Principles of Programming

Section 8: Magnetic Tape Operations

IBM Personal Study Program

Section 8: Magnetic Tape Operations

Magnetic tape provides compact storage for much larger amounts of information than can be contained in core storage and allows for much faster reading and writing than with punched cards. Magnetic tape storage is available and heavily used for virtually all large computers, and can be installed on the IBM 1401. When the 1401 is used as a medium-size computer by itself, magnetic tape provides large-capacity storage for files and input data. When the system is used as an auxiliary machine with a larger computer, magnetic tape is used as a communication device between the larger machine and punch card input or printed output. The following discussion of the physical characteristics of magnetic tape is applicable to all IBM machines.

8.1 Physical Characteristics of Magnetic Tapes

Magnetic tape is wound on a 10½-inch-diameter reel. The tape itself is one-half inch wide and 2,400 feet long. It is coated with a magnetic oxide material on which information can be recorded in the form of magnetized areas. One reel of tape can contain as many as 16,000,000 characters; the actual number depends on how the information on the tape is organized.

Each character on the tape is recorded in a seven-bit code very similar to that used within the computer. Characters are recorded in groups called *blocks*. A block may contain any number of characters. Blocks are separated from each other by about three-quarters of an inch of blank unrecorded tape called an interrecord gap. (The words *block* and *record* are occasionally used as synonyms; in this section we shall attempt to maintain a distinction between a physical block on the tape and the one or more problem records that may be contained in the block. Certain usages are so firmly entrenched, however, that we cannot be completely consistent on this point.)

Each tape character is composed of an *even* number of ones; this is contrasted with the representation within the computer in which each character has an *odd* number of ones. This difference is necessary to maintain compatibility with tapes from some of the large IBM computers.

The tape codes for the characters used in the 1401 are shown in Figure 1. It is necessary to qualify this statement as applying specifically to the 1401, because there are a number of special control characters shown at the right of Figure 1 that do not apply to all IBM machines. The coding of the standard characters is the same in all, however.

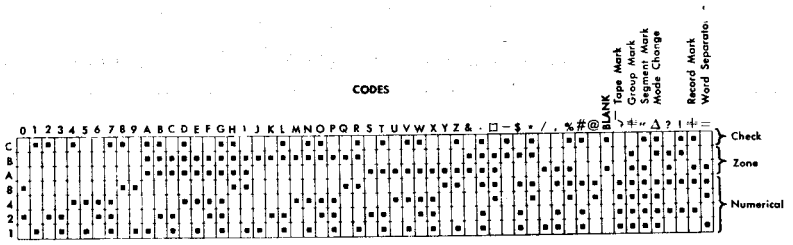

Figure 1. IBM 1401 tape character codes

In Figure 1, the seven-bit vertical groupings are the characters. We speak of the horizontal groupings, containing one bit from each character in the block, as being horizontal rows or, sometimes, channels. Rows are named in the same way as the seven bits of the character within the computer, that is, CBA 8421.

The reading and writing of information with magnetic tape is subject to a considerable amount of built-in checking in IBM tape systems. First, there is parity checking of each character. As each character is written onto tape, the parity bit is computed so as to make the total number of ones in the character representation even. When the tape is later read, the number of ones in each character read from tape is checked to make sure that it is even. If it is even, we have a certain amount of assurance that the character was written and read correctly. To provide further confidence in the accuracy of the tape operations, a parity check is made on the number of ones in each horizontal row of each block; as the block is written a count is maintained of the number of ones in each row of the block. After the last character has been written, another character, called the horizontal check character, is recorded. This contains in each bit position either a one or a zero, whichever is necessary to make the total number of ones in that row even. This horizontal check character is provided automatically by the computer circuitry and need be given no attention by the programmer. When the tape block is read, a count is made of the number of ones in each channel of the entire block, including the check character. If the number of ones in each channel is found to be even, we have further assurance that the tape operation is correct. The check character does not enter storage.

A third form of checking, called relative sensitivity level sensing, provides further assurance of accuracy of tape reading and writing. The engineering details are not of concern to us here.

These checking features are designed to detect errors of two rather different types. One type is actual malfunction of the electronic and mechanical equipment. In modern computers such troubles are not frequent. The other source of tape errors is the tape itself. The quality standards imposed on the manufacture of the tape by the computer must be extremely rigid, since the slightest imperfection can cause a character to be recorded incorrectly. Furthermore, dust particles can cause a weak signal to be recorded, and simple mechanical wear of the oxide coating can cause the quality of the recorded signal to deteriorate. Great pains are taken to minimize troubles caused by the tape itself: the manufacture of the tape is subject to stringent inspection, great care is taken in a well run installation to avoid dust and dirt on the tape, and the tape units are designed to cause as little wear of the oxide as possible.

When a section of tape becomes damaged or worn, it is common practice to cut out the bad part, leaving two shorter tapes. This is no inconvenience, since there is frequent need for tapes shorter than the maximum reel length. Even a full length tape is shortened in use: since the beginning of the tape gets the most wear, by being handled in mounting the reel, the first 20 feet or so is cut off from time to time. (This of course is done at a point in the usage of the tape when no valid information is recorded on the tape!)

The various automatic checks are made only when the tape is read. If anything should turn out to be wrong with the information recorded on tape, no indication of the fact will appear until the tape is read. If the detection of the error were postponed until the tape is used in the next processing cycle, it would be moderately inconvenient to reconstruct the correct information. It is much more desirable to know about the difficulty immediately after the tape is written, while the information is still in core storage ready to be rewritten.

This immediate checking is provided in the tapes used on the 1401 by the *two-gap head*. The term "head" is used to describe the assembly of coils and magnetic pole pieces that reads or writes information on a tape. In many computers, a single head is used both for reading and for writing. In the IBM 729 and 7330 tape units, there is one head for reading and a separate one for writing. The writing head is positioned in front of the reading head, in the sense of the direction of tape motion. Thus, when a character is written, it is automatically read a very short time later by the read head to determine that the information has been recorded readably on tape and that parity counts are correct. If there should happen to be something wrong with the tape or with the tape unit, then it is a fairly simple matter to correct the difficulty since the information is still available in core storage.

Magnetic tape is written and read in a *tape unit*, of which three types may be used with the 1401: the 7330, the 729 Model II, and the 729 Model IV. The number of tape units attached to a computer is

variable at the discretion of the user; in this book we assume a machine with four.

The difference between the three tape units that can be used with the 1401 is in the speed at which the tape moves. In all three, there may be either 200 or 556 characters per inch of tape, depending on the setting of a switch on the tape unit. In the 729 IV, the tape moves at 112.5 inches per second, making the transfer rate either 22,500 or 62,500 characters per second. The corresponding figures for the 729 II are 75 inches per second and either 15,000 or 41,667 characters per second, and for the 7330 they are 36 inches per second and either 7,200 or 20,016 characters per second. These performance characteristics are summarized in Figure 2.

