits from a character (or characters). The rule for this instruction is that you must have a 1 in the bit position of the modifier, in order to get a 1 (a one and a one will give you a one). That is to say:

1 and 1 gives 1
1 and 0 gives 0
0 and 0 gives 0

Example:	110101	011011	- section being modified
	011011	110101	- modifier (mask)
	010001	010001	- result

For our particular problem, we would move the least significant digit of the amount to a work area, since this instruction destroys the character or characters being modified. We would then perform a Logical And using $(100000)_2$ (minus sign) as the mask. This will cause all the bit positions from 2_4-2_0 to become zeros. If the bit in 2_5 is a 0, it will remain a zero; and the resulting character will be $(000000)_2$; if it is a 1, the resulting character will be $(100000)_2$. In addition to modifying the character, the instruction will affect the PRI's. If the resulting field is all zeros (in our case indicating a positive quantity) the PRN is set; if the result contains one or more 1 bits (which shows a negative balance in our example) the PRP is set.

The OP code of this instruction is T.

The N character indicates the number of characters in each operand (0-44).

The A address gives the location of the least significant digit of the field to be modified (and the result).

The B address gives the location of the least significant digit of the modifier.

Having determined whether the balance is negative or positive, we are now capable of taking the appropriate steps. If the result was positive, the record is of no interest to us. If it was negative, we must now set up a print area. Let's assume that the only information contained in the master is:

Customer Number	-	Name	-	Balance
6		25		6

a space being carried between each item. We will also assume that we do not care if the print out contains insignificant zeros, as it is only for internal use. We have only to read this into an even-hundreds location (a legitimate print area) and we will be able to print, except for one step.

In order to get into this path, the balance had to be negative. This means that the least significant digit of the balance has, in addition to its numeric bits, a zone bit. If we printed this character as it stands, we would obtain the following:

if the numeric portion is:	the printed character will be:
0	⊖
1	J
2	K
3	L
4	M
5	N
6	O
7	P
8	Q
9	R

In order to avoid this, we must get rid of the zone bit. This could be done by again using the Logical And instruction with a mask of $(001111)_2$ or $(011111)_2$. The only problem is that none of the RCA 301 characters have these binary configurations. We could construct the necessary character during the housekeeping portion of our program, but it is just as easy to use the EXCLUSIVE OR instruction.

EXCLUSIVE OR:

This instruction also works in individual bits. The rule that is applied is that opposites are required in order to obtain a one bit:

0 and 0 will yield 0
1 and 1 will yield 0
0 and 1 will yield 1

In our example, we will want all the information between 2_4 and 2_0 to come through untouched. To do this we will need 0's. However, the 2_5 bit, which we know to be a 1, must be changed to a zero which we can do by overlaying it with a 1 (two ones will yield a zero). Our modifying character will be 100000, again a minus sign.

The OP code of this instruction is a U.

The N character again indicates the number of characters in each operand (0-44).

The A address gives the HSM location of the least significant digit of the sector to be modified and subsequently the result.

The B address gives the HSM location of the least significant digit of the modifier.

This instruction does not affect the PRI's.

Having thus set up the print area we have only to execute the Print and Paper Advance instruction.

Suppose that in addition to the printed list of credit balances, we must also punch out cards in the format:

Customer Number	Amount
1 - 6	75-80

In addition to simply transferring the desired information in order to set up the punch area, we must also replace the minus bit. This we can do with a LOGICAL OR instruction.

LOGICAL OR:

This instruction allows us to insert bits into a character(s). The rule is a one bit in one position or the other or both will yield a one:

0 and 0 yields 0
1 and 0 yields 1
1 and 1 yields 1

In our case we wish to insert a 1 in the 2 position, and not affect the remainder of the character. Again our modifying character will be a minus, $(100000)_2$.

The OP code is a Q.

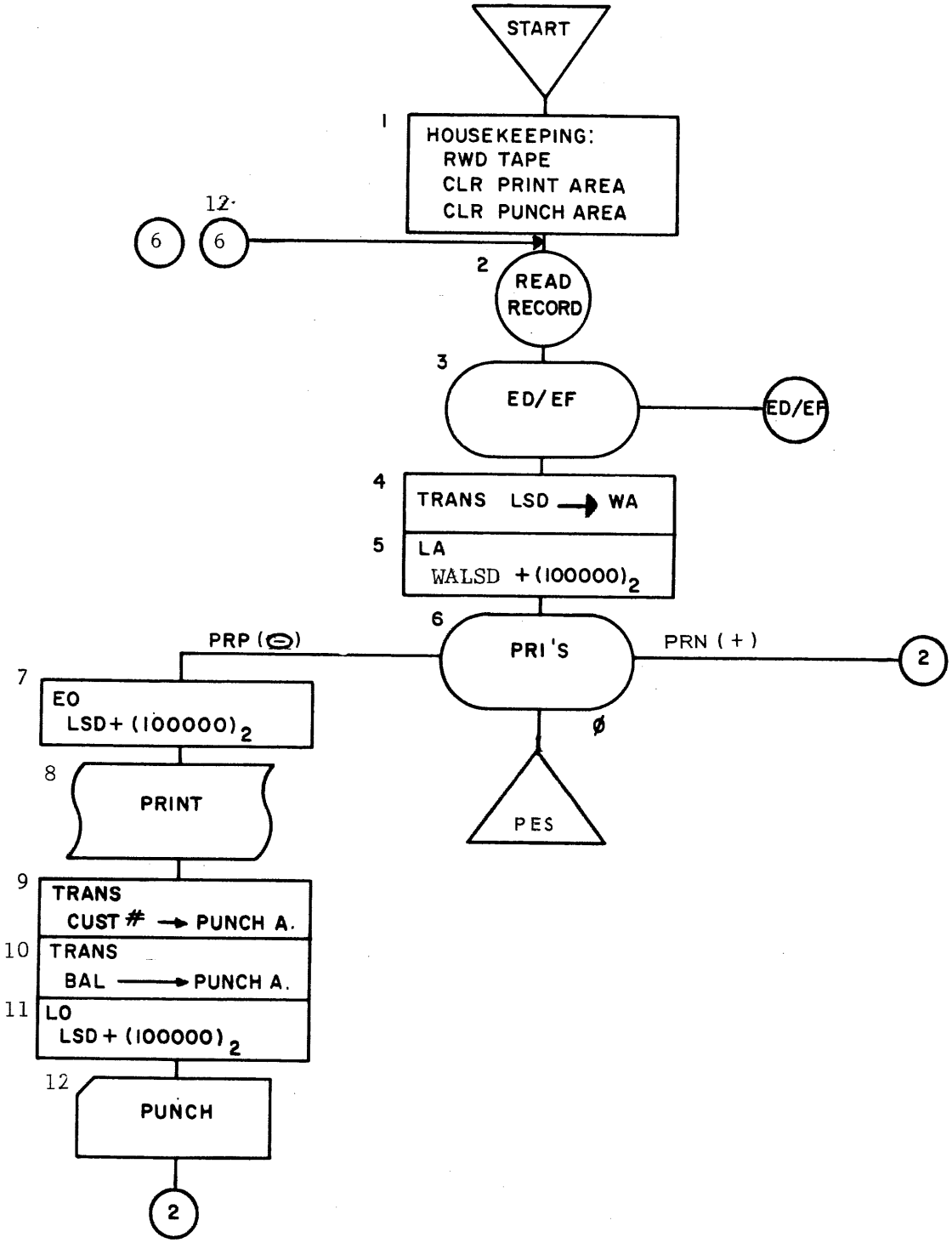
The N gives the number of characters in each operand (0-44).

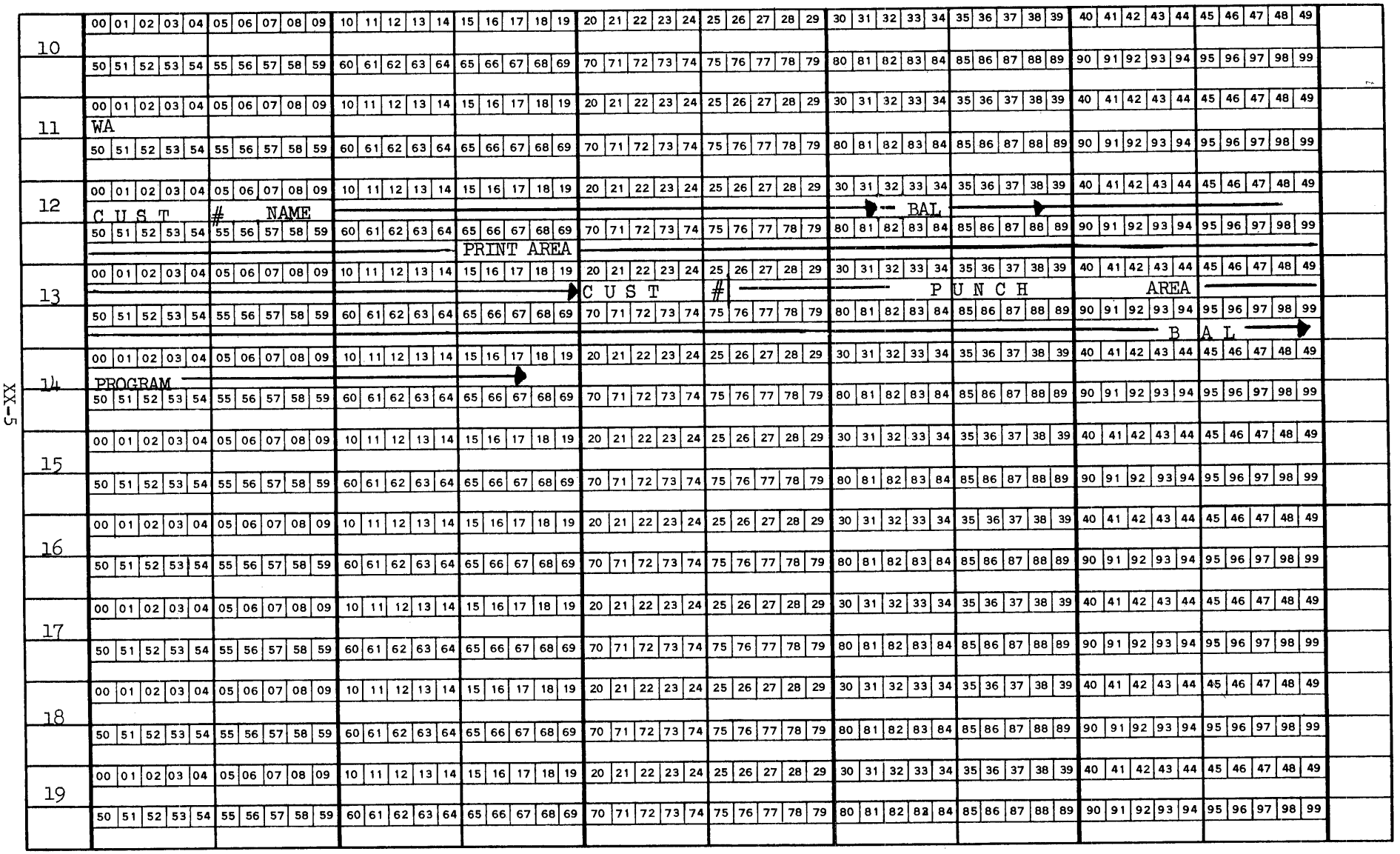
The A address gives the location of the least significant digit of the operand to be modified and the result.

The B address gives the location of the least significant digit of the modifying operand.

This instruction does not affect the PRI's.

Flow charting and coding our entire program, it would appear as follows:


START
1
HOUSEKEEPING:
RWD TAPE
CLR PRINT AREA
CLR PUNCH AREA
12
6
6
2
READ
RECORD
3
ED/EF
ED/EF
4
TRANS LSD → WA
5
LA
WALSD +(100000)2
6
PRI'S
PRP (⊖)
PRN (+)
2
Ø
PES
7
EO
LSD+(100000)2
8
PRINT
9
TRANS
CUST # → PUNCH A.
10
TRANS
BAL → PUNCH A.
11
LO
LSD+(100000)2
12
PUNCH
2



TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ____ OF ____

FORM NO. 1223 25M 6-60

TITLE Logicial Instructions Example

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	140 0	0	0	;	3	0	0	0	0	0	0	0	0		REWIND INPUT	1
		1	0	0	J	−	1	2	0	0	1	3	9	9		CLR PRINT AND PUNCH AREAS	1
1460 1540		2 0	0	0	4	3	1	2	0	0	1	2	3	8		READ RECORD	2
		3 0	0	0	W	8	E	F	E	D	1	4	4	0		EF/ED	3
		4 0	0	0	N	1	1	2	3	8	1	1	0	0		TRANS LSD → WA	4
		5 0	0	0	T	1	1	1	0	0	1	5	5	1		LA TO DETERMINE SIGN	5
	6	146 0	0	0	W	1	1	4	8	0	1	4	2	0		SENSE PRI's PRP → 7 PRN → 2	6
		7 0	0	0	⊡	0	0	0	0	0	0	0	0	0		PES	
1460		8 0	0	0	U	1	1	2	3	8	1	5	5	1		EO TO CHANGE ⊖ TOT	7
		9 0	0	0	B	0	0	0	0	0	1	2	0	2		PRINT AND VT	8
		150 0	0	0	M	6	1	2	0	0	1	3	2	0		TRANS CUST # → PUNCH AREA	9
		1 0	0	0	N	6	1	2	3	8	1	3	9	9		TRANS BAL → PUNCH AREA	10
	6	152 0	0	0	Q	1	1	3	9	9	1	5	5	1		LO TO INSERT ⊖	11
		3 0	0	0	2	0	1	3	2	0	1	3	9	9		PUNCH CARD	12
		4 0	0	0	V	1	0	2	1	9	1	4	2	0		TRANS → 2	
		5 0	0	0	⊡⊖		0	0	0	0	0	0	0	0		MINUS	
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XX-6

IE 1224 REV. 8-60

Logical Operations Exercise:

A. Show the result at the end of the following operations:

1. $(100110)_2$ modified by $(000111)_2$ using a Logical And

2. $(111000)_2$ $(111111)_2$ modified by $(000101)_2$ $(010011)_2$ using a Logical Or.

3. $(101010)_2$ modified by $(110011)_2$ using an Exclusive Or.

B. What instruction(s) would you use to determine if the 2_3 bit of the character located at 1687 is a one or a zero? What would be the modifying character?

C. 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?

D. Suppose you wanted to insert a one bit into the 2^4 position of the character located at 1564. What would be the instruction that you would use and what would be the modifying character?

E. If you wanted to extract a one bit if present from the 2^2 position of the character located at 1743, what would be the instruction you would use? What would be the modifying character?

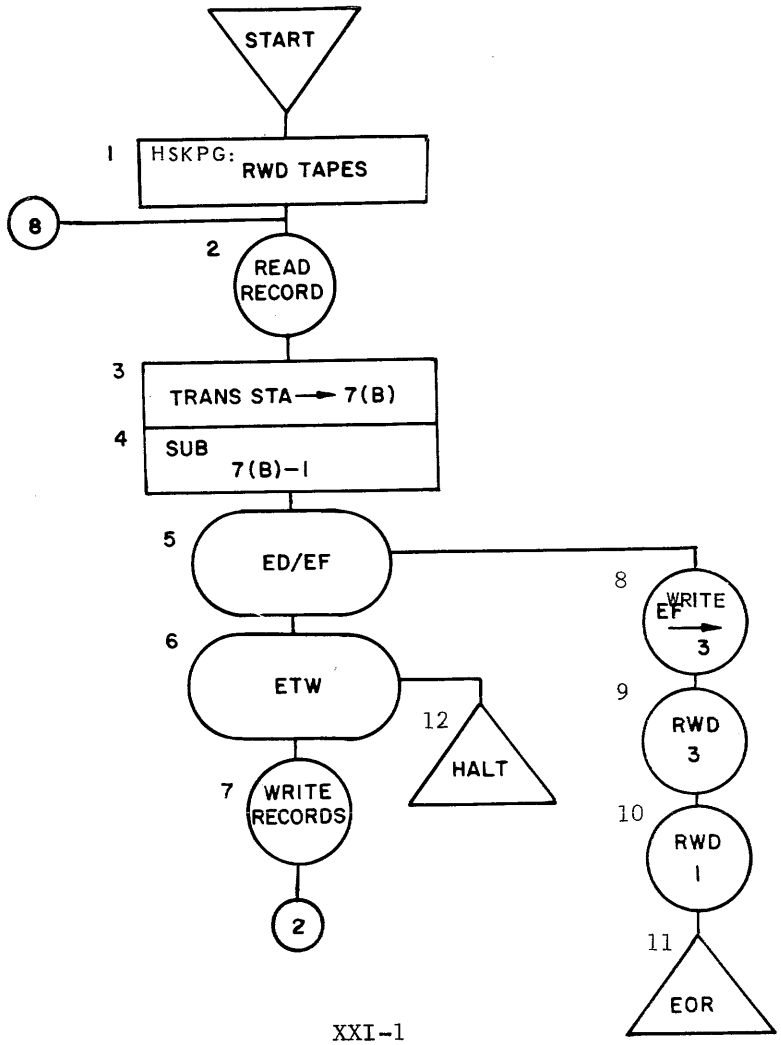
XXI – HANDLING VARIABLE DATA

(TRANSFER DATA BY SYMBOL LEFT, TRANSFER DATA BY SYMBOL RIGHT, LOCATE SYMBOL RIGHT, STORE B)

To date, we have been working with fixed variable records. Initially, however, we pointed out that the RCA 301 was geared to handle both fixed-variable and variable data. We are now concerned with the programming involved in handling the variable data.

Taking the simplest example, suppose we wished to duplicate a one reel file containing variable length messages (the maximum message length being 100 characters). The instruction flow would be basically the same, but there is one important difference. When we are ready to write out the record, we must be able to give both the left and right hand ends of the sector to be written out. We know the left hand end, but what about the right hand end? Since we are reading in variable length data, the rightmost address will vary. It will be necessary, therefore, to modify the B address of the write. The question is, how?

When a Tape Read Forward Normal Instruction is executed, the A register is keeping track of the destination location of the characters. At the end of the operation (when the gap is sensed), the A register holds the address of the location one to the right of the last character placed in memory. This is automatically placed in the STA locations, where it is accessible by the programmer. Knowing the address that is one to the right of the last character, it is only a matter of subtracting one to get the address of the last character in the sector.



TITLE Handling Variable Data Example I
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	420 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1	1
		1 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	1
4280		2 0	0	0	4	1	2	7	0	0	2	7	9	9		READ RECORD	2
		3 0	0	0	N	4	0	2	1	5	4	2	7	9		TRANS STA → 7B	3
		4 0	0	0	-	3	4	2	7	9	4	3	4	9		SUBTRACT 1 FROM 7 (B)	4
		5 0	0	0	W	8	4	2	9	0	4	2	6	0		ED/EF SENSE	5
	6	426 0	0	0	S	3	4	0	0	0	4	3	3	0		ETW SENSE ON #	6
		7 0	0	0	8	3	2	7	0	0	(2	7	(	))	3	WRITE RECORD	7
		8 0	0	0	V	1	0	2	1	9	4	2	2	0		TRANS → 2	
4250		9 0	0	0	8	3	4	3	4	1	4	3	4	1		WRITE EF → 3	8
		430 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	9
		1 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1	10
	6	432 0	0	0	⌐	0	0	0	0	0	0	0	0	0		HALT EOR	11
4260		3 0	0	0	⌐	0	0	0	0	0	0	0	0	0		HALT ERROR	12
		4 0	0	0	⌐	E/F	0	0	0	0	0	0	0	1		CONSTANTS	
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXI-2

IE 1224 REV. 8-60

A second example would occur if we wished to print out a list of employee numbers and names from a file that had a variable format as follows:

EMP # *	NAME *	ST. ADD *	C.S. ADD *	TOTAL GROSS *	TOTAL ISS *	TOTAL WITH *	TOTAL NET
5	25	30	35	7	5	7	7
5	15	20	22	6	4	6	6
100	100	100	100	100	100	100	100

(The * between each item differentiates them.)

We could easily transfer the employee number into the print area but the problem would be the name. Since one name might be JOE_DOE, and the next one might be BENJAMIN_FRANKLIN, we have no way of knowing how many characters to transfer. To facilitate this, there is an instruction that will transfer characters up until the time it picks up and transfers a given symbol. In our case we would want to transfer from left to right starting with the left hand end of the name, the location of which we know, until we transferred the * which separates the name and the street address.

DATA TRANSFER BY SYMBOL LEFT

This instruction transfers data moving left to right from one sector of memory to another, until a selected symbol is sensed.

The OP code is a #.

"N" indicates the selected symbol on which to stop transferring.

The A address gives the location of the leftmost character to be transferred.

The B address gives the destination location of this first character to be transferred (leftmost location of the destination area.)

Assuming that we have read our records into memory at 5100, the portion of memory we know about appears as follows:

	00	01	02	03	04	05	06	07	08	09
51	X	X	X	X	X	*	NAME ———			>

Employee #

The last location that we know specifically is the left hand end of the name, 5106.

Assuming that the print area has been laid out to start at 1600 and has been cleared to blanks, we want to place the employee number in 1600-1604 and start the name in 1610. Knowing this, our instruction would appear as follows:

* 5106 1610

Suppose one record was 23456*JOE_DOE*123_ANY STREET etc.

The print area would appear as follows after the execution of the DL instruction to transfer the employee number and the DSL instruction shown above:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
16	2	3	4	5	6	_	_	_	_	_	J	O	E	_	D	O	E	*	_	_

If the record was 45487*JOHN_SMITH*3543 FIRST_STREET etc, the print area would be:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
16	4	5	4	8	7	_	_	_	_	_	J	O	H	N	_	S	M	I	T	H

20	21	22	23
*	_	_	_

The above records point out a programming necessity when handling variable data, and that is the fact that it will be necessary to clear the portion of the print area that would contain the name before the next name is transferred in, rather than just at the beginning of the program. In the two cases illustrated, JOHN_SMITH would simply overlay JOE_DOE, but suppose the next name was JIM_RAU. The print area would appear as follows, unless the name portion had been filled with spaces:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
16	6	7	5	3	5	_	_	_	_	_	J	I	M	_	R	A	U	*	T	H

20	21	22	23
*	_	_	_

One other point that presents itself is the fact that we end up with the * in the print area. What happens is that the character addressed by the A register is placed at the location addressed by the B register. Both the A and the B registers are increased by one. If the character transferred is identical to the character in the N register, the operation terminates, otherwise the process continues. This means that the terminating symbol (in our case an *) will end up in the destination area. However, since the B register has been keeping track of the destination locations, it will end up with the address one to the right of the destination *. A simple transfer of a blank by a Data Transfer instruction with the B address containing this modified address would solve our problem. The only problem is to obtain the contents of B.

STORE REGISTER:

We have already discussed the Store Register instruction in conjunction with the P register. We also have the ability to store the B register, by simply using an "N" of 4.

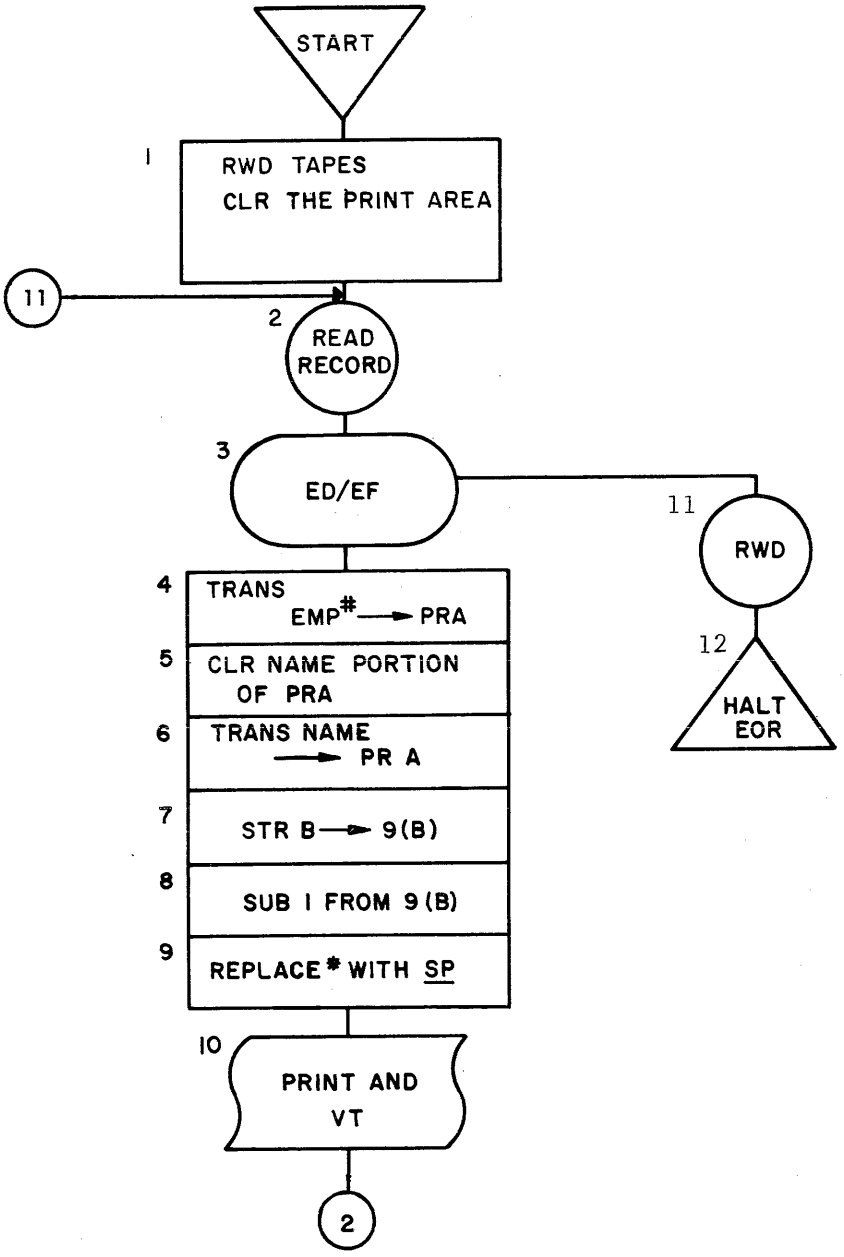
To summarize:

The OP code is V.

"N" would be 4, to refer to the B register.

The A address would be the right hand end of the four locations to receive the contents of the B register.

The B register is zeros (0000).


START
1
RWD TAPES
CLR THE PRINT AREA
11
2
READ
RECORD
3
ED/EF
11
RWD
12
HALT
EOR
4
TRANS
EMP# → PRA
5
CLR NAME PORTION
OF PRA
6
TRANS NAME
→ PR A
7
STR B → 9(B)
8
SUB 1 FROM 9(B)
9
REPLACE * WITH SP
10
PRINT AND
VT
2

TITLE Handling Variable Data Example II

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	410 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1	1
		1 0	0	0	J	-	1	6	0	0	1	7	1	9		CLR PRINT AREA	1
4210		2 0	0	0	4	1	5	1	0	0	5	1	9	9		READ RECORD	2
		3 0	0	0	W	8	4	2	2	0	4	1	4	0		SENSE ED/EF	3
4130		4 0	0	0	M	5	5	1	0	0	1	6	0	0		TRANS EMP# → PRINT AREA	4
		5 0	0	0	J	-	1	6	1	0	1	6	3	4		CLEAR PRINT AREA (NAME)	5
	6	416 0	0	0	#	*	5	1	0	6	1	6	1	0		TRANSFER NAME	6
		7 0	0	0	V	4	4	1	9	9	0	0	0	0		STR B IN 9 (B) - 1	7
		8 0	0	0	⊖	2	4	1	9	9	4	2	4	9		SUB 9 (B) -1	8
		9 0	0	0	N	1	4	2	4	1	(1	6	(	))	7,8	TRANS A BLANK OVER THE *	9
		420 0	0	0	B	0	0	0	0	0	1	6	0	2		PRINT AND VT	10
		1 0	0	0	V	1	0	2	1	9	4	1	2	0		TRANSFER → 2	
4130	6	422 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1	11
		3 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT EOR	12
		4 0	0	0	0	-	0	0	0	0	0	0	0	1		CONSTANTS	
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

9-IXX

IE 1224 REV. 8-60

We could eliminate one instruction by replacing the Transfer Data Right (9) with a Transfer Symbol to Fill instruction which would appear as follows:

J _ ( ) 1634

The A address would be the variable address of the *, obtained by storing B after the Transfer Data by Symbol Left instruction and subtracting one. The B address is the right hand end of the print area that is to receive the maximum name. In this way we not only replace the * with a space, but we also clear the name area, both in one instruction.

Suppose we are required to compress the information coming in from cards in order to put it out on to magnetic tape. The card format is:

Account Number	Name
1-7	9-35

We want to read in the card and write the information out to magnetic tape, first placing a # to differentiate between the two items. The problem is basically one of locating the last significant character in the name. We can do this by doing a Locate Symbol Right instruction.

LOCATE SYMBOL RIGHT:

This instruction allows the programmer to locate the lack of designated symbol.

The OP code is L.

"N" gives the symbol with which we are searching.

The A address gives the right hand end of the sector to be searched.

The B address gives the left hand end of the sector to be searched.

The operation will terminate if:

1) a non-symbol is found, in which case STA gives the address of the last symbol found.
2) the end of the sector is found.

In addition, the PRI's are affected as follows:

PRN is set when the first character searched is not equal to the contents of N.

PRZ is set when all characters searched are equal to the contents of N.

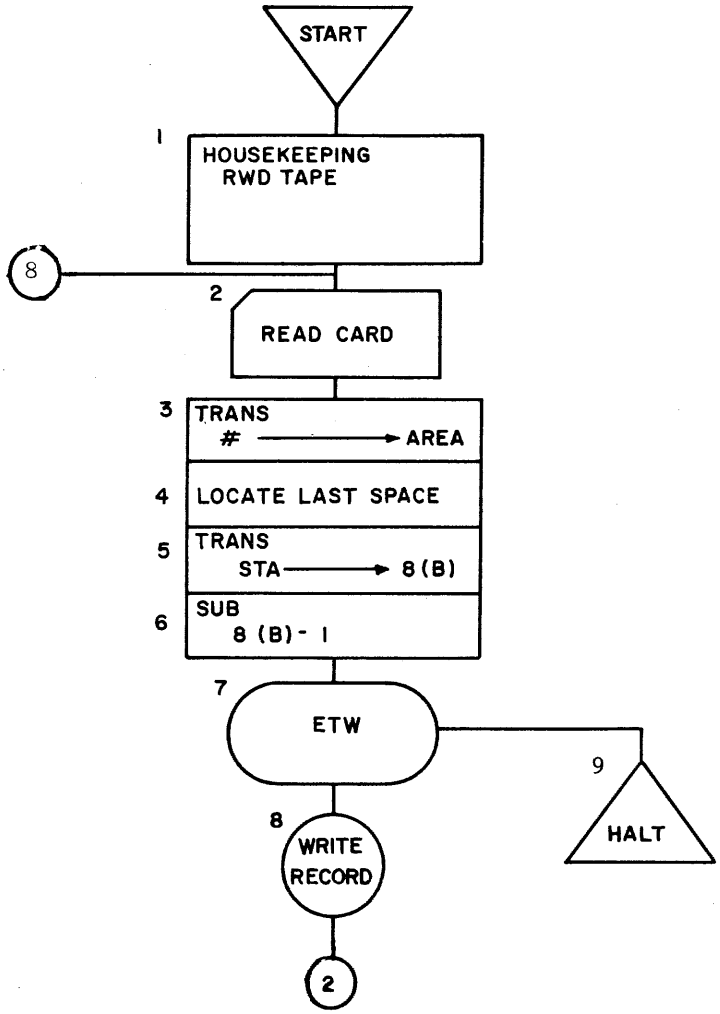
PRP is set if a non-symbol is found after a character equal to the contents of N has been found.

For our particular case, let's assign 6700-6779 as the read in area. The name portion will fall between 6708 and 6734. If our instruction reads:

L _ 6734 6708

when the instruction terminates, STA will hold the address that is one to the right of the last character in the name. By transferring that address and subtracting one from it we come up with the address of the last character. We will not sense the PRI's, as we are assuming that there will always be a name.

Flow charted and coded, our program would look as follows:



Testing out one record, we have:

1234543 JOE_DOE

Reading this into memory we have:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
67	1	2	3	4	5	4	3		J	O	E	_	D	O	E	_	_	_	_	_
	20	21	23	24	25	26	27	28	29	30	31	32	33	34						
	_	_	_	_	_	_	_	_	_	_	_	_	_	_						

Locating the last blank moving from right to left, we end up in STA with 6715. Transferring this to the B address of the write and subtracting 1, we have a write instruction which reads:

8 6 6700 6714

and which will write out just the information in the record.

TITLE Handling Variable Data Example III

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	600	0	0	;	6	0	0	0	0	0	0	0	0		RWD 6	1
6080		1	0	0	0	1	6	7	0	0	0	0	0	0		READ CARD	2
		2	0	0	N	1	6	1	0	1	6	7	0	7		TRANS # → POSITION BETWEEN ACCT # AND NAME	3
		3	0	0	L	_	6	7	3	4	6	7	0	8		LOCATE LAST BLANK MOVING R→L	4
		4	0	0	N	4	0	2	1	5	6	0	7	9		TRANS STA→8 (B)	5
		5	0	0	⊖	2	6	0	7	9	6	1	0	9		SUB 8 (R) - 1	6
	6	606	0	0	S	6	4	0	0	0	6	0	9	0		ETW?	7
		7	0	0	8	6	6	7	0	0	(0	0	0	0)	5,6	WRITE RECORD	8
		8	0	0	V	1	0	2	1	9	6	0	1	0		TRANS→2	
6060		9	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT	9
		610	0	0	⊡	#	0	0	0	0	0	0	0	1		CONSTANTS	
			0	0													
	6		0	0													
			0	0													
			0	0													
			0	0													
			0	0													
			0	0													
	6		0	0													
			0	0													
			0	0													
			0	0													
			0	0													
			0	0													

6-IXX-9

Another problem which might present itself is to be able to find the left hand end of a variable item for comparison purposes. Suppose we had a file that is constructed as follows:

	Stock #	'Description'	Order Date'	Balance	(' acts as a separator)
Max	5	40	6	10	
Avg	5	25	6	8	
%	100	100	100	100	

Our problem is to compare the date against today's date to see if we have to put out an order. If an order is not due we must bring in a new record; if it is, we must prepare a print-out. The portion of the program that we are interested in is simply locating the order date and setting up the compare instruction. Due to the fact that the variable item "Description" preceeds the "Order Date", we must be able to search through it and locate the MSD of the date. To do this, we again utilize a Transfer Data by Symbol Left Instruction. If we start transferring with the first character of the description (one to the right of the ') and transfer in place, the operation will terminate when the ' between the description and the order date is recognized. At that point, STA (and the B register) will hold the address one to the right of the ', or the MSD of that date. By simply transferring this to the A address of a compare instruction, we have set up the compare.

Let's take a sample record to test out this theory:

76587'BOLTS'600930'1098765

If we read this into memory at 5000, we would have:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	
50	7	6	5	8	7	'	B	O	L	T	S	'	6	0	0	9	3	0	'	etc.

Executing the instruction:

5510 # ' 5006 5006

we will end up in STA with 5012. Transferring this to a compare instruction will give us a compare that appears as follows:

Y 6 (5012) 3560 (3560 is the LHE of today's date)

The three steps would look as follows:

```
5510  #  '  5006  5006
5520  N  4  0215  5535
5530  Y  6  (  )  3560
```

Having solved that problem, suppose we had to write a program which would add an interest amount to a balance. We will assume that there will be one transaction against every master.

The master information is in the format:

	Acct. #	/	Balance	(The / serves as a separation between the two items)
Max.	7		8	
Avg.	7		6	
%	100		100	

The transaction information is coming from cards in the format:

	Acct. #	Interest
Cols.	1-7	10-14

We are to check to make sure that the transaction and the master account numbers match, but if they do not, we will HALT on an error condition.

The problem is basically one of reading, comparing account numbers, adding and writing, but with one major difference, and that is that we are now working with variable data. When we add, we must have our information in fixed fields with the operands of equal length. To do this, we will have to move the transaction interest amount to an eight character work area, in order to obtain the necessary insignificant zeros. In addition, we must move the balance item to a work area, since it is of variable length. Once there, we will have to clear the unused portion of the work area (including the /) to zeros.

Once the addition has been accomplished, we will have to locate the most significant digit of the new balance (since we don't want to carry the unnecessary zeros in the master) and transfer this field back to the read in area. In order to stop the transfer, we must arrange to have a terminating character to the right of the work area. In our case we will use a /.

Having reconstructed the record, we can write out the record as long as we know the address of the last character.

Developing the program then, our first steps would be to read in the master record. Immediately, we must transfer the contents of STA to a storage location and subtract one from it, in order to obtain the address of the last character read in, which is the address of the LSD of the balance.

Having sensed for ED/EF, we can read in a transaction and compare account numbers. If they are unequal, we must Halt; if they are equal we can continue processing.

The transfer of the interest amount to a work area can easily be accomplished by a Data Transfer Right instruction, since we know that the interest will always be 5 characters, loaded with insignificant zeros if necessary. The transfer of the balance could not be accomplished as easily however, except for the fact that we stored the address of the right hand end of the balance immediately after our read instructions. If this address were placed in the A address of a Transfer Data by Symbol Right instruction, we would only have to execute the instruction to move the balance to a work area.

TRANSFER DATA BY SYMBOL RIGHT:

This instruction allows the computer to transfer data from one location to another, moving in a right to left direction, until a designated symbol is located and transferred.

The OP code is a P.

"N" gives the character that is to terminate the operation.

The A address gives the right hand address of the item to be transferred, and in our case, this is the address that must be modified for every record.

The B address gives the right hand destination location, for our example, the right hand end of a 9 character work area.

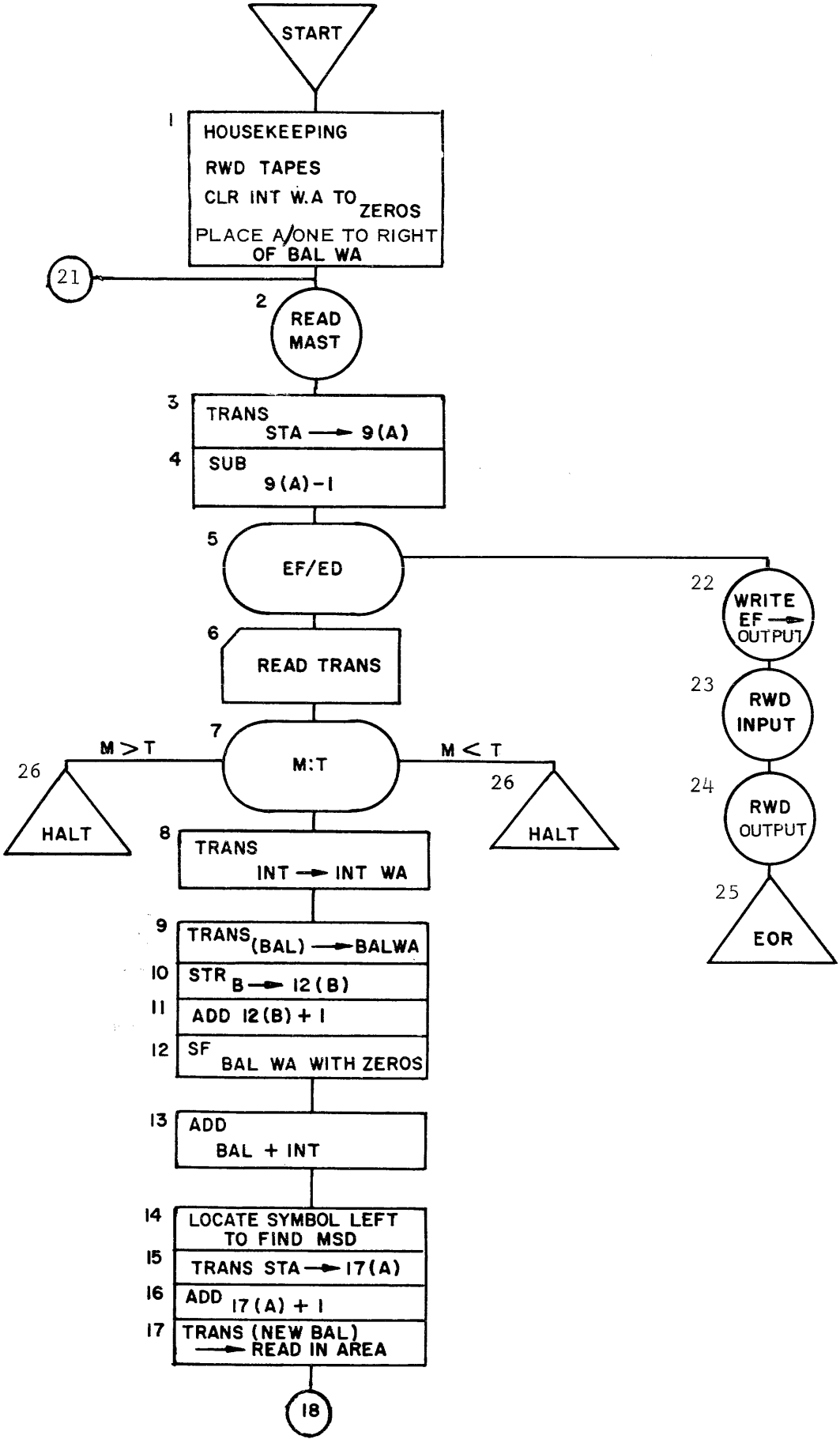
The operation will terminate when the symbol which matches the one held by the N register is sensed and transferred. At that point, STA will hold the address one to the left of the original symbol, the B register will hold the address one to the left of the destination symbol.

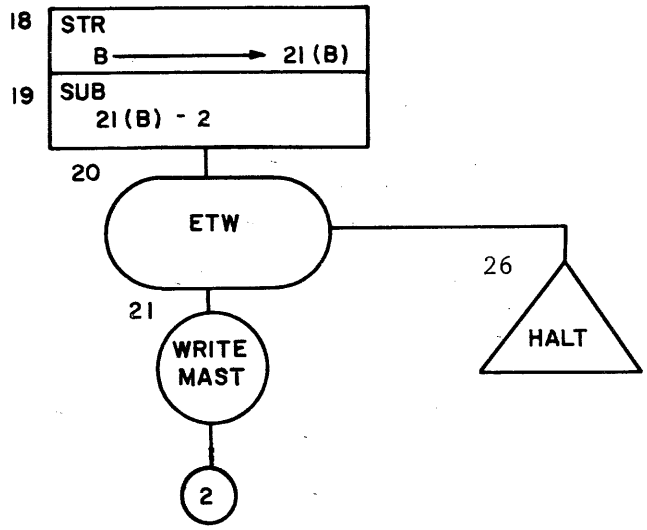
The contents of the B register are very important to us in this case, since we will want to clear to zeros from the left hand end of the work area straight through the /. Since the B register has the address one to the left of the /, we have only to store the contents of the B register in the B address of a Transfer Symbol to Fill instruction, and add one to it. The SF instruction will then load the area with insignificant zeros.

```
J    0    LHE OF WA  (Address of /)
```

Once the Add is accomplished, we will want to find the MSD of the new balance. This can be done by executing a Locate Symbol Left instruction, looking for the lack of a 0. When this instruction terminates, we will have the address of the last 0 sensed in STA. By placing in the A address of a Transfer Data by Symbol Left instruction, and adding 1 to it, we will have the address of the MSD.

In housekeeping, we will place a / one to the right of the work area which is to receive the balance. This / will terminate the instruction. At the end of the instruction, the B register will hold the address one to the right of the destination /, which is also two locations away from the last character of the record. By storing the contents of the B register in the B address of Tape Write Normal instruction, and then subtracting 2 from it, we will be able to write out just the record information. Flow charted and coded the problem would look as follows:


START
1
HOUSEKEEPING
RWD TAPES
CLR INT W.A TO ZEROS
PLACE A ONE TO RIGHT OF BAL WA
21
2
READ MAST
3
TRANS STA → 9(A)
4
SUB 9(A)-1
5
EF/ED
22
WRITE EF → OUTPUT
23
RWD INPUT
24
RWD OUTPUT
25
EOR
6
READ TRANS
7
M:T
M > T
26
HALT
M < T
26
HALT
8
TRANS INT → INT WA
9
TRANS (BAL) → BALWA
10
STR B → 12(B)
11
ADD 12(B) + 1
12
SF BAL WA WITH ZEROS
13
ADD BAL + INT
14
LOCATE SYMBOL LEFT TO FIND MSD
15
TRANS STA → 17(A)
16
ADD 17(A) + 1
17
TRANS (NEW BAL) → READ IN AREA
18


18
STR
B
21 (B)
19
SUB
21 (B) - 2
20
ETW
26
HALT
21
WRITE
MAST
2

	00 01 02 03 04	05 06 07 08 09	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	
40	← ACCT # ———	——→	BALANCE ——→								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
41	← ACCT # ———	——→	← INT ——→								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
42	0 0 0 X X	X X X									
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
43	← BAL ———	WA ——→ /									
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
44	PROG ———	——→									
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
45											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
46											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
47											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
48											
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE Handling Variable Data Example IV

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	440 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	1
		1 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	1
		2 0	0	0	J	0	4	2	0	0	4	2	0	2		FILL INT WA WITH ZEROS	1
		3 0	0	0	N	1	4	7	1	1	4	3	0	9		TRANS /→ BAL WA (RHE + 1)	1
4650		4 0	0	0	4	3	4	0	0	0	4	0	1	5		READ MAST	2
		5 0	0	0	N	4	0	2	1	5	4	5	2	5		TRANS STA→ 9 (A)	3
	6	446 0	0	0	⊖	2	4	5	2	5	4	7	1	9		SUB 9 (A)-1	4
		7 0	0	0	W	8	4	6	6	0	4	4	8	0		ED/EF	5
4470		8 0	0	0	Ø	1	4	1	0	0	0	0	0	0		READ TRANS	6
		9 0	0	0	Y	7	4	0	0	0	4	1	0	0		COMPARE ACCT#	7
		450 0	0	0	W	1	4	7	0	0	4	7	0	0		IF UNEQUAL, HALT	7
		1 0	0	0	N	5	4	1	1	4	4	2	0	7		TRANS INT→ INT W.A.	8
	6	452 0	0	0	P	/	(0	0	0	0)	4	3	0	8	3,4	TRANS (BAL)→BAL W.A.	9
		3 0	0	0	V	4	4	5	5	9	0	0	0	0		STR B → 12 (B)	10
		4 0	0	0	+	3	4	5	5	9	4	7	1	9		ADD 12 (B) + 1	11
		5 0	0	0	J	0	4	3	0	0	(0	0	0	0)	10,11	FILL BAL WA WITH INSIGNIFICANT ZERO	12
		6 0	0	0	+	8	4	3	0	8	4	2	0	7		ADD BAL + INT	13
		7 0	0	0	K	0	4	3	0	0	4	3	0	8		LOCATE LAST INSIGNIFICANT ZERO	14
	6	458 0	0	0	N	4	0	2	1	5	4	6	0	5		TRANSFER STA→ 17 (A)	15
		9 0	0	0	+	1	4	6	0	5	4	7	1	9		ADD 17 (A) +1	16
		460 0	0	0	#	/	(0	0	0	0)	4	0	0	8	15,16	TRANS NEW BAL → READ IN AREA	17
		1 0	0	0	V	4	4	6	4	9	0	0	0	0		STR B→21 (B)	18
		2 0	0	0	⊖	2	4	6	4	9	4	7	2	9		SUB 21 (B) ⊖2	19
		3 0	0	0	S	4	4	0	0	0	4	7	0	0		ETW	20

XXI-16

TITLE Handling Variable Data Example IV
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	464 0	0	0	8	4	4	0	0	0	(0	0	0	0)	18,19	WRITE NEW MASTER	21
		5 0	0	0	V	1	0	2	1	9	4	4	4	0		TRANS → 2	
4770		6 0	0	0	8	4	4	7	1	4	4	7	1	4		WRITE EF → 4	22
		7 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	23
		8 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	24
4630		9 0	0	0	⊡	0	0	0	0	0	0	0	0	0		EOR	25
4500	6	470 0	0	0	⊡	0	0	0	0	0	0	0	0	0		ERROR HALT	26
		1 0	0	0	0	/	0	0	E/F	0	0	0	0	1		CONSTANTS	
		2 0	0	0	0	0	0	0	0	0	0	0	0	2			
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXI-17

Taking a sample record let's step through the program:

The information record is:

2347668/99576

When read into memory it falls as shown:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
40	2	3	4	7	6	6	8	/	9	9	5	7	6	—	—	—

At the end of the Read, STA holds 4013. Transferring that and subtracting 1, we have 4012.

Reading in the transaction for this record, it appears in memory as follows:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
41	2	3	4	7	6	6	7	—	—	—	0	1	0	0	0

Transferring the interest amount to the work area, we have:

	00	01	02	03	04	05	06	07
42	0	0	0	0	1	0	0	0

Transferring the balance by a Transfer Data by Symbol Right instruction which now appears as:

P / (4012) 4308

we have:

	00	01	02	03	04	05	06	07	08	09
43	—	—	—	/	9	9	5	7	6	/

At the termination of this instruction, the B register holds 4302. By storing this and adding 1, we have 4303. Our Transfer Symbol to Fill then reads:

J 0 4300 (4303)

At the completion of that instruction, we have in the work area:

	00	01	02	03	04	05	06	07	08	09
43	0	0	0	0	9	9	5	7	6	/

The Add instruction will change the contents to:

	00	01	02	03	04	05	06	07	08	09
43	0	0	0	1	0	0	5	7	6	/

The next instruction is to locate the last insignificant zero. At the end of the operation, STA will contain 4302. Transferring this and adding one, we have 4303 and our Transfer Data by Symbol Left instruction reads:

/ (4303) 4008

Executing this instruction the read in area will then contain:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
40	2	3	4	7	6	6	8	/	1	0	0	5	7	6	/	—

and the B register will hold 4015. By storing the B register and subtracting 2, our write instruction will then be:

```
8    4    4000    (4013)
```

allowing us to write out just the information pertaining to that record.

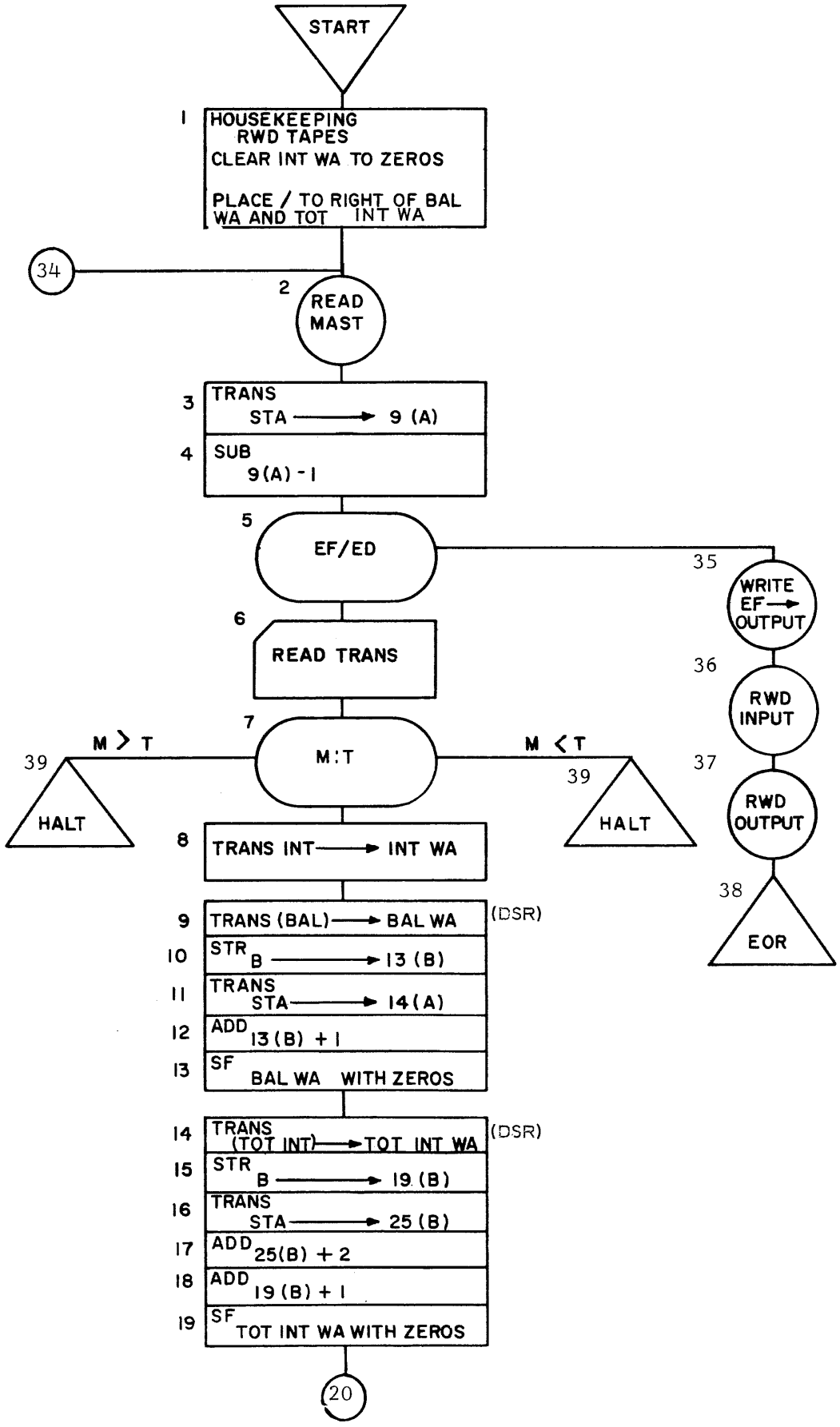
Expanding this problem one step further, suppose our master information was in the format:

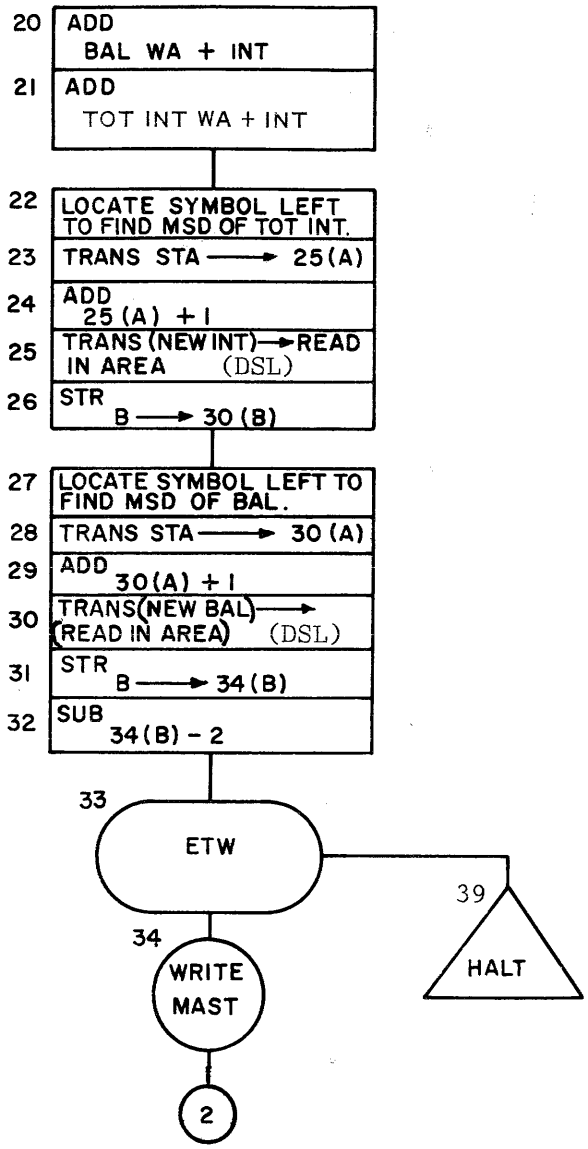
	Acct. #	/	Name	/	Total Interest	/	Balance	(The / serves as a separation between the items)
Max	7		25		7		8	
Avg	7		14		5		6	
%	100		100		100		100	

Now we run the problem of transferring both the Balance and the Total Interest items, both of which are variable. To do this we must transfer STA after the Transfer Data by Symbol Right which places the Balance in the work area. This will give us the address of the location one to the left of the / which separates the Balance and the Total Interest item. This address is the one we need for the A address of another Transfer Data by Symbol Right instruction, which will transfer the Total Interest item to a work area. The storing of the contents of the B register in order to fill the work areas with insignificant zeros will be the same as in the other example.

Another problem is that of reconstructing the record after updating the two items. This can be accomplished if we transfer the contents of STA after the DSR instruction that moves the Total Interest to the work area. That gives us the address one to the left of the / which separates the Name and the Total Interest items. Adding two to this will give us the location which should receive the most significant digit of the new Total Interest amount. Again we would have to locate this most significant digit by locating the last insignificant zero moving from left to right. Having done this, transferring STA to the A address if a Transfer Data by Symbol Left instruction and adding 1 to it will give us the address of the MSD of the Total Interest. If the B address of the DSL instruction then contains the address which is to receive this character, we have only to execute the instruction to put this item back in place, even if the item has been expanded.

We would have to go through a similar process for the new Balance amount, but first we must determine the location which is to receive the MSD of this amount. We can do this by sorting the contents of the B register after the DSL instruction that replaced the Total Interest item. This will give us the address one to the right of the destination location of the terminating /, and this is the location that is to receive the MSD of the new Balance. After the new Balance has been transferred back into the original read in area, we will also want the contents of the B register. Again it will give us the address that is one to the right of the destination location of the terminating /. Subtracting 2 from this will give us the right hand limit of the new, recomposed record.


