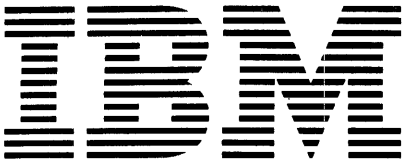


Application Program

H20-0228-0

1440/1460 Administrative Terminal System

(1440-CX-07X and 1460-CX-08X)

Programmer's Manual

This system consists of control and functional programs that permit many different text-processing activities to be carried on simultaneously through different terminals. This reference manual contains a general description of the programs. It also contains information required by a programmer who wishes to add peripheral programs or new functions to the system.

PREFACE

The material in this manual relates to Version 2 of the IBM 1440/1460 Administrative Terminal System program. It obsoletes and replaces the Preliminary Edition issued with Version 1 of the program.

This publication was prepared for production using the IBM 1440/1460 Administrative Terminal System. Page impressions for photo-offset printing were obtained from a typewriter terminal and reduced 15 percent.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, New York 10601.

CONTENTS

INTRODUCTION

It is intended that this manual serve as a reference manual for programmers concerned with the Administrative Terminal System (ATS). The manual includes a description of the system, beginning with a general discussion of techniques involved and developing into a detailed treatment of the programs and indicators. Material is presented to permit each reader to investigate the subject as his own interests and requirements demand.

It is assumed that the reader of this manual is familiar with the following IBM publications:

IBM 1440/1460 Administrative Terminal System, Terminal Operator's Manual (H20-0185)

IBM 1440/1460 Administrative Terminal System, Console Operator's Manual (H20-0227)

It is also assumed that the reader is familiar with the IBM Systems Reference Library publications describing the various components of IBM 1440 and 1460 Data Processing Systems that may be used in connection with the IBM 1440/1460 Administrative Terminal System. The reader should, in addition, be generally familiar with the IBM Systems Reference Library publications describing the IBM 1401/1440/1460 Autocoder (On Disk) programming system.

GENERAL DESCRIPTION

The IBM 1440/1460 ATS program is concerned with allowing up to 40 remote terminals simultaneous access to a data processing system. A wide variety of program-controlled operations are available to each terminal, operating independently.

Keystrokes entered at the terminals are collected by the system as strings of characters called text streams. These text streams are accumulated in core storage in 100-character, dynamically allocated data areas called core blocks. Twenty-five characters of the data area are necessary for control, 75 for text. When 75 characters have accumulated in a core block, the text is written to IBM 1311 Disk Storage using a chaining technique. Thus, the text stream input from each terminal is in the form of a chain of blocks, each link containing up to 75 characters of text. This chain is called Working Storage. Working Storage for each terminal is both on the disk and in core storage.

Hardware and formatting requirements make the Working Storage (input) text stream unsuitable for transmission back to the terminal. Whenever a printout is requested from the terminal, a copy of the original text stream is program-generated. This special output text stream is the logical reverse order from the input text stream. Text streams are illustrated in Figure 1.

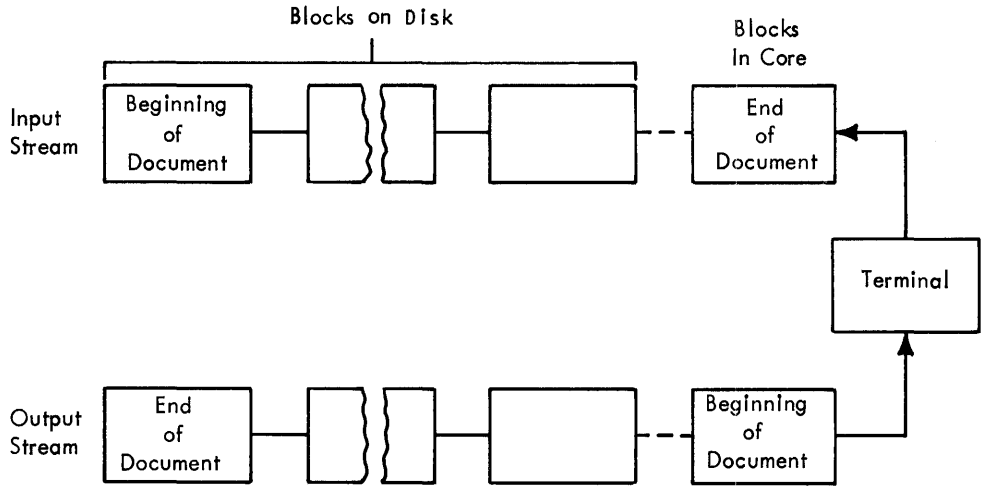


Figure 1. Input and output text streams

The input text stream is kept intact until deliberately destroyed as a result of a program request from the terminal operator. In contrast, the output text stream is destroyed as it is used.

Documents in Working Storage may be copied into another area of the disk called Permanent Storage. The terminal operator assigns the document a number. This number refers to a specific location in an index, which, in turn, contains the disk address of the beginning of the document. Permanent Storage is chained in a manner analogous to Working Storage, in blocks of 900 characters.

When accumulating text from the terminals, the system does not ordinarily examine the text stream for content. Thus, to distinguish a program request from ordinary text, the terminal is provided with

a special key called the Attention key. The Attention key, like all other keys on the keyboard, generates a unique character that becomes part of the accumulated text stream. The terminals and the IBM 1448 Transmission Control Unit (multiplexor), as modified, recognize this character and cause a computer interrupt after it is received. Any line containing an Attention character is a request for a program function. The characters keyed following the Attention character indicate which function is requested. The line cannot be interpreted until it is completely entered. Thus, no action is taken by the system until the Carrier Return character is received, signifying the end of the line. At this time, the line is interpreted, and the request is processed. During this period, the terminal keyboard is locked, and the operator is denied access to the system. In all cases, except where the request is supposed to remain for implicit formatting, the program request line (Attention action) is deleted from the text stream.

ATS is not a single program but a system of three types of programs: the control program, application programs, and peripheral programs. The control program, or Scheduler, schedules the workload for the application programs, performs all disk input and output, and keeps the text flowing between the multiplexor and the processor. The application programs perform the functions specifically requested by the terminal operators. For example, the Format and Print program generates the output stream when a terminal requests a printout. Peripheral programs perform unrelated background functions such as tape to printer.

With one exception, all application programs are resident on the disk. The exception is the End of Unit program (EUNIT), which is described later in this manual. When an Attention action is taken by one of the terminals, the Scheduler reads the appropriate application program from the disk into a special overlay area. When the program is finished with its work, it indicates that fact to the Scheduler, and the next requested program is read into the same area.

Every application program is assigned a unique three-character entry in the Program Status Table (PST). This table contains the location of every application program, whether disk- or core-resident. (Disk addresses are in a compressed form.) The programs are numbered by their relative positions in the PST. This number, called the PST number, identifies the program and determines its relative priority. Thus, when two programs are waiting for the overlay area, the program with the lower PST number (higher in the PST) is overlaid and executed first.

The character immediately following the Attention character in an attention action determines the application program that will process the request. Program identification is accomplished by reference to the Character Status Table (CST). This table is composed of 64 entries, one for every possible BCD character. An entry consists of the PST number of the program that processes the request. When an Attention action is taken by one of the terminals, the character following the Attention character is located, and the entry in the CST corresponding to the character is examined. The entry in the CST will contain the PST number of the requested program. The PST is then examined to locate the program, and the request is processed. For example, if a terminal requested the next number, the characters:

ATTN n CR

would appear in the text stream. The character following the Attention character is "n"; thus, the entry corresponding to "n" would be referenced in the CST. The entry would contain the PST number of the Miscellaneous Attention Actions program (MISAC). The request would

then be sorted into the line or queue of programs waiting for the overlay area according to the PST number. When the queue was processed to the point of the entry, the position indicated by the PST number would be examined in the PST, and the physical location of the program determined. The program would then be read from the disk and executed.

The CST and PST entries are the communication links to the application programs. When a new application program is added, appropriate entries in these two tables are the only system modifications that are required.

PROGRAMMING TECHNIQUES

This section contains a general description of some of the programming techniques used in ATS. It must be thoroughly understood before the detailed operation of the system is studied. The ways that these techniques are used in various parts of the ATS program will be covered in the detailed descriptions of these parts in later sections of this manual.

DYNAMIC STORAGE ALLOCATION

Dynamic Core Allocation

Information is usually stored by means of several contiguous words or blocks having addresses and block lengths that are known to the programmer. Access to a block is then controlled by the known address modified by an index register. An example is shown in Figure 2.

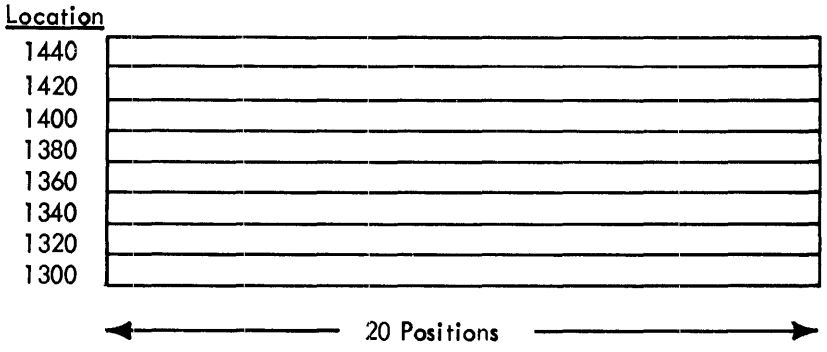


Figure 2. Fixed core blocks

In a fixed table, block entries are numbered from zero to N-1, where N is the total number of blocks. To reference any block, the block number times the length of the block is loaded into an index register. Reference is then made by addressing the desired field in block zero plus the contents of the index. Thus, in the above example, to reference the sixth character in the fifth block, 4 (the block number) times 20 (the length of the field) is loaded into an index register. The number 1305 (the character position in block zero) added to the index will yield character position 1385, or the sixth character in the fifth block.

This organization lends itself well to tables of fixed length that change infrequently. However, when dealing with data of unknown length that must be constantly manipulated, a more flexible technique is desired. Dynamic storage allocation is such a technique.

If a portion of each register or block is allocated to control information, namely, the address of the block following, the scheme would appear as shown in Figure 3.

Location	Pointer	Location	Pointer
300	400	700	800
400	500	800	900
500	600	900	1000
600	700	1000	Blank

Figure 3. Core blocks chained together with pointers

The address in the block is called a pointer because it "points" to the next register. A pointer that is blank denotes the end of the table. This technique is also called chaining.

It is immediately apparent that these registers may be randomly located throughout memory. Thus, contiguous registers are not necessary. Also, sorting becomes merely the manipulation of addresses, rather than the movement of entire blocks.

In most systems using dynamic storage, the concept of "free storage" is used. Free storage is a chain of available blocks which are unused at the moment and which may be requested. The technique is developed so that when a program needs an additional block, it "takes" one from free storage, adds it to its chain of blocks, and reunites the free storage chain around the missing element. Actually, "taking" a block from free storage entails only the changing of the pointers. The pointer in the last block of the chain associated with the program is changed to point to the block to be extracted from free storage, while the chaining of the free storage blocks is altered to exclude the "missing" block. An example is shown in Figure 4.

Before "Taking" Block At 1000

	Program's Chain		Free Storage Chain
Location	Pointer	Location	Pointer
300	400	700	800
400	500	800	900
500	600	900	1000
600	Blank	1000	Blank

After Program "Takes" Block At 1000

	Program's Chain		Free Storage Chain
Location	Pointer	Location	Pointer
300	400	700	800
400	500	800	900
500	600	900	Blank
600	1000		
1000	Blank		

Figure 4. Core blocks in two chains

Conversely, when a program has finished with a block, it is "purged" or "returned" to free storage. Again, the purging of a block entails merely the rechaining of addresses.

Although the examples given above are contiguous, the random nature of this method should be obvious. Also, the program may chain in a block at any point in its Working Storage, not necessarily at the end.

Knowing the address of the first block of a chain enables the program, by following the chaining addresses, to find any particular element in the chain. This search may be facilitated if, in addition to a chaining address pointing to the next block, another address is used to point to the preceding block. Such a scheme is illustrated in Figure 5.

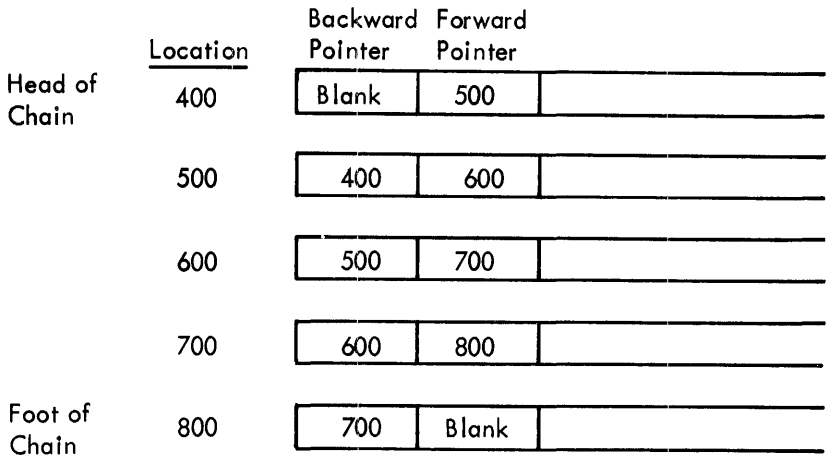


Figure 5. Core blocks with two-way chaining

In this scheme, the first address points "backward" in the chain, while the second address points "forward". With this additional address, it is possible, given any block in the chain, to search in either direction. Note that when the end of the chain is reached going either forward or backward, the chaining address is blank.

Using this method of two-way chaining, it is desirable to know the locations of both the beginning and the end of the chain, which are called the head and the foot of the chain, respectively. Pointers to these locations are stored in fields with fixed locations termed the "head pointer" and the "foot pointer".

Dynamic Disk Allocation

Although the technique described above refers to allocation in core storage, an analogous procedure makes very effective use of random (disk) storage. When dynamically allocated disk storage is used, the chaining addresses become sector addresses. Here again, two-way chaining is desirable to increase the efficiency of a search.

Dynamically allocated disk storage is clearly a superior technique when random-length records must be processed. The storage space used by the chaining addresses is trivial when compared to the large

portions of the disk that are unused in a contiguous technique. Also, because each cylinder is fully utilized before going to the next and because no reshuffling of records is necessary, the number of disk accesses is greatly reduced, especially the disk "seek". Thus, system efficiency is maximized.

With record elements that exceed one sector in length, the dynamic technique can be implemented by chaining groups of sectors. Thus, with long elements, the use of chained groups of sectors (where the groups of sectors are contiguous) results in optimum use of the disk.

A concept analogous to "free storage" is used with dynamic disk allocation. Instead of a chain of available blocks (sectors), a pool of available sector addresses is kept. These addresses are allocated from one end of the disk (highest addresses, for example) and as more sectors are required, they are added to the pool, moving sequentially across the disk. For example, the pool might begin by including all of the addresses in the highest cylinder of the disk. As these addresses are used up, the addresses in the next cylinder down would be requisitioned.

When a program has finished with a sector, that sector is "purged" or made available for reuse. Again, because a pool of disk addresses, not a chain of free storage, is used, a disk sector is purged by returning the address to the pool. No additional access to the disk is required.

Dynamic Allocation in ATS

Several variations of the above-described allocation techniques are used in the ATS. The one-way chaining technique is used to link the elements in the list area, which is described in the section "List Processing". Two-way chaining is used in several variations in the 100-character core blocks used to receive text before writing it to the disk (described in the section "Core Blocks"). In both the list area and the core blocks, the concept of free storage is used.

Disk storage is allocated by chained sectors for the text as it is received and manipulated by the program. A pool of sector addresses is kept as explained above.

In addition to the chained sector storage, called Working Storage, text to be stored permanently is written in chained groups of ten sectors. A similar, but slightly modified, technique of the address pool is used here.

MULTIPROGRAMMING

Multiprogramming is the apparent simultaneous operation of two or more programs. Multiprogramming is not to be confused with multiprocessing, which is the execution of two or more programs that are independently and simultaneously executing instructions. An example of multiprocessing is the simultaneous operation of the Central Processing Unit (CPU) and a data channel in the IBM System/360. Multiprogramming is the apparent simultaneous operation of two or more programs in a single CPU.

When time is not a prime factor, a program can afford to wait for input/output operations to be completed. However, in a real-time situation in which the program must be able to react to external conditions within a specified amount of time, processing time must be conserved.

The "seek" instruction in direct access storage is a very time-consuming input/output operation. This is so because it involves the mechanical motion of the access arm. Because seek instruction may be given, and processing may occur at the same time that the arm is in motion, it does not seriously delay the program. Moreover, an inquiry may be made to see if the arm has reached its destination. If it has not, more processing may be done.

Multiprogramming is a technique used to take advantage of the processing time while the arm is seeking. In its implementation, two or more programs are in core at the same time, each with its separate function to perform. When traffic with the disk is required by one of the programs, it branches to an executive routine conveying the address of the sector(s) that it needs. The executive routine proceeds to give the disk seek and then branches to another program in core. The second program works until it has completed its task or until such time as it needs access to the disk file. If disk access is required, the second program branches to the executive routine with an appropriate calling sequence giving the address of the sector(s) required. This second request is placed in a queue behind the first request.

The logic in the Scheduler is that waiting input/output operations that have been completed are processed before any new programs are allowed to operate. Thus, in the above example, after the second program requested a disk access, the Scheduler would first check to see if the input/output of the first program was completed. If so, it would branch to the first program. If not, it could branch to a third program. The actual disk read or write operation is performed whenever control is given to the Scheduler by an interrupt (see the section "The Multiplexor").

In a multiprogrammed system, from the individual program viewpoint, input/output is instantaneous. It requests I/O and, by the time it executes the next instruction, the operation is completed. Thus, several programs are all operating, from their individual viewpoints, at the same time.

LIST PROCESSING

In a multiprogrammed system with a conceivably long queue of waiting input/output requests, some manageable technique must be devised to keep track of the I/O demands. Moreover, if the system is running in real time, it must react to external conditions arising on a random basis -- conceivably, several arising simultaneously. Thus, demands on the system must also be arranged in a queue.

One solution to these problems is the concept of a list. A list is a group of registers, containing control information, placed in a string or queue so that they may be processed sequentially.

The control information in a list queuing input/output requests, for example, might include the type of operation (that is, read or write), the area in which the operation is to be made, and the address to branch to when the operation is complete. Such a list is shown in Figure 6.

Entry	Operation	Input/ Output Area	Branch Address
1	1	10200	6056
2	1	12500	14561
3	/	10600	4480
4	/	10500	4480
5	1	10400	15975

Figure 6. Example of an input/output request list

The first item of each entry is the operation ("1" a read, "/" a write); the second item is the address of the input/output area; and the third item is the address to which to branch when the operation is completed. This kind of list is used for the Work in Progress and Disk Arm queues in the ATS program.

A different set of controls would be needed in a list used to handle system demands. Such a list is shown in Figure 7.

Entry	Terminal Number	Program Number
1	39	03
2	08	12
3	12	06
4	00	06
5	16	09

Figure 7. Example of a system demand list

The first item in each entry is the terminal number, and the second item is an encoded number (PST number) referencing the program that is to do the work.

The queuing of a list is handled in two ways. Either it is "first in, first out" (FIFO), or it is "last in, first out" (LIFO). Assuming that the list entries were processed by always taking the top entry, a FIFO system would place the latest entry at the bottom of the list, while a LIFO system would place the latest entry at the top. In addition to a basic queuing technique, it is sometimes desirable to have some kind of priority implementation. This can be accomplished in a FIFO list by sorting the entries so that priority requests are always pushed to the top. This kind of list is used for the New Job queue in the ATS program.

With the list concept, an adequate queuing technique is obtained. This technique can be made much more powerful, however, if it is

implemented with the dynamic storage concept. Thus, the list entries can be located at random, connected only by chaining addresses. Figure 8 contains an example of a dynamic list.

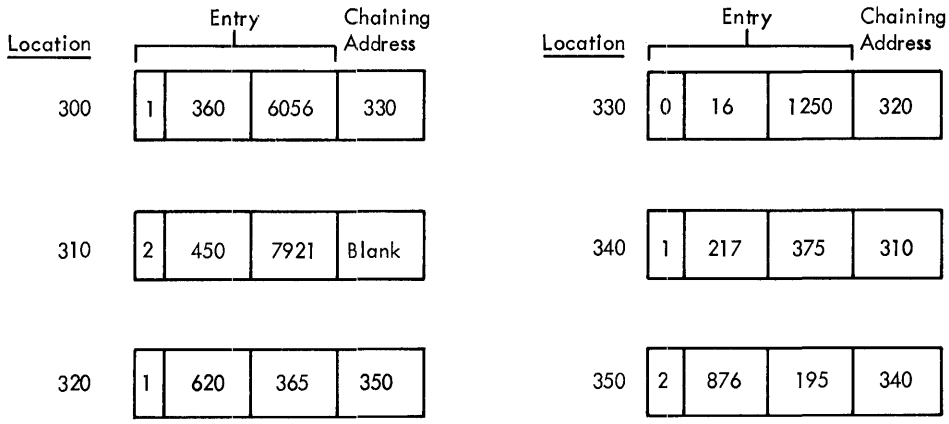


Figure 8. Example of a dynamic list

The last item in each entry is a chaining address (pointer). This particular list begins in location 300 and ends in location 310. Note that the blank chaining address denotes the end of the list.

The immediate advantage of a dynamic list is that the addition of new entries and the task of priority sorting become simply the manipulation of chaining addresses, eliminating the necessity of complete record moves. Also, the program required to manipulate a dynamic list is extremely short.

In a real-time system, there are a number of operations that must be queued. For example, there might be five disk drives attached to the system. If so, it is apparent that I/O requests can be more efficiently handled if they are queued up for the respective arms where access is required. In this case, five queues, or one for each arm, would be desirable. In addition, a separate queue of completed I/O requests would be needed. As explained previously, a queue is necessary to process outside demands on the system. The handling of seven separate lists is considerably alleviated if the free storage concept is considered. If a chain of Available list entries is used, there is no reason why all seven lists cannot be in the same "list area". To implement this condition, eight "pointer registers" must be allocated in fixed positions. Each of these pointers points to the top entry in its own dynamic list chain. The Available Pointer, for example, points to the top entry in the free storage list. Each of the five Arm Pointers points to the top entries in its respective list. The Work in Progress Pointer points to the top entry in the completed I/O list and the New Job Pointer points to the top entry in the list of new demands on the system.

In the implementation described above, as in most dynamic storage allocation schemes, a pointer that is blank denotes the end of a chain. When a new list entry is needed, say for the New Job list, it is unchained from the Available list and rechained in its proper priority order in the New Job list. Thus, the addition and priority sort of a list entry can be done in one operation. When the entry has served its purpose, it is purged, or returned, to the Available list.

An example of a dynamic list area at any one instant is shown in Figure 9.

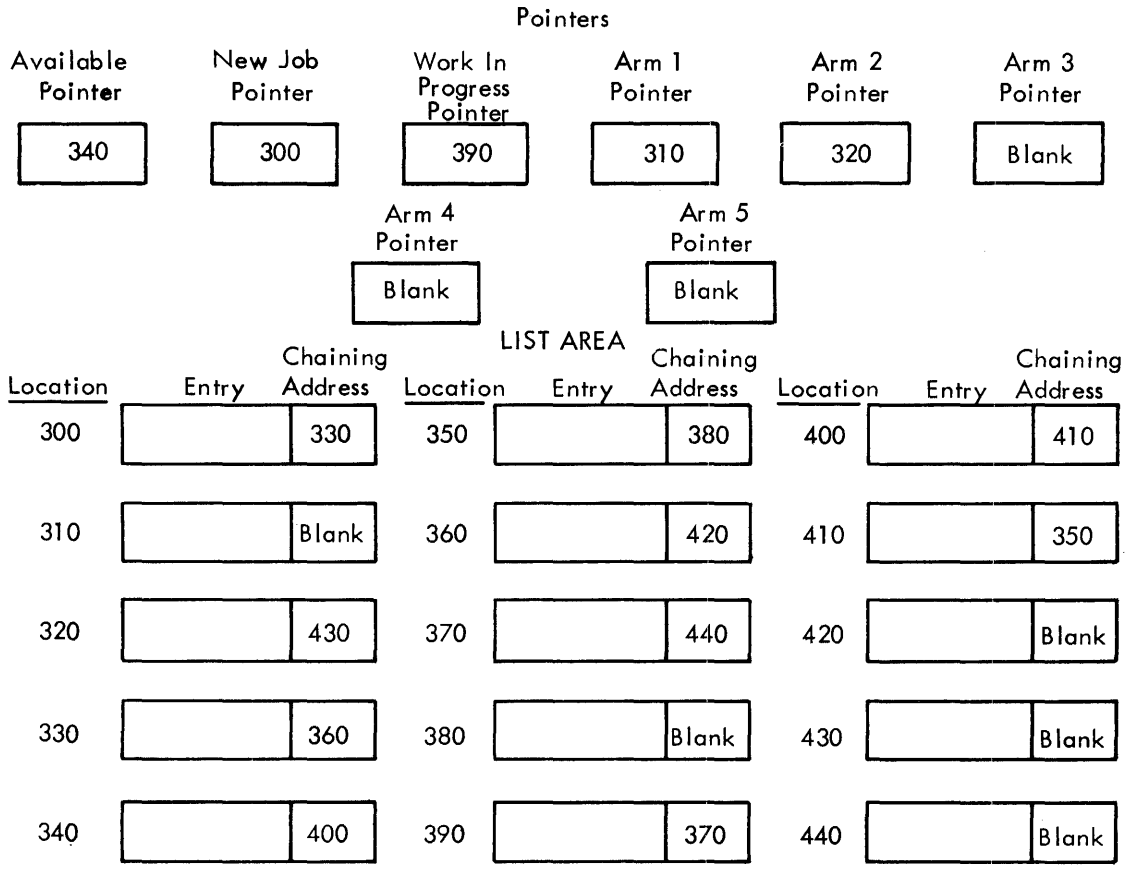


Figure 9. Example of a dynamic list area

Note the blank Arm Pointers, indicating no entries in their lists. Note also that the last entry in each chain has a blank pointer. This is a somewhat simplified example of the List Area in the ATS program.