	7330	729 II	729 IV
Density, Characters per Inch	200 or 556	200 or 556	200 or 556
Tape Speed, Inches per Second	36	75	112.5
Interrecord Gap, Inches	3/4	3/4	3/4
Start/Stop Time, Read/Write Operation, Milliseconds	Varies	10.8	7.3
Character Rate, Characters per Second	7,200 or 20,016	15,000 or 41,667	22,500 or 62,500
Rewind Time, Full Reel, Minutes	13.3 or 2.2	1.2	0.9

Figure 2. Summary of magnetic tape characteristics

In figuring the time required to read or write a tape block, we must consider not only the time to transfer the characters in the blocks, but also the time required to accelerate the tape to full speed before reading or writing. This start/stop time turns out to be about the same as the time required to read past the interrecord gap at full speed.

The total time required to read a tape depends on the fraction of the time the tape is kept in motion. The average character transfer rate depends not only on this figure but on the size of the blocks, bearing in mind that there is about three-quarters of an inch of blank tape between one block and the next. On the 729 tapes at low density, for instance, three-quarters of an inch of tape will hold 150 characters. If the blocks are smaller than this size, we see that less than half of the tape contains information. Therefore, even if the tape were kept moving all the time, the average character transfer rate would be only half the rate at which they are transferred during the reading or writing of a block. We shall see later that this consideration has a significant influence on the programming techniques used with magnetic tapes.

It is necessary to be able to detect the beginning and the ending of the tape, both for design reasons in the magnetic tape unit itself and because of programming considerations. For this purpose, *reflective spots*, also referred to as *photo-sensing markers*, are placed on the tape to enable the tape unit to sense where reading and writing are to begin and to stop. The reflective spot at the front end of the tape is called the *load point*. Appropriate button pushing on the tape unit when the tape is loaded will cause the tape to be positioned just beyond the load point. The reflective spot near the end of the tape is employed when writing, to indicate that the physical end of the tape is about to be reached and that no further information should be written. Detection of the end-of-reel reflective spot during writing turns on an indicator in the computer.

When the end-of-reel spot is detected during a writing operation, we ordinarily write a final block on the tape consisting simply of a *tape mark*. This is a special character that will later be detectable upon reading, to signal the end of the tape by turning on the same end-of-reel indicator that is turned on by the reflective spot when writing. The tape-mark technique of denoting the end of the tape is used for two reasons. First, the tape unit does not turn on the end-of-reel indicator when the reflective spot is detected on reading; we must therefore have some other technique for detection of this condition. The second reason is that we sometimes wish to put on tape an indication that no more information follows, even though the end of the physical tape has not been reached. The tape mark provides this capability.

Since the complete collection of valid data blocks is frequently called a file, the tape mark is also sometimes called an end-of-file mark. We prefer here, however, to reserve the word file for the meaning in which it was used in Section 1, in order to always make a clear distinction between a tape *reel*, and a *file* of information, which may consist of only a few blocks, a complete tape, or many tapes.

Each tape reel is provided with a removable plastic ring on the back side, that is, the side nearest the tape unit. This is called the *file protection ring*. It is not possible to write on a tape unless this ring is inserted in the tape reel. This feature is provided as a precaution against accidental destruction of permanent master files. The usual procedure is to remove the file protection rings from such master tapes after they have been initially written and then require the authorization of some responsible member of the organization before the file protection ring can be inserted in any tape reel.

Most people have at least slight difficulty in remembering whether the tape is protected by inserting or removing the file protection ring. We suggest the mnemonic phrase "no-ring-no-write."

Review Questions

1. *Outline the built-in error checking in IBM magnetic tape systems.*
2. *What is the maximum number of characters that can be written on a 2,400-foot reel of tape at low density? At high density? Why would a reel never contain so many characters?*
3. *What is the purpose of the file protection ring?*

8.2 Magnetic Tape Instructions

The operation of magnetic tapes in a computer system is controlled by the execution of suitable instructions, as is everything else. We shall look into the tape instructions in the 1401 briefly, in order to get a general idea of what the basic machine instructions are and what they do. We shall see in the following subsection, however, that tape operations are seldom actually set up this way in normal programming; instead *macro-instructions* are used, which greatly simplify tape programming. For now, therefore, we wish merely to survey the actual machine instructions in order to understand better what the macro-instructions do.

Tapes are most commonly written and read in the 1401 with two instructions called Write Tape and Read Tape. The operation code for both instructions is M, which is also the operation code for Move; we therefore use the mnemonic operations code MCW. However, both instructions require a d-character, and the net result is that we have two entirely new instructions that are essentially unrelated to a Move. For reading tape the d-character must be R and for writing it must be W.

To write a block on tape in the 1401, we must specify three items of information to the computer:

1. Which of the tape units is to receive the block? This is specified by the A-address of the instruction, which must be of the form %Ux, where x is the number of one of the tape units attached to the system. The percent sign and the U are required by the design of the system to specify magnetic tape. The numbers of the tape units may be set by a dial on each tape unit. An installation with five magnetic tapes, for instance, would most likely establish the convention that the tape units are dialed so as to run through the numbers 1 to 5.

2. Where in storage is the first character of the block to be written? This is answered by the B-address of the tape instruction, which specifies in regular three-character form the high-order character position of the first character. Note carefully that characters in a block are read from successively higher-numbered locations.

3. What is the length of the block to be written? This question is answered by placing in core storage, after the last character of the block, a special symbol called a *group mark* with a word mark. The group mark consists of all ones in the zone and numerical portions of the character. When a group mark with word mark is detected in core storage during writing, the writing operation stops, without having written the group mark with word mark on tape.

Write Tape

	Mnemonic	Op Code	A-add	B-add	d-character
FORMAT	MCW	M	%Ux	xxx	W

FUNCTION — The tape unit designated in the A-address is started. The d-character specifies a tape write operation. The data from core storage is written on the tape record. The B-address specifies the high-order position of the record in storage. A group mark with a word mark in core storage stops the operation. The group mark with a word mark causes an interrecord gap to be created.

WORD MARKS — Not affected.

TIMING — $T = .0115\ (L_I + 1)$ ms $+ T_M$

The actions on reading a tape are very similar, except that the operation is ended in a slightly different way. The operation code is M, the A-address is %Ux where x specifies a tape number, the B-address is the address of the first position into which a character should be read from the tape block, and the d-character is R. The operation is stopped by the occurrence of either of two things. If the interrecord gap is sensed in reading the tape, then a group mark is inserted in core storage following the last character of the block and the operation is stopped. If, on the other hand, a group mark with word mark is sensed in storage before reaching the end of the tape block, then the transmission of characters is stopped immediately, although the tape does move past any remaining characters on the tape until it reaches an interrecord gap.

Five somewhat related instructions for controlling the action of the tape unit without transmitting information complete the repertoire of tape instructions. All of the five instructions have the actual operation code U and the mnemonic CU, for Control Unit. They are distinguished by their d-characters.

Read Tape

FORMAT	Mnemonic	Op Code	A-add	B-add	d-character
	MCW	M	%Ux	xxx	R

FUNCTION — The tape unit specified in the A-address is started. The d-character specifies a tape read operation. The B-address specifies the high-order position of the tape read-in area of storage. The machine begins to read magnetic tape, and continues to read until either an interrecord gap in the tape record or a group mark with a word mark in core storage is sensed. The interrecord gap indicates the end of the tape record, and a group mark (code CBA 8421) is inserted in 1401 core storage at this point.

WORD MARKS — Not affected.

TIMING — $T = .0115\ (L_I + 1)$ ms $+ T_M$
Time varies for type of tape unit and tape density used (see Figure 2).