START
1
HOUSEKEEPING
RWD TAPES
CLEAR INT WA TO ZEROS
PLACE / TO RIGHT OF BAL WA AND TOT INT WA
34
2
READ MAST
3
TRANS
STA → 9 (A)
4
SUB
9(A) - 1
5
EF/ED
35
WRITE EF → OUTPUT
36
RWD INPUT
37
RWD OUTPUT
38
EOR
6
READ TRANS
7
M:T
M > T
39
HALT
M < T
39
HALT
8
TRANS INT → INT WA
9
TRANS (BAL) → BAL WA
(DSR)
10
STR B → 13 (B)
11
TRANS
STA → 14 (A)
12
ADD
13 (B) + 1
13
SF
BAL WA WITH ZEROS
14
TRANS
(TOT INT) → TOT INT WA
(DSR)
15
STR B → 19 (B)
16
TRANS
STA → 25 (B)
17
ADD
25(B) + 2
18
ADD
19 (B) + 1
19
SF
TOT INT WA WITH ZEROS
20


20
ADD
BAL WA + INT
21
ADD
TOT INT WA + INT
22
LOCATE SYMBOL LEFT TO FIND MSD OF TOT INT.
23
TRANS STA → 25(A)
24
ADD
25(A) + 1
25
TRANS (NEW INT) → READ IN AREA (DSL)
26
STR
B → 30(B)
27
LOCATE SYMBOL LEFT TO FIND MSD OF BAL.
28
TRANS STA → 30(A)
29
ADD
30(A) + 1
30
TRANS (NEW BAL) → (READ IN AREA) (DSL)
31
STR
B → 34(B)
32
SUB
34(B) - 2
33
ETW
39
HALT
34
WRITE MAST
2

	00 01 02 03 04	05 06 07 08 09	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	
40	←ACCT # →	/ N A M E →									
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
41	←ACCT # →		← INT →								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
42	0 0 0 x x	x x x INT	WA								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
43	← BAL WA → /		TOT INT WA → /								
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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	
44	PROG →										
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE Handling Variable Data Example V

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	440	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3 INPUT	1
		1	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4 OUTPUT	1
		2	0	0	J	0	4	2	0	0	4	2	0	2		FILL INT WA WITH ZEROS	1
		3	0	0	J	/	4	3	0	9	4	3	0	9		TRANS / TO RIGHT OF BAL W.A.	1
		4	0	0	J	/	4	3	1	9	4	3	1	9		TRANS / TO RIGHT OF TOT INT WA	1
4790		5	0	0	4	3	4	0	0	0	4	0	4	9		READ MASTER	2
	6	446	0	0	N	4	0	2	1	5	4	5	3	5		TRANS STA → 9 (A)	3
		7	0	0	⊖	2	4	5	3	5	4	8	5	9		SUBTRACT 9 (A) - 1	4
		8	0	0	W	8	4	8	0	0	4	4	9	0		EF/ED	5
		9	0	0	0	Ø	4	1	0	0	0	0	0	0		READ CARD	6
		450	0	0	Y	7	4	0	0	0	4	1	0	0		COMPARE ACCOUNT NUMBERS	7
		1	0	0	W	1	4	8	4	0	4	8	4	0		IF NOT EQUAL HALT FOR ERROR	7
	6	452	0	0	N	5	4	1	1	4	4	2	0	7		TRANSFER INT → INT W.A.	8
		3	0	0	P	/	(0	0	0	0)	4	3	0	8	3,4	TRANSFER DATA BY SYMBOT RIGHT (BAL BALWA)	9
		4	0	0	V	4	4	5	7	9	0	0	0	0		STR B → 13(B)	10
		5	0	0	N	4	0	2	1	5	4	5	8	5		TRANS STA → 14 (A)	11
		6	0	0	+	3	4	5	7	9	4	8	5	9		ADD 13 (B) + 1	12
		7	0	0	J	0	4	3	0	0	(0	0	0	0)	10,12	FILL BAL W.A. TO ZEROS	13
	6	458	0	0	P	/	(0	0	0	0)	4	3	1	8	11	DSR (TOT INT → TOT INT WA)	14
		9	0	0	V	4	4	6	3	9	0	0	0	0		STR B → 19 (B)	15
		460	0	0	N	4	0	2	1	5	4	6	9	9		TRANS STA 25 (B)	16
		1	0	0	+	2	4	6	9	9	4	8	5	5		ADD 25(B) +2	17
		2	0	0	+	1	4	6	3	9	4	8	5	9		ADD 19 (B) +1	18
		3	0	0	J	0	4	3	1	1	(0	0	0	0)	15,18	FILL TOT INT WA TO ZEROS	19

TITLE Handling Variable Data Example V

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	464 0	0	0	+	8	4	3	0	8	4	2	0	7		ADD BAL+ INT	20
		5 0	0	0	+	7	4	3	1	8	4	2	0	7		ADD TOT INT + INT	21
		6 0	0	0	K	0	4	3	1	2	4	3	1	8		LOCATE LAST ZERO OF TOT INT	22
		7 0	0	0	N	4	0	2	1	5	4	6	9	5		TRANS STA → 25(A)	23
		8 0	0	0	+	1	4	6	9	5	4	8	5	9		ADD 25(A) + 1	24
		9 0	0	0	#	/	(0	0	0	0)	(0	0	0	0)	16,17 23,24	T SL NEW TOT INT → TOT INT	25
	6	470 0	0	0	V	4	4	7	4	9	0	0	0	0		STR B → 30 (B)	26
		1 0	0	0	K	0	4	3	0	1	4	3	0	8		LOCATE LAST ZERO OF BAL	27
		2 0	0	0	N	4	0	2	1	5	4	7	4	5		TRANS STA → 30 (A)	28
		3 0	0	0	+	1	4	7	4	5	4	8	5	9		ADD 30(A) + 1	29
		4 0	0	0	#	/	(0	0	0	0)	(0	0	0	0)	26,28,29	DSL NEW BAL → BAL	30
		5 0	0	0	V	4	4	7	8	9	0	0	0	0		STR B → 34 (B)	31
	6	476 0	0	0	⊖	2	4	7	8	9	4	8	5	5		SUB 34(B) -2	32
		7 0	0	0	S	4	4	0	0	0	4	8	4	0		ETW	33
		8 0	0	0	8	4	4	0	0	0	(0	0	0	0)	31,32	WRITE MASTER	34
		9 0	0	0	V	1	0	2	1	9	4	4	5	0		TRANS → 2	
4480		480 0	0	0	8	4	4	8	5	1	4	8	5	1		WRITE EF · → OUTPUT (4)	35
		1 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3	36
	6	482 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4	37
4770		3 0	0	0	⌐	0	0	0	0	0	0	0	0	0		EOR	38
4510		4 0	0	0	⌐	0	0	0	0	0	0	0	0	0		HALT ERROR	39
		5 0	0	0	⌐	E/F	0	0	0	2	0	0	0	1			
		0	0	0													
		0	0	0													

XXI-24

IE 1224 REV. 8-60

Suppose our sample record read:

3456436/JOE_DOE/9984/99750

and our transaction amount was $5.00.

When read into memory, we would have:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
40	3	4	5	6	4	3	6	/	J	O	E	_	D	O	E	/	9	9	8	4

	20	21	22	23	24	25
	/	9	9	7	5	0

By picking up the contents of STA after the read and subtracting 1, we have a DSR instruction that reads:

P / (4025) 4308

Transferring STA after this instruction will give us 4019, the address of the LSD of the Total Interest amount. With this in the A address our DSL instruction will appear as follows:

P / (4019) 4318

Remembering that the work areas had to be cleared to insignificant zeros, after the addition we would have:

	00	01	02	03	04	05	06	07	08	09
43	0	0	0	1	0	0	2	5	0	/

	11	12	13	14	15	16	17	18	19
43	0	0	0	1	0	4	8	4	/

Assuming that we transferred the contents of STA after the last DSR instruction and modified it by adding two, and assuming that we located the last insignificant zero and added one to this address, our Transfer Data by Symbol Left instruction would read:

/ (4314) (4016)

At the termination of this instruction, the B register would hold 4022, the location which must receive the MSD of the new Balance item. Having transferred this amount back to the original read in area by an instruction which reads:

/ (4303) (4022)

we would have in the B register 4029. Storing this and subtracting 2 we have the address of the right hand end of the new record, which appears in memory as follows:

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
40	3	4	5	6	4	3	6	/	J	O	E	_	D	O	E	/	1	0	4	8

	20	21	22	23	24	25	26	27	28	29
	4	/	1	0	0	2	5	0	/	_

The write instruction would then be:

8 4 4000 4027

Handling Variable Data Exercise I:

We want to run a totaling program to develop a grand total of the gross dollar amount of the issues to be used at a later date as control information. The information is coming in sorted from tape unit 6 in the format:

	STK.#	/ DESCRIPTION	/ GROSS $ AMT.	/ DISCOUNT AMT.	/ NET $ AMT.
Max.	8	25	7	5	7
Avg.	8	12	6	4	6
%	100	100	100	100	100

It is necessary to locate the Gross Dollar Amount and add it to a 10 digit work area. When the last record has been processed, place the zero suppressed grand total after the EF, and follow this with an ED. This is called a "trailer message".

Assume 1 reel of information.

Handling Variable Data Exercise II:

The problem is to print out a notification for every overdrawn account in the file.

Data Description:

Master file: (1 reel tape unit E)

	Account #	• Name	• St. Add.	• City-State Add.	• Balance
Max.	6	25	30	30	6
Avg.	6	14	16	15	5
%	100	100	100	100	100

(The • (called ISS-Item Separator Symbol) differentiates the items. This symbol appears as a space when printed on the on-line printer.)

PRINT FORMAT

PRINT POSITIONS

```
1_____30              54_____60
ACCOUNT NUMBER
NAME
STREET ADDRESS
CITY-STATE
OVERDRAWN             BALANCE-
```

An example:

```
123456
JOE DOE
123 ANY STREET
SOMEPLACE, N.J.
OVERDRAWN          689-
```

Assume that VT channel is punched appropriately.
Do not flow chart or code ED/EF or ETW routines.

Handling Variable Data Exercise III:

We are to write a program which will update the master insurance file, by posting the received premiums to the appropriate masters. There will never be more than one premium per master.

DATA DESCRIPTION:

Master (tape 2)

	Policy Number	Name	St. Add.	City-State Add.	Total Premiums paid to date	Premiums Due
Max.	9	30	30	30	6	6
Avg.	9	16	17	17	5	5
%	100	100	100	100	100	100

Transactions (tape 3, sorted in ascending order by policy number)

Policy Number	Premium Amount
9	4

(Punch reject cards for any out of sort transactions. These reject cards have the same format as the input).

Do not flow chart or code ED/EF or ETW routines.

Error Recovery Exercise
File Maintenance

XXII – REPETITION OF INSTRUCTIONS

It is often desirable to repeat a given instruction a number of times. For example, suppose we wanted to multiply the gross sales amount by 6% to obtain the discount amount. One way we could do it is to add the weekly gross to itself 8 more times. We could code this with 7 Add instructions:

```
          Gross sales amount
34        34 35 36 37 38 39 40 ≀ 54 55 56 57 58 59 60
           0  7  9  4  6  7  0 ≀  0  7  9  4  6  7  0

     1730 + 7 3440  3460
     1740 + 7 3440  3460
     1750 + 7 3440  3460
     1760 + 7 3440  3460
     1770 + 7 3440  3460
     1780 + 7 3440  3460
     1790 + 7 3440  3460

34        Discount amount
          34 35 36 37 38 39 40
           6  3  5  7  3  6  0  (63.57)
```

With the REPEAT instruction, this same operation would require only two instructions.

REPEAT:

The OP code is R.

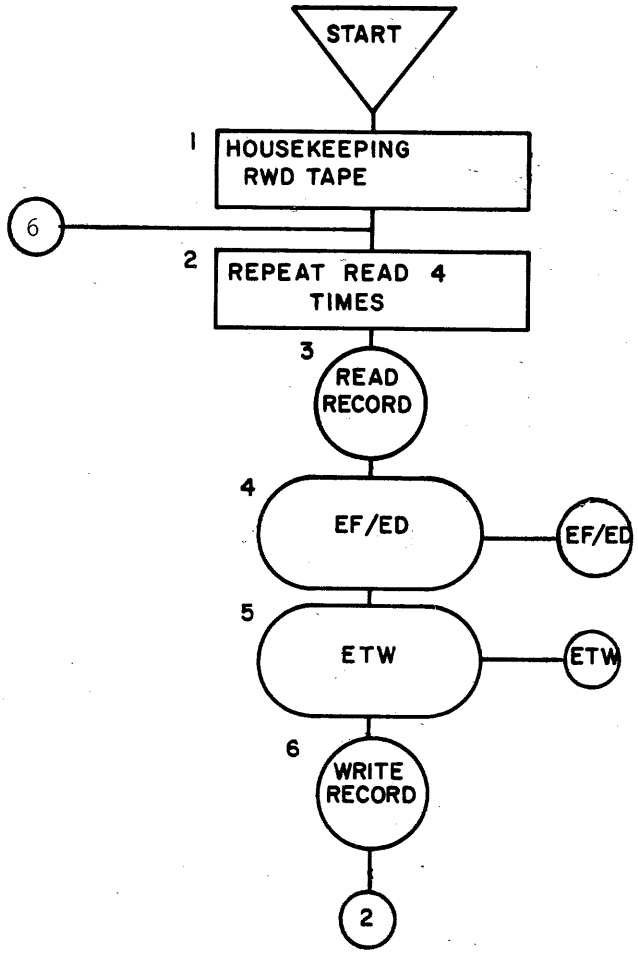
N indicates the number of times (0-14) to repeat the next repeatable instruction, expressed by the decimal digits and 10 is ⊖ , 11 is #, 12 is @, 13 is (, and 14 is).

The A address indicates whether or not the A address of the repeating instruction is to be staticized. If it is 0000, the A address will not staticize. If it is 0001, the A address will staticize.

The B address indicates whether or not the B address of the repeating instruction is to be staticized. If it is 0001, the B address will staticize. Our example would then be coded as follows:

```
1730 R 6 0001  0001
1740 + 7 3440  3460
```

Since a read instruction requires start time of 9.8 ms., it is advisable to "batch" records. This means that instead of having one record per block, we could place multiple records in a block, thus decreasing the number of gaps to be processed. Our problem is to write a program which will "batch" these records. Assuming that there are 10,000 fifty character records on one reel of tape and we are required to batch records into 5 records per block, we would want to read in 5 times, placing the records end to end in memory. At that point we could write out all five records as one block. By using a Repeat instruction, we can repeat the Read 4 times. If we don't staticize the A address, we will always have in the A register the address of the location to receive the next character (after reading in a record, the A register contains the address of the location one to the right of the last character read in). Noting this, our program would be as follows:


START
1
HOUSEKEEPING
RWD TAPE
6
2
REPEAT READ 4
TIMES
3
READ
RECORD
4
EF/ED
EF/ED
5
ETW
ETW
6
WRITE
RECORD
2

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	270 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD INPUT	1
		1 0	0	0	;	5	0	0	0	0	0	0	0	0		RWD OUTPUT	1
2770		2 0	0	0	R	4	0	0	0	0	0	0	0	1		REPEAT READ 4 TIMES	2
		3 0	0	0	4	3	1	0	0	0	1	2	4	9		READ RECORD	3
		4 0	0	0	W	8	E	D	E	F	2	7	5	0		ED/EF	4
		5 0	0	0	S	5	4	0	0	0	E	T	W			ETW	5
	6	276 0	0	0	8	5	1	0	0	0	1	2	4	9		WRITE BATCH	6
		7 0	0	0	V	1	0	2	1	9	2	7	2	0		TRANS 2	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

In the chapter on handling variable data, we did an example which located a variable item. In our particular example, we only had to execute a Transfer Data by Symbol Left instruction once, as there was only one variable item between the last FAA (fixed and always appearing--fixed with regard to number of characters and relation to first character and 100% occuring) item and the item we were looking for. Suppose, however, our file had been as follows:

	Acct. # •	Name •	St. Add •	City-State Add •	Date of Statement•
Max.	6	25	30	30	6
Avg.	6	166	17	15	6

Total Deposits •	Total Checks •	Balance
7	7	6
6	6	5

(An ISS separates each item)

In order to locate the left hand end of the Date, we would have to Transfer Data by Symbol Left three times, each time picking up the contents of STA and the B register to set up the next DSL instruction:

```
     40  41  52  43  44  45  46  47  48  49  50
16 <-------- Acct. # ---------> •     Name
```

```
3440 # • 1647 1647  Transfer name in place                    1
3450 V 4 3479 0000  Str B in 4(B)                            2
3460 N 4 0215 3475  Trans STA to 4(A)                        3
3470 # • (  ) (  )  Transfer St. Add. in place               4
3480 V 4 3509 0000  Str B in 7(B)                            5
3490 N 4 0215 3505  Trans STA to 7(A)                        6
3500 # • (  ) (  )  Transfer City-State Add in place         7
3510 N 4 0215 3525  Trans STA to 9(A)                        8
3520 Y 6 (  ) 1700  Compare Statement Date:  Today's Date    9'
```

We could accomplish this same routine with only 4 steps using a Repeat instruction:

```
3440 R 2 0000 0000  Repeat the DSL 2 times, not stat.        1
3450 # • 1647 1647  Transfer in place (DSL)                  2
3460 N 4 0215 3475  Trans STA to 4(A)                        3
3470 Y 6 (  ) 1700  Compare Statement Date:  Today's Date    4
```

Suppose we were to transfer the 100 characters located in memory from 1500-1599 to locations 2100-2199. We already know that the most we can transfer with one instruction is 44 characters. If we executed 4 transfers of 25 characters each, however, we would get a transfer of 100 characters. We must make sure that the registers do not re-staticize between executions or we will only succeed in transferring the same 25 characters 4 times. We can accomplish this by prohibiting both the A and the B registers from receiving data:

```
R 3 0000 0000
M N 1500 2100
```

Not all instructions in the RCA 301 complement are repeatable. We have mentioned four:

Add
Tape Read Forward Normal
Transfer Data by Symbol Left
Transfer Data Left

The other repeatable instructions are:

Subtract
Tape Read Reverse Normal
Transfer Data by Symbol Right
Transfer Data Right
Logical "OR"
Logical "AND"
Logical "EXCLUSIVE"
Translate by Table

The following outline of operation is included in order to explain the internal logic of the Repeat instruction, which is not as obvious as some of the other instructions:

The contents of the N register are transferred to the N_R register (a one character register which holds and keeps track of the N character of the Repeat instruction). The contents of the P register are transferred to standard HSM locations 0222-0225. Indicators are set specifying whether the staticizing of the A and B addresses are inhibited or not, according to the A and B addresses of the Repeat instruction. The next instruction is staticized disregarding the settings of the indicators. The sequence of instruction executions differs according to whether this instruction (the one immediately following the Repeat instruction) is repeatable or non-repeatable.

If the instruction is repeatable, it is completed and the sequence of instruction executions occurs in the following cycle:

The contents of the N_R register are examined. If zero, the indicators are reset and the next instruction is executed. If other than zero, the contents of the N_R register are decremented by one and the contents of the HSM locations 0222-0225 are transferred to the P register. The instruction addressed by the P register is then staticized as specified by the indicators and is executed. The cycle repeats itself until the N_R register contains a zero.

If the instruction is non-repeatable, it is completed, ignoring the indicators and henceforth all successive non-repeatable instructions and the next repeatable instruction is staticized as specified by the indicators and executed. When a repeatable instruction is encountered, the sequence of instruction executions occurs exactly as shown above. Remember, however, that all the instructions between the Repeat and the first repeatable instruction will be staticized (according to the indicators) and executed each time.

Repetition of Instructions Exercise I:

Write the portion of a weekly payroll which would calculate the Social Security, at a rate of 3%. The information is coming in from cards in the format:

Emp #	Gross Pay
5	5
1-5	10-14

You are to bring in the information, calculate the social security, and place the amount in the proper positions so that the output card will have the format:

Emp #	Gross Pay	Social Security
5	5	3
1-5	10-14	20-22

(You may assume that no salary is over $300.00 per week.)

At this point, your part of the program is complete. Ignore sensing for End of File.

Repetition of Instructions Exercise II:

Indicate how Handling of Variable Data Exercise I could have been shortened if a Repeat had been incorporated.

XXIII – INDIRECT ADDRESSING

Everytime that we have needed the contents of STA, we have been forced to transfer them out of STA and into the instruction that required them. The concept of "indirect addressing" will eliminate many of these transfers.

All of our instructions to date have contained direct addresses; for example, the instruction:

```
M  6  3049  4099
```

will move the 6 characters starting at 3049 to 4099. An indirect address, on the other hand, tells the computer that it is not the address of the data to be operated on, but rather the address of a storage location which contains the direct address. An indirect address identifies itself by containing a zone bit in the 2^4 of the least significant digit. Since direct addresses contain four digits, the storage location for the direct address must also be four characters in length. These four characters must lie in two even diads, and the indirect address must be the address of the right hand end of these two diads. The following table shows the character needed for each numeric:

& for 0	E for 5
A for 1	F for 6
B for 2	G for 7
C for 3	H for 8
D for 4	I for 9

An example will help to clarify this concept. In the last chapter, we worked out a routine to locate the date in a field of variable information. The routine appeared as follows:

```
3440 R 2 0000 0000
3450 # / 1647 1647
3460 N 4 0215 3475
3470 Y 6 (  ) 1700
```

We could eliminate the Transfer Data Right instruction by utilizing indirect addressing. The program would then appear as follows:

```
3440 R 2 0000 0000
3450 # / 1647 1647
3460 Y 6 021E 1700
```

Supposing that our record was in memory as follows:

```
          40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
16         1  3  5  2  5  7  /  J  A  M  E  S  _  S  M  I  T  H  /  2

          60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
           0  _  P  A  R  K  _  S  T  .  /  E  R  I  E  _  P  A  .  /

          80 81 82 83 84 85 86
           6  0  1  0  2  3  /
```

STA at the end of the repeat loop would contain 1680:

```
          12 13 14 15
02         1  6  8  0
```

When the Compare instruction is staticized, the registers will contain:

OP	N	A	B
Y	6	021E	1700

The computer recognized that 021E is an indirect address (by the zone bit) and goes to locations 0212-0215 and replaces the contents of A with the contents of those locations. The registers then contain:

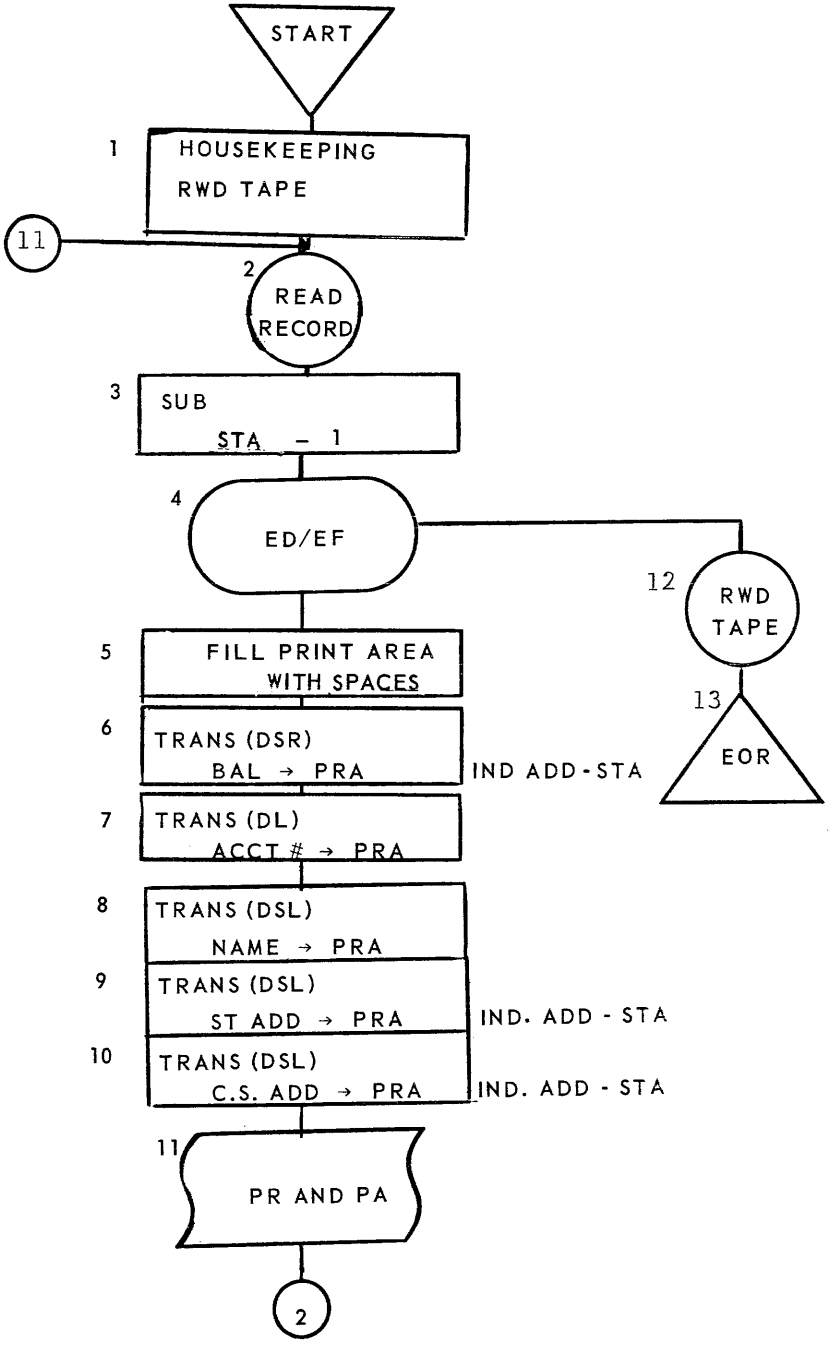
OP	N	A	B
Y	6	1680	1700

and the compare is ready to be executed.

Suppose we wanted to set up a print area and print a summary line for every master in the file we were just discussing (page XXII-3). The print line has a format of:

Account #	• Name	• St. Add.	• City-State Add.	• Balance
1-6	10-34	36-65	67-96	115-120
				(assume only positive balance)

Since the ISS (Item Separator Symbol) is not printable, we do not have to worry about clearing it to spaces. The problem would appear as follows:


START
1
HOUSEKEEPING
RWD TAPE
11
2
READ
RECORD
3
SUB
STA - 1
4
ED/EF
12
RWD
TAPE
13
EOR
5
FILL PRINT AREA
WITH SPACES
6
TRANS (DSR)
BAL → PRA
IND ADD-STA
7
TRANS (DL)
ACCT # → PRA
8
TRANS (DSL)
NAME → PRA
9
TRANS (DSL)
ST ADD → PRA
IND. ADD - STA
10
TRANS (DSL)
C.S. ADD → PRA
IND. ADD - STA
11
PR AND PA
2

	00 01 02 03 04	05 06 07 08 09	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	
10	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
11	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
12	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
13	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
14	00 01 02 03 04	05 06 07 08 09	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	
	ACCT	#	NAME					ST. ADD			
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
				CITY - STATE ADD.							
15	00 01 02 03 04	05 06 07 08 09	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	
				BAL							
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
16	00 01 02 03 04	05 06 07 08 09	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	
									ACCT #	•	NAM
	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
	E										
17	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
18	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	
19	00 01 02 03 04	05 06 07 08 09	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 81 82 83 84	85 86 87 88 89	90 91 92 93 94	95 96 97 98 99	

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

Indirect Addessing Example I

TITLE
CODER
REMARKS

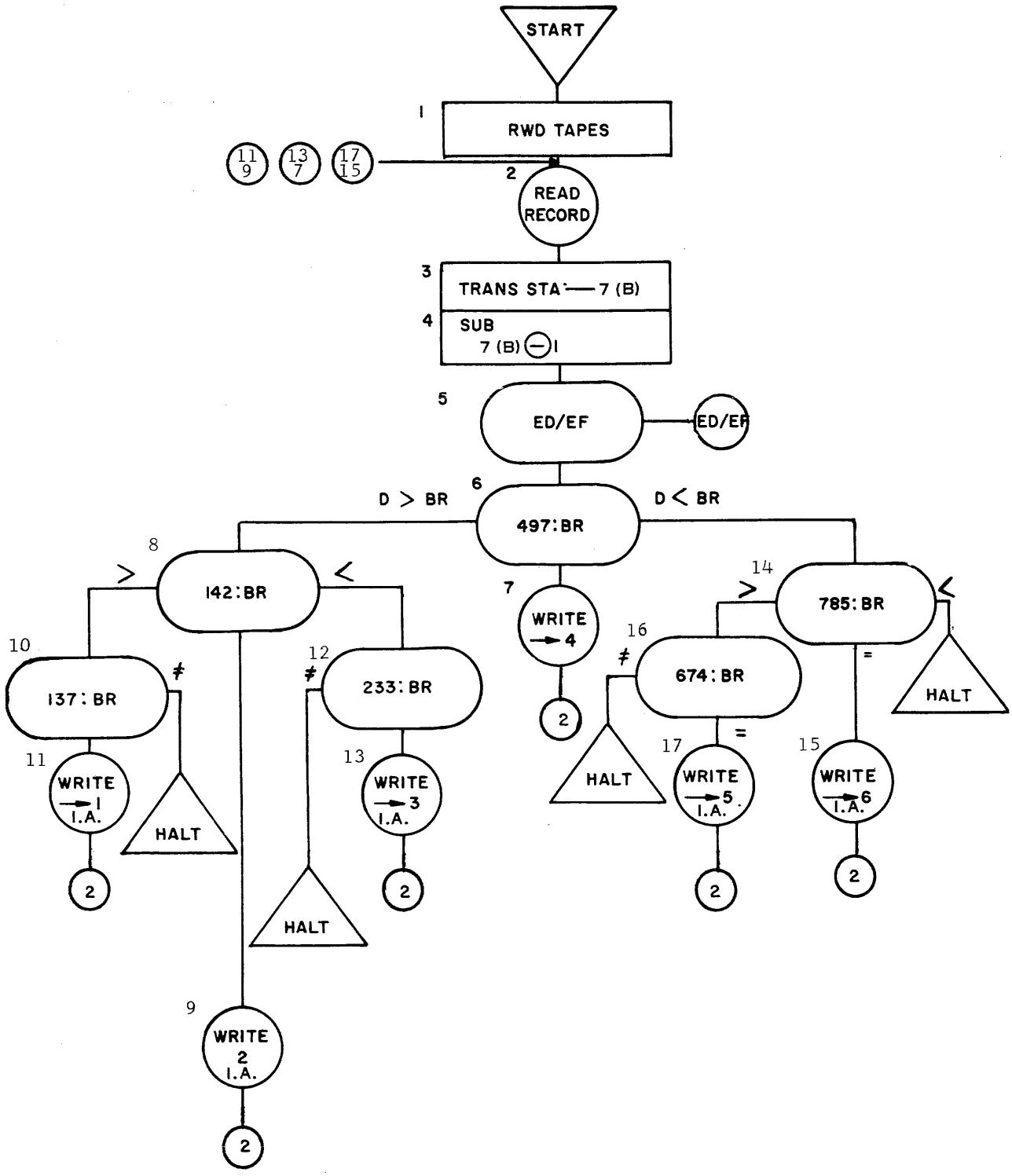
DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	200 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD INPUT	1
2110		1 0	0	0	4	3	1	6	4	0	1	7	6	4		READ RECORD	2
		2 0	0	0	⊖	3	0	2	1	5	2	1	4	9		SUB STA - 1	3
		3 0	0	0	W	8	2	1	2	0	2	0	4	0		ED/EF	4
		4 0	0	0	J	-	1	4	0	0	1	5	1	9		FILL PRINT AREA WITH SPACES	5
		5 0	0	0	P	●	0	2	1	E	1	5	1	9		DSR. (BAL)→PRA IND ADD	6
	6	206 0	0	0	M	6	1	6	4	0	1	4	0	0		D L ACCT #→PRA	7
		7 0	0	0	#	●	1	6	4	7	1	4	0	9		DSL NAME→PRA	8
		8 0	0	0	#	●	0	2	1	E	1	4	3	5		DSL (ST. ADD.)→PRA	9
		9 0	0	0	#	●	0	2	1	E	1	4	6	6		DSL (C-S ADD.)→PRA	10
		210 0	0	0	B	0	0	0	0	0	1	4	0	2		PRINT AND P. A.	11
		1 0	0	0	V	1	0	2	1	9	2	0	1	0		TRANS→2	
2030	6	212 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD INPUT	12
		3 0	0	0	⊡	0	0	0	0	0	0	0	0	0		EOR	13
		4 0	0	0	⊡	0	0	0	0	0	0	0	0	1		CONSTANTS	
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

We have already seen that incorporating indirect addressing wherever possible saves instructions, but in addition it saves time. Performing a transfer of STA takes 91 microseconds (35 us staticizing plus 4 x 14 operating time). Utilizing indirect addressing takes only 14 microseconds per level (indirect addressing may be multi-level; i.e., an indirect address may address an indirect address, which may in turn address an indirect address, etc.). Thus we are saving 77 microseconds every time we use it.

To date, we have only used STA as an indirect address. Any two consecutive diads in memory may be indirect address storage locations, however. Suppose we had to prepare six small work files from one master file, each work file to contain the records for one particular branch. When we read in a record, we have in STA the address of the location one to the right of the last character read in. By picking this up and subtracting one, we would have the address of the last character in the sector. However, we need this address in six different write instructions, since at this point of the program we do not know which branch we are working with. One way to handle it would be to place this address in the most used write instruction, and then having determined another path, transfer the address from this write to the proper write. An even easier and faster way would be to use this the B address of this write as an indirect address in the alternate write instructions. The program would then be as follows: (ED/EF and ETW ignored)

Data Description:
Master (100 character maximum, 1 reel)
Branch code
3
3
100
(137, 142, 233, 497, 674, 785 and 497 is the high hit)


START
1
RWD TAPES
11 9
13 7
17 15
2
READ RECORD
3
TRANS STA — 7 (B)
4
SUB 7 (B) — 1
5
ED/EF
ED/EF
D > BR
6
497: BR
D < BR
8
142: BR
>
<
7
WRITE → 4
2
14
785: BR
>
<
=
HALT
16
≠
674: BR
HALT
17
=
WRITE → 5 I.A.
2
15
WRITE → 6 I.A.
2
10
137: BR
≠
11
WRITE → 1 I.A.
HALT
2
12
≠
233: BR
13
WRITE → 3 I.A.
2
HALT
9
WRITE 2 I.A.
2

TITLE Indirect Addressing Example II

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	300 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 137 TAPE	1
		1 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 142 TAPE	1
		2 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 233 TAPE	1
		3 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 497 TAPE	1
		4 0	0	0	;	5	0	0	0	0	0	0	0	0		RWD 674 TAPE	1
3340 3300		5 0	0	0	;	6	0	0	0	0	0	0	0	0		RWD 78 TAPE	1
3260	6	306 0	0	0	;	A	0	0	0	0	0	0	0	0		RWD INPUT	1
3180 3140 3220		7 0	0	0	4	A	1	0	0	0	1	0	9	9		READ RECORD	2
		8 0	0	0	N	4	0	2	1	5	3	1	3	9		TRANS STA → 8(B)	3
		9 0	0	0	⊖	2	3	1	3	9	3	3	6	1		SUB 8 (B) ⊖ 1	4
		310 0	0	0	W	8	E	D	E	F	3	1	1	0		ED/EF	5
		1 0	0	0	Y	3	3	3	6	3	1	0	0	0		COMPARE 497: BR	6
	6	312 0	0	0	W	1	3	1	5	0	3	2	7	0		+ → 8; ⊖ → 14	7
		3 0	0	0	8	4	1	0	0	0	(0	0	0	0)	3,4	WRITE →1	8
		4 0	0	0	V	1	0	2	1	9	3	0	7	0		TRANS →2	
3120		5 0	0	0	Y	3	3	3	6	7	1	0	0	0		COMPARE 142: BR	8
		6 0	0	0	W	1	3	1	9	0	3	2	3	0		+ ⊖ 10: ⊖ →12	9
		7 0	0	0	8	2	1	0	0	0	3	1	3	I		WRITE →2	9
	6	318 0	0	0	V	1	0	2	1	9	3	0	7	0		TRANS →2	
3160		9 0	0	0	Y	3	3	3	7	3	1	0	0	0		COMPARE 137: BR	10
		320 0	0	0	W	1	3	3	5	0	3	3	5	0		± → HALT	10
		1 0	0	0	8	1	1	0	0	0	3	1	3	I		WRITE →1	11
		2 0	0	0	Y	1	0	2	1	9	3	0	7	0		TRANS →2	
3160		3 0	0	0	Y	3	3	3	7	7	1	0	0	0		COMPARE 233:B	12

8-IIIXX

IE 1224 REV. 8-60

TITLE Indirect Addressing Example II
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	324 0	0	0	W	1	3	3	5	0	3	3	5	0		+ → HALT	12
		5 0	0	0	8	3	1	0	0	0	3	1	3	I		WRITE → 3	13
		6 0	0	0	V	1	0	2	1	9	3	0	7	0		TRANS → 2	
3120		7 0	0	0	Y	3	3	3	8	3	1	0	0	0		COMPARE 785: BR	14
		8 0	0	0	W	1	3	3	1	0	3	3	5	0		+ → 16 → HALT	14
		9 0	0	0	8	6	1	0	0	0	3	1	3	I		WRITE → 6	15
	6	330 0	0	0	V	1	0	2	1	9	3	0	7	0		TRANS → 2	
3280		1 0	0	0	Y	3	3	3	8	7	1	0	0	0		COMPARE 674: Br	16
		2 0	0	0	W	1	3	3	5	0	3	3	5	0		− → HALT	16
		3 0	0	0	8	5	1	0	0	0	3	1	3	I		WRITE → 5	17
3320		4 0	0	0	V	1	0	2	1	9	3	0	7	0		TRANS → 2	
3200 3240		5 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT	
3280	6	336 0	0	0	0	1	0	4	9	7	0	1	4	2			
		7 0	0	0	0	0	0	1	3	7	0	2	3	3		CONSTANTS	
		8 0	0	0	0	0	0	7	8	5	0	6	7	4			
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

6-IIXX

IE 1224 REV. 8-60

Indirect Addressing Exercise I:

The problem is to prepare a summary print out for every master in the stock inventory file:

Stk #	Desc.	Mfg. Name	Mfg. St. Add.	Mfg. C-S Add.	Tot. Iss'd
7	50	35	30	30	10
7	10	15	15	15	10

Tot. Rec'd	Due In	Orders Not Met	Balance
10	5	7	8
10	5	7	8

(ISS's separate the items as shown; numeric fields are carried with insignificant zeros.)

The Print Format is to be as follows:

Stk #	Due In	Orders Not Met	Balance
1-7	20-24	30-36	40-47

Assume that the VT channel has been punched to give all necessary paper advance. No insignificant zeros are to appear on the print out.

Surpress

0600-0606 Stock #
0620-0624 Due In
0630-0636 Orders Not Met
0640-0647 Balance

XXIV – TIMING

Several instances within the text we have mentioned timing and it seems advisable that we take it up in more detail at this point.

Timing within a computer is talked of in terms of "microseconds" and "milliseconds". A millisecond is 1/1000 of a second. A microsecond is 1/1000 of a millisecond or 1/1,000,000 of a second. Each instruction has a formula that will enable you to compute the time it takes to execute it. This formula is made up of two or three parts:

1) 35 microseconds for staticizing the instruction

2) 14 microseconds for STA (if applicable)

3) the operating time

For example, the timing for the Transfer Data Right instruction is:

14n + 35

where n equals the number of characters transferred. If we were transferring STA, for example, the formula would be:

14 x 4 x 35 = 91 microseconds

An additional 14 microseconds must be added for every indirect address level that is used.

The Repeat instruction deserves special notice. Each repeatable instruction, when repeated, requires three additional status levels (21 microseconds) when the contents of the N_R Register exceeds zero, and requires one additional status level (7 microseconds) when the contents of the N_R Register equal zero.

In timing fixed information, the maximum number of characters is used. In timing variable information, the weighted average is used.

In addition to knowing how long it takes to execute a given instruction once, we must also know how many times that particular instruction will be executed during the problem. This we determine by the volumes given.

As an example of timing, lets time out the problem on page XIX-4, assuming that 50% of the inventory is active, and 1/8 of 1% of the transactions are out of sort. We will ignore housekeeping and the EF routine.

BLOCK	INSTRUCTION TIME	VOLUME	TOTAL TIME SECONDS
2	17 Char x 100 us/char + 9.8 ms up to speed time	10,000	115.000
3	49 us	10,000	0.490
4	49 us	10,000	0.490
6	17 Char x 100 us/char + 9.8 ms up to speed time	8,000	92.000
7	49 us	8,000	0.392
8	21 us x 4 (average) + 35us	18,000	2.142
8	49 us	10,000	0.490
8	35 us	7,990	0.280
9	28 us x 10 + 49us	7,990	2.629
9	49 us	7,990	0.392
10	35 us	10	0.0004
11	17 Char x 100 us/char + 9.8ms up to speed time+ 3.9ms gap + 22.4ms delay + 10ms switch	10	0.478
11	49 us	10	0.0005
12	35 us	10,000	0.350
13	17 Char x 100 us/char + 9.8ms up to speed time+ 3.9ms gap + 22.4ms delay + 10ms switch	10,000	478.000
13	49 us	10,000	0.490
	Total Time (approximate) or 11 minutes and 6 seconds		693.6239

This time is only approximate since we did not time any instructions which were executed only once in the program.

The following indicates timing specifications for RCA 301 peripheral equipment:

PAPER TAPE

Paper Tape Reader/Punch - 100 characters per second
Paper Tape Reader - 500 or 1000 characters per second
Paper Tape Punch - 100 characters per second

MAGNETIC TAPE

	10KC	33KC	66KC
Tape speed (inches/sec)	30	100	100
Characters/inch	333	333	667
Hot up to speed time (ms)	9.8*	3.5*	5.5**
Hot gap size (inches)	.3	.45	.55
Switch time (ms)	10	none	none
Write to Read Delay (ms)	22.4±20%	7.2±20%	5.83±20%
Rewind speed (inches/second)	30	100	100

*Does not include staticizing time.

**Includes read after write stop delay, but not staticizing time.

CARD READER

Card Reader - 600 cards per minute
Card Reader/Punch - 800 cards per minute

CARD PUNCH

Card Punch - 100 cards per minute
Card Reader/Punch - 200 cards per minute

ON-LINE PRINTER

The two On-Line Printer Models can be operated in the following combinations at the indicated speeds:

NO OF PRINTERS	MODEL	SYNCHRONOUS SPEED (LPM)	ASYNCHRONOUS SPEED (LPM)	NORMAL MODE(N) OR SIMULTANEOUS MODE(S)
1	120 Column	1,000	800	N or S
1	160 Column	1,070	835	N
1	160 Column	715	600	N or S
2	120 Column #1	1,000	800	N or S
	120 Column #2	1,000	800	N or S
2	160 Column #1	715	600	N or S
	160 Column #2	715	600	N or S

NOTE: Timing for the Data Record File and the Data Disc File will be discussed in their respective sections.

A complete table of instruction times is given in Appendix IV of the RCA 301 Programmers' Reference Manual.

TIMING EXERCISE I:

Time the problem found on pages XIX-8 - XIX-13. Assume the following volumes:

10,000 masters
9,500 transactions
9,000 to be posted against existing masters
500 new masters

Do not time ED/EF, ETW, or Housekeeping.

TIMING EXERCISE II:

Time the program on pages XXI-10 to XXI-19. Assume the number of masters to be 10,000. Do not time Housekeeping or EF routines.

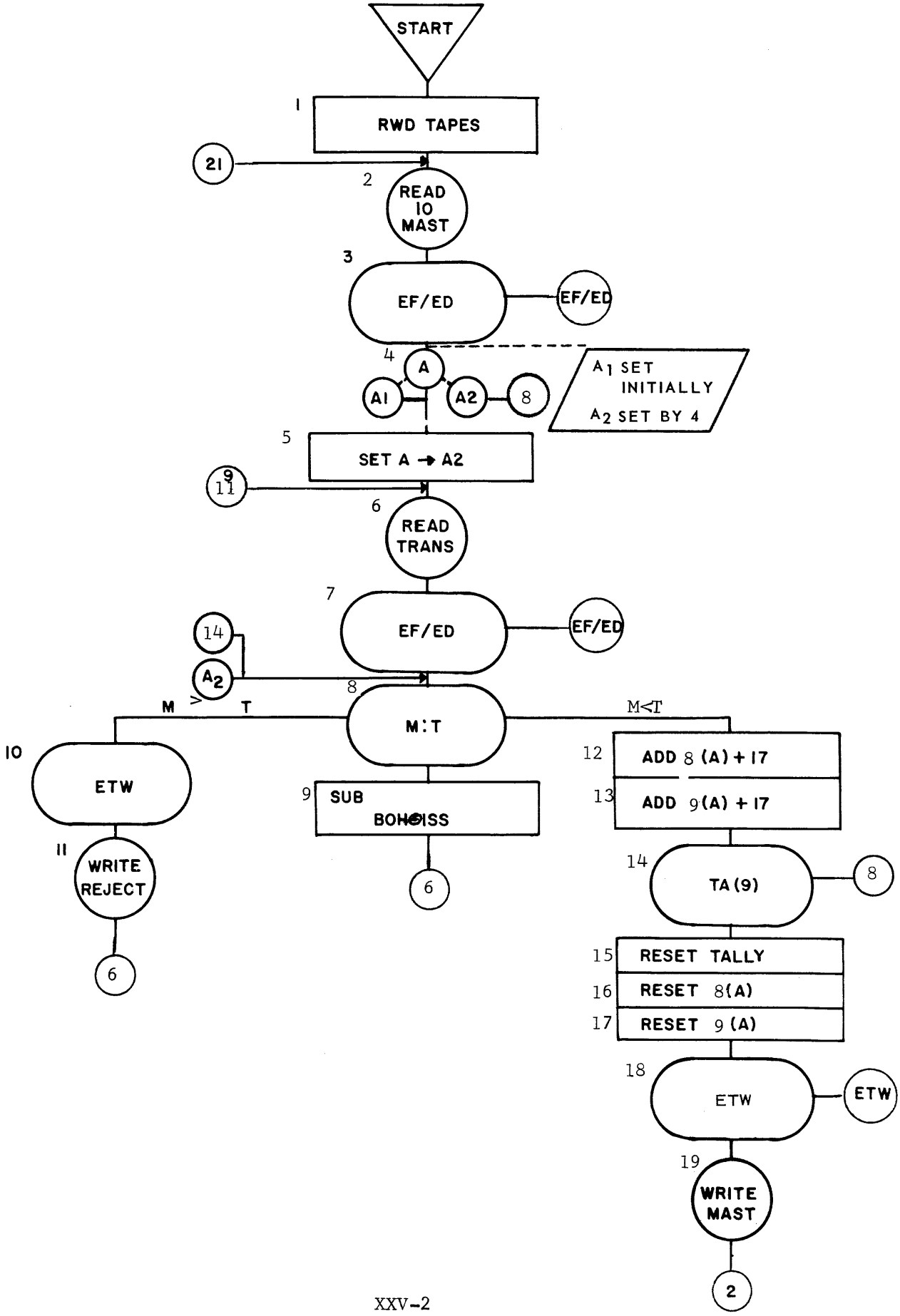
XXV — BATCHING

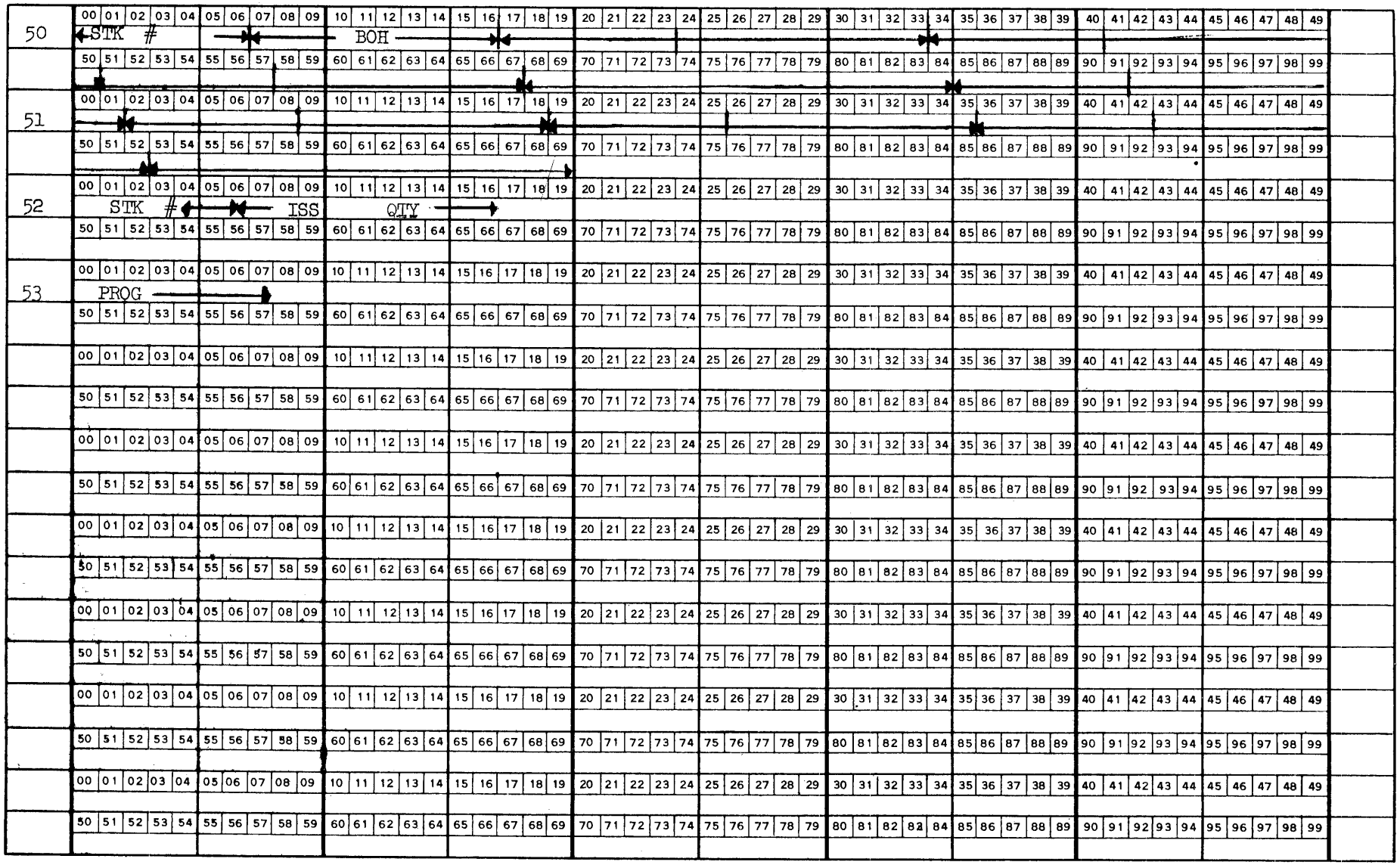
If we were to break down and analyze the timing of the problem illustrated on page XXIV-2, we would discover that only 9 seconds of the time was used for computation and that 11 minutes and 4 seconds was required to read in the information and write out the updated master. Of course, this relationship is due primarily to the fact that this is a very simplified program.

Examining the time even further, however, we can see that the major portion of this time (11 minutes) was needed for start time. Here again, this extreme proportion is due to the fact that the size of the records is unrealistically small. However, it does point up the fact that the gap size and the up-to-speed time does play an important part in timing problems.

Obviously, if we could cut down the number of gaps to be processed, input-output time would decrease significantly. This is where "batching" becomes very important. As we have already mentioned, batching refers to the process of carrying numerous records in one block (a block being all the information between two gaps on tape). Taking the same problem, and batching the master records into groups of ten, but leaving our transactions in a one record per block state, our program would now appear as follows (ignoring the EF routine).

Timing this program, we would find that we have added approximately 3 1/2 seconds (for the Adds, Tally, and resetting Transfers), but we have cut the tape time by 8.4 minutes. The program time is now approximately 6.7 minutes.


START
1
RWD TAPES
21
2
READ 10 MAST
3
EF/ED
EF/ED
4
A
A1
A2
8
A1 SET INITIALLY
A2 SET BY 4
5
SET A → A2
9
11
6
READ TRANS
7
EF/ED
EF/ED
14
A2
8
M > T
M:T
M<T
10
ETW
11
WRITE REJECT
6
9
SUB
6
12
ADD 8 (A) + 17
13
ADD 9(A) + 17
14
TA (9)
8
15
RESET TALLY
16
RESET 8(A)
17
RESET 9 (A)
18
ETW
ETW
19
WRITE MAST
2



TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE___ OF__

FORM NO. 1223 25M 6-60

TITLE **Batching Example**
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	530 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD 1 MAST	1
		1 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD 2 TRANS	1
		2 0	0	0	;	3	0	0	0	0	0	0	0	0		RWD 3 NEW MAST	1
		3 0	0	0	;	4	0	0	0	0	0	0	0	0		RWD 4 REJECT	1
5540		4 0	0	0	4	1	5	0	0	0	5	1	6	9		READ TEN MAST	2
		5 0	0	0	W	8	E	F	/E	D	(5	3	6	0)	4	ED/EF	3-4
5350	6	536 0	0	0	N	4	5	5	5	5	5	3	5	9		SET A→A2	5
5420		7 0	0	0	4	2	5	2	0	0	5	2	1	6		READ TRANS	6
5450 / 5480		8 0	0	0	W	8	E	F/	E	D	5	3	9	0		ED/EF	7
5350		9 0	0	0	Y	7	(5	0	0	0)	5	2	0	0	17,13	COMPARE MAST: TRANS	8
		540 0	0	0	W	1	5	4	3	0	5	4	6	0		M > T→10 M < T→12	9
		1 0	0	0	⊖	&	(5	0	1	6)	5	2	1	6	18,14	SUB BOH - QTY	8
	6	542 0	0	0	V	1	0	2	1	9	5	3	7	0		TRANS 6	9
5400		3 0	0	0	S	4	4	0	0	0	E	T	W			ETW	10
		4 0	0	0	8	4	5	2	0	0	5	2	1	6		WRITE REJECT	11
		5 0	0	0	V	1	0	2	1	9	5	3	7	0		TRANS 6	
5400		6 0	0	0	+	3	5	3	9	5	5	5	5	9		INCREMENT COMPARE	12
		7 0	0	0	+	3	5	4	1	5	5	5	5	9		INCREMENT SUBTRACT	13
	6	548 0	0	0	X	0	5	5	6	5	5	3	9	0		TALLY 9 TIMES → 7	14
		9 0	0	0	N	2	5	5	6	3	5	5	6	5		RESET TALLY CTR	15
		550 0	0	0	N	3	5	5	6	9	5	3	9	5		RESET 7(A) (COMPARE)	16
		1 0	0	0	N	3	5	5	7	5	5	4	1	5		RESET 9(A) (SUBTRACT)	17
		2 0	0	0	S	3	4	0	0	0	E	T	W			ETW	18
		3 0	0	0	8	3	5	0	0	0	5	1	6	9		WRITE MAST	19

XXV-4

TITLE Batching Example
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	554 0	0	0	V	1	0	2	1	9	5	3	4	0		TRANS 2	
		5 0	0	0	0	0	5	3	9	0	0	0	1	7		CONSTANTS: A_2 INC AMT	
		6 0	0	0	0	0	0	9	(0	9)	5	0	0	0		TALLY AMT + CTR INIT ADD	
		7 0	0	0	0	0	5	0	1	6	0	0	0	0		INTI ADD	
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXV-5

9-ΔXX

| | | |

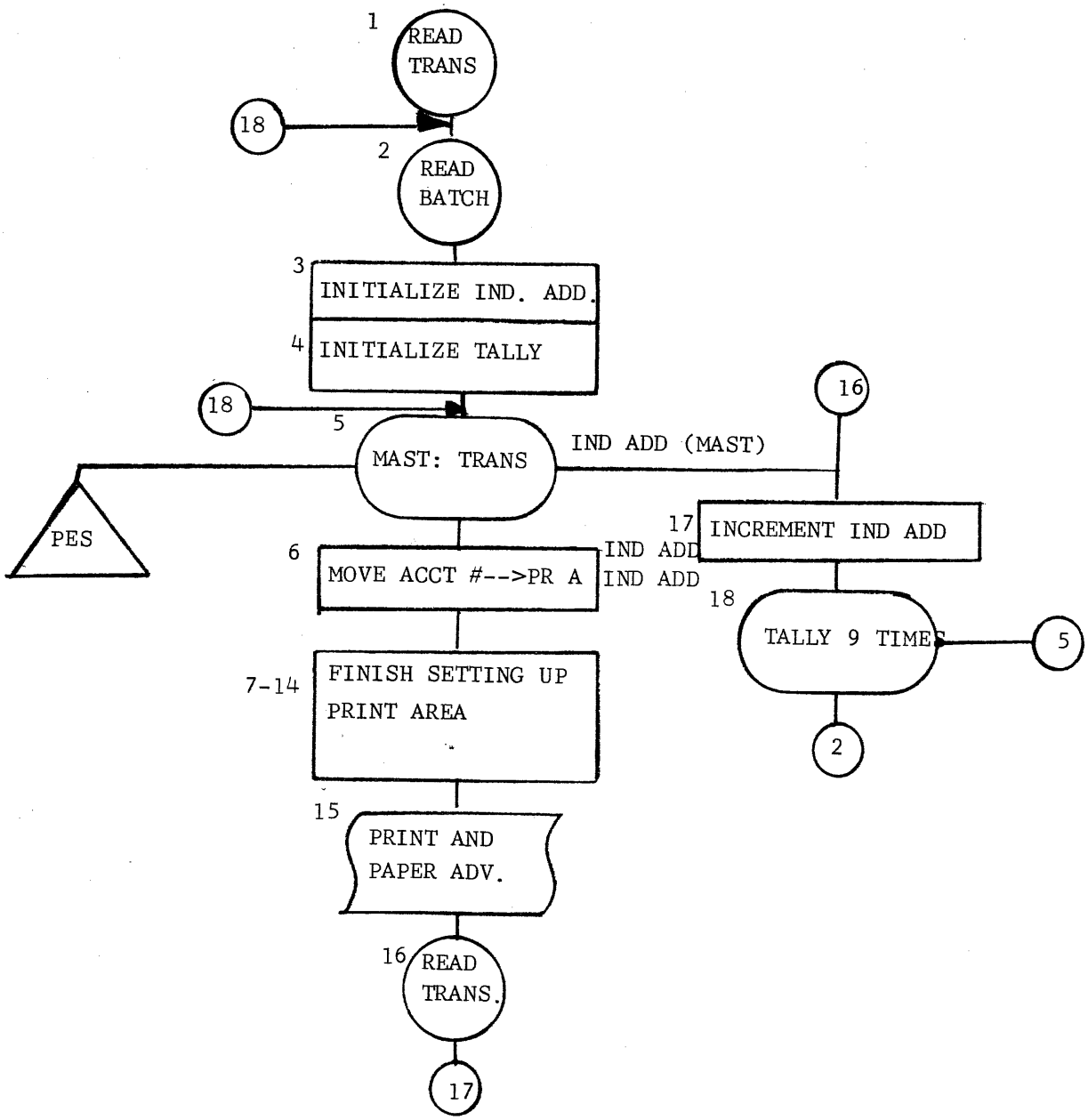
TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
2090	6	200 0	0	0	N	2	2	1	0	1	2	1	0	3		INITIALIZE TALLY	1
		1 0	0	0	N	4	2	1	1	5	0	2	1	5		INITIALIZE STA WITH ADDRESS OF FIRST MASTER	2
		2 0	0	0	N	4	2	1	1	9	2	0	4	9		INITIALIZE LIST (5B) TO RECEIVE SECOND ADDRESS	3
2060		3 0	0	0	4	1	0	2	1	E	6	0	3	9		READ MASTER (USE STA AS INDIRECT ADDRESS FOR A)	4
		4 0	0	0	N	4	0	2	1	5	5	0	0	7		TRANS STA → LIST	5
		5 0	0	0	+	2	2	0	4	9	2	1	0	5		INCREMENT LIST BY 4	6
	6	206 0	0	0	X	0	2	1	0	3	2	0	3	0		TALLY 9 TIMES → 4	7
		7 0	0	0	⊖	4	0	2	1	5	2	1	0	9		SUBTRACT STA ⊖ 1	8
		8 0	0	0	8	3	5	0	0	0	0	2	1	E		WRITE BATCH (USE STA AS INDIRECT ADDRESS FOR B)	9
		9 0	0	0	V	1	0	2	1	9	2	0	0	0		TRANSFER → 1	
		210 0	0	0	0	9	0	9	0	4	0	0	0	1		CONSTANTS	
		1 0	0	0	0	0	5	0	4	0	5	0	0	7			
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

PAGE OF

28-00-004


1
READ TRANS
18
2
READ BATCH
3
INITIALIZE IND. ADD.
4
INITIALIZE TALLY
18
5
MAST: TRANS
IND ADD (MAST)
16
PES
17
INCREMENT IND ADD
6
MOVE ACCT #-->PR A
IND ADD
IND ADD
18
TALLY 9 TIMES
5
2
7-14
FINISH SETTING UP PRINT AREA
15
PRINT AND PAPER ADV.
16
READ TRANS.
17

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

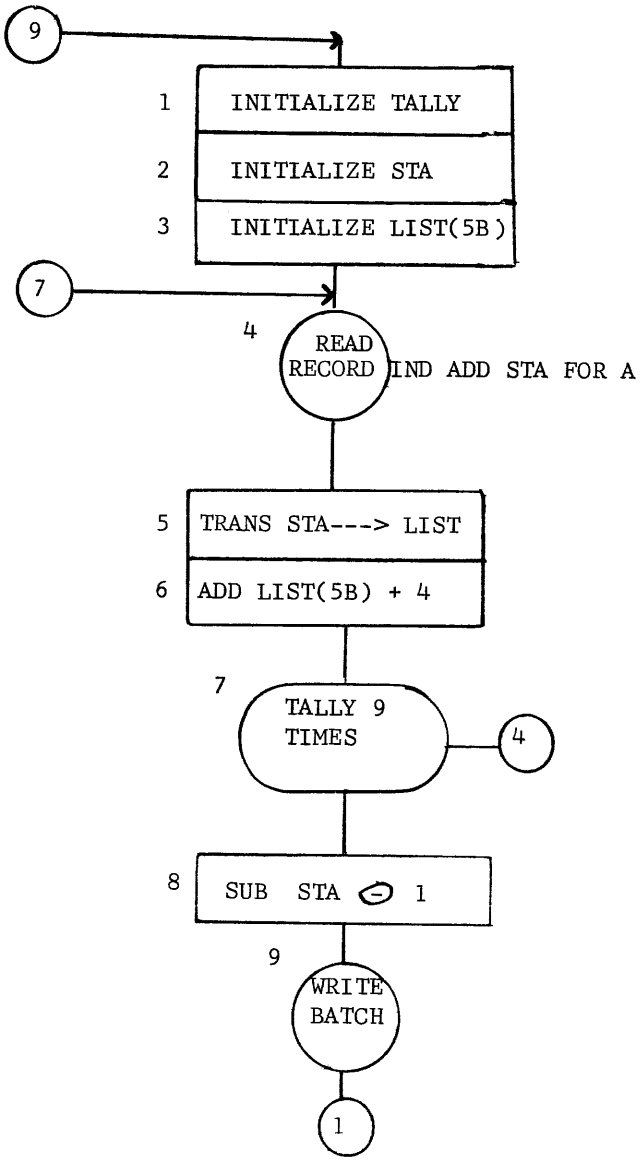
FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	210 0	0	0	4	8	7	0	0	0	7	0	0	4		READ TRANSACTION	1
2290		1 0	0	0	4	3	5	0	0	0	6	0	3	9		READ BATCH	2
		2 0	0	0	N	4	2	3	1	5	2	1	4	5		INITIALIZE INDIRECT ADDRESS (5A)	3
		3 0	0	0	N	2	2	3	1	7	2	3	1	9		INITIALIZE TALLY	4
2280		4 0	0	0	Y	5	[5	0	0	C]	7	0	0	0		COMPARE M:T (MASTER IS IND. ADD. IN LIST)	5
		5 0	0	0	W	1	2	3	0	0	2	2	7	0		PRP → PES PRN → 17	5
	6	216 0	0	0	M	5	2	1	4	E	8	0	0	0		MOVE ACCT # → PRINT AREA (IND ADD 5A)	6
		7 0	0	0	)											PREPARE PRINT LINE	7
		8 0	0	0	)											)	
		9 0	0	0	)											)	
		220 0	0	0	)											)	
		1 0	0	0	)											)	
	6	222 0	0	0	)											)	
		3 0	0	0	)											)	
		4 0	0	0	)											)	
		5 0	0	0	B	0	0	0	0	0	8	0	0	2		PRINT	15
		6 0	0	0	4	8	7	0	0	0	7	0	0	4		READ NEXT TRANSACTION	16
2150		7 0	0	0	+	2	2	1	4	5	2	3	1	1		INCREMENT LIST IND. ADD. + 4	17
	6	228 0	0	0	X	0	2	3	1	9	2	1	4	0		TALLY 9 TIMES → 5	18
		9 0	0	0	V	1	0	2	1	9	2	1	1	0		TRANSFER → 2	
2150		230 0	0	0	[@	0	0	0	0	0	0	0	0	0		PES	
		1 0	0	0	0	4	5	0	0	C	0	9	[0	9]		CONSTANTS	
		0	0	0													
		0	0	0													

PAGE OF

28-00-004

To Illustrate tow more concepts which can be applied to indirect addressing, let·s assume that we have a reel of master information maintained in batched-variable fromat. Our batch size is 10 records, and the maximum size record is 100 characters.