Although this technique may appear complicated from a programming standpoint, it is simple to implement. Consider the problem of chaining a list element from the Available list to the bottom of the New Job list. Only a two- or three-instruction loop is required to step to the bottom of the New Job list. The address in the Available Pointer is then moved into the pointer of the last New Job entry. The pointer (address) in the new New Job entry is copied into the Available Pointer and is then cleared.

THE SYSTEM

This section discusses some programming aspects of the systems that use the ATS program. Except where the operation of specially modified equipment is described, the IBM Systems Reference Library publications describing the equipment should also be consulted.

THE MULTIPLEXOR

In 1440/1460 ATS, a modified IBM 1448 Transmission Control Unit (multiplexor) is used to buffer text being sent to and received from the terminals. This unit is modified in the sense that many of its functions have been disabled. Communication between the processor and the 1448 is controlled by a table consisting of one ten-character entry for every line attached to the 1448. In the ATS program, this table is called the Multiplexor Status Table (MST). Use of the multiplexor is point-to-point. This means that only one terminal is attached to each line, and the polling function is not used. Because of this, the 1448-1440/1460 communication consists of only a status character (STAC) and a running address (RUNA). The status character occupies the first position of the MST entry and is followed by the three-character running address. The remaining six characters are used for program control and are not referenced by the 1448. They are described in the section "Multiplexor Status Table". An MST entry is organized as shown in Figure 10.

Figure 10. Format of a Multiplexor Status Table (MST) entry

Status Characters

The following status characters may occur in the ATS Multiplexor Status Table:

Receive-Idle (A-bit and 1-bit). This status is set by the program after an End-of-Block (EOB) character is received from the terminal. It resets the 1448 so that it will receive characters. It also causes the 1448 to send the Keyboard Unlock character to the terminal.

Receive (A-bit only). This status is set by the 1448 when any character is received from a terminal.

Receive End of Block (A-bit and wordmark). A special End-of-Block (EOB) character is transmitted to the 1448 by the terminal immediately following the Attention or Carrier Return characters. The EOB character never reaches core, however. Instead, the wordmark bit is set in the status character for that line by the 1448.

Receive End of Storage (A-bit and 8-bit). This status is set by the 1448 whenever it attempts to store characters over a groupmark/wordmark.

Receive Check (A-bit and 2-bit). If an invalid character is received, or if the two-character buffer storage overflows, this status is set by the 1448.

Transmit (B-bit only). This status is set by the program to transmit text to the terminals.

Transmit End of Block (B-bit and wordmark). This status is set by the 1448 when it recognizes a record mark in an output area, indicating the end of the storage area. A wordmark is set in the status character for the appropriate line.

Control (4-bit only). This status can be set by the program after an EOB condition to hold the terminal in a stable waiting condition while it is processing for that line.

The status character appears as follows:

Bit	Status-Function
B	Transmit
A	Receive
8	End of Storage
4	Control
2	Check
1	Idle
WM	End of Block

Whenever an EOB bit is turned on for any line, the EOB indicator is also set. Thus, the program can inquire with a branch on indicator instruction if there is an EOB for any line. If not, an unnecessary search through the Multiplexor Status Table for wordmarks can be avoided.

The processor is allowed to change a status character only after an EOB character is received. Any attempt to change a status character from Receive or Transmit to any other value before an EOB character is received will halt the system. The processor is permitted to set the status character to Control after an EOB is detected. It is then permissible for the processor to later change Control to either Receive-Idle or Transmit.

Running Address

The processor sets the running address to the core address where the 1448 is to deposit characters received or get characters for transmission. Once it is set, the 1448 will increment the running

address each time it deposits or transmits a character. Between scans, the running address will always point to one character beyond where the last operation occurred. Although the running address will not step past a groupmark/wordmark when receiving text, a character will never be stored in place of a groupmark/wordmark. Whenever a character might be stored in place of a groupmark/wordmark in the Receive status, the End-of-Storage bit is set in the status character and all subsequent characters received are lost. This loss can be prevented if the Early Warning feature (see below) is used. Encountering a groupmark (with or without wordmark) in the Transmit status will cause the system to Check Reset.

The running address, unlike the status character, may be reset at any time by the Scheduler or by an application program. A word of caution is necessary here. The 1448 Transmission Control Unit will not step the running address across a 4000-character boundary. Since such a boundary occurs in the half-track area (the 8K boundary; see the section "Core Allocation"), this fact should be well noted.

Interrupt

The 1448 will set a request interrupt condition in the processor whenever it receives the EOB character from any terminal. An interrupt is also requested when the 1448 buffers are full in Receive status, or empty in Transmit status. An interrupt cannot occur when a program is executing instructions that are less than five characters in length. The rationale for this is that the program might be chaining, and an interrupt would surely bring disaster. When an instruction greater than four characters is read into the A* and B* registers, the instruction counter is immediately changed, effecting a branch to position 181 (182 in some systems).

The processing of an interrupt is done by the Scheduler. Other programmers need not be concerned about the interrupt as their program will be completely unaware of it. Control is returned to the interrupted instruction with index registers, overflow indicator, and high-low-equal indicators restored.

It is possible to disable and enable the interrupts. All of the system subroutines (available for use by any program) disable the interrupts to prevent entries from more than one program at the same time. Interrupts are enabled when control is returned to the program calling them. When changing the status character for a terminal, it is imperative that application programs set the running address before the status character is changed.

Early Warning

One of the most important features of the 1448 Transmission Control Unit is the Early Warning Indicator. If a character being received from the terminal is stored in place of a groupmark (without a wordmark), the Early Warning Indicator goes on. In the ATS program, characters are received and transmitted from 100-character, dynamically allocated core blocks. When receiving from the terminal, the block is allowed to fill up, and then it is written to the 1311 Disk Storage. The Early Warning Indicator eliminates the wait that a terminal would experience when a new block is assigned. When the Early Warning Indicator goes on after a 1448 scan, the Scheduler locates the terminal requiring service, starts the characters flowing into a new block and

IBM 1440/1460 Administrative Terminal System
Terminal Code Translation

Terminal To Processor				Processor To Terminal	
Keyed	Received		Transmitted	Printed
1	1		1	1
2	2		2	2
3	3		3	3
4	8		4	5
5	4		5	7
6	6		6	6
7	5		7	8
8	7		8	4
9	=		9	0
0	9		0	Z
A	P		A	G
B	,		B	=
C	X		C	F
D	V		D	P
E	U		E	;
F	C		F	Q
G	A		G	,
H	Z		H	/
I	O		I	Y
J	+		J	M
K	W		K	.
L	Y		L	V
M	J		M	'
N	S		N	R
O	Q		O	I
P	D		P	A
Q	F		Q	O
R	N		R	S
S	R		S	N
T	@		T	U
U	T		U	E
V	L		V	D
W	$		W	K
X	/		X	C
Y	1		Y	L
Z	0		Z	H
.	K		.	-
,	G		,	B
;	E		+	J
'	M		@	T
/	H		/	X
1	-		$	W
=	B		=	9
-	.		-	1

TERMINAL SHIFT CHARACTERS (Where Different)

Downshift	Upshift
1	±
2	@
3	#
4	$
5	%
6	¢
7	&
8	*
9	(
0	)
-	_
=	+
'	°
;	:
/	?

FUNCTIONAL CODES

Name	Graphic	Bits
Upshift	GR	(001110)
Downshift	LS	(111110)
Tab	*L	(111101)
Return	*R	(101101)
Attention	WS	(011101)
Backspace	. ,	(101110)
Space		(000000)
Dummy	(	(011100)

Program-Generated Codes

*	Record Mark	RM (011010)
**	Groupmark	GM (111111)

* A RM in the input stream indicates a stop code; in the output stream it stops transmission to the 1448.

** A GM in the input stream indicates the end of valid text in that block.

Note: In ATS text stream there is a wordmark over the initial downshift of each unit and over every Carrier Return entered in the automatic mode.

Figure 11. Translation table for ATS terminal transmissions

queues the old block in the appropriate Arm list, to be written to disk storage.

The Scan

The 1448 scan instruction is given by the Scheduler. When this instruction is executed, the 1448 empties its buffers if receiving text, fetches two characters if transmitting, sets the status character for each terminal, and turns the EOB and Early Warning Indicators on if appropriate. The Scheduler must then react to any EOB or Early Warning Indicator set by the 1448.

THE TERMINALS

Transmission Code

Terminals for the 1440/1460 ATS have a typewriter mechanism similar to that of the IBM SELECTRIC® typewriter. The 6-bit codes generated by depressing the various letter and function keys (Shifts, Tabs, Backspaces, and Carrier Returns) on the terminal keyboard do not correspond to the 6-bit BCD codes normally used in IBM 1440/1460 Data Processing Systems. This means that ATS programs that are concerned with the content of the text stream generated by the terminal must identify these codes. Also, when data is transmitted to standard input/output units, it must be translated to standard BCD codes to get satisfactory results. However, all text stream data is stored within ATS in the terminal code. A two-way translation table is given in Figure 11.

Line Control

Line control codes used to convey control information between the multiplexor and the terminals are generated and removed automatically by the system. When the Attention or Carrier Return keys are depressed the appropriate character codes are transmitted, followed by line codes in character time. The terminal keyboard is locked. The terminal keyboard will remain locked until the multiplexor is set to Receive-Idle status for the terminal. This causes a Restore character to be transmitted to the terminal (by the 1448), which unlocks the terminal keyboard.

Output Timing Considerations

Since the fixed time-out feature of the 1448 is not used in ATS, there is no interlock when functional control characters are transmitted to the terminals. Because of this, text sent immediately behind a Tab, Index, or Carrier Return character will print while the SELECTRIC print element is in motion. To avoid this problem, a series of dummy characters (Upshifts or Downshifts, as appropriate) are put into the output stream following Tab, Carrier Return, or Index characters.

The number of these dummy characters is calculated so that the print element comes to rest just before the next printing character is transmitted. The Attention character (word separator) will index the platen if transmitted to the terminal. An Index character (Attention character) requires one dummy character. The formula for computing the number of dummy characters required for Tabs and Carrier Returns is the number of character positions to travel divided by ten, plus two, thus:

$$\text{Number of Dummy Characters} = \frac{\text{Number of Character Positions}}{10} + 2$$

THE DISK FILES

Disk operations in ATS are performed by the Scheduler and an I/O control program (IOCNX) with the help of a subroutine (SEEKS). These operations are made by requesting the system to perform them and are never attempted by application programs directly.

To obtain disk input/output, an application program must first set up the disk control field. This is a ten-character field immediately preceding the area where the operation takes place. The first character of the field is an asterisk (a right parenthesis or lozenge for 1301 addresses), as the alternate code feature is not used in the system. The next six positions are occupied by the sector address. This address is the actual address of the sector to be read or written. The next three positions are the sector count or the number of contiguous sectors that are to be read or written, beginning with the sector in the sector address portion of the field. A disk control field is shown in Figure 12.

Alternate Code	Sector Address	Sector Count	Area to be Read or Written
* or ¤	002999	001	

Figure 12. Example of a disk control field

An IBM 1316 Disk Pack used on the IBM 1311 Disk Storage Drive has ten recording surfaces. Each recording surface is composed of 100 concentric rings or tracks. These tracks are divided into 20 addressable sectors of 90 characters each (all disk operations in the system are in the Load Mode). There are ten access heads, vertically aligned, that move in unison on an arm. These heads read or write information on the tracks. Since the concentric tracks are also aligned vertically, when the arm is in position on any track, all of the tracks in vertical alignment (a "cylinder") are available. The 1311 can theoretically handle up to 200 contiguous sectors in a cylinder with one disk operation. Although the 1311 will switch automatically from track to track within a cylinder, it is important to note that automatic switching from cylinder to cylinder is impossible. Attempting this will cause a disk error condition.

Starting from the bottom of the outermost cylinder, the sectors are addressed beginning with 0 and ending with 199 at the top of the cylinder. The second cylinder is addressed from 200 to 399, and so on. Thus, the low address in each cylinder is addressed as an even multiple of 200.

The addresses in a multiple drive system continue in sequence from drive to drive. Thus, the highest sector address on drive 0 is 19999, and drive 1 commences with address 20000. The programmer need only be concerned with the address, however, as the hardware knows which drive to reference by any sector address.

A groupmark/wordmark in the area receiving or writing the information ends the operation. This groupmark/wordmark should coincide exactly with the end of the area as established by the sector count. In ATS, this is taken care of automatically by the Scheduler. However, application programs should never have groupmark/wordmarks within the area where the operation is to take place or else an error condition known as "wrong length record" will occur.

In ATS, all possible disk errors are checked. These errors are: wrong length record, address compare, access inoperable, and parity. If any error should arise, seven additional attempts are made to perform the operation. If it is still unsuccessful, the Scheduler sets an indicator that an application program can test when control is returned to it.

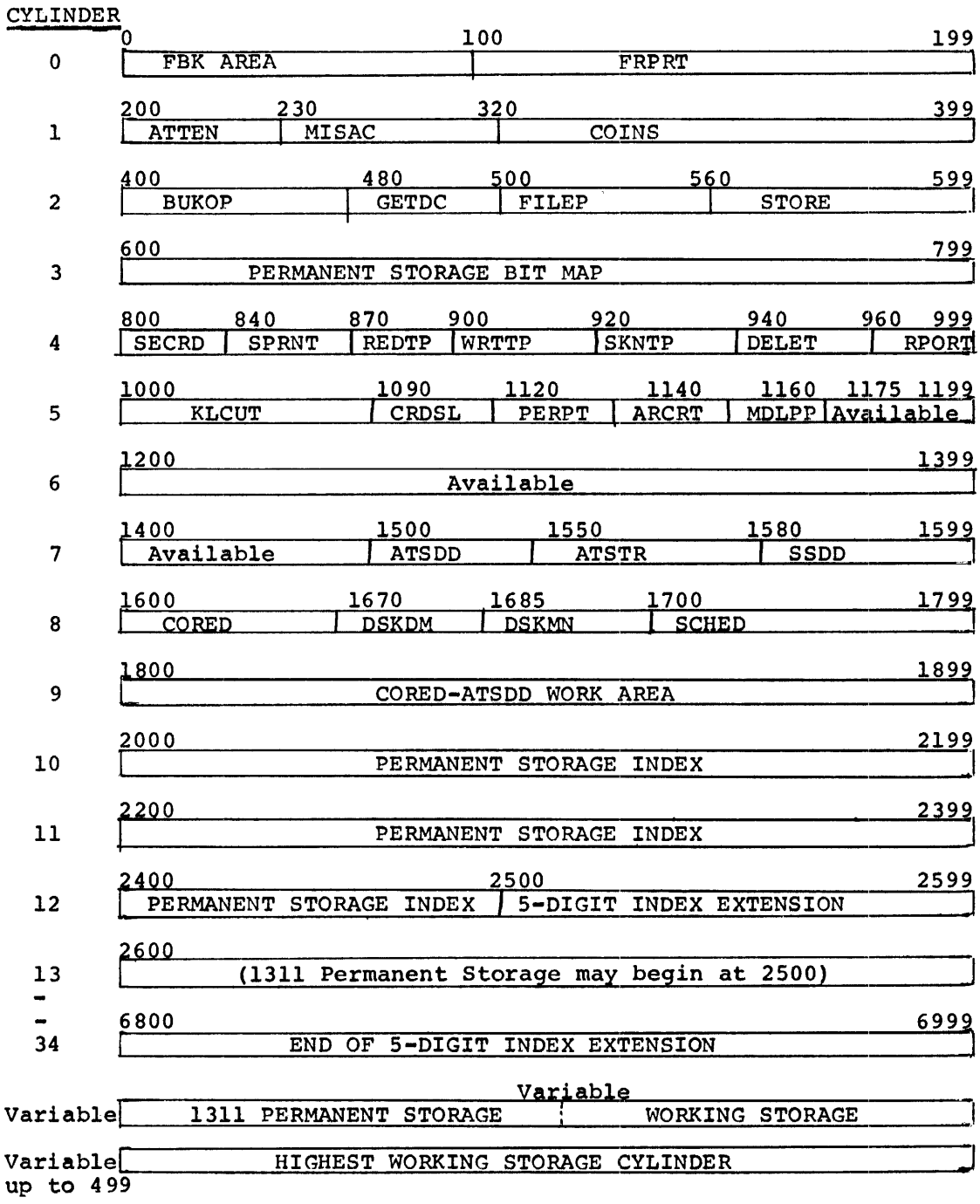


Figure 13. Disk storage allocation

DISK STORAGE ALLOCATION

In ATS, all disk operations are performed in the sector format and the Load Mode (with wordmarks). A disk pack has 20 sectors per track and 10 tracks per cylinder, or 200 sectors per cylinder. Since there are 100 cylinders per pack, there are 20,000 sectors per pack to be allocated. The minimum configuration for ATS is one disk drive (two are recommended), yielding a minimum capacity of 20,000 sectors. The first 20 cylinders or 4,000 sectors are allocated to programs and control information. All other sectors are used for storing text. Figure 13 contains a layout of disk storage.

Significant disk areas are:

1. First Block Area (sectors 0 through 39). The first 40 sectors are allocated to control information, one sector for each terminal. These sectors hold the tab stop settings; control information for corrections and insertions; control information for text formatting, and the pointer to the first block (disk) in a terminal chain. This pointer is termed the ANXB (Address of the Next Block), and it occupies the last five positions of each block. Here again, a pointer that is blank indicates that there are no sectors chained for a terminal.

2. Additional Terminal Area (sectors 40 through 79).

3. Peripheral Device Control (sectors 80 through 99).

4. Application Programs (sectors 100 through 599 and 800 through 1119). This area contains the main system application programs. An application program consists of a basic routine plus overlays if needed. The basic routine is 20 sectors long, while an overlay is 10 sectors. As many overlays as are necessary may be appended to an application program.

5. Bit Map (sectors 600 through 799). The bit map is a map of all of the half-tracks allocated for permanent storage and indicates which half-tracks are available.

6. Scheduler, System Subroutines, EUNIT and HSKPG (sectors 1700 through 1799). This 100-sector section is read into core in one operation by the bootstrap routine to initialize or restart the system. Sectors 1794 through 1799 of this section contain the ATS COMPOOL (see the section "COMPOOL").

7. Permanent Storage Index (sectors 2000 through 2500 if four-digit document numbers are used, or 2000 through 6999 if the five-digit document number option is used). The Permanent Storage Index holds the pointers for each dynamically allocated chain of Permanent Storage. Each entry in the index is associated with a number assigned to a document by a terminal. These numbers are between 1 and 9999, for the four-digit option or between 1 and 99999 for the five-digit option. The sector corresponding to document zero is used to contain the pointer for the next half-track available (NHTA).

8. Peripheral Overlay Library and User-Written Application Programs (sectors 1120 through 1599). Parts of this area may be used by peripheral programs that may be supplied with future versions of the system. Modification of the locations of programs stored in this area is discussed in the section "Writing Peripheral Programs".

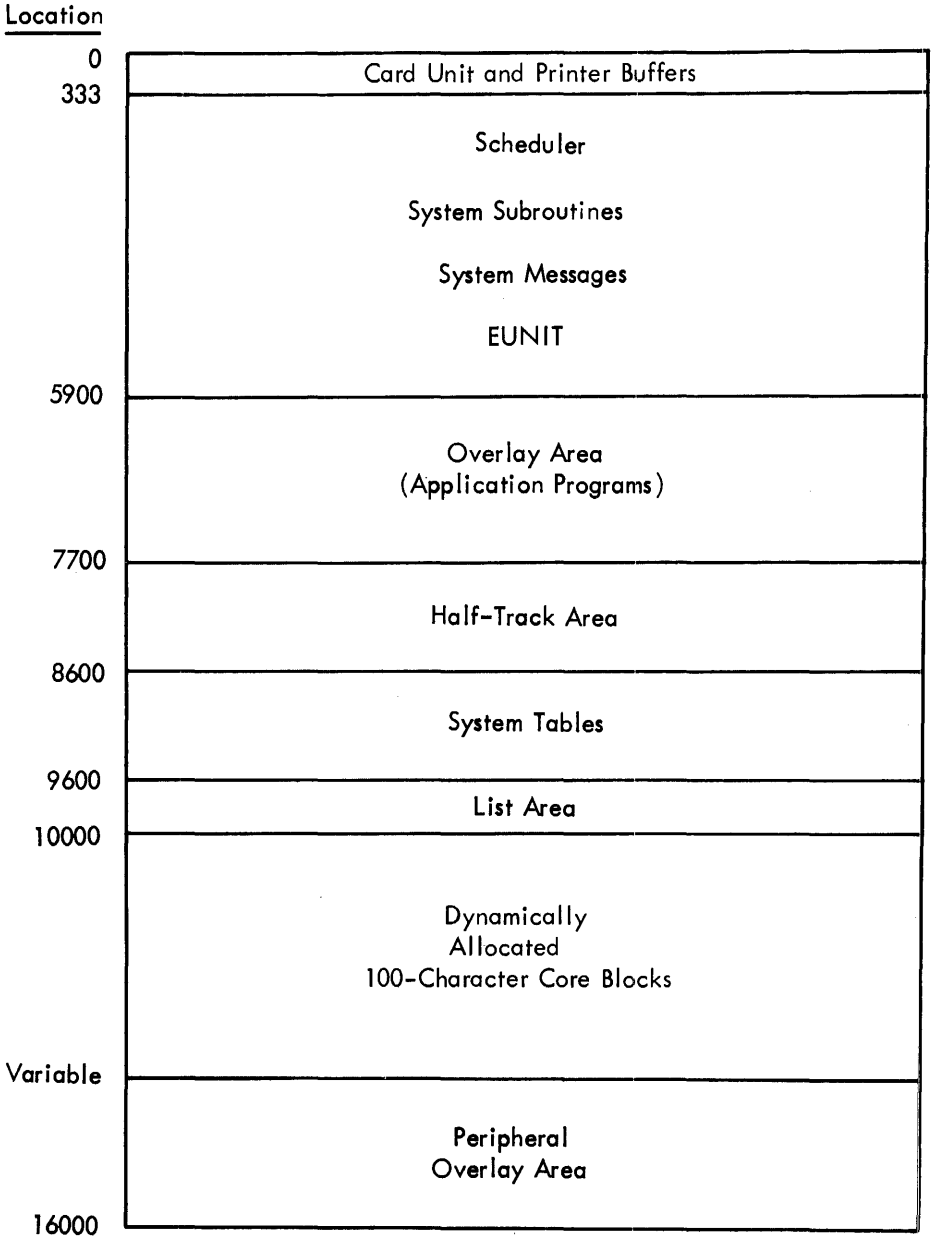


Figure 14. Core storage allocation

9. Permanent Storage (both upper and lower bounds are determined by the user). If an IBM 1301 Disk Storage is installed, all Permanent Storage is on the 1301. Permanent Storage consists of dynamically chained groups of ten contiguous sectors (half-tracks). Messages transmitted between terminals are handled similar to Permanent Storage and also occupy this area.

10. Working Storage (from address determined by user downward). Working Storage consists of single sectors dynamically allocated as explained in the section "Dynamic Disk Location".

CORE STORAGE ALLOCATION

The core storage requirements for 1440/1460 ATS are 16,000 positions. A clearer understanding of the organization of the system can be gained by understanding the physical allocation of core storage, which is shown in Figure 14.

Core storage may be divided into eight areas:

1. Card Buffers, 0 through 332. This area is used for peripheral and main system card operations.

2. Main System Programs and EUNIT, 333 through 5899. The Scheduler is the major program in the system. Its function is to control the flow of text to and from the terminals, give the reads or writes queued up for the disk file, and schedule work among the various functional (application) programs.

In addition to the Scheduler, the input/output control program (IOCNX) is also in this area. IOCNX is responsible for queuing input/output requests. Also in this area are the system subroutines that may be used by application programs. Among the more important subroutines are the following:

Set New Job (SNJBX) is responsible for the queue of new jobs for application programs.

New Core Block (NCBKX), Head of Free Storage (HDFSX), and Foot of Free Storage (FTFSX) are responsible for the maintenance of the free storage chain of 100-character core blocks.

Next Disk Block (NDBKX) and Purge Disk Block (PDBKX) are responsible for the pool of availablle disk sector addresses used for Working Storage.

Also in this area, in addition to the main system programs, is a small application program called End of Unit (EUNIT), which deletes nonprinting characters preceding the Carrier Return Character at the end of a unit. EUNIT is the only application program that is resident in core while the system is operating.