The Back Space Tape instruction (d-character: B) causes the tape unit specified in the A-address to move backwards over one tape block. The first interrecord gap encountered in the backward direction stops the operation.

Backspace Tape

FORMAT	Mnemonic	Op Code	A-address	d-character
	CU	U	%Ux	B

FUNCTION — The tape unit specified in the A-address backspaces over one tape record. The first interrecord gap encountered stops the operation.

WORD MARKS — Not affected.

TIMING — $T = .0115\ (L_I + 1)$ ms $+ T_M$

The Write Tape Mark (d-character: M) causes a tape mark to be recorded immediately following the last block on tape, to indicate whatever the programmer wants it to indicate. Most commonly it denotes the end of valid information on the tape, and/or that the physical end of the tape is about to be reached. It is, therefore, sometimes called the end-of-reel indicator, but it can be used for several other purposes. The tape mark has zone bits of 00 and numerical bits of 1111. When a tape mark is later encountered in reading the tape, the end-of-reel indicator in the computer is turned on; it may be tested with a Branch If Indicator On instruction. It should be carefully noted that the Write Tape Mark instruction creates a separate block preceded by an interrecord gap. Thus, when the tape is read, the tape mark is *not* detected by reading the last block before the tape mark. It is detected only by reading the block that contains the tape mark. Therefore, after every Read Tape instruction there should ordinarily be a Branch If Indicator On instruction to determine whether data was read or the tape mark encountered. It should also be noted carefully that there is only the one indicator for this purpose in the entire system. Therefore, the Branch If Indicator On instruction will always test whether a tape mark was detected on the tape most recently read. It is not possible to read from two tape units and then use a Branch If Indicator On instruction to determine whether there was a tape mark on the first one. The indicator is turned off by selecting a new tape unit or by testing it.

Write Tape Mark

FORMAT	Mnemonic	Op Code	A-address	d-character
	CU	U	%Ux	M

FUNCTION — This instruction causes a special character (8421) to be recorded immediately following the last record on tape, to indicate an end-of-reel condition. When the tape mark is read back from a tape, the end-of-reel indicator is turned on. This signals the 1401 program that the end of the utilized tape has been reached.

WORD MARKS — Not affected.

TIMING — $T = .0115\ (L_I + 1)$ ms $+ T_M$

The Skip and Blank Tape instruction (d-character: E) is used to erase about 3¾ inches of tape. It is used when repeated attempts to write on an area of tape have shown that a readable tape record cannot be written there. By erasing the bad area of tape, we get the effect of an unusually long interrecord gap. The idea is that the bad section of tape may be limited to one small area and that tape farther along may be usable.

Skip and Blank Tape

FORMAT	Mnemonic	Op Code	A-address	d-character
	CU	U	%Ux	E

FUNCTION — The tape unit designated by the A-address spaces forward and erases 3¾ inches of tape. The actual skip occurs when the next Write Tape instruction is executed.

WORD MARKS Not affected.

TIMING $T = .0115 (L_I + 1)$ ms
Processing can continue immediately after this operation. However, 47 ms for IBM 729 II and 27 ms for IBM 729 IV must be added to the next Write Tape instruction time.

The Rewind Tape instruction (d-character: R) causes the selected tape unit to rewind its tape to the load point. If there is less than about 400 feet of tape to be rewound, the tape simply moves backwards past the heads. In the 729 units, if there is more than about 400 feet, the heads are raised, the tape is lifted out of the vacuum columns, and the rewinding is done at a much higher speed. The total time to rewind the tape does not exceed the figure shown in Figure 2 for each tape unit, regardless of how much tape there is on the takeup reel: the more tape there is, the faster the high speed rewind goes. Processing may continue during the rewinding.

Rewind Tape

FORMAT	Mnemonic	Op Code	A-address	d-character
	CU	U	%Ux	R

FUNCTION The tape unit designated by the A-address is rewound to its load point.

WORD MARKS Not affected.

TIMING $T = .0115 (L_I + 1)$ ms + 10 ms
Rewind time is 1.2 minutes per 2,400-foot reel for the IBM 729 II, .9 minutes for the IBM 729 IV, and 13.3 minutes for the 7330, but it is not calculated with program time. Processing can continue approximately 10 ms after this instruction is interpreted.

The Rewind Tape and Unload (d-character: U) performs the same functions as the Rewind, but in addition makes it impossible for the computer to use that tape unit until a switch on the tape unit is manually depressed. This is ordinarily used when a tape has been written that should be dismounted from the unit and a new tape mounted so that processing can continue with a new reel. Without this feature, there would be the constant danger that a recently written tape could be destroyed by writing new information on it, in the mistaken hope that the operator would have changed it by the time the new information was to be written.

Rewind Tape and Unload

FORMAT	Mnemonic	Op Code	A-address	d-character
	CU	U	%Ux	U

FUNCTION This instruction causes the tape unit specified in the A-address to rewind its tape. At the end of the rewind, the tape is out of the vacuum columns, and the reading mechanism is disengaged. The unit is effectively disconnected from the system, and is not available again until the operator restores it to a ready status.

WORD MARKS Not affected.

TIMING $T = .0115 (L_I + 1)$ ms + 10 ms

Rewind time is 1.2 minutes per 2,400-foot reel for the IBM 729 II, .9 minutes for the IBM 729 IV, and 2.2 minutes for the 7330, but it is not calculated with program time. Processing can continue approximately 10 ms after this instruction is interpreted.

For an elementary example of the use of some of these instructions, consider the first part of a job that involves putting the information from a deck of cards onto tape. The tape, once written, might be used in later operations in the 1401 or it might be used as input to a larger computer. In order to illustrate the basic operations without unduly complicating the logic, we shall consider an extremely simplified and somewhat unrealistic approach. Each card will simply be written onto the tape on tape unit 1, exactly as it is read into storage, with one card in each tape block.

The block diagram of this program is shown in Figure 3, and the program in Figure 4. We load a group mark with word mark as a constant into position 81, which is the next character position after the read area. At the beginning of the program we read a card and immediately write the information onto tape 1. Next, a test is made to see whether the information written was readable when it passed the read head after writing. If it was, we test to determine whether the card just read was the last one. If it was not the last card, we branch back to read another card. If the tape was not readable, we write on the printer a short message to the operator that we ran into difficulty, and halt. In actual practice, using the Input/Output Control System, considerably more pains than this will be taken to attempt to write the tape correctly and skip over the bad tape if it cannot be written in a few attempts. If the last-card test showed that the end of the deck had been reached, we write a tape mark, rewind the tape, print a message that the job is ended, and halt. This job-ended message is not too crucial here since the operator would readily enough see that all the cards had

been read, but it is good practice to be fairly free with such messages (or other signals), so that the operator need never be in doubt as to the status of the program. When appropriate, we also like to be able to write out messages specifying what action to take next.