The problem with which we are concerned deals with the printing of the information in particular records specified by paper tape input. In order to do this, it will be necessary to obtain the address of the first character in each record. When initially batching the information, it would be possible to develop a list of addresses of the left hand end of each record: (we will eliminate the ED/EF and ETW senses)



When we must program our print run, it will be quite easy to reach each record. We will read in each batch, and using the first address in the list of addresses as an indirect address, we will be able to locate the 5 digit account number of the first master. If we don't find a match between the transaction account number and the account number of the first master in the batch, it is possible to increment the list's indirect address by 4 to obtain the indirect address of the second master. This is due to the fact that it is possible to add (or subtract) a numeric from another numeric, the least significant digit of which has a zone bit in 2^4 (indicating an indirect address). The numeric portions are added together (or subtracted), and the zone bit is inserted in the LSD of the sum (or difference):

500C + 0004 =
5003 (with a zone bit in 2^4 of the 3) + 0004 =
5007 (with a zone bit in 2^4 of the 7) =
500G

When we finally find a match, the A address of the Compare instruction holds the indirect address of the correct master. By using this A address of the Compare instruction as an indirect address, we have created a multi-level indirect address, which will result in the placing of the address of the MSD of the proper account number into the A register before the Compare is executed:

```
                00 01 02 03 04 05 06 07 etc.
LIST:   50       5  0  4  0  5  1  2  3

MASTER READ IN AREA:

                40 41 42 43 44 45 etc.
        50       1  2  3  4  5

                23 24 25 26 27 28 etc.
        51       1  3  6  4  2
```

PROGRAM:

```
        2140  Y 5 500G 7000   Compare Master: Transaction
        2150  W 1 2300 2270
        2160  M 5 214E 8000   Move Master Acct.# ---> Print Area
```

REGISTERS:

```
        OP  N     A      B
         M  5   214E   8000
```

214E in the A register indicated indirect address so replaced by contents of 2142-2145, which in our example are 500G:

```
        OP   N      A      B
         M   5    500G   8000
```

500G in the A register indicates indirect address so replaced by contents of 5004-5007, which in our example are 5123:

```
        OP   N      A      B
         M   5    5123   8000
```

The instruction will now begin to execute.

XXVI – SIMULTANEITY

It is obvious by this time that one of the prime requisites of a good program is that it must be efficient with regard to time and space (although compromises must be made depending upon which is more important for a particular application).

Timewise, a program can be written to be accomplished in a minimum amount of time, but once that minimum is reached even the best programmer in the world can not improve on it as long as he is forced to process the data one step at a time. The RCA 301 system allows the customer to better this minimum time, if he desires, by renting or purchasing an additional logic device, called the Simultaneous Program Control Unit. It is the utilization of this "mode" which is our next topic of investigation.

We have already seen that computers are input-output bound; i.e., that to read a character into memory (or write it out from memory) takes a much longer time than any type of processing on that character once it is in memory. In an effort to handle this situation, all the input-output devices are "buffered". A buffer is an intermediary device which assures the speed compatability between two units. For example, we have already determined that it takes 100 us. to read one character from 10 kc magnetic tape. We also know that it only takes 7 us. (one cycle) to place a character in memory. To make the two time cycles compatible, the character is brought from tape into a one character buffer, and from there it is transmitted to the Memory Register and then into memory.

Analyzing this time, it is obvious that the computer itself is busy only 7 us. of the 100 us. time, and that the remaining 93 us. of the time it is waiting for the next character. This same sort of relationship is true with each type of input-output device as the following table shows:

DEVICE	INSTRUCTIONS MUST ACCESS MEMORY FOR	OUT OF EVERY
Card Reading Equipment		
Card Reader	13.44ms	100ms
Card Reader/Punch	13.44ms	75ms
Card Punching Equipment		
Card Punch - 100 cpm	6.72ms	600ms
200 cpm	13.44ms	300ms
Card Reader/Punch	6.72ms	240ms
Paper Tape Read or Punch (100cps)	7us	10ms
Paper Tape Reader (1000cps)	7us	1ms

Cont·d. on following page

DEVICE	INSTRUCTIONS MUST ACCESS MEMORY FOR	OUT OF EVERY
Printer 120 col: Synchronous Mode	21ms	60ms
Asynchronous Mode	28ms	76ms
160 col:		
Synchronous Mode (full speed)	27ms	55.8ms
Asynchronous Mode (full speed)	37ms	71.8ms
Synchronous Mode (Reduced speed)	27ms	83.9ms
Asynchronous Mode (Reduced speed)	37ms	100ms
Monitor Printer	7us	100ms
Data Record File	7us	400us
Data Disc File	7us	62.5us
Hi-Data Magnetic Tape Read-Write	7us	100us
33kc Magnetic Tape Read-Write	7us	30us
66kc Magnetic Tape Read-Write	7us	30us

Having proved that the time is available in which to do another simultaneous operation, we must discover how the computer has the ability to keep track of two instructions operating at the same time.

With the simultaneous program logic, the computer is equipped with a second set of registers. This second set of registers is the counterpart of the OP, N, A and B registers which keep track of the instruction being executed in the Normal Mode. These registers, (SOR(Simultaneous Operation Register), M, S, and T) keep track of the instruction being executed in the Simultaneous Mode. Entry to these registers is through the Normal Registers. That is, a Simultaneous instruction is staticized in the Normal Mode and if the Simultaneous Mode is unoccupied, it will drop into the Simultaneous Registers before it begins to execute, leaving the Normal Mode free to receive the next instruction. If the Simultaneous Mode is occupied, the Simultaneous instruction simply waits (does not execute) in the Normal Mode until the Simultaneous registers can accept it. During a situation like this the only thing that would be executing would be the instruction already in the Simultaneous Mode.

In the RCA 301 System, then the HSM is time-shared; i.e., during the time that an input-output instruction is in progress, but is not using memory, another input-output or compute instruction can use the memory.

If an input-output operation is being executed in each mode, there can be a direct overlap; i.e., a 100ms. card read can be completely buried in a 100 ms. tape read. This is accomplished due to sequenced servicing of the input-output buffers. If, however, an input-output instruction is being executed in the Simultaneous Mode while a compute instruction is in process in the Normal Mode, the computation will have to be "interrupted" by the simultaneous operation in order to transfer the characters from the buffer into memory (or from memory to the buffer). This is obvious, since computation instructions (transfers, adds, compares, etc.) require the constant use of the Memory Register. Timewise, this means that if we were doing a 100 ms. tape write in the Simultaneous Mode, we would have 94.74 ms available for computation in the Normal Mode.

The following matrix chart shows some operations that can be done simultaneously:

SIMULTANEOUS OPERATION:	MAGNETIC TAPE:				PAPER TAPE:		CARDS:		PRINTER:
	READ	WRITE	READ (33)	WRITE (33)	READ	PUNCH	READ	PUNCH	PRINT
NORMAL OPERATION:	(10)	(10)	(66)	(66)					
MAGNETIC TAPE: READ (10)	*	*	Y	Y	Y	Y	Y	Y	Y
WRITE (10)	*	*	Y	Y	Y	Y	Y	Y	Y
READ (33) (66)	Y	Y	%	%	Y	Y	Y	Y	Y
WRITE (33) (66)	Y	Y	%	%	Y	Y	Y	Y	Y
PAPER TAPE READ	Y	Y	Y	Y	N	%	Y	Y	Y
PUNCH	Y	Y	Y	Y	%	N	Y	Y	Y
CARDS READ	Y	Y	Y	Y	Y	Y	N	Y	Y
PUNCH	Y	Y	Y	Y	Y	Y	Y	N	Y
PRINTER PRINT	Y	Y	Y	Y	Y	Y	Y	Y	@
COMPUTATION	Y	Y	Y	Y	Y	Y	Y	Y	Y

Cont•d.

Where:

Y means yes, the two operations can be done simultaneously.
N means no, the two operations can not be done simultaneously since they would both be addressing the same device.
% means yes, if 2 tapes are available and a different one is being used by each mode.
* means yes, if both tape clusters are available, and one operation is being executed by a tape deck in one cluster, while the other operation is being executed by a tape deck in the other cluster.
@ means yes, if both Printers are available, and a different one is being used by each Mode.
% means yes, if both a Paper Tape Reader and a Paper Tape Reader/Punch were included in the system.

As the chart shows, the only limitation is one of device ability. Obviously we couldn·t print in both the Normal and the Simultaneous modes at one time if we only have one Printer.

To do an instruction in the Simultaneous Mode, it must be a Simultaneous instruction. Each output-input instruction we have had to date has a simultaneous counterpart:

TAPE READ FORWARD SIMULTANEOUS:

The OP code is 5.

N contains the identification character of the input device (1-6, A-F, J or N, L or P, 8 or U)

The A address is the HSM location which is to receive the first character on tape.

The B address is the HSM location which is to receive the last character read from tape.

The instruction is executed in the simultaneous registers, which means that S_f will be the address of the location one to the right of the last character read into HSM and T_f will be the same as T_i (B_i).

As in the case in the Tape Read Forward Normal, the operation will terminate when the gap following the block is sensed. If S should equal T before this gap is located, the character will be read but not placed in memory. The PRI S are NOT affected as they are used in conjuction with the Normal Mode. An ED or EF alone is accepted as a legitimate one character block and will set a Simultaneous ED/EF indicator which can be sensed using the CTC instruction with a - (minus) in the N character.

TAPE WRITE SIMULTANEOUS

The OP code is 9.
N identifies the output device
The A address gives the location of the first character to be written, punched or typed.

The B address gives the address of the last character to be written, punched or typed.

The operation terminates when the entire sector has been written and at that point S_f holds the address one to the right of the last character written.

Since there are two models of Card Reader and two models of Card Punch in the RCA 301 system, we shall discuss the simultaneous card instructions for the Card Reader (600CPM) and Card Punch (100CPM) first, and the instructions for the Card Reader/Punch (800CPM read and 250CPM punch) second.

CARD READ SIMULTANEOUS (for 600CPM Reader)

The OP code is 1

N -

N	RESULT
1	Reads single card or terminates (600CPM) continuous card reading
2	Used in continuous card ending routine (600CPM)
4	Used in continuous card reading cycle (600CPM)
M	Used in alternate card reading cycle (300CPM)
8	Used in alternate card ending routine (300CPM)

The A address gives the HSM location to receive the first character read from punched cards.

The B address is zeros (0000).

S_f holds the address one to the right of the location of the character last rea in.

CARD PUNCH SIMULTANEOUS (for 100CPM Punch)

The OP code is 1
N is zero (0)

The A address gives the HSM location of the first character to be punched.

The B address gives the HSM location of the last character to be punched.

The operation terminates when the information in the sector designated has been punched to cards. This allows for multiple card punching with one instruction. S_f is the HSM address one to the right of the last character punched.

CARD READ SIMULTANEOUS (for 800CPM Reader/Punch)

The OP code is 1

N is

SYMBOL	RESULT
K	Binary read mode specified
1	Translate read mode specified

The A address gives the HSM location to receive the first character read from punched cards.

The B address is zeros (0000).

S_f contains the address one location to the right of the last character read into HSM.

CARD PUNCH SIMULTANEOUS (250CPM Reader/Punch)

The OP code is 3.

N is

N	CARD PUNCH MODE
&	Binary
0(zero)	Translate

The A address gives the HSM location of the first character to be punched.

The B address gives the HSM location of the last character to be punched.

S_f is the address of the location one to the right of the last character punched.

PRINT AND PAPER ADVANCE SIMULTANEOUS:

The OP code is C.

N is the same as in the normal mode.

The A address is zeros (0000).

The B address gives in $B_{0, 1, 2,}$ the address of the left hand end of the print area and in B_3 the type of paper advance:

Type of Paper Advance	B_3
No paper advance	0
Paper advnace using N register as count	1
Vertical Tab channel control	2
Page Change channel control	3

In addition to these Simultaneous instructions, there are several Normal instructions that are used in conjunction with the Simultaneous Mode.

CONDITIONAL TRANSFER OF CONTROL:

The CTC allows us to sense two different states that relate to the Simultaneous Mode. One is the sense of the Simultaneous ED/EF indicator. To do this a minus (⊖) must be placed in the N character. The A address gives the location of the next instruction which must be executed if the indicator has been set; the B address, the location if it has not been set.

The second test that can be made is of the type of instruction being executed in the Simultaneous Mode. This is done by placing a 4 in the N position. The A address gives the address of the next instruction to be executed if there is an Input instruction being executed in the Simultaneous Mode. The B address gives the location of the next instruction to be executed if an output instruction is being executed in the Simultaneous Mode. If the Simultaneous mode is unoccupied, no transfer will take place and the next instruction in sequence will be the one to be performed.

STORE REGISTER:

We discovered that the final contents of the A register (placed for us in most cases in STA) were important to us in many instances. The same will be true of the final contents of the S register, so that we must make sure that the Simultaneous instruction has been completed before we store the contents of the S register with a STORE REGISTER instruction. To do this, we must place an 8 in the N character. The A address gives the right hand end of two diad sector which is to receive the address in the S register. The B address is zeros. An illustration of this will be shown later in this chapter.

INPUT-OUTPUT SENSE:

Another way of keeping track of what is going on in the Simultaneous Mode is to sense the device being utilized by the instruction which is being executed in the Simultaneous Mode. This can be done with an INPUT-OUTPUT SENSE instruction. The interrogations that would be utilized in this manner would include:

1. magnetic tapes (1-6 or A-F for 10kc; J or N for 33kc; L or P for 66kc)
   a. is tape now in motion (A_0 is 2)
2. paper tape reader or punch (8 and 9 respectively)
   a. is the selected device operating (A_0 is 2)
3. card reader or punch (card reader (, and) is the card punch)
   a. is the selected device operating (A_0 is 2)
4. printers (7 or G)
   a. is a line being printed (A_0 is 2)
   b. is the paper advancing (A_0 is &)

Having obtained a basic understanding of Simultaneity, we should apply what we have learned to a number of example problems. If we remember one rule, we will never have any difficulty when laying out a program with simultaneity. The rule is never to do processing in the same area at the same time in both modes. For example, it would be disastrous if we started to read in the Simultaneous mode and immediately began to process on that data in the Normal Mode. Since we already know that it takes 9.8ms. for the tape to get up to speed, it is quite obvious that we would not be processing the new data that we are planning to read in but rather the old data that was in the area. In most cases, all "no" answers to the following tests will prevent any difficulties:

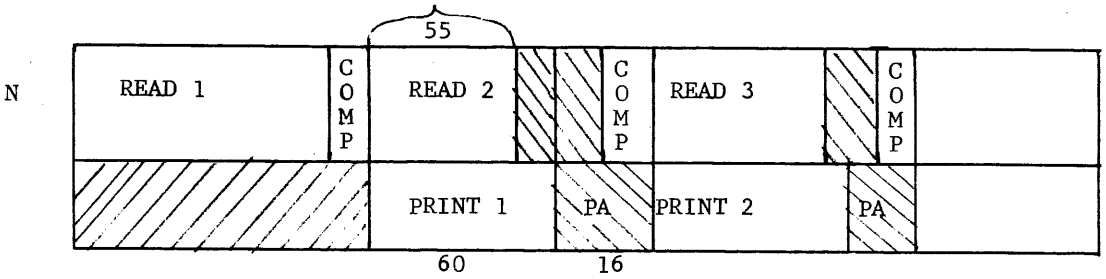
1) Are we processing data that we haven't brought in?

2) Are we putting out data that we haven't finished processing?

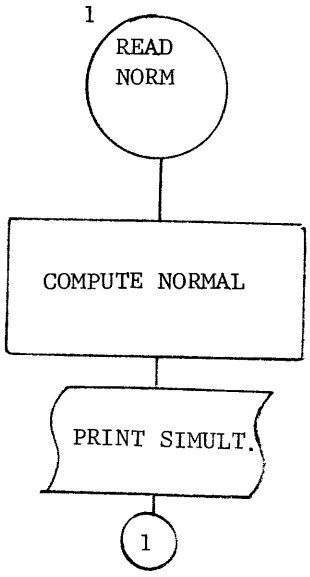
3) Are we reading in new data to an area before we are finished processing the old data?

Example I:

We have 10,000 records to bring in, compute, and print. The approximate read time per record is 55ms.;the compute time is 5ms.; and print is 76ms. Timing this program without using simultaneity, our time is approximately 23 minutes. Suppose, however, we overlap the Read and the Print as follows:



REPRESENTS UNUSED TIME



Timewise, we can see that we have processed one record completely (Read, Compute, and Print) under the bracket. We know that the Print takes 60 ms. and since the Read takes only 55 ms. we can bury it completely in the Print. We will not want to start computation, however, until the Print is entirely done, since the computation is primarily setting up for the next print. However, there is 16 ms. of paper advance time which must be done before the next print can be accomplished, therefore the compute time is actually buried in this. Our time now is only the print time of 76 ms. per record or a total of 12 and 2/3 minutes, a savings of approximately 10 minutes.

Checking our rules, we can see that we can't compute until we have finished reading simply because of the sequence of the Normal instructions. We can't Print before we have finished computing, again due to instruction sequence. We don't care if we Read in a new record while we are in two different areas. We do, however, run into a problem due to the fact that we have the possibility of starting to set up the new print area before we finish printing out the old line. To prevent this, we can incorporate one of two senses. One way is to determine if an output instruction (Print) is being done in the Simultaneous Mode and if it is we can transfer to the inquiring instruction:

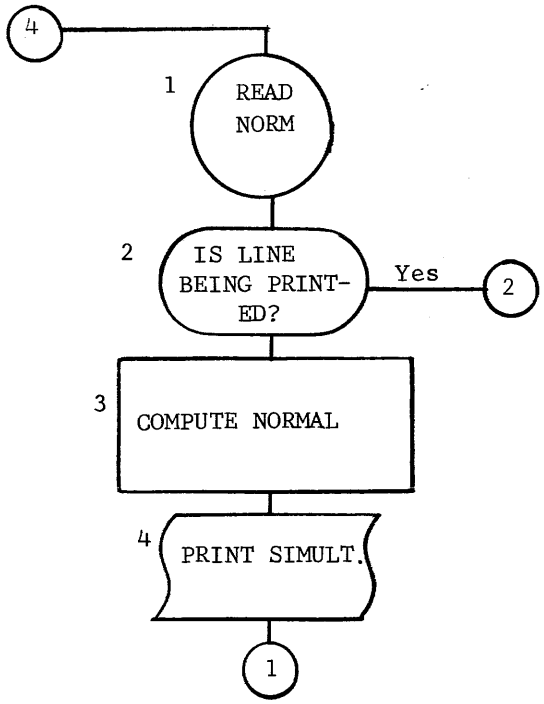
```
1760 W  4  1830  1760
1770 start the computation
1830 ⊡  0 0000 0000  There should not be an input instruction going on in
                     the Simultaneous Mode, as none have been staticized.
```

This sets up a program loop that will not be ended until the Print is finished.

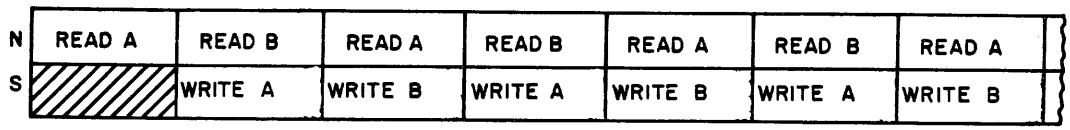
Another way to handle it is to inquire of the Printer whether or not a line is being printed. If the answer is yes, we will again set up a loop. If the answer is no, either the paper advance has begun or the entire instruction has been completed, and in either case we are safe to start setting up the next print line.

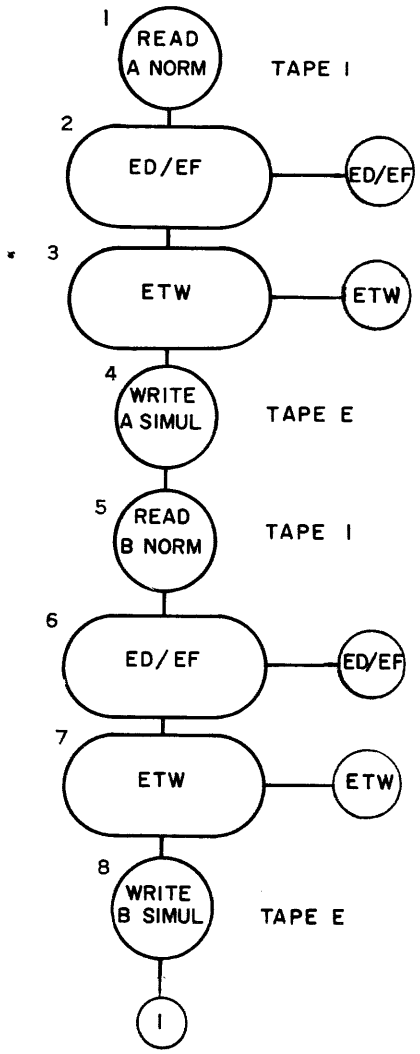
```
1760 S 7 2000 1760
1770 start computation instructions
```

Flow-charted the program now appears as follows:



Taking another example suppose we wished to duplicate a file of 10,000 records. Assuming that we have 2 Hi Data Tape Groups and that we read from one group and write to the other, our read time (and write time) is 70 ms. per record (600 character records), and our compute time is not worth including. Processing this in the Normal Mode would take 23.3 minutes. If we could overlap the reads and the writes, however, we could cut our time almost in half. To this we will have to utilize dual areas, that is to say, we will set up two read in-write out areas instead of one. While we are reading into A, we could be writing out from B; then while we read into B, we could be writing out from A; etc.





Timewise, we can overlap the reads and writes completely. Our total time for processing 1 record, then will be 70 ms. The total processing time will be 11.7 minutes.

No safeguards are needed since the simultaneous operations act as a guard on each other; i.e., it would be impossible to read into A before the write of A is completed since it is impossible to start the write of B (which is Simultaneous) until the write of A has been completed in the Simultaneous Mode. The write of B, therefore, holds up the read of a new A until the old A has been written out. Coded, it would appear as follows:

TITLE Simultaneity Example 1
CODER
REMARKS

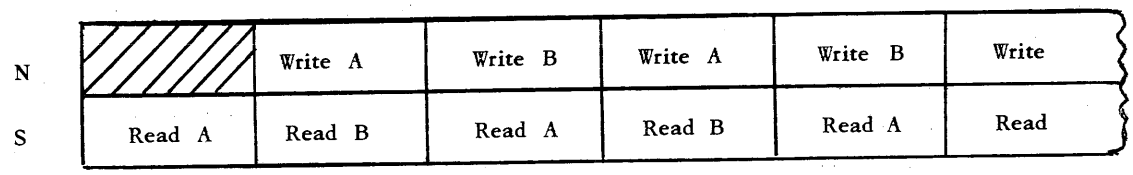
DATE
SEGMENT NO.

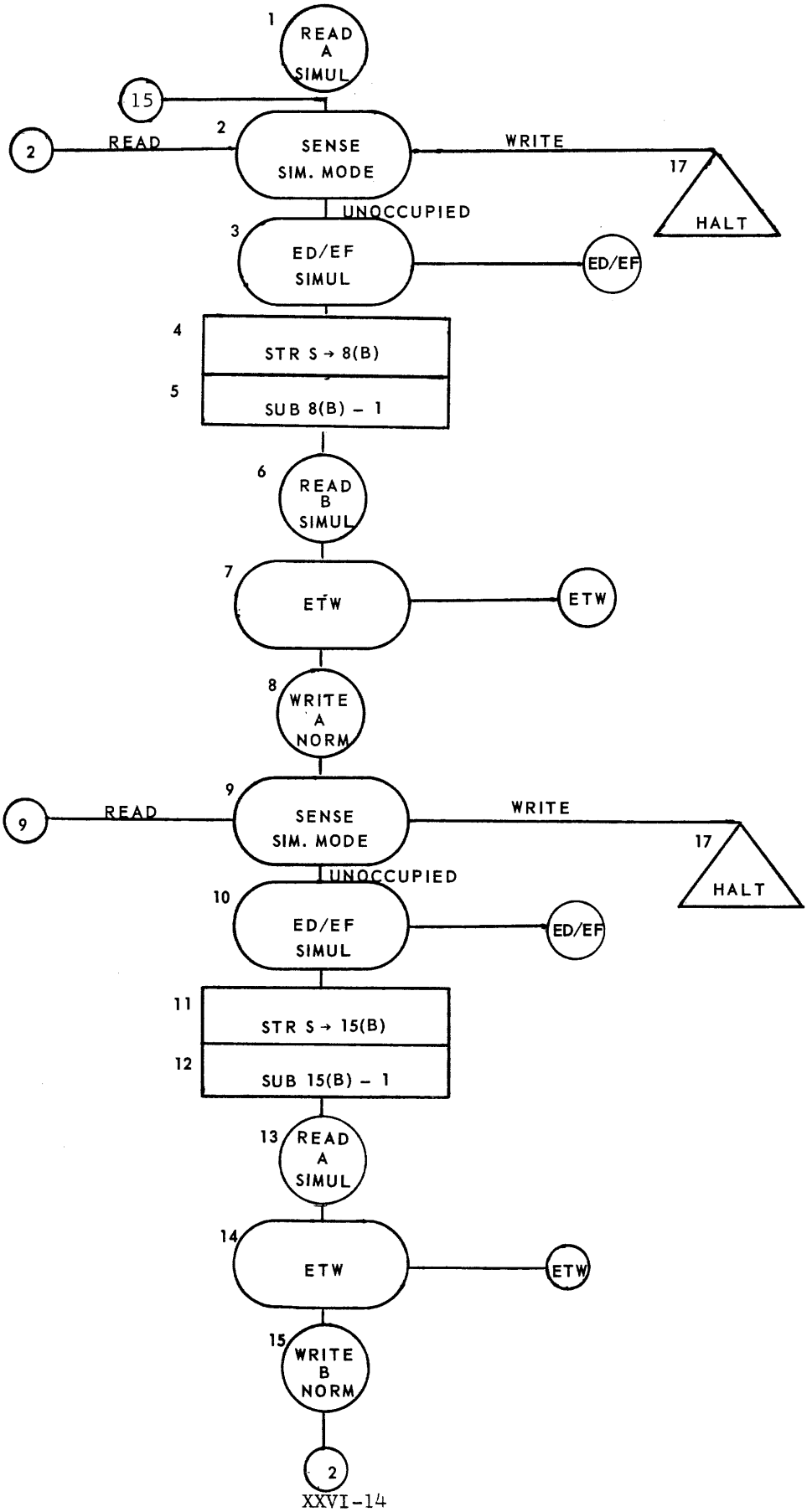
FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	100 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD INPUT	
		1 0	0	0	;	A	0	0	0	0	0	0	0	0		RWD OUTPUT	
1100		2 0	0	0	4	1	5	0	0	0	5	5	9	9		READ A	1
		3 0	0	0	W	8	E	D/	E	F	1	0	4	0		ED/EF?	2
		4 0	0	0	S	A	4	0	0	0	E	T	W			ETW?	3
		1 S 0	0	0	9	A	5	0	0	0	5	5	9	9		WRITE A SIMULTANEOUS	4
	6	106 0	0	0	4	1	6	0	0	0	6	5	9	9		READ B	5
		7 0	0	0	W	8	E	D/	E	F	1	0	8	0		ED/EF?	6
		8 0	0	0	S	A	4	0	0	0	E	T	W			ETW?	7
		9 0	0	0	9	A	6	0	0	0	6	5	9	9		WRITE B SIMULTANEOUS	8
		110 0	0	0	V	1	0	2	1	9	1	0	2	0		TRANS 1	
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

IE 1224 REV. 8-60

Suppose the file had been of variable length and we had done the Reads in the Simultaneous Mode. We would have to modify the different write instructions before executing them, in order that they might contain the proper righthand limit. To do this, we must Store S after the read is completed.

Bar graphed, flow charted, and coded, the program would appear as follows:




1
READ A SIMUL
15
2
READ
2
SENSE SIM. MODE
WRITE
17
HALT
UNOCCUPIED
3
ED/EF SIMUL
ED/EF
4
STR S → 8(B)
5
SUB 8(B) – 1
6
READ B SIMUL
7
ETW
ETW
8
WRITE A NORM
9
READ
9
SENSE SIM. MODE
WRITE
17
HALT
UNOCCUPIED
10
ED/EF SIMUL
ED/EF
11
STR S → 15(B)
12
SUB 15(B) – 1
13
READ A SIMUL
14
ETW
ETW
15
WRITE B NORM
2

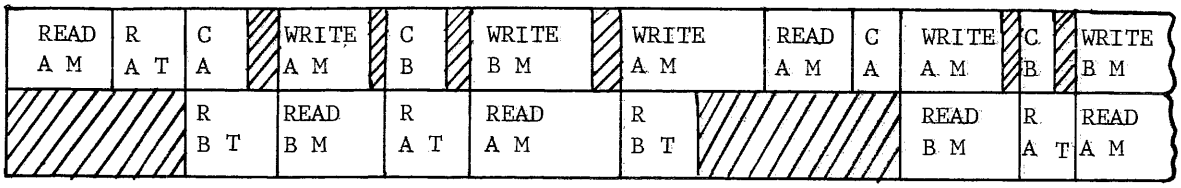
TITLE Simultaneity Example 2
CODER
REMARKS

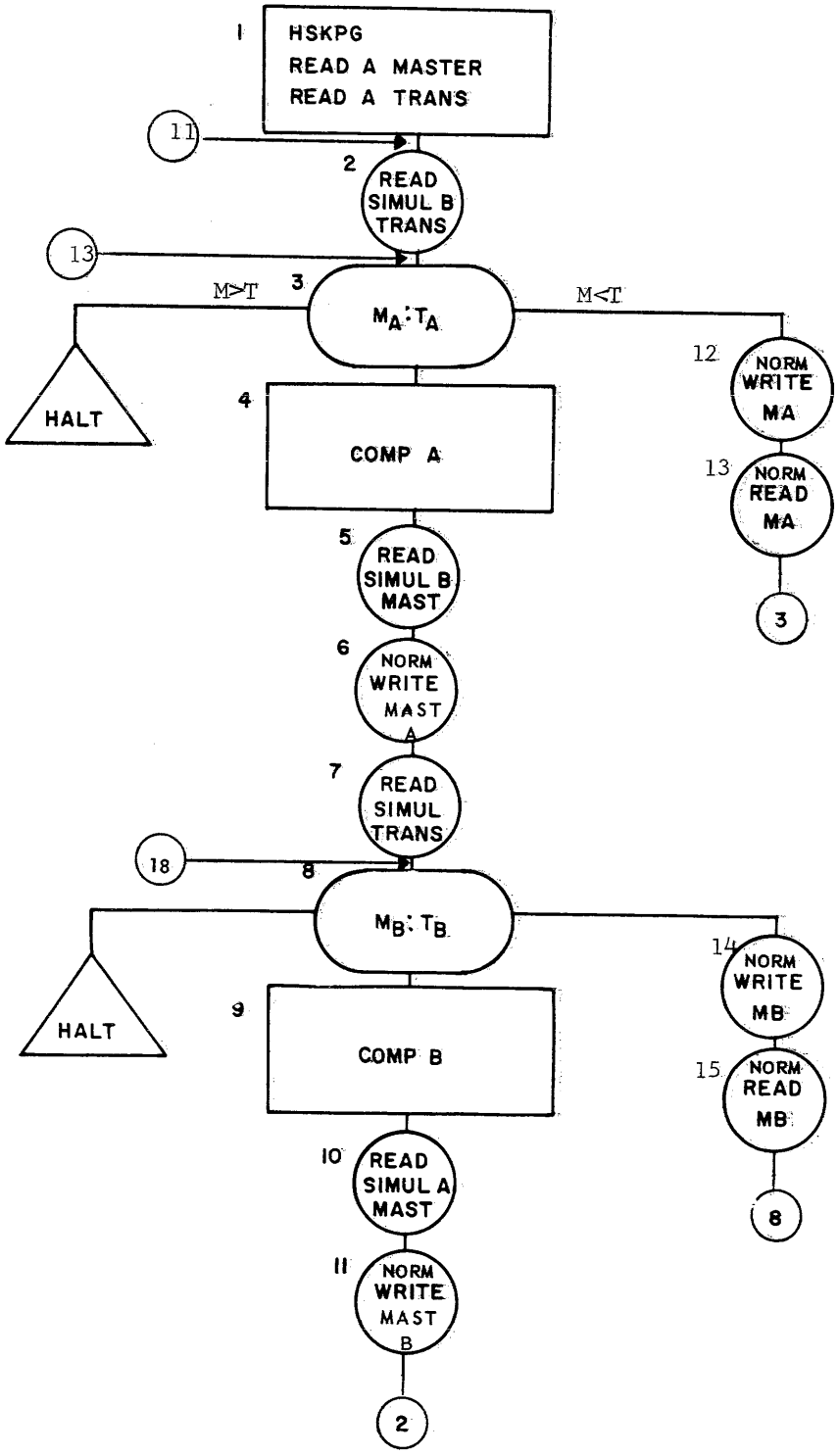
DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	100 0	0	0	;	1	0	0	0	0	0	0	0	0		RWD INPUT	
		1 0	0	0	;	A	0	0	0	0	0	0	0	0		RWD OUTPUT	
		2 0	0	0	5	1	5	0	0	0	5	5	9	9		READ A SIMULTANEOUSLY	1
1170 1030		3 0	0	0	W	4	1	0	3	0	1	1	8	0		SENSE SIMO MODE READ → 2; WRITE → HALT	2
		4 0	0	0	W	⊖	E	D/	E	F	1	0	5	0		ED/EF SIMUL?	3
		5 0	0	0	V	8	1	0	9	9	0	0	0	0		STR S → 8 (B)	4
	6	106 0	0	0	⊖	3	1	0	9	9	1	1	9	9		SUB 8 (B) ⊖1	5
		7 0	0	0	5	1	6	0	0	0	6	5	9	9		READ B SIMULTANEOUSLY	6
		8 0	0	0	S	A	4	0	0	0	E	T	W			ETW?	7
		9 0	0	0	8	A	5	0	0	0	(0	0	0	0)	4,5	WRITE A NORMAL	8
1110		110 0	0	0	W	4	1	1	0	0	1	1	8	0		SENSE SIMO. MODE READ → 9; WRITE → HALT	9
		1 0	0	0	W	⊖	E	D/	E	F	1	1	2	0		ED/EF SIMUL?	10
	6	112 0	0	0	V	8	1	1	6	9	0	0	0	0		STR S → 15 (B)	11
		3 0	0	0	⊖	3	1	1	6	9	1	1	9	9		SUB 15 (B) ⊖1	12
		4 0	0	0	5	1	5	0	0	0	5	5	9	9		READ A SIMULTANEOUSLY	13
		5 0	0	0	S	A	4	0	0	0	E	T	W			ETW?	14
		6 0	0	0	8	A	6	0	0	0	(0	0	0	0)	11,12	WRITE B NORMAL	15
		7 0	0	0	V	1	0	2	1	9	1	0	3	0		TRANS 2	
1110 1030	2	118 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT (ERROR)	17
		9 0	0	0	⊡	0	0	0	0	0	0	0	0	1		CONSTANTS	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

All these problems deal with single files, but we have already discovered that most computer programs are of an updating nature, which means we will have two (or more) files to process. Suppose we had this problem to solve with SIMULTANEITY:

At the end of the month, we are going to post all payments against our Accounts Payable file. There are 1,000 masters in the Accounts Payable file containing an average 200 characters. There are 990 payments being made, and the data in this transaction consists of an 8 character Account Payable code and a 7 character payment (fixed). Compute time is approximately 15 ms. per updated master. Flow charted our program would look as follows: (Assume HALT for error out of sort transactions, and that both clusters are available. The Master Input is on tape deck 1, the Transaction Input on 3, and the Master Output on A. Assume 1 Transaction per Master maximum).


READ A M
R A T
C A
WRITE A M
C B
WRITE B M
WRITE A M
READ A M
C A
WRITE A M
C B
WRITE B M
R B T
READ B M
R A T
READ A M
R B T
READ B M
R A T
READ A M


1
HSKPG
READ A MASTER
READ A TRANS
11
2
READ SIMUL B TRANS
13
M>T
3
$M_A : T_A$
M<T
HALT
4
COMP A
12
NORM WRITE MA
13
NORM READ MA
3
5
READ SIMUL B MAST
6
NORM WRITE MAST A
7
READ SIMUL TRANS
18
8
$M_B : T_B$
HALT
9
COMP B
14
NORM WRITE MB
15
NORM READ MB
8
10
READ SIMUL A MAST
11
NORM WRITE MAST B
2

As we can see, the program is completely safeguarded because of the Reads. Since only 1% of the masters will not have transactions, we have arranged to write out the non-updated masters from their read in areas and replace them with the next master in the file. This will mean that we can always be posting a transaction in the A area against a master in the A area, and a B transaction against a B master. This will save us programming and in this case it is advisable due to the low hit on the master without transaction path.

Timewise, we can figure out the approximate time it would take to complete this program in the Normal Mode:

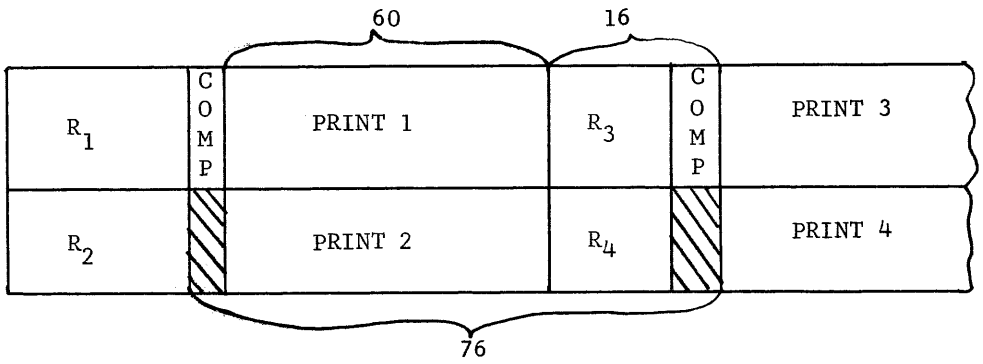
Read of a Master:	39.8 ms. times 1000 records	39.8 sec.
Write of a Master:	56.3 ms. times 1000 records	56.3 sec.
Read of a trans:	22.3 ms. times 990 records	22.1 sec.
Computation	15 ms. times 990 updatings	14.9 sec.
		133.1 sec.

or 2 minutes 13 seconds.

Timing with Simultaneity, we can overlap the Reads and Write of 990 of the masters. The Read can be completely overlapped. For 10 of the masters we are not overlapping, so that both the read and the write time must be included. The computation is easily taken care of in the read of the transaction, so that we need include only the read time:

56.3 ms x 990	55.7 sec.	
39.8 ms x 10	.4 sec.	
56.3 ms x 10	.56 sec.	
22.3 ms x 990	22.1 sec.	
	78.76	or 1 minute 19 seconds.

Taking another problem, suppose we had 100,000 two hundred character records which were to be read, and printed, and suppose further that we had two 33 kc tapes and 2 printers available. This would mean that we could process half the data on one reader and printer, and the other half on the other reader and printer. Our bar chart would appear as follows:



76 ms. X 50,000 records is approximately 1 hour and 4 minutes. Comparing this with the processing time it would take if we only had 1 tape unit, 1 printer and no simultaneity, we find that we have cut our time in half.

At this point we should examine the principle of "interrupt" more closely. Memory interrupt occurs whenever an input/output device is accessing memory. For example, the 7us. it takes to transfer a character from the buffer to memory in a tape read instruction or the scanning process in a print instruction. Interrupt becomes significant when working in 2 or 3 modes at once since the simultaneous or R.F. Mode instructions will "interrupt" the normal mode when it is ready to access memory. For this reason there will be less available time in the normal mode. For example:

When reading from tape and computing in the normal mode and printing in the simultaneous mode, interrupt will occur. If the printing is at a rate of 1000 lines per minute in the synchronous mode, the print instruction must be executed within 60 ms. The interrupt time is 20.069 ms. (for scanning etc).

```
  44      ms read time
 20.069   interrupt
 24.931   time available in the normal mode.
+ 16      ms P.A. time = a total of 40.931 ms.
```

a tolerance of +3% to ⊖5% should be allowed for the printer so that approximately 38 ms are available for reading and computing.

SIMULTANEITY EXERCISE I:

We have a stack of 1200 cards which we are to read in one at a time, do some computation on and print out a line of summary information. The compute time is 15 ms. per card, and assume single spacing.

Bar graph and flow chart the problem.

SIMULTANEITY EXERCISE II:

We are to process one reel of magnetic tape consisting of 1200 six hundred character records. The computation will take approximately 15 ms. and then a summary line is to be printed (single space).

Bar graph and flow chart the problem.

SIMULTANEITY EXERCISE III:

Referring to the Exercise given on page XVII-7 (ITERATIVE CODING EXERCISE I) bar graph and flow chart this same problem using SIMULTANEITY.

XXVII – EDITING

In the prior chapters, we have assumed that all printing has been done on pre-printed forms that would have appropriately placed dollar signs and lines to separate dollars from cents and thousands from hundreds. The only thing that we were responsible for was suppressing any insignificant zeros and right justifying the amounts so that the columns would be properly aligned.

Suppose, however, we were given a program to write in which we were responsible for inserting the dollar sign, decimal point, and commas, in addition to the above editing. How would we handle this?

Let's take a sample application. Suppose that we are to print bank statements and that one entry is the balance in the account, which we will limit to a maximum of 8 characters. We are to print out this field with all the necessary accounting symbols, with no insignificant zeros and with the proper sign. For this problem, we will assume that the dollar sign will be in a fixed location on each statement.

Initially, in housekeeping, we should place a decimal point and a dollar sign in the edit field (the print area), so that it would appear as follows:

				comma, if needed				decimal point			sign
$											
1	2	3	4	5	6	7	8	9	10	11	12

Note that the edit field will require 12 positions: 8 for the maximum amount, 1 for a dollar sign, 1 for a decimal point, 1 for a comma (if needed) and 1 for a sign position.

The first step in the editing portion of the program would be to clear from position 2 through position 5 to spaces. It would not be necessary to clear the cents position since this will always appear on every statement, even if it is only ".00".

Then, in the original data area, we must find the rightmost insignificant zero. In order to safeguard the program against the possibility of a maximum field, we will locate the lack of a zero up to, but not including, the cents positions (assume a dummy position immediately to the left of the data field). In this way we will end up with an address in STA which will be the location of the rightmost insignificant zero (if there are any present) or the location immediately to the left of the data field (if there are no zeros). Then using STA as an indirect address, we will be able to fill with spaces from the dummy position through the address in STA, thus clearing everything except the significant digits.

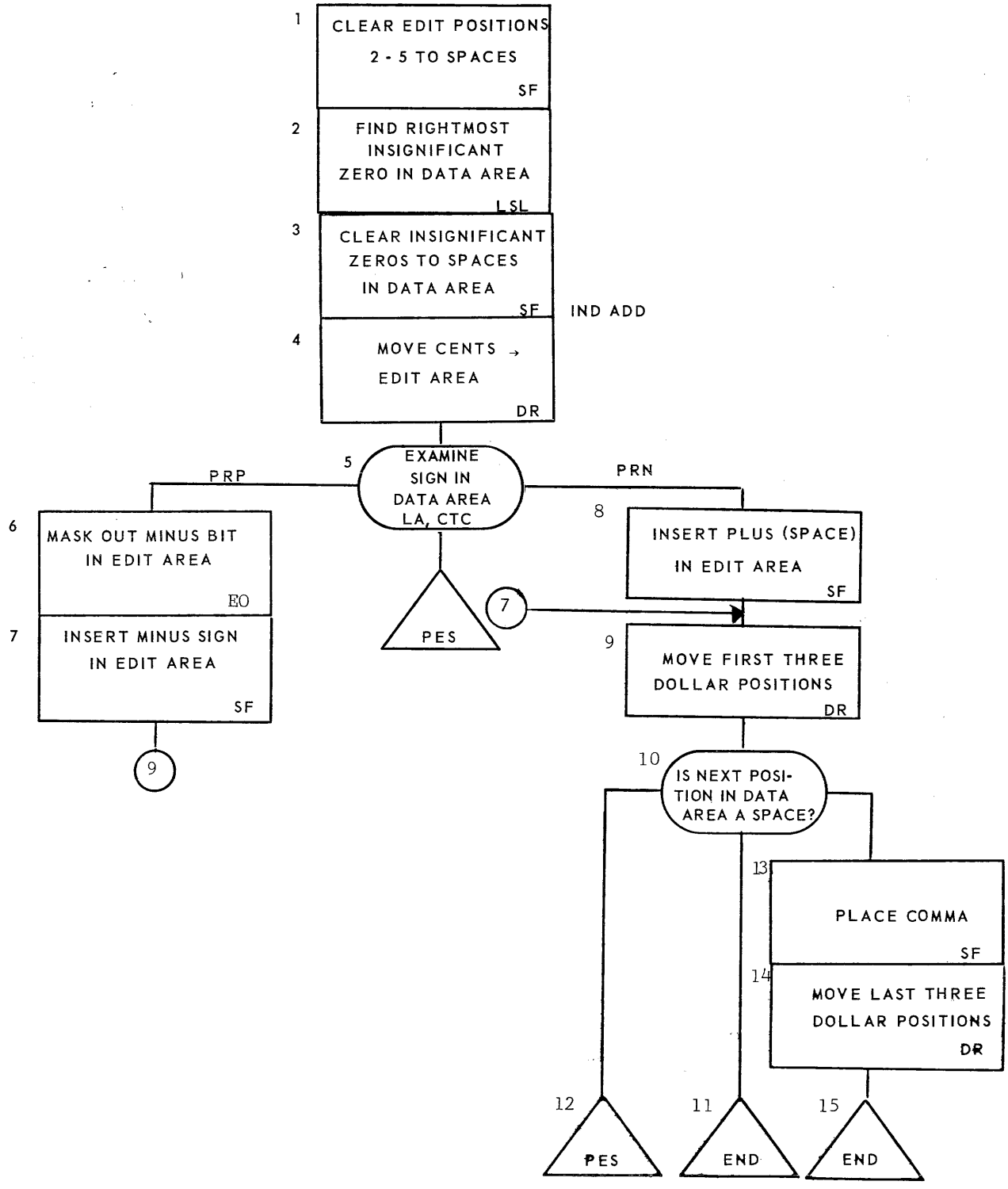
The next step would be to move just the cents to the edit area so that they would fall to the right of the decimal point.

Assuming that the original data area was of no further use, we could now examine the least significant digit for the sign. This could be accomplished by using a Logical And Instruction to mask out all the bits except the 2^5, and then use a Conditional Transfer of Control testing the PRI·s to find if the result had any 1 bits in it. If it did, PRP would be set and this would indicate a negative amount. This condition would require masking out the 2^5 bit of the least significant digit in the edit area, to eliminate the sign bit, and then inserting a minus sign in the sign position of the edit area. If the PRN were set, this would indicate a positive amount and the only step that would have to be taken would be placing a space (which represents a plus sign) in the sign position.

At this point we could move the right three dollar locations in the original area to the left of the decimal point in the edit area.

In order to determine when the operation finishes, we could then test the thousand·s position in the original area. If a space is present in that location, the routine is finished. If the character in that position is greater than a space, some error must have occurred (the data is not numeric) and the routine should act accordingly. If the character is less than a space, there are more numeric characters left to be transferred, so a comma should be placed in the edit area and then the last three dollar positions should be moved from the original area to the edit area.

Flow charted, the routine would look as follows:


1
CLEAR EDIT POSITIONS 2 - 5 TO SPACES
SF
2
FIND RIGHTMOST INSIGNIFICANT ZERO IN DATA AREA
LSL
3
CLEAR INSIGNIFICANT ZEROS TO SPACES IN DATA AREA
SF
IND ADD
4
MOVE CENTS → EDIT AREA
DR
5
EXAMINE SIGN IN DATA AREA LA, CTC
PRP
PRN
6
MASK OUT MINUS BIT IN EDIT AREA
EO
7
INSERT MINUS SIGN IN EDIT AREA
SF
9
PES
7
8
INSERT PLUS (SPACE) IN EDIT AREA
SF
9
MOVE FIRST THREE DOLLAR POSITIONS
DR
10
IS NEXT POSITION IN DATA AREA A SPACE?
12
PES
11
END
13
PLACE COMMA
SF
14
MOVE LAST THREE DOLLAR POSITIONS
DR
15
END

We could time each step in this routine in every program that used it, or we could develop a timing formula for it initially, and then when we incorporate this particular editing approach in a program, we could incorporate its timing formula into the total time calculations. This formula is, of course, nothing but an accumulation of the individual instruction times. For the flow charted routine it would be (in microseconds):

567 + 21n + a + b

where

n = the number of insignificant zeros

a = 0 if the amount is positive or
105 if the amount is negative

b = 0 if the amount contains less than 6 digits
133 if the amount contains 6 or more digits

EXAMPLES:

1.

E* X T R A	0	3	6	7	3	1	2	3
1	2	3	4	5	6	7	8	9

*Extra one character needed for maximum field protection.

Locating the rightmost insignificant zero between position 2 and 7 of the data field and then using STA as the indirect address in the B address of a sector fill, we can clear from position 1 - 2 to spaces.

-	-	3	6	7	3	1	2	3

Transferring the cents to the edit area, it appears as follows:

EDIT AREA

$								,	2	3	

By masking out all bits of the least significant digit in the data field except the 2^5 position and then doing a test of the PRI•s we can determine that this is a positive amount and therefore insert a plus sign (space) in the edit area. Transferring the dollars, tens, and hundreds figure we then have:

$					7	3	1	,	2	3	—

Examining the thousands digit in the data area, we find that it is a decimal digit (< space), and therefore, insert a comma and transfer the last three digits:

$	-	3	6	,	7	3	1	.	2	3	—

Time:

567 + 21 x 1 + 0 + 133 = 721 us.

2.

E X T R A	0	0	0	0	1	2	7	N

EDIT AREA

$	-	-	-	-	-	1	2	•	7	5	⊖

Time:

567 + 21 x 4 + 105 + 0 = 756 us.

For a second example, let's assume that we are to print checks. For security purposes, the dollar sign should always be printed immediately to the left of the left of the most significant digit of the amount, so this will require what is called a "floating" dollar sign. The routine is basically the same as with the fixed dollar sign, with certain modifications:

1) It will now be necessary to clear the edit area from position 1 to 5, since the dollar sign will be placed in the edit area in a different position each time, rather than being placed once in housekeeping.

2) After clearing the insignificant zeros to spaces, it will be necessary to again use STA as an indirect address, and place a dollar sign ($) over the rightmost space in the data area. This will place the dollar sign immediately to the left of the most significant digit. It seems unnecessary to place a space in this location and then a dollar sign over it, but this is cheaper, timewise, than a modification of STA would be.

3) When sensing the thousands' position for a space, we now have the possibility of discovering a character which has a binary code greater than a space. This would be the dollar sign, and would indicate that the operation could be terminated simply by placing a dollar sign in position 5 of the edit area.

The time formula for this routine is slightly greater due to the addition of the above steps:

637 + 21n + a + b (in microseconds)

where:

n = the number of insignificant zeros

a = 0 if the amount is positive or
105 if the mount is negative

b = 147 if the edited amount contains more than 5 digits
0 if the edited amount contains less than 5 digits
56 if the edited amount contains exactly 5 digits

EXAMPLE 1:

DATA AREA

EXTRA	0	0	2	6	5	1	2	3
-	-	-	2	6	5	1	2	3
-	-	$	2	6	5	1	2	3

EDIT AREA

-	-	$	2	,	6	5	1		2	3	-

Time:

637 + 21 x 2 + 0 + 147 = 826 us.

EXAMPLE 2:

DATA AREA

EXTRA	0	0	0	0	0	2	3	1
-	-	-	-	-	-	2	3	1
-	-	-	-	-	$	2	3	1

EDIT IN PLACE

EDIT AREA

-	-	-	-	-	-	$	2	.	3	1	+

Time:

637 + 21 x 5 + 0 + 0 = 742 us.

Editing Exercise I:

A file of fixed length, 116 character records is to be read into memory and then printed out to the on-line printer (one record per line) from this initial read-in-area. All the information except the last 8 characters is already edited for printing (either fixed in length such as stock number, or alphabetic and carrying spaces, such as description). These last 8 characters, however, represent a dollar amount and carry insignificant zeros if needed. Your portion of the program is to edit this amount in order to:

1) eliminate insignificant zeros, if any;

2) place a fixed dollar sign at the leftmost position of the amount field;

3) place the proper sign in the rightmost position of the amount field;

4) insert a decimal point;

5) right justify;

6) insert a comma, if necessary.

Note that the amount will fall in positions 109-116, which will give you the additional 4 locations (117-120) needed for the editing.

Flow chart your routine, and develop a timing formula for it. (Assume for timing purposes only that at least one insignificant zero will be present and at least 1 character that is not equal to 0 will be present.)

XXVIII – PROGRAM CONTROLS

In every program there are certain controls which must be incorporated. These are not actually a part of the production program, but they are necessary in order to process the data in a smooth, orderly, and controlled fashion.

These routines include such things as:

1) Housekeeping
2) ED routines
3) ETW routines
4) EF routines
5) Error stop routines

It must be remembered that although certain things must be accomplished by each of these routines, the complexity and the manner in which they are handled are a matter of programming specifications of the individual installation.

Using a sample problem, let·s now investigate how we might program for these sub-routines:

We have 20,000 employees each with the following master:

EMP NO.	SEC	DEPT	NAME	ST. ADD	C-S ADD.	NO OF DEPENDENTS
5	3	2	25	20	20	2

TOTAL WITH.	TOTAL S.S.	TOTAL GROSS
6	5	7

We will have no more than one transaction for each master, but there may be transactions with no masters, and vice-versa, masters with no transactions. The transactions have the format:

EMP. NO.	WEEKLY WITH.	WEEKLY S.S.	WEEKLY GROSS
5	4	3	5

If a transaction comes in with no master, it will have the same format as a master and must simply be written out to the master tape.

We will assume that the master file is multi-reel, but that the transactions are contained on one reel.

The tape allocations are as follows:

Master input	1 and 4
Master output	2 and 5
Trans. input	3

SYSTEMS CHECKS:

Since it is absolutely necessary to maintain a strict control over the data being processed, many safeguards are incorporated in programs to assure their accuracy. The number and type of safeguards programmed will depend, of course, on the requirements of the specific installation. The following are just a few of the common ones:

Tape Labels:

In order to be sure that the operator has mounted a specific reel of data, tapes are given unique labels or names. This label is the first block on the tape. The program is then written in such a way that before the tape is processed this first block is read in and compared with the proper name of the reel. If equality exists, the processing may begin. If the labels are not alike, the operator must be notified and he must then mount a new tape. This sort of procedure will continue until the correct tape is on the tape unit.

The actual format of a tape label is decided upon by the individual installation. Our example, however, appears below:

IDENTIFICATION	REEL NUMBER	DATA WRITTEN	PURGE DATE
8	3	6	6

Each file is given a unique identification number (or alpha name).

In our problem we will use:

MASTER:

MASPAYRL (MASTER PAYROLL)

TRANSACTION

PYRLTRNS (PAYROLL TRANSACTIONS)

Each reel within that file has a reel number: 001, 002, etc.

The date written is, of course, the date the tape was prepared, and the purge date is the date when this particular reel will again be available for new data. This last item is extremely important, as most installations work on what is termed the "grandfather-father-son" system. This simply means that two back-up files are kept for each file so that if it becomes necessary to reconstruct the file it can be accomplished with the least amount of work.

When a tape label is constructed, the DATE WRITTEN is obviously today·s date. The PURGE DATE could be developed by the program simply by increasing today·s date by the number of processing periods that we are going to keep this tape as back up. For example, if the processing cycle is, as in our example, 1 week, we would want to keep this tape no less than 3 weeks. In order to determine if the tape mounted is the correct tape, we would have to compare the tape label already on the reel with the correct tape label. At the beginning of the program (for our master) we would have the file identification and 001 indicating the first reel.

The DATE WRITTEN would be today's date less one week. To avoid having to go through the portion that modifies the date, we will assume that we will bring in at the beginning of the run:

TODAY'S DATE	DATE WRITTEN FOR THE MASTER	DATE WRITTEN FOR TRANS.	PURGE DATE OF THE NEW MASTER
6	6	6	6

Not only will we have to check the tape label of the master file, we will also have to check the tape label of the transaction file. Since the master is a multi-reel file, we will have to modify the tape label as to reel number in order to check the tape labels of all the subsequent reels.

So far we have discussed only the input tapes. We can easily see that we must also label the output tapes, so that these labels can act as a check during future processing.

Trailer Record:

In order to ascertain that all records in a file have been processed, it is common to carry an additional record that appears after the ED (or EF) on a reel. This is called a "trailer record". This trailer record may contain a number of things, but the most common is a count of the number of records on that reel. When the tape is first prepared, this count is developed by simply adding one each time a record is written to the tape. When ETW is sensed, this trailer record is placed on the reel. The next time the reel is processed, one is added to a work area each time a record is read in from the tape. When the ED is sensed, it is a simple matter to read in the trailer record and compare the two counts to make sure that each record in the file has been processed.

Another type of total that is often placed in the trailer record is called a "hash count". This is the sum of all the criteria of the records in the file (for example, the sum of all the employee numbers). This is developed and used in the same way as the record count, but becomes even more important if we should gain or lose a record. If, for example, we have the following trailer record:

```
# OF RECORDS.8000;  HASH TOTAL 34625347
```

and at the end of processing the file we find:

```
# of records 7999
hash total 34612345
```

we can easily determine that 1 record has not been processed and that record has an employee number of 13002.