3. Overlay Area (application programs), 5900 through 7699. All of the application programs, with the exception of EUNIT described above, are resident on the disk file. These programs process specific requests from the terminals. When a request is made, SNJBX places a program request in the New Job queue, and when convenient for the system, the appropriate program is read into the Overlay Area and given control. The application programs in the system are:

Attention (ATTEN) handles the erasing and deleting functions in the system. ATTEN also serves as a preprocessor to other

application programs and processes system errors or transmission errors.

Bulk Move and Erase (BUKOP) processes the moving and deletion of text in Working Storage.

Corrections and Insertions (COINS) processes the calling out of lines and units for correction, and the insertion of lines or units into previously keyed text.

Card to SELECTRIC (CRDSL) creates documents in ATS format from card or unblocked tape input.

Delete (DELET) updates the bit map to indicate available half-tracks released by deleted documents.

Files Preprocessor (FILEP) does the initial processing of requests pertaining to Permanent Storage: Store, Get, Delete, Transmit, Message, and Permanent Storage Tape.

Format and Print (FRPRT) generates a formatted text stream for transmission to the terminals.

Get Document (GETDC) retrieves documents from Permanent Storage.

Housekeeping (HSKPG) is the exception that does not respond to requests from the terminals. It prepares core storage and initializes the cycling of the system when the system is loaded. It also restarts the system in a system failure situation. Unlike all other overlay programs, HSKPG is not initialized through the New Job queue. Instead, a bootstrap routine is used that brings both the main system routines and HSKPG into core from the disk file and gives control to HSKPG.

Calculate (KLCUT) is an algebraic interpreter that provides arithmetic functions and formula solutions.

Miscellaneous Attention Actions (MISAC) is responsible for setting the terminals to active status, mode setting (that is, Automatic or Uncontrolled Modes), returning the terminal to inactive status, and processing the "next number" request and format control requests (that is, ATTN ! and + actions). In addition, MISAC is a preprocessor for FRPRT.

Permanent Storage Print (PERPT) prints a listing of the contents of Permanent Storage on the online printer.

Read Tape (REDTP) reads documents from the Permanent Storage Tape into Permanent Storage on disk.

SELECTRIC to Card (SECRD) translates text entered in card format from terminal code to BCD and produces punched cards, magnetic tape, or online printer output.

Scan Tape (SKNTP) searches the Permanent Storage Tape for a document if a read is requested, or to the end of the file if a write is requested.

Special Print (SPRNT) prints formatted output on the uppercase and lowercase chain printer, if present, or writes it on tape in print image format.

Store (STORE) stores documents in Permanent Storage.

Storage Report (RPORT) prints a listing of the contents of Permanent Storage on a terminal.

Write Tape (WRTPP) writes documents from Permanent Storage on disk to the Permanent Storage Tape.

In addition to these application programs, special purpose application programs may be added to the system.

4. Half-Track Area, 7700 through 8599. This area is named for the ten-sector elements or half-tracks that Permanent Storage programs manipulate in this area. In addition to this use of the area, most application programs use it to hold their own respective overlays. Some programs have as many as six overlays. Core restrictions force this second-level overlay technique.

5. System Tables, 8600 through 9599. This area contains fixed tables and indicators and is divided into six sections. They are:

a. System Status Table. This is a general area that contains fixed registers and indicators. "Miscellaneous" would be an appropriate synonym for the table. Among the more important items in this area are the pointers to the queues in the list area, and the free storage chain of core blocks. It is described in more detail in the section "System Status Table".

b. Program Status Table (PST). This is a table of the location of every application program. It is described in more detail in the section "Program Status Table".

c. Character Status Table (CST). This table is used in conjunction with the Program Status Table to identify the program needed from characters keyed from the terminals. It is described in more detail in the section "Character Status Table".

d. Arm Table (ARM). This table contains five entries, one for each possible disk drive connected to the system. The ARM entries contain the status of the drive and the pointer to the top I/O request for that arm in the list area. It is described in more detail in the section "Arm Table".

e. Terminal Status Table (TST). This table contains one entry for each possible terminal in the system. Each entry consists of a four-character "Input Buffer" that is used to receive characters when the terminal is in the inactive status. The mode request that initializes the terminal is sensed in this buffer. When the terminal is active, the input buffer is used to keep the line count. It is described in more detail in the section "Terminal Status Table".

f. Multiplexor Status Table (MST). This table is required for communication with the 1448 Transmission Control Unit. There is an entry in the table for each line attached to the system. Each entry contains the status for that line (Receive, Transmit, or Control) and the address where characters are to be transmitted or received. In addition, there are a number of control indicators used for various purposes. It is described in more detail in the section "Multiplexor Status Table".

6. List Area, 9600 through 9999. This area contains the dynamic list entries that make up the New Job, Work in Progress, Arm, and Available queues. Application programs process their text by reading it into the core blocks and manipulating it there. The number of blocks needed is determined by the number of terminals.

7. Block Area, 10000 through XXXXX (15899 maximum). This area is composed of the 100-character, dynamically allocated core blocks used to buffer text between the terminals and the disk files. Application programs process their text by reading it into the core blocks and manipulating it there. The number of blocks needed is determined by the number of terminals.

8. Peripheral Overlay Area. When all possible core blocks are not required for terminals, the highest positions in core may be used as an overlay area for programs doing peripheral operations. These peripheral operations can include card to tape, tape to printer, or even small unit record type programs that involve some computing between card reading and printing. Card punching introduces system delays and is not recommended while terminals are in operation.

SYSTEM TABLES

This section describes, in detail, the contents of tables in the Compool or common parameter area in the ATS program. The Autocoder labels of the fields described are given in capital letters.

SYSTEM STATUS TABLE

The System Status Table is not really a table in the strict sense of the word, but a collection of indicators and registers grouped together for convenience. They are:

LPSA (five positions). Lowest Permanent Storage Address is the disk address specified by the user of the lower boundary of Permanent Storage (low-order zero is understood).

NHTP (five positions). Next Half-Track Possibly is the disk address (low-order zero understood) of the next half-track to be tested for availability in the bit map. It is used to eliminate redundant searches when storing a document.

HPSA (five positions). Highest Permanent Storage Address is the disk address specified by the user of the upper boundary of Permanent Storage (low-order zero understood).

NWSA (five positions). Next Working Storage Sector Address is the disk address (high-order zero understood) of the next sector to be used for Working Storage. At the start of a run it is set equal HWSA. It is documented by NDBKX to increase the pool of Working Storage Addresses.

LWSA (five positions). Lowest Working Storage Sector Allowed is the user-specified lower bound of Working Storage (high-order zero understood).

HWSA (five positions). Highest Working Storage Sector Allowed is the user-specified upper bound of Working Storage (high-order zero understood).

NTRN (five positions, equated to HWSA). Next Tape Record contains the tape document number of the next document to be written to the archive tape.

OTRM (ten positions, wordmark bit). Output Terminal is a table of peripheral output devices attached to a system.

TAPE (ten positions, equated to OTRM, character bits only). TAPE is a table of mounted archive tape reels. Each character position represents tape drive numbers zero to nine. When an archive tape is mounted, the alphabetic reel designator (in SELECTRIC code) is placed in the character position corresponding to the drive number.

TT00 (one position, character bits only). This character, 1301 indicator, contains the number of 1301 modules available. If no 1301 is attached the position is blank.

FDDN (one position, wordmark bit, equated to TT00). Five-digit Document Numbers indicate whether the five-digit option is allowed. (Wordmark indicates five-digit numbers.)

WRUT (one position). Write Unit contains the number of the tape drive on which the archive tapes are written.

MACH (one position). MACH describes the computer on which the system is operating. A 6 indicates the 1460, a 4 the 1440. A wordmark indicates the translate instruction is installed on the machine; no wordmark indicates the translate instruction is not installed. The system will adjust its instructions automatically to conform to the appropriate system configuration.

ATLI (two positions). Number of Attached Lines contains the number of lines attached to the 1448 for purposes of setting the groupmark/wordmark in the Multiplexor Status Table.

NAVB (two positions). Number of Available Blocks is the count of core blocks in free storage at any one time.

ADPR (three positions). This field contains the address of the first character of the peripheral program. It is set by MISAC when a new peripheral program is called in by terminal 0.

DATE (six positions). This field is set by the HSKPG program with the date entered by the computer operator at the 1447 Console. It is used by STORE to date documents when they are entered into Permanent Storage.

SCLK (two positions). Simulated Clock is used by the Scheduler to keep track of the amount of time available for disk operations. It is set to 99 (100 milliseconds) each time the Scheduler is entered, and is decremented an appropriate amount for every function that is performed. It is checked just before disk operations are attempted. Forty milliseconds are allowed for each disk operation, making two operations possible if no other work was done.

MODE (one position). A wordmark on the MODE switch indicates that the Scheduler is operating; no wordmark indicates that an application program is operating. This indicator is used by some of the system subroutines and IOCNX to perform certain operations that are done only for the Scheduler. In addition, some of the subroutines will trick other subroutines into thinking they are working for Scheduler to get additional operations performed. The 1- and 2-bits of MODE are used as request bits for servicing the peripheral program: a 1-bit means branch to the address in ADPR after every 1448 scan; a 2-bit means branch whenever 100ms of available computer time exists.

OVBZ (one position). A wordmark in the Overlay Busy indicator is set by the Scheduler whenever a program is in, or is in the process of being read into, the overlay area. With the switch on, the Scheduler will not attempt to read in another program until the current one is finished with its work. The Scheduler turns the switch off when an application program returns control, indicating that its work is done.

PROV (three positions). Program in Overlay is set to the Program Status Table Location (PST number) for the program currently operating in the overlay area (see the section "Program Status Table"). The rationale for this indicator is that two requests might appear in the NJB list for the same program, making a second reading of the program unnecessary. The Scheduler checks the PROV indicator for every NJB entry that it processes.

HDCC (five positions). Head of Clear Chain is the disk address (high-order zero understood) of the top sector of a chain of sectors to be purged. It is set to blanks if no sectors are to be purged.

HDPT (three positions). The Head Pointer points to the top entry of the free storage chain of core blocks.

FTPT (three positions). The Foot Pointer points to the foot of the free storage chain of core blocks.

NDPT (three positions). The Next Disk Pointer points to the top currently available entry in the pool of sector addresses used for Working Storage on disk.

NBDA (one position). Number of Blocks of Disk Addresses is the number of core blocks being used to hold the pool of Working Storage disk addresses.

NJPT (three positions). The New Job Pointer points to the top entry in the New Job list.

WPPT (three positions). The Work in Progress Pointer points to the top entry of completed disk input/output list.

AVPT (three positions). The Available Pointer points to the top available (free storage) entry in the list area.

The above indicators should not ordinarily be read or changed by an application program. Control should be given instead to the particular subroutine that is responsible for the indicator, such as SNJBX for the New Job queue.

PROGRAM STATUS TABLE

The Program Status Table (PST) is a contiguous list of every application program that can be operated through the New Job queue. The elements of this table are three characters long. If the program is resident in core, such as EUNIT, the three characters contain the core address of the first instruction. However, as is usually the case, if the program is resident on the disk pack, the three characters reference the disk address with one low-order and two high-order zeros understood. Since the overlay area holds 20 sectors, the sector count is understood to be 20 regardless of program length. If a program exceeds 20 sectors, it must read in its own 10-sector overlays into the half-track area.

Two indicators are used in the PST. A wordmark in the high-order position indicates that the entry is a core address and the program is resident in core. No wordmark indicates a disk address. This indicator is called Program Location. Another indicator is Program Busy (BUSY). Obviously, it is possible for two requests for the same program to be in the New Job queue at the same time. In such a situation, unless precautions were taken, a program could be processing two requests at once. To prevent this from occurring, the BUSY-bit

(wordmark in the tens position of the entry) is turned on whenever a program is operating. Other requests for the program wait in the queue until it is available.

In the Autocoder language, an example of an entry for a core program would appear as follows:

```
PEUNIT  EQU  0
        DCW  +EUNIT
```

The label for the entry is prefaced by a P (meaning PST position for EUNIT) to avoid presenting label ambiguity to the assembler. Equating PEUNIT to zero is in reality equating the label to the high-order position of the DCW statement following (assuming the table began at zero). Since each entry is three characters long, it will have its high-order position in increments of three from the beginning of the table. Thus PEUNIT is the first entry in the table, as the first entry is equated to zero. +EUNIT means the high-order address of the field with that label, in this case an instruction. The DCW statement contains the core address in this case.

A disk program entry would appear like this:

```
PATTEN  EQU  3
    DC  @020@
```

Note: In this example, the entry is a DC statement that is assembled as a three-character field without a wordmark. The @ signs preceding and following the 020 indicate a literal. Attention is the second entry in the table and is, therefore, equated to three. In this case, the DC statement implies a disk address.

The order of the entries specifies program priority. Thus, PATTEN equated to three has a higher priority than PMISAC, which is equated to six.

The number that the PST label is equated to is used in conjunction with the Character Status Table (see below) to identify a program with keyed terminal characters. Note the ease with which a program can be added to the system. An entry is made in the PST; another is made in the Character Status Table; the program is read onto the disk; and a new function is added to the system.

CHARACTER STATUS TABLE

The Character Status Table (CST) consists of 64 contiguous entries of two characters each. These entries are arranged in order, according to the 64-bit combinations in the BCD character, beginning with a blank and continuing until all bits are on. For example, entry 1 corresponds to a blank character; entry 2 corresponds to a 1-bit; entry 3 to a 2-bit; etc.

The two-character entries contain the PST location for the program that corresponds to its appropriate terminal code character. If no program corresponds, the characters are blank. The beginning of the CST in Autocoder language appears as follows:

```
DC  @ @
DC  @09@
DC  @09@
```

One of the responsibilities of the Attention program (ATTEN) is to interpret Attention actions. This interpretation is accomplished first by identifying the bit code in the character following an Attention character and then by finding the CST entry that corresponds to that character. If the entry is blank, the Attention action is illegal. Otherwise, a New Job entry is set for the program designated by the CST entry.

To enter a new program into the system, the PST location must be inserted into the CST entry corresponding to the character that will be used to request the function.

ARM TABLE

The ARM Table consists of five entries of 20 characters each, one for each possible arm (Disk Drive Access Arm) in the system. An ARM entry is shown in Figure 15.

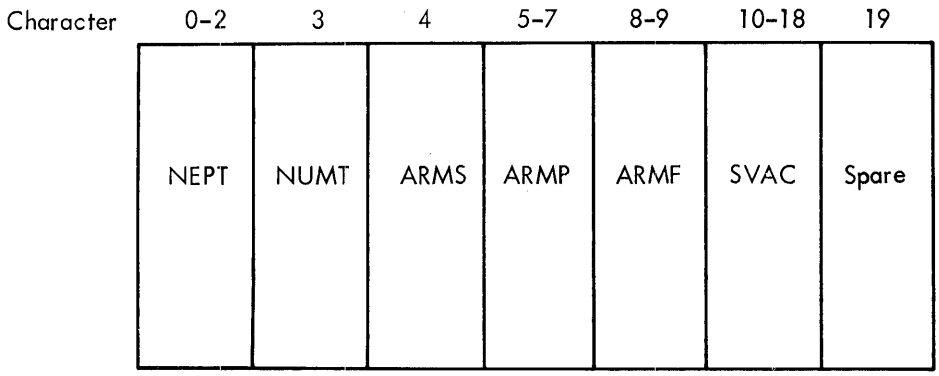


Figure 15. ARM Table entry

The fields in an entry are:

NEPT (three characters, positions 0 through 2). The Next Entry Pointer points to the top entry for the arm corresponding to this table entry (see "List Entries" below).

NUMT (one character, position 3). It is possible, because of a failure of a disk drive or a program specifying an incorrect sector address, for a disk error condition to arise. Should this occur, the Scheduler will reseek and retry the operation seven times. The count of tries is kept in the Number of Tries position of the appropriate ARM entry.

ARMS (one character, position 4). The ARM Status character indicates whether the arm is seeking (wordmark) and the type of operation that is waiting. A 1 indicates a read waiting; a 2 indicates a write waiting; and a blank indicates nothing is waiting.

ARMP (three characters, positions 5 through 7). Since the direct seek feature is used, it is important to know the Arm Position in order to make the necessary computation. The ARMP register holds the hundreds, thousands, and ten thousands position of the current sector address, rounded down to an even cylinder number.

ARMF (two characters, positions 8 and 9). Arm Failures are used to keep a record of the number of I/O requests that required at least one reseek. This indicator is a diagnostic indicator. It makes it possible to request maintenance before a drive fails altogether.

SVAC (nine characters, positions 10 through 18). This portion of the ARM entry is called Saved Address and Count. A disk control field is kept in SVAC so that it may be restored immediately after a disk operation.

SPARE (one character, position 19). This character is reserved for possible expansion of the system.

All 1311 input/output requests are placed in the arm queue determined by the ten-thousands digit of the disk control field; 0 and 1 to arm queue 0, 2 and 3 to arm queue 1, 4 and 5 to arm queue 2, 6 and 7 to arm queue 3, and 8 and 9 to arm queue 4. This scheme is also employed for 1301 input/output requests except that all those which would go to arm queues 0 and 1 are placed in arm queues 3 and 4 respectively. Since most systems utilizing 1301 Disk Storage have two 1311 Disk Storage Drives this scheme places all 1311 input/output requests in arm queues 0 and 1 and all 1301 input/output requests in arm queues 2, 3 and 4.

TERMINAL STATUS TABLE

The Terminal Status Table (TST) consists of one to 40 entries of five characters each, one for every attached terminal in the system. The last character in every entry is a groupmark/wordmark.

Inactive Status

As far as the 1448 and the processor are concerned, a terminal is never offline. The so-called inactive status is an invention that is completely contained in the program. Because of this, the running address for every inactive line must be set. When a terminal is inactive, its running address is set to the first character of its respective TST entry. In this situation, the TST entry is termed the Input Buffer (INBF). As characters are entered into the system, the INBF fills up until the running address is pointing to the groupmark/wordmark. All subsequent characters entered are lost. When the Carrier Return is transmitted from the terminal, the Scheduler resets the running address to the first INBF character again. However, a terminal may request to go online and, because of this, the Scheduler will check the INBF for the exact sequence ATTN a CR or ATTN u CR. If the sequence is sensed, all succeeding text will be entered into a core block, and the terminal will be considered to be active. If not, the running address is reset to the first INBF character for that terminal.

Active Status

A terminal is placed in active status when the initial ATTN a or ATTN u action is taken when in the offline status. When a terminal is active, its TST entry is used to count the number of lines or units

entered. The ATTN a or u CR is considered the first line, and all subsequent lines or units cause the number in the TST entry (now called LINE) to be incremented. This number is used with the line count stored in each block to locate specific lines for correction or insertion.

MULTIPLEXOR STATUS TABLE

The Multiplexor Status Table (MST) is the communication between the 1448 and the processor. In addition, a number of indicators that are very important to the program occupy this area. The table itself consists of one to 40 entries of ten characters each, or one entry for each line attached to the 1448. A groupmark/wordmark is placed immediately after the entry corresponding to the last attached line. An MST entry is shown in Figure 16.

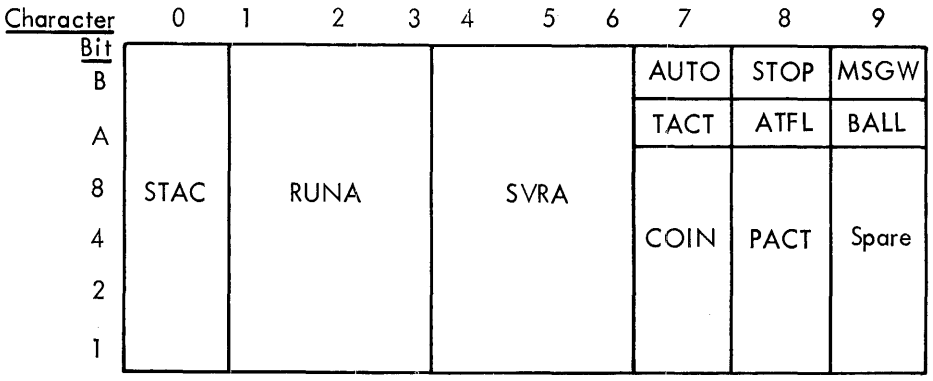


Figure 16. Multiplexor Status Table (MST) entry

The fields in an entry are:

STAC (one character, position 0). This character indicates the status for the line (see the section "The Multiplexor" for the possible status characters).

RUNA (three characters, positions 1 to 3). RUNA points to where the 1448 may deposit or pick up characters.

SVRA (three characters, positions 4 to 6). SVRA is the mnemonic for Saved Running Address. These positions are used for temporary storage of a running address that will replace the current one. An example would be to save the running address to a core block while the current running address points to a system message in the message area.

AUTO (B-bit, position 7). If the AUTO-bit is one, the terminal is in the Automatic Mode. Off indicates the Uncontrolled Mode.

TACT (A-bit, position 7). This bit indicates whether or not a terminal is in active status. If the TACT-bit is on, the terminal is active.

COIN (numeric, position 7). This indicator gets its name from the COINS (Corrections and Insertions) program, which sets the field to 1 if a correction is in progress, or to a 2 if an insertion is in progress. The FILEP program will set it to 4 if a document ident is being entered, or to 8 if a document is in the process of being deleted from Permanent Storage. The BUKOP program will set COIN to 3 if a bulk move has been requested, or to 9 in the case of a bulk erase. The value of this indicator is immediately apparent in view of the fact that an application program cannot remain in the overlay area while it is waiting for operator response, as this would cut off all functions to all other terminals.

STOP (B-bit, position 8). This bit is set by certain programs when they have finished generating output. The Scheduler, upon sensing the STOP-bit, will reactivate the program in PACT after the Attention character is received.

PACT (numeric, position 8). PACT is the mnemonic for Program Activation Required. PACT is used in conjuncion with STOP to reactivate certain programs.

ATFL (A-bit, position 8). The Attention Flag is set by the Scheduler whenever the Attention key is depressed, and indicates that an Attention action is being entered and must be interpreted and processed when the Carrier Return is received.

MSGW (B-bit, position 9). This bit, Message Waiting, is set by the STORE program to indicate that a message has been queued up for that terminal. Application programs, sensing this bit, insert the characters (MSG) in certain system messages transmitted to the terminal.

BALL (A-bit, position 9). This bit indicates the type of SELECTRIC printing element on the terminal. If the bit is on, a manifold or similar printing element is indicated, and the numeric one is used in all system-generated messages sent to the terminal. If the bit is off, the lowercase L is used for the numeral one.

SPARE (numeric, position 9). These bits are reserved for possible expansion of the system.

LIST ENTRIES

Availability Entry

The list area is organized in the manner described at the end of the section "List Processing". An entry in the free storage chain of list entries contains two fields:

Not Referenced (seven characters, positions 0 through 6). This area contains whatever was entered into it previously. The entries are never cleared.

Pointer (three characters, positions 7 through 9). This pointer points to the next entry in the queue. A blank pointer indicates the end of the list.

NJB Entry

A program request entered in the queue contains four fields:

Not Referenced (two characters, positions 0 and 1).

Terminal Number (two characters, positions 2 and 3). This is the 1448 line number of the terminal requesting service.

PST Number (three characters, positions 4 through 6). This number is relative to location of the program entry in the PST.

Pointer (three characters, positions 7 through 9). The pointer points to the next entry in the queue; a blank pointer denotes the end of the list. List entries are one-way chained, FIFO, and priority sorted.

ARM Entry

An entry queued up for a particular disk arm appears as shown in Figure 17.

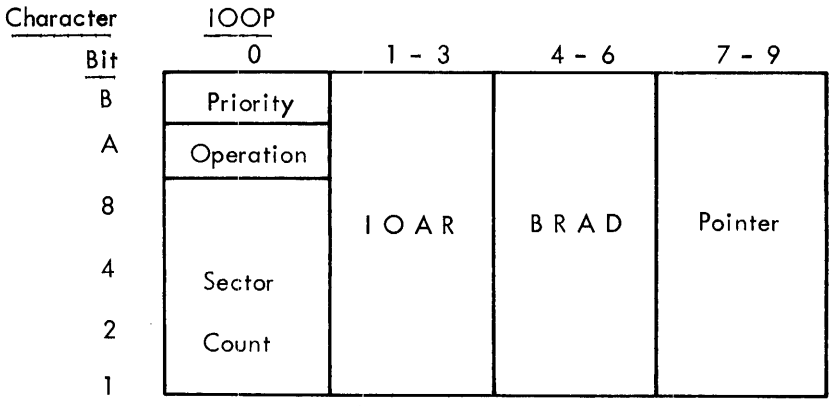


Figure 17. ARM and WPA list entry

The entry contains 4 fields:

IOOP (one character, position 0). The Input/Output Operation character contains certain coded information. The B-bit indicates priority; ON is high priority, OFF is low priority. The A-bit indicates the type of operation; ON is a write, OFF is a read. The numeric portion of the IOOP character indicates the sector count; a number from one to nine is a sector count from one to nine; a zero is a sector count of ten; and a blank indicates that the count is already set in the disk control field.

IOAR (three characters, positions 1 through 3). The Input/Output Area is the address of the high-order positions of the disk control field (the asterisk or lozenge).

BRAD (three characters, positions 4 through 6). This field is the Branch Address where control is to be returned when the input/output operation is complete.

Pointer (three characters, positions 7 through 9). The pointer points to the next entry in the queue, with a blank pointer indicating the end of the chain. ARM entries are FIFO and priority sorted.

WPA Entry

The entries in the queue of completed I/O (Work in Progress) are simply ARM entries rechained in the WPA queue. In other words, ARM and WPA entries have exactly the same information. They are simply queued behind different pointers. WPA entries are FIFO.