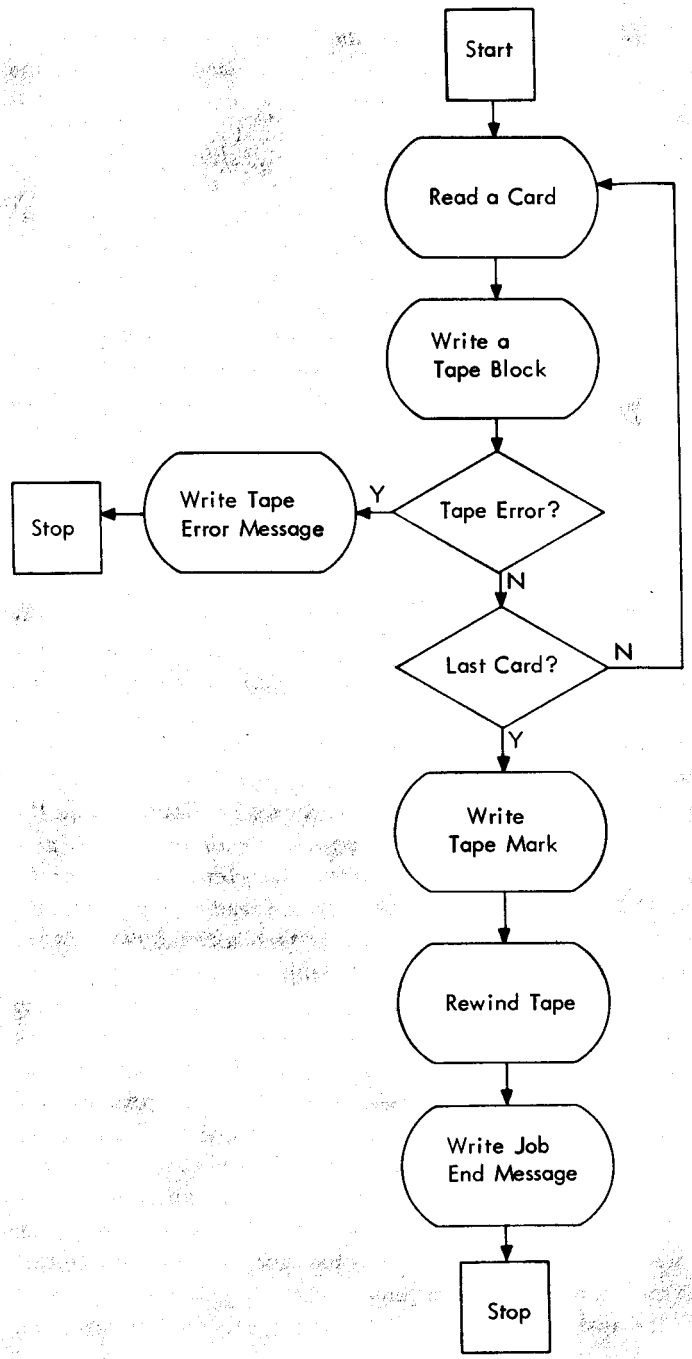

Figure 3. Block diagram of a program to write the information from a deck of cards onto magnetic tape

FORM X24-1152-2
PRINTED IN U.S.A.

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

Program _____ Date _____ Page No. 1 2 of _____

Programmed by _____ Identification 76 80

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	(A) OPERAND ± 23	(A) OPERAND CHAR. ADJ.	(B) OPERAND ADDRESS	(B) OPERAND ± 34	(B) OPERAND CHAR. ADJ.	d	COMMENTS
3 5 6 7 8		14 15	16 17	20	23	27 28	34	34	38	39	40 55
010		START	R								READ CARD
020			MCW	%U1							WRITE ON TAPE
030			B	ERRHLT							TAPE ERROR? Q
040			B	FINISH							NO, LAST CARD Q
050			B	START			NO				
060		FINISHCU		%U1							MWRITE TAPE MARK
070		CU		%U1							REWIND
080			LCA	MSSG1			0212				WRITE JOB
090			W		-003						END MESSAGE
100			H	*							
110		ERRHLT	LCA	MSSG2			0219				WRITE ERROR
120			W								HALT MESSAGE
130			H	*	-003						
140	01		DCW	008I	#						GROUP MARK
150	12	MSSG1	DCW	*	J08		FINISHED				
160	19	MSSG2	DCW	*	BAD		TAPE J08 HALTED				
170			END	START							
180											
190											
200											

Figure 4. SPS program to write the information from a deck of cards onto magnetic tape

This example is obviously vastly oversimplified. We shall see in the next subsection that the same general task can be done much more simply using the Input/Output Control System; in spite of the simplification of the programming, the job will be done in a much more thorough fashion.

Review Questions

1. *How does the core storage addressing of information to be read from tape or written on tape differ from other data addressing on the 1401?*
2. *What are the two ways that tape reading may be stopped in the 1401?*
3. *When is a tape mark detected?*

8.3 Tape Programming with Autocoder and IOCS

The effective use of magnetic tapes in a data processing system requires consideration of many factors. If every programmer had to take all these factors into account himself, and then write the detailed program properly to handle them, a great deal of time would be wasted; the same problems face anyone who ever writes a tape program. Fortunately, this is not the case; a standard tape programming system is available. This package, called the 1401 Input/Output Control System, or IOCS, handles all of the normal input and output programming considerations with a minimum of programmer effort.

IOCS is one of the major parts of an advanced coding language similar to SPS but considerably more powerful, called Autocoder. The basic ideas of Autocoder are generally the same as discussed in Section 4: symbolic addresses may be used in place of numerical addresses; mnemonic operation codes replace actual; the symbolic source program is translated into an actual object program by an assembly process. The major differences between SPS and Autocoder are these:

1. A *free-form* coding sheet is used, which means here that there are no fixed fields for the operands. Instead, the programmer uses as much space as required for each operand, and separates the operands by commas if there is more than one.

2. Augmented mnemonic operation codes are used. This relieves the programmer of writing the d-character in most instructions, and makes it unnecessary to write the %U in the A-address of a tape instruction, for instance. A complete list of Autocoder mnemonics appears in Section 12.

3. It is possible to use *literals*—that is, instead of writing the *address* of a constant in an instruction, we may write the constant *itself*. The Autocoder processor assigns a location to the constant, and fills in the assigned address in the instruction.

4. *Macro-instructions* are provided. We shall be concerned here only with the IOCS macros, with which the programmer can specify in a very condensed form the tape operations that he wants to perform; the processor translates these into routines of dozens or hundreds of instructions. In short, *one* instruction in the source program is translated into *many* actual machine instructions; this characteristic makes Autocoder a *compiler* rather than an assembler as SPS is.

Autocoder also provides the programmer with the flexibility of making up macros for the purposes of his program. Thus, some frequently used group of instructions can be called for in writing the source program simply by writing the macro, which is a much simpler matter than writing all the instructions themselves. Setting up a new macro is not appreciably more difficult than writing the detailed instructions to do the processing once; after that, all similar operations can be handled merely by writing the macro.

Autocoder represents a significant advance in programming sophistication over SPS. We shall not attempt to give a complete description of all the features of the language. A summary of the IOCS macros follows, and some of the other features will be illustrated in the programs.

The IOCS macros are of three types. The first, DIOCS (for Define IOCS), is used by the programmer to define the machine configuration on which the object program will run, along with certain general information about the files and their processing. The second type, DTF (for Define Tape File), is used to describe in detail each of the files used in a problem. The DIOCS and DTF are called *declarative* macros: they provide information to the Autocoder processor, but do not result in any action in the object program.

The next four macros, on the other hand, do cause action in the object program, and are therefore called *imperative* macros. The OPEN, CLOSE, GET, and PUT macros lead to the creation of object program instructions that actually carry out the desired processing of file information.