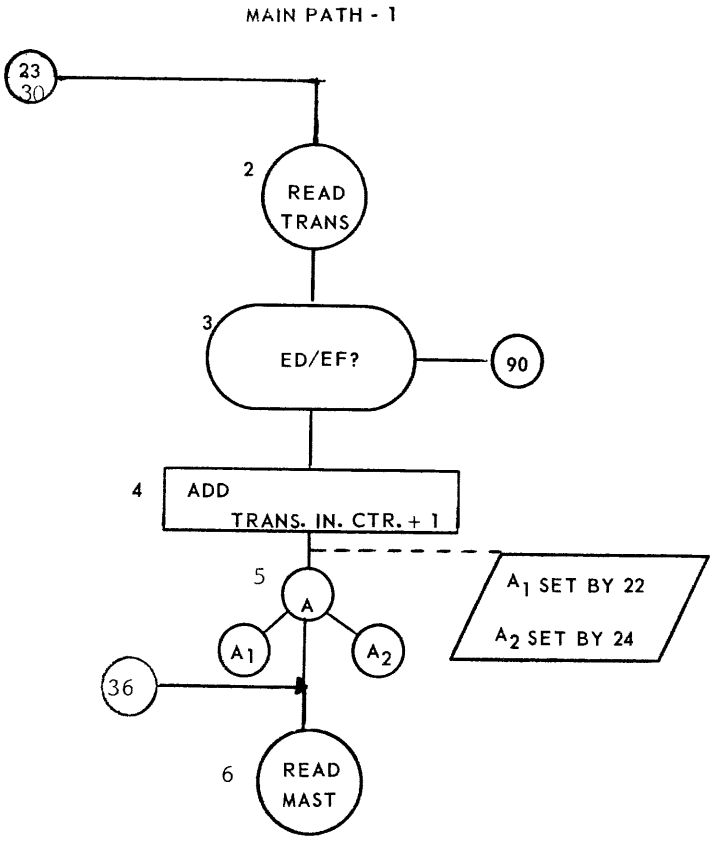
CONTROL TOTALS:

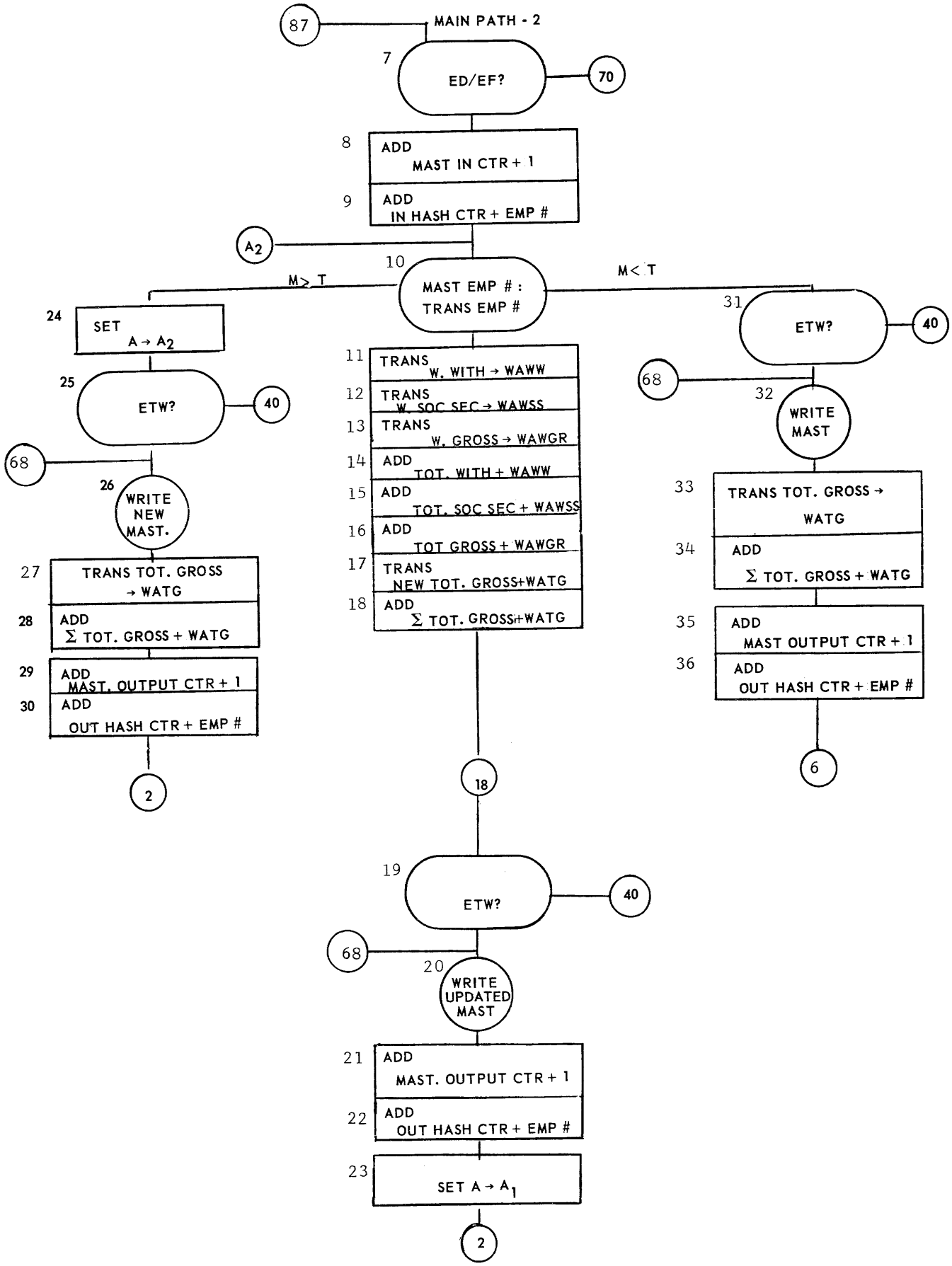
To verify any arithmetics which might be done in the program, control totals may be utilized. For example, the trailer record of the transaction file may contain the number of records and the sum of all the weekly gross salaries.

The trailer record at the end of the master file may contain, in addition to the record count and the hash count, the sum of all the total gross salaries. By adding these two together, we should be equal to the new sum of the total gross salaries, which will then become part of the trailer record on the new file.

LIMIT CHECKS:

Another type of check is called a "limit check". This is performed while processing each record. For example, in our problem we might have checked to make sure that no weekly gross salary exceeds $250. If any exceed this limit the transaction could be treated as a reject.




87
MAIN PATH - 2
7
ED/EF?
70
8
ADD
MAST IN CTR + 1
9
ADD
IN HASH CTR + EMP #
A2
10
MAST EMP # :
TRANS EMP #
M > T
M < T
24
SET
A → A2
25
ETW?
40
68
26
WRITE
NEW
MAST.
27
TRANS TOT. GROSS
→ WATG
28
ADD
Σ TOT. GROSS + WATG
29
ADD
MAST. OUTPUT CTR + 1
30
ADD
OUT HASH CTR + EMP #
2
11
TRANS
W. WITH → WAWW
12
TRANS
W. SOC SEC → WAWSS
13
TRANS
W. GROSS → WAWGR
14
ADD
TOT. WITH + WAWW
15
ADD
TOT. SOC SEC + WAWSS
16
ADD
TOT GROSS + WAWGR
17
TRANS
NEW TOT. GROSS+WATG
18
ADD
Σ TOT. GROSS+WATG
18
19
ETW?
40
68
20
WRITE
UPDATED
MAST
21
ADD
MAST. OUTPUT CTR + 1
22
ADD
OUT HASH CTR + EMP #
23
SET A → A1
2
31
ETW?
40
68
32
WRITE
MAST
33
TRANS TOT. GROSS →
WATG
34
ADD
Σ TOT. GROSS + WATG
35
ADD
MAST OUTPUT CTR + 1
36
ADD
OUT HASH CTR + EMP #
6

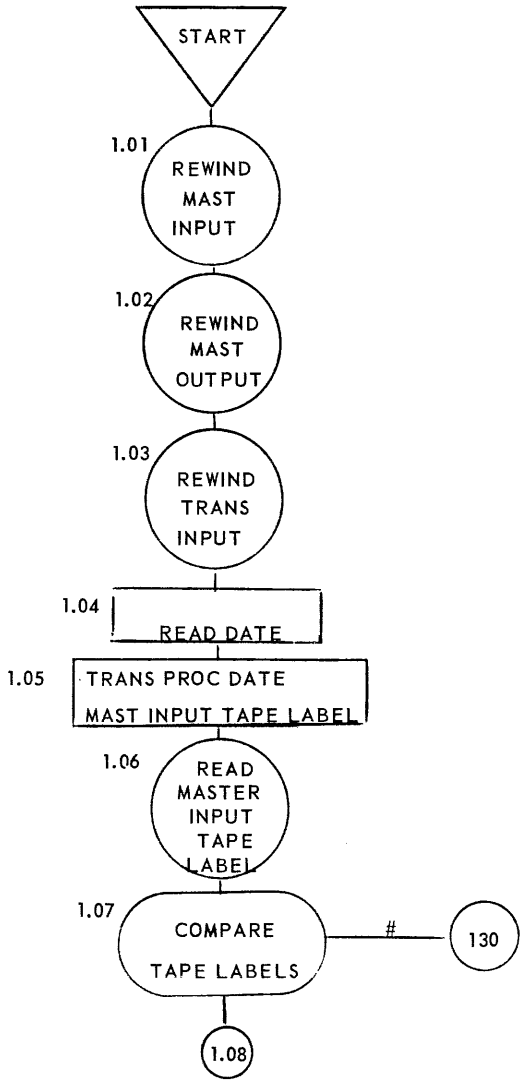
HOUSEKEEPING:

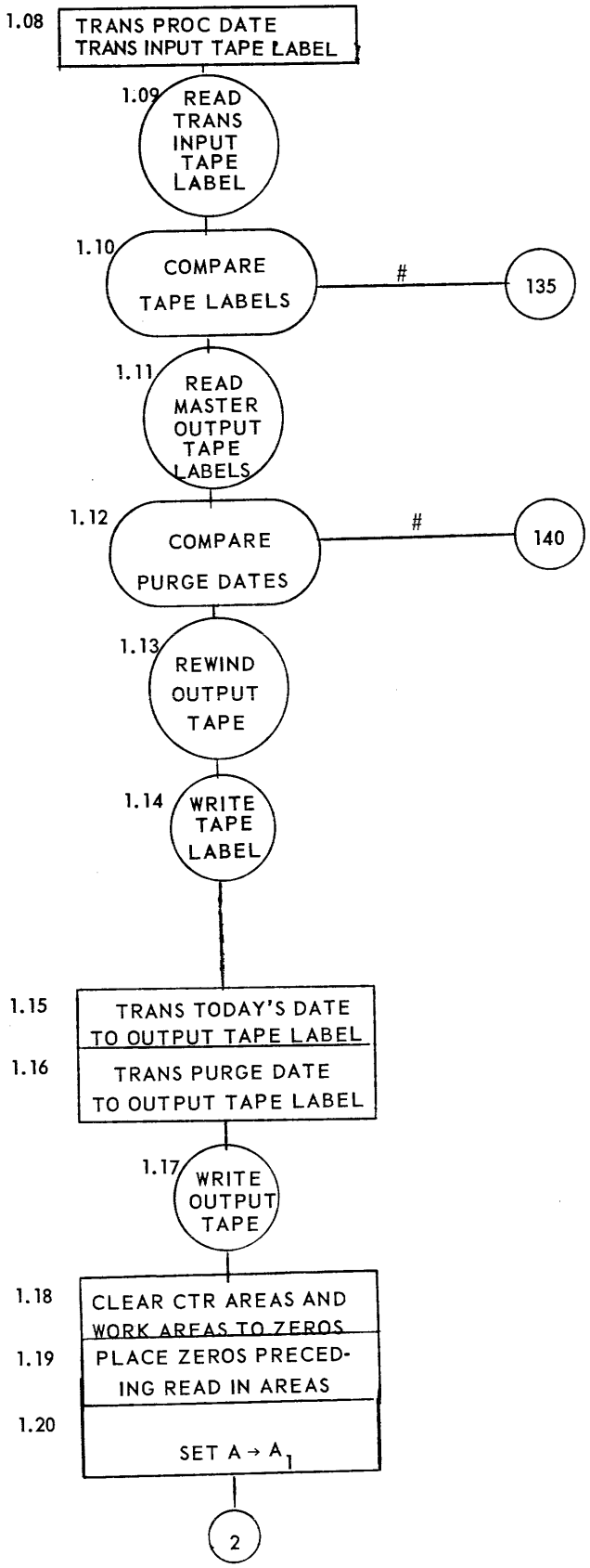
We have already discovered that a number of steps must be executed before the production part of the program is begun.

Summarizing:

1) all tapes must be positioned at BTC
2) tape labels on input tapes must be checked
3) tape labels on output tapes must be checked to see if the purge date has passed and the tapes are available for re-use
4) tape labels must be put on the output tapes
5) any areas to be used as counters must be cleared to zeros
6) variable connectors and modified addresses should be set at their initial condition to facilitate re-start if necessary
7) any steps unique to the program itself should be performed

HOUSEKEEPING - 1




1.08
TRANS PROC DATE
TRANS INPUT TAPE LABEL
1.09
READ TRANS INPUT TAPE LABEL
1.10
COMPARE TAPE LABELS
#
135
1.11
READ MASTER OUTPUT TAPE LABELS
1.12
COMPARE PURGE DATES
#
140
1.13
REWIND OUTPUT TAPE
1.14
WRITE TAPE LABEL
1.15
TRANS TODAY'S DATE TO OUTPUT TAPE LABEL
1.16
TRANS PURGE DATE TO OUTPUT TAPE LABEL
1.17
WRITE OUTPUT TAPE
1.18
CLEAR CTR AREAS AND WORK AREAS TO ZEROS
1.19
PLACE ZEROS PRECEDING READ IN AREAS
1.20
SET A → A1
2

ETW ROUTINES:

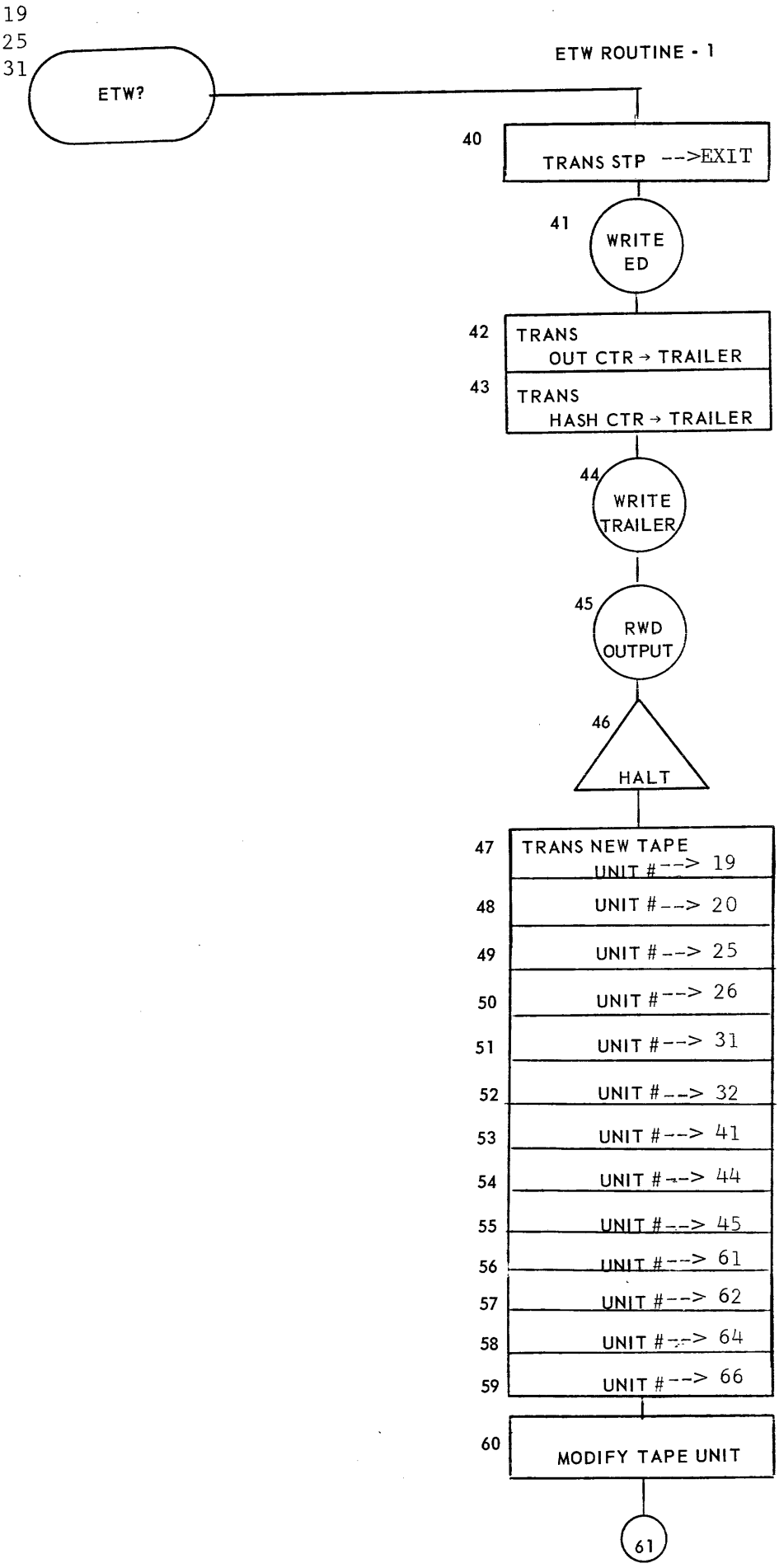
Throughout the program we are checking for the ETW market on the tape. When it is recognized, an indicator is set and sense of this indicator will transfer the program to an ETW routine. At this point we must place an ED on this reel, and follow it with the appropriate trailer record. The trailer counters can then be put back to zeros. Once this is on the tape, the next step would be to initiate the Rewind of that reel. At this point it will be necessary to notify the operator that the tape on that particular tape deck is exhausted and that a new reel will be needed. How this is accomplished will be discussed under the topic HALT ROUTINES. Having notified the operator, we have two choices. One is to simply wait until he has mounted a new tape, but this could means forcing the computer to stand idle for four or five minutes. The other choice is to perform a "tape swap" which will simply change all the references to this first tape deck to a second tape deck, on which there is a new reel. The processing can continue utilizing this second reel while the operator is replacing the first reel. The next time through we can swap back to the first reel while the second reel is being replaced. This can continue until the file is exhausted. To keep the programming at a minimum, it would be advisable to have one location which would always hold the address of the new tape unit and then simply have a number of instructions that would transfer from that location to each instruction which must know the tape unit (ETW, WRITE, etc.). To do this, we must have the ability to change (in our example) a 2 to a 5, and vice versa. Using the Exclusive OR we find that this is relatively simple:

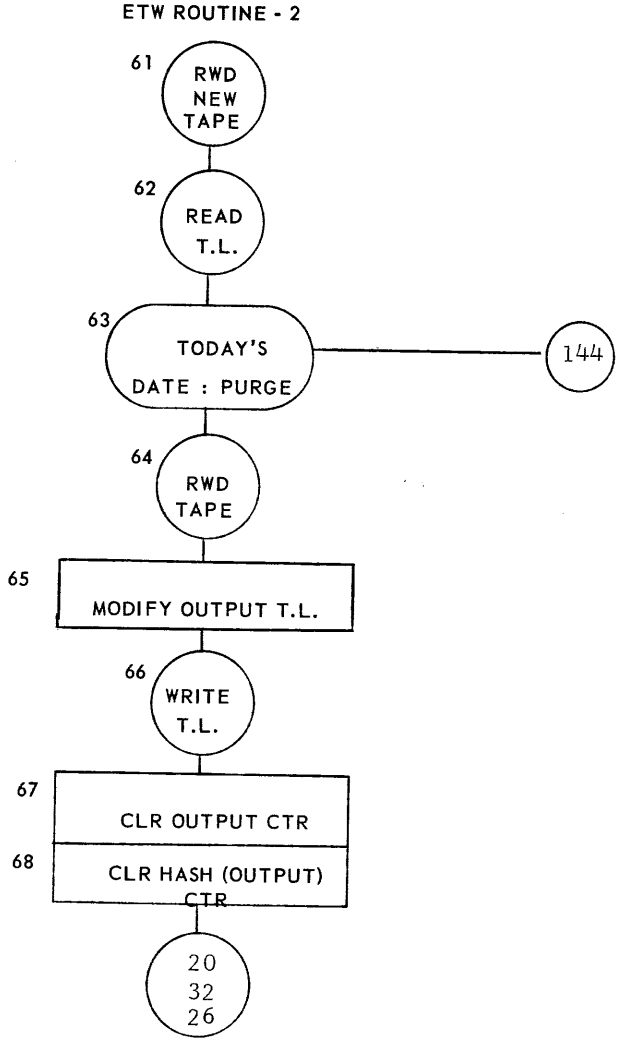
```
     2 is  000010
modify by  000111
  yields   000101   which is 5

     5 is  000101
modify by  000111
  yields   000010   which is 2
```

etc.

We will want to make sure that the new reel of tape is available for processing (today's date > purge date), and if it is, we will want to replace the old tape label, with the proper tape label. (Every tape in the system must have a tape label of some sort, or at least a control symbol such as ED at the beginning of the tape. We will assume that tapes in our example system are all marked with full tape labels). Once this is done, we are ready to transfer back to the program where we left off.

Since there are often many senses for ETW on the same tape, it is advisable to write a common ETW routine for each tape unit. The only thing that will be modified, therefore, is the re-entry point, and this is easily determined by picking up the contents of STP after the I-O Sense.

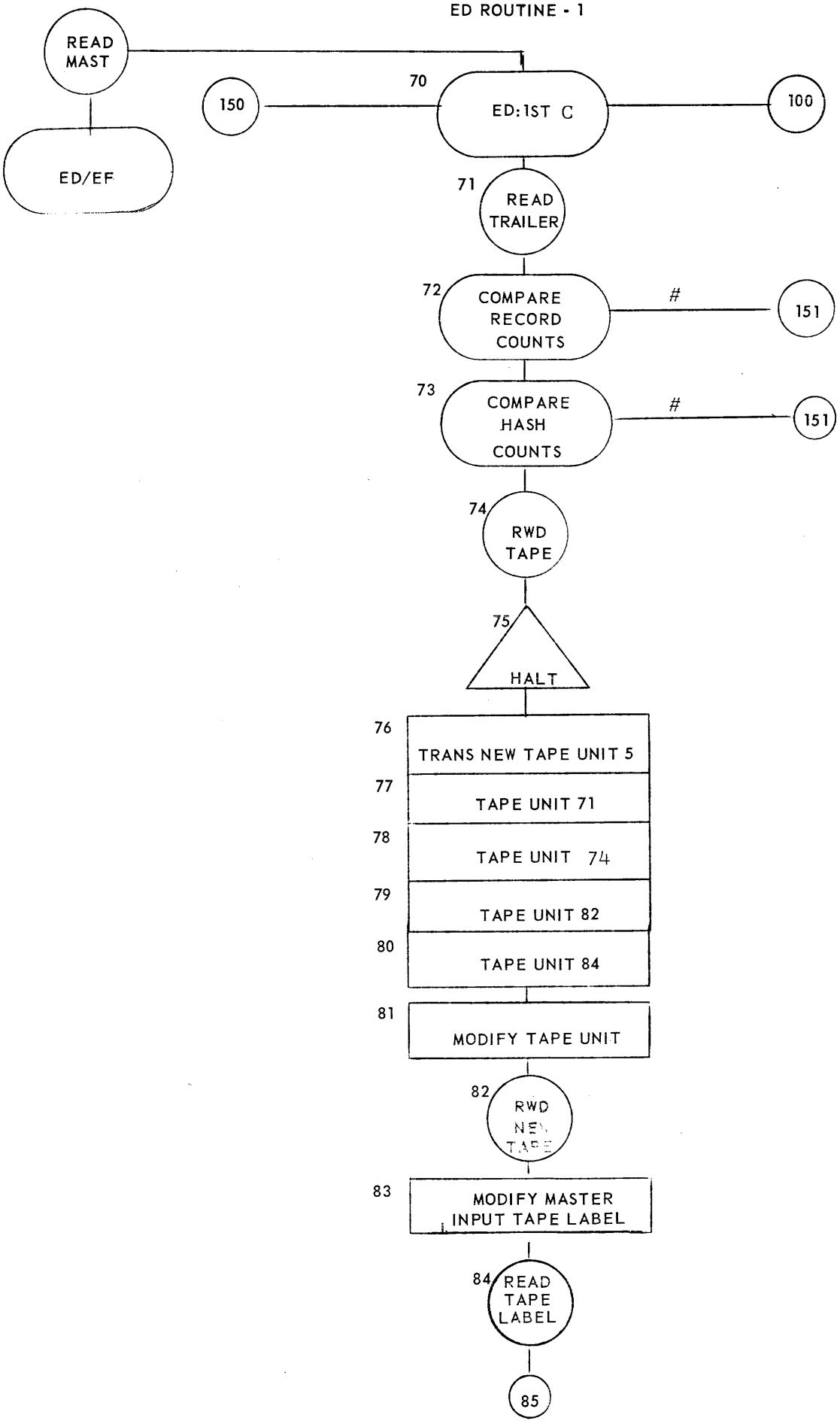

19
25
31
ETW?
ETW ROUTINE - 1
40
TRANS STP -->EXIT
41
WRITE ED
42
TRANS OUT CTR → TRAILER
43
TRANS HASH CTR → TRAILER
44
WRITE TRAILER
45
RWD OUTPUT
46
HALT
47
TRANS NEW TAPE UNIT #--> 19
48
UNIT #--> 20
49
UNIT #--> 25
50
UNIT #--> 26
51
UNIT #--> 31
52
UNIT #--> 32
53
UNIT #--> 41
54
UNIT #--> 44
55
UNIT #--> 45
56
UNIT #--> 61
57
UNIT #--> 62
58
UNIT #--> 64
59
UNIT #--> 66
60
MODIFY TAPE UNIT
61



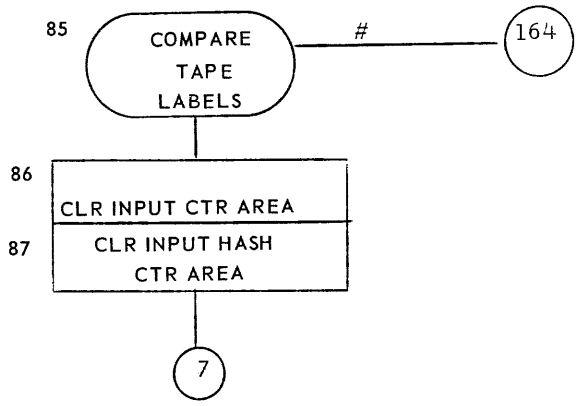
ED ROUTINES:

Everytime a record is read from tape, we sense the ED/EF indicator to see if this has been set. If so, we will transfer to an ED/EF routine. If we are working with a multi-reel file, we must ascertain which one of these control symbols has set the indicator. This can be accomplished by comparing the first character in the read in area against an ED. If it doesn·t match, we would then transfer to an EF routine; if it does match we must perform those steps which are required in a routine that terminates a reel of a file, but not the last reel.

The first step would be to bring in the trailer record and compare it with the totals that have been developed during the program. If they compare, these total areas can be put back to zero. The tape can then be rewound and the operator notified. Again, it may be desirable to swap tapes. When the new tape is checked as to tape label, the program should transfer back to read in a record from the new tape.


READ MAST
ED/EF
150
70
ED:1ST C
100
71
READ TRAILER
72
COMPARE RECORD COUNTS
#
151
73
COMPARE HASH COUNTS
#
151
74
RWD TAPE
75
HALT
76
TRANS NEW TAPE UNIT 5
77
TAPE UNIT 71
78
TAPE UNIT 74
79
TAPE UNIT 82
80
TAPE UNIT 84
81
MODIFY TAPE UNIT
82
RWD NEW TAPE
83
MODIFY MASTER INPUT TAPE LABEL
84
READ TAPE LABEL
85

ED ROUTINE - 2



EF ROUTINES:

When we have determined that it is the EF symbol which has set the ED/EF indicator, we know that we have finished processing that entire file. If this were a one file program, we would then be ready to do into an End of Run routine (EOR), which would "wrap up" the program and allow us to get off the computer. This EOR routine would consist of such things as:

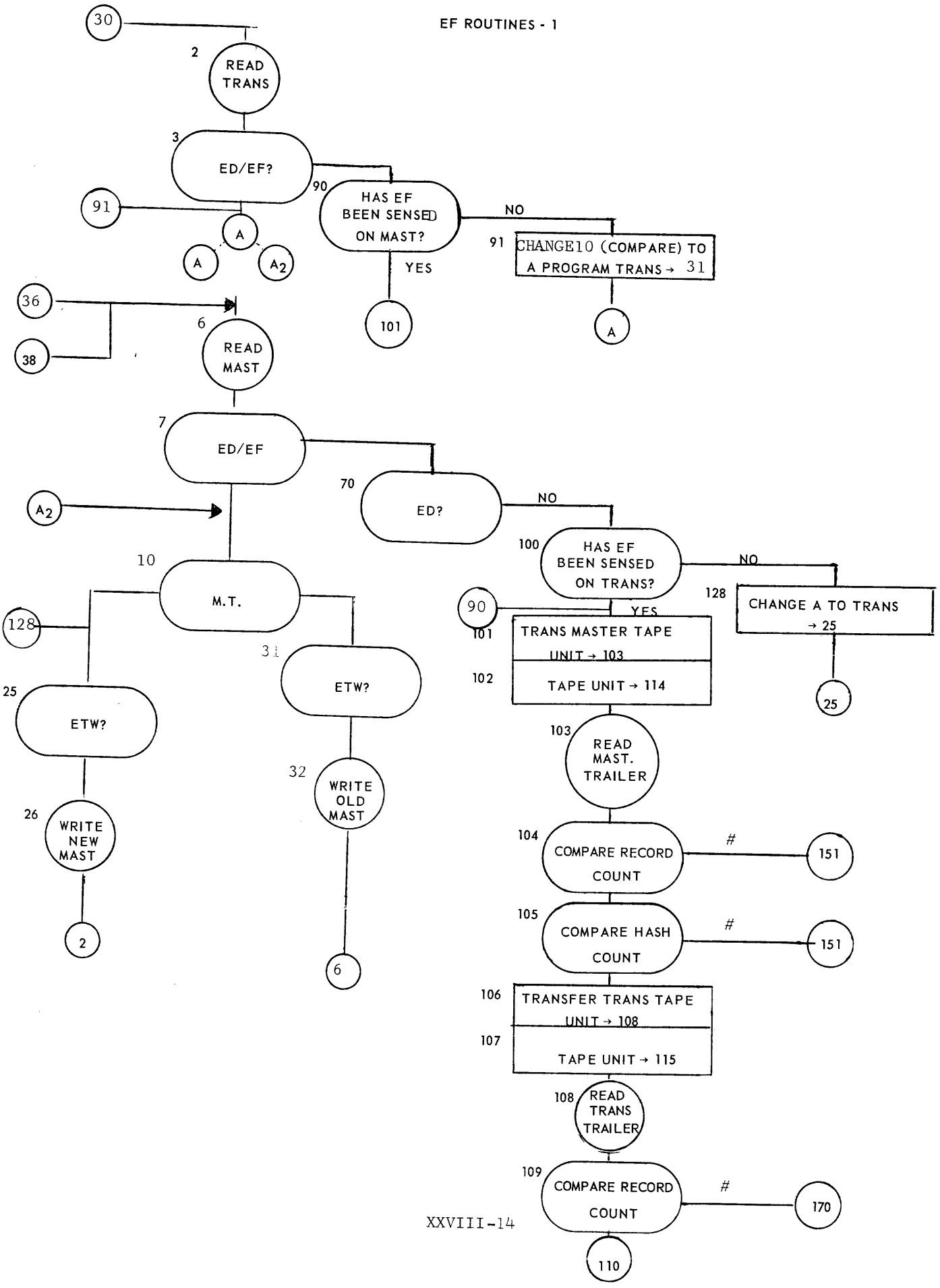
1) checking the last trailer records, which may contain control totals as well as the record count and hash count;
2) placing an EF and the appropriate trailer record on each of the output tapes, and possibly an ED;
3) rewind the tapes;
4) performing any steps which are unique to this program, such as preparing and printing of a summary report, etc.;
5) HALT, notifying the operator of EOR.

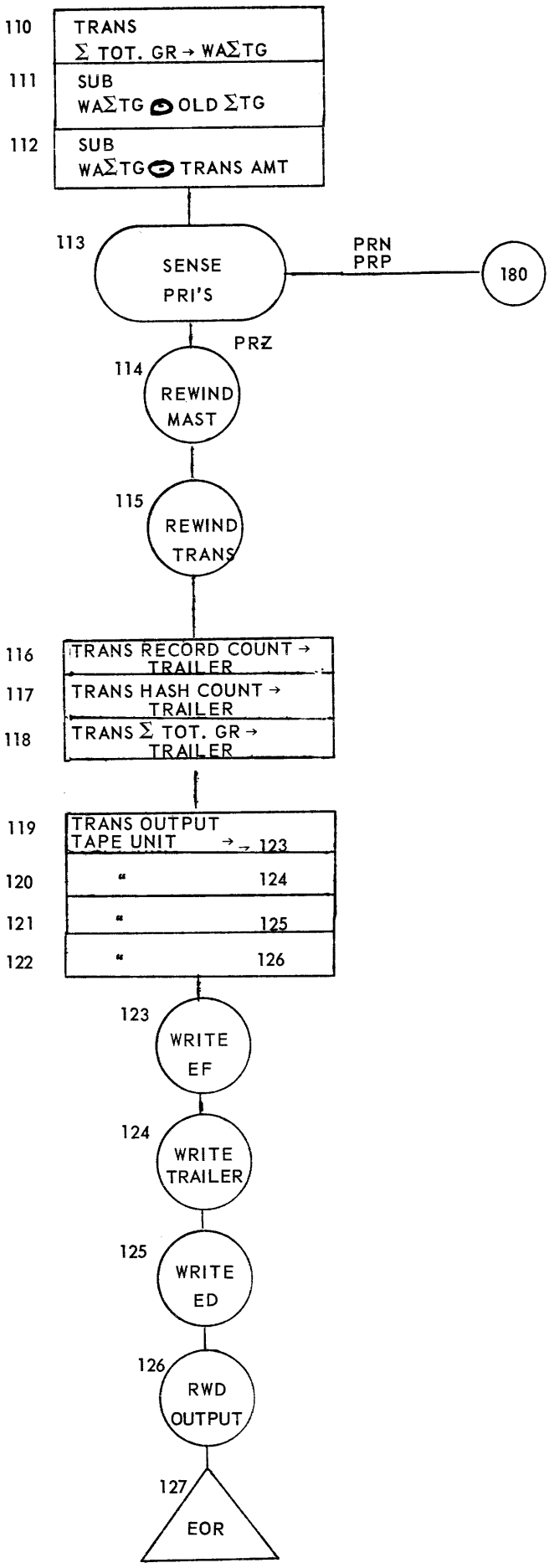
Suppose, however, we have multiple files to be processed, as in our problem. Obviously, if we go to EOR upon sensing the EF of the transaction file, we may not have all the masters on the new master file. If we go to EOR upon sensing the EF of the master file, we may not have processed all of the transactions. To prevent this, we must make sure that an EF has been found on each of the files before terminating the run. This can be done quite simply.

If we sense an EF on the transaction file, we need only sense the first character of the master read in area to see if that also has an EF. If so, we can transfer to the EOR routine. If not, we will want to set up a loop that will bring in a master, write it out, bring in another master, write it out, etc. This can be done without a lot of recoding if we simply modify the program as it stands in memory. (Since we will no longer be using the program in memory to do the regular processing, we are not harming anything.) The modification would simply be to change the instruction that compares the Master and the Transaction employee numbers to an instruction which will force a transfer to step 31. In other words, we will overlay the Compare Left instruction with a Store P Register instruction which transfers to the I-O Sense instruction (step 31). One additional thing must be noted, however, and that is the possibility of already having a master in memory that must be written out before a new master is read in over it. This can be accomplished by simply transferring from the EF routine back to the "A" variable connector, which is, in effect, keeping track of this condition for us.

The same sort of procedure can be done with regard to sensing the EF of the master before the EF of the transaction. The difference is that now we must loop on reading a transaction, incorporating a new master on the new master tape, etc. The program modification is even simpler this time, as we have only to modify the "A" variable connector to transfer to step 25 and then transfer back from the Master EF path to step 25 in order to take card of the transaction which is already in memory.

Obviously, either of these loops would be broken once the EF of the remaining file was sensed.


30
2
READ TRANS
3
ED/EF?
90
HAS EF BEEN SENSED ON MAST?
NO
91
CHANGE10 (COMPARE) TO A PROGRAM TRANS → 31
YES
101
A
91
A
A2
36
38
6
READ MAST
7
ED/EF
A2
70
ED?
NO
100
HAS EF BEEN SENSED ON TRANS?
NO
128
CHANGE A TO TRANS → 25
25
90
YES
101
TRANS MASTER TAPE UNIT → 103
102
TAPE UNIT → 114
10
M.T.
128
31
ETW?
25
ETW?
32
WRITE OLD MAST
26
WRITE NEW MAST
2
6
103
READ MAST. TRAILER
104
COMPARE RECORD COUNT
#
151
105
COMPARE HASH COUNT
#
151
106
TRANSFER TRANS TAPE UNIT → 108
107
TAPE UNIT → 115
108
READ TRANS TRAILER
109
COMPARE RECORD COUNT
#
170
110


110
TRANS
Σ TOT. GR → WAΣTG
111
SUB
WAΣTG ⊖ OLD ΣTG
112
SUB
WAΣTG ⊖ TRANS AMT
113
SENSE
PRI'S
PRN
PRP
180
PRZ
114
REWIND
MAST
115
REWIND
TRANS
116
TRANS RECORD COUNT → TRAILER
117
TRANS HASH COUNT → TRAILER
118
TRANS Σ TOT. GR → TRAILER
119
TRANS OUTPUT
TAPE UNIT → 123
120
" 124
121
" 125
122
" 126
123
WRITE
EF
124
WRITE
TRAILER
125
WRITE
ED
126
RWD
OUTPUT
127
EOR

HALT ROUTINES:

Since there is only one HALT instruction in the RCA 301 complement, an identifying character in the N should be used to differentiate between the EOR stop and the error stops. The character 0 (zero) is to be used for the EOR HALT, while the character 1 denoted the error HALTS.

In addition, each error halt routine should do as much as it is possible for the computer to handle before stopping for operator action. The error halt routine should allow the operator to simply correct the situation, and then hit the start button in order to re-enter the program as the required location. It should be remembered that the least amount of human interference possible is the best.

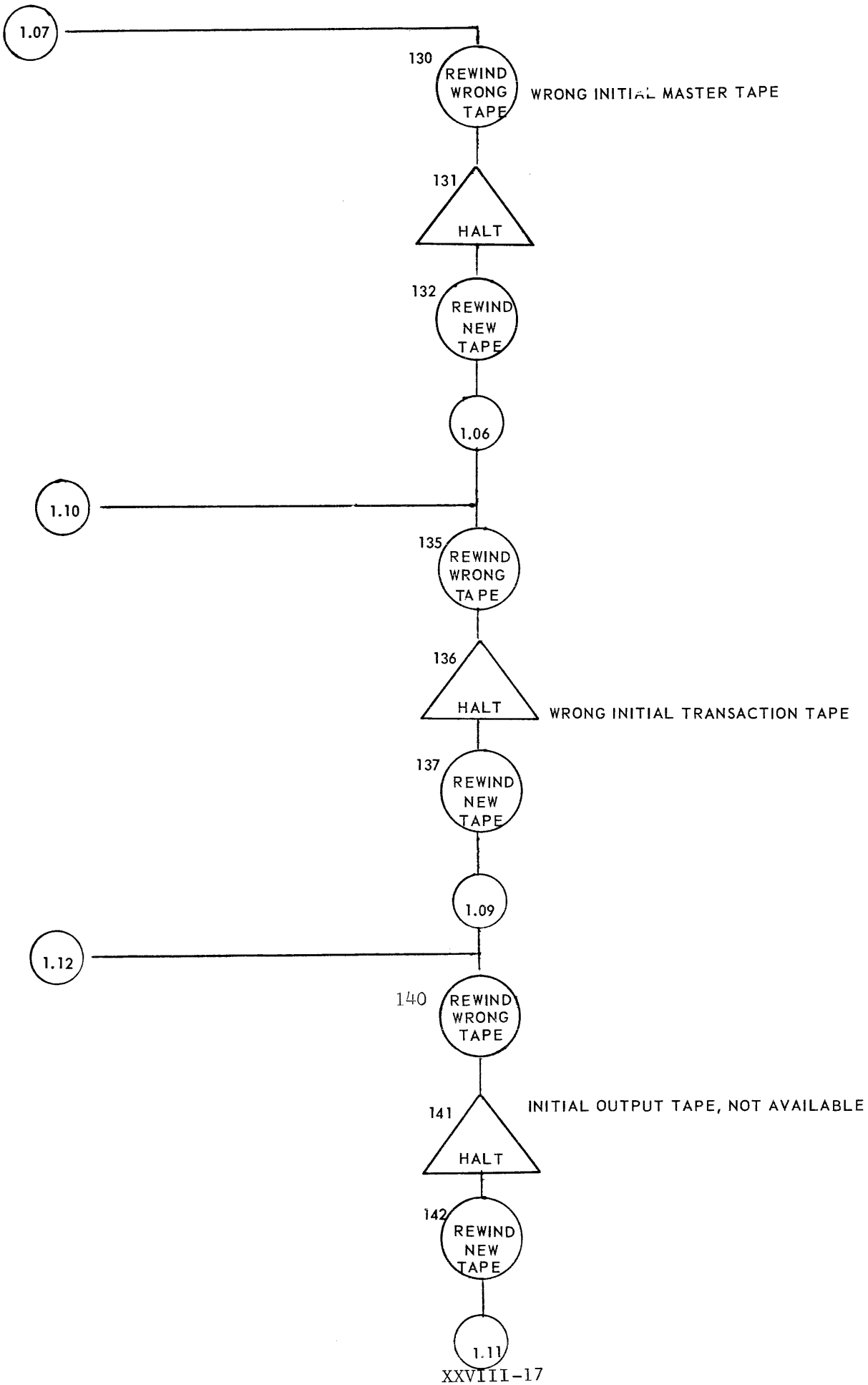
DOCUMENTATION OF THE PROGRAM

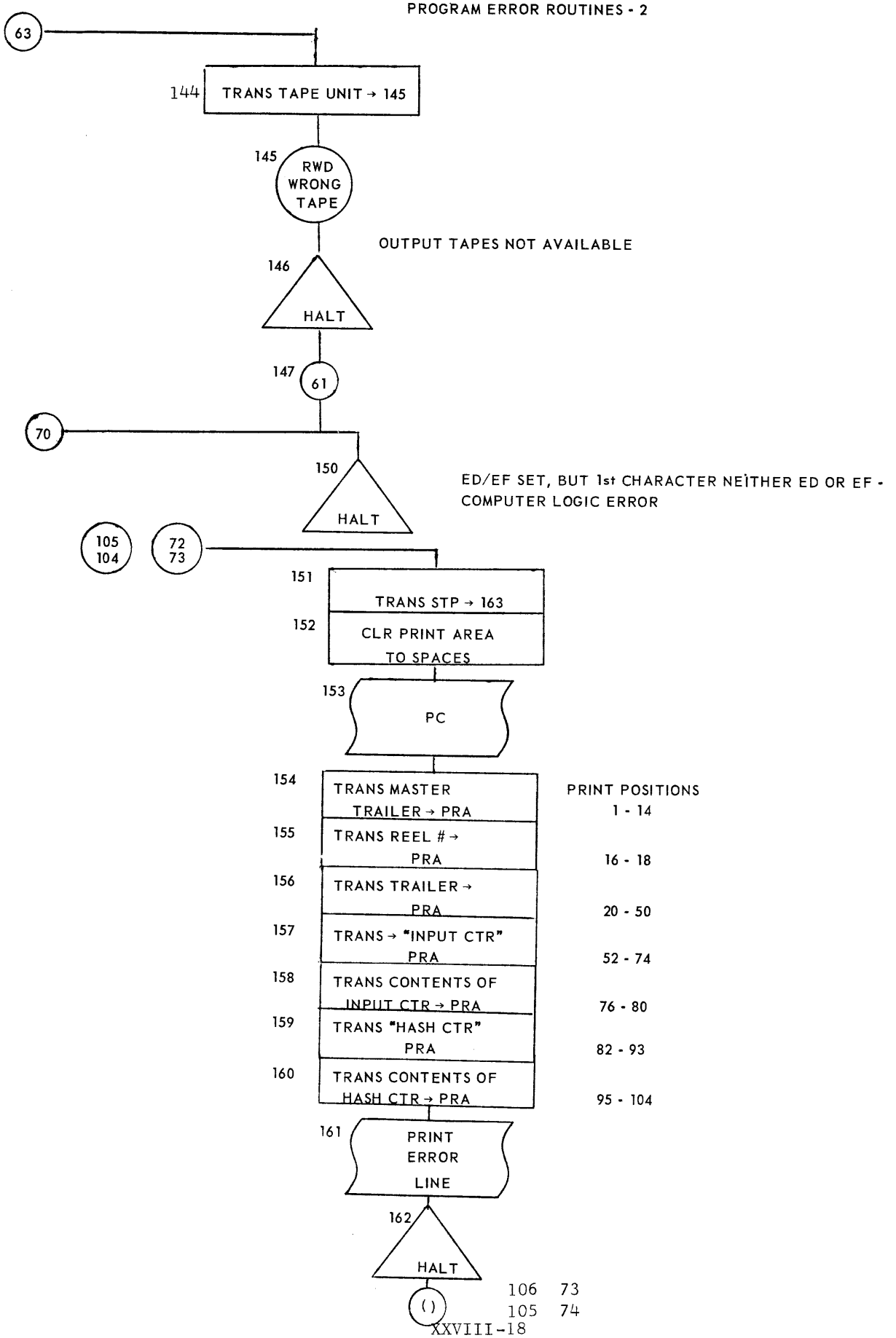
Every program written must be accompanied by a written documentation which covers such things as:

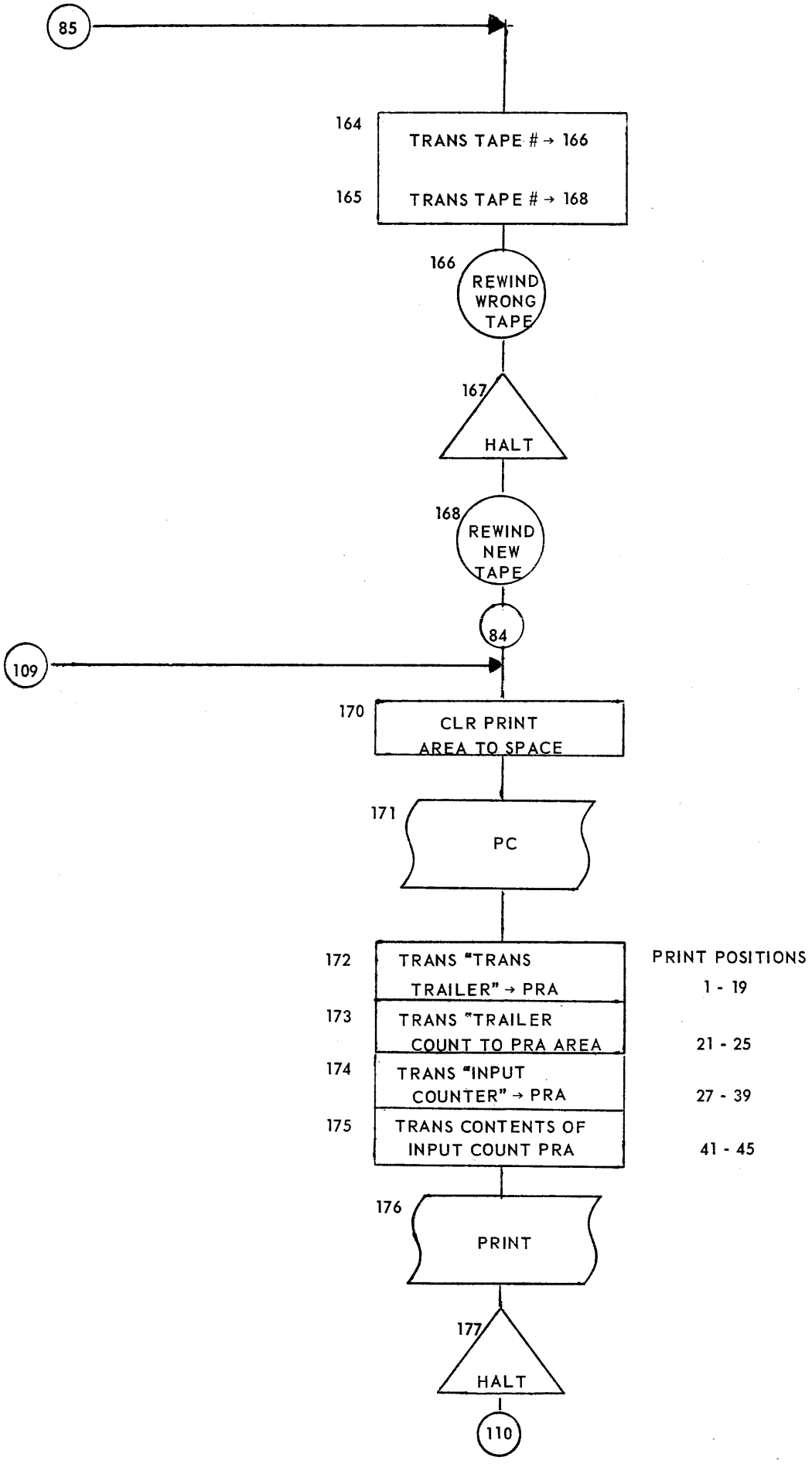
1) Name of Program
2) Name of Programmer
3) Date of final debugging
4) Explanation of what it will do
5) Explanation of how it will do it
6) Detailed flow chart
7) HSM Layout
8) Coding
9) Operational Procedures

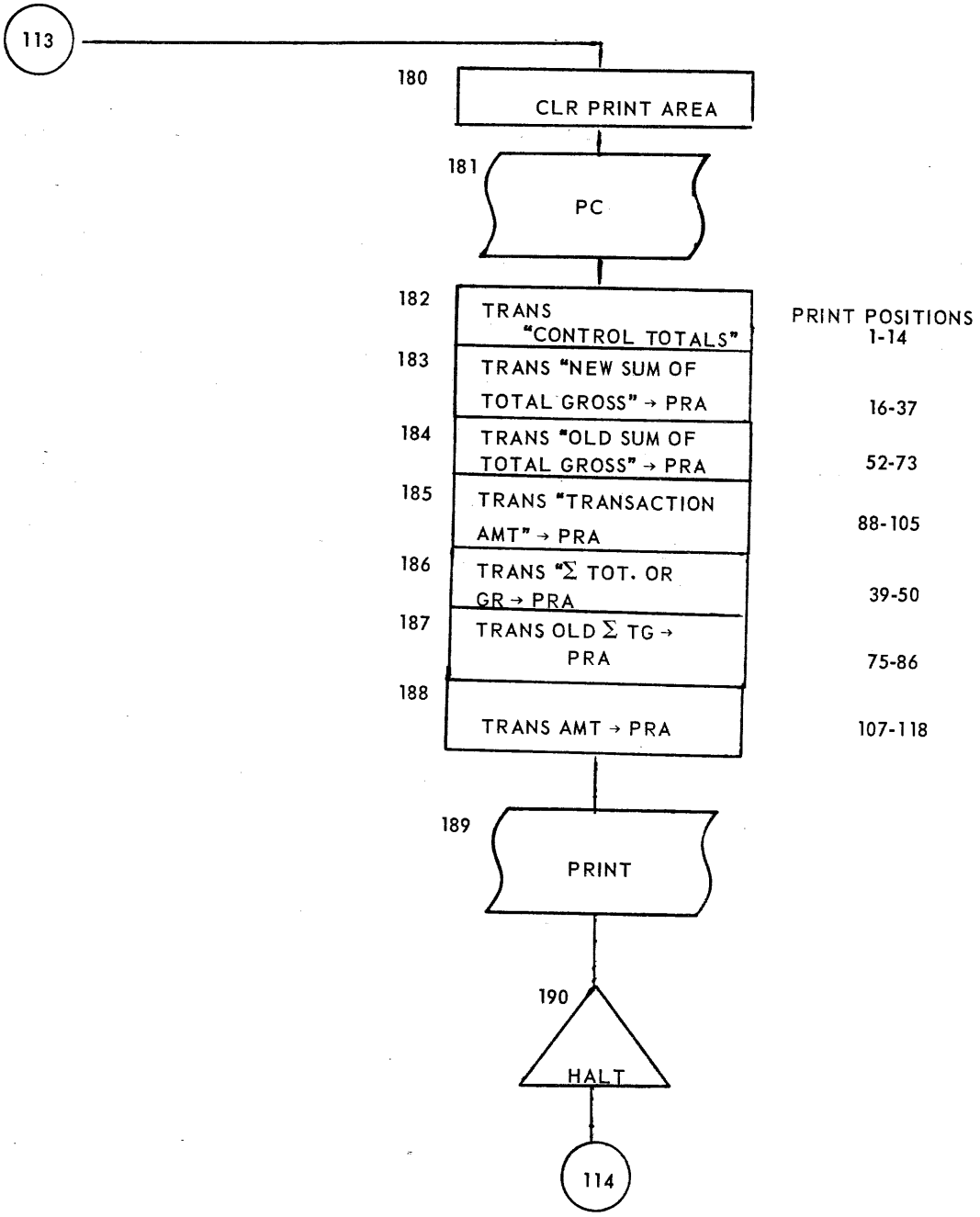
This will enable operators to use it with ease and will also allow for corrections, additions, or deletions, as time goes by.