TEXT STREAM PROCESSING

This section describes the internal handling and processing of text by the ATS. The format in which text is processed by ATS is called the ATS text stream. This section must be understood by anyone wishing to write a program that will process the ATS text stream.

WORKING STORAGE

In the offline condition, all the characters keyed by a terminal are deposited in the four-character buffer (INBF) in the Terminal Status Table for that line. If a request to go online (ATTN a CR, or ATTN u CR) is sensed in the buffer, a core block is obtained from the free storage list, and the running address in the Multiplexor Status Table is set so additional key strokes are captured in the core block. When 74 or 75 characters have been accumulated, an entry for the block is placed in the appropriate ARM queue to be written to the disk, and another block is obtained from free storage for the next 75 characters. Every block written to the disk has the disk address of the block that will follow and the block that preceded it. These pointers are called the Next Block Address (NXBA) and the Last Block Address (LSBA), respectively. A diagram of a Working Storage text stream may be found in Figure 18.

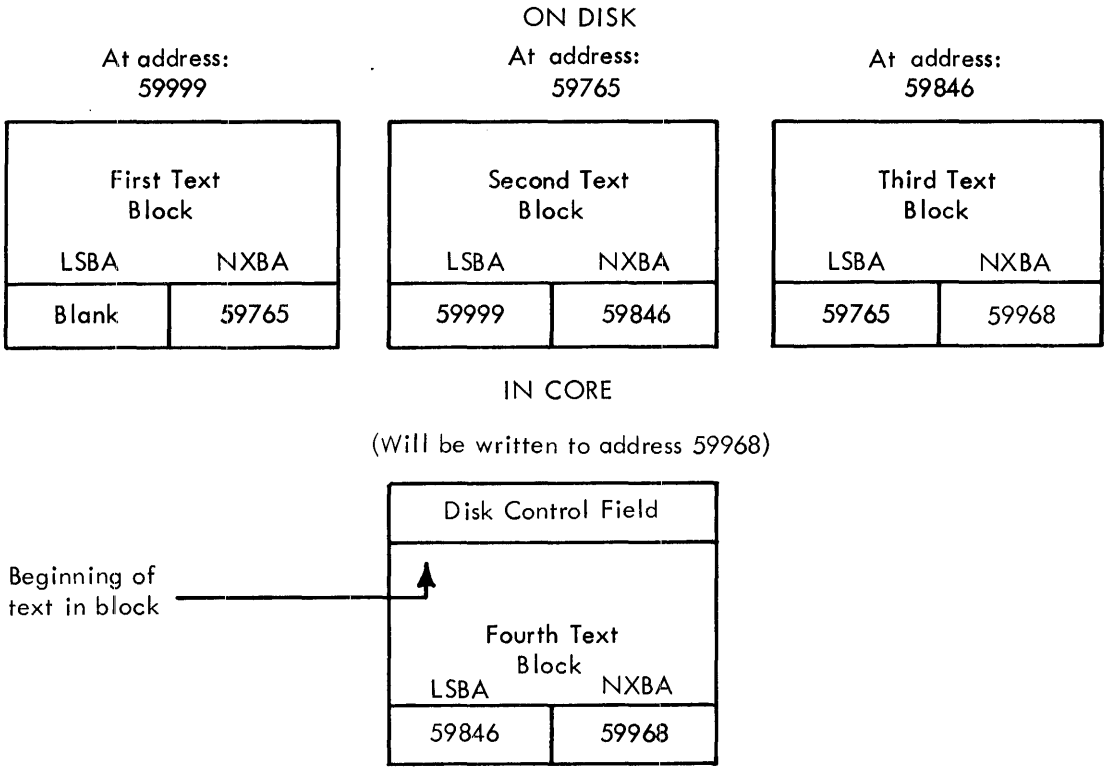


Figure 18. Working Storage text stream

Unlike most of the dynamic allocation schemes in ATS, the end of the stream in Working Storage is not indicated by a blank pointer, but by the pointer pointing to itself. In Figure 18, the last block (in core) has in the forward pointer (NXBA) the address of where the block is to be written and thus points to the block, itself, on disk. The backward pointer (LSBA) is blank in the first block. The disk control field for the block in core storage is not set until it is placed in the ARM queue. At that time, the address in the NXBA is loaded into the disk control field for the block, and a new address, obtained from the available pool, is loaded into the NXBA. This address will become the address of the following Working Storage block. When a second core block is obtained and the address of the old block being written is loaded into the LSBA, the new address is loaded into the NXBA. The new block is then ready to receive text.

If the terminal made a program request, the Attention and Control characters would appear in the Working Storage block in core storage as they are received by the system. There is no guarantee, however, that the entire Attention line is in a single block. The program processing the request would always find the end of the line in the core storage block only by finding the Carrier Return character. The core address of the Carrier Return can be located by the running address for the terminal in the Multiplexor Status Table. Generally, the application program will scan to the beginning of the Attention line and then interpret the line.

As mentioned previously, output streams are program-generated in Working Storage blocks. Once generated, the first block of the stream is read into core storage, and transmission to the terminal is started beginning with the first text character in the block. Transmit blocks are chained in the same manner as input blocks with an LSBA and NXBA. Thus, when the block in core is almost exhausted, the address of the following block is obtained from the NXBA and is placed in a core block, and a read request is placed in the appropriate ARM queue. Normally, the next block will be in core by the time the current transmit block is exhausted. If not, transmission will wait until the block is in core. At this time, the running address is changed to begin transmission from the new block, the old core block is returned to the free storage chain and its disk address to the available pool.

The first block is important to both the transmit and the input text streams. It is always the sector with the same address as the terminal. For example, the first block for terminal 25 is sector 25. This sector contains static control information such as the address of the first text block, tab settings, lines per page, etc. When a terminal is receiving a program-generated text stream, the first block contains the disk address of the last block in the text stream that was previously being transmitted from the terminal. This parameter is necessary to restore the terminal to the Receive Mode from the Transmit Mode at the end of text stream transmission to the terminal.

CORE BLOCKS

The dynamically allocated, 100-character core blocks begin at core address 10000 and continue to the peripheral overlay area, if it is used, or to 15899. The first position of a core block is always at an even hundreds position. For example, a core block might have addresses of 10200 through 10299. Core blocks are essentially buffers that hold text between the terminals and the disk file. Text being received from a terminal accumulates in a core block before being

written to the disk pack. Text being transmitted is read or generated in these blocks.

In addition, application programs work with raw text and generate output streams in these blocks.

Block in Free Storage

All core blocks are initially located in a free storage chain. Free storage is two-way chained, with the HDPT (Head Pointer) and the FTPT (Foot Pointer) pointing to the ends of the chain (see Figure 19).

A block in free storage contains two control fields:

BKCB (three characters, positions 1 through 3). The Backward Core Block is the backward chaining address in the block and points toward the top of the chain. The first block in the chain contains a blank BKCB.

FWCB (three characters, positions 4 through 6). The Forward Core Block points toward the bottom of the chain. The last block in the chain contains a blank FWCB.

Block in Receive Status

A core block in Receive status is accumulating text from a terminal. The first ten characters comprise the disk control field; the last ten characters are the disk chaining addresses; and five characters are used for control, leaving 75 characters per block for text (see Figure 20).

Since a block is written in the Load Mode, the 90 characters following the disk control field are written out. A block in Receive status has several important locations and fields:

Asterisk (one character, position 0). The first character of the disk control field contains an asterisk, since the alternate code is not used.

SADD (six characters, positions 1 through 6). The Sector Address portion of the disk control field contains the full six-character address of the sector to which the block is to be written.

SCNT (three characters, positions 7 through 9). The Sector Count is always 001 for a core block.

FTXC (one character, position 10). The First Text Character label identifies the address where the text begins in the block.

LTXC (one character, position 84). The Last Text Character label identifies the location of the last valid text character.

Groupmarks (two characters, positions 83 and 84). The groupmarks at LTXC-1 and LTXC turn on the Early Warning Indicator when characters from the terminal are stored over them.

Groupmark/Wordmark (one character, position 85). The groupmark/wordmark will stop the reception of text and turn on the End-of-Storage bit in the status character if the multiplexor attempts to store a character over it.

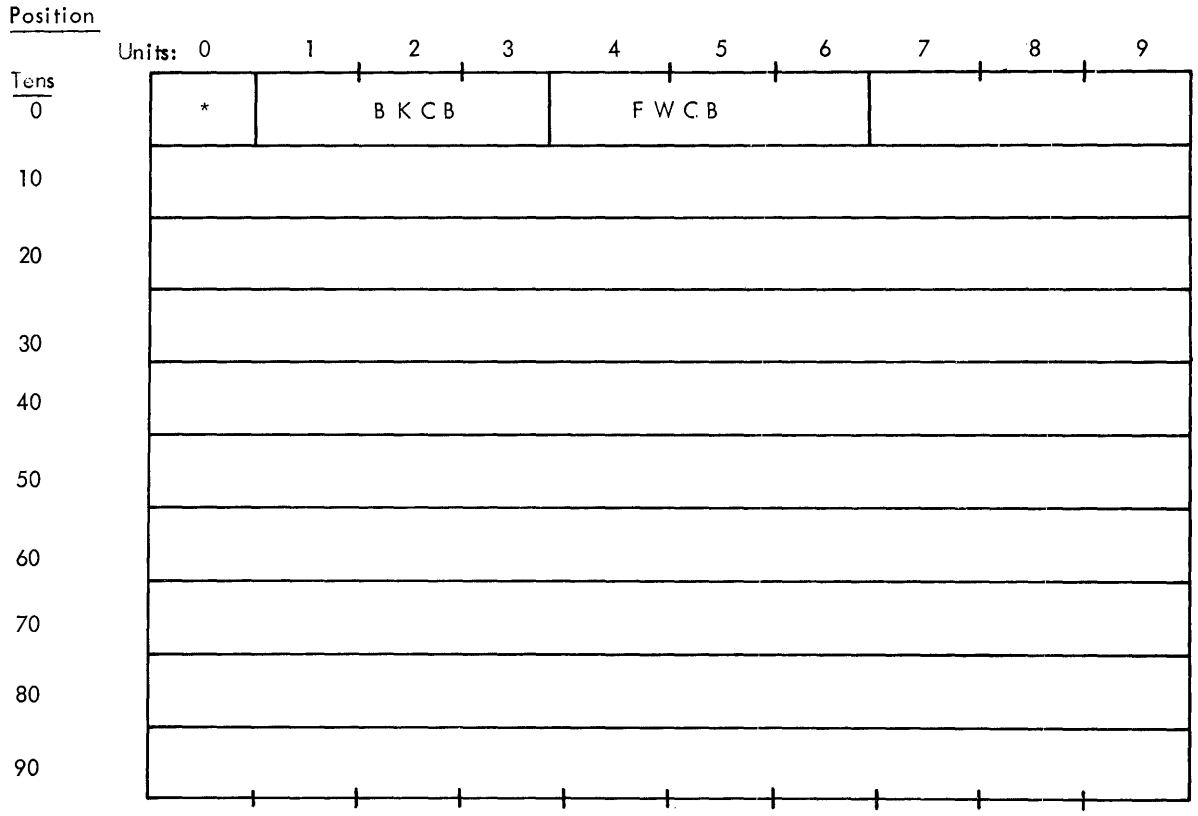


Figure 19. Core block in free storage

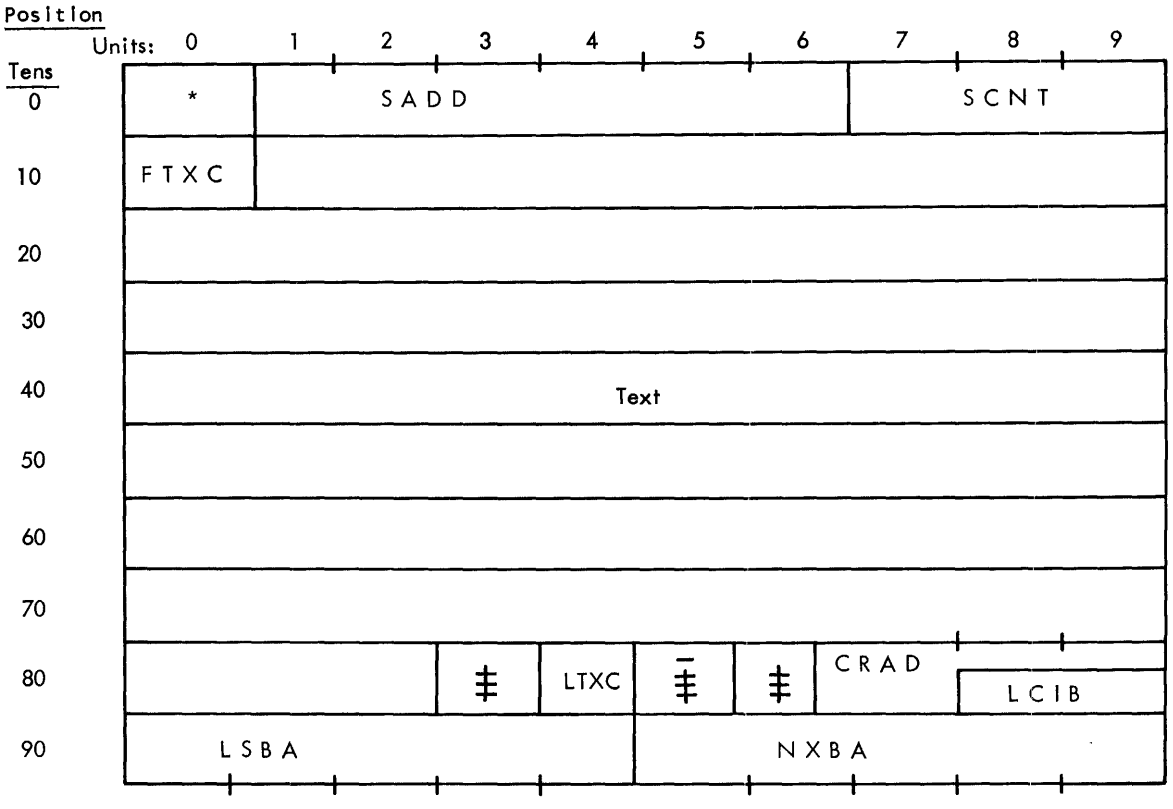


Figure 20. Core block in receive status

Groupmark (one character, position 86). This position can be used as an indicator, but it is currently a spare.

CRAD (one character and zone-bits of the next two, positions 87 through 89). CRAD contains the Core Address of the core block that preceded the current one. Since the core blocks originate at even hundreds, only the first character and the two following zone-bits are needed. The address references the asterisk (position 0) character. This address is used to locate characters in a previous block. Although the previous block has been written to the pack and has been returned to the free storage chain, there is a possibility that the block has not been reassigned. The Scheduler increases the odds by always releasing a block to the foot of free storage. (Blocks are always assigned from the head.) If the block is still there, a time-consuming disk read can be avoided. This is a tally of the number of lines or units that begin in the block, and it is used with LINE in the TST to locate specific lines for correction or insertion.

LSBA (five characters, positions 90 through 94). The Last Block Address is the backward disk pointer, pointing to the preceding block. The first block on the disk chain has a blank LSBA. (This should not be confused with the "First Block Area" discussed below.)

NXBA (five characters, positions 95 through 99). The Next Block Address points forward to the succeeding disk block. While in core, NXBA is the address of where the core block is to be written when filled. Just before writing, it is changed to point forward (in anticipation) to the next disk block in the chain. The last block of an input chain has an NXBA equal to its sector address.

Block in Transmit Status

A block in Transmit status (see Figure 21) has several important fields and locations:

SADD (six characters, positions 1 through 6). This is the sector address, and it is set by the Scheduler prior to reading a block from the pack.

SCNT (three characters, positions 7 through 9). SCNT is the sector count portion of the disk control field and is always 001.

FTXC (one character, position 10). FTXC is the label for the First Text Character.

TREW is composed of a record mark followed by a 1. When the 1448 senses the record mark in its buffers, it sets the EOB (WM) bit for the appropriate line and requests an interrupt. The Scheduler, upon sensing that a 1 follows the record mark, will immediately queue up the next block to be read from the pack, setting the SVRA in the MST to the core block address for the new block. Application programmers need not be concerned with setting TREW. The Scheduler will set it automatically.

LTXC (one character, position 84). The Last Text Character label identifies the location of the last position that may contain a character for transmission.

Record Mark/Groupmark (two characters, positions 85 and 86). The record mark, when sensed in the 1448, will cause the EOB (WM) bit to be set in the status character for that line and request an interrupt. The Scheduler, finding that a groupmark follows the record mark, will set the running address to the next block of output (read in when TREW was encountered).

TWCS (two characters, positions 87 and 88). Two Character Storage is where the two characters replaced by TREW are stored until Transmit Early Warning is sensed by the Scheduler. At that time, the two saved characters are moved back into their proper positions.

LSBA (five characters, positions 90 through 94). The Last Block Address is the backward pointer for the dynamically allocated output chain.

NXBA (five characters, positions 95 through 99). The Next Block Address is the forward pointer for the dynamically allocated output chain. The last block of an output stream must have a blank NXBA.

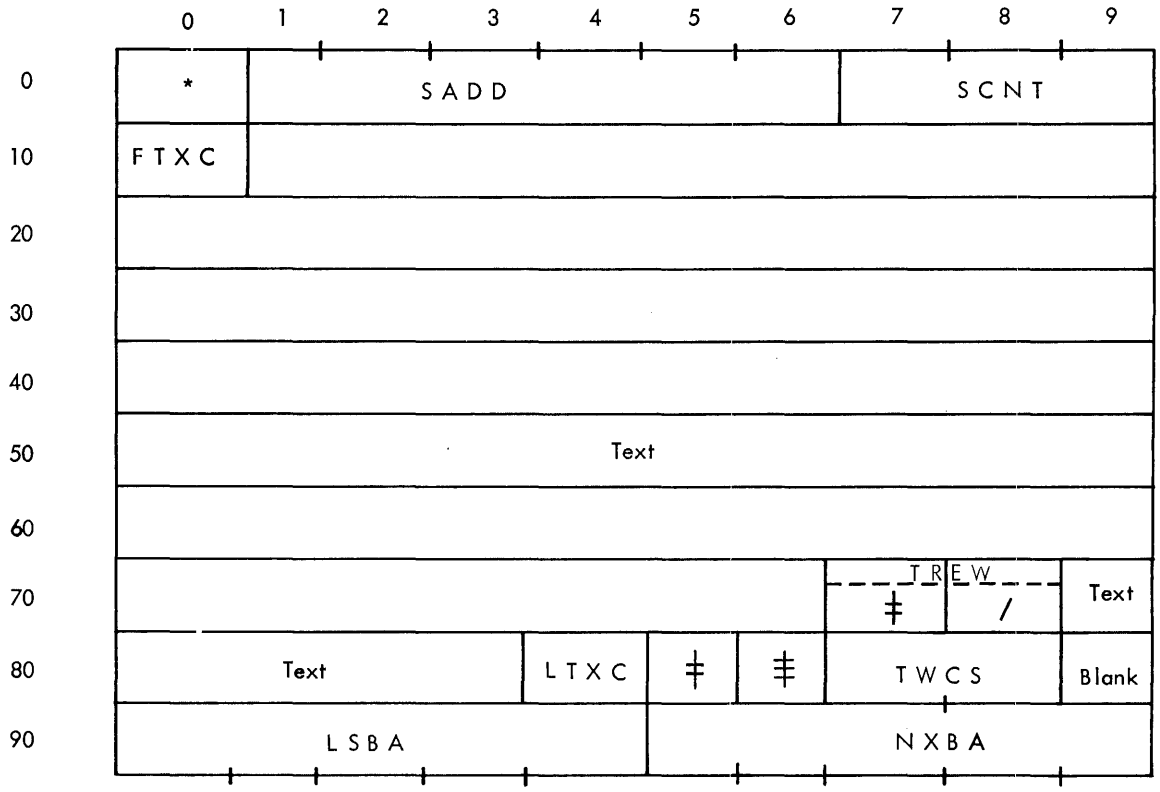


Figure 21. Core block in transmit status

Receive Status

The conventions followed in receiving and storing text are that the text in a block always begins with FTXC and ends with the first groupmark. The latter convention is important to note because it is quite common for text in a block not to extend all the way to LTXC. If corrections or insertions are made, blocks must be split apart, leaving some partially filled. An application program scanning text should finish the scan of a block with the first groupmark encountered.

Some Attention actions are imbedded within the text stream and serve to control the Format and Print program (FRPRT) when it is generating output. These actions are: ATTN a, ATTN u, ATTN +, ATTN h, ATTN f. The Attention character will precede the action indicator and should be checked for by an application program.

In order to start each line with a known condition, a Downshift always is inserted after a Carrier Return. At the beginning of a line or unit, this Downshift character is wordmarked. In the Automatic Mode, all Carrier Returns are also wordmarked. As an example, a line in the Uncontrolled Mode might appear as follows:

```
Wordmarks:   -
Characters:  ]<> R Q$ OR @ZU @OJUK]
```

Translation: Carrier Return, wordmarked Downshift, Upshift, "N", Downshift, "ow is the time.", Carrier Return. (Characters underlined in the example are shown in quotation marks in the translation.)

An example in the Automatic Mode would be:

```
Wordmarks: ---              -                  --
Characters: ]]<>O<C IQT VQSM@]<R$OSAG VQSM@ NOSAK]]
```

Translation: Wordmarked Carrier Returns (2), wordmarked Downshift, Upshift, "I", Downshift "f you don't" Wordmarked Carrier Return, Downshift, "swing, don't ring", Wordmarked Carrier Returns (2). (Characters underlined in the example are shown in quotation marks in the translation.)

Note that in this example the imbedded Carrier Returns are wordmarked.

Transmit Status

Since a record mark sets the EOB (WM) bit, it is used for Transmit Early Warning, to end a block, and to end system messages. The character immediately following the record mark tells the Scheduler what needs to be done next.

A 1 following the record mark denotes Transmit Early Warning. The Scheduler will queue the next block of output for a read from the pack and restore the two saved characters.

A 2 following the record mark indicates the end of a system message, and the Scheduler moves the saved running address to the running address and sets the status to Receive-Idle.

A 3 following the record mark indicates that if the terminal is in the Automatic Mode, an index will be transmitted.

A 4 indicates that the COINS program has just finished transmitting a correctable line to a terminal. This causes the Scheduler to change the block from Transmit status to Receive status and the status character to Receive-Idle.

A 5 following the record mark indicates a stop code. The Scheduler will set the running address to FTXC of the current block, change the core block in Transmit status to Receive status, and set the status character to Receive-Idle.

A 6 indicates the end of all generated output. MISAC is called to set the running address to the end of the input chain, to clear indicators, and to set the status to Receive-Idle.

A groupmark following a record mark indicates the end of output in a block, and the address of the next block is moved into the running address from the saved running address. In summary:

Record Mark Code	Description
‡ 1	Transmit early warning
‡ 2	End of system message
‡ 3	Index to follow system message if automatic mode
‡ 4	End of correctable line
‡ 5	Stop code
‡ 6	End of all generated output
‡ ǂ	End of text in block

Application programs using customized messages should assemble the message in a core block and set the running address to the first character of the message. The saved running address should also point to that character, so that the characters keyed by the operator when the status is returned to Receive-Idle will be stored over the message. In no case should customized messages be transmitted from the overlay or half-track areas, as they will be destroyed by other application programs and their overlays.

All application programs generating output, including system messages, should insert "dummy" timing characters into the text stream following all Tab, Index, and Carrier Return codes. The rules for doing this are specified in the section "The Terminals".

FIRST BLOCK AREA

Because many indicators must be saved by application programs, a section of disk storage is allocated as the so-called "first block area" where application programs can store indicators. These blocks are one sector in length, the address being the 1448 line number for each respective terminal. Thus, the first block for terminal 0 is located in sector 0; that for terminal 39 is in sector 39.

Five fields in the first block are important to most application programs:

TABS (20 characters, positions 0 through 19). The first 20 characters of the first block contain the Tab Setting for the terminal. These characters may be thought of, and are used as, a 6 x 20 binary matrix. Each bit is in one-to-one correspondence to the first 120 positions on the typewriter carriage. A bit in any position indicates that a tab stop is set in the corresponding position on the typewriter.

The matrix is actually divided in half, the first 10 character positions corresponding to the first 60 tab settings, and the latter the next 60 settings. Beginning with the 1-bit in character position 0, successive tab settings are indicated across the 1-row to position 9, then jump back to the 2-bit in position 0. The 2-row continues the same as the 1-row, and successive tab settings continue from the 2-row in character position 9 to the 4-bit in character position 0. This procedure continues until the B-bit is reached in character position 9. From there, successive tab settings are found beginning with the 1-bit in character position 10 and continuing across the 1-row to character position 19 and so on. These settings can be diagrammed as follows:

Bit	Characters 0-9	Characters 10-19
B	50---------59	110---------119
A	40---------49	100---------109
8	30---------39	90--------- 99
4	20---------29	80--------- 89
2	10---------19	70--------- 79
1	0--------- 9	60--------- 69

The setting of the tab stops in TABS is accomplished by MISAC when it interprets the appropriate attention action.