In order to see how these macros operate and to illustrate their use, we must investigate some of the considerations that affect tape programming.

Record Blocking. It is usually quite inefficient of tape space, and therefore of time, to make tape blocks as short as they would be with normal tape records. Thus in the program of Figure 4, if we were using a 729 tape at high density, the card characters would take up only about 20% of the total tape space. All the rest would be interrecord gaps. Therefore, it is common practice to write a number of what might be called "problem records" or "logical records" in one tape block. This is called *tape blocking;* the number of logical records

in one tape block is called the *blocking factor*. In the special case where each block consists of one logical record, as in the previous example, we speak of unblocked records, or of a blocking factor of 1.

Tape blocking is a virtual necessity if the computer system is to be used effectively, but it does create certain programming problems. When a tape block is read, several problem records are brought into core storage. The processing instructions must be arranged to pick up the records in succession from the storage area into which the block was read. This *deblocking* can be handled by moving the records from the block input area to a *working storage* area as needed, or by using an index register to select the records in succession from the block input area.

In writing, the records must be assembled, or blocked, in the output area and then written when a complete block has been assembled. Once again, this can be done either with a working storage area or with an index register.

Variable vs. Fixed Length Records. It fairly often happens that the amount of information in a tape record varies greatly from one record to another, typically because a small fraction of the file items require additional data not needed for the bulk of the file. For instance, a typical master record in an electric utility billing system contains a minimum of 200 characters, an average of 300, and a maximum of 600. Most customers have only one meter, but a certain fraction have two; bills are generally sent to the same address as the meter address, but not always; if a customer has two meters, they are generally at the same address—but not always. It is clear that if the master records were set up to contain the maximum information that could ever be necessary—the simplest approach—a great deal of tape space would be wasted in the large majority of the records, which require only half as much information. The same sort of thing happens in many other types of applications.

A better choice is to let the length of the records vary according to the amount of information that must be recorded. Now, some means must be provided to indicate the length of each record; this is quite easily handled by placing in each record a number that specifies the total number of characters in the record.

Block Counts. It is often very useful to know precisely how many blocks there are on a tape. This block count can readily be generated as the tape is written, and the block count recorded as part of a separate block at the end of the tape. (This is the *trailer label* described below.)

Record Counts. It is also frequently useful to know how many records there are on the tape. This is usually not just the product of the number of blocks and the blocking factor, because the number of records in the file may not be a multiple of the blocking factor. This count can also be generated by the program and written in the trailer block.

Control Totals. To provide a check on the accuracy of programming and of machine operation, it is valuable to have in the trailer block the sum of some field in each record on the tape. This might be, for instance, the sum of the dollar amounts in all records. Such a control total can be accumulated as the tape is written; when the same tape is later processed, the control total can be developed again and compared with the control total in the trailer. This gives a fairly strong assurance that all records were processed. (Failure to process a record or two under unusual circumstances is a surprisingly common programming error.)

The field summed actually need not have any meaning as a number by itself, as a dollar total ordinarily does. Forming the sum of all the account numbers, or all the city codes, or almost anything else, gives just about the same degree of checking. When a control total has no meaning in itself, it is called a *hash total.*

Tape Labels. Many computer installations have hundreds or even thousands of reels of tape, making it crucial that there be no mixups in tape identification. Running a major job with the wrong input tapes, or writing over a tape that should not have been reused, can be a minor catastrophe. Unfortunately, such mixups can happen all too easily, making it most desirable to have some sort of identification recorded on the tape itself, in addition to the paper label attached to the reel. This is the function of the *header label.* A normal header label contains the file name, a reel number, a reel sequence number within the file if there is more than one reel to the file, the date of creation of the tape, and the retention cycle. These last two items serve important purposes: they prevent a tape from being reused when the information on it is still needed, and they prevent the information on a tape from being used after it is outdated.

A program to use labeled tapes must obviously provide for the creation of labels on new output tapes, for checking the labels of input tapes to determine that the right data is being used, and for checking labels of output tapes to be sure that valid data is not being destroyed. Label creation and label checking are two of the many functions provided by IOCS, with next to no effort required of the programmer.

Restart. It occasionally happens that a job must be stopped when it is partially completed. This can happen as a result of machine trouble, operator error, or because a higher-priority job must be run. When the job is restarted, a number of problems arise: How far along was the job when it was stopped? What was in core storage? Which tapes were mounted on the tape units? Where was each tape positioned? The general idea of a restart procedure is to provide the answers to these questions at a number of points through the running of the job; these are called *checkpoints.* Anytime the job must be stopped, it is necessary only to return to the most recent checkpoint and start from there, rather than going back to the beginning of the job.

Routines must be provided that will take care of all problems of establishing checkpoints and restarting a job that was stopped before it was completed. At the completion of processing of every tape, and at any other point the programmer wishes to specify, the entire contents of core storage must be dumped onto a separate tape; this establishes where the program was at the time of the checkpoint. This dump must also contain the identification of every tape then mounted, and block counters that tell the position of each tape. If it is necessary to restart, the special routines can be called into operation. They will print out instructions to the operator as to what tapes to mount; they will position each tape at the point where it was at the time of the selected checkpoint, and call back into storage the exact storage contents at the time of the checkpoint. The program can now continue just as if nothing had happened.

Let us now return to the consideration of the IOCS package itself. As noted above, the programmer writes macro-instructions in his source program. The first of these is the Define Tape File (DTF) macro, of which there must be one for each tape file. We shall not be concerned here with the details of writing the DTF macros, which, although presenting no conceptual difficulties, would take too much space to describe completely. Therefore, the examples below will be somewhat "schematic," in that we shall summarize the information that would have to be in the DTF macro, without actually displaying the form in which it would be written.

The DTF macros *define* the files; the following macros call for action upon them.

OPEN. Before the processing of a file can be started, the file must be initialized by the use of the macro-instruction OPEN. This macro may have any symbol in the label field, has the code OPEN in the operation code field, and has the name of the file in the operand field. The name must be the same as the name used in the DTF macro.

The OPEN macro-instruction performs the following operations on the file when the object program is run:

1. The file is made available for processing.
2. The tape is rewound (if desired).
3. The tape label is processed if the DTF indicates that the tape is labeled. For input files, the OPEN macro reads and checks the header tape label; for output files, OPEN checks the retention code of the mounted reel and writes a new label if the code indicates that the information is no longer valid.

The operations performed for the first reel of a multi-reel file are performed automatically for each succeeding reel within the file. The checks are made as the end of one reel is reached and before the use of records from the next reel. This is done as an automatic part of the GET and PUT macros; the programmer need write only one OPEN for all reels of a multi-reel file.

CLOSE. When a tape file is no longer needed, it is removed from use by executing the macro-instruction CLOSE. Like the other macros, this one may have a symbolic label; the operand field contains the names of the files being closed, with the names separated by commas if there is more than one. The following operations are performed on output files:

1. The file is made unavailable for processing.
2. The tape is rewound (if desired).
3. Any records still in the output area are written on tape, which takes care of partly filled blocks. The routine then writes a tape mark, followed by the trailer label, followed by another tape mark.

For input files, the operations are:

1. The trailer label block and/or record counts are checked, if this action has been specified in the DTF macro.
2. The tape is rewound.