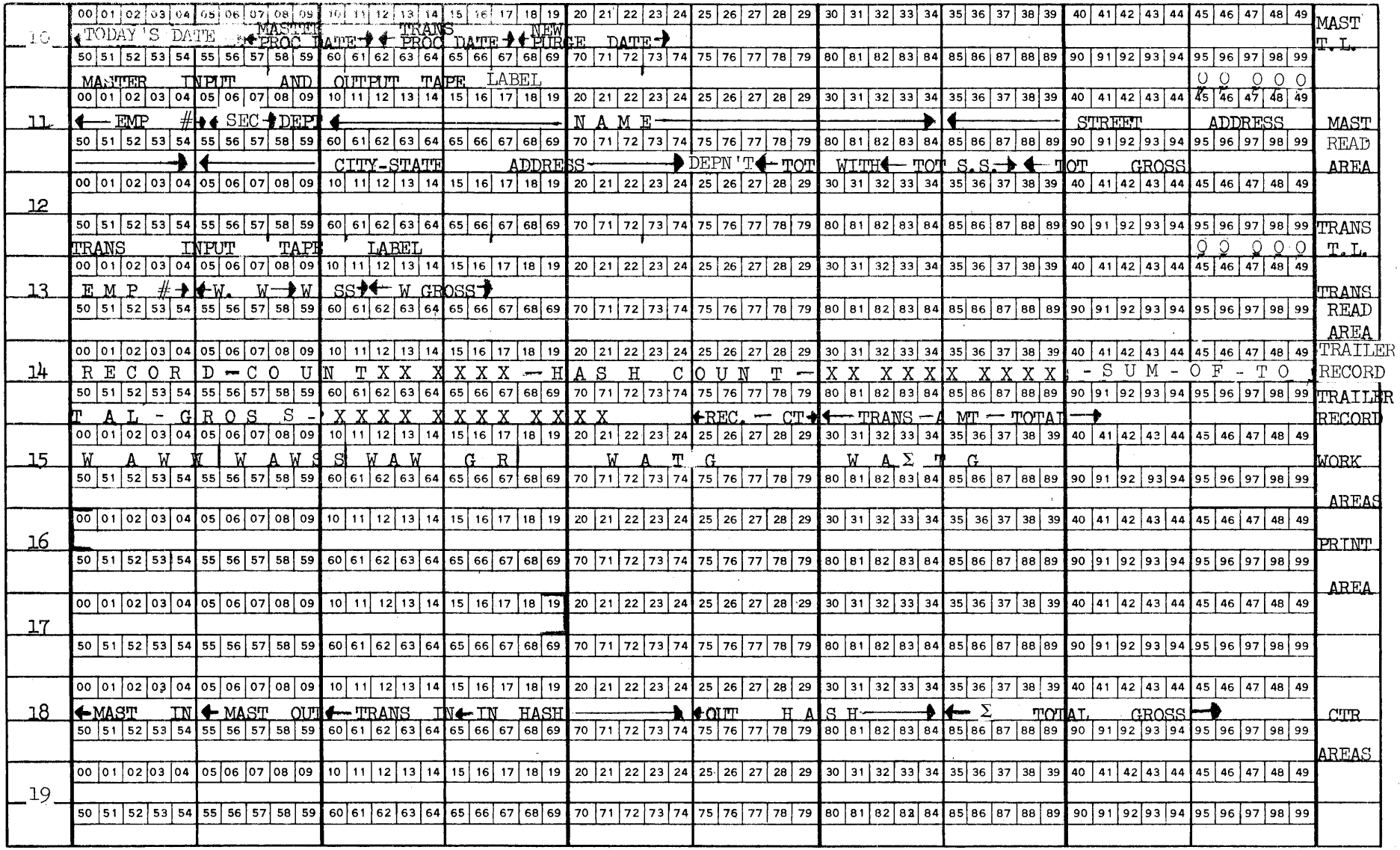
Due to space problems, we will not include the entire Documentation of the Program in this chapter. The flow chart is broken into its individual parts and incorporated with the proper topic. The coding, HSM layout, and the Operation Procedures follow:


1.07
130
REWIND WRONG TAPE
WRONG INITIAL MASTER TAPE
131
HALT
132
REWIND NEW TAPE
1.06
1.10
135
REWIND WRONG TAPE
136
HALT
WRONG INITIAL TRANSACTION TAPE
137
REWIND NEW TAPE
1.09
1.12
140
REWIND WRONG TAPE
141
HALT
INITIAL OUTPUT TAPE, NOT AVAILABLE
142
REWIND NEW TAPE
1.11


63
144
TRANS TAPE UNIT → 145
145
RWD WRONG TAPE
OUTPUT TAPES NOT AVAILABLE
146
HALT
147
61
70
150
HALT
ED/EF SET, BUT 1st CHARACTER NEITHER ED OR EF - COMPUTER LOGIC ERROR
105 104
72 73
151
TRANS STP → 163
152
CLR PRINT AREA TO SPACES
153
PC
154
TRANS MASTER TRAILER → PRA
PRINT POSITIONS 1 - 14
155
TRANS REEL # → PRA
16 - 18
156
TRANS TRAILER → PRA
20 - 50
157
TRANS → "INPUT CTR" PRA
52 - 74
158
TRANS CONTENTS OF INPUT CTR → PRA
76 - 80
159
TRANS "HASH CTR" PRA
82 - 93
160
TRANS CONTENTS OF HASH CTR → PRA
95 - 104
161
PRINT ERROR LINE
162
HALT
( )
106 73
105 74


85
164
TRANS TAPE # → 166
165
TRANS TAPE # → 168
166
REWIND WRONG TAPE
167
HALT
168
REWIND NEW TAPE
84
109
170
CLR PRINT AREA TO SPACE
171
PC
172
TRANS "TRANS TRAILER" → PRA
173
TRANS "TRAILER COUNT TO PRA AREA
174
TRANS "INPUT COUNTER" → PRA
175
TRANS CONTENTS OF INPUT COUNT PRA
PRINT POSITIONS
1 - 19
21 - 25
27 - 39
41 - 45
176
PRINT
177
HALT
110


113
180
CLR PRINT AREA
181
PC
182
TRANS "CONTROL TOTALS"
183
TRANS "NEW SUM OF TOTAL GROSS" → PRA
184
TRANS "OLD SUM OF TOTAL GROSS" → PRA
185
TRANS "TRANSACTION AMT" → PRA
186
TRANS "Σ TOT. OR GR → PRA
187
TRANS OLD Σ TG → PRA
188
TRANS AMT → PRA
PRINT POSITIONS
1-14
16-37
52-73
88-105
39-50
75-86
107-118
189
PRINT
190
HALT
114



TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		210 0	0	0													
		1 0	0	0													
		2 0	0	0													
		3 0	0	0													
	6	214 0	0	0	0	0	0	0	0	0	0	0	0	0			
		5 0	0	0	0	0	0	0	0	0	0	0	0	0			
		6 0	0	0	4	5	3	-	-	-	-	-	5	7		TAPE TABLE (MODIFIERS)	
		7 0	0	0	M	A	S	P	A	Y	R	L	0	0		MASTER INPUT TAPE LABEL	
		8 0	0	0	1	X	X	X	X	X	X	-	-	-			
		9 0	0	0	P	Y	R	L	T	R	N	S	0	0		TRANS INPUT TAPE LABEL	
	6	220 0	0	0	1	X	X	X	X	X	X	-	-	-			
		1 0	0	0	M	A	S	P	A	Y	R	L	0	0		MASTER OUTPUT TAPE LABEL	
		2 0	0	0	1	X	X	X	X	X	X	X	X	X			
		3 0	0	0	X	X	X	-	-	-	-	-	-	-			
		4 0	0	0	;	1	0	0	0	0	0	0	0	0		REWIND MASTER INPUT	1.01
		5 0	0	0	;	2	0	0	0	0	0	0	0	0		REWIND MASTER OUTPUT	1.02

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	226 0	0	0	;	3	0	0	0	0	0	0	0	0		REWIND TRANSACTION INPUT	1.03
		7 0	0	0	0	0	1	0	0	0	0	0	0	0		READ DATES FROM CARD	1.04
		8 0	0	0	N	6	1	0	1	1	2	1	8	6		TRANS. MAST. PROC. DATE TO M.T.L. (INPUT)	1.05
3990		9 0	0	0	4	1	1	0	5	0	1	0	7	2		READ INPUT MASTER TAPE LABLE	1.06
		230 0	0	0	Y	G	1	0	5	0	2	1	7	0		COMPARE TAPE LABELS	1.07
		1 0	0	0	W	1	3	9	6	0	3	9	6	0		IF NOT IDENTICAL, STOP	1.07
	6	232 0	0	0	N	6	1	0	1	7	2	2	0	6		TRANS TRANSACTION PROC. DATE TO T.T.L.	1.08
4030		3 0	0	0	4	3	1	2	5	0	1	2	7	2		READ TRANS TAPE LABEL	1.09
		4 0	0	0	Y	G	1	2	5	0	2	1	9	0		COMPARE TAPE LABELS	1.10
		5 0	0	0	W	1	4	0	0	0	4	0	0	0		IF NOT IDENTICAL, STOP	1.10
4070		6 0	0	0	4	2	1	0	5	0	1	0	7	2		READ MASTER OUTPUT TAPE LABELS	1.11
		7 0	0	0	Y	6	1	0	0	0	1	0	6	7		COMPARE TODAY'S DATE : T.L. PURGE DATE	1.12
	6	238 0	0	0	W	1	2	3	9	0	4	0	4	0		TD > PD →1.13 TD < PD → 140	1.12
2380		9 0	0	0	;	2	0	0	0	0	0	0	0	0		REWIND OUTPUT TAPE	1.13
		240 0	0	0	N	6	1	0	0	5	2	2	2	6		TRANS TODAY'S DATE →OUTPUT TAPE LABEL (MAST)	1.14
		1 0	0	0	N	6	1	0	2	3	2	2	3	2		TRANS PURGE DATE→OUTPUT TAPE LABEL (MAST)	1.15
		2 0	0	0	8	2	2	2	1	0	2	2	3	2		WRITE TAPE LABEL TO OUTPUT TAPE	1.16
		3 0	0	0	J	0	1	5	0	0	1	9	9	9		CLEAR CTR AND WORK AREA	1.17
	6	244 0	0	0	J	0	1	0	9	5	1	0	9	9		PLACE 5 0'S BEFORE MASTER READ IN AREA	1.18
		5 0	0	0	J	0	1	2	9	5	1	2	9	9		PLACE 5 0'S BEFORE TRANS READ IN AREA	1.18
		6 0	0	0	N	4	2	4	9	4	2	5	3	9		SET A→ A_1	1.19
		7 0	0	0	V	1	0	2	1	9	2	5	0	0		TRANS→2	
		8 0	0	0	E_D	E_F	-	-	-	-	2	5	8	0		CONSTANTS FOR MAIN PROGRAM	
		9 0	0	0	-	2	5	4	0	0	0	0	0	1		A_2 and A_1 SETTINGS, 1 FOR CTR ADD (5)	

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
2730 2810 2470	6	2500	0	0	4	3	1	3	0	0	1	3	9	4		READ TRANSACTION	2
		10	0	0	W	8	3	4	8	0	2	5	2	0		ED/EF SENSE	3
2510		20	0	0	+	5	1	8	1	4	2	4	9	9		ADD 1 TO TRANS. INPUT CTR	4
3520		30	0	0	V	1	0	2	1	9	(2	5	4	0)	22,24, 128	SWITCH A $A_1 \rightarrow 5$ $A_2 \rightarrow 9$	5
2880 2530 3970		40	0	0	4	1	1	1	0	0	1	1	9	4		READ MASTER	6
		50	0	0	W	8	3	2	5	0	2	5	6	0		ED/EF SENSE	7
2550	6	2560	0	0	+	5	1	8	0	4	2	4	9	9		ADD 1 TO MASTER INPUT CTR	8
		70	0	0	+	&	1	8	2	4	1	1	0	4		ADD EMP # TO INPUT HASH CTR	9
2530		80	0	0	Y	5	1	1	0	0	1	3	0	0	92	COMPARE MASTER EMP # : TRANS EMP #	10
		90	0	0	W	1	2	7	4	0	2	8	2	0		M > T → 24 M < T → 31	10
		2600	0	0	N	4	1	3	0	8	1	5	0	5		TRANS WEEKLY WITH → WORK AREA WW	11
		10	0	0	N	3	1	3	1	1	1	5	1	0		TRANS W. SOC. SEC. → WAWSS	12
	6	2620	0	0	N	5	1	3	1	6	1	5	1	7		TRANS W. GROSS → WAWGR	13
		30	0	0	+	6	1	1	8	2	1	5	0	5		ADD TOT. WITH + WAWW	14
		40	0	0	+	5	1	1	8	7	1	5	1	0		ADD TOT. SOC. SEC + WAWSS	15
		50	0	0	+	7	1	1	9	4	1	5	1	7		ADD TOT. GROSS + WAWGR	16
		60	0	0	N	7	1	1	9	4	1	5	2	9		TRANS NEW TOT. GROSS → WATG	17
		70	0	0	+	B	1	8	4	6	1	5	2	9		ADD Σ TOT. GROSS + WATG	18
	6	2680	0	0	S	2	4	0	0	0	2	9	3	0		ETW SENSE	19
		90	0	0	8	2	1	1	0	0	1	1	9	4		WRITE MASTER	20
		2700	0	0	+	S	1	8	0	9	2	4	9	9		ADD 1 TO MASTER OUTPUT CTR	21
		10	0	0	+	&	1	8	3	4	1	1	0	4		ADD EMP # TO OUTPUT HASH CTR	22
		20	0	0	N	4	2	4	9	4	2	5	3	9		SET A → A_1	23
		30	0	0	V	1	0	2	1	9	2	5	0	0		TRANSER TO 2	

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
2590	6	274 0	0	0	N	4	2	4	8	9	2	5	3	9		SET A → A_2	24
		5 0	0	0	S	2	4	0	0	0	2	9	3	0		ETW SENSE	25
		6 0	0	0	8	2	1	3	0	0	1	3	9	4		WRITE NEW MASTER	26
		7 0	0	0	N	7	1	3	9	4	1	5	2	9		TRANS TOT. GROSS → WATG	27
		8 0	0	0	+	B	1	8	4	6	1	5	2	9		ADD ΣTG + WATG	28
		9 0	0	0	+	5	1	8	0	9	2	4	9	9		ADD MASTER OUTPUT + 1	29
	6	280 0	0	0	+	&	1	8	3	4	1	3	0	4		ADD OUT HASH + EMP #	30
		1 0	0	0	V	1	0	2	1	9	2	5	0	0		TRANS → 2	
2590		2 0	0	0	S	2	4	0	0	0	2	9	3	0		ETW SENSE	31
		3 0	0	0	8	2	1	1	0	0	1	1	9	4		WRITE MASTER	32
		4 0	0	0	N	7	1	1	9	4	1	5	2	9		TRANS TOT. GROSS → WATG	33
		5 0	0	0	+	B	1	8	4	6	1	5	2	9		ADD ΣTG + WATG	34
	6	286 0	0	0	+	5	1	8	0	9	2	4	9	9		ADD MASTER OUTPUT + 1	35
		7 0	0	0	+	&	1	8	3	4	1	1	0	4		ADD OUT HASH + EMP #	36
		8 0	0	0	V	1	0	2	1	9	2	5	4	0		TRANS --> 6	37
		9 0	0	0	R	E	C	0	R	D	-	C	0	U		TRAILER RECORD	
		290 0	0	0	N	T	-	X	X	X	X	X	-	H			
		1 0	0	0	A	S	H	-	C	0	U	N	T	-			
	6	292 0	0	0	X	X	X	X	X	X	X	X	X	X			
2680 2750 2820		3 0	0	0	N	4	0	2	1	9	3	2	3	9		TRANS STP---> EXIT	40
		4 0	0	0	8	2	2	4	8	0	2	4	8	0		WRITE ED → OUTPUT	41
		5 0	0	0	N	5	1	8	0	9	2	9	0	7		TRANS RECORD COUNT → TRAILER	42
		6 0	0	0	N	B	1	8	3	4	2	9	2	9		TRANS HASH COUNT → TRAILER	43
		7 0	0	0	8	2	2	8	9	0	2	9	2	9		WRITE TRAILER TO OUTPUT	44

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	298 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD OUTPUT	45
		9 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT	46
		300 0	0	0	N	1	2	1	6	1	2	6	8	1		TRANS NEW UNIT → 19	47
		1 0	0	0	N	1	2	1	6	1	2	6	9	1		" " " → 20	48
		2 0	0	0	N	1	2	1	6	1	2	7	5	1		" " " → 25	49
		3 0	0	0	N	1	2	1	6	1	2	7	6	1		" " " → 26	50
	6	304 0	0	0	N	1	2	1	6	1	2	8	2	1		" " " → 31	51
		5 0	0	0	N	1	2	1	6	1	2	8	3	1		" " " → 32	52
		6 0	0	0	N	1	2	1	6	1	2	9	4	1		" " " → 41	53
		7 0	0	0	N	1	2	1	6	1	2	9	7	1		" " " → 44	54
		8 0	0	0	N	1	2	1	6	1	2	9	8	1		" " " → 45	55
		9 0	0	0	N	1	2	1	6	1	3	1	4	1		" " " → 61	56
	6	310 0	0	0	N	1	2	1	6	1	3	1	5	1		" " " → 62	57
		1 0	0	0	N	1	2	1	6	1	3	1	8	1		" " " → 64	58
		2 0	0	0	N	1	2	1	6	1	3	2	0	1		" " " → 66	59
		3 0	0	0	U	1	2	1	6	1	2	1	6	9		MODIFY TAPE UNIT	60
4110		4 0	0	0	;	5	0	0	0	0	0	0	0	0		REWIND NEW TAPE	61
		5 0	0	0	4	5	1	0	5	0	1	0	7	2		READ T.L. FROM NEW TAPE	62
	6	316 0	0	0	Y	6	1	0	0	0	1	0	6	7		COMPARE TODAY'S DATE :-PURGE DATE	63
		7 0	0	0	W	1	3	1	8	0	4	0	8	0		TD ≥ PD → 64 TD < PD → 144	63
3170		8 0	0	0	;	5	0	0	0	0	0	0	0	0		RWD NEW TAPE	64
		9 0	0	0	+	3	2	2	2	0	3	2	4	9		MODIFY OUTPUT TAPE LABEL (REEL #)	65
		320 0	0	0	8	5	2	2	1	0	2	2	3	2		WRITE TAPE LABEL TO NEW OUTPUT TAPE	66
		1 0	0	0	J	0	1	8	0	5	1	8	0	9		FILL RECORD CTR (OUTPUT) WITH ZEROS	67

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	322 0	0	0	J	0	1	8	2	5	1	8	3	4		FILL HASH CTR WITH ZEROS (OUTPUT)	68
		3 0	0	0	V	1	0	2	1	9	(0	0	0	0)	40	TRANS→ ø ,26, or 32	
		4 0	0	0	E_D	E_F	-	-	-	-	-	0	0	1		REEL NO. MODIFIER	
2550		5 0	0	0	Y	1	1	1	0	0	3	2	4	0		1st CHARACTER COMPARED TO AN ED	70
		6 0	0	0	W	1	3	6	2	0	4	1	2	0		1st CHAR > ED → EF 1st < ED → STOP	70
		7 0	0	0	4	1	1	4	0	0	1	4	3	9		READ TRAILER RECORD	71
	6	328 0	0	0	Y	5	1	4	1	3	1	8	0	0		COMPARE RECORD COUNTS	72
		9 0	0	0	W	1	4	1	7	0	4	1	7	0		IF UNLIKE →151	72
		330 0	0	0	Y	&	1	4	3	0	1	8	1	5		COMPARE INPUT HASH COUNTS	73
		1 0	0	0	W	1	4	1	7	0	4	1	7	0		IF UNLIKE→151	73
		2 0	0	0	;	1	0	0	0	0	0	0	0	0		REWIND MAJTER TAPE	74
		3 0	0	0	□	0	0	0	0	0	0	0	0	0		HALT	75
	6	334 0	0	0	N	1	2	1	6	0	2	5	4	1		TRANSFER NEW MASTER TAPE NO. →5	76
		5 0	0	0	N	1	2	1	6	0	3	2	7	1		" " " " " →71	77
		6 0	0	0	N	1	2	1	6	0	3	3	2	1		" " " " " →74	78
		7 0	0	0	N	1	2	1	6	0	3	4	0	1		" " " " " → 82	79
		8 0	0	0	N	1	2	1	6	0	3	4	2	1		" " " " " → 84	80
		9 0	0	0	U	1	2	1	6	0	2	1	6	8		MODIFY TAPE UNIT IN TABLE	81
	6	340 0	0	0	;	0	0	0	0	0	0	0	0	0		REWIND NEW TAPE	82
		1 0	0	0	+	3	2	1	8	0	3	2	4	9		MODIFY REEL NO. IN INPUT MASTER T.L.	83
4360		2 0	0	0	4	4	1	0	5	0	1	0	7	2		READ NEW TAPE LABEL	84
		3 0	0	0	Y	G	1	0	5	0	2	1	7	0		COMPARE TAPE LABELS	85
		4 0	0	0	W	1	4	3	1	0	4	3	1	0		IF UNLIKE → 164	85
		5 0	0	0	J	0	1	8	0	0	1	8	0	4		CLR INPUT RECORD CTR	86

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	346 0	0	0	J	0	1	8	1	5	1	8	2	4		CLR INPUT HASH AREA	87
		7 0	0	0	V	1	0	2	1	9	2	5	4	0		TRANS → 7	
2510		8 0	0	0	Y	1	1	1	0	0	3	2	4	1		SENSE FOR E F IN MAST READ AREA	90
		9 0	0	0	W	1	3	5	1	0	3	5	1	0		IF NOT → 91	90
		350 0	0	0	V	1	0	2	1	9	3	6	4	0		TRANS → 101	91
3490		1 0	0	0	M	&	3	5	3	0	2	5	8	0		PLACE A STR REG INST OVER 9 TC → 31	92
	6	352 0	0	0	V	1	0	2	1	9	2	5	3	0		TRANS A	93
		3 0	0	0	V	1	0	2	1	9	2	8	2	0		INSTRUCTION CHANGE TC → 31	
		4 0	0	0	R	E	C	O	R	D	-	C	O	U			
		5 0	0	0	N	T	-	X	X	X	X	X	-	H			
		6 0	0	0	A	S	H	-	C	O	U	N	T	-			
		7 0	0	0	X	X	X	X	X	X	X	X	X	X			
	6	358 0	0	0	-	S	U	M	-	O	F	-	T	O		FINAL TRAILER RECORD	
		9 0	0	0	T	A	L	-	G	R	O	S	S	-			
		360 0	0	0	X	X	X	X	X	X	X	X	X	X			
		1 0	0	0	X	X	-	-	-	-	2	7	5	0		LOCATION OF 25	
3260		2 0	0	0	Y	1	1	3	0	0	3	2	4	1		SENSE FOR E F IN TRANS READ INN AREA	100
		3 0	0	0	W	1	3	9	4	0	3	9	4	0		IF NOT → 128	100
3500	6	364 0	0	0	N	1	2	5	4	1	3	6	6	1		TRANS MASTER TAPE # → 103	101
		5 0	0	0	N	1	2	5	4	1	3	8	0	1		" " " 114	102
		6 0	0	0	4	(1)	1	4	0	0	1	4	3	9		READ MASTER TRAILER	103
		7 0	0	0	Y	5	1	4	1	3	1	8	0	0		COMPARE RECORD COUNT	104
		8 0	0	0	W	1	4	1	7	0	4	1	7	0		IF UNLIKE → 151	104
		9 0	0	0	Y	&	1	4	3	0	1	8	1	5		COMPARE HASH COUNT	105

XXVIII-28

IE 1224 REV. 8-60

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	370 0	0	0	W	1	4	1	7	0	4	1	7	0		IF UNLIKE →151	105
		1 0	0	0	N	1	2	5	0	1	3	7	3	1		TRANSFER TRANS TAPE UNIT →108	106
		2 0	0	0	N	1	2	5	0	1	3	8	1	1		" " " " →115	107
		3 0	0	0	4	3	1	4	7	5	1	4	9	1		READ TRANS TAPE LABEL	108
		4 0	0	0	Y	5	1	4	7	5	1	8	1	0		COMPARE RECORD COUNT	109
		5 0	0	0	W	1	4	3	9	0	4	3	9	0		IF UNLIKE →170	109
	6	376 0	0	0	N	B	1	8	4	6	1	5	4	1		TRAN Σ TOT GR WAΣTG	110
		7 0	0	0	⊖	B	1	5	4	1	1	4	7	1		SUB WAΣTG ⊖ TOT GR→IN MAST TRAILER	111
		8 0	0	0	⊖	B	1	5	4	1	1	4	9	1		SUB WAΣIG ⊖ TRANS AMT	112
		9 0	0	0	W	1	4	5	6	0	4	5	6	0		SENSE PRIS	113
4670		380 0	0	0	;	(1)	0	0	0	0	0	0	0	0	102	REWIND MASTER	114
		1 0	0	0	;	(3)	0	0	0	0	0	0	0	0	107	REWIND TRANSACTION	115
	6	382 0	0	0	N	5	1	8	0	9	3	5	5	7		TRANS OUTPUT RECORD COUNT →FINAL TRAILER	116
		3 0	0	0	N	&	1	8	3	4	3	5	7	9		TRANS OUTPUT HASH COUNT →FINAL TRAILER	117
		4 0	0	0	N	B	1	8	4	6	3	6	1	1		TRANS ΣTG →OUTPUT FINAL TRAILER	118
		5 0	0	0	N	1	2	6	9	1	3	8	9	1		TRANS OUTPUT TAPE NO →123	119
		6 0	0	0	N	1	2	6	9	1	3	9	0	1		" " " " →124	120
		7 0	0	0	N	1	2	6	9	1	3	9	1	1		" " " " →125	121
	6	388 0	0	0	N	1	2	6	9	1	3	9	2	1		" " " " →126	122
		9 0	0	0	8	(2)	3	2	4	1	3	2	4	1	119	WRITE EF	123
		390 0	0	0	8	(2)	3	2	4	0	3	6	1	1	120	WRITE TRAILER	124
		1 0	0	0	8	(2)	3	2	4	0	3	2	4	0	121	WRITE ED	125
		2 0	0	0	;	(2)	0	0	0	0	0	0	0	0	122	REWIND MASTER	126
		3 0	0	0	⊡	0	0	0	0	0	0	0	0	0		END OF RUN	127

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
3630	6	394	0	0	N	4	3	6	1	9	2	5	3	9		SET A ⟶ 25	128
		5	0	0	V	1	0	2	1	9	2	7	5	0		TRANS ⟶ 25	
2310		6	0	0	;	1	0	0	0	0	0	0	0	0		REWIND WRONG MASTER TAPE	130
		7	0	0	⊡	1	2	0	0	1	0	0	0	0		HALT	131
		8	0	0	;	1	0	0	0	0	0	0	0	0		REWIND NEW MASTER TAPE	132
		9	0	0	V	1	0	2	1	9	2	2	9	0		TRANS ⟶ 1.06	
2350	6	400	0	0	;	3	0	0	0	0	0	0	0	0		REWIND WRONG TRANSACTION TAPE	135
		1	0	0	⊡	1	2	0	0	3	0	0	0	0		HALT	136
		2	0	0	;	3	0	0	0	0	0	0	0	0		REWIND NEW TRANSACTION TAPE	137
		3	0	0	V	1	0	2	1	9	2	3	3	0		TRANS ⟶ 1.09	
		4	0	0	;	2	0	0	0	0	0	0	0	0		REWIND OUTPUT TAPE - NOT AVAILABLE	140
		5	0	0	⊡	1	3	0	0	2	0	0	0	0		HALT	141
	6	406	0	0	;	2	0	0	0	0	0	0	0	0		REWIND NEW OUTPUT TAPE	142
		7	0	0	V	1	0	2	1	9	2	3	6	0		TRANS ⟶ 1.11	
3170		8	0	0	N	1	3	1	5	1	4	0	9	1		TRANS TAPE UNIT ⟶ 145	144
		9	0	0	;	(5)	0	0	0	0	0	0	0	0		REWIND OUTPUT TAPE - NOT AVAIABLE	145
		410	0	0	⊡	1	3	0	2	5	0	0	0	0		HALT	146
		1	0	0	V	1	0	2	1	9	3	1	4	0		TRANSFER ⟶ 61	
3260	6	412	0	0	⊡	1	1	0	0	1	0	0	0	0		HALT — ED/EF SET BUT NOT PRESENT	150
		3	0	0	M	A	S	T	E	R	-	T	R	A			
		4	0	0	I	L	E	R	-	I	N	P	U	T		CONSTANTS	
		5	0	0	-	C	O	U	N	T	E	R	H	A			
3680 3700 3290 3310		6	0	0	S	H	-	C	O	U	N	T	E	R			
		7	0	0	N	4	0	2	1	9	4	3	0	9		TRANS STP ⟶ 163	151

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	418 0	0	0	J	-	1	6	0	0	1	7	1	9		CLR PRINT AREA	152
		9 0	0	0	B	0	0	0	0	0	1	6	0	3		PAGE CHANGE	153
		420 0	0	0	M	D	4	1	3	0	1	6	0	0		TRANS "MASTER TRAILER" → PRA	154
		1 0	0	0	M	3	2	1	7	8	1	6	1	5		TRANS REEL NUMBER → PRA	155
		2 0	0	0	M	"	1	4	0	0	1	6	1	9		TRANS TRAILER RECORD → PRA	156
		3 0	0	0	M	&	1	4	3	0	1	6	4	9		TRANS TRAILER RECORD → PRA	156
	6	424 0	0	0	M	C	4	1	4	5	1	6	6	1		TRANS "INPUT COUNTER" RECORD → PRA	157
		5 0	0	0	N	5	1	8	0	4	1	6	7	9		TRANS CONTENTS OF INPUT CTR → PRA	158
		6 0	0	0	M	B	4	1	5	8	1	6	8	1		TRANS "HASH COUNTER" → PRA	159
		7 0	0	0	N	&	1	8	2	4	1	7	0	3		TRANS COUNTERTS OF HAST COUNTER → PRA	160
		8 0	0	0	B	2	0	0	0	0	1	6	0	1		PRINT ERROR LINE	161
		9 0	0	0	⊡	1	4	0	0	1	0	0	0	0		HALT	162
	6	430 0	0	0	V	1	0	2	1	9	(0	0	0	0)		TRANS → 106 73 / 105 74	
3440		1 0	0	0	N	1	3	4	2	1	4	3	3	1		TRANS TAPE UNIT → 166	164
		2 0	0	0	N	1	3	4	2	1	4	3	5	1		TRANS TAPE UNIT → 168	165
		3 0	0	0	;	(0)	0	0	0	0	0	0	0	0		REWIND WRONG MASTER TAPE	166
		4 0	0	0	⊡	1	2	0	1	4	0	0	0	0		HALT	167
		5 0	0	0	;	(0)	0	0	0	0	0	0	0	0		REWIND NEW MASTER TAPE	168
	6	436 0	0	0	V	1	0	2	1	9	3	4	2	0		TRANS → 84	
		7 0	0	0	T	R	A	N	S	A	C	T	I	O		CONSTANTS	
		8 0	0	0	N	-	T	R	A	I	L	E	R	-			
3750		9 0	0	0	J	-	1	6	0	0	1	7	1	9		CLEAR PRINT AREA	170
		440 0	0	0	B	0	0	0	0	0	1	6	0	3		PAGE CHANGE	171
		1 0	0	0	M	I	4	3	7	0	1	6	0	0		TRANS "TRANSACTION TRAILER" → PR. A.	172

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	4420	0	0	N	5	1	4	7	9	1	6	2	4		TRANS TRAILER COUNT → PRA	173
		30	0	0	M	C	4	1	4	5	1	6	2	6		TRANS "INPUT COUNTER" → PRA	174
		40	0	0	N	5	1	8	1	4	1	6	4	4		TRANS TRANSACTION INPUT COUNT — PRA	175
		50	0	0	B	2	0	0	0	0	1	6	0	1		PRINT	176
		60	0	0	□	1	4	0	0	2	0	0	0	0		HALT	177
		70	0	0	V	1	0	2	1	9	3	7	6	0		TRANS → 110	
	6	4480	0	0	C	O	N	T	R	O	L	-	T	O			
		90	0	0	T	A	L	S	N	E	W	-	S	U			
		4500	0	0	M	-	O	F	-	T	O	T	A	L			
		10	0	0	-	G	R	O	S	S	O	L	D	-		CONSTANTS	
		20	0	0	S	U	M	-	O	F	-	T	O	T			
		30	0	0	A	L	-	G	R	O	S	S	T	R			
	6	544	0	0	A	N	S	A	C	T	I	O	N	-			
		50	0	0	A	M	O	U	N	T	-	-	-	-			
4790		60	0	0	J	-	1	6	0	0	1	7	1	9		CLEAR PRINT AREA TO SPACES	180
		70	0	0	B	0	0	0	0	0	1	6	0	3		PAGE CHANGE	181
		80	0	0	M	D	4	4	8	0	1	6	0	0		TRANSFER "CONTROL TOTALS" → PR A	182
		90	0	0	M	L	4	4	9	4	1	6	1	3		TRANS "NEW SUM OF TOTAL GROSS" → PR A	183
	6	4600	0	0	M	L	4	5	1	6	1	6	5	1		TRANS "OLD SUM OF TOTAL GROSS" → PR A	184
		10	0	0	M	H	4	5	3	8	1	6	8	7		TRANS "TRANSACTION AMOUNT" → PR A	185
		20	0	0	N	B	1	8	4	6	1	5	4	9		TRANS Σ TOT GR → PR A	186
		30	0	0	N	B	1	4	7	1	1	6	8	5		TRANS OLD Σ T G → PR A	187
		40	0	0	N	B	1	4	9	1	1	7	1	7		TRANS TRANSACTION AMOUNT → PR A	188
		50	0	0	B	2	0	0	0	0	0	6	0	1		PRINT	189

XXVII-32

IE 1224 REV. 8-60

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	2	466	0	0	⊡	1	4	0	0	3	0	0	0	0		HALT	190
		70	0	0	V	1	0	2	1	9	3	8	0	0		TRANS → 114	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

OPERATIONAL INFORMATION:

TAPE UNITS:

1 for first Master Tape; alternate 4
2 for first Output; alternate 5
3 for Transaction tape

TAPES:

Master Tape Label: MASPAYRL
Transaction Tape Label: PYRLTRNS

ADDITIONAL INPUT:

Card in format:

cols.	1-6	7-12	13-18	19-24
	Today·s Date	Master·s Processing Date	Transaction·s Processing Date	New Purge Date

PROGRAM LIMITS:

2140-4719

START LOCATION:

2240

STOP LOCATIONS:

END OF RUN at 3930

A REGISTER:	REASON:
1001	LOGIC ERROR, ED/EF INDICATOR SET BUT ED/EF NOT PRESENT, NOTIFY ENGINEER
2001	TAPE LABEL INCORRECT-MASTER ON TAPE 1 (INITIAL)
2003	TAPE LABEL INCORRECT-TRANSACTION ON TAPE 3(INITIAL)
2014	TAPE LABEL INCORRECT-MASTER ON TAPE 1 or 4
3002	PURGE DATE GREATER THAN TODAY·S DATE-TAPE 5 (INITIAL)

A REGISTER: REASON: (Cont·d.)

3025	PURGE DATE GREATER THAN TODAY' S DATE-TAPE 2 or 5
4001	MASTER TRAILER INCORRECT AS TO COUNT AND HASH
4002	TRANSACTION TRAILER INCORRECT AS TO COUNT
4003	CONTROL TOTALS PROVE INCORRECT

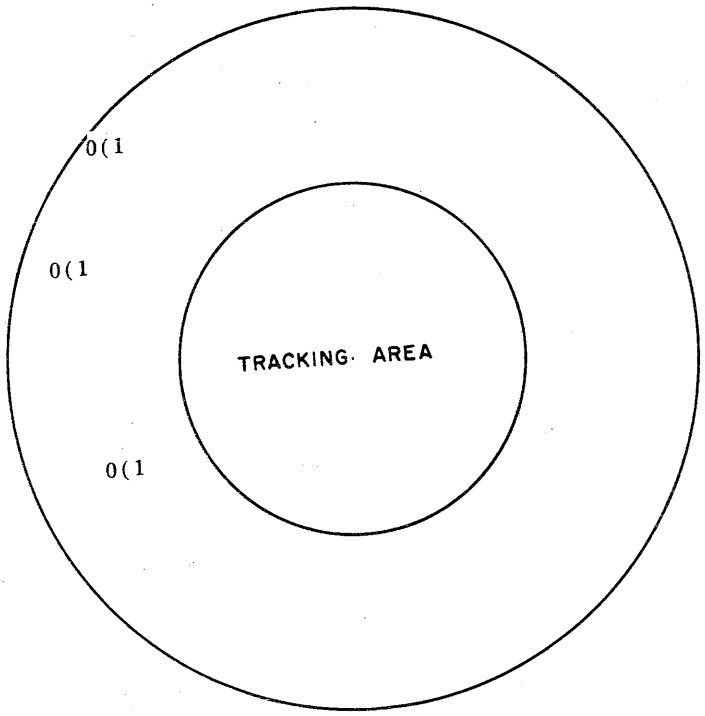
CORRECTION PROCEDURES: FOR ALL ERRORS CODED 2-4, NOTE CONDITION AND RESTART BY HITTING START BUTTON. ERROR CODED 1, MACHINE FAILURE, NOTIFY ENGINEER AND RESTART PROGRAM FROM BEGINNING.

XXIX – DATA RECORD FILE

Although we mentioned the Data Record File in the early chapters of this text, we have not been utilizing it in our programs to date. This storage device has many important applications, however, so it is very important for us to be completely familiar with it.

As was stated earlier, each single data record file, with which we are concerned in this chapter, is a magnetic disc storage device. It can hold up to 128 discs at one time in a basket, which simply acts as a holding device for the records.

Each disc has two faces, and each face has two bands. These bands are arranged in parallel spiralling in toward the center:



The arm has two vertical contacts, one which physically rests on the record in the tracking area to insure proper reading and writing, and the second which is actually two read-write heads, one for each band.

When a band is requested by the program, the basket turns to the proper slot, the record is extracted and placed on a turntable with the proper side up, the arm is placed on the record so that the read-write heads are at the beginning of the bands, and the proper read-write head is activated. Then, again by program control, information can be read from the band or written to it.

So far we have discovered that there are 128 records, each with four bands (two per side), yielding 512 bands which are numbered from 000 to 511. Each band, in turn, contains 10 cells and these cells are separated by inter-cell gaps. Each cell has the ability to hold up to 900 characters of information, and the inter-cell gap is approximately the length of 100 characters. It is possible to address any one cell within a band, or any group of consecutive cells. The cells are numbered 0 - 9.

The following instructions are used with the Data Record File:

BAND SELECT NORMAL:

The instruction initiates a search of the Data Record File for the disc containing the desired band, puts the disc on the turntable, positions the arm, and activates the proper read-write head. Once initiated, the disc and the band will be selected independent of computer operation.

The OP code is a D.

N is used to indicate whether or not the proper disc face is positioned on the same face. If there is a 1 bit in 2^0, this indicates that the proper disc face is present on the turntable, and the B_3 character is then examined to discover if it is odd or even. If B_3 is odd, the read-write head is activated for the odd numbered band. If B_3 is even, the even read write head is activated. If there is a 0 bit in 2^0, this indicates to the Record File that the disc on the turntable (if there is one there) is not the one desired and it will be placed back into the basket before the Select begins. In addition, a 0 or 1 in the 2^4 bit will indicate the Data Record File to be used.

	Correct disc	New disc
R_F 1	1	0
R_F 2	A	&

The A address is ZEROS.

The B address contains, in B_1, 2, and 3, the number of the desired band. B_0 is ZERO(0).

Examples:

D 0 0000 0412

Any disc now on the turntable of the first Data Record File will be placed in the basket and the disc containing band 412 will be selected and placed on the turntable. The first read-write head will be activated and positioned at the beginning of the band.

D 1 0000 0413

The disc on the turntable is indicated to be the correct one, so the only operation is to activate the second (or odd) read-write head.

In timing this operation, we must remember that the computer is completely free to continue operation while the select is being accomplished. The average select time, which means that the basket must turn 90° (the worst possible case is a 180° turn, as the basket can rotate in either direction and will automatically select the shorter path), is 3.1 seconds. If there is a disc on the turntable when the select is begun and if this disc must be replaced in the basket before the select can be started, this will take an additional 1.6 seconds. Obviously, if the band desired is on the opposite side of the disc on the turntable or on an adjacent disc the access time is reduced. It will take 3.4 seconds to select the disc if the band is on the opposite side and .9 seconds if the adjacent band is desired. It takes 1.6 seconds to select an adjacent disc.

BLOCK READ RECORD NORMAL:

Once the proper band has been selected, information will either be read from it or written to it. The BLOCK READ FROM RECORD NORMAL instruction reads from a selected cell of the band to a designated HSM location. From one to ten consecutive blocks of information may be transferred from the band to memory with one instruction. As was stated before, a cell can contain up to 900 characters. If less information is kept in the cell (for example, 750 characters) the end of the block of data should be marked with a control symbol called an End of Block (EB).

The OP code is an F.

The N character indicates the number of cells to be read from the band. This is expressed by 1-9, with 0 equalling 10. In addition, a 0 or 1 in the 2^{4} position indicates which Data Record File.

The A address gives the HSM location to receive the first character.

The B address contains a number of different pieces of information:

B_0 is ZERO(0)

B_1 determines whether or not the disc is to remain on the turntable. if B_1 is 1 the disc is returned to the cage. If it is 0 the disc stays on the turntable and the arm is placed at the beginning of the band.

B_2 determines what is to stop the read. If B_2 is 1 the entire 900 characters in the cell will be read in. If it is 0 the read terminates when an EB is sensed or when the 900 characters have been read in, whichever occurs first.

The B address (Cont'd)

B_3 addresses the first cell to be read into memory. The cells are numbered sequentially from 0-9.

Suppose the data we wanted was in cell 4, and we wanted the full 900 characters. After reading the cell, we will be doing some updating, and then will place the updated information back into the cell, so we will not want the disc returned to the basket. The read instruction will be:

F 1 1500 0014

This tells the computer to read in 1 cell starting at cell 4 placing the data starting at HSM location 1500. The full 900 characters are to be read in, and the disc is to remain on the turntable at the end of the operation.

STA, upon termination of the instruction, will hold the address one to the right of the last character read in.

Characters are passed over and read into memory at a rate 2500 characters per second. In our example, we must pass over 4 cells to reach the fifth (cell 4) and then read in the fifth cell. This means that just less than 5000 character locations must be processed, and this will take approximately 2 seconds.

BLOCK WRITE TO RECORD NORMAL:

This instruction writes from a designated HSM location to designated cells of a selected band. From one to ten blocks of information may be transferred to the band with one instruction.

The OP code is H.

N gives the number of blocks to be written and the Data Record File. The number of blocks can vary from one to ten, with ten being specified by 0.A 1 or 0 in 2^4 indicates the device.

The A address gives the location of the first character to be written.

The B address expresses the same information indicated by the B address of the READ.

B_0 is ZERO(0).

B_1 determines if the disc is to stay on the turntable after the operation (1, returns to basket; 0, stays and arm positioned at the beginning of the band).

B_2 determines the terminating conditions (1, 900 count; 0, 900 count or EB, whichever appears first).

B_3 addresses the first cell to receive data (0-9).

The rate is again 2.5 KC (2500 characters per second).

DATA RECORD FILE TIMING

Character transfer rate - 2.5KC

Maximum number of characters/read-write access - 9,000

Band Select Timing

A. Processor time to initiate selection (in microseconds) 21 (B_1) + 14 (B_2) + 42 = (staticizing time + establishing address counter).

B. Select data record from cage - since the 128 position data record storage wheel rotates in either direction, the maximum data record position from the selection point is 64.

# of data record positions rotated	Time (in seconds)*
1	1.6
2	1.8
3	2.0
4	2.2
5	2.4
6	2.6
7	2.8
8	3.0
9	2.65
16	2.8
32	3.1
64	3.4

*Variance ± 10%

Note: The cage rotation proceeds at low speed from one to eight data record positions from the selection point. The cage rotation proceeds at fast speed from nine to sixty-four data record positions from the selection point. Use straight line interpolation for interim positions on the table.

C. To select a band on the opposite side of a data record on the turntable (head at ready to head at ready) = 3.4 seconds.

D. To select the adjacent band of a data record on the turntable and return the head to ready = 0.9 seconds.

Read or Write Timing

A. The data record is positioned on the turntable and the read/write arm is in the "ready" position.

1. Read/write arm transferred to data record = .62 to .85 sec.

2. Latency (to reach the first character of the first cell on on the data record) = .2 to .4 seconds.

3. Inter-cell gap (fixed time between 900-character cells) = .04 seconds

4. Transfer time per cell (characters and gap) = .4 seconds.

B. To return the data record to the record storage wheel

1. If the read/write arm is on the data record = 1.6 seconds

2. If the read/write arm is in the "ready" position = 1.84 seconds

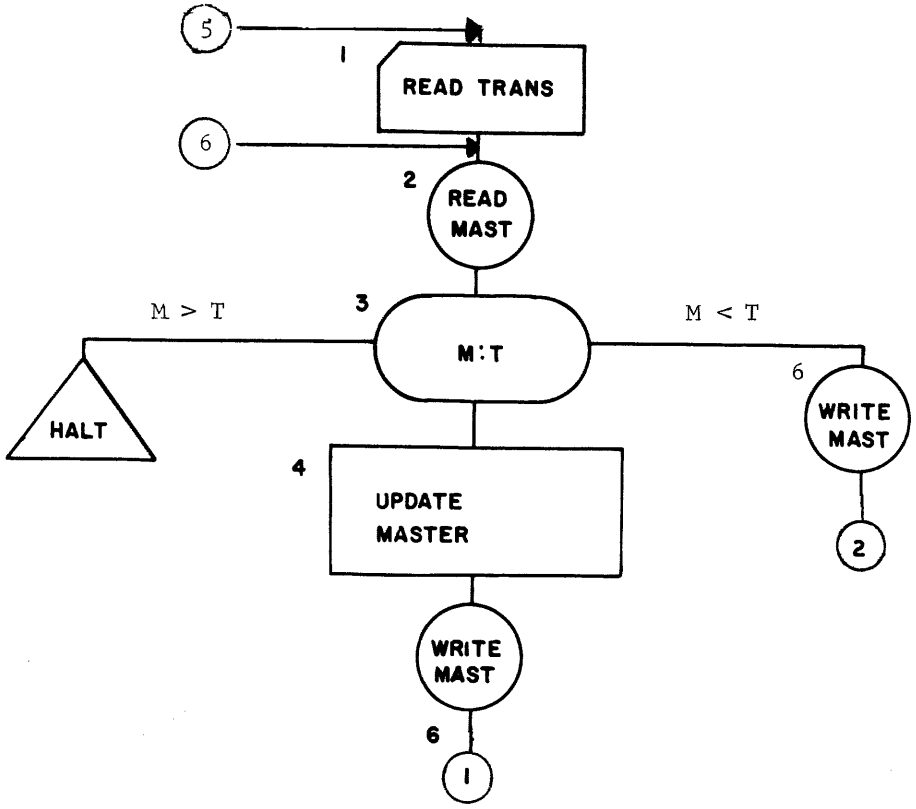
There are many applications for the Data Record File, a primary one being the storage of all the programs of the installation and the ability to sequence these programs. (This will be covered in more detail in the chapter on Supplied Routines).

Updating can be handled on the Data Record File, in a manner similar to using magnetic tape, but it must be remembered that the access time and the read-write rate are considerably slower, so that the program running time will be increased. Here it is a matter of balancing dollars and minutes in order to come up with the most efficient system.

Using as an example an updating run, we can become familiar with the technique of using the data record as well as the time considerations.

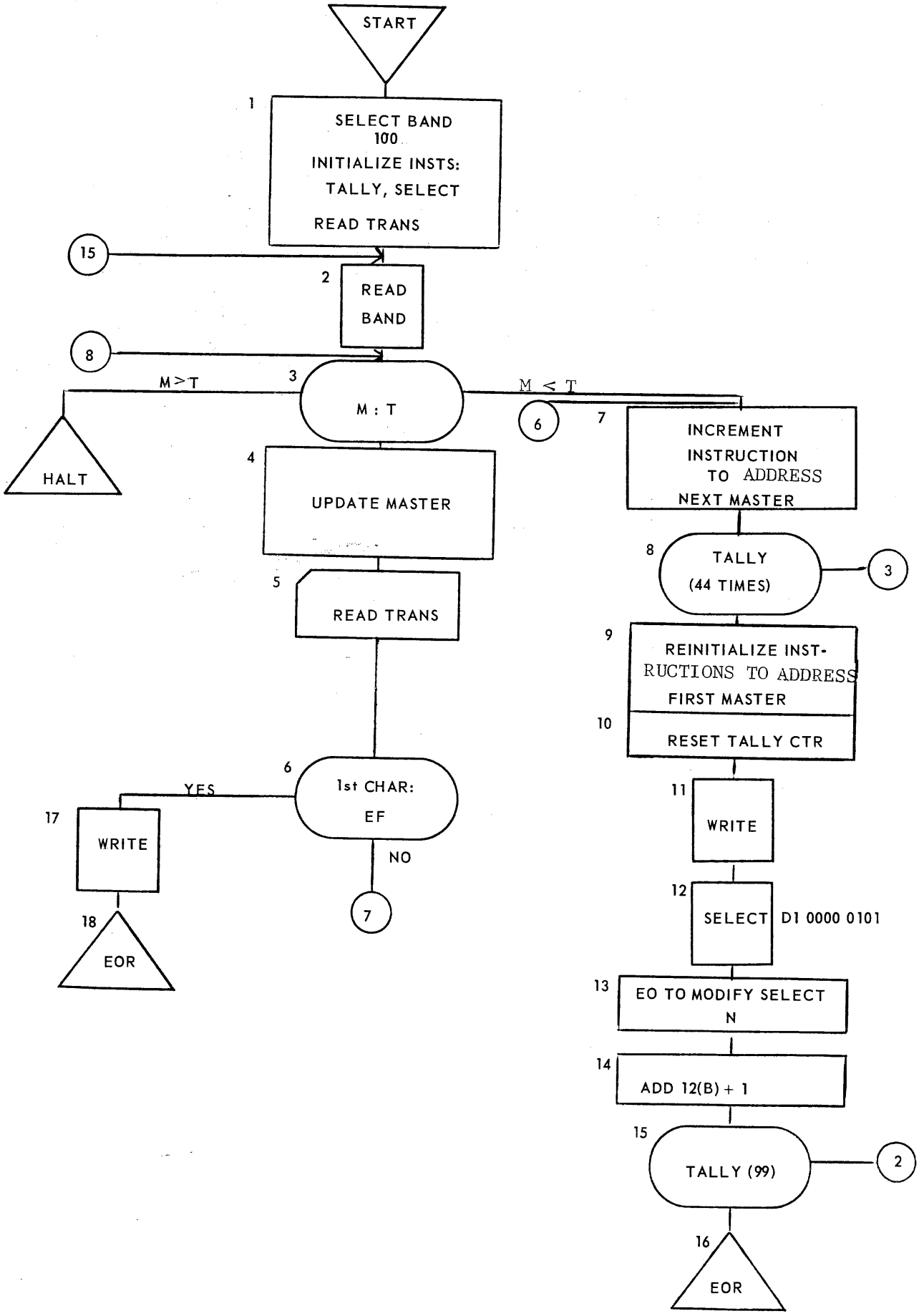
The problem is as follows:

There are 4500 one-hundred character records to be updated by no more than one transaction apiece. 4000 transactions are coming in from cards and are in a completely sorted order. We are to HALT if any are out of sort. If we were to handle this problem using magnetic tape, it would appear flow-charted as follows:



If, however, the masters are arranged on the Data Record File so that 9 records are maintained (in fixed field) in a cell, we must program this slightly differently in order to bring a new group of cells when needed, and to allow for the most efficient program.

If we had a 20,000 character location memory, we would have enough room to bring in the entire band at one time. Suppose, however, we are limited to a 10K memory. It will be necessary to bring in only a portion of the band, process that, write it out and then bring in the next portion of the same band. Since the arm is positioned at the beginning of the band at the end of the operation (if we request that the disc remain on the turntable), it will mean that we must pass over the first portion in order to reach the second position. For example, if we brought in the first 5 cells (app. 2 seconds), the second 5 cells would take 4 seconds. To avoid this, it is obvious that if the data could be placed at the beginning of the band, the time could be greatly reduced. In our problem, let's stipulate that the data is packed in the first 5 cells of every band and it will therefore take 100 bands to hold the full 4500 records. The bands containing this data are 100-199. We will assume that there is only one record control in our system. Assume an EF card at end of transactions.


START
1
SELECT BAND 100
INITIALIZE INSTS: TALLY, SELECT
READ TRANS
15
2
READ BAND
8
3
M : T
M>T
HALT
M < T
6
7
INCREMENT INSTRUCTION TO ADDRESS NEXT MASTER
4
UPDATE MASTER
8
TALLY (44 TIMES)
3
5
READ TRANS
9
REINITIALIZE INSTRUCTIONS TO ADDRESS FIRST MASTER
10
RESET TALLY CTR
6
1st CHAR: EF
YES
17
WRITE
18
EOR
NO
7
11
WRITE
12
SELECT D1 0000 0101
13
EO TO MODIFY SELECT N
14
ADD 12(B) + 1
15
TALLY (99)
2
16
EOR

When the first select instruction is done, it will take (average) 3.1 seconds. The Read (F 5 5000 0010) will bring in 45 masters. The comparison begins with the first master, the criterion of which is located starting at 5000. If the master and transaction match, the master is updated and a new transaction is read in. Since we now have finished with the old master, we can increment the compare (and all the other instructions which refer to the master) by 100 locations, so that the second master is addressed (the criterion will be at 5100, etc.) When we have processed all 45 masters in a similar fashion, the tally will be exhausted and we will reset all the instructions to again address the first master in the read in area. We will also place 44 back into the Tally Counter. Writing the 5 cells back out, we can select the next band. Since every other band will be on the disc face, we can arrange to have an N of 1 in the Select instruction the first time, allowing us to address band 101, and alternate 0, 1, 0, 1 each time from there on in.

The last Tally (step 15) or sensing the EF card will allow us to terminate the program. Ignoring the computation, which will be increased due to the incrementing and the tally, we can see that the input-output time for the master is increased.

10KC Mag. Tape		Record File		
READ	1 min. 32 sec.	First select -		3.1 sec.
WRITE	4 min. 21 sec.	Select Adjacent band - .9 sec x 100		90 sec.
	5 min. 53 sec.*	Select band on opposite side.	-3.4 sec x 50	170 sec.
		Select new RCD -	1.6 sec (replace used RCD)	
			1.6 sec (select next position)	
			1.0 sec (ARM movement)	
			4.2 sec x 49 - 20	sec.
		All reads - 2 sec. x 100		200 sec.
		All writes- 2 sec. x 100		200 sec.
				683.1 sec.
		or 11 min. 38 sec.		

*NOT BATCHED

In addition to sequential updating on the Data Record File, there are many schemes for "random access"; that is, locating just the information that is needed at this particular time. These techniques will be discussed in more detail in the chapter on Randomizing, but right now we will take up the simplest of these methods, called "direct addressing".

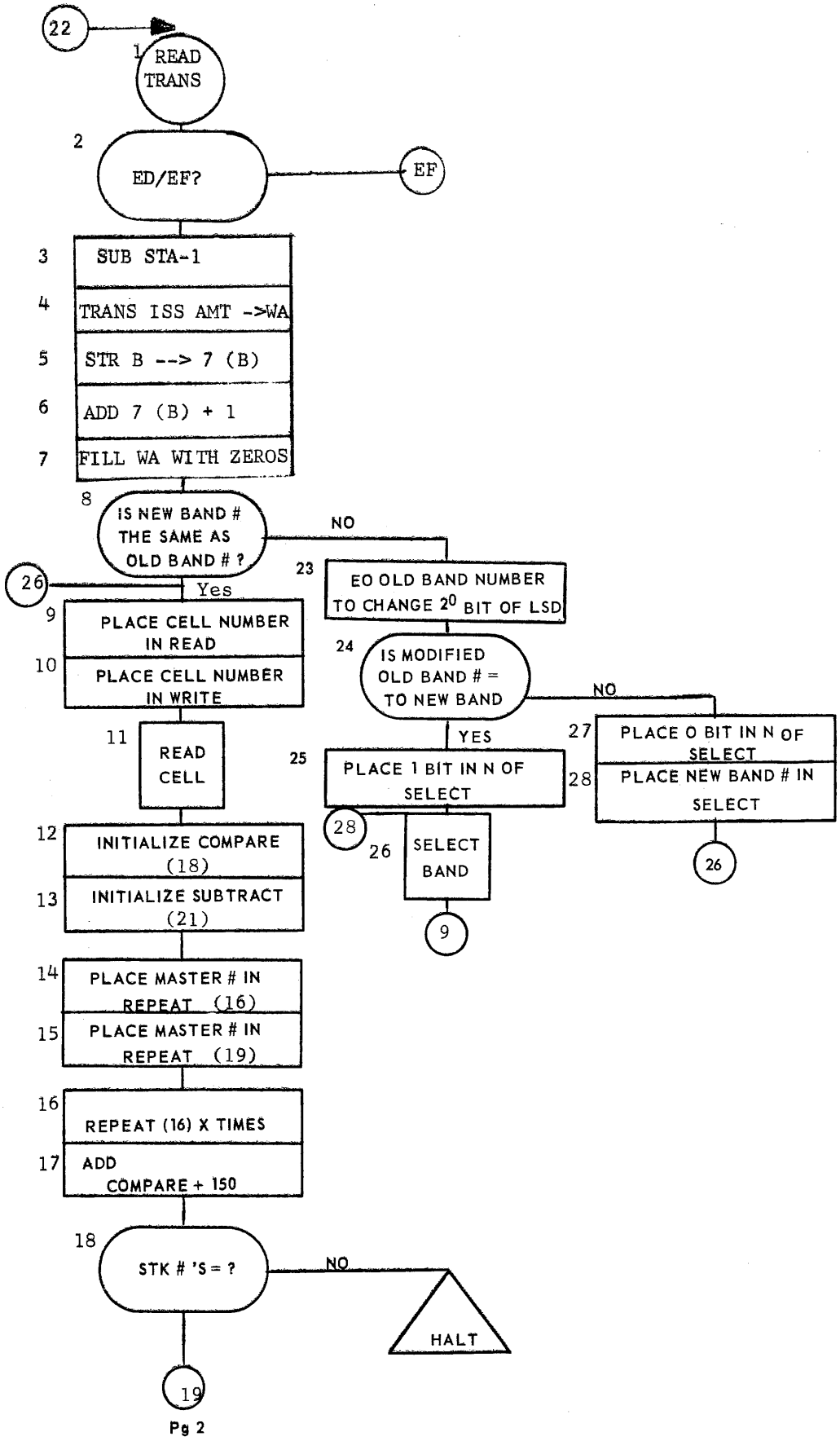
Suppose we had the ability to place information on discs in the data record file, and then compose an identifying code that was not only the unique code for that particular information, but also was the address of its storage cell. Obviously, we could then simply use this code to pick up the proper disc, activate the proper band's read-write head, and read in the appropriate cell.

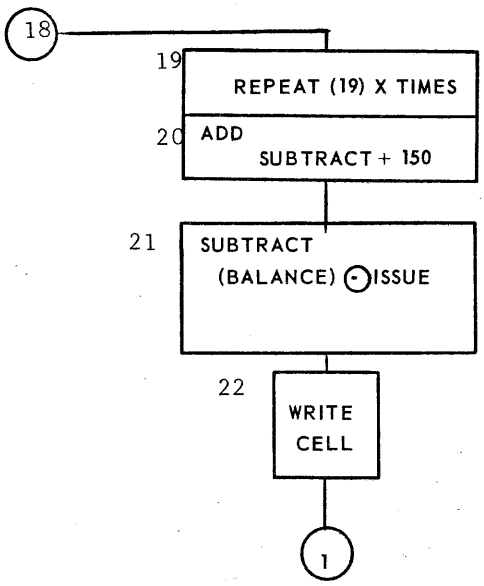
For example, we are working for an installation that is perfectly willing to set up a new series of stock numbers. Each master record contains 150 characters and since there are 12000 of these masters, it will require 200 bands (50 discs) to accomodate them. When these records were written to the data record file, they were placed on bands 100-299. In addition, as they were assigned,

the band number, cell number, and a positional number (to indicate the position within the cell) were made into an identifying data stock number:

Examples: 15964 is the stock number of the master located on band 159, in cell 6, and it is the 5th master in from the beginning of the cell (to be consistant with the numbering of the bands and the cells, the masters are numbered 0-5).

Suppose that any one day we might have 1000 issues coming in in random order, containing a 5 digit stock number and a 3-10 digit issue amount proceeded by an ISS. In addition, we will assume that the first five digits of each master is the stock number, and that the balance is the last 10 digits. Our program must subtract the issue amount from the proper balance.


22
1
READ TRANS
2
ED/EF?
EF
3
SUB STA-1
4
TRANS ISS AMT ->WA
5
STR B --> 7 (B)
6
ADD 7 (B) + 1
7
FILL WA WITH ZEROS
8
IS NEW BAND # THE SAME AS OLD BAND # ?
NO
Yes
26
23
EO OLD BAND NUMBER TO CHANGE 2^0 BIT OF LSD
24
IS MODIFIED OLD BAND # = TO NEW BAND
NO
YES
27
PLACE 0 BIT IN N OF SELECT
28
PLACE NEW BAND # IN SELECT
26
25
PLACE 1 BIT IN N OF SELECT
28
26
SELECT BAND
9
9
PLACE CELL NUMBER IN READ
10
PLACE CELL NUMBER IN WRITE
11
READ CELL
12
INITIALIZE COMPARE (18)
13
INITIALIZE SUBTRACT (21)
14
PLACE MASTER # IN REPEAT (16)
15
PLACE MASTER # IN REPEAT (19)
16
REPEAT (16) X TIMES
17
ADD COMPARE + 150
18
STK # 'S = ?
NO
HALT
19
Pg 2



By examining the program, we can see that an effort is being made to cut down on unnecessary Select time. If the new band number and the old band number do not agree (if they did, no Select would be necessary), one further test is made, and that is to see if the new band is on the same disc face as the old band. This is done by changing the 2^0 bit of the least significant digit of the old band number and then comparing this modified band number to the new band number. If they are equal, the Select need only activate the alternate read-write head. If they are still unequal, it will be necessary to replace the current disc and Select an entirely new disc, in order to obtain the proper band. For example:

1) old band number 135 new band number 134

least significant digit (in binary)	000101
modify 2^0 using EO	000001
modified digit	000100

modified band number, 134, which equals new band number, proving that both bands are on the same disc face.

2) old band number 266 new band number 267

least significant digit	000110
modifying character	000001
modified digit	000111

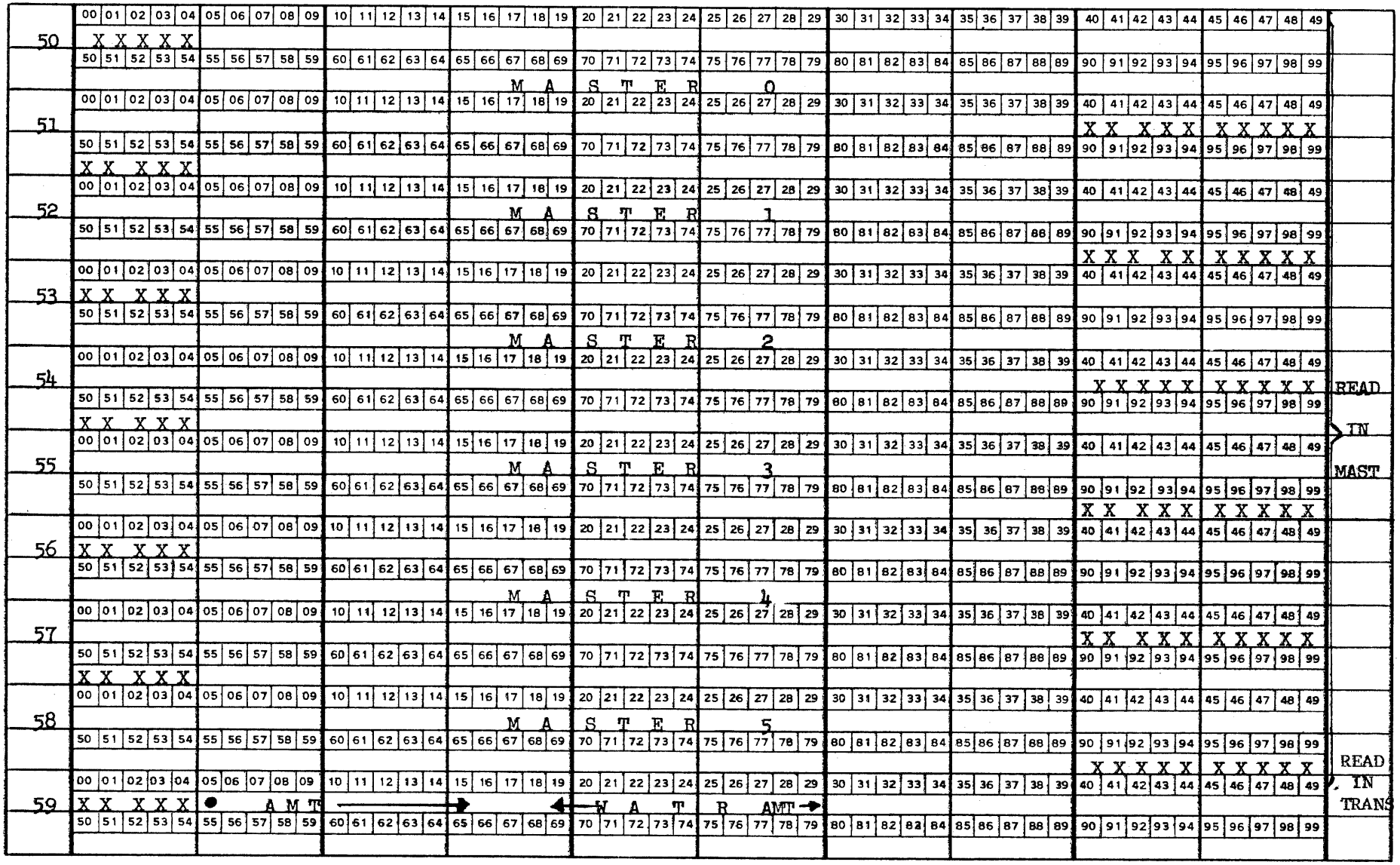
modified band number, 267, which equals the new band number.

3) old band number 197 new band number 198

least significant digit	000111
modifying character	000001
modified digit	000110

modified band number 196; new band number 198; therefore, a new band on a different disc face is needed.