LLDA (five characters, positions 69 through 73). Any program generating multiple block output sets the Last Line Disk Address to the disk address (with high-order zero understood) of the end of the input chain. After such a program writes out the last input block, it sets LLDA to the value of that block's SADD. After the output has been transmitted, the MISAC program will be read in. MISAC will perform housekeeping on the indicators, read in the block referenced by LLDA, set the running address so that the Attention action that initiated the output generating program will be stored over by the next characters entered by the operator, and set the status to Receive-Idle. Thus, everything is set so that keystrokes may again be received from the terminal.

LLRF (four characters, positions 74 through 77). The Last Line Referenced is the line number last requested by the terminal and is set by the COINS program.

LLRL (seven characters, positions 78 through 84). The Last Line Referenced Location is the disk and character address of the start of the last line that the COINS program located for corrections or insertions.

LLRL consists of a disk address (five characters) with one high-order zero understood, and the relative character address within the block. Because core blocks originate at even hundreds, the character addresses will always be the same from block to block. (Character addresses are the two low-order positions of a core block address.) Thus, with the disk address, a block may be

read into core in any core block and the precise line that was last referenced may be located. LLRL could be diagrammed as follows:

27 29753

The number 29753 in the above example implies the disk control field:

*029753001

This is the sector address of the last referenced block.

The number 27 in the example is the numeric portion of the two low-order characters of a core address. It can be used with the high-order character and two low-order zones of the core block address into which the disk block is read. For example:

Zone:	B B
Numeric:	2bb + b27 = K2P
Translation:	102bb + 27 = 10227

This shows how the address 10227, or position 27 in the block that begins at address 10200, could be constructed.

ANXB (five characters, positions 85 through 89). The Address of the Next Block field is the disk address (with high-order zero understood) of the first block of the terminal's chain. This address is initially set by MISAC when the terminal first requests the A or U Mode.

The importance of the tab rack setting is immediately apparent. For example, a program that is formatting text for output must know how many characters a tab will cross to avoid exceeding a specified line width. Dummy characters following the tab are computed according to the number of character positions the print element must move. In addition, some applications attach special meaning to tab stops. An example of the latter case would be entering text in the form of card images where the tab key is analogous to the skip key on a keypunch.

LLRF, LLRL, and ANXB are specially useful to a program that must search through a terminal's text stream for a particular line or unit. Since the end of the stream has a known location (the running address), there are three known points in the chain: the beginning (ANXB), the last line referenced (LLRF, LLRL), and the end (running address). Thus, a search can begin at whichever point is closest to the desired line by following the chaining addresses forward or backward.

All of the indicators in the first block have special meaning to the standard system application programs. Although these indicators may be used by any application program, they should not be altered by these programs, except if the greatest care is that no undesired side-effects will occur.

WARNING: If an application program alters the text stream in such a way as to make LLRL invalid, LLRF and LLRL must be set to all blanks.

PERMANENT STORAGE

Permanent Storage is composed of dynamically allocated blocks of 900 characters (ten sectors) each. These blocks are termed half-tracks since each occupies one-half of an IBM 1316 Disk Pack track. The beginning of every chain is to be found in the Permanent Storage

index, located on 1311 disk storage, drive zero. This index consists of Head Pointers to the first half-track of each document.

Entries in the index are not allocated sequentially as documents are stored. Instead, each entry corresponds to a unique document number. This number is converted into the disk address and character address of the index entry for that document. For example, document 25 is converted into sector address 02000, character address 20 (within the sector). This specific location on the disk will be examined each time reference to document 25 is made.

Normally, 1311 disk addresses must be expressed in a minimum of five digits. However, the disk addresses of half-tracks will always contain a low-order zero. ATS takes advantage of this fact and stores the addresses of half-tracks as four-digit numbers, with a low-order zero understood. The entries in the index are therefore four characters each.

If 1301 Disk Storage is used, all Permanent Storage resides on the 1301. The Permanent Storage index accommodates four-digit addresses with a low-order zero understood. In order to obtain the high-order digit required for the 1301 the zone bits over the two high-order positions are encoded. In this case the index entry has the following format:

```
B-bit   B   B
A-bit   A   A
Numeric N   N   N   N
```

The numerics plus an understood low-order zero comprise the five low-order digits. The high-order digit is encoded as follows:

<u>High-Order</u>

```
B = 8     B = 2
A = 4     A = 1
```

The highest possible address would appear as follows:

```
B-bit   B
A-bit       A
Numeric 9   9   9   9 = 999990
```

The encoding of the high-order digit always appears in a 1301 system. If the address contains a leading zero then the B-bits in the two high-order characters are on, indicating an 8+2 or zero (Modulo 10). Encoding never appears in a 1311 system.

The first sector of the first half-track of every document contains control information pertaining to the document. For example, it contains the identification and lockword for the document.

No document (or message) stored in Permanent Storage may be longer than 99 half-tracks. This is the equivalent of about 90,000 characters of text. This arbitrary restriction has been added to ATS to help prevent the occurrence of excessively long documents. It should also be noted that Working Storage cannot contain a document with more than 9,999 units.

Permanent Storage Index

The Index Block Address (INBA) for any document is always the base address of the index plus the increment appropriate to the document number. This increment is found as follows:

1. Multiply the document number by 4.
2. Take the document number and change its low-order digit to zero.
3. Add the numbers found in steps 1 and 2 together.

For example, to find the increment for document 25:

1. 4*25=100
2. 25 is changed to 20
3. INBA increment=100+20=120.

The INBA is always a seven-digit number. The first five digits contain the sector address of the entry. The last two digits contain the relative character address within the sector. The INBA for any document is always the base address of the index plus an increment as described above. The base address is sector address 02000, character address 00. This address would appear in seven-digit form as follows:

0200000

The INBA is obtained for any document by adding the appropriate increment. For example, for document 25, the increment is 120. Thus, the INBA for document 25 is:

```
0200000
  + 120
-------
0200120
```

or sector address 2001, character address 20.

An index entry is composed of a four-digit disk address, with a low-order zero understood, of the first half-track of the document. The INBA for that document will point to the high-order position of the index entry.

For example, document 25 will be considered again. In sector 02001, positions 20 to 23, the number 5176 might be found. This would mean that the first sector of the first half-track of document 25 is sector 51760.

An index entry that contains a document pointer will always have a wordmark at the high-order or INBA position. An unused entry will always be blank and may or may not have a wordmark in the high-order position.

CAUTION: The first character in any sector is always position 0. Character positions are numbered from 0 to 89.

The number conversion technique used by ATS leaves blanks in character positions 40 to 49. By convention, these entries are used to hold messages for the terminals and output devices. Character positions 40 to 43 of the first 100 sectors are thus reserved for terminals 0 through 99. Terminals 40 to 95 are unassigned. Terminals 96 to 99 have the following meaning:

Terminal	Device
96	Upper- and lowercase chain
97	80-80 tape output
98	Printout with line numbers
99	Punched card output

The INBA for any message queue may be found by adding the terminal number to the disk address portion and forcing the character address to 40. For example, consider terminal 98:

```
0200000    Base disk address
+ 98       Terminal number
0209800    Sector in index
```

Therefore INBA = 0209840. The end of the message chain for a terminal may be found by the pointer in positions 44 to 47. This will be the disk address of the last half-track of the last message.

Index entries corresponding to even hundreds document numbers are not currently used in the system, since such a request is similar to a request for a storage report of the documents in that 100-number band.

Half-Tracks

Half-tracks in Permanent Storage are one-way chained. The last four characters of each half-track contain the address, with a low-order zero understood, of the next half-track in the chain. Instead of a backward chaining address, each half-track contains the INBA belonging to the document. Every half-track of a document should contain the same INBA. Any document deviating from this convention has been mischained and is detected by an offline diagnostic program.

It is possible that more than one message is queued for a terminal or output device. However, all of the messages must be in a single chain behind the appropriate pointer in the index. In order to distinguish where one message ends and the next begins, a single character in position 888 of the half-track (position 78 of the last sector) is reserved for a Text Continuation Indicator (TXCI). If TXCI contains a groupmark, the next half-track in the chain is part of the message. If it is blank, the next half-track is the start of the next message. Because it is desirable to use the same code for both messages and documents, the TXCI convention is used in all half-tracks whether messages or documents. Also, in both cases, a blank forward pointer indicates the end of the chain.

Text in Permanent Storage is in the same format as it was in Working Storage. Each unit begins with a wordmarked Downshift, and Carrier Return characters in the Automatic Mode are wordmarked.

The first sector of a document or message contains control information pertaining to the document. This is as follows:

Positions	Field or Label	Meaning
0-49	IDENT	Contains the identification keyed by the operator if it is a document, or the tab rack setting from the first block (FBK) if a message.

50-53	Not Used	Blanks
54	REEL	Contains the reel identified (single alphabetic character) if the document was retrieved from the Permanent Storage Tape. Otherwise, it is blank.
55-59	Tape No.	Contains the tape document number if the document was retrieved from the Permanent Storage Tape. Otherwise it is blank.
60-61	DCOT	Number of the terminal that created the document.
62-63	Not Used	Blanks
64-69	DCDC	Date document was created in the form MMDDYY.
70-71	DCQH	The number of half-tracks used to store this document. This number can range from 1 to 99.
72-73	Not Used	Blanks
74	DCMD	Mode of last unit of document. B-bit = Automatic Mode. No B-bit = Uncontrolled Mode.
75	Not Used	Blank
76-79	DCUN	Number of units in the document.
80-84	DCLW	Lockword (if any). Since the space (blank) is a valid character in a lockword, this field is set to delta characters (card code: 11-7-8) if no lockword exists, as the delta cannot be keyed from the terminal.
85-89	Not Used	Blanks

The last sector of every half-track contains chaining information as follows:

Positions	Field or Label	Meaning
0-77	Text	
78	TXCI	Contains a groupmark if the next half-track is part of the same document. Otherwise it is blank.
79-85	INBA	Contains the disk and character addresses of the entry for the document in the index.
86-89	NXHT	Contains the disk address, with a low-order zero understood, of the next half-track in the chain. In 1301 disk storage the NXHT will use the zone bits on the hundreds and thousands position. A NXHT of blank signifies the end of the chain.

The control fields, (except TXCI) in all half-tracks have a wordmark in the high-order position.

Half-Track Allocation and Deletion

ATS uses a "circulating file" concept. The heart of this approach is the bit map, which is a map of every half-track available for Permanent Storage, one bit per half-track. If a half-track is in use, the bit corresponding to it will be on. Conversely, if a half-track is available, the bit will be off. The storing of a document involves the search of the bit map for available half-tracks. This search is serial, beginning with the last bit checked. Eventually, the end of the map is searched, at which time the search continues at the beginning. The circulation through the bit map is what gives the circulating file its name. The deletion of a document involves turning off the bits, corresponding to the constituent half-tracks.

Each 1301 module requires 40 sectors of disk storage for the bit map. This will accommodate a five-module 1301 system on one cylinder (1311 storage). In a 1311 system only the first 20 sectors are used. The remainder of the cylinder may be used at the user's discretion. Six bits of every character position in the bit map correspond to six half-tracks. The first character corresponds to the first six half-tracks. The second corresponds to the next six half-tracks, and so on. The allocation within a character is as follows:

B	Half-track 5
A	Half-track 4
8	Not Used
4	Half-track 3
2	Half-track 2
1	Half-track 1
WM	Half-track 0

The 8-bit is not used to avoid the generation of a groupmark/wordmark.

PERMANENT STORAGE TAPE

Many installations find it desirable to keep seldom referenced documents on Permanent Storage tape and thus free the disk space for more active documents. Documents on tape are numbered by the Reel Identification character plus their relative position on the tape. Unlike those in the Permanent Storage on disk, these documents are numbered by the system, not by the operator.

The Permanent Storage tape consists of 900-character (half-track image) records in the Load Mode (even parity) followed by a single tapemark. There is no header label.

The body of the tape is composed of Permanent Storage half-tracks in SELECTRIC code copied from the disk (with the exception of the chaining addresses, which are blank), and all Attention characters (0-5-8) are changed to segment marks (0-7-8).

The previous section mentioned that the first half-track of every document contains control information pertaining to that document. When a document is first stored in disk Permanent Storage, the reel and tape number fields are blank. At the time the document is written to tape, the appropriate reel and document numbers are loaded into

their respective fields (reel character in position 55 and document number in positions 56-59). The tape is then written. These fields have no meaning for a document stored only on the disk; hence they are usually blank on the disk. They appear in disk Permanent Storage only when a document has been retrieved from tape.

The tapemark signifies the end of the tape. There are no tapemarks between documents. When the system scans the tape, the beginning of a document is sensed by testing for a wordmark and no zone bits in position 60 of each record. Position 60 is the high-order digit of a field containing the two-character number of the terminal that stored the document. This field is in the first half-track of every document on the tape. It is possible, however, to encounter a wordmark in position 60 in a record that is not the first record of the document. The text stream contains wordmarked Downshifts at the beginning of every unit and, if the text is in the Automatic Mode, wordmarks on the Carrier Returns as well. It will be noted, however, that the terminal number will never contain a zone bit, but both Downshifts (BA 842) and Carrier Returns (B 841) will contain a zone bit. Thus, if a wordmark is encountered at position 60, and if the character contains no zone bits, the record is the beginning of a document. If the character contains zone bits, it is not the beginning of a document.

ITEMS IN LOWER CORE STORAGE

The branch addresses of the various system subroutines begin at position 336 and continue to position 597. They are located here so that the address will not change on each reassembly of the Scheduler. In this area, 160 positions (401 through 480 and 501 through 580) are reserved to accept binary images from cards during peripheral card-to-tape operations.

The standard ATS system messages are stored commencing at position 609 and continuing through position 999. These are also located here to avoid their changing with each reassembly of the system.

Position 181 (or 182 in some systems) is the interrupt entry to the Scheduler. The main body of the Scheduler begins at position 1000. It is followed by IOCNX and the other system subroutines extending to position 5890.

SYSTEM MESSAGES

All 21 standard ATS system messages are resident in core with a fixed location for each. Any program may use these messages and, in fact, all terminals may receive the same message at the same time. However, no program may alter these messages in any manner. Custom-made messages are transmitted from a core block. Following the ATS text-handling conventions, all system messages end with a record mark 2 or record mark 3 sequence. The 21 messages are:

System Message A - Underscore - backspace sequence

System Message B - END YOUR UNIT

System Message C - ILLEGAL ACTION

System Message D - UNACCEPTABLE NUMBER

System Message E - IDENT PLEASE:

System Message F - AUTOMATIC MODE

System Message G - UNCONTROLLED MODE

System Message H - HEADING MODE

System Message I - FOOTING MODE

System Message J - ***LINE CANCELED

System Message K - SIGNAL WHEN READY

System Message L - "A" MODE REQUIRED

System Message M - "U" MODE REQUIRED

System Message N - CLEARED

System Message P - NONE

System Message Q - TRANSMITTED

System Message R - END OF STORAGE

System Message S - Single index used with automatic mode

System Message T - GO ON-LINE

System Message U - XX USERS

System Message V - READY TAPE

THE SCHEDULER

The Scheduler is a special purpose monitor designed especially for ATS. It handles the servicing of the 1448 multiplexor, 1311 and 1301 disk input/output, and work scheduling among the various application programs on a multiprogrammed basis.

There are four entry points to the Scheduler. One of these is termed SCHAX (Scheduler entry A). It is the point at which a 1448 interrupt gives control to the Scheduler. The Scheduler saves the status (that is, return address, index registers, high-low-equal compare latches, and the overflow indicator) and goes to SCHBX (Scheduler entry B).

SCHBX is the point where the multiplexor scan is executed. After the scan, the Scheduler processes any Early Warning or EOB conditions that have occurred. If the 1-bit of MODE is on after these tasks, the Scheduler will branch to the address contained in ADPR. Following the peripheral program operation, disk input/output is accomplished by examining the respective ARM queues, top entries first. The Scheduler will attempt as many disk operations as possible (a maximum of two), depending on the time available. The priority of operations is: (1) Write Check, (2) Write, and (3) Read. However, if there is time for both a Read and a Write operation, and one of each is waiting, the Read will be performed first. This is because once a Write operation has been performed, no further disk operation may be attempted until the corresponding Write Check is done upon the next Scheduler entry.

The SCLK (Simulated Clock in the System Status Table) is used by the Scheduler to schedule time for each operation. At the SCHBX entry to the Scheduler, the clock is set to 99 (100 milliseconds) and is subsequently decremented the maximum amount of time that may be used on any one operation. If Early Warning must be processed, SCLK is decremented by 28 milliseconds. Fifteen milliseconds are allocated for EOB processing. Usually, 15 milliseconds suffice for a peripheral operation. It takes a maximum of 40 milliseconds to perform a Read or Write operation on the disk files. If SCLK is greater than 80 when the disk code is entered, two operations will be performed. If it is less than 80, only one will be performed. In any case, one disk operation can be performed with each pass through the Scheduler. Since there is a maximum of 120 milliseconds between scans of the multiplexor, an application program is guaranteed at least 20 milliseconds of processing time between interrupts.

After completing disk input/output, the Scheduler will restore the saved program status, if it was entered at SCHAX. Otherwise, it will go to the FINIX (Program Finished entry).

Since the system operates in real time, it must have a basic program cycle or loop in which to operate when it is not giving control to application programs. SCHBX is the "top" of the main Scheduler loop; FINIX is the "bottom".

FINIX is the third entry into the Scheduler. Not only does the SCHBX entry fall through to here, as explained above, but all application programs branch to FINIX after they have finished their processing. The exit parameter for an application program following a branch to FINIX is a DCW statement containing the address constant of the start of the routine. Thus, the exit of COINS to the Scheduler looks like this:

```
B       FINIX
DCW     +COINS
```

where the label COINS appears on the first instruction of the program.

The FINIX portion of the Scheduler is responsible for the scheduling of work among the various application programs. Its rule is that Work in Progress (WPA entries) must be serviced before any New Jobs (NJB entries) are attempted. Thus, the WPA queue is examined first, and control is given to the program corresponding to the top entry. If the WPA pointer is blank (no entries), the New Job queue is examined. Beginning with the top entry in the queue, the Scheduler checks to see if the program is busy (has lost control because it has requested disk I/O but still has processing to do). If the program is not busy, the entry is checked to see if the overlay area is required, and if so, if the overlay area is in use. If an entry cannot be serviced immediately, it is left in the queue, and the next entry is checked. When an entry is found that can be processed and that needs the overlay area, the Program in Overlay (PROV) indicator is checked to see if the program is already in it. (The program would be there if it were the last program in use in the overlay area.) If it is, the program is given immediate control; if not, a high-priority read request is given for the program. In either case, the BUSY (Program Busy), OVBZ (Overlay Busy), and PROV indicators are all set. If the program is resident in core, like EUNIT, only the BUSY indicator is set.

The fourth entry to the Scheduler is SCHCX. It must be used by application programs whenever they must have 100 milliseconds of unencumbered computer time. This is required for reading cards or writing long tape records. All that is required is to branch to SCHCX before giving the input/output instruction. The Scheduler will return when 100 milliseconds of time are available. No registers are preserved.

If there are no entries in either the WPA or NJB queues, or if no NJB entry can be serviced, SCLK is checked. If it is not 99, control passes to SCHBX. If it is 99, control passes to the next instruction in an application program if it previously entered the Scheduler at SCHCX. If that test fails, control passes to the address in ADPR if the 2-bit of MODE is on. The return branch from the peripheral area encounters a branch to SCHBX, which repeats the cycle.

IOCNX is the program entered by any program, including the Scheduler, when disk input/output is required. It is the responsibility of IOCNX to place disk I/O requests in the proper ARM queue. Following the queuing of the request, IOCNX branches to the SEEKS subroutine. SEEKS will compute and give the direct seek if the arm is not busy. Control is then returned to IOCNX. When it finishes processing the request, IOCNX will return immediately to the next instruction if it was entered from the Scheduler. It will branch to FINIX if it was entered from an application program.

To request input/output from or to the disk, a program must have a branch to IOCNX followed by a calling sequence. The calling sequence, which makes up part of the information in the ARM entry, is as follows:

```
                B     IOCNX
ONE             DCW   #1          CONDITION COMMUNICATION
TWO             DCW   #1          IOOP
THREE           DCW   #3          IOAR
```

ONE is the condition communication field. When control is returned to the program, this indicator will be blank if input/output was successful. It contains a one if a disk error condition was encountered.

TWO is the IOOP. Its configuration is precisely the same as the IOOP field in the ARM and WPA entries. The B-bit indicates priority: ON is high priority, OFF is low priority. The A-bit signifies the type of operation: ON is a write, OFF is a read. The numeric portion signifies the sector count: blank means that the count is already set in the disk control field; a number from one to nine indicates a sector count from one to nine; and a zero indicates a count of ten.

THREE is the high-order address of the disk control field where the operation will take place. Since an asterisk or a lozenge is always the first character of the disk control field, the IOAR field will always contain the address of this character.

Something must now be said about wordmarks in the disk control field. The SEEKS subroutine clears the wordmark in the high-order position of the sector count and forces a wordmark into one position less than the high-order position of the sector address. A wordmark in the first or second position of the disk control field will be ignored. However, since the disk control field is saved and restored, any other wordmarks will cause unpredictable and erroneous results.

The ATS program will halt if an input/output operation is not successful. Since the 1311 and the 1031 disk files are very reliable, any error condition is almost always a programming error. Most programs are unable to continue processing anyway if they cannot complete a disk operation successfully.

It is important to note that no indicators or index registers are saved by IOCNX. If it is important for an application program to have the system status saved, provision must be made for doing this in the application program itself.

CHAIN (CHAIX)

The CHAIX subroutine is used principally by the Scheduler, but can also be used by application programs. Its purpose is to queue up a full block that is in the Receive Status for a write to the disk pack and to assign a new block for the terminal. The running address is set to FTXC in the new block. Return is immediate, and index registers 1 and 3 are preserved.

The entry sequence is a branch to CHAIX, with index register 1 equal to ten times the terminal number (this is used to reference the appropriate entry in the MST), and index register 2 equal to any address within the full block. When control is returned, index register 2 and the terminal's running address will have the address of FTXC in the new block.

It should be emphasized that this subroutine is to be used for blocks in the Receive status only. The new block is completely set up for the program including LSBA, NXBA, and groupmarks.

NEXT DISK BLOCK (NDBKX)

The NDBKX subroutine is responsible for the maintenance of the pool of available sector addresses. It will assign new sectors from the highest sector address in the system downward as they are needed.

All programs have free access to NDBKX. The entry sequence is a branch to NDBKX, with index register 3 equal to the low-order position where the disk address is to go. For example, if the block being worked with is addressed by index register 2 and a new sector address is to be loaded into the NXBA of that block, the sequence would appear as follows:

```
SBR  X3,NXBA+X2

B    NDBKX
```

Return is immediate, and all index registers are preserved. However, it is possible that there are no available sector addresses (if all of disk storage is completely used up). If NDBKX provides a sector, return will be four position beyond the branch to NDBKX. If NDBKX is not successful, return will be to the instruction immediately following the branch to NDBKX. It is an ATS practice that no halts are permitted due to absence of disk blocks. If this condition should occur, the application program should cease processing and type the END OF STORAGE message.

PURGE DISK BLOCK (PDBKX)

PDBKX is the subroutine that is responsible for purging of disk addresses that are no longer in use. PDBKX fills core blocks (to a maximum of three) with the sector addresses as they are purged. When three core blocks have been filled, one is written to disk storage. These addresses of purged sectors are all allocated by NDBKX before fresh addresses are generated.

The entry sequence is a branch to PDBKX, with index register 3 pointing to the low-order core address of a five-character sector address to be purged. For example, if index register 2 contains the address of the core block whose SADD field is the sector address to be purged, the sequence would appear as follows:

```
SBR X3,SADD+X2
B   PDBKX
```

Return is immediate, and all index registers are preserved.

NEXT CORE BLOCK (NCBKX)

The NCBKX subroutine supplies the requesting program with the location of the block at the head of free storage. The block is also cleared at this time. Return is immediate, with index register 3 equal to the address of the asterisk in the new block. Index registers 1 and 2 are preserved. A program need only branch to NCBKX; no entry parameters are required. If an error condition arose because there were no available blocks, NCBKX would halt the system.

HEAD OF FREE STORAGE (HDFSX)

Core blocks containing text that clearly will not be needed again are always released to the top of the free storage chain. The HDFSX subroutine is responsible for this release.

To release a block to the head of free storage, a branch to HDFSX is executed, with index register 3 containing the address of any position within the block to be purged. Return is immediate, with index register 3 containing the address of the first (asterisk) position within the purged block. Index registers 1 and 2 are preserved.

FOOT OF FREE STORAGE (FTFSX)

If it is possible that text in a block might be referenced again, it is good practice to release the block to the foot of the free storage chain. Since NCBKX always assigns blocks from the head of free storage, releasing to the foot of the chain increases the likelihood that if the text is needed again it will still be available in the free storage chain.

FTFSX is the subroutine that is responsible for releasing blocks to the foot of the free storage chain. The entry sequence is a branch to FTFSX, with index register 3 containing the address of any position within the block to be released. Return is immediate, with index register 3 set to the first (asterisk) position of the released block. Index registers 1 and 2 are preserved.

SET NEW JOB (SNJBX)

The SNJBX subroutine enters program requests into the New Job queue in the list area. It also sorts the new entry according to priority based on the relative PST location of the requested program. In the case where two or more requests are for the same program, the requests are handled first in, first out (FIFO).

To set a New Job entry, a branch to SNJBX is executed with index register 1 set equal to ten times the terminal number, and index register 2 containing the PST number of the desired program. Return is immediate, with index register 1 preserved. Index registers 2 and 3 are not preserved.