GET. This macro performs all the operations required to obtain another record and make it ready for processing. The programmer is thus relieved of the hours or days of programming required to accomplish all of the following:

1. If all the records in the previous block have been processed, another tape block is read.
2. If all the records have not been used, a new record is made available.
3. If a tape error is detected in reading, the routine backspaces the tape and reads again. If the difficulty is nothing more serious than a speck of dust, which is often the case, the backspacing and rereading will often dislodge it and the second reading will be correct. However, if the tape is still not readable after several attempts to reread it, the routine takes whatever action the programmer has decided should be taken in such a situation.
4. If the end-of-reel condition is detected in reading, the trailer label block and/or record counts are checked, and a character in the trailer label is inspected to determine whether another reel of the same file follows. If the DTF macro specifies special routines for the end-of-reel and end-of-file conditions, a branch is made. If not, the tape is rewound and the tape on the alternate unit for this file is prepared for use.

The GET macro, which as usual may have a symbolic label, specifies in the operand field the file from which a record is to be obtained. All of the above operations follow automatically (as far as the programmer is concerned).

PUT. This macro is analogous to GET, except that it refers to output files. It performs the following operations:

1. A record from an input area (or from a working storage area) is moved to an output area. If this record fills the output area, the block is written on tape.

2. If an error is detected in the writing, the tape is backspaced and rewritten. If the record is still bad, a section of tape is erased and the record is written again. If an extended section of tape is bad the routine will take whatever action the programmer has decided should be taken in such a situation.

3. If the end-of-reel reflective spot is detected during writing, the trailer label is written (with an indication that another reel follows), the reel is rewound, and another reel used for further writing with this file.

It is realized that a quick sketch of this sort does not give the reader enough information to begin writing useful programs with IOCS. It is hoped, however, that if the general idea of how the system can be used has been grasped, then the reader will have no particular difficulty in picking up the details. In order to get a little better feel for the use of the system, we may consider some examples.

Let us first rewrite the illustrative program of the previous subsection, with blocking of the tape. The block diagram, Figure 5, is considerably simpler, even though a great deal more is being done. The OPEN box, for instance, takes care of all label checking. The PUT box takes care of all tape writing, blocking, and checking. The fact that the output tape is blocked would be specified in the DTF for this file, which we are not showing. Except for the DTF, all we have to do to handle blocking is to set up an output area large enough to hold a complete block. The CLOSE box takes care of writing a trailer label, writing tape marks, and rewinding the tape.

The Autocoder program shown in Figure 6 is not especially difficult either, although the new coding form makes it appear a little strange. Notice the free form in which the operands are written. About the only restriction in writing the operands is that there be not more than one blank space within the operand field, since two consecutive blanks indicate the end of the field. The remarks may begin anywhere after two blanks; it is common practice to start all remarks in the same column, for ease of reading. Notice also that the operation field is now five columns instead of three, to allow for the augmented operation codes and the macro operation codes.

This program uses two new Autocoder pseudo operations. The first is Define Area, for which the operation code is DA. It is used to set up an area of storage that can be referred to in the PUT macro. The 3 specifies that three groups of 80 characters are being set up for this area; the X1 specifies that index 1 is to be used by the object program in stepping through the records in the block; the G specifies that a group mark with a word mark is to be set in the character position to the right of the 240-position area. We shall see in the next subsection how the DA instruction can also be used to define fields within the area.

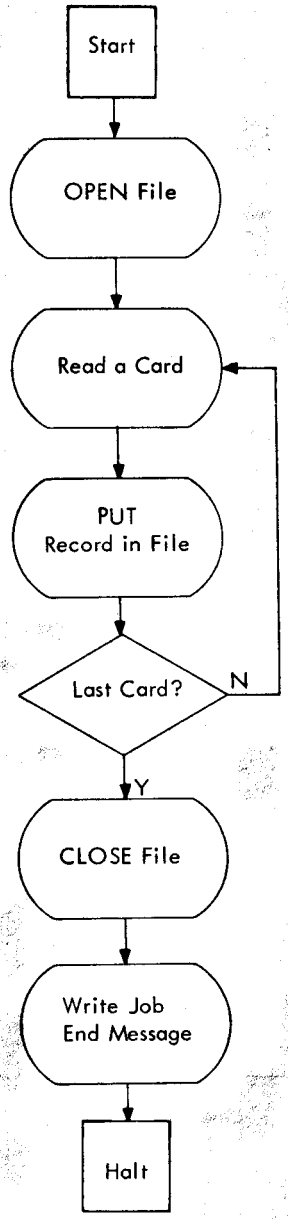

Figure 5. Block diagram of the Autocoder operations to do the job diagrammed in Figure 3

Line	Label	Operation	OPERAND
01	START	OPEN	OUTPUT
02	READ	R	READ A CARD
03		PUT	CARD,OUTPUT
04		BLC	LAST CARD Q
05		B	NO
06	LASTCD	CLOSE	OUTPUT
07		LCA	MESSGI,,212
08		W	
09		H	*-3
10	MESSGI	DCW	@JOB FINISHED@
11	OUTPUT	DA	3X80,,X1,,G
12	CARD	EQU	1
13		END	START
14			

Figure 6. Autocoder program to do the job diagrammed in Figure 5

The second new pseudo instruction is Equate, for which the operation code is EQU. It is used here to establish 0001 as the absolute equivalent of the symbol CARD. This would ordinarily be done with the DA operation, but it is always illegal to use an instruction of this type to set up an area or a constant in the card read area, since this will ordinarily disturb program loading operations.

Note that there is no "count" field on the Autocoder form. Numerical constants are entered into the program with as many character positions as there are digits in the constant. Alphameric constants must be preceded and followed by the symbol @.

Review Questions

1. *Distinguish between a record count, control total and hash total. List advantages and disadvantages of each, in terms of such considerations as simplicity of computation, degree of checking provided, and types of errors checked against.*
2. *Why is record blocking used?*
3. *It would seem that as many records as possible should be placed in a block—that is, that the blocking factor should be as large as possible. What sets a limit on the degree of blocking?*
4. *List the functions of a header label.*
5. *What does the programmer do to incorporate the desired IOCS routines in his program?*
6. *What are the principal differences between SPS and Autocoder?*

8.4 Inventory Control Case Study

The following case study will illustrate the use of many of the techniques and concepts that have been discussed in this section.

Inventory control, as used here, refers to the process of keeping an up-to-date record of the status of every part in the inventory of a manufacturing company. In the example to be studied here, we are given a master inventory tape containing a record for each stock item. Each record contains the part number and the quantity on hand. The object of the inventory control application is to maintain this file so that it represents the status of the actual inventory as of the most recent updating of the file.

Changes in stock status are introduced into the data processing system in the form of a deck of cards. Each card shows the part number, a code to indicate whether the card represents a receipt of more

stock items, a recount, or an issue to the manufacturing operation or to a customer. There can be more than one card per part number, such as when a shipment of stock items has been added to the inventory and a shipment has been issued to a customer, during the period represented by the update. Again, there could very well have been several shipments to customers during the period. Before the transaction deck reaches the computer, it has been sorted on part number and classification code in such a way that for any one part number, recounts are first, then receipts, and then issues. This sequencing of the cards within one part number will guarantee that if a large shipment is received and another large shipment issued during the transaction period, for instance, the data processing system will not erroneously indicate that the large issue created an out-of-stock condition.