It is also possible to perform the Block Read from Record and the Block Write to Record in the Simultaneous Mode (remember that the Select is free of computer control). When using the Simultaneous Mode, Memory must be accessed 7 us. out of every 400 us. This forms an available time percentage of 98.25%, which must be considered if computation is going on in the Normal Mode while characters are actually being read and transferred by the Simultaneous Mode.



XXIX-14

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___

FORM NO. 1223 25M 6-60

TITLE Record File Example 1

CODER

REMARKS

DATE

SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	3	600 0	0	0													
		1 0	0	0													
		2 0	0	0													
		3 0	0	0	4	9	9	9	4	8	5	0	0	1		CONSTANTS	
		4 0	0	0	0	1	0	1	0	0	0	1	5	0			
6290		5 0	0	0	4	8	5	9	0	0	5	9	1	5		READ TRANSACTION	1
	6	606 0	0	0	W	8	ED/EF				6	0	7	0		ED/EF SENSE	2
		7 0	0	0	0	2	0	2	1	5	6	0	3	9		SUBTRACT 1 FROM STA	3
		8 0	0	0	P	•	0	2	1	E	5	9	2	9		TRANSFER ISSUE AMOUNT TO WORK AREA (STA --IND ADD)	4
		9 0	0	0	V	4	6	1	1	9	0	0	0	0		STORE B IN 7 (B)	5
		610 0	0	0	+	2	6	1	1	9	6	0	3	9		ADD 1 TO 7 (B)	6
		1 0	0	0	J	0	5	9	1	8	(0	0	0	0)		FILL WA ANT WITH INSIG ZEROS	7
	6	612 0	0	0	Y	3	5	9	0	0	6	3	4	7		COMPARE NEW BAND # : OLD BAND #	8
		3 0	0	0	W	1	6	3	0	0	6	3	0	0		UNEQUAL → 23	8
6350		4 0	0	0	N	1	5	9	0	3	6	1	6	9		TRANS CELL # READ (11)	9
		5 0	0	0	N	1	5	9	0	3	6	2	8	9		TRANS CELL # WRITE (22)	10
		6 0	0	0	F	1	5	0	0	0	0	0	1	(0)	8	READ CELL	11
		7 0	0	0	N	4	6	0	3	7	6	2	3	5		INITIALIZE COMPARE (18)	12
	6	618 0	0	0	N	4	6	0	3	3	6	2	7	5		INITIALIZE SUBTRACT (21)	13
		9 0	0	0	N	1	5	9	0	4	6	2	1	1		PLACE MASTER # IN REPEAT (16)	14
		620 0	0	0	N	1	5	9	0	4	6	2	5	1		PLACE MASTER # IN REPEAT (19)	15
		1 0	0	0	R	(0)	0	0	0	1	0	0	0	1	13	REPEAT 16 (X) TIMES	16
		2 0	0	0	+	3	6	2	3	5	6	0	4	9		ADD 18 (A) + 150	17
		3 0	0	0	Y	5	(0	0	0	0)	5	9	0	0	11,16	COMPARE MASTER STK # : TRANS STK #	18

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	624 0	0	0	W	1	6	3	9	0	6	3	9	0		UNEQUAL → HALT	18
		5 0	0	0	R	(0)	0	0	0	1	0	0	0	1	14	REPEAT 19 (X) TIMES	19
		6 0	0	0	+	3	6	2	7	5	6	0	4	9		ADD 21 (A) + 150	20
		7 0	0	0	⊖	&	(0	0	0	0)	5	9	2	9	12,19	SUBTRACT BALANCE - ISSUE	21
		8 0	0	0	H	1	5	0	0	0	0	0	1	( )	9	WRITE CELL	22
		9 0	0	0	V	1	0	2	1	9	6	0	5	0		TRANS → 1	
2130	6	630 0	0	0	U	1	6	3	4	9	6	0	4	1		EO TO CHANGE 2^0 BIT OF LSD OF OLD BAND #	23
		1 0	0	0	Y	3	6	3	4	7	5	9	0	0		COMPARE MODIFIED OLD BAND # : NEW BAND #	24
		2 0	0	0	W	1	6	3	6	0	6	3	6	0		UNEQUAL → 27	24
		3 0	0	0	N	1	6	0	4	3	6	3	4	1		PLACE 1 IN N OF SELECT	25
6380		4 0	0	0	D	()	0	0	0	0	0	(0	0	0)	26,28,29	SELECT BAND	26
		5 0	0	0	V	1	0	2	1	9	6	1	4	0		TRANS → 9	
6320	4	636 0	0	0	N	1	6	0	4	4	6	3	4	1		TRANS 0 TO N OF SELECT	27
		7 0	0	0	M	3	5	9	0	0	6	3	4	7		TRANS NEW BAND # SELECT	28
		8 0	0	0	V	1	0	2	1	9	6	3	4	0		TRANS → 26	
6240		9 0	0	0	⊡	0	0	0	0	0	0	0	0	0		ERROR HALT	
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXIX-16

IE 1224 REV. 8-60

BLOCK READ FROM RECORD SIMULTANEOUS:

This instruction is similar in all respects to the Block Read from Record Normal except that is is accomplished in the simultaneous mode.

The OP is G.

N indicates the number of blocks to be read (1-9, 0=10), and the Data Record File Control.

The A address gives the HSM location to receive the first character.

B indicates:

B_0 Zero(0).

B_1 1: disc returned to cage
0: disc remains on turntable and arm positioned at beginning of band

B_2 1: 900 count terminates instruction
0: EB or 900 count terminates instruction

B_3 addresses first cell to be read (0-9)

If the address one to the right of the last character read in is needed, the S register must be stored.

BLOCK WRITE TO RECORD SIMULTANEOUS:

The OP is I.

N indicates the number of blocks to be written (1-9, 0=10), and the Data Record File Control.

The A address gives the HSM location of the first character to be written

The B address indicates:

B_0 Zero(0).

B_1 1: disc returned to cage
0: disc remains on turntable and arm is placed at the beginning of the band

B_2 1: 900 count terminates instruction
0: 900 count or EB terminates instruction

B_3 address first cell to receive data (0-9)

Example Problem:

Suppose a utility company has 8000 accounts maintained on a data record file in sequential order. Each master contains 450 characters, which means that two masters

can be stored in each cell (20 per band, 100 discs for the entire file). In addition, let's assume that this installation has 20,000 character locations of memory.

The problem is to issue statements for each master. These statements are to be produced on the on-line printer so that two customers' bills are produced side by side (each statement takes up 60 print positions). Each set of statements has 5 lines of printing and a 3 line skip before the next set:

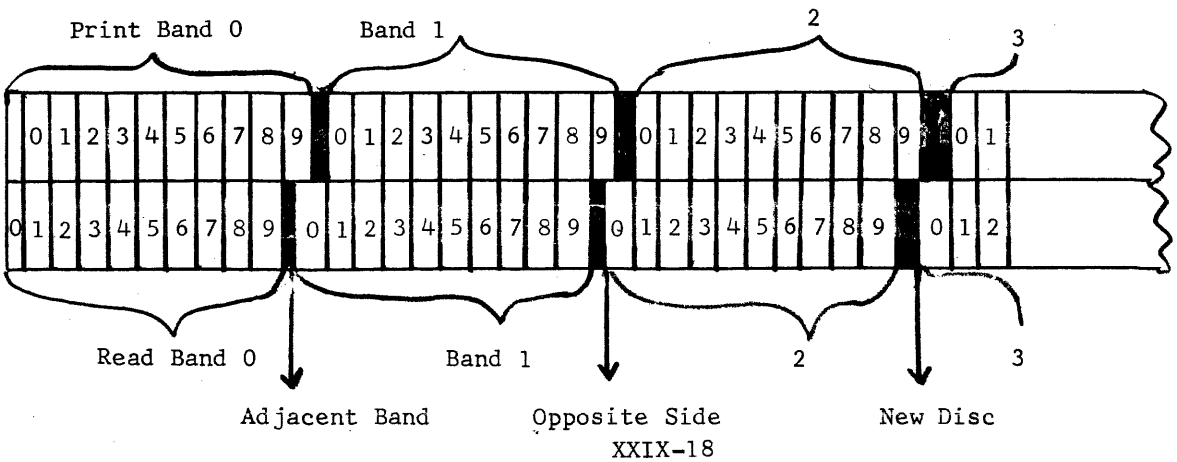
Date Name Address Line 1 Address Line 2 Balance Due	Date Name Address Line 1 Address Line 2 Balance Due
Date Name etc.	Date Name etc.

If this problem were to be programmed without simultaneity, we would have the following time breakdown (assume that set up can be handled during paper advance):

	Time in seconds
READ - 400 Bands	1600.0
Select New Disc (100 times)	160.0
Return Record to Cage (100 times)	160.0
Select band on opposite side of record (100 times)	340.0
Select adjacent bands (200 times)	180.0
Latency (400 times)	80.0
Read/Write Arm transferred to record (400 times)	248.0
Print 5 lines & Pa 2 lines (4000 times)	1573.6
	4341.6

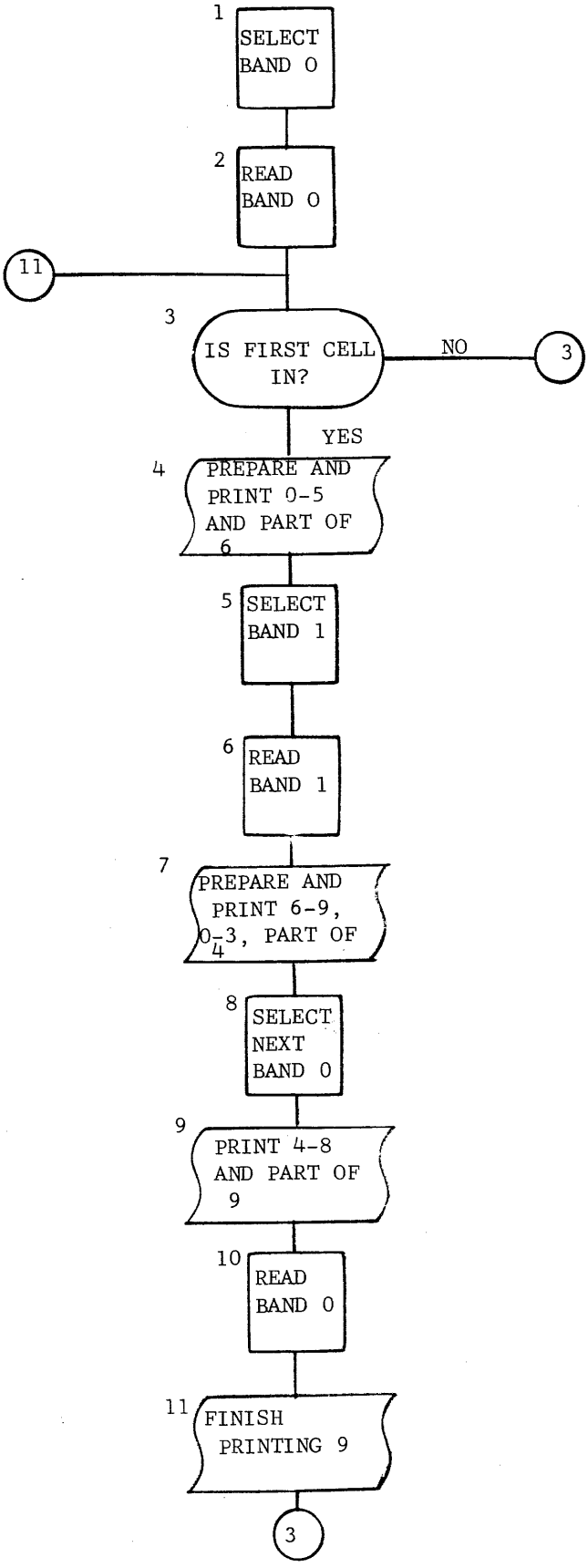
or 1 hour 12 minutes 4 seconds

However, if simultaneity were available, we could program the reads in the Simultaneous Mode and do the printing and set up in the Normal. In addition, we could start the computation and printing immediately upon receipt of the information and could start the read of the second band into the same area, thus saving memory. The bar graph would appear as follows:



Timewise, we need only include the Read and Select time which totals 46 minutes, 1 second. This is nearly half the processing time required without simultaneity.

The flow charted program appears as follows:


1
SELECT BAND 0
2
READ BAND 0
11
3
IS FIRST CELL IN?
NO
3
YES
4
PREPARE AND PRINT 0-5 AND PART OF 6
5
SELECT BAND 1
6
READ BAND 1
7
PREPARE AND PRINT 6-9, 0-3, PART OF 4
8
SELECT NEXT BAND 0
9
PRINT 4-8 AND PART OF 9
10
READ BAND 0
11
FINISH PRINTING 9
3

Data Record File Exercise I:

Flow chart and code the following program:

A utility company maintains its customer billing masters on a Record File. Each master contains 450 characters, so that two may be stored on a cell. There are 8000 customer masters in the entire file, filling bands 100-499. The first eight (8) digits of each master is the account number. Each account number is actually a direct address:

XXXX	XXX	X
Customer	Band	Cell
Number	#	#

The last 7 digits in the field is the amount due the utility company. Each day, approximately 400 payment transactions are posted to the masters. The payment information is coming in from punched cards: the account number (8 digits) is located in columns 1-8, and the payment amount (loaded with insignficant zeros) is located in columns 74-80.

You are to assume that the transactions are coming in in completely random order. An EF card terminates the file. Program to eliminate unnecessary select time and unnecessary read and write time (if the proper cell is in memory, there is no need to change information). Halt if the account number is not found in the proper cell.

Data Recrod File Exercise II:

A company keeps its payroll master file on a record file. There are 3000 employees, and each employee has a 300 character record, so that data concerning 3 employees may be maintained on one cell (30 per band).

Against these masters there will transactions in a ratio of 1 card for every master.

Each master will require about 30 ms. of computation (does not include set up of print lines which can be buried in the paper advance time). In addition to writing out the updated information to the record file, there will be two print outputs (one to each of two printers). The first will be a one line per employee payroll register (single spaced), and the second will be a check containing 3 lines of print. You are to double space between the first and second lines and between the second and third lines, but leave 3 spaces between the last line and the beginning of the next form. The forms are set up so that you are printing checks for 3 employees at a time:

Employee 1	Employee 2	Employee 3
XXXXXXXXXX	XXXXXXXXX	XXXXXXXXXX
XXXXXXXXXXX	XXXXXXXXXX	XXXXXXXXX
Employee 4	Employee 5	Employee 6

Show the total amount of time it would take to process this program without simultaneity, and then bar graph (with times) the problem using simultaneity and calculate a total time for comparison purposes.

Assume a 20K memory.

XXX – DATA RECORD FILE MODE

In addition to the initial record files, it is possible to obtain up to 4 more data record files under the control of the Data Record File Mode. This data record file mode has its own set of registers, called the FOR, L, U and V registers, which directly correspond to the Normal Registers. As in the case of simultaneity, if a Data Record File Mode instruction is staticized in the Normal Registers, and if the Data Record File Mode is free, the instruction will drop into the Data Record File Mode and be executed there, leaving the Normal Mode free for additional operation.

It is feasible, therefore, that if an installation had both the Simultaneous Mode and the Data Record File Mode, up to three operations could be performed on a time-sharing basis. For example, a program could be reading in the Data Record File Mode while printing in the Simultaneous Mode and computing in the Normal Mode.

INSTRUCTIONS:

BAND SELECT RECORD FILE MODE:

This instruction selects a band from one of the four Data Record File units controlled by the Data Record File Mode Control Unit. Once initiated, the band will be selected independent of computer operation.

The OP code is E.

N indicates two things:

UNIT				EFFECT
1	2	3	4	
0	&	-	"	Return record to cage, if any. Place new RCD. on turntable and select proper band
1	A	J	/	Record on turntable is the correct one

The A address is zeros(0000).

The B address gives in $B_{1, 2, \text{ and } 3}$ the number of the band desired (000-511) Timing is the same as in the Band Select Normal.

RECORD FILE MODE READ:

This instruction reads from selected cells located on the selected band. The instruction is executed only in the Data Record File Mode.

The OP code is *.

N gives the number of blocks to be read in the numeric portion, and the specific data record file in the zone bits (00, 01, 10, 11).

The A address gives the location to receive the first character.

The B address is broken into parts:

B_0 is zero

B_1 indicates the position of the disc at the end of the operation (1, returned to cage; 0, still on turntable with arm at beginning of Band).

B_2 indicates the terminating condition (1, 900 count; 0, 900 count or EB).

B_3 gives the address of the first cell to be read (0-9).

RECORD FILE MODE WRITE:

This instruction is executed in the Data Record File Mode and writes data from memory to selected cells.

The OP code is %.

N gives both the number of blocks (in the numeric portion) and the Data Record File addressed (in the zone portion).

The A address gives the HSM location of the first character to be written.

The B address indicates in:

B_0 zero

B_1 the location of the disc at the end of the operation

B_2 the terminal condition

B_3 the address of the first cell to receive data.

If instructions are going to be executed in the Data Record File Mode, we may desire to obtain the final contents of the U register. To do this, we can store the U register by placing an & in the N character of the Store Register Instruction.

We also have the ability to sense the Data Record Files. This is done by the Input-Output Sense instruction:

The OP is S.

N indicates the device:

R is the first Data Record File, Z is the second.

@ ⊡ , indicate the additional four record files under control of the Data Record File Mode.

The tests that can be made are designated in A_0:

1 bit in the 2^0 Is the selected Data Record-File inoperable?

1 bit in the 2^1 Is the selected Data Record File operating?

1 bit in the 2^2 Is a disc on the turntable of the selected Data Record File?

The B address gives the transfer location if a yes answer is obtained.

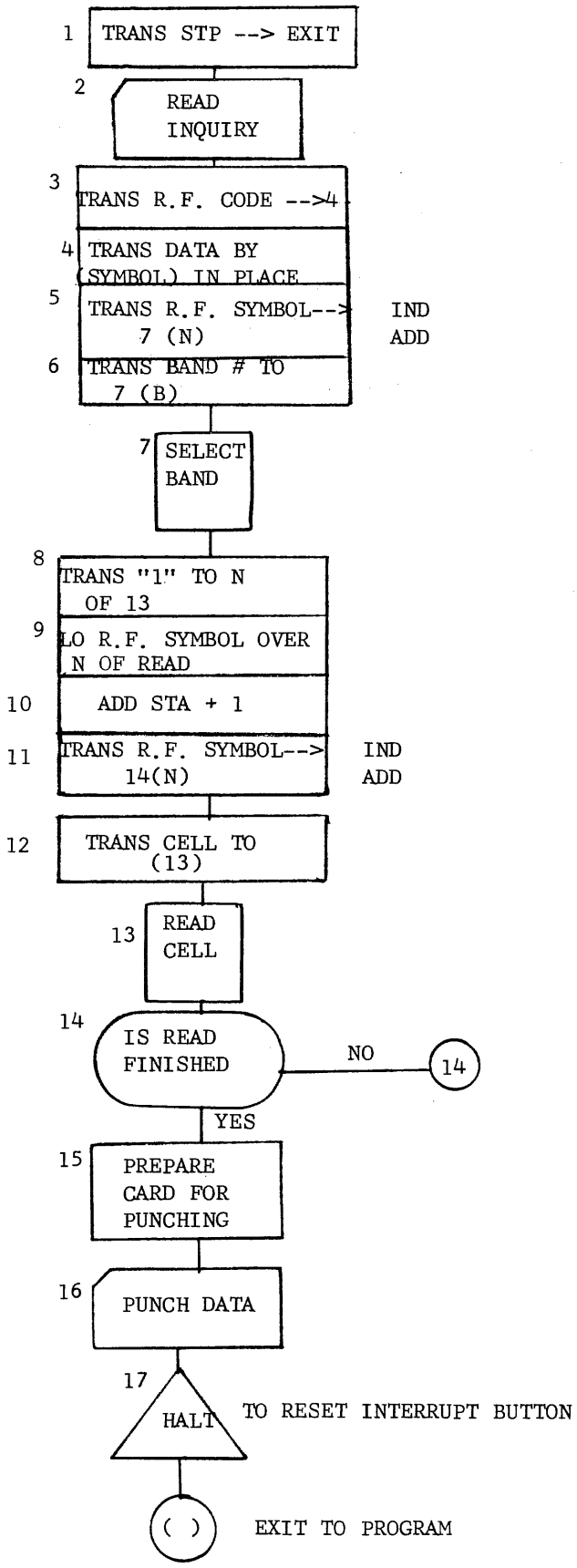
To illustrate these instructions and the programming applications, let's examine an inquiry routine.

We have stored on the additional four data record files 20, 240 records, each 640 characters in length. Each record occupies 1 cell, and is terminated by an EB symbol. The identifying number of each record contains the following:

1 character Record File identification	3 character band identification	1 character cell identification
1	000-511	0-9
2		
3		
4		

In our main updating program, we have included a CTC to sense the Interrupt Indicator on the console. This is done by placing an & in N, the address of the interrupt routine in A, and the address of the next instruction to be executed if the indicator is not set in B. The interrupt button must be manually set and reset:

Our problem is to write the interrupt routine, which is to read a card containing the identifying number of the record desired, select and read it, and punch out a card with some pertinent information regarding balance.


1 TRANS STP --> EXIT
2 READ INQUIRY
3 TRANS R.F. CODE -->4
4 TRANS DATA BY (SYMBOL) IN PLACE
5 TRANS R.F. SYMBOL--> 7 (N)
IND ADD
6 TRANS BAND # TO 7 (B)
7 SELECT BAND
8 TRANS "1" TO N OF 13
9 LO R.F. SYMBOL OVER N OF READ
10 ADD STA + 1
11 TRANS R.F. SYMBOL--> 14(N)
IND ADD
12 TRANS CELL TO (13)
13 READ CELL
14 IS READ FINISHED
NO
14
YES
15 PREPARE CARD FOR PUNCHING
16 PUNCH DATA
17 HALT
TO RESET INTERRUPT BUTTON
( )
EXIT TO PROGRAM

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	900 0	0	0	N	4	0	2	1	9	9	2	1	9		TRANS STP ⟶ EXIT	1
		1 0	0	0	0	1	9	9	0	0	0	0	0	0		READ INQUIRY (RF BAND CELL X XXX X)	2
		2 0	0	0	N	1	9	9	0	0	9	9	3	1		TRANS RF CODE ⟶ 4 (N)	3
		3 0	0	0	#	(0)	9	2	2	8	9	2	2	8	3	LOCATE RF CODE	4
		4 0	0	0	M	1	0	2	1	E	9	0	6	1		TRANS RF SYMBOL ⟶ 7 (N) IND ADD	5
		5 0	0	0	M	3	9	9	0	1	9	0	6	7		TRANS BAND ⟶ 7 (B)	6
	6	906 0	0	0	E	(0)	0	0	0	0	0	(0	0	0)	5.6	SELECT	7
		7 0	0	0	N	1	9	2	2	1	9	1	2	1		TRANS "1" TO 13 (N)	8
		8 0	0	0	Q	1	9	1	2	1	9	0	6	1		LO RF SYMBOL OVER 13 (N)	9
		9 0	0	0	+	2	0	2	1	5	9	2	2	5		ADD STA + 1	10
		910 0	0	0	M	1	0	2	1	E	9	1	3	1		TRANS RF SYMBOL ⟶ 14 (N) IND ADD	11
		1 0	0	0	N	1	9	9	0	4	9	1	2	9		TRANS CELL ⟶ 13 (B_3)	12
	6	912 0	0	0	*	(0)	9	2	5	0	0	3	2	(0)	3,9,12	READ CELL	13
		3 0	0	0	5	(0)	2	0	0	0	9	1	3	0	11	I-O SENSE IS READ FINISHED?	14
		4 0	0	0													15
		5 0	0	0													
		6 0	0	0												PREPARE CARD	
		7 0	0	0													
	6	918 0	0	0													
		9 0	0	0	2	0	9	9	0	0	9	9	7	0		PUNCH CARD	16
		920 0	0	0	⊡	0	0	0	0	0	0	0	0	0		HALT TO RESET INTERRUPT BUTTON	17
		1 0	0	0	V	1	0	2	1	9	(0	0	0	0)	1	TRANS PROG	
		2 0	0	0	⊡	1	0	0	0	1	0	0	1	0		CONSTANTS AND TABLE	
		3 0	0	0	#	2	&	$	3	⊟	⊡	4	"	,			

XXX-5

IE 1224 REV. 8-60

The first step sets up a re-entry into the main program by modifying the terminating program transfer instruction. Once the inquiry card containing the identifying number is in memory, it is a simple matter to set up the Select and Read instructions. The first thing that must be determined is the Data Record File being addressed. In the identification, the files are designated by 1, 2, 3, or 4. The Select and Read instructions, however, need a particular bit configuration in the N character (2^5 and 2^4 must be 00, 01, 10, or 11), then the Input-Output Sense instruction recognizes the different data record files by #, $, ., or ,. For this reason, we must program to obtain the desired symbols for that particular record file. If we have in memory the following character configuration:

10#2&$3⊖⊡ 4",

and then we transfer the record file code from the card area to the N character of a Transfer Data by Symbol instruction, which has as A and B addresses the left-hand limit of the above data, when we executed the instruction we would have in STA the address one to the right of the symbol (1, 2, 3, or 4). By using this as an indirect address, we could transfer the 0, &, ⊖, or " to the N character of the select instruction and also use the character as a Logical Or modifier of a 1 in the N character of the Read. STA could then be increased by 1 and the transfer of the #, $, . or , to the N character of the Input-Output Sense instruction can be accomplished. This input check will make sure that the Read from the Record File has been finished before it will allow the computation of the data to begin.

Suppose a company had a Data Record File Control Unit with all four of the addition Record Files attached. On these record files there are 20, 480 master records, each one of which is 800 characters in length, and takes up 1 cell.

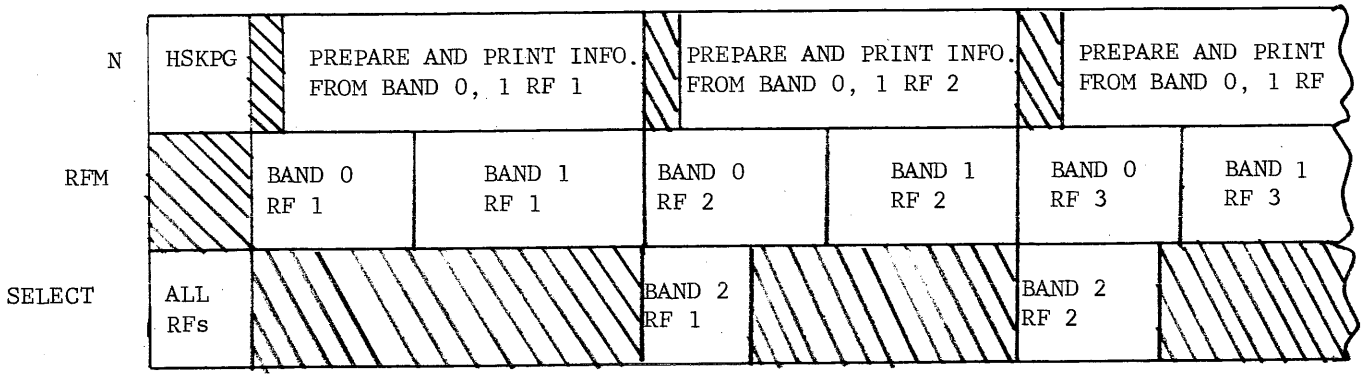
The problem is to print out a three line summary (double space between the 1st and the 2nd lines, and between the 2nd and 3rd lines, and leave 4 spaces between master prints) for every master. The order of the print is not important.

If the Data Record File Mode were not available, the time for this program would be as follows:

	Time in seconds
READ 2048 BANDS	8,192.0
SELECT 512 DISCS	819.2
RETURN 512 DISCS	819.2
BAND ON OPPOSITE SIDE OF DISC 512 TIMES	1,740.8
BAND ON ADJACENT SIDE OF DISC 1024 TIMES	921.6
LATENCY 2048 TIMES	409.6
R/W ARM TRANSFERRED TO RECORD 2048 TIMES	1,269.76
PRINT 20,480 MASTERS	5,355.52
	19,527.68

OR 5 HRS 25 MIN 46 SECONDS

With the availability of the Data Record File Mode Control, however, we can cut our time substantially. The first step would be to make the initial select on each record file. Once the read from the first band of the first record file has been accomplished, computation and printing can be done on that data while the next cell is being brought into memory. In this way, there can be a continual read going on in the RFM. As soon as the 10 cells of the first band are in, the read of the second band can begin. When that is finished, the read of the first band on the second record file can begin, etc. In the meantime, a Select can be done to obtain the next band on the first record file. Bar graphed, the program would appear as follows:



This means that we can completely bury the preparation of the print lines, the printing and the select time in the Read time, so that the only thing that need be included is the 8192 seconds, or 2 hours, 16 minutes, and 32 seconds, an improvement of nearly 50% over the other time mentioned.

Record File Mode Exercise:

Flow chart the program just discussed, showing the preparation of the print line as only one block labeled "preparation". Ignore terminal condition charting.

XXXI – DATA DISC FILE

One module of the Data Disc File has 6 data discs.

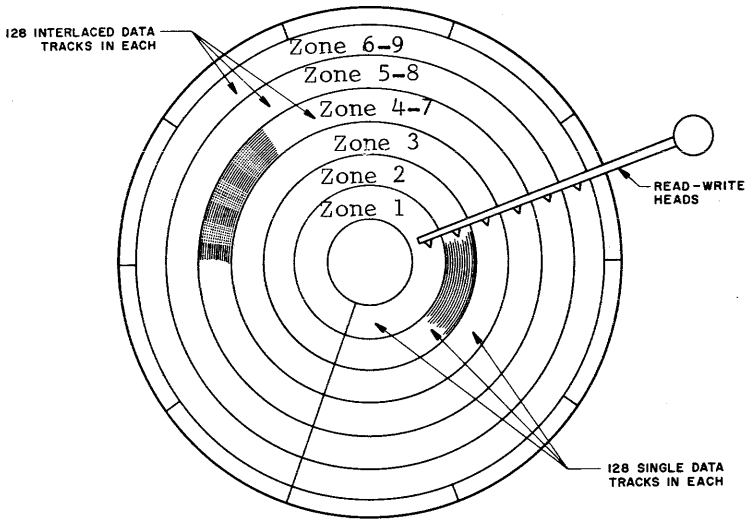
Each disc surface has 9 zones.

Each zone has 128 tracks.

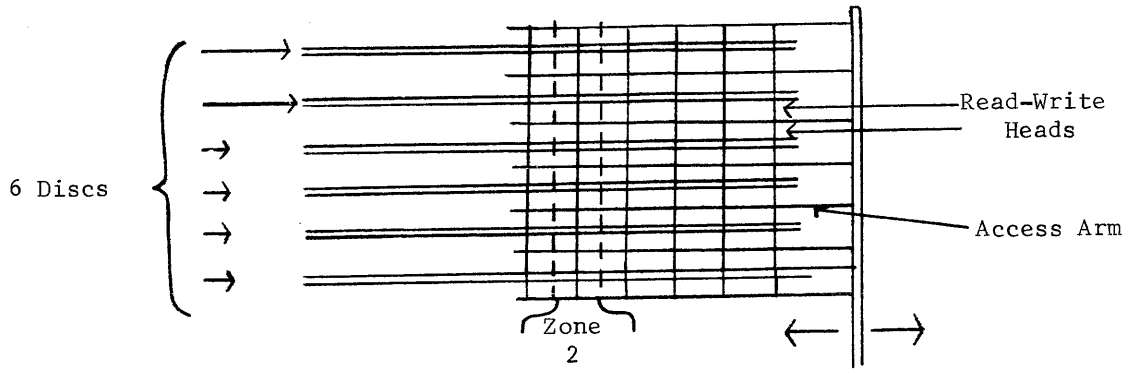
Each track has 10 sectors.

Each sector has the ability to hold 160 characters.

The total capacity of one module is 22,118,400 alphanumeric characters.



Since there is more disc surface available for data nearer the rim of the disc, the tracks of the three outer zones are interlaced - zone 4 with zone 7, 5 with 8, and 6 with 9. This means there are 6 read-write heads required on one disc surface - the outer 3 reading and writing 2 zones each. Therefore, one Data Disc File Module requires 72 read-write heads.



Since in one movement all the access arms of a file are positioned so that all the read-write heads are over the same track in each zone, 108 related tracks are available with no extra movement (in one module). These available tracks are called "Strata".

A Track Select instruction positions the read-write heads in the file selected to the proper track, which are numbered 000-127.

A Sector Read or Write instruction is used to read or write from one to ten sectors of any one of the tracks of the strata for which the read-write heads are positioned. Any number of these instructions may be given after a Track Select instruction. There are no gaps between sectors.

Track Select Timing:

Staticizing	42 us
Average time to reach proper track	75 ms

Sector Read-Write Timing:

Average time to reach proper sector	25 ms
Read one sector	5 ms
Transfer Rate	32,000 char/sec.
Time to switch from read or write to same track	35 us + time to locate sector
Interrupt	7 us out of 31.25 us

Up to 2 Data Disc Files can be attached to the RCA 301 system. Each file may be 1 to 4 modules in size.

TRACK SELECT:

The OP code is D.

N - R (First Data Disc File)
  - Z (Second Data Disc File)

A address - (0000) zeros.

B address - B_0 = 0 (zero), will not override a DDF Simultaneous Instruction.
= &, will override any DDF Sim. Instr.

B_1, B_2, B_3 = Designate track address.

SECTOR READ DISC NORMAL:

The OP code is F.

N - Specifies additional number of sectors to be read as follows:

Additional Sectors	0	1	2	3	4	5	6	7	8	9
First File	0	1	2	3	4	5	6	7	8	9
Second File	&	A	B	C	D	E	F	G	H	I

A address - Address of HSM location to receive the first character read from the Data Disc File. This address must be an even number (left-hand end of a diad).

B address - B_0 - designates first sector to be read:

First Sector to be read	0	1	2	3	4	5	6	7	8	9
Will Not Override Any DDF Simultaneous Instruction	0	1	2	3	4	5	6	7	8	9
Will Override Any DDF Simultaneous Instruction	&	A	B	C	D	E	F	G	H	I

B_1, B_2, B_3 - specify the zone from which data is to be read:

MODULE	ZONE
1	000 to 107
2	108 to 215
3	216 to 323
4	324 to 431

SECTOR READ DISC SIMULTANEOUS:

The OP code is G.

The N, A address, and B address are programmed the same as the Sector Read Disc Normal Instruction.

SECTOR WRITE DISC NORMAL:

The OP code is H.

The N, A address, and B address are programmed the same as in the Sector Read Disc Normal Instruction.

SECTOR WRITE DISC SIMULTANEOUS:

The OP code is I.

The N, A address, and B address are programmed the same as in the Sector Read Disc Normal Instruction.

The Input-Output Sense Instruction can be used with the Data Disc File.

The OP code is S.

N = R - First Data Disc File.
Z - Second Data Disc File.

A_0 = 1 - is the device inoperable
2 - is the device operating
4 - is the track select complete
8 - has incorrect parity been read

A_1, A_2, A_3 = zeros (000).

B address - HSM location of the next instruction to be executed if the condition or conditions being tested are present.

Any instruction containing a bit in the B_0 2^4 position will terminate any DDF instruction being executed in the Simultaneous Mode. The instruction with the B_0 2^4 bit will then be executed in the normal manner.

Example:

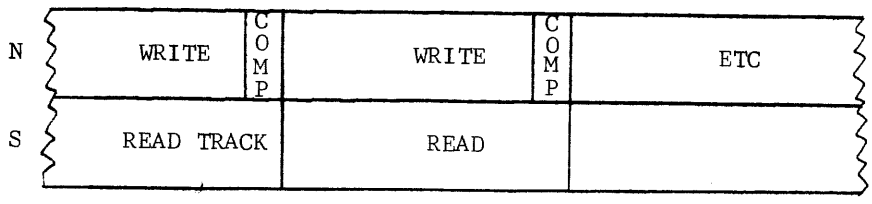
Suppose we had a master file on Data Discs that we wish to duplicate onto tape (33kc) for backup purposes. We wish to do this the fastest way possible (simultaneity) and yet supply enough information so that editing this tape will not be too difficult.

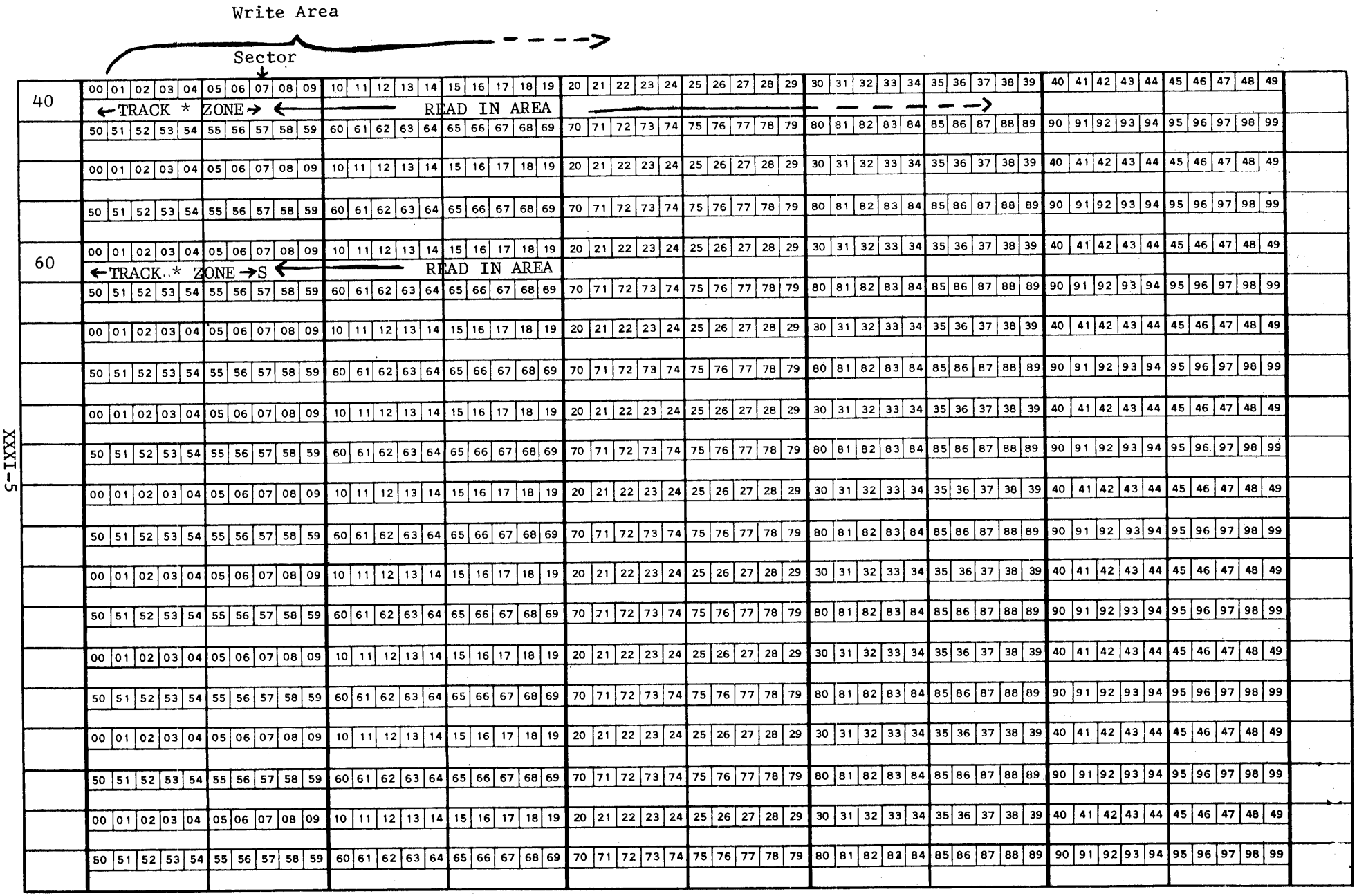
Let us assume 160,000 records of 80 characters per record distributed on a one-module Data Disc File as follows:

Records	1 → 20	Track 001	Zone 001
	21 → 40	001	002
	41 → 60	001	003
	etc.		
Records	2141 → 2160	Track 002	Zone 001
	4281 → 4300	003	001
	etc.		

We can read a track in the Simultaneous Mode while writing and editing in the Normal Mode. By placing the track number, zone number, and the sector number from which we started reading in front of the read-in area, we can write the read-in area plus this identification in one write instruction.

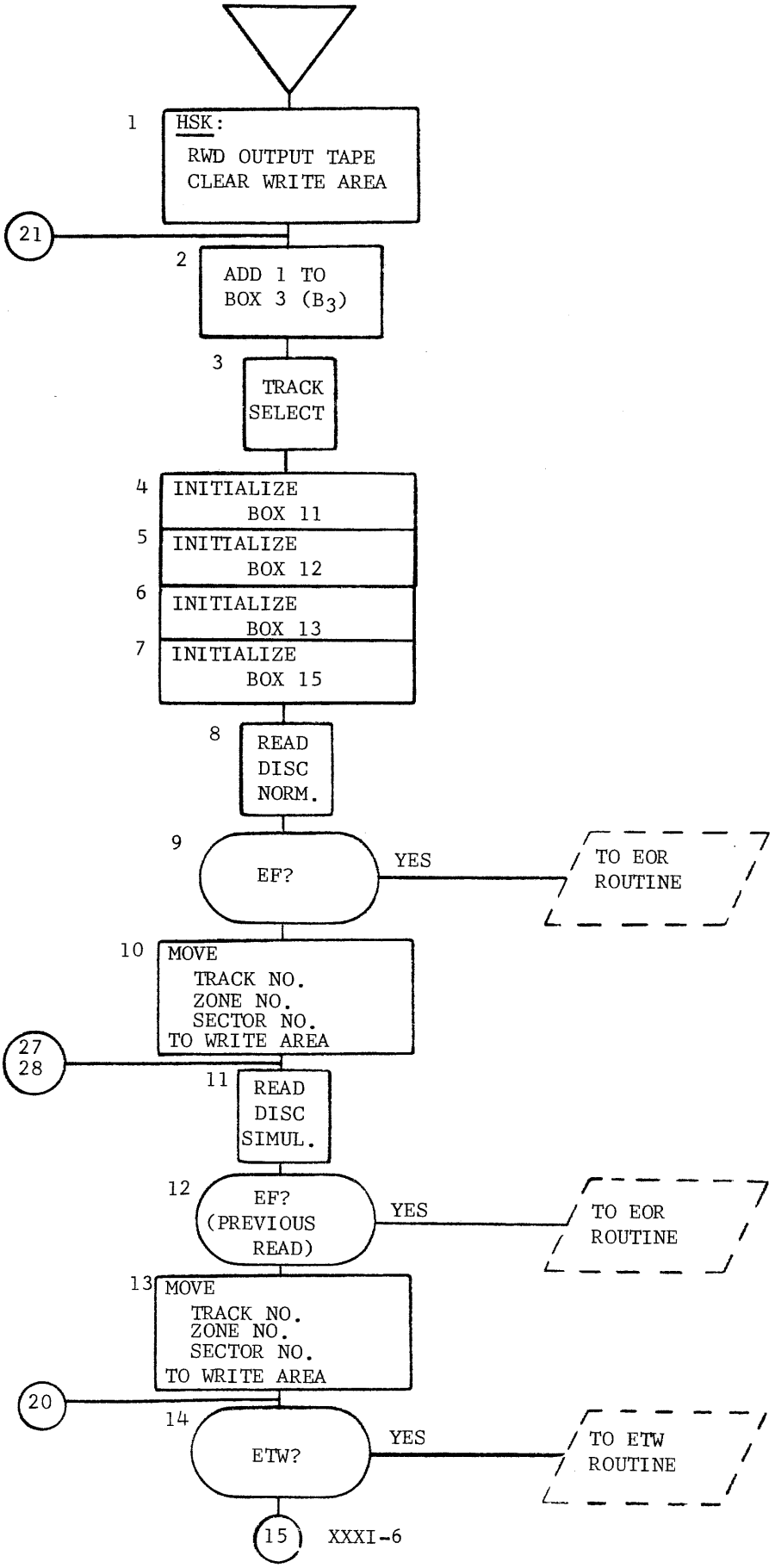
Assuming an EF at the beginning of a track to signify the end of the file, the program could be flowcharted and coded as follows:

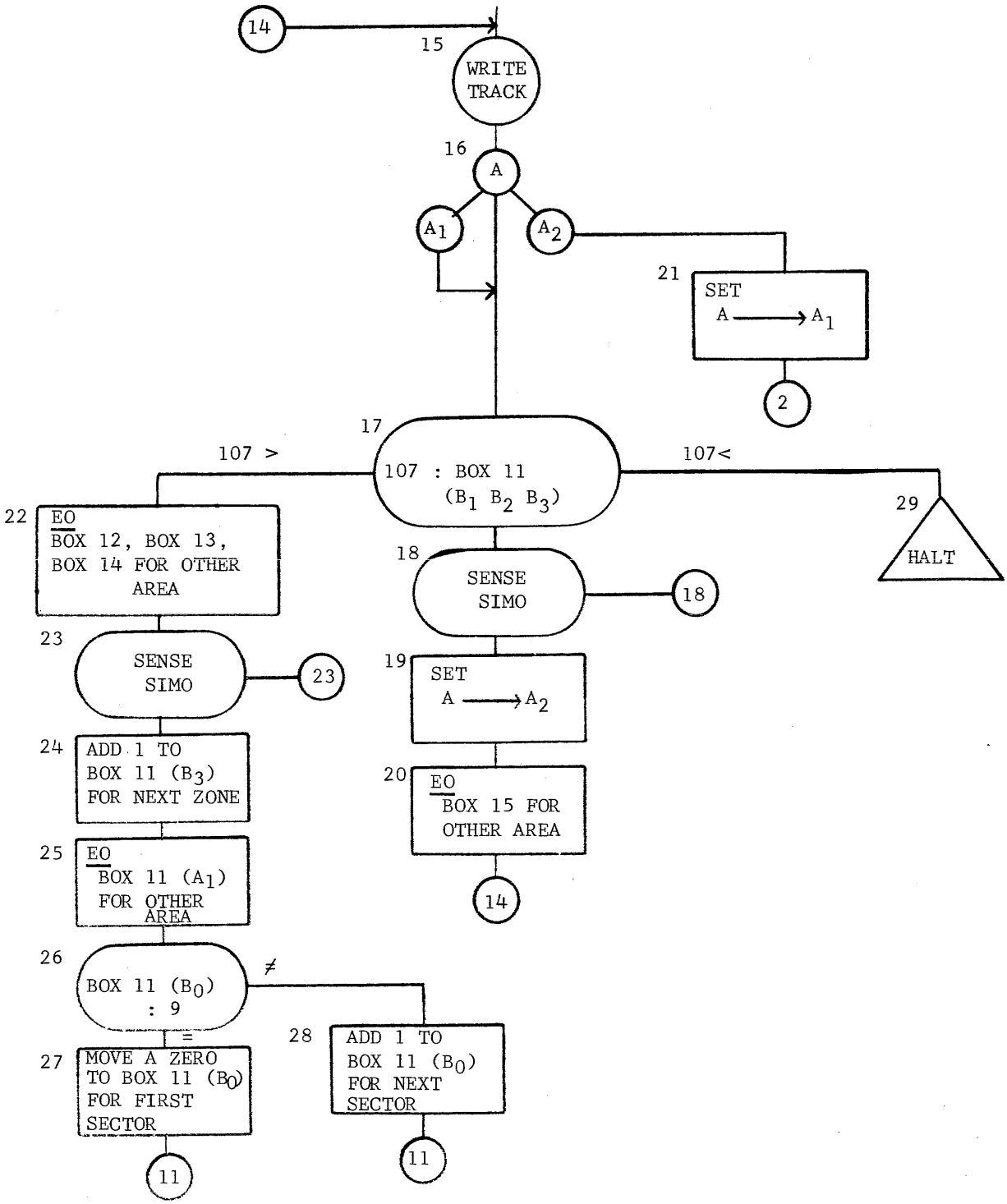




XXXI-5

TITLE: _____ BLOCK NO.: _____ INDEX NO.: _____ PROGRAMMER: _____ DATE _____ PAGE ___ OF ___


1
HSK:
RWD OUTPUT TAPE
CLEAR WRITE AREA
21
2
ADD 1 TO
BOX 3 (B_3)
3
TRACK
SELECT
4
INITIALIZE
BOX 11
5
INITIALIZE
BOX 12
6
INITIALIZE
BOX 13
7
INITIALIZE
BOX 15
8
READ
DISC
NORM.
9
EF?
YES
TO EOR
ROUTINE
10
MOVE
TRACK NO.
ZONE NO.
SECTOR NO.
TO WRITE AREA
27
28
11
READ
DISC
SIMUL.
12
EF?
(PREVIOUS
READ)
YES
TO EOR
ROUTINE
13
MOVE
TRACK NO.
ZONE NO.
SECTOR NO.
TO WRITE AREA
20
14
ETW?
YES
TO ETW
ROUTINE
15


14
15
WRITE TRACK
16
A
A1
A2
21
SET A → A1
2
17
107 >
107 : BOX 11 (B1 B2 B3)
107<
22
EO BOX 12, BOX 13, BOX 14 FOR OTHER AREA
29
HALT
18
SENSE SIMO
18
23
SENSE SIMO
23
19
SET A → A2
24
ADD 1 TO BOX 11 (B3) FOR NEXT ZONE
20
EO BOX 15 FOR OTHER AREA
25
EO BOX 11 (A1) FOR OTHER AREA
14
26
BOX 11 (B0) : 9
≠
=
28
ADD 1 TO BOX 11 (B0) FOR NEXT SECTOR
27
MOVE A ZERO TO BOX 11 (B0) FOR FIRST SECTOR
11
11

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	200 0	0	0	;	2	0	0	0	0	0	0	0	0		RWD Tape	1
		1 0	0	0	J	sp	4	0	0	0	8	0	0	0		Clear Read-Write Area	1
2360		2 0	0	0	+	1	2	0	3	9	2	5	4	5		Increment Track Select	2
		3 0	0	0	D	R	0	0	0	0	0	0	0	0		Track Select	3
		4 0	0	0	N	&	2	5	3	9	2	1	8	9		Initialize Box 11	4
		5 0	0	0	J	4	2	1	9	2	2	1	9	2		" Box 12	5
	6	206 0	0	0	J	6	2	2	2	6	2	2	2	6		" Box 13	6
		7 0	0	0	J	6	2	2	3	6	2	2	3	6		" Box 13	6
		8 0	0	0	J	6	2	2	4	6	2	2	4	6		" Box 13	6
		9 0	0	0	J	4	2	2	6	2	2	2	6	2		" Box 15	7
		210 0	0	0	J	5	2	2	6	6	2	2	6	6		" Box 15	7
		1 0	0	0	F	9	4	0	0	8	0	0	0	1		Read Disc Normal	8
	6	212 0	0	0	Y	1	4	0	0	8	2	5	4	0		Check for EF	9
		3 0	0	0	W	1	2	1	5	0	2	1	5	0			9
		4 0	0	0	V	1	0	2	1	9	X	X	X	X		To End of Run Routine	
2130		5 0	0	0	N	3	2	0	3	9	4	0	0	3		Move Track No. to Write Area	10
		6 0	0	0	N	3	2	1	1	9	4	0	0	6		Zone No.	10
		7 0	0	0	N	1	2	1	1	6	4	0	0	7		Sector No.	10
2510 2490	6	218 0	0	0	G	9	(6)	0	0	8	(1)	0	0	(2)		Read Disc Simul	11
		9 0	0	0	Y	1	(4)	0	0	8	2	5	4	0		Check for EF	12
		220 0	0	0	W	1	2	2	2	0	2	2	2	0			12
		1 0	0	0	V	1	0	2	1	9	X	X	X	X		To End of Run Routine	
2200		2 0	0	0	N	3	2	0	3	9	(6)	0	0	3		Move Track No. to Write Area	13
		3 0	0	0	N	3	2	1	1	9	(6)	0	0	6		Zone No.	13

8-IXXX-8

28-00-004

PAGE OF

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	224 0	0	0	N	1	2	1	1	6	(6)	0	0	7		Sector No.	13
2340		5 0	0	0	S	2	4	0	0	0	X	X	X	X		Sense ETW	14
		6 0	0	0	8	2	(4)	0	0	1	(5)	6	0	7		Write Track	15
		7 0	0	0	V	(0)	0	2	1	9	2	3	5	0		A Switch	16
		8 0	0	0	Y	3	2	1	8	7	2	5	4	1		Check for End of Strata	17
		9 0	0	0	W	1	2	3	7	0	2	5	2	0			
2300	6	230 0	0	0	W	4	2	3	0	0	2	3	0	0		Clear Simo	18
		1 0	0	0	J	1	2	2	7	1	2	2	7	1		Set A --> A_2	19
		2 0	0	0	U	1	2	2	6	6	2	5	4	4		} Modify Write for Other Area	20
		3 0	0	0	U	1	2	2	6	2	2	5	4	4		}	20
		4 0	0	0	V	1	0	2	1	9	2	2	5	0		To Write	
2270		5 0	0	0	J	0	2	2	7	1	2	2	7	1		Set A ----> A_1	21
	6	236 0	0	0	V	1	0	2	1	9	2	0	2	0		To Begin Next Strata	
2290		7 0	0	0	U	1	2	1	9	2	2	5	4	4		Modify Boxes 12, 13, and 14	22
		8 0	0	0	U	1	2	2	2	6	2	5	4	4			22
		9 0	0	0	U	1	2	2	3	6	2	5	4	4		} To Address Other Area	22
		240 0	0	0	U	1	2	2	4	6	2	5	4	4			22
		1 0	0	0	U	1	2	2	6	2	2	5	4	4			22
	6	242 0	0	0	U	1	2	2	6	6	2	5	4	4			22
2430		3 0	0	0	W	4	2	4	3	0	2	4	3	0		Clear Simo	23
		4 0	0	0	+	1	2	1	8	9	2	5	4	5		Increment Zone No. in Read	24
		5 0	0	0	U	1	2	1	8	2	2	5	4	4		Change Read to Other Area	25
		6 0	0	0	Y	1	2	1	8	1	2	1	8	6		Check Sector No. for 9	26
		7 0	0	0	W	1	2	5	0	0	2	5	0	0			26

6-IXXX-9

28-00-004

PAGE OF

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	248	0	0	J	0	2	1	8	6	2	1	8	6		Move a Zero to Sector No.	27
		9	0	0	V	1	0	2	1	9	2	1	8	0		To Read Next Track	
2470		250	0	0	+	1	2	1	8	6	2	5	4	5		Add 1 to Sector No.	28
		1	0	0	V	1	0	2	1	9	2	1	8	0		To Read Next Track	
2290		2	0	0	⊡	0	0	0	0	0	0	0	0	0		Error Halt	29
		3	0	0	G	9	6	0	0	8	1	0	0	2			
	1	254	0	0	E/F	1	0	7	2	1	0	0	0	0		Constants	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXXI-10

28-00-004

PAGE OF

XXXII — TABLE LOOK-UP TECHNIQUES

The term "table look-up" simply refers to a programming technique used to locate one particular thing in a table of like values. For example, we can see where this would come in very handy when dealing with the data record file since it is not common to have a direct addressing scheme similar to the ones which we have been using. We could get around the problem of knowing that exact address for each master by setting up a table which contains both the different stock numbers (for example) and the corresponding addresses. Normally this table would be laid out in one of two formats, either in order by stock number or in order by frequency of use. In either case we would have the following:

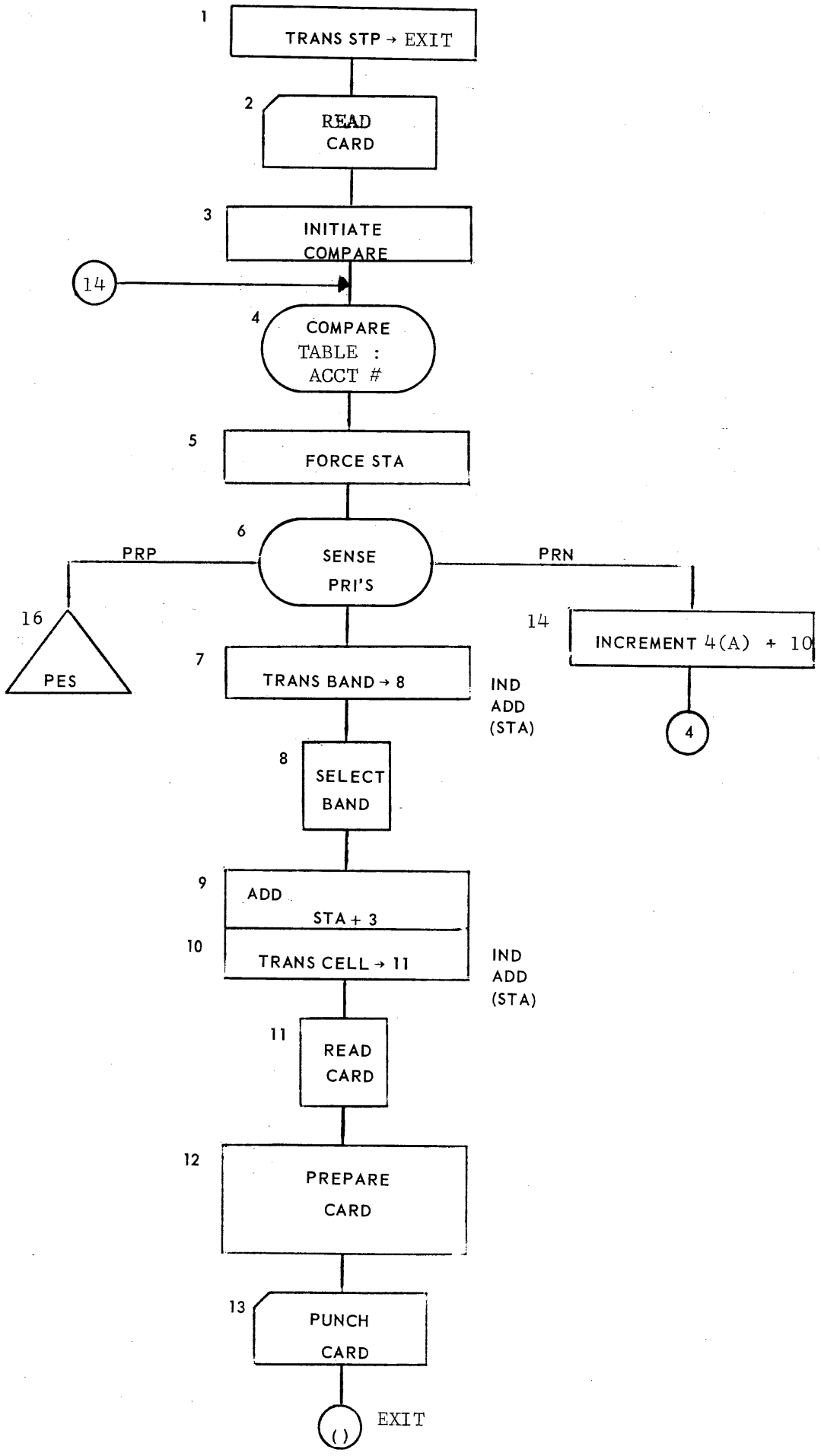
SSSSSAAAAASSSSSAAAAASSSSSAAAAASSSSSAAAAA etc.

STK # ADDRESS

Suppose we are handling a program similar to the interrupt program we discussed previously. The only difference is that now we will bring in a card with an account number on it and from that must locate the information. We can assume that we have a table set up in memory in ascending order by account number:

Account #	Band and Cell	Account #	Band and Cell	etc.
6	4	6	4	

The last account number is a dummy with the highest binary code possible. We will assume that we have a 20K memory and that the last 10K is used as table storage. We have 1000 records stored in 1000 cells on the record file. Assume that all requests have masters. The program would look as follows:


1
TRANS STP → EXIT
2
READ CARD
3
INITIATE COMPARE
14
4
COMPARE TABLE : ACCT #
5
FORCE STA
6
SENSE PRI'S
PRP
PRN
16
PES
14
INCREMENT 4(A) + 10
4
7
TRANS BAND → 8
IND ADD (STA)
8
SELECT BAND
9
ADD STA + 3
10
TRANS CELL → 11
IND ADD (STA)
11
READ CARD
12
PREPARE CARD
13
PUNCH CARD
EXIT

As we can see, we form a loop until we find the account number in the table. At that point, STA will contain the address of the MSD of the band. By forcing STA and using indirect addressing, we can transfer both the band and later, by modifying STA, the cell location.