SAME UNIT (SMUNX)

SMUNX is the subroutine responsible for placing the Downshift in text being received in the Automatic Mode, following a Carrier Return in the middle of a unit. It is an ATS text-handling convention that Carrier Returns imbedded in a unit are always followed by a Downshift so that each line entered begins in a known condition. SMUNX performs this function. If the Downshift will exceed LTXC in the core block, SMUNX will set up the next block via CHAIX. Before returning to the calling program, SMUNX sets the line status to Receive-Idle.

The entry sequence is a branch to SMUNX, with index register 1 set equal to ten times the terminal number, and the running address equal to that of the last entered character, plus 1. Return is immediate, with index register 1 preserved. Index registers 2 and 3 are not preserved.

NEW UNIT (NWUNX)

The NWUNX subroutine sets the wordmarked Downshift character required in the first position of any new line (in the Uncontrolled Mode) or unit (in the Automatic Mode). If the wordmarked Downshift will exceed LTXC in the block, a new block is set up via CHAIX. Before returning to the calling program NWUNX sets the line status to Receive-Idle. The entry sequence is a branch to NWUNX, with index register 1 set equal to ten times the terminal number, and the running address equal to that of the last entered character, plus 1. Return is immediate, with index register 1 preserved. Index registers 2 and 3 are not preserved.

RETRIEVE (RTRVX)

If text in a block that has been released is discovered to be needed by a program, a disk operation can be avoided if the text is still

intact in the block in free storage. The RTRVX subroutine attempts to retrieve the immediately preceding block from free storage. This is accomplished by finding the previous block via the CRAD register in the current block (see the section "Core Blocks"). The check made to see if the previous block still contains the desired text is whether its NXBA equals the current block's SADD. If so, the block is unchained from free storage, SADD and SCNT are set and core block position 96 is set to blank. If the block's NXBA is not equal, the current block's SADD core block position 96 is set to 1 and a new core block will be set up, including its SADD, so that the desired block may be read from the disk pack.

The entry sequence is a branch to RTRVX, with index register 3 containing any address in the current block. If RTRVX is successful, core position 96 will be set to blank and index register 2 will contain the address of the LTXC position in the retrieved block. If it is not successful, core position 96 is set to 1, and index register 2 will be set to the address of the first (asterisk) position in a new block ready to receive the block to be read from disk storage. In either case, return is immediate, and index register 1 is preserved.

MISCELLANEOUS PROGRAMMING NOTES

SOURCE PROGRAM PREPARATIONS

Programming Language

All ATS programs are written in the language of, and must be assembled by, the IBM 1401/1440/1460 Autocoder (on disk), program number 1401-AU-008.

Source Program Decks

A modular approach has been used throughout ATS. All application programs are assembled and handled as individual entities. A diagram of a source program deck appears in Figure 22. Comments on various parts of the deck appear below.

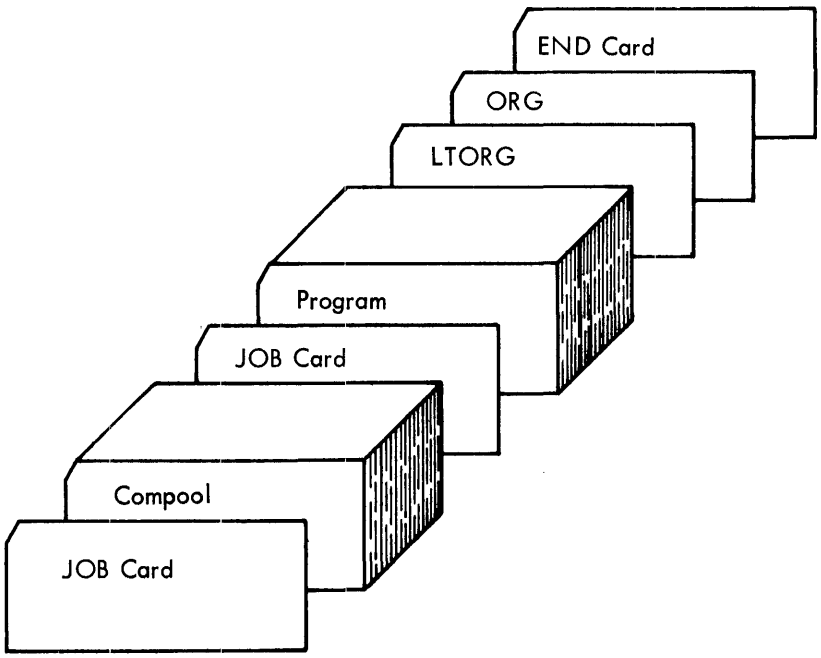


Figure 22. Source program decks ready to assemble

Job Card

The last five characters in the Job card will be punched into every card of the object deck. To aid in the separation of the decks, it is advisable to begin each overlay with a Job card with the five-character mnemonic of the overlay punched in the last five columns.

Literal Origin (LTORG) Statements

An application program with multiple overlays must keep the literals used in a segment, within that section of code. Ordinarily, literals are assigned addresses at the end of the program. In the case of a program with two overlays, for example, the literals would normally be placed after the code in the second overlay. Literals used in the main program would then be accessible only when the last overlay was in core. Literals used in the first overlay would never be in core. To solve this problem, a LTORG statement must be inserted between each program segment, main program or overlay. This forces the literals used in a segment to be located at the end of the segment.

Origin (ORG) Statements

Each application program segment must begin with an ORG statement to locate the segment appropriately in core. The main program segment will be located in the Overlay Area and the overlay segments (customarily) in the Half-Track Area. In the latter case, multiple overlays can begin at the same location with no confusion to the assembler. In addition, the disk control field used to write the segment to the disk must be located immediately preceding the first instruction of the section. The ATS DSKLD (Disk Load) program (see below) uses this field to write the segment to disk storage. The first card of every segment (besides the Job card and comment cards) must be an origin statement. The second card must be the disk control field. For example, for a main program segment, the initial cards would be as follows:

```
          JOB       PROGMB (Application Program Name)
*Any number of comment cards
          ORG       OVAREA-10       MAIN SEG IN OVLAY AREA
          DCW       @*001200020@    DISK CTL FIELD
PROGMB    PR10      1st Instruction
```

The sector count for the main program segment is always 20 sectors. The sector address is the location of the program decided upon by examining the Disk Map (see "Disk Storage Allocation"). In the above example it is sector 1200. Program overlays are read by the main program segment and thus are under control of the application program. They may be located anywhere within the Overlay or Half-Track Areas, may be of any length (the sector count may vary) within those areas and may be located anywhere on the disk. The most convenient method of defining overlays is to locate them at the beginning of the Half-Track Area and make them ten sectors long. If the overlays are located sequentially on the disk immediately following the main segment, then the incremental disk addressing scheme mentioned in the section "Program Overlays" may be used. If the customary approach is followed, the beginning of an overlay would be as follows:

```
          LTORG   *Previous Segment

            EX          DSKLD           Previous Segment

            ORG         HFTRAR-10

            DCW         @*001220010@    DISK CTL FIELD

            JOB         Program Name--Overlay Number

     *Any number of comment cards

            PV10 Overlay Program
```

The disk address is the ten sectors immediately following the main program. The use of the Execute (EX) statement will be explained in the section "Disk Load Program", below.

COMPOOL (Communication Pool)

To avoid label ambiguities, a special deck has been set up. This deck contains the addresses of all the system tables, indicators, system messages, and system subroutines. It is assembled with every program in the system. The COMPOOL deck is always placed before the program to be assembled.

DISK LOAD PROGRAM (DSKLD)

DSKLD is an ATS utility program provided with the system. Its function is to write ATS and user-written application programs to the disk. An Execute statement is the last card of every program segment except the last segment. This macro executes the disk-write portion of the DSKLD program. The End card executes the disk-write portion of DSKLD on the last program segment. The label on the Execute and End statements is DSKLD, which is identified in the COMPOOL. As mentioned in the section "Origin Statements", the disk control field used by DSKLD is assembled with the program. The DSKLD program origins at core location 15600. If a program to be written to the disk is not going to be assembled with the COMPOOL the DSKLD label must be identified in the program. This might be accomplished by an Equate statement:

```
                    DSKLD  EQU    15600
```

Peripheral programs are not assembled with the COMPOOL.

The DSKLD program requires the disk control that is to be used to write the program to origin at core location 100 or above. Also, since DSKLD sets a groupmark wordmark at the end of the program segment the area written must not overlap core location 15600.

To load an assembled, condensed program deck, the Autocoder Bootstrap cards and the COMPOOL cards, if present, must be removed from the front of the deck. The DSKLD program is then placed in front of the deck and the program is ready to load. The COMPOOL cards are easily identifiable by slash characters (0-1 punch) in columns 76 to 80 (see Figure 23 for a diagram of a condensed program deck ready to load).

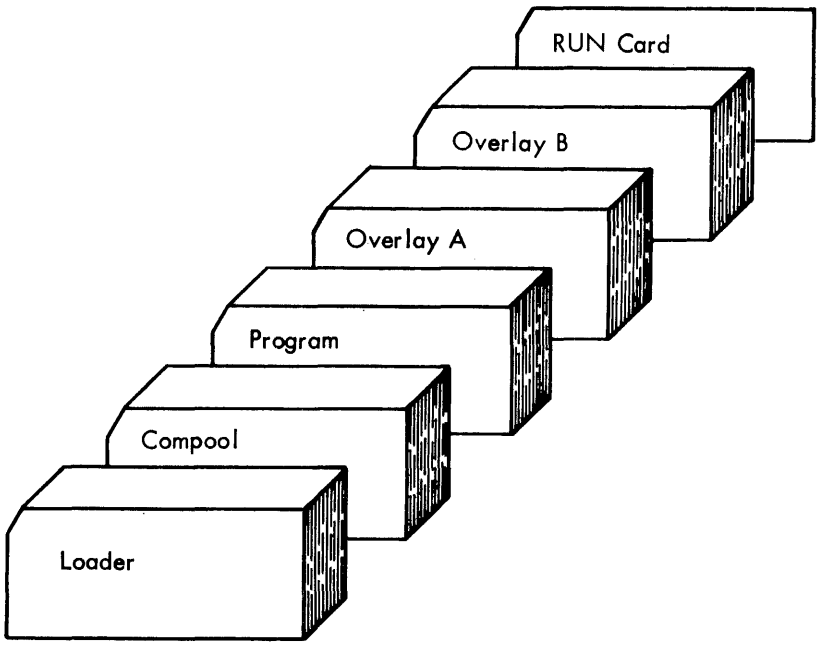


Figure 23. Preparation of condensed (object) program deck for loading

Peripheral programs are called from the disk into core storage by a specific request from the console operator through terminal 0. Communication to peripheral programs is through sense switches, as they have no communication with any terminal.

A peripheral program operates as part of the Scheduler. Thus, it may get control whenever the Scheduler is entered because of a 1448 interrupt or application program disk activity.

The ATS program is very time-sensitive. If the multiplexor needs service within 50 milliseconds and a peripheral program reads a card (requiring 75 milliseconds), characters in the 1448 buffer will be lost. Also, if the peripheral program uses all the time between multiplexor scans, there will be no time available for the application programs, and terminal requests will be processed too slowly.

Two types of entries are provided for peripheral programs, depending on the time requirements of the program. One entry will give control to the peripheral program at every multiplexor scan. This entry is used by programs that require 50 milliseconds or less. An example of this type of peripheral program would be a tape-to-printer program that reads tape records (preferably blocked) and writes single lines to the printer. This timing is why the Print Storage feature is required for all online printer operations. The second peripheral program entry gives control to the program whenever 100 milliseconds are available. A program reading cards for a card-to-tape operation is an example of a peripheral program requiring the second type of entry.

A peripheral program is given control as soon as it is read from the disk. The console operator, however, will usually call the program into core before setting the sense switch communications. Thus, the program may find the sense switches set, in the process of being set, or not set at all. The peripheral program should check the sense switches on every entry, and, if they are incorrect or inconsistent, it should immediately return control to the Scheduler. Sense switches may be used to indicate to the program if output devices are ready. One sense switch may be used as an actuator, such that the program will take no action until it is set ON.

For example, in a tape-to-printer operation, the operator will call the program into core, ready the required devices, and then set the sense switches. Sense switches are also used to express variable conditions. For example, a switch may be set to indicate whether a tape is in even or odd parity. The sense switches available to peripheral programs are D, E, F, and G. Switches A, B, and C are reserved for the main system. ATS is operated with the I/O Check Stop Switch OFF. Card reader and printer errors will not cause the processor to halt, but will set an error latch that may be tested by various branch-on-indicator instructions. The program should make every possible test to be certain that a peripheral device is ready before using it. It is quite possible for a peripheral program to be entered often enough to exceed printer speed. Thus, the Printer Busy indicator should always be checked. If an error or busy condition is detected, return to the Scheduler should be immediate.

The return from a peripheral program to the Scheduler is to the instruction following the branch to the peripheral program. Because this address may change with different assemblies of the system, the peripheral program must save the return address with a Store B Register instruction. This sequence might appear as follows:

```
CARTA     SBR     CA50+3      SAVE RETURN ADDRESS
                  ----------
                  ----------
                  Body of Program
                  ----------
                  ----------
CA50      B       0           RETURN TO SCHEDULER
```

Disk Dump (DSKDM) is a peripheral program provided as a standard feature with every ATS (see IBM 1440/1460 Administrative Terminal System, Console Operator's Manual (H20-0227), for a program description and operating instructions). In addition to the dump function, DSKDM contains a table with the identification and parameters for every peripheral program in the system. The MISAC program reads this table into the peripheral overlay area whenever the console operator requests a peripheral program. The table is checked to see if the program exists, where it is located on the disk, and what its communication parameters are. The peripheral program is then read into the proper area of core and communication with the Scheduler set. The peripheral program then gets control on the next multiplexor scan. Entries in the peripheral table should be made in the DSKDM symbolic program, and that program should then be reassembled. The point for including these entries is marked clearly in the program listing.

A peripheral program table entry in DSKDM consists of five fields for every program. The first is the five-letter name of the program in SELECTRIC code. This entry is checked by MISAC to identify the program by comparing it with the console operator's request. The second field is the disk address of the program. This is a four-position literal, with a high-order zero understood. The third field is the number of sectors required by the program. This is a two-position literal. The fourth field is the core address of the beginning of the program. This address constant determines both where the program will be read into core and where the Scheduler will branch. The program must begin with the first instruction at this address. The final field is the type of Scheduler entry required by the peripheral program. This is a one-position literal. It is set to 1 if entry is desired on every scan, or 2 if entry is to be made only when 100 milliseconds are available, or 3 if entry is to be made in both instances. The source program for an entry in the table for a program called Model Peripheral Program (MDLPP) might appear as follows:

```
DCW @JUYDD@      Name of program in SELECTRIC code

DCW @2035@       Disk address of program, high-order zero understood

DCW @15@         Number of sectors used by the program

DCW +PERIPH      Core address of beginning of program

DCW @2@          Scheduler entry type -- to be entered when 100
                     ms are available

DCW @‡@          A record mark follows the last entry signifying
                     the end of the table
```

The fifth parameter is set in the MODE indicator in the System Status Table and is checked by the Scheduler before control is given to the peripheral program. Peripheral programs are loaded to the disk using the DSKLD program like application programs.

It should be noted that any peripheral program will degrade system response. Whether or not this degradation is tolerable can be learned

only by experience. Tape blocking is usually a critical factor in this regard. If possible, tape-to-printer operations should always use blocked tape. Blocked tape will eliminate time-consuming tape operations on every entry of the program.

WRITING APPLICATION PROGRAMS

Coding Practices

The first instruction of an application program is always labeled with the five-character mnemonic of the name of the routine. This instruction will always be an unconditional branch to the start of the program. Following this branch, three DCW statements appear. For example, the ATTEN program begins:

```
ATTEN B   AT11
      DCW @00@
AT10  DCW #2
      DCW #3
```

The first DCW is used to designate the modification level of the program. The second will receive the number of the terminal requesting service. The third is used by the Scheduler for temporary storage and should never be used by the program.

The terminal number (which will be located at AT10 in the above example) is multiplied by ten and is then used to reference the proper entry in the MST. (There are ten characters per entry in the table.)

This terminal number is usually used in index register 1. The instructions to load the index register are:

```
MLC @0@,X1
MLC AT10
```

Reference to the running address, for example, is then:

```
MLC RUNA+X1,TEMP
```

It will be found that coding can be reduced by letting index register 1 equal ten times the terminal number and index register 3 equal the address of the first position of the core block being worked with. Since this is the scheme used by the system subroutines, following the same convention will save the code necessary to set subroutine parameters. Input/output with the disk file must be accomplished through IOCNX. It is well to remember that index registers are not saved by IOCNX. They must be saved before giving control to that subroutine if they are important to a program. The overlays of a program must be read in by the program itself, as the Scheduler will read only 20 sectors into the overlay area.

Locating the Attention Line

When an application program is read into core and executed, the only parameter conveyed from the Scheduler is the terminal number. This number will be located in the seventh and eighth positions of the program as described above. The program must determine what is requested by scanning the Attention line. The Attention line is found

by referencing the running address for the terminal in the MST (see the section "Multiplexor Status Table"). The running address is always one greater than the address of the Carrier Return in the Attention line.

If the line is to be interpreted, the beginning of the line must be found. Every unit in the text stream, including an Attention line, will begin with a wordmarked Downshift. If the Attention line happens to be in a single block, the wordmarked Downshift can be located by the following technique:

```
MLC    @0@,X1          SET 10 TIMES THE
MLC    TERM            TERMINAL NUMBER IN X1
MLC    RUNA+X1,X2      GET CR ADDRESS PLUS 1 IN X2
C      0+X2            SCAN TO WORD MARK
SBR    X2              SAVE WM ADDRS-1
```

Often, however, the line will begin in the previous block. This might appear as shown in Figure 24.

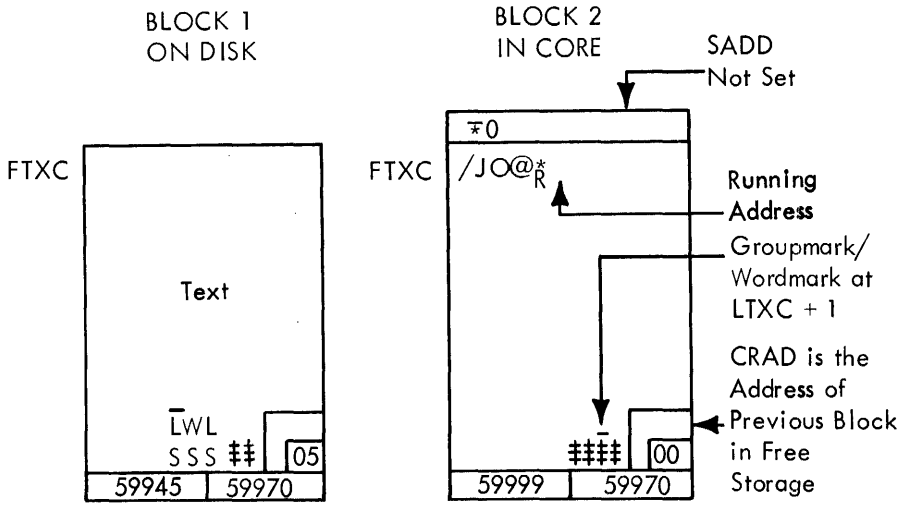


Figure 24. An Attention line split between two blocks

The translation of the Attention line in Figure 24 is: Wordmarked Downshift, Attention, Downshift, X,M,I,T, Carrier Return. The running address is always the address of the Carrier Return plus one. Although an Attention line can span more than two blocks, two is usually the maximum number.

In this case, the block containing the beginning of the line must be retrieved from free storage or read from the disk using RTRVX. It is desirable to have the entire line in core. (An application program may use up to three core blocks.) Thus, special core chaining addresses between the blocks must be set. SKANB in conjunction with RTBLK are standard subroutines that may be used for most of the operations outlined above.

Sample coding for these routines is shown in Appendices A and B. Anyone wanting to include this coding in his own ATS application program should compare these samples with the source program listings of the MISAC program from the version of the system he is using. It is recommended that this code be copied from the program listings.

SKANB will bring the entire Attention line into core whether it is split between blocks or not. Upon setting a program switch, SKANB will purge the dangling block(s) and disk address(es). RTBLK will read the previous block from the disk if SKANB is unable to retrieve it from free storage. SKANB should be entered near the beginning of the application program to bring the Attention line into core. This entry is a branch to SKANB with index register 1 set to ten times the terminal number. Control will return to the next sequential instruction with index register 2 containing the address of the wordmarked Downshift, minus 1. The blocks will be chained both ways with the CRAD of the last block pointing to the previous block and the SCNT (sector count) in the previous block pointing forward to the last block. After this is done, the blocks shown in Figure 24 would appear as shown in Figure 25.

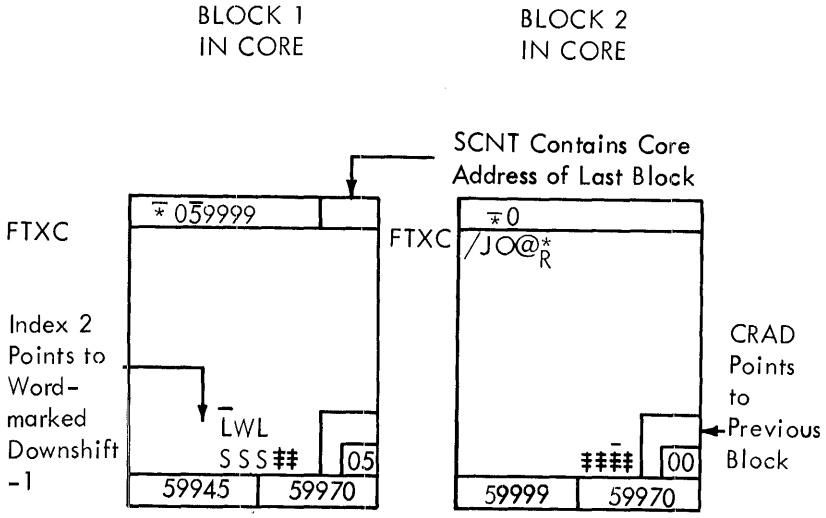


Figure 25. Core blocks with an Attention line ready to process

After the program has finished processing the request, the running address must be set to where the next character from the terminal is to be stored. This will usually be the address of a wordmarked Downshift in the Attention line, plus 1. In the example in Figure 24, the next character from the terminal will be stored over the Attention character. In this case, the block containing the end of the Attention action must be returned to free storage, the disk address (NXBA) returned to the available pool, and the previous block set to the end of text stream status. After this is done, the block shown in Figure 25 would appear as shown in Figure 26.

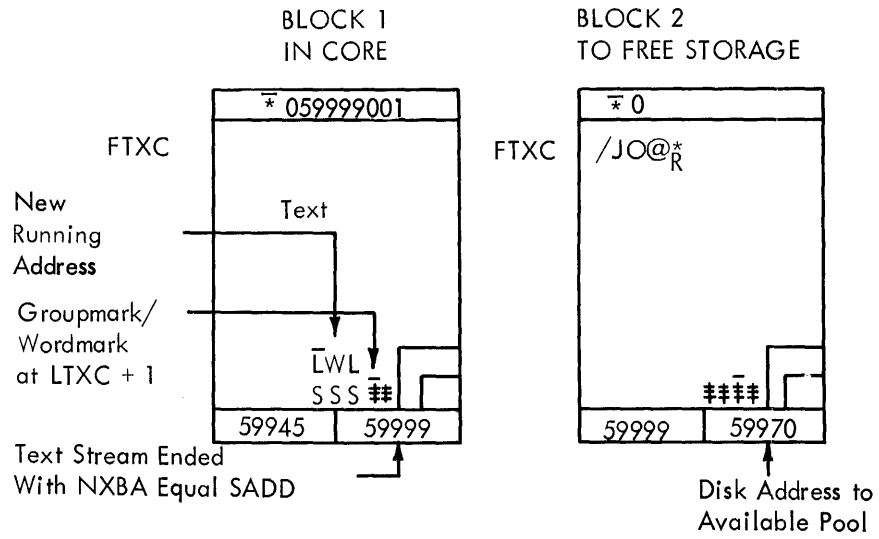


Figure 26. Core blocks after processing an Attention line

The SKANB subroutine may be used to accomplish this by setting wordmarks on the program switches at SK16+4 and SK17+4. These switches are of the form:

```
SK16     BIN  SK17,(   BRANCH TO PURGE CORE BLOCKS
```

This instruction normally appears only in Field Engineering diagnostic programs. It is a branch on parity error if the Check Stop Switch is OFF. Since ATS always runs with the Check Stop Switch ON, the instruction is normally a NOP. If a wordmark is set on the D character, the instruction becomes an unconditional branch. To purge the dangling block, this entry sequence should be used:

```
MLC @0@,X1            SET TEN TIMES
MLC TERM              TERMINAL NUMBER TO X1
SW  SK16+4,SK17+4     SET SWITCHES FOR PURGE
B   SKANB             PURGE DANGLING BLOCKS
```

Return is to the next sequential instruction with the dangling block and disk address purged. The NXBA of the previous block points to itself, thus ending the text stream. Index register 2 contains the address of the wordmarked Downshift, minus 1. The wordmark on the groupmark at LTXC+1 is not set, however, and must be set by the main program.

WARNING: The line must have been brought into core previously by a branch to SKANB without these switches set before attempting to purge dangling blocks by a second entry to SKANB with the switches set.