Recounts are coded so as to sort at the front of the transactions. This is done on the assumption that the recount quantity is taken *before* any transactions. Another common way to handle adjustments is to enter them as changes, either plus or minus, rather than entering a complete new count as we have done here.

The master file is in ascending sequence on part number.

Our job is to use the transaction deck and the old master file to produce a new master file that shows the inventory status after the changes described by the transaction file.

A block diagram of the computer operations, including card reading and tape handling, is shown in Figure 7. The actions called for by this block diagram are best understood by considering several situations and seeing what the block diagram says to do for each. Suppose first that there is a single transaction card for the first item in the master file. We begin by setting a switch to what is called the ON position. We read this first card and GET the first master record. We next ask whether the part numbers in the transaction and the master are the same. They are, by assumption, so we use the transaction code to determine whether this is a recount, receipt or issue, and update the master record accordingly. We ask if this was the last card. The answer is no, so we read another card and return to the comparison to determine whether the part numbers are equal. We assumed a one-card group for the first stock item, so the comparison this time will show that the master part number is less than the transaction part number. This will cause the updated master record to be PUT into the output area, after which we return to GET another master record.

If there had been several cards for the first part number in the file, they would have been processed without writing the new master record, by the repeated use of the loop containing the classification code test.

If the first transaction card is for some master file record other than the first, then all of the records prior to this one will be PUT into the output area before the part number comparison shows equal.

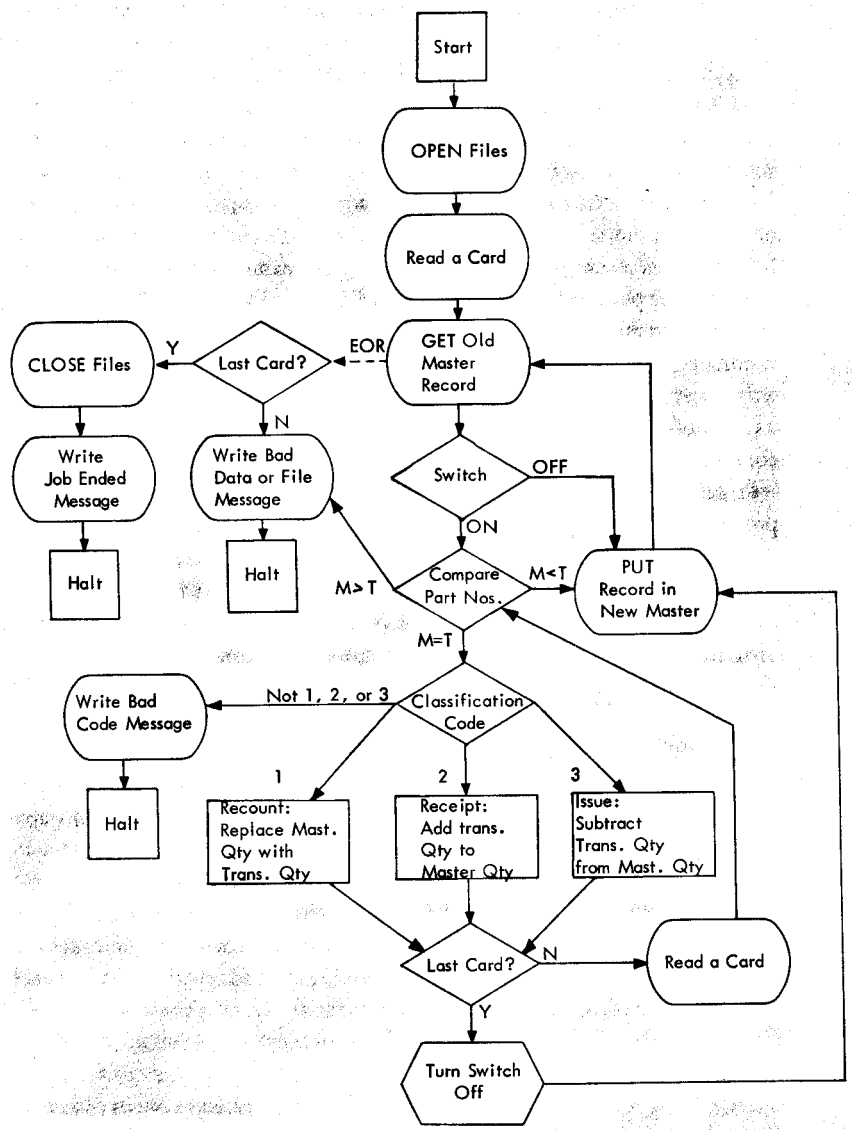

Figure 7. Block diagram of an inventory control procedure

If at any time the part number of a master record turns out to be greater than the transaction part number, then an error is indicated. This could be caused either by an out-of-sequence file or by an incorrect transaction part number. It could happen, for instance, if the first transaction card had a part number smaller than that of the first master record. It is clear that in this case reading more master records is never going to find a matching part number, since both files are assumed to be in ascending sequence on part number. It could also happen if a transaction card had a part number that was not the same as the part number of any record in the master file. This test does not give an absolute guaranty that no transaction part numbers are incorrect, but only that if an incorrect part number does not match we will catch it.

Suppose now that the last transaction card is a single-card group corresponding to the last master record. This time the last-card test will give a yes answer, which causes the switch to be set to the OFF position, after which we write this last new master and go back and try to GET another master record. This will be found instead to be the tape mark, and the GET routine will take us out of the processing loop. We shall see why another last-card test is necessary in a moment.

If the last master record has corresponding to it several transaction records, then the analysis is the same as in the preceding paragraph except that we will go around the updating loop the necessary times before finding the last card.

Suppose next that the last transaction card corresponds to a master record before the end of the tape. In this situation we must still copy the remaining master records in unchanged form from the old master to the new master. This is why the switch is necessary. After the master record corresponding to the last item in the transaction file has been PUT, all following master records are going to show larger than the final part number. But now this is not an error. Therefore, when the last card is detected, we set a switch to bypass the comparison and simply read and copy master records until the tape mark is sensed and the GET routine takes us out.

One final error possibility remains. The last transaction card might have an incorrect part number greater than that of the last master record. In this case, we would go around the master record reading and writing loop looking for a matching master part number. Naturally we would never find it, but would instead finally detect the tape mark. Now, when we ask whether the card most recently read was the last one, the answer will be no, indicating that the end of the master file was reached without having reached the end of the transaction deck. This is, of course, an error and we so indicate.

This is a very standard type of block diagram in sequential file data processing. Its basic logic applies to many applications besides inventory control and it applies when both files are on tape. Even the

reader who may not be directly concerned with magnetic tapes will profit by a careful study of the logic of this block diagram.

We may now consider the details of the actual program for this example. First assume that the transaction cards have a five-character part number in columns 1 to 5, a single-digit classification code in 6 and a four-digit quantity in 7 to 10. The classification code is a 1 if the transaction represents a recount quantity that should replace the master record quantity, a 2 if it represents a receipt, and a 3 if it represents an issue. The master records each consist of a five-character part number and a five-digit quantity. The master tape is blocked with 20 records in each block. If the last block is not full, it is padded out with blanks.