Another example of table look-up might run as follows:

We have a table that contains 26 different letter codes (A-Z) each followed by a specific digit hourly rate. When doing a payroll problem we find one of these 26 codes in each master and we must then transfer the appropriate rate to a work area. A portion of the table appears as follows:

A150B155C160D165E170 etc.

We could program another compare loop again, but an even simpler and faster way to do it would be as follows:

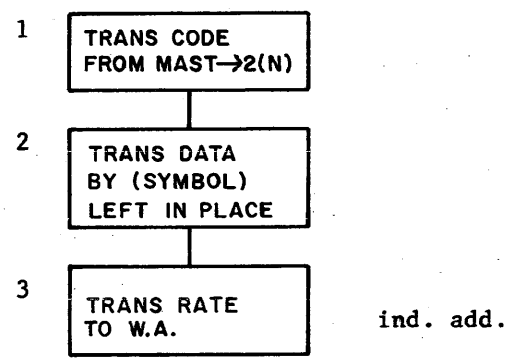


Table 2000-2103

Master

```
Read
In            EMP #     RATE CODE
Area          XXX            X

          00  01  02  03
10         X   X   X   X

          2010 N 1   1003 2021  DR THE SYMBOL
          2020 # ()  2000 2000  TSL TO FIND CODE
          2030 M 3   Ø21E 1200  DL TO MOVE RATE TO WORK AREA
```

Another type of table look-up could be called "direct addressing". This is the same type of thing that is done when executing an ADD or SUBTRACT instruction. There is an instruction in the RCA 301 complement that is called Translate by Table. This actually is a table look-up instruction which allows a translation from any 7 bit binary code to any other 7 bit binary code. Its primary use is for compatability between RCA systems, but many other applications can be developed for it.

TRANSLATE BY TABLE:

This instruction translates a specified number of characters in one area from one code to another, by the use of a specified table.

The OP code is A.

The N character is the number of characters to be translated. This is limited to 0-44, but since this instruction is repeatable, the actual limit is 660 (44 x 15), without a second Translate.

The A address gives the HSM location of the leftmost character to be translated. This will also be the leftmost location of the translated data, since the result replaces the original information.

The B address gives the HSM location of the first character in the table to be used for translation. This table must be stored beginning at a hundreds location; i.e., the last two characters in the address must be 00.

The direction of this operation is left to right, and $(A)_f$ will hold the address of the location one to the right of the last character translated in the original area. $(B)_f$ is identical to $(B)_i$.

In order to operate, it is necessary to break each character to be translated into two parts, which then becomes the address of the location in the table that holds the translated equivalent. Suppose, for example, that we were translating RCA 301 code to RCA 501 code. An "A" in 301 is represented by $(010001)_2$, in 501 by $(100000)_2$. The 301 "A" would be pulled from memory and the 2^2, 2^1, and 2^0 bits transferred to the corresponding location of D_3 (D register). The 2^5, 2^4, and 2^3 bits would be transferred to the 2^2, 2^1, and 2^0 positions of D_2. The other bits of the D register would be zero, so that the figures constructed would be a 2 and a 1:

RCA 301 code for an "A" = 010001

D_2

5	4	3	2	1	0
2	2	2	2	2	2
0	0	0	0	1	0

which is the RCA 301 code for a 2.

D_3

5	4	3	2	1	0
2	2	2	2	2	2
0	0	0	0	0	1

which is the RCA 301 code for a 1.

These characters, in combination with the contents of the B register, are used to address the particular table being used for translation. For example, if our table started at 7500, at location 7521 there should be the binary character 100000, which is the RCA 501 code for an "A".

Obviously, these tables can be made up for any purpose. Suppose, for another example, that our program we were to ascertain that a particular field, coming in on the transactions, was numeric. One way to do this would be to compare each location in the field to a 9, and if equal to or less than a 9, continue on and sense the next character in the field, but if greater than a 9, register an error. For a field of 10 digits, this would require 10 compares and 10 senses of the PRI's, or timewise, 10(56 + 49) = 1050 us. Suppose, however, that we set up a table in memory that would translate all numerics to 0's and all other characters to 1's. We would have to transfer our 10 digit field to a work area, since it would be destroyed during translation, translate it, search through the result area for the lack of a zero, and then test the PRI's for a PRZ setting (a character other than 0 in the field will set a PRP or a PRN in a Locate Symbol Left or Right instruction). This would require 175 us. for transferring the data, 245 us. for translation, 210 us. for searching for a non-zero, and 35 for sensing the PRI's, or a total of 665 us., a

considerable savings over the other approach. The table would also be very easy to construct, since the decimal digits would be the only characters that would develop into 00, 01, 02, 03, 04, 05, 06, 07, 10, or 11 addresses. By placing 0's in these locations and 1's in all the other positions (12-77), the table would be accurate.

Table Look-Up Exercise I:

A shipping company pays its employees by cash whenever a ship lands. It is the accounting departments responsibility to make up a list of the number of hundred dollar bills, fifties, twenties, tens, fives and ones, and the number of fifty cent pieces, quarters, dimes, nickels and pennies needed to meet any particular payroll. Having converted to EDP, it is your job to write a program which will develop these totals, and bring the final results out on the on-line printer. You may assume no more than 5 digit amounts of any one denomination (99999 one dollar bills, for example; that no persons make over $99999.99; that data is fixed; and that there is only one reel of information. Flow chart this program only.

Since you will need tables to solve this program, you should indicate the construction of the table. You need not show the entire set of tables.

XXXIII – RANDOMIZING

With regard to the Data Record Files, we have discussed two methods of locating information. One was the direct addressing scheme and the other was the table look-up method. There is also a third method called "randomizing". This is simply utilizing the criterion to form the address. For example, one method is to take the criterion and add parts of it together. The break up of the criterion is decided upon statistically, in order to come up with the fewest possible duplications of addresses.

Stock # 34562436

3456
2436
5892

Obviously the next step would be to break this down into two parts, band and cell. This could be done by subtracting the highest possible address from the band portion if it is 512 or over.

589
511
078

We now have an address for the information connected with stock #34562436 and that is band 078, cell 2.

By working out an address for each of the stock records, we could then assign them to that particular location. The one problem is, of course, duplication of address. If we had 140 character records, we could get 6 records in the same cell, but if more than 6 records have the same address, we must come up with a technique to take care of the overflow.

One method is "chaining". By this method, when we have placed our six records, and we find more with the same address, we could place the address of another cell at the end of this cell, so that if the desired record is not in this cell, the program has only to pick up this new address and select, read, and search the alternate cell. This technique would be efficient only if the high-hit records were placed in the first cell; i.e., the idea is to arrange the data records by use rather than by sequence of criteria.

Our example assumed that we were using only the single data record file, other similar steps could be developed to allow use of all the data record files.

There are hundreds of randomizing techniques. The choice of the correct one or possibly the development of a new one is based on a statistical study of the data to be used.

XXXIV – ARITHMETIC OPERATIONS

Although we have discussed Adding and Subtracting throughout the text, we have not mentioned certain qualifying conditions.

For example, we have assumed that all of our problems have allowed for maximum fields, but what would happen if we had an instruction to add two 3 digit fields together and the result was 1197? The field would not be expanded to a 4 digit field, but rather a zone bit called an "overflow" bit would be placed in the 2^4 position of the MSD of the sum. In our example, the answer would then be A97 (the binary code for an A is 010001).

This condition can be tested for by the Conditional Transfer of Control Instruction with a 2 in the N character. If the overflow indicator has been set, a transfer will occur to the address given in the A address, if the indicator has not been set, the program will transfer to the address given in the B address.

A second situation should then occur to us. What would happen if we had an illegitimate bit, such as a 1 bit in 2^5 of anything except the LSD (in which case it indicates a negative number) or a 1 bit in 2^4 of anything except the MSD? A 1 bit in the 2^5 position of anything except the LSD will cause a parity alarm. A 1 bit in the 2^4 position of anything except the MSD will cause the computer to hang on an ARIE (Arithmetic Error) alarm. In addition, if we try to add two operands which both have overflow bits, or if we try to add two operands where one had an overflow bit and where a carry is generated by the addition, we will also obtain an ARIE alarm. Subtracting two fields which both have overflow bits will not cause an alarm, so long as each operand is at least 2 characters in length.

For example:

C3216 plus D4831 will hang the computer on an ARIE since there are two overflow bits;
C43 plus 942 will hang the computer, since there is already one overflow bit and a carry will be generated;
C342 plus 0238 will not hang the computer
D2184 minus A3216 will not hang the computer
D minus A will hang the computer since there are not 2 digits in each operand
321E54 minus 213583 will hang the computer due to the bit in the 2^4 position which is not the MSD of the operand
21N4982 plus 2938475 will hang the computer
A21 minus E78 will not hang the computer even though the subtrahend is greater than the minuend since the answer will only be negative.
E212 minus A43L will hang the computer, since the subtraction is of a minus number from a positive number, which will cause an addition of the two operands, and both have overflow bits

The final arithmetic topic which has not been covered is how to multiply and divide. We have already noted that the RCA 301 has both an Add and a Subtract instruction. To aid the users, RCA will supply multiply and divide subroutines, which will perform these functions. A sub-routine is simply a group of instructions which will accomplish a given task. These particular sub-routines will be "closed" sub-routines, which means that when the programmer wishes to execute a multiply or divide, he must transfer to the second instruction of the sub-routine (since these routines will be floatable (capable of being placed at any free area in memory) this will be the float address plus 10), the routine will then pick up STP, which tells where the transfer came from, and the last instruction will be a transfer back to this location.

There are three multiply routines and two divide routines. The multiply routines allow for operation on an 8 X 8, 10 X 10 or 17 X 17 field. The divide routines allow for operation on a 10 X 10 or 17 X 17 field. If an operand does not have the exact number of characters called for in the routine, it must be loaded with insignificant zeros.

It is the programmer's responsibility to place the operands at standard HSM locations, and any unused portions (including the product and the quotient areas) must be cleared of zeros.

MULTIPLY	MULTIPLICAND	MULTIPLIER	PRODUCT	FILL WITH ZEROS
8 X 8	0808-0816 LSC at 0816	0791-0798 LSC at 0798	0817-0834 LSC at 0834	0791-0798 0808-0816
10 X 10	0800-0809 LSC at 0809	0785-0794 LSC at 0794	0815-0834 LSC at 0834	0785-0794 0800-0809
17 X 17	0799-0816 LSC at 0816	0782-0798 LSC at 0798	0799-0834 LSC at 0834	0782-0798 0799-0816
DIVIDE	DIVIDEND	DIVISOR	QUOTIENT	
10 X 10	0825-0834 LSC at 0834	0800-0809 LSC at 0809	0785-0794 MSC at 0782	0800-0809 0825-0834
17 X 17	0818-0834 LSC at 0834	0800-0816 LSC at 0816	0782-0798 MSC at 0782	0800-0816 0818-0834

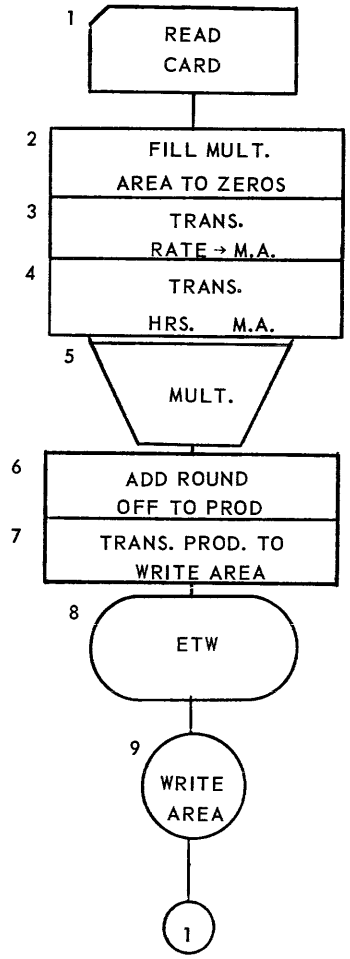
The 8 X 8 multiply requires 580 locations; the 10 X 10 600 locations; the 17 X 17 690 locations. The 10 X 10 divide requires 600 locations; the 17 X 17 470 locations. The multiplicand is destroyed, but the multiplier is restored. The divisor is restored but the dividend is destroyed; the remainder is left in the dividend field.

Example:

Suppose we had to write a program which would read in a card, calculate the total gross wage by multiplying the hourly wage by the rate, and transcribe this information to tape. The initial card is in the format:

cols.	1-5	6-9	10-13
	Emp #	Rate	Hours (XX.XX) without the decimal point)

The transcription should appear in the same format, except that an additional 5 digit field should be added to the end of the record containing the gross salary. This must be rounded to the nearest penny. The flow chart would appear as follows:



THIS SYMBOL SHOWS ENTRANCE TO, EXCECUTION OF, AND EXIT FROM A SUBROUTINE.

Suppose one particular card had the information:

1234502554750

which means that employee 12345 works at a rate of $2.55 and worked this week 47.5 hours.

Once the multiply area has been filled with zeros and the operands moved to their respective fields, the multiplication will yield 1211250, or $121.12 and 5 mills. In order to round off, we must add 0000050 to the product area, which will give us 1211300, and moving just the leftmost 5 digits (12113) to the write area will place this weeks gross earnings in the proper position for writing the record to tape.

The coding follows (assume that the multiply was placed immediately following the program) on page XXIV-4.

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
6090	6	600 0	0	0	0	1	5	0	0	0	0	0	0	0		READ CARD	1
		1 0	0	0	J	0	0	7	9	1	0	8	1	6		FILL MULT. AREA WITH ZEROS	2
		2 0	0	0	N	4	5	0	0	8	0	8	1	6		TRANS RATE M.A.	3
		3 0	0	0	N	4	5	0	1	2	0	7	9	8		TRANS HOURS M.A.	4
		4 0	0	0	V	1	0	2	1	9	6	1	2	0		TRANS TO MULTIPLY SUB-ROUTINE	5
		5 0	0	0	+	7	0	8	3	4	6	1	0	9		ADD ROUND-OFF TO PROD	6
	6	606 0	0	0	N	5	0	8	3	2	5	0	1	7		TRANS PROD. TO WRITE AREA	7
		7 0	0	0	S	3	4	0	0	0	E	T	W			SENSE ETW	8
		8 0	0	0	8	3	5	0	0	0	5	0	1	7		WRITE RECORD	9
		9 0	0	0	V	1	0	2	1	9	6	0	0	0		TRANS → 1	
		610 0	0	0	0	0	0	0	0	0	0	0	5	0		CONSTANTS	
		1 0	0	0												MULT ROUTINE (410 LOC)	
	6	2 0	0	0												ENTRANCE	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXXIV-4

IE 1224 REV. 8-60

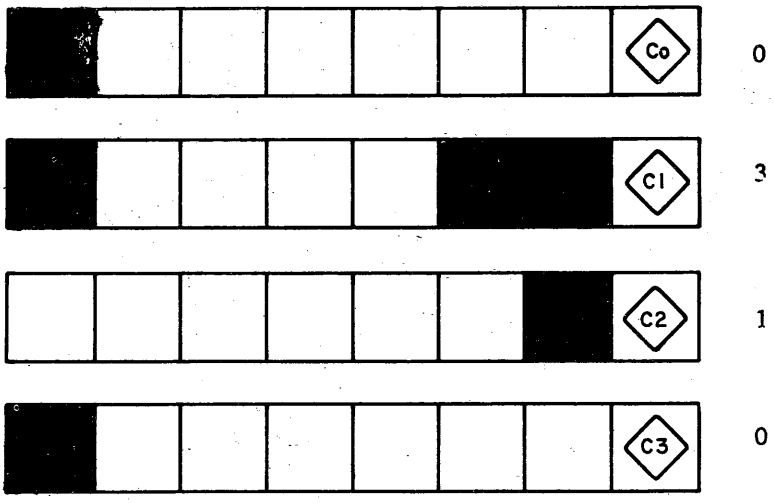
XXXV – CONSOLE

The Console is normally the operators domain, but since it never does the programmer any harm to be familiar with the layout of the panel and the various indicators, we will take time to discuss the layout of the console, the abbreviations, and the functions of the indicators and buttons.

GEN RES: The General Reset button resets all the registers and counters and most of the flip flops, when depressed.

START: The Start button will begin execution of the status level displayed.

P, A, B, NOR/N, D, MR, S, T, SOR/M, U, V, FOR/L indicate the various registers. By depressing any one of these you will cause the contents of the register to be displayed. The contents of these registers will not be visable unless they are particularly called for in this manner. For example, say that we were interested in the contents of the A register when the computer stopped. By selecting the A register (depressing the button marked A) the address in the A register would appear in binary in the display area:



Reading this, we can see that the address is 0310.

The P, A, B, S, T, U, and V register will appear as four characters in length; the NOR/N, SOR/M, and FOR/L and the D and MR registers appear as only 2 characters.

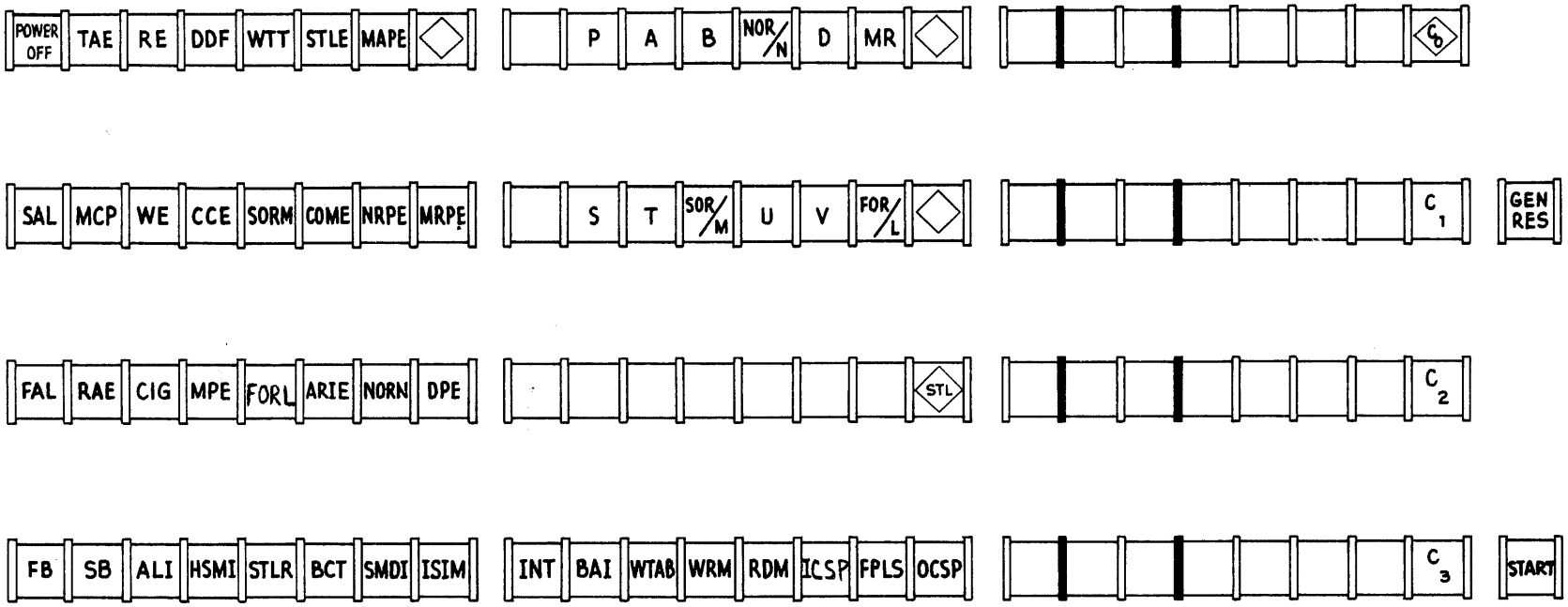
The Special Purpose switches are alternate action switches and include:

OCSP: One Cycle Stop, which permits the operation to be performed one status level at a time by stopping the computer at the end of every status level.

ICSP: Instruction Complete Stop, which permits one instruction at a time operation by stopping the computer prior to the staticizing the next instruction.

FPLS: First Processing Level Stop, which permits the computer to stop after staticizing the instruction.

RCA 301 CONSOLE



◇All buttons with ◇configurations are reset buttons.

RDM: Read Memory, which allows the displaying of any diad in HSM.

WRM: Write Memory, which permits the placing of two characters at a time into any diad in HSM.

HSMI: HSM Inhibit, which inhibits information from going to or coming from HSM.

BAI: Bus Adder Inhibit, which inhibits the adding or subtracting ability of the Bus Adder.

STLR: Status Level Repeat, which inhibits the changing of the current status level.

ISIM: Inhibit Simultaneity, causes all instructions to be executed serially, although they take place in the mode desired.

BCT: Bypass Card Translation, which causes the automatic card translation to be inhibited, and places the information into memory so that 2 characters represent 1 column.

INT: Interrupt button, which is sensed by the CTC instruction when N = &.

WTAB: Write to Table allows the arithmetic tables (0000-0199) to be written into HSM. When not set, any attempt to write to the tables will cause an alarm.

SMDI: Simultaneous Mode Inhibit, causes all Simultaneous Instructions to be performed in the Normal Mode. It also will store the A register when the S register is indicated in the Store Register instruction. However, storing the A register only means forcing STA, and will not automatically place the contents in any given destination area.

ALI: Alarm Inhibit, when depressed the Computer will not stop on any error condition, but the alarm indicator will light.

The Alarm Indicators include:

Alarms for Parity Errors in the following registers:

SORM: Simultaneous Operation or M Registers

NORN: Normal Operation or N Registers

FORL: V or L Registers

NRPE: Repeat Register

MAPE: Memory Address Register

MRPE: Memory Register

DPE: D Register

STLE: Status Level

alarms for such other errors as:

COME: Error in the Comparator

ARIE: Arithmetic error caused by one of the following in a 10K or 20K computer:
2^4 bit is a "1" in MSD of both operands in an Add instruction;
2^5 or 2^4 bit is a "1" in other than MSD or LSD of either operand;
2^4 bit is a 1 in the MSD of one of the operands and there is a carry.

Processor with 40,000 character memory:

a. An address ADD or address SUBTRACT instruction with a resultant negative address whenever an operand is distinguishable from data by having a zone bit in the MSD position.

b. 2^4 or 2^5 bit is a "one" in other than the MSD or LSD of either operand.

c. 2^5 bit is a "one" in the MSD of one of the operands and there is a carry.

WTT: Write To Table indicates an attempt to write to the arithmetic table without setting WTAB.

DDF: Device Doesn't Follow means that the device addressed is inoperable.

RE: Read Error indicates that an error has occurred during a read instruction or when data is transferred from HSM to the printer output buffer.

TAE: Tape Address Error indicates that there is a parity error in the tape address.

CCE: Card Compare Error indicates the second read station does not compare with the first read station on the Card Reader, or that the read station does not compare with the punch on the Card Punch.

WE: Write Error indicates that an error has occurred during a write instruction or when data is transferred between the memory register and an input or output buffer.

MCP: Missing Clock Pulse shows that there is an error on the 33 kc or 66 kc tape station(s).

SAL: Simultaneous Alarm occurs when an input-output alarm has occurred and the instruction is in the Simultaneous Mode.

MPE: Multipunch Error indicates that a non-301 character has been recognized on a card with the BCT switch off.

CIG: (Character in the Gap) shows that a block of less than 3 characters has been read or indicates that a Card Read or Card Punch instruction has not been staticized in the required amount of time when utilizing Read or Punch release for the Card Reader/Punch.

RAE: Record File Address Error indicates a parity error in the Record File Address Register.

FAL: Record File Alarm shows that an error has occurred in an instruction in the Record File Mode.

The remaining miscellaneous indicators are:

SB, which indicates that the Simultaneous Mode is busy.

FB, which indicates that the Record File Mode is busy.

Power Off, which will turn off the power to the computer. Turning the power on must be done at the Power Supply.

XXXVI – RCA 301 PROGRAMMING STANDARDS

In all the programs we have done up to this point, we have been interested only in doing the work defined in a correct and efficient manner. However, if we were programming for actual use, rather than training, we would be required to follow certain programming standards. In order that these requirements will be completely understood and familiar, it seems advisable to discuss them, in detail, at this point.

The first requirement which is made is that machine coded programs be prepared in a particular format. By utilizing the RCA 301 Computer Program Record, that we have been using throughout the text, this requirement is automatically met. This form has many columns which are self explanatory, such as "from instruction location", "HSM location", "Referred to by", "Remarks" and "Box No.". Two columns, Float A and Float B, have not been defined as yet. All of our programs have been written so that the first character falls in a pre-designated location and all characters fall sequentially after it. With this method, each character in the program is addressable by a specific or "fixed" address. Suppose, however, that we wanted to be able to write a program that would fit into memory in any area that we had available at any one time. This type of program we would call "floatable". To enable us to do this, the routines written to load the programs into memory from the initial input media have the ability to "float" programs by incrementing every address which is denoted to be a "float" address by the HSM address of the first location in the area to receive the program. If we coded our program to start at location 0000 (even though we know that this is not normally permitted) and placed a 1 in the Float A and/or Float B box (depending upon if we wanted the A and/or B address modified) we would have a floatable program. For example, the following program does nothing but read, write and transfer. The first solution shows it coded as a fixed program, beginning at location 2000. The second solution shows it as floatable program.

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	OP 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
1	3	200 0	0	0	4	2	2	0	3	0	2	0	9	9		READ	
		1 0	0	0	8	4	2	0	3	0	2	0	9	9		WRITE	
		2 0	0	0	V	1	0	2	1	9	2	0	0	0		TRANSFER	
		3 0	0	0													
		4 0	0	0													
		5 0	0	0													
	6	206 0	0	0												READ AREA	
		7 0	0	0													
		8 0	0	0													
		9 0	0	0													
		0	0	0													
		0	0	0													
2	3	000 0	1	1	4	2	0	0	3	0	0	0	9	9		READ	
		1 0	1	1	8	4	0	0	3	0	0	0	9	9		WRITE	
		2 0	0	1	V	1	0	2	1	9	0	0	0	0		TRANSFER	
		3 0	0	0													
		4 0	0	0													
		5 0	0	0													
	6	006 0	0	0												READ AREA	
		7 0	0	0													
		8 0	0	0													
		9 0	0	0													
		0	0	0													
		0	0	0													

IE 1224 REV. 8-60

PAGE OF

If we then loaded the second solution into the area starting at 2000, 2000 would be added to each address marked for floating:

2000	4	2	2030	2099
	8	4	2030	2099
	V	1	0219	2000

If we had choosen 9500, the program would appear in memory as follows:

9500	4	2	9530	9599
	8	4	9530	9599
	V	1	0219	9500

Thus we can re-float any program simply by re-loading it with a new float address.

In addition, we must code the programs so that if a block of instructions (a block is contained within the heavy black lines) contains less than 6 instruction they must be located at the top of the block with the location to receive the first character in the HSM LOCATION box and the number of instructions in the NO. OF INS. box. This enables the loader routines, which we will discuss later, to place our program into memory at the proper locations. Note the following example:

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	1	3990 0	0	0	X	X	X	X	X	X	X	X	X	X		PROGRAM	
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	4000 0	0	0	X	X	X	X	X	X	X	X	X	X			
		1 0	0	0	X	X	X	X	X	X	X	X	X	X			
		2 0	0	0	X	X	X	X	X	X	X	X	X	X		PROGRAM	
		3 0	0	0	X	X	X	X	X	X	X	X	X	X			
		4 0	0	0	X	X	X	X	X	X	X	X	X	X			
		5 0	0	0	X	X	X	X	X	X	X	X	X	X			
	2	4060 0	0	0	X	X	X	X	X	X	X	X	X	X			
		0	0	0	X	X	X	X	X	X	X	X	X	X			
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

XXXVI-4

247

IE 1224 REV. 8-60

PAGE OF

Writing and Card Key-Punching Standards for All RCA 301 Characters

Writing and Over-Punching Standards

Due to the nature of RCA 301 coding, standards have been established to facilitate the writing and card punching of 301 characters. The standards are defined as follows:

Character Description	Writing Standard	Card Key-Punching
Zero	0	0
One	1	1
Two	2	2
Three	3	3
Four	4	4
Five	5	5
Six	6	6
Seven	7	7
Eight	8	8
Nine	9	9
Space or Underline		No Punch
Number	#	3/8
At	@	4/8
Op n Parenthesis	(	5/8
Closed Parenthesis	)	6/8
Ampersand	&	Y (Zone Punch)
A	A	A
B	B	B
C	C	C
D	D	D
E	E	E
F	F	F
G	G	G
H	H	H
I	I	I
Plus	+	B/8
Period	.	C/8
Semicolon	;	D/8
Colon	:	E/8

Character Description	Writing Standard	Card Key-Punching
Apostrophe	⊙	F/8
Minus	⊖	X (Zone Punch)
J	J	J
K	K	K
L	L	L
M	M	M
N	N	N
O	Ø	O
P	P	P
Q	Q	Q
R	R	R
End of Information	E/I	K/8
Dollar	$	L/8
Asterisk	*	M/8
End Data	E/D	N/8
End File	E/F	Ø/8
Quotes	"	X (Zone Punch)/0
Slant	/	0/1
S	S	S
T	T	T
U	U	U
V	V	V
W	W	W
X	X	X
Y	Y	Y
Z	Z	Z
End Block	E/B	S/8
Comma	,	T/8
Percentage	%	U/8
Item Separator	•	V/8
Equal	=	W/8

Program Card

The instructions written on the RCA 301 COMPUTER PROGRAM RECORD (Exhibit 1) are punched on paper tape or punched cards in instruction groups of 6 or less. The format of the 301 INSTRUCTION CARD is (Exhibit 2) as follows:

Card Column	Interpreter Position	Interpreter Line	Field
1	1-2	1	Number of Instructions
2-5	5-8	1	Beginning HSM Address
6-7	11-12	1	Instr. #1 - Float Digits
8-9	14-15	1	OP-N
10-13	17-20	1	A Address
14-17	22-25	1	B Address
18-19	28-29	1	Instr. #2 - Float Digits
20-21	31-32	1	OP-N
22-25	34-37	1	A Address
26-29	39-42	1	B Address
30-31	45-46	1	Instr. #3 - Float Digits
32-33	48-49	1	OP-N
34-37	51-54	1	A Address
38-41	56-59	1	B Address
42-43	11-12	2	Instr. #4 - Float Digits
44-45	14-15	2	OP-N
46-49	17-20	2	A Address
50-53	22-25	2	B Address
54-55	28-29	2	Instr. #5 - Float Digits
56-57	31-32	2	OP-N
58-61	34-37	2	A Address
62-65	39-42	2	B Address
66-67	45-46	2	Instr. #6 - Float Digits
68-69	48-49	2	OP-N
70-73	51-54	2	A Address
74-77	56-59	2	B Address
78-80	5-7	2	Identification #

Another requirement which must be met deals with segmenting a program. The small programs we have been writing could easily be placed in 10,000 character locations but suppose we wrote a program containing 1300 instructions. This would require 13,000 memory locations and obviously would not fit in memory. For this reason, the program must be broken into segments containing no more than 700 instructions, thus leaving room for read-in areas, work areas, etc.

A segment can be any number of instructions (up to 700) that are independent of the rest of the program. That is, no instructions in a segment can directly affect instructions in another segment, since only one segment of a program will be in memory at one time. For example, housekeeping could be a segment of a program since most of the steps pertain to such things as clearing work areas, checking labels, etc. (The exception would be instructions used to initialize switches and other instructions, since these would not yet be in memory). Once these steps are executed, there is no longer any reason why they should remain in memory, since they will not be executed again during the running of this program. For this reason it would be possible to incorporate steps which would bring in the next segment and place it over the housekeeping instructions, in effect wiping housekeeping out of memory.

This next segment might be the production portion. Once this is completed, the End of Run routine could be performed without having the main instructions in memory so that, if memory requires it, we could make the EOR steps the third segment of the program, and have the second segment (main portion) incorporate coding to read in these EOR instructions when needed.

We have really been discussing two things, segmenting and overlaying. Segmenting is simply deciding where to break the coding, while overlaying is the incorporation of the program steps which will read the next segment into memory over that portion of the program which is already in memory but of no further use. Three rules must be remembered in segmenting:

1) the first instruction of each program segment must contain the letters PS as the first two characters and HSM location to receive this first digit must be placed in the A address. The B address must contain information to identify the program segment;

2) the second instruction of the first program segment must be the entrance location; i.e., first instruction to be executed. This is due to the fact that the insertion routines (which will load the program and set the P register to address the first instruction to be executed) will assume that this condition exists;

3) the last instruction in the entire program must transfer or exit to a standard HSM location.

Since the insertion routine will insert only the first segment of each program, it is the responsibility of the programmer to handle all insertions of additional segments and overlays. In addition to incorporating into the segment those steps necessary to read in the next segment, it is also necessary to do the following:

1) place the segment number (always 2 digits) into HSM locations 0858-0859,

2) place a STORE V REGISTER instruction into HSM locations 0870-0879. This instruction must read:

V 1 0219 XXXX (where XXXX is the return address to the main program)

3) transfer control to HSM location 0860.

There are certain locations in memory which are restricted from the programmer's use. These are called Standard HSM Areas and can be broken into two groups. The first group are those locations required for use by the 301 Computer. This area of memory falls between 0000-0225 and includes:

0000-0099	Sum Table
0100-0199	Difference Table
0202-0205	Card Punch Instruction Location, which is used by the computer in order to keep track of the first of 80 positions to be scanned for each card.
0206-0209	Arithmetic Instruction Location, which is used to hold the address of the sign of the sum or difference so that the computer can correct it, if the result indicates correction is necessary.
0212-0215	STA
0216-0219	STP
0222-0225	Store P during REPEAT
	MODEL 303 PROCESSOR
9900-9977	Print Table
9978-9999	Reserved
	MODEL 304 PROCESSOR
I900-I977	Print Table
I978-I999	Reserved
	MODEL 305 PROCESSOR
Z900-Z977	Print Table
Z978-Z999	Reserved

The second group includes those locations which are required by the service routines, which we will discuss in the next chapter. These fall between 0226-1999, and include:

0200-0200	Storage location for the program tape number for program insertion from magnetic tape (used only for magnetic tape libraries).
0226-0228	Storage locations for the band number position of the standard record file. (Used only for disc libraries and only by the service routines, but whenever a Select instruction is executed the band number must be transferred to these locations.)
0230-0599	Program Insertion Routine
0600-0759	Program Insertion routine work area (also the first 120 locations are used as a standard print area by the service routines or by the programmer, if needed).

0760-0767 Standard Date, this should be loaded as an initial step each day.

0770-0779 Standard Exit, all programs should transfer to this position at EOR.

*0780-0834 Multiply and Divide Parameters.

*0835-0849 Tape Table, which appears as follows:

35	36	37	38	39	40	41	42	43	44	45	46	47	48
08	1	2	3	4	5	6	A	B	C	D	E	F	J	N

This tape table will be initialized by the Insertion Bootstrap. Each program should utilize this table for tape swapping. In this manner, if a particular tape is inoperable, it is possible to make a substitution in the table and modify the entire program. (If you refer back to the entire program done in the chapter on Program Controls, you will note that we also had a "tape table" except that we incorporated ours in the program itself. With this external tape table, we can accomplish uniformity between programs.)

*0850-0879 Overlay Area, which has already been discussed.

*0880-1999 Storage Area for Debugging Routines.

Those areas that are marked with *'s may be used for other purposes when they are not used for the indicated purposes.

The programmer has available from HSM location 2000 to the last 100 locations in memory for his program.

Since only one Stop instruction is available in the RCA 301, it is necessary to indicate whether it is a normal stop or an error stop. The following convention should be used:

a. The N character of a Stop Instruction will be 0 for a programmed normal stop and a 1 for a programmed error stop.

b. The A address of the Stop will be numerically coded to indicate the meaning of the Stop when no print-out accompanies the Stop.

Care must be exercised when using the Translate by Table Instruction. The translate table must be located in a HSM location ending in 00. Particular care should be exercised in floatable programs.

When the Record File is utilized for program storage, an EB character can only appear as the last character in a program segment. Therefore, when need arises to use an EB character for a count, another means must be adopted, such as using a lower count and repeating the instruction. Likewise, if need arises for a constant EB, the last character of the segment may be used.

When using the Record File, the band number of the disc must be stored in the standard locations 0226-0228 after each band select. This requirement is only necessary for the standard record file, i.e., it does not apply for use of the optional record file units. This standard is necessary for any type of program interruption. For example, the Sampler will disturb the position of the Record File while sampling, and after each Sampling Point, will restore the position of Record File according to the information contained in locations 0226-0228.

301 and 501 Translation Tables

Alphabetic, numerics, and following characters:

( ) " : $ % ; ⊖ * / , # E/F E/D ISS • ' &

will be translated from their RCA 501 Configurations to corresponding RCA 301 Configurations and vice versa.

The following table gives the translation for all remaining characters from RCA 501 Configurations to RCA 301 Configurations and vice versa.

RCA 501		RCA 301	
Symbol	Configuration	Symbol	Configuration
Blank	000000	@	001100
Space	000001	SP	001010
Cross	000010	+	011010
Carriage Shift	001111	CS	001111
Page Change	010000	PC	011111
Line Shift	010001	LS	101111
Carriage Normal	011111	=	111110
End Message	111101	EI	101010
Start Message	111110	EB	111010
Delete	111111	Unused	111111

The RCA 501 character $(77)_8$ will be translated to RCA 301 character $(77)_8$. This RCA 301 character can be used for sensing or writing to a Record File or tape on a Tape Adaptor. However, it cannot be written to a tape on the High Data group. Octal characters $(17)_8$, $(37)_8$, $(57)_8$, and $(77)_8$ cannot be introduced into the RCA 301 system via cards or paper tape; and therefore, must be generated if desired for sensing purposes.

501 to 301 Translation Table (501 Magnetic Tape Input)

The following table shows the construction of a translate table in 301 HSM used to translate RCA 501 characters (introduced via a magnetic tape) to the corresponding RCA 301 character configurations. The chart is as the table would appear on a coding sheet where XX is arbitrary.

	00	01	02	03	04	05	06	07	08	09
XX00	@	_	+	(	)	"	:	$	Free	Free
10	%	;	&	'	⊖	*	.	CS	"	"
20	PC	LS	/	0	1	2	3	4	"	"
30	5	6	7	8	9	,	#	=	"	"
40	A	B	C	D	E	F	G	H	"	"
50	I	J	K	L	M	N	Θ	P	"	"
60	Q	R	S	T	U	V	W	X	"	"
70	Y	Z	E/F	E/D	•	EI	EB	$(77)_8$	"	"

501 to 301 Translation Table (501 Paper Tape Input)

The following table shows the construction of a translate table in HSM used to translate RCA 501 characters (introduced via 501 paper tape) to the corresponding RCA 301 Character Configurations. The chart is as the table would appear on a coding sheet where XX is arbitrary.

	00	01	02	03	04	05	06	07	08	09
XX00	$77_{(8)}$	E/B	E/I	•	E/D	E/F	Z	Y	Free	Free
10	X	W	V	U	T	S	R	Q	"	"
20	P	Θ	N	M	L	K	J	I	"	"
30	H	G	F	E	D	C	B	A	"	"
40	=	#	,	9	8	7	6	5	"	"
50	4	3	2	1	0	/	LS	PC	"	"
60	CS	.	*	⊖	'	&	;	%	"	"
70	$	:	"	)	(	+	_	@	"	"

301 to 501 Translation Table

The following table shows the construction of a translate table in HSM used to translate RCA 301 Characters to the corresponding RCA 501 Character Configurations. The chart is as the table would appear on a coding sheet where XX is arbitrary. The symbols internal to the table contain the proper octal configurations to produce the correct translation to 501 characters.

	00	01	02	03	04	05	06	07	08	09
XX00	C	D	E	F	G	H	I	+	Free	Free
10	.	;	1	'	O	3	4	CS	"	"
20		⊖	J	K	L	M	N	Θ	"	"
30	P̄	Q	2	)	9	6	#	&	"	"
40	@	R	E/I	$	*	E/D	E/F	LS	"	"
50	"	/	•	7	(	,	E/B	A	"	"
60	5	B	S	T	U	V	W	X	"	"
70	Y	Z	=	:	8	%	PC	77_8	"	"

Standards for the Organization of Data on Cards, Magnetic and Paper Tape, and Data Record File

Legend

* { E/B represents the RCA 301 End Block Symbol
E/I represents the RCA 301 End Information Symbol
E/F represents the RCA 301 End File Symbol
E/D represents the RCA 301 End Data Symbol

BL represents the Beginning Label
EL represents the Ending Label
BLK represents a Block of Data or One Cell which contains Data

represents an Interblock Gap

represents a single 80-character card

* NOTE:

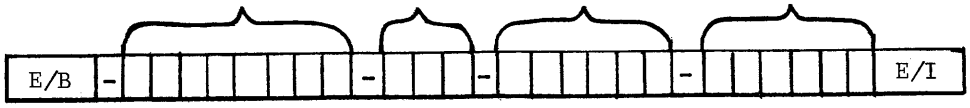
These symbols are used for control purposes only and must not appear within data.

I MAGNETIC AND PAPER TAPE STANDARDS

A. Standard Label Formats

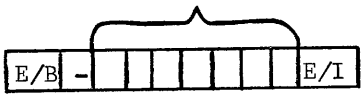
1. Beginning Label (29 Characters)

IDENTIFICATION	REEL-NUMBER	DATE-WRITTEN	PURGE-DATE
8 Chars.	3 Chars.	6 Chars.	6 Chars.



2. Ending Label (10 Characters)

BLOCK-COUNT
7 Chars.



B. Tape Label Conventions

The 301 User is permitted the following label conventions:

1. Standard Labels may be indicated.
2. Beginning and Ending Labels may be omitted.
3. A Beginning Label may be specified without an Ending Label. However, an Ending Label may not be specified without a Beginning Label.
4. The User may specify the format of the entire label(s) in which none of the standard items appear.
5. Additional information may appear in the label following all standard items indicated above, except the E/I symbol. The E/I symbol must always be the last character of the label.

NOTES:

1. The PURGE-DATE of a file is the earliest date on which the contents of the file can be destroyed.
2. The PURGE-DATE of a label written on paper tape always contains six zeros.
3. The BLOCK-COUNT of a file is the total number of data blocks on a tape, and does not include Beginning or Ending Label Blocks.

C. RCA 301 Organization of Data on Tape

1. All data records are terminated by an E/I symbol.

2. A batch may contain:

   a. A variable number of fixed length records,
   b. A fixed number of fixed length records,
   c. A variable number of variable length records,
   d. A fixed number of variable length records.

3. A batch is always terminated by an E/F symbol. The E/F must be the last character in the batch.

4. When the input to a run is composed of all single records, then no E/F symbol appears after the E/I symbol in any of the records. However, when the input to a run is batched, then each batch contains an E/F symbol as the last character of the batch even though some batches may consist of single records.

5. When labels are not present:

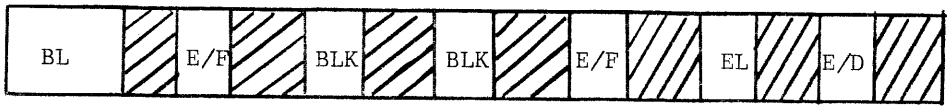
   a. Each reel of a file begins with a block consisting of a single E/F symbol.
   b. Initial and intermediate reels of a multi-reel file are terminated by a block consisting of a single E/D symbol.
   c. The final reel of a multi-reel file, or a single reel file, is terminated by two successive blocks, the first of which consists of a single E/F symbol, and the second, a single E/D symbol.

6. When labels are present:

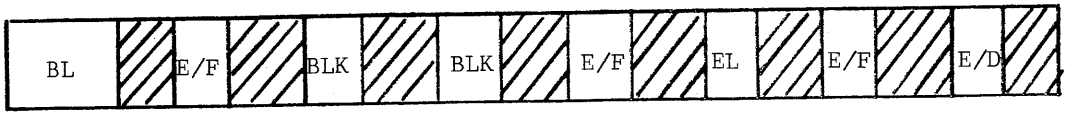
   a. Each reel of a file begins with a Beginning Label block, followed by a block consisting of a single E/F symbol.
   b. Initial and intermediate reels of a multi-reel file are terminated by three successive blocks:
      (1) A block consisting of a single E/F symbol.
      (2) An Ending Label block.
      (3) A block consisting of a single E/D symbol.
   c. The final reel of a multi-reel file, or a single reel file, is terminated by four successive blocks:
      (1) A block consisting of a single E/F symbol.
      (2) An Ending Label block.
      (3) A block consisting of a single E/F symbol.
      (4) A block consisting of a single E/D symbol.

D. Format of Data on Tape

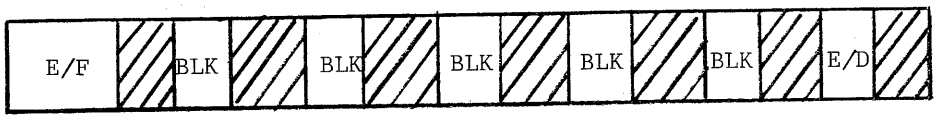
1. Initial and Intermediate Reels of a Multi-Reel File (With Labels)



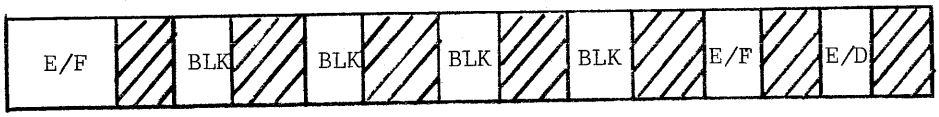
2. Final Reel of a Multi-Reel File (With Labels) or Single Reel File (With Labels)



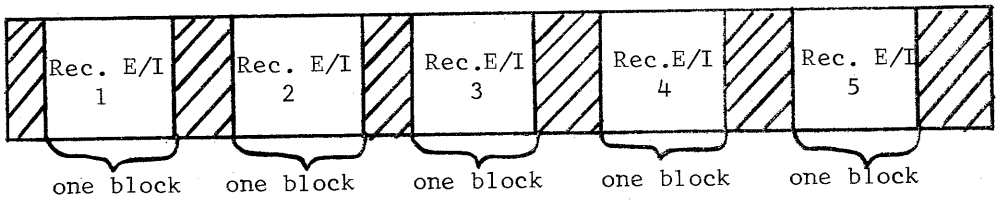
3. Initial and Intermediate Reels of a Multi-Reel File (With No Labels)



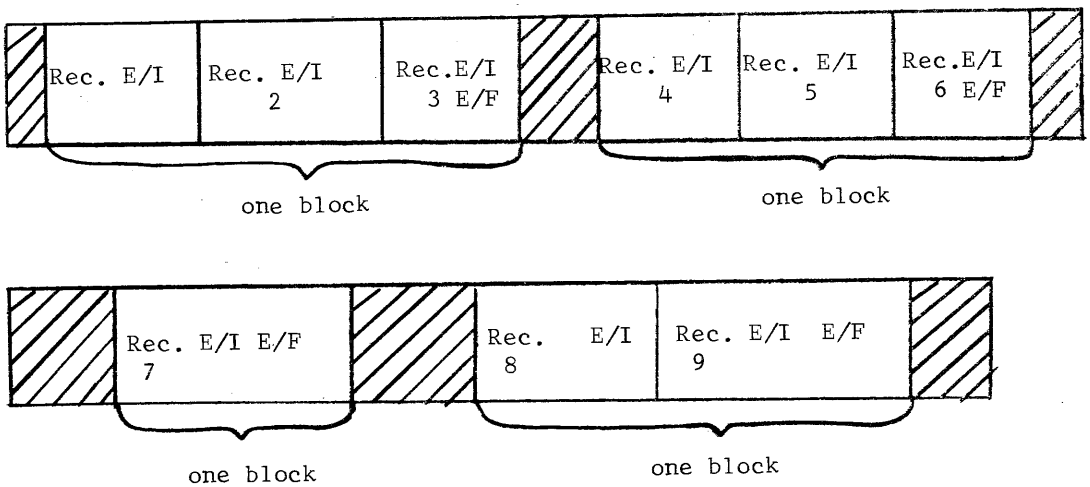
4. Final Reel of a Multi-Reel File (With No Labels) or Single Reel File (With No Labels)



5. Single Records on Tape



6. Batched Records on Tape



II Card Standards

A. Standard Label Formats

The standard label formats for magnetic and paper tape also apply to card files with the following exception:

> The STANDARD label items, REEL-NUMBER and PURGE-DATE do not have logical significance for card files. Thus, the REEL-NUMBER will always be 001 and the PURGE-DATE will always contain six zeros.

B. Card Label Conventions

The label conventions for magnetic and paper tape also apply to card files with the following exception:

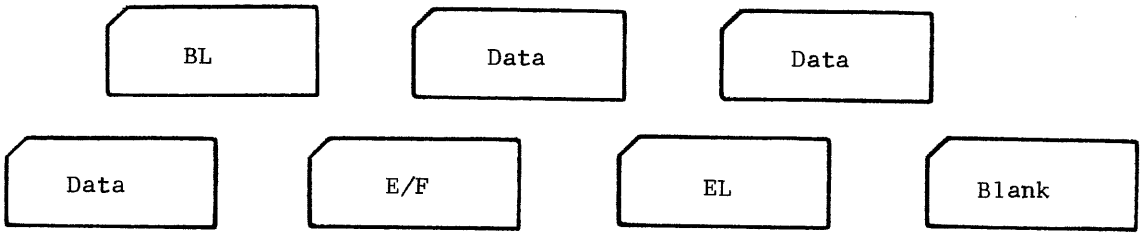
> The length of a user-defined label must not exceed 80 characters including the E/B _(space) and E/I symbols.

C. RCA 301 Organization of Data on Cards

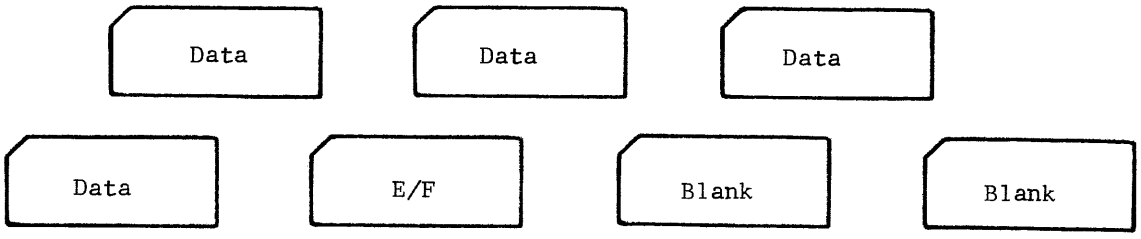
1. Data records are not terminated by an E/I symbol, nor any other symbol. Note that this rule does not apply to labels.
2. Data records may not be batched.
3. Data records must be a fixed 80 characters in length.

D. Format of Data on Cards

1. With labels



2. Without labels



Notes:

1. An E/F card signifies the end of the card file. The E/F Symbol must appear in the first column, and the remainder of the card must be blank.

2. A card file with labels must be followed by at least one blank card; a card file without labels must be followed by at least two blank cards.

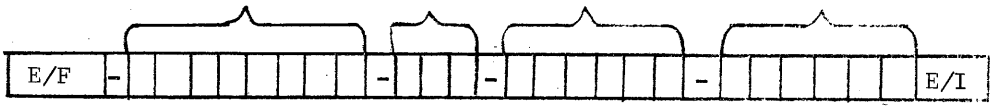
III Data Record File Standards

A. Standard Label Formats

1. Beginning Label (29 Characters)

When Beginning Labels are used, they appear in the first cell of the initial band on each record side.

IDENTIFICATION	SIDE-NUMBER	DATE-WRITTEN	PURGE-DATE
8 Chars.	3 Chars.	6 Chars.	6 Chars.



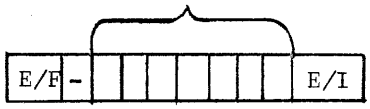
The SIDE-NUMBER is a count of the current record side in the file. The SIDE-NUMBER starts at 001 for the first record side in the file.

2. Ending Label (10 Characters)

When an ending label is used, it appears only on the last record side of a file, and must always appear in the beginning of a cell. The ending label contains the count of the number of data blocks in the file. The format is as follows:

BLOCK-COUNT

7 Chars.



B. Data Record File Label Conventions

The 301 user is permitted the following label conventions:

1. Standard Labels may be indicated.

2. Beginning and Ending Labels may be omitted.

3. A Beginning Label may be specified without an Ending Label, however, an Ending Label may not be specified without a Beginning Label.

4. The user may specify the format of the entire label(s) in which none of the standard items appear.

5. The E/B symbol must always follow the E/I symbol in a Beginning Label.

6. The E/F E/B or E/F E/D E/B symbols must always appear following the E/I symbol in the Ending Label.

7. Additional information may appear in the label following all standard items. This information must precede the E/I symbol.

C. RCA 301 Organization of Data on the Data Record File

1. All data records are terminated by an E/I symbol.

2. No records may be split over cells.

3. All records must be fixed in size. If batching is used, the batch must contain a fixed number of fixed-length records.

4. Data in each cell is terminated with an E/B symbol. Data is restricted to 897 characters per cell to allow for the writing of E/F E/D E/B in the same cell.

5. When labels are not present, the following sentinels can appear immediately following the E/I symbol of the last record of the last used cell, or in the very next cell of the band, as indicated. When labels are present, the following sentinels must always appear following the last character of the Ending Label, as indicated.

   a. The sentinel E/D E/B terminates a partially filled record side. Additional information for the same file would appear on the next logical side to be processed. (Note that this sentinel is never used following an Ending Label.)

   b. The sentinel E/F E/D E/B indicates the end of data for the last file to be processed.

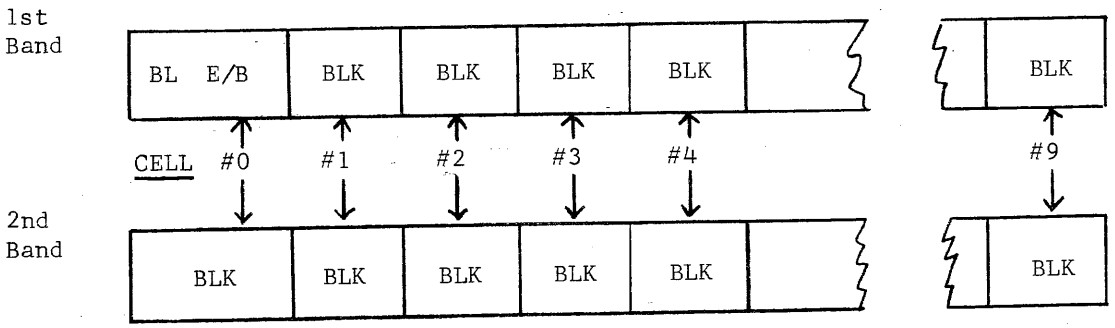
   c. The sentinel E/F E/B indicates the end of data for a file.

6. The five character sentinel E/F END E/B signifies the end of data in the first band of a record side and immediately follows the E/I symbol of the last data record. If there is insufficient space in the last data cell it will appear as the first five characters of the following cell. When this sentinel appears, the first cell in the second band must contain the next data block.

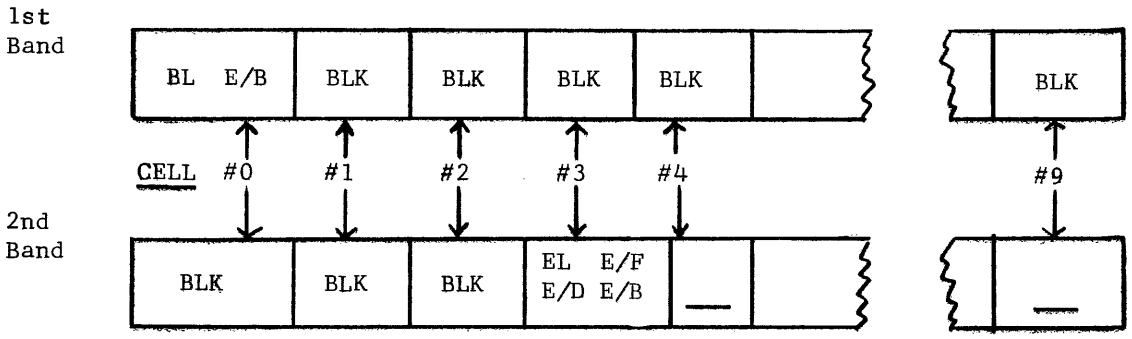
   This sentinel is not used if data appears in all cells of a band or if the sentinels E/F E/B, E/D E/B, or E/F E/D E/B appear in the band.

D. Format of Data on Data Record File

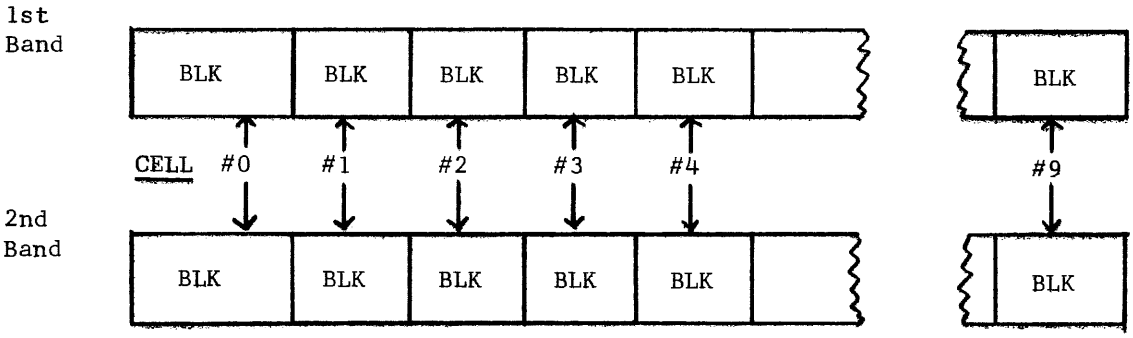
1. Initial and Intermediate Side of a Multi-Side File (With Labels)



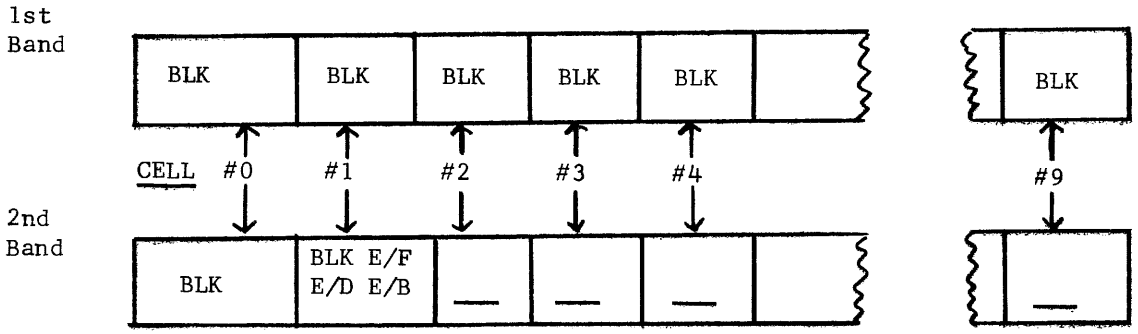
2. Final Side of a Multi-Side File (With Labels) or Single Side File (With Labels)



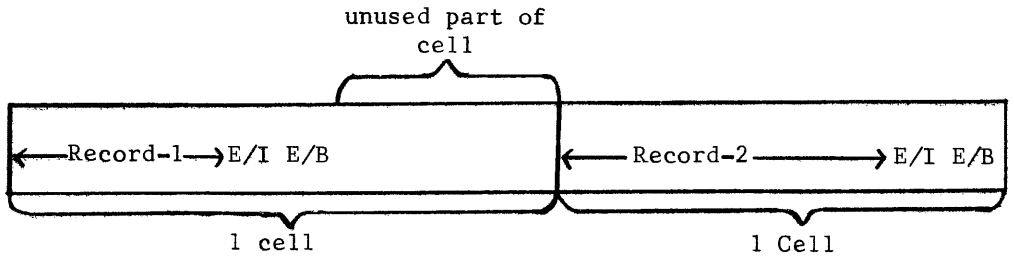
3. Initial and Intermediate Side of a Multi-Side File (With No Labels)



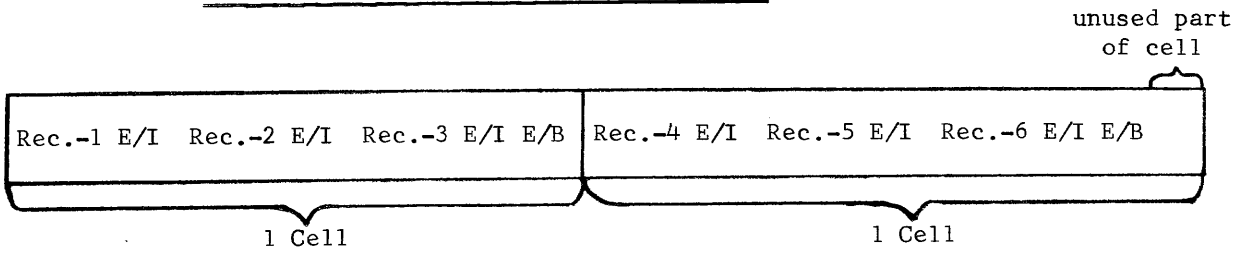
4. Final Side of A Multi-Side File (With No Labels) or Single Side File (With No Labels)



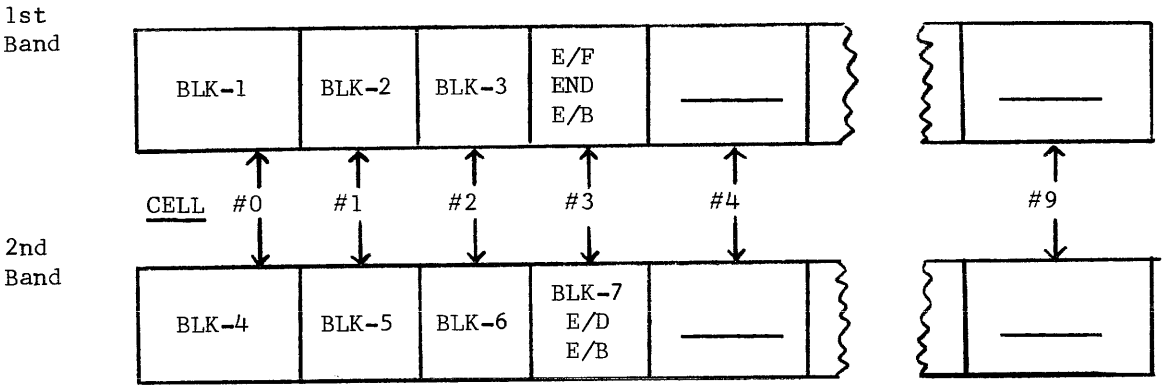
5. Single Records on the Data Record File



6. Batched Records on the Data Record File



7. Partially Filled Record Side



XXXVII – RCA 301 SUPPLIED ROUTINES

To date, the programs we have discussed have been "Production" Programs. That is, they perform some function of the particular installations application. There are other programs necessary for EDP, however.