In some applications, it is desirable to keep the Attention line in the text stream. In these cases, the previous core blocks must be returned to the free storage chain. The contents of the blocks are also safely on the disk. The next characters keyed by the terminal

will enter the block following the last line. The application program is responsible for setting the status for the new line. Thus, a wordmarked Downshift is inserted following the Carrier Return character. The LCIB field in the last block and the Line field in the TST must be incremented by one. If the text is in the Automatic Mode, the Carrier Return on the Attention line must be wordmarked. Examples of this type of Attention line are the requests for the Automatic and Uncontrolled Modes. This type of Attention line may also be split between two blocks. This might appear as shown in Figure 27.

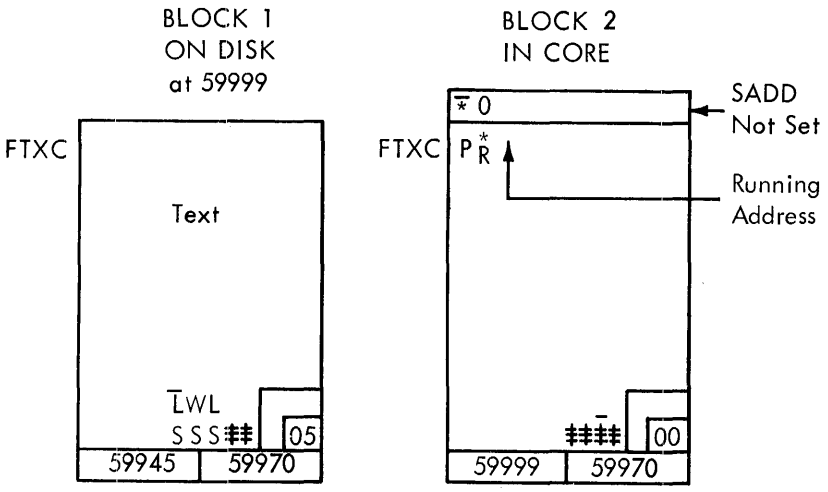


Figure 27. A request for Automatic Mode split between two blocks

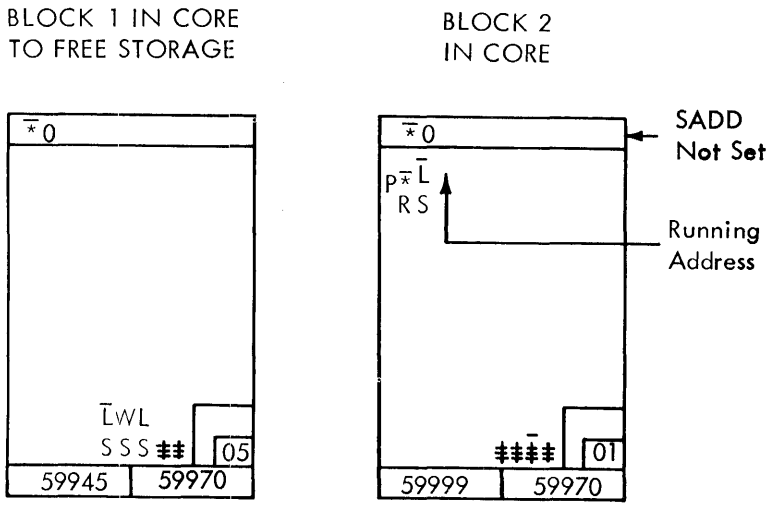


Figure 28. Core blocks after processing a retained Attention line

In the example in Figure 27, the entire Attention Line may be brought into core using SKANB and then interpreted. If the program decides to keep the line in the text stream, Block 1 in core (containing the Attention character) is returned to free storage. A new line status is set following the Carrier Return. After this operation, the blocks will appear as shown in Figure 28, where the illustrated attention action is a request for Automatic Mode.

The return of the core block to free storage is accomplished by a branch to SKANB with the single switch set at SK16+4. The following entry sequence is used:

```
MLC @0@,X1        SET 10 TIMES THE
MLC TERM          TERMINAL NUMBER IN X1
SW  SK16+4        SET TO PURGE CORE BLOCKS
B   SKANB         BRANCH TO PURGE
```

Return is to the next sequential instruction with index register 2 containing the address in the purged block of the wordmarked Downshift, minus 1. The address of the Carrier Return must be obtained from the running address. The new unit status may be set by the NWUNX subroutine. Entry is by a branch to NWUNX with index register 1 set to ten times the terminal number.

Interpreting the Attention Line

Text streams are always scanned one character at a time. The technique for doing this is to prepare indicators and then set them when appropriate characters are encountered. At some logical point, such as when a Carrier Return is encountered, these indicators are examined to determine the content of the line.

An index register (normally index register 2) is used to step through the block. By convention, the block ends with the first groupmark encountered. The groupmark is usually the first character checked after incrementing the index register. In the case of the Attention line when the SKANB subroutine is used, the SSPRT subroutine can be used to step the index pointer to the next block. (Sample coding for this routine is shown in Appendix C.)

Numeric Character Translation

Attention lines containing numeric information, such as line or document numbers, must be converted from SELECTRIC code to BCD before any arithmetic operations may be performed with them. Entry to the routine is with the unknown character in a wordmarked temporary register.

A typical open subroutine is used for this purpose in ATS application programs. This routine serves the double function of identifying digits and translating them. The code for such a routine is given below.

```
*LITERALS REQUIRED FOR OPERATION                                    *

TEMP      DCW =1              HOLDS UNKNOWN CHARACTER
BFER      EQU *+1             HOLDS TRANSLATED CHARACTERS
          DCW =5
TRAN      DCW -MP20+7         TRANSLATE AD CON
TRTB      DCW @Y9=75648321@   TRANSLATE LITERAL
RAWD      DCW =3              DATA AREA FOR TRANSLATION
          ----------
*   BEGIN TRANSLATE ROUTINE                                         *
MP19      SBR X3,BFER         SET POINTER TO BFER START
          ----------
          ----------
          MLC TEMP,MP20+7     MOVE CHAR TO BCE D CHAR
MP20      BCE MP30,TRTB,*     BRANCH IF A 1
          BCE                 BRANCH IF A 2
          BCE                 BRANCH IF A 3
          BCE                 BRANCH IF A 4
          BCE                 BRANCH IF A 5
          BCE                 BRANCH IF A 6
          BCE                 BRANCH IF A 7
          BCE                 BRANCH IF A 8
          BCE                 BRANCH IF A 9
          BCE                 BRANCH IF A 0
          BCE                 BRANCH IF AN L
          ----------
*     (FELL THROUGH, NOT A DIGIT)                                   *
          ----------
          ----------
MP30      SBR RAWD            SAVE BCE ADDR+1
MP31      MA  TRAN,RAWD       SUBTRACT BASIC ADDRESS
MP32      MN  RAWD,0+X3       GET TRANSLATION TO BFER
MP33      SBR X3,1+X3         INCREMENT POINTER
          B   MP10            BRANCH TO MAIN LINE
```

The Store B Register instruction will store the three-position number or address in the B Address Register into the three-position field addressed by the A field of the instruction. The machine reads the entire instruction before executing it. Thus, in the case of the SBR at MP19 in the coding above, the B Address Register will be set by the B field of the instruction itself. After execution of this instruction, index register 3 will contain the machine address of BFER, the high-order address of a five-character field. The machine performs indexing before execution. This is used to advantage in the SBR instruction at MP33 in the coding above. In this case, the contents of index register 3, incremented by 1, will be stored back into index register 3. When the processor executes a branch instruction, the address of the next sequential instruction is forced into the B Address Register. A four-character SBR instruction, such as the one at MP30 in the coding above, will save this address in the location indicated by the A field of the instruction.

The TRTB (Translate Table) literal contains all the numeric codes generated by the terminals in ascending order from right to left. The TRAN address constant is assembled as the 16,000's complement of the address of the D character at MP20. RAWD is the work area where the translation is performed.

The Branch Character Equal instructions, beginning with MP20, step through the Translate Table. Full chained BCE instructions leave the A Address Register unaltered but decrement the B Address Register. Thus, the branch address remains MP30, but the TRIB literal is stepped through. The D character is unaffected by this operation. If any BCE causes a branch, the address of the next sequential instruction

will be saved by the SBR instruction at MP30. When the address of the D character (MP20+7) is subtracted from the saved branch address, the relative location of the character in the table will be obtained. Since the table is in sequence according to numeric value, this number will also be the BCD translation of the corresponding code. The lowercase L is often used for the number 1. The relative position of the L terminal code Y is eleven, which has a low-order 1; thus, a move numeric of the low-order position of the translation is required (MP32). When the Carrier Return is sensed at the end of the line, index register 3 will point one position beyond the last translated digit in BFER. The number in BFER will be left-justified. To right-justify a number for arithmetic operations, the following instruction can be used:

```
MLC 15999+X3,ADDR
```

ADDR is the data field where arithmetic operations will occur.

Shift Characters

Shift characters are redundant in ATS. This means that any number of Upshifts and Downshifts in any order may occur on a line. Many application programs are not concerned with the shift status and simply step by shift characters. If shift status is important, an indicator is set to indicate the shift. For example, a wordmark may be set for an Upshift character and cleared when a Downshift character is encountered.

Overlays

Often an application program is too large to fit into the overlay area and must be overlaid itself. The overlaying of code generally, but not necessarily, occurs in the half-track area. The half-track area accommodates ten sectors. Thus, overlays are broken up into 900-character segments. A program with its first overlay in core might appear as shown in Figure 29.

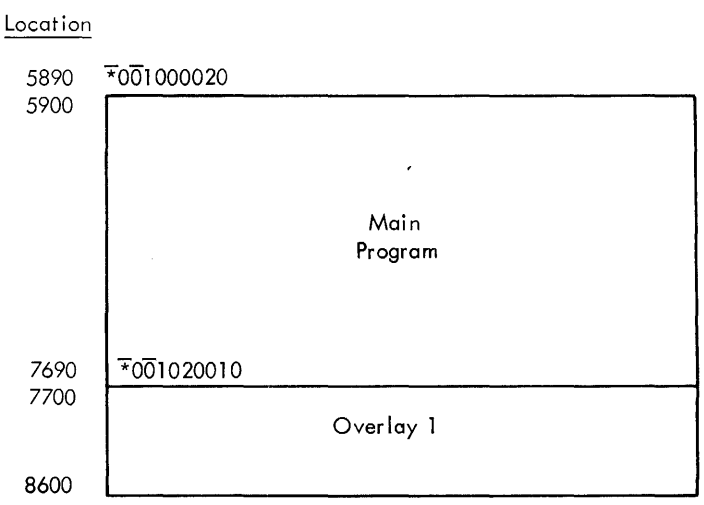


Figure 29. An application program with an overlay

The disk address of the overlay will always be the address of the main program plus an increment. The disk control field that was used to read the main program in will be in correct form immediately preceding the overlay area at core position 5890. Since an application program is 20 sectors long by definition, the first overlay will be located 20 sectors beyond the disk address of the main program. As overlays are ten sectors long, the second overlay will be ten sectors beyond the first overlay. Thus, if the main program is located at sector address 1000, the first overlay will be at sector address 1020, the second overlay at address 1030, and so on. It is a good programming practice to construct disk control fields rather than to use absolute addresses. This technique makes it possible to relocate a program on the disk without being forced to alter the code. Sample coding for a subroutine to read overlays is found in Appendix D.

Communication between overlays may be accomplished by branching to a communication table. A sample of such a communication table is found in Appendix E.

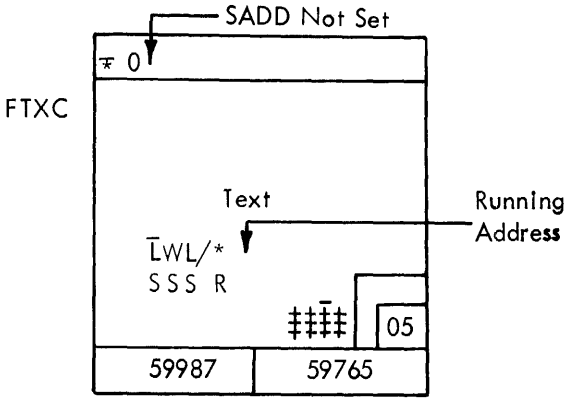
Special Messages

Special messages generated by application programs are always transmitted from a core block. Usually these messages will be less than 75 characters long and will fit into a single block. There is no guarantee that there will be room for the message in the last text block. If there is not room in this block, the last block should be written to the disk and a new block attached to the end of the chain as explained below. The amount of room remaining in the block can be found by examining the two low-order numeric positions of the address of the wordmarked Downshift of the last line. If the SKANB subroutine is used to purge dangling blocks, the numeric address of the wordmarked Downshift, minus 1, will be found in SK22. The message should begin immediately following the wordmarked Downshift of the last line and end with a record mark followed by a 2 or a 3. The running address and the saved running address for the terminal are set to the address of the beginning of the message, and the line status in the MST is set to Transmit. At this point, the application program has finished its processing and branches to FINIX, followed by the appropriate exit address constant. The message will be transmitted to the point of the record mark. The Scheduler, sensing the 2 or 3 following the record mark, will move the saved running address to the running address and set the status character to Receive-Idle. The next character keyed from the terminal will be stored over the message.

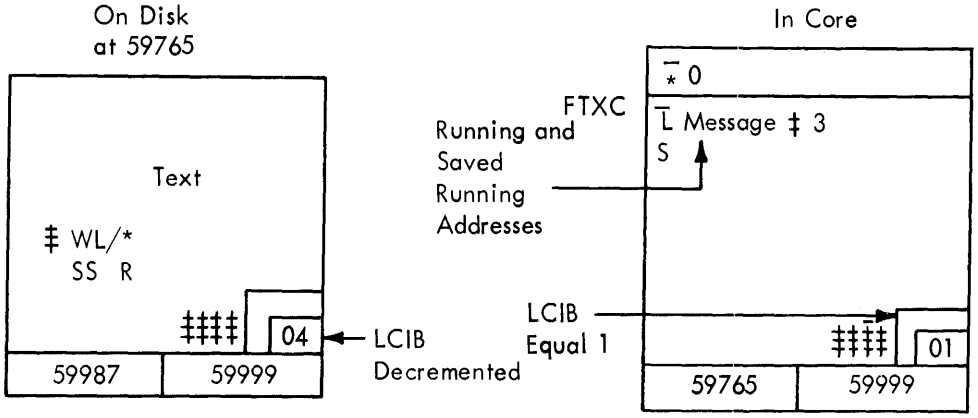
In the case where there is no room in the last block for the message, there are two methods for adding a new block. One method is to move a groupmark (ending the block) on top of the wordmarked Downshift of the last line, and to clear the wordmark. This will end that block. Since one less line will now occupy the block, the LCIB must be decremented by one. This may be accomplished by a modify address instruction:

```
MA     @I9I@,LCIB+X2
```

BLOCK IN CORE AT TIME OF PROGRAM ENTRY



BLOCKS AFTER PROGRAM COMPLETE--METHOD 1



BLOCKS AFTER PROGRAM COMPLETE--METHOD 2

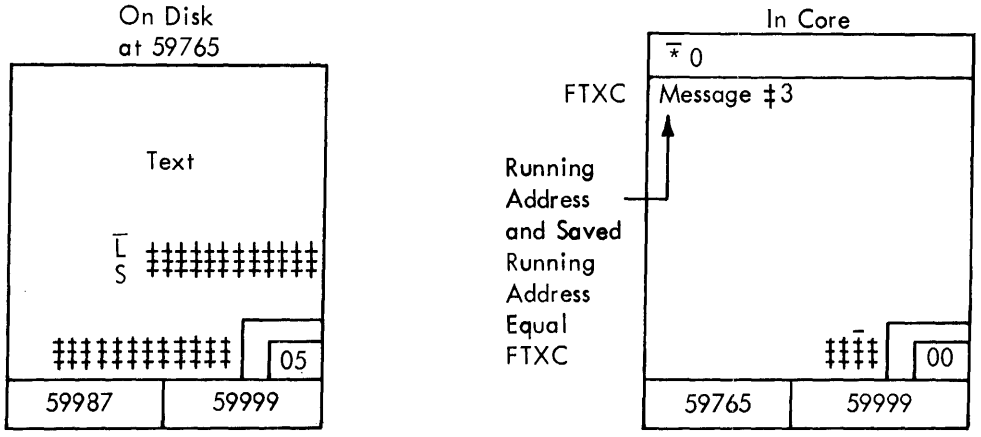


Figure 30. Methods for adding a block for message transmission

A new core block is obtained from free storage with NCBKX. THE NXBA of the last block is loaded into the SADD of the last block and the LSBA of the new block. A new disk address must be obtained via NDBKX to go into the NXBA of both blocks. The old block is now ready to be written to the disk. A wordmarked Downshift is loaded into FTXC of the new block and the LCIB set to 1. The Early Warning groupmarks must be set, including a groupmark/wordmark at LTXC+1. Sample coding for a subroutine (from the BLKOP program) that chains Working Storage to another block may be found in Appendix F.

The second technique is to leave the wordmarked Downshift in the last block. In this case, LCIB is not altered, and the LCIB in the new block is not set to 1. However, from LTXC+1 back to the character following the wordmarked Downshift, the first block must be set to a contiguous series of groupmarks. These two techniques are illustrated in Figure 30.

Output Text Streams

Output streams and messages exceeding one block constitute a program-generated text stream. The procedure for generating an output text stream is to write out the last text block, saving the five-character disk address in the LLDA field in the First Block (FBK). The text stream is then generated. The LCIB and CRAD indicators are not important. Wordmarks may appear anywhere in the text and will not affect transmission. The text in each block, except for the last block, must begin at FTXC and end with a record mark followed by a groupmark. The last block begins at FTXC and ends with a record mark followed by a 6. After the stream has been generated, the first Transmit block is read into core, the running address set to FTXC, and the status set to Transmit. There must be no other blocks pertaining to the terminal in core except the beginning Transmit block. The application program now exits. The Scheduler keeps the text flowing to the terminal until the record mark followed by a 6 is encountered. At this time, the MISAC program is called into core. MISAC obtains the address of the last text block from the LLDA field in the First Block (FBK) and sets the line status for the terminal to Receive-Idle.

Priority Interrupt

Programs with low priority and requiring long periods of processing must be able to relinquish the overlay area to programs of higher priority. The procedure for doing this begins by saving important pointers and indicators. This may be done in a core block; in the first block (FBK); in the TSLS, TSNS, or LLDA fields; or in a Working Storage block on the disk. If fields in the first block are used, they must be blanked before the final exit from the application program. Wordmarks may not be left in unexpected locations. At the time of program exit, only one block pertaining to the terminal may be in core.

The program relinquishing the overlay area sets a New Job entry for itself via SNJBX. The entry parameters are index register 1 set to ten times the terminal number, and index register 2 set to the PST number of the program. The NJB entry will be sorted into its priority level in the NJB queue. It is possible that there are already several requests in the NJB queue for the interrupting program. In this case, the entry will be sorted to the top of all entries for the program.

In cases where it is desirable to circulate through all pending requests for one program, the PROV indicator in the System Status Table should be blanked before branching to SNJBX. The New Job entry will then be sorted to the bottom of the requests for the program. This latter technique will process short requests faster than longer requests.

The NJB queue is checked by moving the NJB pointer to an index register and checking the top program in the queue.

CAUTION: The pointer must be checked to see if it is blank. The coding to accomplish this might be:

```
        BCE EX25,NJPT,          BRANCH IF BLANK POINTER
        MCW NJPT,X3             GET NJB LOCATION
        C   6+X3,PSTN           DOES PROGRAM HAVE HIGHER PRIORITY
*
        BL  EX25                NO, CONTINUE
       ----------
        Interrupt Routine
       ----------
PSTN    DCW +PPROGM             PST NUMBER OF PROGRAM
```

The interrupting program must be able to distinguish between a new task and a partially completed task. To accomplish this, an indicator is set in core. Usually, the PACT indicator (numeric) in the MST is set to 9. When the program is entered, it checks this character. If it is a 9, the program locates the status of a task in progress that it saved and continues processing. The PACT entry must be cleared before the final exit of the program. When the status character is set to Control, the MST entry is not altered by the multiplexor or any other program.

SYSTEM OUTPUT FORMATS

Some users of 1440/1460 ATS may wish to use the various possible system outputs as input to further computer processing. This section lists briefly the formats of all magnetic tape records that can be produced by ATS and that may be of interest for further processing.

Permanent Storage Tape

The Permanent Storage Tape consists of Permanent Storage half-track records read from the disk and written to tape in the Load mode with even parity. These records are 900 characters in length with a tapemark following the last record. The 900 characters on tape are identical to the 900 characters on disk with one exception: the Attention character (word separator) cannot be saved on tape in the Load mode and in order to preserve this character, it is translated to a segment mark. When ATS reads the tape, all segment marks are converted back to Attention characters.

For a detailed description of the record format, see the "Permanent Storage Tape" section of this manual. The text contained in these records is in Terminal Code with all of the ATS text stream conventions observed.

Card Image Tape

This tape is produced when a document is transmitted to terminal 97. It consists of 80-character records written in the Move mode with even parity. Tapemarks are not generated automatically but must be entered by the originating terminal. This is accomplished by typing a tapemark using the ATS multipunch feature (7 BKSP 8) in column 1 of the appropriate card.

The card images are an exact duplicate of the originating terminal's input as it appears printed on the terminal, one line per card. Beginning at the left-hand margin, every character position on the terminal represents a card column in the final output. Tab stops are set in character positions representing the beginning of data fields. These tab stop settings are associated with a special Heading mode request. (See the section "Keypunching" in IBM 1440/1460 ATS, Terminal Operator's Manual (H20-0185) for further information.)

Print with Line Numbers Tape

This tape is produced when a document is transmitted to terminal 98 and Sense Switch B is ON (up). It consists of 133-character print line image records, with a blank carriage control code in the first position, written in Move mode with even parity. A carriage control code of "1" is used in the first record of each document. A tapemark is written after each document, and two tapemarks after all documents have been written on tape.

The printout is similar to the ATTN pn0 printout at the terminal with the following exceptions: All alphabetic characters are uppercase; special characters, such as colons and brackets, do not print; the line or unit number is located at the left-hand margin; all line or unit numbers are printed.

Upper- and Lowercase Chain Print Tape

This tape is produced when a document is transmitted to terminal 96 and Sense Switch B is ON (up). The tape is written in the Load mode in even parity with a tapemark following the last record. The records may be either 133 or 15 characters long. In either case the first character of the record is a carriage control character:

Control Character	Meaning
blank	Single-space after printing
J	Immediate one-space skip
K	Immediate two-space skip
L	Immediate three-space skip
1	Page eject
+	Print and suppress space
A	Eject page after printing

The 133-character records contain formatted print images with a carriage control character. The 15-character records contain only the carriage control character. The last record, and only the last record, of each document will always contain the carriage control character A.

This printout is formatted according to the last printout requested by the originating terminal. Page width and depth are also the same as the settings for the originating terminal. The Courier typehead is assumed, which means that brackets (not ± and 1) are considered the standard characters. (For information on the way of entering characters that do not appear on the Courier 72 typehead, see the section "Upper- and Lowercase Chain Output" in IBM 1440/1460 ATS, Terminal Operator's Manual (H20-0185).)

Underscored and other overstruck character positions will be written as multiple records, all except the last of which will have a control character of "+".

Almost all 7-bit (including wordmark) character combinations are different characters on this tape. A complete listing of the characters available follows.

