PART V

External Storage and File Processing

17

EXTERNAL STORAGE

OBJECTIVE

To explain the design and uses of magnetic tape and disk storage devices.

A file, or data set, is a collection of related data records. Most data processing applications involve data files of such volume that they require large external magnetic tape and disk storage devices. Tape and disk provide mass external storage, extremely fast input/output, reusability, and records of almost any length.

This chapter introduces the various file organization methods and describes the architecture for magnetic tape and disk drives. The next three chapters cover the processing of files.

FILE ORGANIZATION METHODS

In any system, a set of related records is arranged into a file and organized according to the way in which programs are intended to process them. Once you create a file under a particular organization method, all programs that subsequently process the file must do so according to the requirements of the method. Let's take a brief look at the most common organization methods.

Sequential File Organization

Under sequential organization, records are stored one after another. They may be in ascending sequence (the usual) or descending sequence by a particular key or keys (control word), such as customer number or employee number within department, or, contrary to what the name sequential organization implies, records need not be in any particular sequence.

Transaction records may be accumulated into a file in random sequence. You can either use the file in its unsorted form for random updating of a master file or sort it into a specified order for sequential updating.

You can store a sequentially organized file on any type of device and for any type of file, such as master, transaction, and archival.

Indexed Sequential File Organization

Indexed sequential organization for master files lets you access records in ascending sequence and also supports indexes that enable you to access any record randomly by key, such as customer number.

Direct File Organization

Direct file organization facilitates direct access of any record in a master file. The main advantage is that this method provides fast accessing of records and is thus particularly useful for online systems.

Virtual Storage Access Method

Virtual storage access method (VSAM) supports three organization types. Entry-sequenced is equivalent to sequential organization, key-sequenced is equivalent to indexed, and relative-record is equivalent to direct.

Disk storage devices, but not tape, support indexed sequential, VSAM, and direct organization. Chapters 18, 19, and 20 cover sequential, VSAM, and indexed sequential, respectively.

ACCESS METHODS

An access method is the means by which the system performs input/output requests. The methods depend on the file organization and the type of accessing required. DOS supports four methods and OS supports seven.

File Organization Method	DOS	OS
Sequential	SAM	QSAM *or* BSAM
Virtual	VSAM	VSAM
Indexed	ISAM	QISAM *or* BISAM
Direct	DAM	BDAM
Partitioned	-	BPAM

PROCESSING OF EXTERNAL STORAGE DEVICES

Major similarities between tape and disk are that records may be of virtually any length, of fixed or variable length, and clustered together into one or more records per block.

There are, however, two major differences in processing tape and disk. First, each time you read or write, the tape drive starts, transfers the data, and then stops, whereas a disk drive rotates continuously. Second, whenever you update (add, change, or delete) records on tape, you rewrite the entire changed file on another reel, whereas you can update disk records directly, in place.

Identification of External Devices

Both disk and tape have unique ways of identifying their contents to help in locating files and in protecting them from accidental erasure.

Tape file identification. At the beginning of the tape reel is a volume label, which is a record that identifies the reel being used. Immediately preceding each file on the tape is a *header label*, which describes the file that follows. This record contains the name of the file (for example, INVENTORY FILE) and the date the file was created. Following the header label are the records that comprise the data file.

The last record following the file is a *trailer label*, which is similar to the header label but also contains the number of blocks written on the reel. The operating system automatically handles the header and trailer labels.

Disk file identification. To keep track of all the files it contains, a disk device uses a special directory (*volume table of contents*, VTOC) at the beginning of its storage area. The directory includes the names of the files, their locations on disk, and their present status.

Packed and Binary Data

Tape and disk records can contain numeric fields defined as zoned, binary, or packed. Packed format involves two digits per byte plus a half-byte for the sign, such as

```
PAYMENT DS PL4
```

In this case, the field length is 4 bytes, stored as dd | dd | dd | ds, where d is a digit and s is the sign.

If the field is defined as binary, watch out for erroneous alignment of the field when you read it into main storage. The following binary fields are both 4 bytes long:

Aligned on a fullword boundary:	PAYMENT1	DS	F
Not aligned on a boundary:	PAYMENT2	DS	FL4

The assembler automatically aligns PAYMENT1 on a fullword boundary, whereas the assembler defines PAYMENT2 at its proper (unaligned) location.

Unblocked and Blocked Records

Disk and tape devices recognize *blocks* of data, which consist of one or more records. A blank space, known as an interblock gap (IBG), separates one block from another. The length of an IBG on tape is 0.3 to 0.6 inches depending on the device, and the length of an IBG on disk varies by device and by track location. The IBG has two purposes: (1) to define the start and end of each block of data and (2) to provide space for the tape when the drive stops and restarts for each read or write of a block.