In an effort to facilitate the user's development and maintenance of an efficient system, RCA provides a series of "service routines". Service routines are special computer programs for performing tasks common to all Electronic Data Processing Systems. For example, at the point that the production programs are operational (i.e., performing the task for which they were written), they must be completely free of logic or coding errors, in order to produce accurate results. The checking of these programs is called "debugging", which simply means getting any "bugs" or mistakes out of the programs. To aid in this procedure, RCA supplies to its users a series of service routines designed and written for this purpose.

Since a basic understanding of the service routines available is a prime requisite of a good programmer, we will investigate the philosophy and functions of the RCA 301 Service Routine System. We will not go into the use and operation of these routines, since it would be more detail than we need at this point. Such information can be obtained from the RCA 301 Service Routines Specifications or the RCA 301 Service Routine Manual.

Since the RCA 301 is a flexible system with regard to input-output equipment, and long term storage, it is only logical that the Service Routine System be developed to be adaptable to any particular installation's needs. For this reason, there are actually three Service Routine libraries. The first is written for magnetic disc storage of the programs; the second is written for a magnetic tape library, and the third is written for a card library. We will discuss each of these in turn.

DATA RECORD FILE LIBRARY:

To understand the Systems Operation, it is necessary to first understand the library concept. In this system we are discussing a disc library. In other words, each program is maintained on one or more discs, and there is never more than one program on a disc. The first cell of each disc contains information which identifies the program contained on the disc. The first segment of the program may not exceed 700 instructions; each segment must start on a new cell. The program discs are placed in the basket, so that the disc which contains the insertion routine, which we will discuss in a moment, is located at a standard location. Following that are the discs containing the programs which are to be used and the last disc is called a terminal disc, which simply indicates that beyond this point there are no more programs.

The function of the Insertion Routine is to read the first segment of the desired program into memory and set the P register to address the first instruction. This insertion routine can be set up to do a single insertion or automatic multiple insertions. The single insertion will load only one program (the program desired is designated by a "load message", which is input to the Insertion Routine), execute this program, stopping at EOR. For run-to-run operation, the discs must be arranged in the order of processing sequence, following the insertion disc, and terminated by the terminal disc. The bootstrap routine, (a short routine maintained in memory for the purpose of calling in the insertion routine) brings in this insertion program, which in turn calls in the first segment of the program, sets the P register and stops for operator convenience. The operator can then start the execution of the

program by depressing the Start button. Additional segments of the program must be called in from the discs by the coding. When EOR has been reached, the program must transfer to a common exit location, which in turn transfers to the bootstrap and the process is repeated for the second program. The procedure will continue until the terminal disc is sensed, at which point the computer will stop operation.

The next classification of Service Routines in this library is the General Processing Routines. These include the Multiplication and Division subroutines discussed earlier, the Sort and Merge routines to be discussed in the next chapter, and such other routines as:

1. File Maintenance, which will position, correct, duplicate, and/or compare tape or disc stores files.

2. Tape-To-Printer, which edits information from 301 or 501 tapes and prints the information on the on-line printer.

3. Card-To-Tape or Tape-To-Card, which edits information from 301 or 501 tapes and punches cards, or reads cards and prepares tapes for 301 or 501 use.

4. Abstract routine, which is used to obtain a summary of the information contained on magnetic tape.

5. Disc Print routine, which prints the information contained on specified discs.

Machine Code Oriented Programs

Program preparation in machine coding means that the program is prepared for debugging once it has been coded, desk checked, and prepared as input to the computer. The Program Disc Transcriber routine will read the information from punched cards or paper tape, relocate the instructions to a new HSM location if the program is floatable, and then transcribe the program to the disc in proper format. A print copy of the program can be obtained by using the Program Disc Print routine.

MAGNETIC TAPE LIBRARY:

The second major RCA 301 Service Routine Library is designed around the use of magnetic tapes for program storage.

The Tape Library routines are designed to allow for COBOL oriented information or machine code oriented data.

COBOL Oriented Service Routines:

The first step in any program is development of the coding. Here we are discussing a program written according to COBOL specifications. Once this has been written and desk checked by the programmer and his associates, it must be prepared as input to the COBOL program, which will translate it to 301 machine code.

Once this is obtained, the next step is program testing. In order to test the accuracy of a program, it is necessary to develop good test data (sample records that will force the program down every possible path). With this sample data and the predetermined results, it is possible to check the program and see if it is acting correctly. If not, corrections must be made. To aid in this procedure, there are a number of Service routines, called the RCA 301 COBOL VALIDATION SYSTEM, often referred to as the CONSOLIDATA routines: The CONSOLIDATA routines are a set of remote program check-out routines. It contains a number of routines which will:

1) allow the programmer to package all of his validation routines, test data, and the program itself onto one or more discs for ease of handling;

2) permit him to distribute the information on the disc(s) to the appropriate storage facilities in order to start debugging, thus greatly cutting down set up time;

3) sample data fields at designated points during debugging, giving the programmer a printed output of this information;

4) print out any designated areas of memory, as well as the contents of the output tapes, when the program hangs. In addition, the position of the input tapes will be given.

When the errors have been discovered, corrections can be made in COBOL and a recompilation can be made, producing a corrected machine program. When the machine program finally tests out perfectly, it can then be incorporated as part of the program library.

Programs initially prepared in machine coding are fed into the computer through either paper tape or cards. The Program Tape Transcriber routine floats the program, if necessary, and then transcribes it to magnetic tape in the proper program tape format. A printed copy of the program may be obtained by using the Program Tape Print routine.

Program testing would incorporate an Insertion routine, which would load the program into memory. The HSM Print routine would give the programmer a picture of memory at the point that the program hung, and the Tape Print routine would print out any information or the output tapes. The File Maintenance routine would be used to correct any errors which are found during the testing procedures.

The new program, having been completely checked out, is stored on magnetic tape, either in a general library or as part of a "run tape", which is used in an automatic multiple program run. The Insertion routine is used to insert the first program segment (limit for a segment is 700 instructions) into HSM. (The program itself must incorporate steps to load all additional segments). This Insertion routine can handle single or multiple program insertions.

Multiple program insertions can be accomplished by compiling a "run tape". The first three blocks of the tape must be the Insertion Bootstrap and the Insertion routine. The first block must be manually read into a standard HSM area, the program tape number placed in a standard HSM location (2000), and the P register must be set to a standard location. Then hitting the Start button will bring in the Insertion routine, which in turn loads the first segment of the first program on the "run" tape. The Insertion routine will cause a stop for operator convenience, but hitting the Start button will cause the execution of the program. Termination of the program must be a transfer to a standard exit location, which will cause a return to the Insertion routine. This procedure will continue until all the programs on the run tape have been processed.

Single program insertions can be accomplished in the same manner, except that only that portion of the Insertion routine which handles single insertions is brought into memory. The identification of the program to be executed is fed to the computer through either paper tape or cards. When the program is completed, the operation terminates.

The General Processing routines are the same as for the Disc Library, except that the Sort and Merge routines will be for tape only, since a Record File is not available.

CARD LIBRARY:

This system was designed to meet the needs of an RCA 301 basic card system. Again, the programs are all machine code oriented.

Program preparation is the same as before, when dealing with machine code, except that now the program must be prepared on punched cards.

Testing can be accomplished by use of the Loader routine, which will place the program into memory at the designated locations (floating it, if necessary), and the High Speed Memory Print, which will prepare hard copy pictures of memory. Again, the debugging procedure will be a run-to-hang operation. The Loader routine can be used to correct any errors found in the program during testing. The initial wrong cards do not have to be removed from the deck, unless desired, if the correction cards are placed as the last cards in the deck. This is true since each card is treated individually, and the information on the correction cards would simply overlay the erroneous data.

System operation can be accomplished by using the Loader routine to bring in the production program. The Loader routine cards must preceed the production program cards and the insertion of the Loader routine must be manual from the console, as must be the setting of the P register, in order to initiate the running of the Loader routine. The Loader routine need not be inserted each time, as long as the production program does not destroy it.

The General Processing routines are generally the same as those available for the other two libraries.

XXXVIII – COMPATIBILITY

Having discussed the RCA 301 as an individual system, it is necessary to mention that the RCA 301 is also compatible with the other RCA EDP Systems, such as the RCA 501 and the RCA 601. For this reason, it is feasible that many applications will use the RCA 301 as a "satellite" computer to these larger systems. This compatibility is possible due to several inherent features of the RCA 301.

In order to be compatible, it must be possible to feed data from one computer to the next. This can be accomplished by one of two methods. First of all, the 33 kc and 66 kc magnetic tape(s) which can be connected to the 301 can be utilized by both the 501 and 601 systems. Secondly, RCA has a computer communications system called Da-Span, which permits data to be sent over telegraph or telephone lines, thus tying distant computer systems together. This long distance, rapid communication between the master and satellite computers achieves the long desired results of immediate electronic data processing even in an industry which is widely dispersed geographically.

Once the data is obtained, it must be converted from the original code to the RCA 301 code. As was pointed out earlier, the Translate by Table instruction will convert any 7 bit binary code to any other 7 bit binary code. This means that 501 or 601 code could be translated to 301 code or vice-versa, thus making all the codes compatible.

TAPE READ REVERSE NORMAL:

This instruction operates exactly as the TAPE READ FORWARD NORMAL, except that the tape is moving in a reverse direction and the data is being placed in memory from right to left.

The OP code is 6.

N indicates the tape unit or the paper tape reader.

The A address gives the location to receive the first character.

The B address gives the location to receive the last character.

The operation terminates when a gap is sensed on tape. If this condition should exist before A equals B, PRN is set. If the gap is found at the same time that the registers become equal, PRZ is set. If the registers reach equality, no more data will be transferred into memory, but the tape will keep moving until the gap is sensed. In this case the PRP is set.

A_f (STA) will contain the address one to the left of the last character read in. This instruction is repeatable.

TAPE READ REVERSE SIMULTANEOUS:

This instruction is the same as the TAPE READ REVERSE NORMAL except that it is executed in the Simultaneous Mode.

The OP code is 7.

Everything else is the same.

The PRI's are not affected, but S_f will contain the address one to the left of the last character read in.

The Merge routine is a separate routine which will collate data that is stored on from two to eight tapes, preparing output in sequential order. Again the data must be fixed in length, but it may be batched. The output batch is specified by the user and own coding and re-run are permitted.

XXXIX – SORTING AND MERGING

Since transactions come in in random order, and since it is normally not efficient to post them against their masters in this random order, it is necessary to have some way to order or "sort" this raw data into sequential order based on the same criteria by which the master file is sorted. Manually, this could be accomplished by waiting until the transactions are in, sorting the hard copy, and then preparing the paper tape for input, or by sorting the card transactions before entering them into the computer run. This, however, is seldom efficient or even practical (especially in the case of paper tape input), so that it is necessary to have a program which will read this raw data, sort the records based on a specified criteria, and prepare a final output which contains the information in sorted order. The sort and merge programs needed to accomplish this are service routines supplied by RCA to its users. These routines are generalized in nature, that is the sort could be used on payroll data in one format one time, and stock information in an entirely different format the next time. The only requirements are that data records must be fixed in length and maximum limits as to record size and criteria size are not exceeded. Since the RCA 301 may have many configurations of peripheral gear, it is necessary to have more than one Sort routine. Specifically, there will be sorts to handle the following specifications:

1) a 2 way merge sort using a 10K or 20K memory and one Hi-Data tape group

2) a 2 to 5 way merge sort using 20K memory and two Hi-Data tape groups

3) a 2 way merge sort using 10K memory and one Record File

4) a 2 way merge sort using 20K memory and one Record File

In addition, the sort routines will allow for:

1) disc or tape program storage of the tape sorts (the disc sort will be written only for disc storage)

2) input from magnetic tape

3) operation in the Normal Mode only or in the Normal and Simultaneous Modes

The output of the sort routines will be ascending or descending order and will be stored on the medium of the sort (magnetic tape or disc) in the batch size specified by the user.

Each installation will be able to include some own-coding (instructions to perform a task unique to that particular installation, but not common to all EDP systems) during the first and last passes of each of the sorts. Re-run (the ability to re-start without having to begin at the beginning) will be available during the general merge pass.

The Sorts will be made up of internal sort, which will develop strings or batches of sorted data until the input data is exhausted and then a merge pass which will merge these strings into batches of the specified length.

The tape sort routines incorporate two instructions which we have not yet covered. These are the Reverse Reads. The advantage of these instructions is fairly obvious if we consider that sorting is actually reading, writing, reading the previously written data, re-writing it in another order, etc. The reverse reads permit reading the written information without the necessity of rewinding the tape, cutting the tape time drastically.

APPENDIX A – MICR SORTER READER

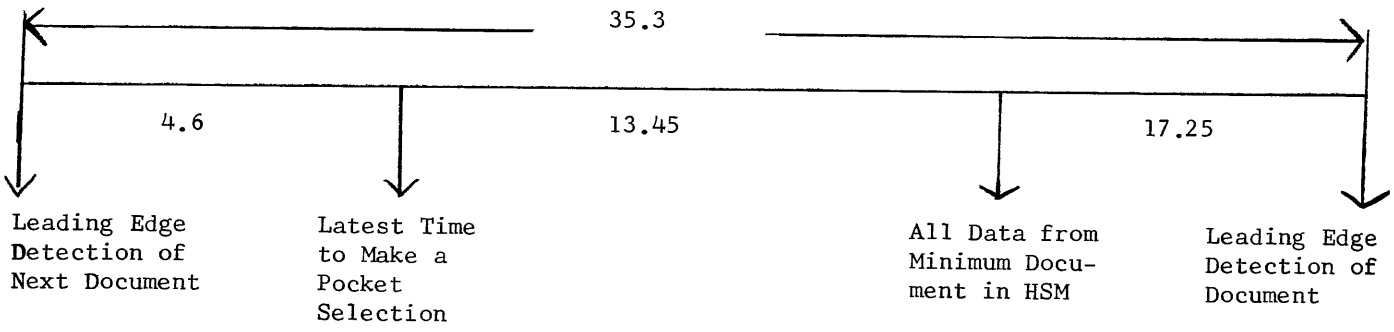
A MICR Sorter/Reader is an on-line device which enables a computer, under program control, to read total or partial images of a paper document into HSM. The computer can accumulate and verify data from magnetically encoded documents and control the sorting of these documents into the stackers or pockets of the Sorter Reader.

The Sorter/Reader has the ability to read 14 magnetic characters. These consist of ten numerals (0 thru 9) and four special symbols (amount symbol, on-us symbol, dash symbol, and transit number symbol). The numerals are represented in RCA 301 as numerals. The amount symbol is represented in the RCA 301 as a dollar sign ($), the on-us symbol as an equals sign (=), the dash symbol as a minus sign (-), and the transit number symbol as a number sign (#).

The Sorter/Reader has a maximum rated speed of 1,560 6-inch documents per minute. A timing cycle of 35.3 ms per document should be used. Approximately 31.424 ms are available for computing during a timing cycle.

MICR TIMING

The 35.3 ms it takes to process one document is divided up as follows:



As can be seen, 17.25 ms are needed from the detection of the reading edge of a document to the completion of reading 42 characters into HSM. This leaves 18.05 ms remaining for processing. Moreover, since it takes a minimum of 1.680 ms to load the 6 character buffer and only 91 us to transfer the contents of the buffer to HSM, even more free computing time is available. (This is assuming 7 read instructions reading 6 characters each) A read instruction may read any number of characters in multiples of 6. It is not limited to 6 characters. However, if more than 6 characters are read with one instruction, free computing time between buffer dumps is not available). Subtracting staticizing, reading, and STA time (91 us) from 1.680 ms leaves 1.589 ms available for computing.

The fifth time the buffer is dumped into HSM a series of automatic checks are performed which decreases the available compute time to 1.029 ms.

The time available between the detection of the leading edge of a document and the latest time that the first 6 characters may be read is 4.700 ms. Because of the subroutine necessary for leading edge detection, this time is cut to a maximum of 4.4 ms.

Therefore the maximum total free compute time in one document time cycle is:

	4.400 ms
1.589 X 5 Buffer Dumps	7.945 ms
1.029 after Fifth Dump	1.029 ms
18.050 after Last Dump	18.050 ms
	31.424 ms Computing Time

If processing is going on in the simultaneous mode, of course, this time will be interrupted and therefore decreased.

MICR INSTRUCTIONS

Control Document Feeder

This instruction starts or stops the feeding of documents into the transport mechanism of the Sorter/Reader.

The OP code is ; .

N - 0 (Zero)

A Address - 0000 (Zeros)

B Address - B_0, B_1, B_2 = 000 (Zeros)
B_3 0 1 = START
B_3 - 0 = STOP

Document Read Reverse Normal

This instruction reads information from magnetically encoded documents in the Sorter/Reader into HSM via the MICR Control Buffer.

The OP code is 6.

N - 0 (Zero)

A Address - HSM Location to Receive the First Character Read

B Address - HSM Location to Receive the Last Character Read

When the MICR Control Buffer is filled with 6 characters, the first character is transferred into the HSM location addressed by the A register. The contents of the A & B registers are compared and if equal, PRZ is set and the instruction terminates. If unequal the A register is decreased by 1 and the process repeats. Should the buffer be empty, continued execution of this instruction is held of until the buffer is filled. If a malfunction occurs or no documents are in transport, the instruction terminates and PRN is set. If the leading edge of a document is sensed and the instruction is awaiting initial execution, PRP is set and the instruction terminates. (Blank fields are treated as spaces)

Document Read Reverse Simultaneous:

The OP code is 7.

N - 0 (Zero)

A Address - Address of HSM Location to Receive First Character Read

B Address - Address of HSM Location to Receive Last Character Read

There are no PRI settings for this instruction.

Pocket Select:

This instruction designates the pocket in which the processed check can be placed.

The OP code is 8.

N - 0 (Zero)

A Address - HSM Location of pocket number

B Address - HSM Location of pocket number

Possible pocket numbers:

0
1
2
3
4
5
6
7
8
9
+
- (minus)

If this instruction is not given the document will go to the reject pocket.

Input Output Sense:

The OP code is S.

N - 0 (Zero)

A Address - A_0 specifies the test to be performed as follows:

Ao	TEST
1	Is the Sorter/Reader inoperable?
2	Has the feeder been turned off?
4	Has transporting problem caused delay of feeding?
8	Is feeder hopper empty?
&	Was pocket selection in error?

A_2, A_2, A_3 = 000 (Zeros)

B Address - HSM Location of next instruction if the condition or conditions being tested are present.

Documents are automatically rejected to the reject pocket if:

1. They are too close in transport
2. They are too large
3. A pocket selection is not made

APPENDIX B — ARITHMETIC INSTRUCTIONS

In addition to the arithmetic instructions and subroutines we have discussed, there are other arithmetic instructions which may be used with a 20K or 40K processor if a high speed arithmetic unit is attached. This unit and Fixed and Floating Point Arithmetic Instructions provide for higher forms of arithmetics such as the multiplication and division of exponential numbers.

The high speed arithmetic unit consists of an addressable Accumulator and Product Remainder Register, two non-addressable arithmetic registers, and a high speed parallel adder. The addressable Accumulator and PR Register are 16 positions in length (8 each) and are not contained in HSM. The result of every arithmetic operation appears in the Accumulator and, if indicated, in memory. In some cases the Accumulator and the PR Register will contain pertinent information after an arithmetic operation. The contents of the Accumulator may be addressed, stored or shifted and the PR Register may be stored or shifted.

The addresses of the Fixed and Floating Point instructions can be automatically modified by three different Index Fields. These are 4 characters in length and are contained in standard HSM locations. Three Increment Fields, each associated with a particular Index Field, are also available. These are 4 characters in length, contained in standard HSM locations, and are used to increment the Index Fields via the Tally instruction.

Indexing and Incrementing are algebraic additions. The range of quantities that can be used are -19,000 to +19,999 or -39,999 to +39,999 in 20K or 40K processors respectively. If the result of Indexing produces a negative address or an address which exceeds memory an alarm stop follows. Only one level of Index Field modification is permitted for each address of each instruction.

A fixed point operand is a quantity which must be 8 characters in length. The decimal point can be assumed to anywhere within the operand. It is up to the programmer to align the decimal points prior to addition or subtraction and calculate its position after a multiplication or division operation.

A floating point operand must be 10 characters in length. The most significant 8 digits indicate the numerical value of the number and are called the mantissa. The least significant 2 digits indicate the magnitude of the mantissa and are called the exponent. All Floating Point Operands must be "normalized" before being used in arithmetic operations.*

This means that the exponent must be adjusted (done by processor) so that the assumed decimal point is to the left of the MSD of the mantissa. After floating point operations are completed, the exponent indicates the position of the decimal point.

All operands must have their MSD in an even HSM location and must be addressed by their least significant diad. All operand characters should be numeric unless indicating a negative number or an overflow condition.

MSD OF	OVERFLOW
1. Fixed Point Operands	"one" bit in the 2^4 position
2. Exponent	"one" bit in the 2^4 position

LSD OF	NEGATIVE
1. Fixed Point Operands	"one" bit in the 2^5 position
2. Exponent	"one" bit in the 2^5 position
3. Mantissa	"one" bit in the 2^5 position

An overflow condition is not possible in the mantissa. If attempted the Accumulator would shift one position to the right, a decimal "one" would be inserted in the MSD of the mantissa, and the exponent would be adjusted accordingly. Overflow is possible only after an arithmetic operation. If present when initiating an arithmetic operation an alarm stop will occur.

All floating point operation results are automatically normalized and rounded. Fixed point operation results are not.

Examples of 301 floating point format:

NUMBERS			NORMALIZED NUMBERS
+ 0.12345678	=+	$.12345678X10^0$	1234567800
+123.45678	=+	$.12345678X10^3$	1234567803
-123.45678	=-	$.12345678X10^3$	1234567Q03
0	=	$0 X10^0$	0000000000
+ 0.00012345	=+	$.12345 X10^{-3}$	123450000L
- 0.00000001	=-	$.1 X10^{->}$	10000000-OP
+ $.123456789X10^{79}$			1234567879

Any number between 10^{-99} to 10^{+99} may be represented in floating point format.

HSM LOCATIONS	CONTENTS
0226-0229	Increment A Index Fields
0230-0233	A Index Fields
0234-0237	Increment B Index Fields
0238-0241	B Index Fields
0242-0245	Increment AB Index Fields
0246-0249	AB Index Fields

ARITHMETIC INSTRUCTIONS

The N code in each arithmetic instruction designate the type of address modification (if any) and the location of the arithmetic operands. The following table can be used to determine the N character for any particular instruction.

ADDRESS		A,B	A	B	A,B	A,B
ADDRESS MODIFIED BY INDEX FIELD:		None	A	B	A,B	AB
OPERAND RESULT LOCATIONS	A= contents of A Address B= contents of B Address Result stored into contents of A Address and accumulator	0	4	2	6	1
	A= contents of accumulator B= contents of B Address Result stored into contents of A Address and accumulator	-	M	K	Q	J
	A= contents of A Address B= contents of accumulator Result stored into contents of A Address and accumulator	&	D	B	F	A
	A= contents of accumulator B= contents of accumulator Result stored into contents of A Address and accumulator	"	U	S	W	/
	A= contents of A Address B= contents of B Address Result stored into accumulator	8	@	(space)	)	9
	A= contents of accumulator B= contents of B Address Result stored into accumulator	Q	*	E/I	E/F	R
	A= contents of A Address B= contents of accumulator Result stored into accumulator	H	;	+	:	I
	A= contents of accumulator B= contents of accumulator Result stored into accumulator	Y	%	E/B	=	Z

Staticizing and execution time for each instruction will be indicated under its description. Additional time, depending upon the specific instruction, may be applicable.

CONDITION	ADDITIONAL TIME
Indexing an address	-21 us
Fetching a fixed point operand from HSM (A or B)	-28 us
If and "end around condition" exists in an operand	- 7 us
Storing a fixed point result in HSM	-28 us
Fetching a floating point operand from HSM (A or B)	-35 us
Alignment of an operand during floating point operations	- 7 us PER SHIFT
Normalization of operands during floating point operations	- 7 us PER SHIFT
Rounding of a floating point result	- 7 us
Storing a floating point result in HSM	-35 us

@ Fixed Point Add (FXA) Non-Repeatable

General Description

This instruction performs decimal addition in accordance with algebraic rules producing a non-zero suppressed sum in the Accumulator. Depending upon the conditions as specified by the N character, the addresses of the operands can be indexed and the operands may appear in HSM or in the Accumulator. The sum may be placed in HSM if it is indicated. The operands are eight characters in length and obey the rules concerning addressing sign, and overflow as described in the fixed point word format description.

Format

Operation ___@
N ___as discussed previously.
A Address ___HSM location of least significant diad of augend and/or sum.
B Address ___HSM location of least significant diad of addend.

Direction

Right to left.

PRI's

PRP is set if the sum is positive.
PRZ is set if the sum is zero.
PRN is set if the sum is negative.

Timing

35 us + 7 us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
@	0	2009	2017

HSM Before Instruction is Executed.

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
8	2	3	4	1	0	2	1	0	3	6	7	7	7	8	0

HSM After Instruction is Executed.

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
8	6	0	1	8	8	0	1	0	3	6	7	7	7	8	0

Contents of Accumulator After Instruction is Executed.

8	6	0	1	8	8	0	1

Accumulator

PRI

PRP is set.

(Fixed Point Subtract (FXS) Non-Repeatable

General Description

This instruction performs decimal subtraction in accordance with algebraic rules producing a non-zero suppressed difference in the Accumulator. Depending upon the conditions as specified by the N character, the addresses of the operands can be indexed and the operands may appear in HSM or in the Accumulator. The difference may be placed in HSM if it is indicated. The operands are eight characters in length and obey the rules concerning addressing sign, and overflow as described in the fixed point word description.

Format

Operation ___ (
N ___ as described previously.
A Address ___ HSM location of least significant diad of the minuend and/or difference.
B Address ___ HSM location of least significant diad of the subtrahend.

Direction

Right to left.

PRI's

PRP is set if the difference is positive.
PRZ is set if the difference is zero.
PRN is set if the difference is negative.

Timing

35 us + 7 us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
	8	3363	3371

HSM Before Instruction is Executed

3356	3357	3358	3359	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371
3	7	4	3	0	7	8	9	6	1	3	4	3	2	1	R

HSM After Instruction is Executed

3356	3357	3358	3359	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371
3	7	4	3	0	7	8	9	6	1	3	4	3	2	1	R

Accumulator After Instruction is Executed

9	8	7	7	4	0	0	8

PRI

PRP is set.

) - Fixed Point Multiply (FXM) Non-Repeatable

General Description

This instruction performs decimal multiplication in accordance with algebraic rules producing a non-zero-suppressed product in the Accumulator and the PR Register. The LSD of the product will be in the LSD position of the PQ Register. The sign of the product will be indicated only by the 2^5 bit of the LSD of the Accumulator.

Depending on the conditions as specified by the N character, the addresses of the operands can be indexed and the operands may appear in the HSM or in the Accumulator. The product may be placed in HSM if it is indicated. The operands are eight characters in length and obey the rules concerning addressing as described in the fixed point word description.

If one operand is zero, the resultant zero product will produce all zeros in both the Accumulator and the PR Register. Execution time for the multiply will be faster if the multiplier is the smaller operand.

Format

Operation ___)
N ___ as described previously.
A Address ___ HSM location of the least significant diad of the multiplicand and/or product.
B Address ___ HSM location of the least significant diad of the multiplier.

Direction

Right to left.

PRI's

PRP is set if the product is positive.
PRZ is set if the product is zero.
PRN is set if the product is negative.

Timing

Average Execution Time 35 us + 315 us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
)	4	3105	4615

HSM Before Instruction is Executed.

4600	4601	4602	4603	4604	4605	4606	4607	4608	4609	4610	4611	4612	4613	4614	4615
2	1	0	3	4	2	9	L	6	8	3	4	2	7	6	5

A - IR

1502

HSM After Instruction is Executed.

4600	4601	4602	4603	4604	4605	4606	4607	4608	4609	4610	4611	4612	4613	4614	4615
1	4	3	7	5	4	1	P	6	8	3	4	2	7	6	5

Contents of Accumulator and PR Register After Instruction is Executed.

1	4	3	7	5	4	1	P	4	3	4	4	0	1	4	5

Accumulator PR Register

PRI

PRN is set.

& - Fixed Point Divide (FXD) Non-Repeatable

General Description

This instruction performs decimal division in accordance with algebraic rules producing a non-zero suppressed quotient in the Accumulator and Remainder in the PQ Register. The sign of the quotient will be indicated by the 2^5 bit of the LSD of the Accumulator. The sign of the dividend will be retained and indicated by the 2^5 bit of the LSD of the PR Register. If the decimal point of the operands are aligned, the decimal point of the quotient will precede the MSD.

The absolute value of the dividend must be less than that of the divisor. If this rule is violated, the computer stops with a ARIE Alarm Stop.

Depending on the conditions as specified by the N character, the addresses of the operands can be indexed and the operands may appear in HSM or in the Accumulator. The quotient may be placed in HSM if it is indicated. The operands are eight characters in length and obey the rules concerning addressing as described in the fixed point word description.

If the dividend is zero, the quotient will be zero, producing all zeros in both the Accumulator and the PQ Register. If the divisor is zero, the computer stops with a ARIE Alarm Stop.

Format

OP ___ &
N ___ as described previously.
A ___ HSM location of the least significant diad of dividend and/or quotient
B ___ HSM location of the least significant diad of divisor.

Direction

Right to left.

PRI's

PRP is set if the quotient is positive.
PRZ is set if the quotient is zero.
PRN is set if the quotient is negative.

Timing

Average execution time - 35 us + 322 us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
&	0	4583	4575

HSM Before Instruction is Executed.

4568	4569	4570	4571	4572	4573	4574	4575	4576	4577	4578	4579	4580	4581	4582	4583
0	0	0	3	2	1	0	0	0	0	0	0	6	7	8	9

HSM After Instruction is Executed.

4568	4569	4570	4571	4572	4573	4574	4575	4576	4577	4578	4579	4580	4581	4582	4583
2	1	1	4	9	5	3	2	0	0	0	0	6	7	8	9

Contents of Accumulator and PR Register After Instruction is Executed.

2	1	1	4	9	5	3	2	0	0	0	2	2	8	0	0

Accumulator PR Register

PRI

PRP is set.

$ - Floating Point ADD (FLA)

General Description

This instruction performs floating point decimal addition in accordance with algebraic rules producing a rounded normalized floating point sum in the Accumulator. Depending upon the conditions specified by the N character, the addresses of the operands may be indexed and the operands may appear in HSM or in the Accumulator. The sum may be placed in the HSM if it is indicated. The operands are ten characters in length and obey the rules concerning addressing, sign, and overflow as described in the floating point data format description.

The Floating Point Add sum is rounded in the following manner:

During the alignment of the operands, the mantissa of the operand containing the algebraically smaller exponent will be shifted right with overflow digits entering the PQ Register. The addition is extended to these digits in the PQ Register with zeros assured for the extension of the operand with the algebraically larger exponent. If normalization requires the sum to be shifted left, the required number of digits will be shifted from the PR Register to the Accumulator. Upon completion of normalization, the MSD of the PR Register will be examined, and if 5 or greater, a one will be added to the LSD in the Accumulator. In the exceptional case, where this addition produces a carry through the entire eight digits of the Accumulator, the Accumulator will automatically be shifted right one position, a one will be inserted in the MSD, and the exponent is incremented by one.

Format

Operation - $
N - as described previously.
A Address - HSM location of the least significant diad of augend and/or sum.
B Address - HSM location of the least significant diad of addend.

Direction

Right to left.

PRI's

PRP is set if the sum is positive.
PRZ is set if the sum is zero.
PRN is set if the sum is negative.

Timing

35 us + 7 us + additional timing.

Example

Operation	N	A Address	B Address
$	0	4609	4619

HSM Before Instruction is Executed.

4600	4601	4602	4603	4604	4605	4606	4607	4608	4609	4610	4611	4612	4613	4614	4615	4616	4617	4618	4619
4	6	7	0	3	4	6	9	0	4	3	4	5	6	7	3	4	5	0	1

HSM After Instruction is Executed.

4600	4601	4602	4603	4604	4605	4606	4607	4608	4609	4610	4611	4612	4613	4614	4615	4616	4617	4618	4619
4	6	7	3	8	0	3	6	0	4	3	4	5	6	7	3	4	5	0	1

PRI

PRP is set.

: - Floating Point Subtract (FLS) Non-Repeatable

General Description

This instruction performs floating point decimal subtraction in accordance with algebraic rules producing a rounded normalized floating point difference in the Accumulator. Depending upon the condition specified by the N character, the addresses of the operands may be indexed and the operands may appear in HSM or in the Accumulator. The difference may be placed in HSM if it is indicated. The operands are ten characters in length and obey the rules concerning addressing, sign, and overflow as described in the floating point data format.

The difference is rounded in the identical manner as described under the Floating Point Add instruction.

Format

Operation - :
N - as described previously.
A Address - HSM location of the least significant diad of the minuend and/or difference.
B Address - HSM location of the least significant diad of the subtrahend.

Direction

Right to left.

PRI's

PRP is set if the difference is positive.
PRZ is set if the difference is zero.
PRN is set if the difference is negative.

Timing

35 us + 7 us + additional timing.

Example

Operation	N	A Address	B Address
:	0	5837	5827

HSM Before Instruction is Executed.

5818	5819	5820	5821	5822	5823	5824	5825	5826	5827	5828	5829	5830	5831	5832	5833	5834	5835	5836	5837
1	3	6	7	3	2	0	0	0	K	8	7	3	2	1	0	1	3	0	3

HSM After Instruction is Executed.

5818	5819	5820	5821	5822	5823	5824	5825	5826	5827	5828	5829	5830	5831	5832	5833	5834	5835	5836	5837
1	3	6	7	3	2	0	0	0	K	8	7	3	2	0	8	7	6	0	3

PRI

PRP is set.

" - Floating Point Multiply - (FLM) Non-Repeatable

General Description

This instruction performs floating point decimal multiplication in accordance with algebraic rules producing a rounded, normalized floating point product in the Accumulator. Depending upon the conditions specified by the N character, the addresses of the operand may be indexed and the operands may appear in HSM or in the Accumulator. The product may be placed in HSM if it is indicated. The operands are ten characters in length and obey the rules concerning addressing and sign as described in the floating point data format description.

During normalization, digits are shifted from the PR register to the Accumulator (if required) and rounding is performed as discussed under the Floating Point Add instruction.

If either operand is zero, the Accumulator and the PR register will be all zeros, including the exponent.

Format

Operation - "
N - as described previously.
A Address - HSM location of least significant diad of the multiplicand and/or product.
B Address - HSM location of least significant diad of the multiplier.

Direction

Right to left.

PRI's

PRP is set if the product is positive.
PRZ is set if the product is zero.
PRN is set if the product is negative.

Timing

Average execution time - 35us + 322us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
"	0	6743	6743

HSM Before Instruction is Executed.

6734	6735	6736	6737	6738	6739	6740	6741	6742	6743	6744	6745	6746	6747	6748	6749	6750	6751	6752	6753
6	3	1	5	6	7	8	9	0	7	3	4	5	6	7	8	9	6	0	2

HSM After Instruction is Executed.

6734	6735	6736	6737	6738	6739	6740	6741	6742	6743	6744	6745	6746	6747	6748	6749	6750	6751	6752	6753
2	1	8	3	1	9	7	3	0	9	3	4	5	6	7	8	9	6	0	2

Contents of Accumulator and PR Register After Instruction is Executed.

2	1	8	3	1	9	7	3	1	3	8	4	5	9	4	4

Accumulator PR Register

PRI

PRP is set.

/ Floating Point Divide (FLD) Non-Repeatable

General Description

This instruction performs floating point decimal division in accordance with algebraic rules producing a rounded, normalized floating point quotient in the Accumulator. Depending upon the conditions specified by the N character, the addresses of the operands may be indexed and the operands may appear in HSM or in the Accumulator. The quotient may be placed in HSM if it is indicated. The operands are ten characters in length and obey the rules concerning addressing and sign as described in the floating point data format description.

If the first digit produced in the quotient is zero, normalization takes place immediately, so that eight digits are produced in the quotient. Rounding is performed by increasing the LSD of the quotient by one if the remainder is greater than one half the divisor. Since the quotient is rounded, the remainder becomes meaningless and is not available.

If the dividend is zero, the Accumulator and the PR Register will be all zeros including the exponent. If the divisor is zero, the computer stops with a ARIE Alarm Stop.

Format

Operation - /
N - As described previously.
A Address - HSM location of the least significant diad of the dividend and/or quotient.
B Address - HSM location of the least significant diad of the divisor.

Directions

Right to left.

PRI's

PRP is set if the quotient is positive.
PRZ is set if the quotient is zero.
PRN is set if the quotient is negative.

Timing

Average execution time - 35us + 329us + additional timing.

Example

Instruction

Operation	N	A Address	B Address
/	0	1009	1019

HSM Before Instruction is Executed.

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019
8	3	4	3	2	1	0	1	0	2	3	6	2	0	1	3	2	2	0	1

HSM After Instruction is Executed.

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019
2	3	0	4	6	7	0	0	0	2	3	6	2	0	1	3	2	2	0	1

PRI

PRP is set.

The final register settings for the high speed arithmetic instruction are primarily determined by the fetching of operands and/or storing of the result. These eight possible settings are indicated below. Where 8 characters are indicated, a fixed point operand was considered, and where 10 characters are indicated, a floating point operand was considered. Because of the diad addressability of these instructions, A_F and B_F settings will be on the corresponding character of the diad.

Final Register Setting Chart

Location of A Address or A Operand	Location of B Address or B Operand	Storage of Result	A Final	B Final
A	B	A	A_i-8 or 10 ch.	A_F
ACC	B	ACC	A_i-2 ch.	A_i
A	ACC	A	A_i-8 or 10 ch.	A_f
ACC	ACC	ACC	A_i-2 ch.	A_i
ACC	ACC	A	A_i-2 ch.	A_i-8 or 10 ch.
ACC	B	A	A_i-2 ch.	A_i-8 or 10 ch.
A	ACC	ACC	A_i-8 or 10 ch.	B_i
A	B	ACC	A_i-8 or 10 ch.	A_i

There are several instructions available which aid the user to utilize the new high speed arithmetic instructions more fully. Two of these are the Store Accumulator instruction and the Shift Accumulator instruction.

Z Store Accumulator (SAC) Non-Repeatable

General Description

This instruction places the contents of the Accumulator and the PR Register, as designated by the N character, into HSM. The sign and overflow bits are stored.

Format

Operation - Z
N code designates the type of address modification (if any) and what is to be stored. The following table indicates the possible alternatives.

ADDRESS		A,B	A	B	A,B	A,B
ADDRESS MODIFIED BY INDEX FIELDS		None	A	B	A,B	AB
To	Accumulator and PQ Reg. - 16 ch.	8	@	-	)	9
Be	Accumulator - 10 ch.	&	D	B	F	A
Stored	Accumulator - 8 ch.	⊖	M	K	⊖	J
	PQ Register - 8 ch.	0	4	2	6	I

A Address - 0000 (zeros).
B Address - HSM location of least significant diad where designated portion is to be stored.

Direction

Right to left.

PRI's

None

Final Register Settings

$A_f = A_i$
$B_f = B_i$-8, 10 or 16 characters

Timing

Staticizing - 35 microseconds
8 character store - 23 microseconds
10 character store - 35 microseconds
16 character store - 56 microseconds
Each address modification - 21 microseconds

Example

Instruction

Operation	N	A Address	B Address
Z	2	0000	4349

HSM Before Instructions are Executed

4562	4563	4564	4565	4566	4567	4568	4569
Z	Z	Z	Z	Z	Z	Z	Z

Index Field B

0	2	2	0

Accumulator and the PR Register Before Instruction is Executed

1	8	2	9	3	0	4	1	5	2	6	3	7	4	8	1

Accumulator PR Register

4562	4563	4564	4565	4566	4567	4568	4569
5	2	6	3	7	4	8	1

Timing

35us + 23us + 21us.

Final Register Settings

A_f = 0000
B_f = 4561

= Shift Accumulator (SHA) Non-Repeatable

General Description

This instruction shifts the Accumulator and the PQ Register a specified number of times in either direction. In most cases, 8 places will be the maximum shifts desired since this number permits the shifting of the entire contents of the Accumulator or PQ Register. In the rare cases where more than 8 places are desired to be shifted, up to 15 places may be shifted by indicating the number of places in binary notation. (Use 301 N COUNT).

Characters are shifted off the end of the Accumulator and lost, while vacated positions on the other end are filled with zeros. The sign positions in the Accumulator and PQ Register are not affected by shifting. However, the overflow bit (if present) is destroyed.

Format

Operation - =
N - 2^0 = 1 A and B addresses are modified by the AB Index Field.
= 0 Ignored
2^1 = 1 B address modified by B Index Field
= 0 Ignored
2^2 = 1 A address modified by A Index Field
= 0 Ignored
2^3 = 1 Couple Accumulator and the PQ Register (shift as one unit)
= 0 Uncouple Accumulator and the PQ Register (shift separately)
2^4 = 1 Shift Right
= 0 Shift Left
2^5 = 1 Shift Accumulator
= 0 Shift PQ Register

Only one level of Index Field Address Modification is permitted.

The various combination of bits that may be used in the N character are indicated by the following table:

			4	2	6	1
Address Modification	Address Portion	A,B	A	B	A,B	A,B
	by Index Field	None	A	B	A,B	AB
Shift Accum. and PQ Reg. Left		8	@	-	)	9
Shift Accum. and PQ Reg. Right		H	;	+	'	I
Shift Accumulator Left		⊖	M	K	ϴ	J
Shift Accumulator Right		"	U	S	W	/
Shift PQ Register Left		0	4	2	6	1
Shift PQ Register Right		&	D	B	F	AA

A address - 0000 (zeros).
B address - B_0, B_1, B_2 - Ignored
B_3 = Number of shifts.

Direction

As specified by the 2^4 bit of the N character

PRI's

None

Final Register Settings

$A_f = A_i$
$B_f = B_i$

Timing

Staticizing - 35 microseconds
Couple Shift Time - 7 microseconds per shift
Uncouple Shift Time - $7x \frac{N}{2}$

where N = number of shifts
the time must be a multiple of 7 microseconds

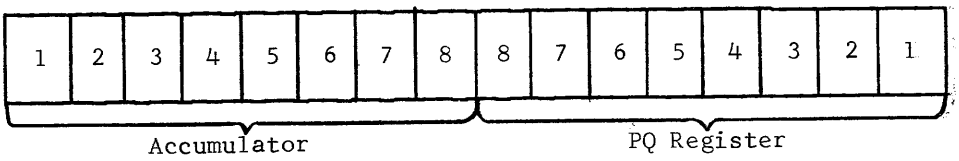
Each address modification - 21 microseconds.

Example

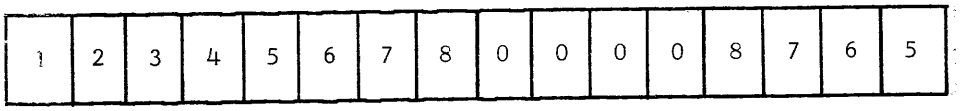
Instruction

Operation	N	A Address	B Address
=	&	0000	0004

Accumulator and PQ Register Before Instruction is Executed



Accumulator and PQ Register After Instruction has been Executed



Timing

35us + 14us

Final Register Settings

$A_f = 0000$
$B_f = 0004$

Tally Instruction

Non-Repeatable

General Description

The TALLY instruction enables the programmer to increment an Index Field by its associated Increment Field. All three, or any combination of the Index Field may be incremented by a single Tally instruction. Except for the ability to increment the Index Field, the Tally instruction operates exactly as in Chapter

Format

Operation X

N

- $2^0 = 1$ Increment AB Index Field by the quantity in the AB Increment Field.
- $= 0$ Ignored
- $2^1 = 1$ Increment B Index Field by the quantity in the B Increment Field.
- $= 0$ Ignored
- $2^2 = 1$ Increment A Index Field by the quantity in the A Increment Field.
- $= 0$ Ignored
- $2^3, 2^4, 2^5 = 0$ Ignored

A Address - HSM address of the diad containing the quantity to be tested
B Address - HSM address of the next instruction to be performed if quantity being tested has not been exhausted.

Final Register Settings

$A_f = A_i$
$B_f = B_i$

Timing

Additional timing associated with the incrementing is 42 microseconds per Index Field.

Example

Instruction

Operation	N	A	B
X	7		

Index Field Before Instruction	Increment Field	Index Field After Instruction
AB = 7561	AB = 1435	AB = 8996
B = 7602	B = 2000	B = 9602
A = 3211	A = 3150	A = 6361

Additional Timing

126 microseconds

Tape | Dept #2 | Sect #3 | 10 Employ # | 6 Tot | 29 Total | read in 1000

RCA 301 COMPUTER PROGRAM RECORD

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	200			;	1	0	0	0	0	0	0	0	0		Rwd BCT	
		1			4	1	1	0	0	0	1	0	2	7		Read Tap	
		2			W	8	2	4	5	0	2	0	3	0		Sense ED/EF Indicator	
		3			X	0	2	1	2	1	2	1	6	0		Tally	
		4			Y	2	1	0	0	0	2	1	2	2		Compare Dept #	
Min 1		5			W	1	2	2	7	0	2	2	1	0		Testing PRIs of Compare Instruction	
	6	206			V	3	1	0	0	2	2	1	2	4		Compare Section #	
		7			W	1	2	2	5	0	2	2	1	0		Test PRIs	
Acc		8			N	7	1	0	2	7	2	1	3	5		Acc	
		9			+	9	2	1	4	4	2	1	3	5			
		210			+	9	2	1	5	3	2	1	3	5			
		1			V	1	0	2	1	9	2	0	1	0		Transfer Back to Rd Tape	
	6	2			0	1	0	0	0	0	0	0	0	0		Set Counter & Work Areas	
		3			0	0	0	0	0	0	0	0	0	0		↓ ① Save Areas for Dept & Sect #	
		4			0	0	0	0	0	0	0	0	0	0		Constants ② 9 Character Fields for Adds	
		5			0	0	0	0	E	0	J	½	½				
		6			M	5	1	0	0	0	2	1	2	2		Save Dept # Sect #	
		7			J	sp	0	6	0	0	0	7	1	9		Clear Print Area	
	6	218			M	2	1	0	0	0	0	6	0	0		Move Dept # → Print Area	
		9			B	1	0	0	0	0	0	6	0	1		Print Dept	
		220			V	1	6	2	1	9	2	0	8	0			
		1			8	7	2	2	4	0	2	2	4	9		Print to Monitor "Out of Sequence"	
		2			•	1	1	1	1	1	1	1	1	1		Halt 1	
		3			V	1	0	2	1	9	2	2	2	0			

28-00-004

PAGE 1 OF

RCA 301 COMPUTER PROGRAM RECORD

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP 0	N 1	A 2	A 3	A 4	A 5	B 6	B 7	B 8	B 9	REFERRED TO BY	REMARKS	BOX NO.
	6	224 0	0	0	S	E	Q	.	-	E	R	R	.	EF			
		5 0	0	0	V	1	2	4	4	9	2	3	8	0		Value of P	
		6 0	0	0	V	1	0	2	1	9	2	0	8	0		Transfer → Acc	
Maj 1		7 0	0	0	V	1	2	4	4	9	2	3	8	0			
		8 0	0	0	J	-	0	6	0	0	0	7	1	9		Clr Print Area	
		9 0	0	0	M	2	2	1	2	2	0	6	0	0		Mov t sAved Dept#	
	6	230 0	0	0	M	9	2	1	3	6	0	6	1	9			
		1 0	0	0	B	0	0	0	0	0	0	6	0	3		Print & Page Change	
		2 0	0	0	M	2	1	0	0	0	2	1	2	2		Save New Dept	
		3 0	0	0	J	0	2	1	3	6	2	1	4	4		Clr Acc Area	
		4 0	0	0	J	-	0	6	0	0	0	7	1	9		Clr Print Area	
		5 0	0	0	M	2	1	0	0	0	0	6	0	0			
	6	236 0	0	0	B	1	0	0	0	0	0	6	0	1			
		7 0	0	0	V	1	0	2	1	9	2	0	8	0		Back → Acc Exit	
		8 0	0	0	J	-	0	6	0	0	0	7	1	9			
		9 0	0	0	M	3	2	1	2	4	0	6	0	2			
		240 0	0	0	M	9	2	1	4	5	0	6	0	9			
		1 0	0	0	B	1	0	0	0	0	0	6	0	1			
	6	2 0	0	0	M	3	1	0	0	2	2	1	2	4			
		3 0	0	0	J	0	2	1	4	5	2	1	5	3			
		4 0	0	0	V	1	0	2	1	9	2	4	4	0			
		5 0	0	0	V	1	1	0	0	0	2	1	5	8		Compare 1st Pos Input w EF	
		6 0	0	0	W	1	2	5	2	0	2	5	2	0			
		7 0	0	0	V	1	2	3	7	9	2	2	7	0		→ Major	

RCA 301 COMPUTER PROGRAM RECORD

TITLE
CODER
REMARKS

DATE
SEGMENT NO.

FROM INST. LOC.	NO. OF INS.	HSM LOCATION	FLOAT A	FLOAT B	OP	N	A				B				REFERRED TO BY	REMARKS	BOX NO.
					0	1	2	3	4	5	6	7	8	9			
	6	248 0	0	0	;	1	0										
		9 0	0	0	8	7	2	1	5	4	2	1	5	7			
		250 0	0	0	.	9	9	9	9	9	9	9	9	9			
		1 0	0	0	V	1	0	2	1	9	2	5	0	0			
		2 0	0	0	V	1	1	0	0	0	2	1	5	7		Comp ED	
		3 0	0	0	W	1	2	5	7	0	2	5	7	0			
	6	4 0	0	0	;	1	0	0	0	0	0	0	0	0			
		5 0	0	0	.	2	2	2	2	2	2	2	2	2			
		6 0	0	0	V	1	0	2	1	9	2	0	0	0		Back → Rd	
		7 0	0	0	.	3	4	4	4	4	5	5	5	5		Illogical Halt	
		8 0	0	0	V	1	0	2	1	9	2	5	7	0			
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
	6	0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													
		0	0	0													

28-00-004

INDEX

Page No.

A

A Address VIII-1
Add XI-2
Address Modification XVIII-1
A Register IX-1
ARIE XXXIV-1, XXXV-4
Arithmetic Unit III-3
Assertion Box X-5, XIX-4

B

B Address VIII-1
Band IV-2, XXIX-1
Band Select Normal XXIX-2
Band Select Record File Mode XXX-1
Base V-1
Batch XXII-1
Batching XXV-1
Binary Addition V-3
Binary Code V-2
Binary Numbering System V-1
Binary Subtraction V-3
Bit VI-2
Block VI-3
Block Read from Record Normal XXIX-3
Block Read from Record Simultaneous XXIX-17
Block Write to Record Normal XXIX-4
Block Write to Record Simultaneous. XXIX-17
Bootstrap XXXVII-1
Box No. Column VIII-1
B Register IX-1
BTC XIII-1
Buffer XXVI-1
Bus Adder IX-1

C

Card Punch Normal ..... XII-2, -4
Card Punch Simultaneous ..... XXVI-5, -6
Card Read Normal ..... XII-1, -3
Card Read Simultaneous XXVI-5
Cell IV-2, XXIX-2
Chaining XXXIII-1
Character VI-2
Closed Subroutine XXXIV-1
Code II-1
Column VI-1
Compatibility XXXVIII-1
Complementing V-3
Computer Elements III-1
Conditional Transfer of Control
  ED/EF Indicator XIII-3
  Interrupt Indicator XXX-3
  Overflow Indicator XXXIV-1
  PRI's XIV-1
  Simultaneous ED/EF Indicator XXVI-7
  Simultaneous Mode XXVI-7

Page No.

Console III-2, XXXV-1
Consolidata XXXVII-3
Control Totals XXVIII-3
Control Unit III-3
Conversion II-1
Conversion
  Binary to Decimal V-3
  Decimal to Binary V-2
Core III-2

D

Da-Span XXXVIII-1
Data Disc File XXXI-1
Data Record File XXIX-1
Data Record File Mode XXX-1
Data Register XI-4, XIV-1
Debugging III-2, XXXVII-1
Detailed Flow Chart ..... X-1, -4, -9
Diad VII-1
Direct Addressing XXXII-3
Disc IV-5, XXIX-1
Divide XXXIV-2
Documentation of a Program XXVIII-16
Dual Recording IV-2

E

EAM Cards III-1, IV-4
EB XXIX-3
ED VI-3
ED Routines XXVIII-10
Editing XXVII-1
EF VI-3
EF Routines XXVIII-12
End Around Condition XI-3
EOR X-5, XXVIII-12
ETW XIII-1
ETW Routines XXVIII-8
Exclusive OR XX-2

F

FAA XXII-4
Field VI-1
Fixed Format VI-1
Fixed-Variable Format VI-1
Floatable XXXIV-1, XXXVI-1
Floating XXVII-5
Flow Charting X-1
FOR XXX-1
From Instruction Location VIII-1
Functional Flow Chart ..... X-4, -8

Page No.

G

Gap..... III-4, IV-2

H

Halt..... XIII-3
Halt Routines..... XXVIII-16
Hash Count..... XXVIII-3
Hi-Data Clusters..... IV-4
High Speed Memory..... VII-1
Hole Count..... IV-1
Housekeeping..... XXVIII-6
HSM Location Column..... VIII-1

I

Indirect Addressing..... XXIII-1
Input..... IV-1
Input-Output Sense.....XIII-1, XXVI-7, XXX-2
Instruction Format..... VIII-1
Internal Speed..... III-3
Interrogating Typewriter..... IV-3
ISS..... XXI-26
Item..... VI-2

L

Limit Checks..... XXVIII-4
Locate Symbol Left..... XVIII-5
Locate Symbol Right..... XXI-7
Logical AND..... XX-1
Logical OR..... XX-3
Loop..... XIII-1

M

Magnetic Core.....III-3, IV-5
Magnetic Disc..... III-3, IV-2, -3
Magnetic Drum..... III-3
Magnetic Tape..... III-2
  Hi-Data Clusters..... IV-2
  33KC..... IV-2
  66KC..... IV-2
Memory..... III-3
Memory Addressing Register..... IX-1
Memory Cycle..... IX-2
Memory Register..... IX-1
Merging..... XXXIX-1
Microsecond..... IX-2, XXIV-1
Millisecond..... XXIV-1
Mode..... XXVI-1
Monitor Printer..... IV-4
M Register..... XXVI-2
Multiply..... XXXIV-2

Page No.

N

N Character..... VIII-1
N Register..... IX-1
N Table..... XI-3

O

Operation Code..... VIII-1
OP Register..... IX-1
Output..... IV-3
Overflow..... XXIV-1

P

Paper Tape.....III-2, VI-4
Parity..... VI-2
Percentage of Occurrence..... VI-5
PET..... XIII-1
PRI's..... XI-4
Print and Paper Advance Normal.... XVI-3
Print and Paper Advance
  Simultaneous..... XXVI-6
Printer..... III-2, XVI-1
PRN..... XI-4
Production Programs..... XXXVII-4
Program..... III-5
Proposal..... II-1
P Register..... IX-1
PRP..... XI-4
PRZ..... XI-4
Punched Card..... III-1
Purge Date..... XXVIII-2

R

Random Access..... XXIX-9
Randomizing..... XXXIII-1
Read-Write Head..... XI-1
Record..... VI-3
Record File Mode Read..... XXX-1
Record File Mode Write..... XXX-2
Regeneration..... IX-1
Register..... III-6, IX-1
Remarks Column..... VIII-1
Repeat..... XXII-1
Repeat Logic..... XXII-5
Rewind to BTC..... XIII-1
Row..... VI-2
Run..... II-1
Run Tape..... XXXVII-3

S

Satellite.....XXXVIII-1

INDEX (Cont'd)

	Page No.
Sector	XXXI-1
Sector Compare Left	XIV-1
Sector Read from Disc Normal	XXXI-2
Sector Read from Disc Simultaneous	XXXI-3
Sector Write Disc Normal	XXXI-3
Sector Write Disc Simultaneous	XXXI-3
Segment	XXXVI-7
Sequence Check	XV-4
Service Routines	XXXVII-1
Simultaneous Mode	XXVI-1
Simultaneous Program Control Unit.	XXVI-1
SOR Register	XXVI-2
Sorting	XXXIX-1
S Register	XXVI-2
STA	XVIII-6
Standard HSM Locations	XXXVI-8
Staticizing	IX-2
Storage	IV-5
Store Register	
A	XXXII-3
B	XXI-4
P	XI-5
S	XXVI-7
U	XXX-2
STP	XI-5
Sub-item	VI-2
Subtract	XV-1
Switch	XIX-2
Systems Check	II-2, XXVIII-2
Systems Flow Chart	X-1, -2, -7
T	
Table Look-Up	XI-3, XXXII-1
Tally	XVII-1
Tape Adapters	IV-2
Tape Tables	XXVIII-2, XXXVI-13, -16, -18
Tape Loop	XVI-3
Tape Read Forward Normal	XI-1
Tape Read Forward Simultaneous	XXVI-4
Tape Swap	XXVIII-8
Tape Write Normal	XI-4
Tape Write Simultaneous	XXVI-4
Template	X-1
Time Sharing	XXVI-3
Timing	XXIV-1
Timing Formula	XXIV-1
Track Select	XXXI-2
Trailer Message or Record	XXI-26, XXVIII-3
Transfer Data by Symbol Left	XXI-3
Transfer Data by Symbol Right	XXI-11
Transfer Data Left	XV-3
Transfer Data Right	XV-2
Transfer Symbol to Fill	XV-1
Translate by Table	XXXII-3, XXXVIII-1
T Register	XXVI-2
U	
U Register	XXX-1
Up to Speed Time	XXIV-3
V	
Variability	VI-3
Variable Connector	XIX-2
Variable Format	VI-3
Volume	XXIV-1
V Register	XXX-1
W	
Weighted Average	VI-4
Work Area	XV-2
Z	
Zone	IV-3

THE MOST TRUSTED NAME IN ELECTRONICS

Radio Corporation of America