The program for this job is shown in Figure 8. We begin by opening the two files and reading a card. OLDMST and NEWMST are the names of the two files; these names would have to be in the DTF macros for the two files, which we are not showing. The DTF macros would also specify, among other things, the blocking factors for the two files, and would name EOR as the routine to which the GET routine should branch when the end-of-reel is detected in reading the old master.

Next, an old master is placed in the work area. The switch is coded as a No Operation, so that it will have no effect until it is changed to a Branch. After this we compare the part number of the transaction from the card with the part number from the old master. The next two instructions branch out to the appropriate routines for the cases where the two are not equal. If they are the same, we reach the three branch instructions that determine whether the transaction is a recount, receipt or issue.

This testing of the classification code uses a new variation of the Branch instruction, called Branch If Character Equal. This instruction says to branch to the instruction shown in the I-address if the single character at the B-address is the same as the d-character. If it is not, we go on in sequence. This is used here because we do not like to assume that the classification code will always be 1, 2 or 3. This is placing a little too much reliance on fallible human beings. Therefore we test explicitly against all three codes, branching to the correct instruction when one of them is found. If the code turns out to be not any of the three, as it could through mispunching, then we write an error message and halt.

If the transaction is a recount, we replace the master quantity with the transaction quantity. If it is a receipt, we add the transaction quantity to the master quantity, and if it was an issue we subtract the transaction quantity from the master.

Setting the switch is a simple matter of moving a B to the operation code, thus changing it from a No Operation to a Branch.

Line	Label	Operation	OPERAND
01	START	OPEN	OLDMST,,NEWMST
02		R	READ A CARD
03	READM	GET	OLDMST
04	SWITCH	NOP	B ØP CHANGED TØ BRANCH LATER
05	COMP	C	TRANPN,,MSTPN
06		BH	ERRØR ERRØR IF MASTER HIGH
07		BL	B GØ TØ WRITE IF MASTER LØW
08		BCE	RECØUN,,CØDE,,1 TEST CØDE
09		BCE	RECPT,,CØDE,,2 X
10		BCE	ISSUE,,CØDE,,3 X
11		LCA	MESSG1,,225 ERRØR IF NØT 1,, 2,, ØR 3
12		W	
13		H	*-3
14	RECØUN	MCW	TRANQY,,MSTQY
15		B	LCTEST
16	RECPT	A	TRANQY,,MSTQY
17		B	LCTEST
18	ISSUE	S	TRANQY,,MSTQY
19	LCTEST	BLC	SWSET LAST CARD Q
20		R	CØMP NØ — READ CARD & BRANCH
21	SWSET	MCW	SWBR,,SWITCH
22			

Figure 8. Autocoder program for the inventory control procedure diagrammed in Figure 7

Line	Label	Operation	OPERAND
01	B	PUT	ØLDMST,,NEWMST
02		B	READM BACK TØ GET ANØTHER MASTER
03	EØR	BLC	WRAPUP LAST CARD Q
04	ERRØR	LCA	MESSG2,,229 NØ — ERRØR
05		W	
06		H	*-3
07	WRAPUP	CLOSE	ØLDMST,,NEWMST
08		LCA	MESSG3,,210 WRITE JØB END MESSAGE
09		W	
10		H	*-3
11		H	
12			

Figure 8 Continued

Page No. 3 of 3

Line	Label	Operation	OPERAND
01	TRANPN	EQU	5
02	CØDE	EQU	6
03	TRANQY	EQU	10
04	ØLDMST	DA	20X10,X1,G
05	MSTPN		1,,5
06	MSTQY		6,,10
07	NEWMST	DA	20X10,,X2,,G
08	MESSG1	DCW	@BAD CLASS CØDE JØB HALTED@
09	MESSG2	DCW	@FILE ØR DATA ERRØR JØB HALTED@
10	MESSG3	DCW	@JØB FINISHED@
11	SWBR	DCW	@B@
12		END	START
13			

Figure 8 Continued

The Define Area instructions perform a new function for us in this program. Notice that following each DA there are a few lines with no operation code. These are part of the DA, defining fields within it. The first location of the defined area is considered location 1. The high-order and low-order positions of the fields are punched beginning with the leftmost column of the operand field. These two numbers are separated by a comma. The processor places a word mark over the leftmost location of each field defined in this way. The fields within the defined areas may now be referred to symbolically, without, of course, our knowing where they are located in storage. Both tapes have blocking factors of 20, which requires setting up sufficient space in the input and output areas for 200 characters.

The wrap-up for this job consists simply of closing both files and writing the "job finished" message to the operator.

This case study has been deliberately simplified to let us concentrate on the tape operations involved. It should be realized that a normal inventory control task includes a great deal more than we have shown here. Furthermore, we have taken as straightforward an approach to the tape manipulation as possible in order to keep this first sample of work with blocked tapes as uncomplicated as possible. Section 10 is devoted entirely to a thorough study of a more nearly complete inventory control application, both from an application standpoint and from a programming standpoint.

Review Questions

1. *In the program of this subsection, when would the result of the last-card test following the tape mark test ever be yes?*
2. *Suppose that the last card of the transaction deck were inadvertently placed at the front of the deck. What would the program do?*
3. *What would the program as shown do if the old master file erroneously had two records with the same part number?*
4. *Why is it more convenient to have two card-reading instructions than to have only one?*

Exercises

*1. Given a tape with unblocked records of 100 characters each, with a tape mark following the last record. Draw a block diagram and write an SPS program to print each record exactly as it appears on tape. Each page of the output is to have a maximum of 50 lines.

2. Given a tape with unblocked records of ten characters each, consisting of a four-digit account number and a six-digit dollar amount. The records are in ascending sequence on account number; there are duplicate account numbers. At the end of the tape there is a tape mark, followed by a trailer label, followed by another tape mark. Positions 6-10 of the trailer contain a count of the number of blocks in the tape.

Draw a block diagram and write an SPS program to summarize the amounts by account number and make a block count check.

3. Estimate the time required to execute the program of Exercise 2, assuming that there are 10,000 blocks on the tape and that a 729 II tape at low density is used.

4. Draw a block diagram of the operations necessary to read a block, check for tape errors, and reread up to nine times if there is an error. If the error persists, an error message is to be written.

*5. Modify the block diagram and program of the inventory control case study as follows. Another type of transaction, with a code of zero, may be present in the deck: an addition. An addition record represents a new part number, the record for which is to be added to the master file. However, do not add the new record to the file if it has the same part number as a record already in the file. "Adding" the record to the file consists of getting the card record into the proper tape format and moving the part number and quantity to the output area.

6. Modify the block diagram and program of Exercise 5 as follows. A fifth type of transaction, with a code of 4, may be present in the deck: a deletion. A deletion record represents a part number the record of which is to be deleted from the file, that is, not written into the new master. Write a short tape containing all deleted master records.

*7. Given two tapes having the same record format: a four-character employee number, followed by 90 characters of information about the employee. Both tapes are in ascending sequence on employee number, and there are no duplications. Draw a block diagram and write an Autocoder program to merge the two tapes—that is, produce a third tape containing in ascending sequence all records from the two input tapes.

8. Same as Exercise 7, except that a sequence check is to be made on both input tapes. This will require storage of the most recent employee number from each input tape, to use in determining whether the record just read is greater than the previous—for each tape.