1440/1460 ATS SUPPORT OF 120-CHARACTER
COURIER PRINT CHAIN (Part 823380)

Chain Position	Character	Chain Graphic	BCD Code	Load Mode Card Code	Terminal Courier 72 Representation
1	Numeral One	1	1	1	1 BKSP °
2	Numeral Two	2	2	2	2
3	Numeral Three	3	21	3	3
4	Numeral Four	4	4	4	4
5	Numeral Five	5	4 1	5	5
6	Numeral Six	6	42	6	6
7	Numeral Seven	7	421	7	7
8	Numeral Eight	8	8	8	8
9	Numeral Nine	9	8 1	9	9
10	Numeral Zero	0	8 2	0	0
11	Equal	=	8 21	3-8	=
12	Apostrophe	'	84	4-8	'
13	Greater Than	>	84 1	1-4-8	) BKSP °
14	Less Than	<	842	2-4-8	(BKSP °
15	Plus	+	BA	12	+
16	Slash	/	A 1	0-1	/
17	Lowercase s	s	A 2	0-2	s
18	Lowercase t	t	A 21	0-3	t
19	Lowercase u	u	A 4	0-4	u
20	Lowercase v	v	A 4 1	0-5	v
21	Lowercase w	w	A 42	0-6	w
22	Lowercase x	x	A 421	0-7	x
23	Lowercase y	y	A8	0-8	y
24	Lowercase z	z	A8 1	0-9	z
25	Up arrow	↑	A8 2	0-2-8	@ BKSP *
26	Comma	,	A8 21	0-3-8	,
27	Left Paren.	(	A84	0-4-8	(
28	Minus (or Hyphen)	-	B	11	-
29	Vertical Line	|	A842	0-6-8	/ BKSP °
30	Lower Left Corner	⌞	A8421	0-7-8	[BKSP *
31	Lowercase j	j	B 1	11-1	j
32	Lowercase k	k	B 2	11-2	k
33	Lowercase l	l	B 21	11-3	l
34	Lowercase m	m	B 4	11-4	m
35	Lowercase n	n	B 4 1	11-5	n
36	Lowercase o	o	B 42	11-6	o
37	Lowercase p	p	B 421	11-7	p

Chain Position	Character	Chain Graphic	BCD Code	Load Mode Card Code	Terminal Courier 72 Representation
38	Lowercase q	q	B 8	11-8	q
39	Lowercase r	r	B 8 1	11-9	r
40	Exclamation Point	!	B 8 2	11-0	!
41	Dollar Sign	$	B 8 21	11-3-8	$
42	Asterisk	*	B 84	11-4-8	*
43	Lower Right Corner	┘	B 84 1	11-5-8	] BKSP *
44	Upper Left Corner	┌	B 842	11-6-8	[BKSP ,
45	Upper Right Corner	┐	B 8421	11-7-8	] BKSP ,
46	Lowercase a	a	BA 1	12-1	a
47	Lowercase b	b	BA 2	12-2	b
48	Lowercase c	c	BA 21	12-3	c
49	Lowercase d	d	BA 4	12-4	d
50	Lowercase e	e	BA 4 1	12-5	e
51	Lowercase f	f	BA 42	12-6	f
52	Lowercase g	g	BA 421	12-7	g
53	Lowercase h	h	BA8	12-8	h
54	Lowercase i	i	BA8 1	12-9	i
55	Horizontal Line	—	BA8 2	12-0	- BKSP ° (or ° BKSP-)
56	Period	.	BA8 21	12-3-8	.
57	Right Parenthesis	)	BA84	12-4-8	)
58	Crossed Lines	┼	BA84 1	12-5-8	+ BKSP °
59	Right Bracket	]	BA842	12-6-8	]
60	Left Bracket	[	8421	7-8	[
61	Superscript One	^{1}	w 1	0-5-8/1	1 BKSP *
62	Superscript Two	^{2}	w 2	0-5-8/2	2 BKSP *
63	Superscript Three	^{3}	w 21	0-5-8/3	3 BKSP *
64	Superscript Four	^{4}	w 4	0-5-8/4	4 BKSP *
65	Superscript Five	^{5}	w 4 1	0-5-8/5	5 BKSP *
66	Superscript Six	^{6}	w 42	0-5-8/6	6 BKSP *
67	Superscript Seven	^{7}	w 421	0-5-8/7	7 BKSP *
68	Superscript Eight	^{8}	w 8	0-5-8/8	8 BKSP *
69	Superscript Nine	^{9}	w 8 1	0-5-8/9	9 BKSP *
70	Superscript Zero	^{0}	w 8 2	0-5-8/0	° or 0 BKSP *
71	Number Sign	#	w 8 21	0-5-8/3-8	#
72	Quotation Mark	"	w 84	0-5-8/4-8	"
73	Greater Than or Equal	≥	w 84 1	0-5-8/5-8	) BKSP =
74	Less Than or Equal	≤	w 842	0-5-8/6-8	(BKSP =
75	Ampersand	&	wBA	0-5-8/12	&
76	Backward Slash	\	w A 1	0-5-8/0-1	/ BKSP -
77	Uppercase S	S	w A 2	0-5-8/0-2	S
78	Uppercase T	T	w A 21	0-5-8/0-3	T
79	Uppercase U	U	w A 4	0-5-8/0-4	U
80	Uppercase V	V	w A 4 1	0-5-8/0-5	V

Chain Position	Character	Chain Graphic	BCD Code	Load Mode Card Code	Terminal Courier 72 Representation
81	Uppercase W	W	w A 42	0-5-8/0-6	W
82	Uppercase X	X	w A 421	0-5-8/0-7	X
83	Uppercase Y	Y	w A8	0-5-8/0-8	Y
84	Uppercase Z	Z	w A8 1	0-5-8/0-9	Z
85	Left Arrow	←	w A8 2	0-5-8/0-2-8	@ BKSP -
86	Semicolon	;	w A8 21	0-5-8/0-3-8	;
87	Percent Sign	%	w A84	0-5-8/0-4-8	%
88	High Hyphen	¯	wB	0-5-8/11	- BKSP * or *BKSP -
89	At Sign	@	w A842	0-5-8/0-6-8	@
90	Logical And	^	w A8421	0-5-8/0-7-8	& BKSP ,
91	Uppercase J	J	wB 1	0-5-8/11-1	J
92	Uppercase K	K	wB 2	0-5-8/11-2	K
93	Uppercase L	L	wB 21	0-5-8/11-3	L
94	Uppercase M	M	wB 4	0-5-8/11-4	M
95	Uppercase N	N	wB 4 1	0-5-8/11-5	N
96	Uppercase O	O	wB 42	0-5-8/11-6	O
97	Uppercase P	P	wB 421	0-5-8/11-7	P
98	Uppercase Q	Q	wB 8	0-5-8/11-8	Q
99	Uppercase R	R	wB 8 1	0-5-8/11-9	R
100	Question Mark	?	wB 8 2	0-5-8/11-0	?
101	Underscore	_	wB 8 21	0-5-8/11-3-8	
102	Subscript One	₁	wB 84	0-5-8/11-4-8	₁ BKSP ,
103	Subscript Two	₂	wB 84 1	0-5-8/11-5-8	2 BKSP ,
104	Subscript Three	₃	wB 842	0-5-8/11-6-8	3 BKSP ,
105	Subscript n		wB 8421	0-5-8/11-7-8	n BKSP ,
106	Uppercase A	A	wBA 1	0-5-8/12-1	A
107	Uppercase B	B	wBA 2	0-5-8/12-2	B
108	Uppercase C	C	wBA 21	0-5-8/12-3	C
109	Uppercase D	D	wBA 4	0-5-8/12-4	D
110	Uppercase E	E	wBA 4 1	0-5-8/12-5	E
111	Uppercase F	F	wBA 42	0-5-8/12-6	F
112	Uppercase G	G	wBA 421	0-5-8/12-7	G
113	Uppercase H	H	wBA8	0-5-8/12-8	H
114	Uppercase I	I	wBA8 1	0-5-8/12-9	I
115	Plus or Minus	±	wBA8 2	0-5-8/12-0	+ BKSP =
116	Colon	:	wBA8 21	0-5-8/12-3-8	:
117	Bullet	•	wBA84	0-5-8/12-4-8	. BKSP °
118	Not Equal	≠	wBA84 1	0-5-8/12-5-8	/ BKSP =
119	Right Brace	}	wBA842	0-5-8/12-6-8	] BKSP -
120	Left Brace	{	w 8421	0-5-8/7-8	[BKSP -

Storage Report Tape

When a Storage Report is requested on tape, it will be written as 133-character print line image records in the Move mode with even parity. The first character will be a control code. This code is the same as the forms control character that would appear for the equivalent function in the D-modifier of the control carriage instruction. For example, the D-modifier for an immediate skip to Channel 1 is the digit 1 and it is this number that would appear in the first position of the record. The normal carriage action is an immediate skip of one line after printing, which is noted by the "/" (slash) character that normally appears in the first position of the record. The print tape is written with the same carriage control functions as would be used if the report appeared on the high-speed printer.

```
0299 ************************************************************************  MISAC
0300 *      SKANB WILL SEARCH BACK FOR A WM-ED CHAR (STEPPING FROM BLOCK*      MISAC
0301 *      TO BLOCK AS REQUIRED) BEGINNING AT ONE LESS THAN THE TERMIN-*      MISAC
0302 *      AL-S RUNNING ADDRESS.  AFTER THE SKANNED-OUT LINE HAS BEEN  *      MISAC
0303 *      PROCESSED, THE PROGRAM MAY SET-WM ON SK16+4 (TO PURGE ANY   *      MISAC
0304 *      DANGLING CORE BLKS) OR SK16+4 AND SK17+4 (TO PURGE ANY      *      MISAC
0305 *      DANGLING DISK BLKS AS WELL AS CORE BLKS).  ENTER WITH X1 EQ *      MISAC
0306 *      TO TEN TIMES THE TERMINAL NUMBER.  EXIT WITH X2 EQ ONE LESS *      MISAC
0307 *      THAN THE WM-ED CHAR.  X1 WILL BE PRESERVED.                 *      MISAC
0308 ************************************************************************  MISAC
0309 SKANB     SBR  SK13+3              SAVE RETURN ADDRESS                     MISAC
0310           MCW  RUNA+X1,X2          GET TERMINALS RUNNING ADDRESS           MISAC
0311           SBR  X2,15999+X2         STEP IT TO LAST RECEIVED CHAR           MISAC
0312 SK11      C    0+X2                SKAN BACK FOR A WM                      MISAC
0313           SBR  X2                  SAVE 1 LESS THAN WM-ED CHAR             MISAC
0314           BCE  SK14,X2-1,9         BRANCH IF OVERSHOT BLOCK                MISAC
0315           MN   X2,SK22             GET CHARACTER ADDR                      MISAC
0316           MN                                                               MISAC
0317           C    SK21,SK22           IS IT LOWER THAN FTXC-1                 MISAC
0318           BL   SK15                YES. SKAN OVERSHOT VALID TEXT.          MISAC
0319           CW   SK16+4,SK17+4       CLEAR SWITCHES                          MISAC
0320 SK13         KGO                   ENABLE INTERRUPTS AND RETURN            MISAC
                . R
0321 SK14      SBR  X2,1+X2             STEP TO ORIGINAL BLOCK                  MISAC
0322 SK15      MN   @00@,X2             CLEAR X2 CHAR ADDR                      MISAC
0323           MN                                                               MISAC
0324 SK16      BIN  SK17,(              BRANCH TO PURGE CORE BLOCKS             MISAC
0325           MCW  X2,X3               MOVE BLK ADDR TO X3                     MISAC
0326           B    RTRVX               BRANCH FOR PREVIOUS BLK                 MISAC
0327           BCE  SK11,96,            BRANCH IF BLOCK IN CORE                 MISAC
0328           B    RTBLK               BRANCH TO READ FROM DISK                MISAC
0329           MZ   X2,CRAD+X3          HSKP FORMER CRAD POINTER                MISAC
0330           MZ                                                               MISAC
0331           MCW                                                              MISAC
0332           B    SK11                BRANCH TO CONTINUE SKAN.                MISAC
0333 SK17      BIN  SK19,(              BRANCH TO PURGE DISK BLOCKS TOO         MISAC
0334           BCE  SK13,CRAD-2+X2,     IF NO CRAD, DO NOT STEP FURTHER         MISAC
0335           MCW  CRAD+X2,X3          GET ADDR OF NXT CORE BLK BACK           MISAC
0336           B    HDFSX               BRANCH TO PURGE IT                      MISAC
0337              KL                    DISABLE INTERRUPTS                      MISAC
                 S
0338           SBR  X2,LTXC+X3          SET X2 TO LTXC IN PURGED BLOCK          MISAC
0339           B    SK11                BRANCH TO SKAN IT                       MISAC
0340 SK19      SBR  X3,NXBA+X2          SET X3 TO A(NXBA/THIS BLK)              MISAC
0341           B    PDBKX               BRANCH TO PURGE DISK ADDR               MISAC
0342           MZ   CRAD+X2,X2          GET ADDR OF NEXT BLK BACK               MISAC
0343           MZ                                                               MISAC
0344           MCW                                                              MISAC
0345           MCW  SADD+X2,NXBA+X2     MOVE SADD OF THAT BLK TO NXBA           MISAC
0346           B    HDFSX               PURGE CORE BLK                          MISAC
0347           SBR  X2,LTXC+X2          SET X2 TO LTXC IN NEW BLK               MISAC
0348           B    SK11                BRANCH TO SKAN IT                       MISAC
0349 SK21      DCW  +FTXC-1             USED TO DETECT BLK OVERSHOOT            MISAC
0350 SK22      DCW  =2                  TEMP STORAGE FOR CHAR ADDR              MISAC
```

APPENDIX B: SAMPLE CODING FOR A RTBLK SUBROUTINE

```
0397 ********************************************************************** MISAC
0398 *      RTBLK IS ENTERED BY STEPB OR SKANB WHEN RTRVX SETS UP X2 TO *  MISAC
0399 *      READ A BLK.  ENTER AFTER RTRVX TURNS ON UNEQUAL LATCH.  EXIT*  MISAC
0400 *      WITH BLK IN CORE, LTXC IN X2, SCNT EQ TO X3 BLK.  X1 WILL   *  MISAC
0401 *      BE PRESERVED.  X3 WILL BE SET TO XXX00.                     *  MISAC
0402 ********************************************************************** MISAC
0403 RTBLK      SBR  RT53+3              SAVE RETURN ADDRESS               MISAC
0404            MCW  X1,RT52+6           SAVE X1                           MISAC
0405            MCW  X2,RT49             X2 TO IOAR                        MISAC
0406            MZ   X3,RT51+6           X3 BLK ADDR TO -SBR- INST.        MISAC
0407            MZ                                                         MISAC
0408            MCW                                                        MISAC
0409            B    IOCNX               BRANCH TO READ BLK                MISAC
0410 RT48       DCW  =1                  CONDITION COMMUNICATION           MISAC
0411            DCW  @1@                 READ, 1 BLK, LOW PRIORITY         MISAC
0412 RT49       DCW  =3                  ADDRESS OF BLK (IOAR)             MISAC
0413            MCW  RT49,X2             MOVE BLK ADDR TO X2               MISAC
0414 RT51       SBR  SCNT+X2,0           SET SCNT TO NEXT BLK FWD IN CORE  MISAC
0415            SBR  X3                  RESTORE X3                        MISAC
0416            SBR  X2,LTXC+X2          SET X2 TO LTXC IN BLK             MISAC
0417 RT52       SBR  X1,0                RESTORE X1                        MISAC
0418 RT53       BCE  0,RT48,             RETURN TO PROGRAM IF I/O OK       MISAC
0419            H                        HALT. CATASTROPHIC ERROR.         MISAC
```

APPENDIX C: SAMPLE CODING FOR A SSPRT SUBROUTINE

```
0361 ********************************************************************************  MISAC
0362 *      SSPRT STEPS THE X2 POINTER FWD.  ENTER WITH X2 EQ TO A BLK   *  MISAC
0363 *      ADDR.  EXIT WITH X2 EQ TO FTXC IN THE NEXT BLK FORWARD.      *  MISAC
0364 *      X1 AND X3 ARE PRESERVED.                                     *  MISAC
0365 ********************************************************************************  MISAC
0366 SSPRT      SBR   SS13+3              SAVE RETURN ADDRESS                    MISAC
0367            MN    @00@,X2             CLEAR CHAR ADDR IN X2                  MISAC
0368            MN                                                               MISAC
0369            MCW   SCNT+X2,X2          MOVE NEXT BLK FWD ADDR TO X2           MISAC
0370            SBR   X2,FTXC+X2          SET X2 TO FTXC IN NEW BLK              MISAC
0371 SS13       B     0                   RETURN TO MAIN PROGRAM                 MISAC
```

```
0354 ********************************************************************** KLCUT
0355 *     RFOVY WILL READ THE FIRST OVERLAY. ALL INDEXES ARE             * KLCUT
0356 *     PRESERVED                                                      * KLCUT
0357 ********************************************************************** KLCUT
0358 RFOVY     SBR  RD20+3              SAVE RETURN ADDRESS                KLCUT
0359           MCW  @2@,RF11            MOVE 1ST OVLY INCREMENT TO CHECK   KLCUT
0360 RF08      MCW  OVAREA-5,RF10       MOVE IN BASIC DSK ADDRS FOR CHECK  KLCUT
0361           A    RF11,RF10           INCREMENT BASIC ADDRS FOR OVRLY    KLCUT
0362           C    HFTRAR-5,RF10       IS OVRLY ALREADY IN                KLCUT
0363           BE   RD20                YES, ABORT READ                    KLCUT
0364           MCW  RF10,HFTRAR-5       NO, SET DISK ADDRESS FOR OVRLY     KLCUT
0365           MCW  @!@,RE17            SET IOOP FOR READ, HI PRIORITY     KLCUT
0366           SBR  RE18,HFTRAR-10      SET IOAR                           KLCUT
0367           B    RE12                BRANCH FOR I/O                     KLCUT
0368 RF10      DCW  =3                  CHECK REGISTER FOR OVRLY ADDRS     KLCUT
0369 RF11      DCW  =1                  INCREMENTAL REGISTER FOR OVRLY     KLCUT
0370 ********************************************************************** KLCUT
0371 *     RSOVY WILL READ THE SECOND OVERLAY. ALL INDEXES ARE            * KLCUT
0372 *     PRESERVED                                                      * KLCUT
0373 ********************************************************************** KLCUT
0374 RSOVY     SBR  RD20+3               SAVE RETURN ADDRESS               KLCUT
0375           MCW  @3@,RF11             MOVE IN 2ND OVRLY INCREMENT       KLCUT
0376           B    RF08                 BRANCH FOR I/O                    KLCUT
0377 ********************************************************************** KLCUT
0378 *     RTHVY WILL READ THE THIRD OVERLY. ALL INDEXES ARE              * KLCUT
0379 *     PRESERVED                                                      * KLCUT
0380 ********************************************************************** KLCUT
0381 RTHVY     SBR  RD20+3               SAVE RETURN ADDRESS               KLCUT
0382 RF01      MCW  @4@,RF11            MOVE IN 3RD OVRLY INCREMENT        KLCUT
0383           B    RF08                 BRANCH FOR IO                     KLCUT
0384 ********************************************************************** KLCUT
0385 *     RFTOVY   WILL READ THE FOURTH OVERLAY.  ALL INDEXES ARE        * KLCUT
0386 *     PRESERVED.                                                     * KLCUT
0387 ********************************************************************** KLCUT
0388 RFTVY     SBR  RD20+3              SAVE RETURN ADDRESS                KLCUT
0389           MCW  @5@,RF11            MOVE IN 4TH OVRLY INCREMENT        KLCUT
0390           B    RF08                BRANCH FOR I/O                     KLCUT
0391 ********************************************************************** KLCUT
0392 *     RFFVY WILL READ THE FIFTH OVERLAY.  ALL INDEXES ARE            * KLCUT
0393 *     PRESERVED                                                        KLCUT
0394 ********************************************************************** KLCUT
0395 RFFVY     SBR  RD20+3              SAVE RETURN ADDRESS                KLCUT
0396           MCW  @6@,RF11            MOVE IN 5TH OVRLY INCREMENT        KLCUT
0397           B    RF08                BRANCH FOR I/O                     KLCUT
0398 ********************************************************************** KLCUT
0399 *     RSXVY WILL READ THE SIXTH OVERLAY.  ALL INDEXES ARE            * KLCUT
0400 *     PRESERVED.                                                     * KLCUT
0401 ********************************************************************** KLCUT
0402 RSXVY     SBR  RD20+3              SAVE RETURN ADDRESS                KLCUT
0403 RF02      MCW  @7@,RF11            SET 6TH OVLY INCREMENT             KLCUT
0404           B    RF08                BRANCH TO READ IN OVERLAY          KLCUT
```

APPENDIX E: SAMPLE TABLE FOR COMMUNICATION BETWEEN OVERLAYS

```
0127 ********************************************************************************  KLCUT
0128 *       THE BRANCH TABLE PROVIDES THE COMMUNICATION BETWEEN                    *  KLCUT
0129 *       OVERLAYS.                                                              *  KLCUT
0130 ********************************************************************************  KLCUT
0131 INTRP      B     RSOVY              READ SECOND OVERLAY                    P     KLCUT
0132            B     IN12               BRANCH TO INTERPRET                    G     KLCUT
0133 FORMT      MCW   @3@,INMD           SET FORMAT ERROR FLAG                        KLCUT
0134            B     FNISH              BRANCH TO STOP PROCESSING                    KLCUT
0135 OVERF      MCW   @4@,INMD           SET OVERFLOW FLAG                            KLCUT
0136            B     FNISH              BRANCH TO STOP PROCESSING                    KLCUT
0137 WBALL      MCW   @5@,INMD           SET WRONG BALL FLAG                          KLCUT
0138            B     FNISH              BRANCH TO STOP PROCESSING                    KLCUT
0139 ENDOS      MCW   @6@,INMD           SET END OF STORAGE FLAG                      KLCUT
0140            B     FNISH              BRANCH TO CONCLUDE PROCESSING                KLCUT
0141 ILEXP      MCW   @7@,INMD           SET INVALID EXPONENT FLAG                    KLCUT
0142            B     FNISH              BRANCH TO CONCLUDE PROCESSING                KLCUT
0143 ZEDIV      MCW   @8@,INMD           SET ZERO DIVISOR FLAG                        KLCUT
0144 FNISH      B     RFTVY              READ FOURTH OVERLAY                          KLCUT
0145            B     FI10               BRANCH TO CONCLUDE JOB                       KLCUT
0146 STORE      B     RTHVY              READ THIRD OVERLAY                           KLCUT
0147            B     SO18               BRANCH TO STORE                              KLCUT
0148 SO12       B     RSOVY              READ SECOND OVERLAY                          KLCUT
0149 SO14       B     0                  RETURN                                       KLCUT
0150 GETVB      SBR   SO14+3             SAVE RETURN ADDRESS                          KLCUT
0151            B     RTHVY              READ THIRD OVERLAY                           KLCUT
0152            B     GE10               BRANCH TO GET NUMBER                         KLCUT
0153 SCANM      SBR   SO14+3             SAVE RETURN ADDRESS                          KLCUT
0154            B     RTHVY              READ THIRD OVERLAY                           KLCUT
0155            B     SO16               BRANCH TO STORE TEMP TOTAL                   KLCUT
0156 ROOTC      B     RFFVY              READ FIFTH OVERLAY                           KLCUT
0157            B     RO14               CONTINUE ITERATING                           KLCUT
0158 PAUSE      SBR   PA01+3             SAVE RETURN ADDRESS                          KLCUT
0159            BCE   PA01,NJPT,         BRANCH IF NO NEW JOBS                        KLCUT
0160            MCW   NJPT,X1            MOVE LIST POINTER TO INDEX                   KLCUT
0161            C     6+X1,TERM+3        DOES NJB HAVE HI PRIORITY                    KLCUT
0162            BH    PA03               YES, BRANCH                                  KLCUT
0163            C     6+X1,PA05          IS EUNIT WAITING                             KLCUT
0164            BE    PA03               YES, INTERRUPT PROCESSING                    KLCUT
0165 PA01       B     0                  NO, RETURN                                   KLCUT
0166 PA03       B     RSXVY              READ SIXTH OVERLAY                           KLCUT
0167            B     PA09               PERFORM INTERRUPT PROCESSING                 KLCUT
0168 PA05       DCW   +PEUNIT            PST NUMBER OF EUNIT                          KLCUT
```

APPENDIX F: SAMPLE CODING FOR A SUBROUTINE TO CHAIN WORKING STORAGE TO ANOTHER BLOCK

```
0563 ********************************************************************************  BLKOP
0564 *      CHAIN CHAINS WORKING STORAGE TO ANOTHER BLOCK                           *  BLKOP
0565 ********************************************************************************  BLKOP
0566 CHAIN      SBR   CH20+3              SAVE RETURN ADDRESS                          BLKOP
0567            SBR   BL50,2+X2           SET SVRA FOR END OF STORAGE                  BLKOP
0568            SBR   BL49,SYSMSR         ASSUME END OF STORAGE MESSAGE                BLKOP
0569            B     CNXTO               CLEAR X2 NUMERICS                            BLKOP
0570            SW    LTXC+1+X2           ASSUME END OF STORAGE                        BLKOP
0571            SBR   X3,NXBA+X2          SET NDBKX PARAMETER                          BLKOP
0572            B     NDBKX               BRANCH FOR A DISK BLOCK                      BLKOP
0573            B     BL245               NONE, TYPE END OF STORAGE                    BLKOP
0574            CW    LTXC+1+X2           CLEAR GMWM                                   BLKOP
0575            MCW   BL50,X3             GET WMDWNSHFT ADDRS IN INDEX                 BLKOP
0576            MCW   GMWM,15999+X3       END CURRENT BLOCK WITH A GM                  BLKOP
0577            CW    15999+X3            NO GMWMS ALLOWED                             BLKOP
0578            MA    @191@,LCIB+X2       DECREMENT LINE COUNT BY 1                    BLKOP
0579            B     WRBLK               WRITE OUT OLD BLOCK                          BLKOP
0580            B     NCBKX               GET A NEW CORE BLOCK                         BLKOP
0581            LCA   NXBA+X2,NXBA+X3     GIVE BLOCK NEW ADDRESS                       BLKOP
0582            LCA   SADD+X2             SET BKWD CHAINING                            BLKOP
0583            LCA   X2                  AND CRAD                                     BLKOP
0584            MCW   GMWM+1              AND EW GMS                                   BLKOP
0585            MCW   GMWM+1                                                           BLKOP
0586            MCW   X3,BL50             SAVE X3 POINTER                              BLKOP
0587            MCW   X2,X3               GIVE OLD BLOCK TO X3                         BLKOP
0588            B     FTFSX               AND PURGE IT                                 BLKOP
0589            MCW   BL50,X2             GIVE BLOCK ADDRESS TO X2                     BLKOP
0590            LCA   NXBA+X2,SADD+X2     SET SADD IN NEW BLOCK                        BLKOP
0591            MN    @01@,LCIB+X2        SET LINE COUNT TO 1 IN BLK                   BLKOP
0592            MN                                                                     BLKOP
0593            LCA   @L@,FTXC+X2         START BLK WITH A WMDWNSHFT                   BLKOP
                        S
0594 CH20       B     0                   RETURN                                       BLKOP
```

Printed in U.S.A. H20-0228-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601