Records that are stored one to a block are called *unblocked*. As shown in Fig. 17-1(a), following each block is an IBG.

To reduce the amount of tape and disk storage and to speed up input/output, you may specify a *blocking factor*, such as three records per block, as shown in

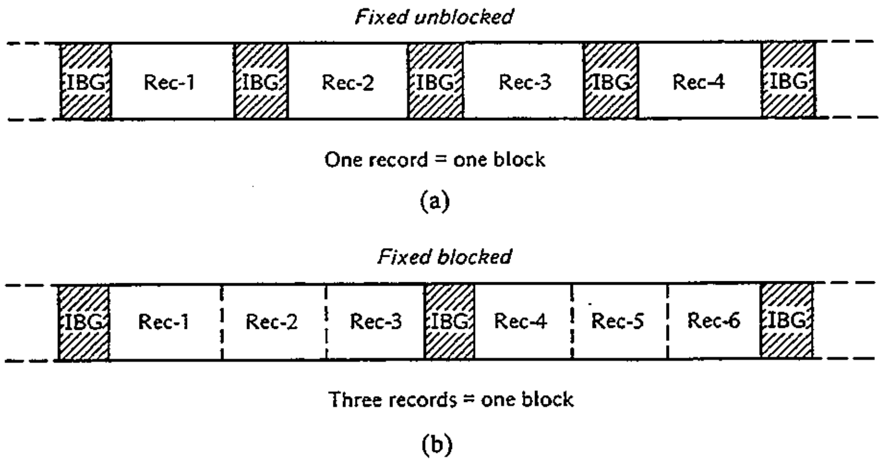


Figure 17-1 (a) Unblocked records. (b) Blocked records.

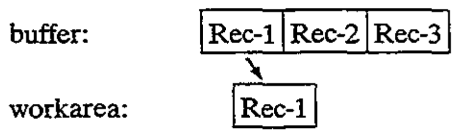
Fig. 17-1(b). In this format, the system writes an entire block of three records from main storage onto the device. Subsequently, when the system reads the file, it reads the entire block of three records from the device into storage. All programs that subsequently read the file must specify the same record length and block length.

Blocking records makes better use of disk and tape storage but requires a larger buffer area in main storage to hold the block.

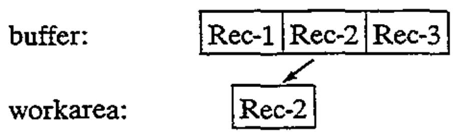
Input Buffers

The action of an input operation depends on whether records are unblocked or blocked. If unblocked, the operation transfers one record (block) at a time from the device into the input/output buffer in your program.

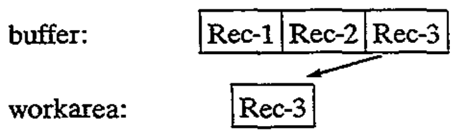
The following example of blocked records assumes three records per block. Initially, the input operation transfers the first block from the device into the buffer (I/O area) in your program and delivers the first record to your program's workarea:



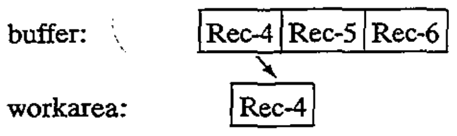
For the second input executed, the operation does not have to access the device. Instead, it simply delivers the second record from the buffer to your program's workarea:



And for the third input executed, the operation delivers the third record from the buffer to your program's workarea:



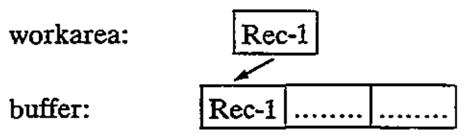
While the program processes the third record in the workarea, the system can read ahead and transfer the second block from the device into the buffer in your program. For the fourth input executed, the operation delivers the first record from the buffer to your program's workarea:



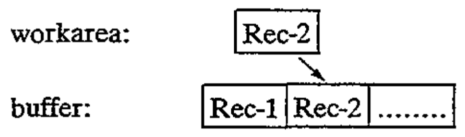
Output Buffers

The action of an output operation depends on whether records are unblocked or blocked. If unblocked, the output operation transfers one record (block) at a time from your workarea to the buffer in your program and then to the output device.

The following example of blocked records assumes three records per block. The first output operation writes the record in the workarea to the first record location in the output buffer:



No actual physical writing to the output device occurs at this time. The second output operation writes the record in the workarea to the second record location in the buffer:



Similarly, the third output operation writes the record in the workarea to the third record location in the buffer. Now that the buffer is full, the system can physically write the contents of the buffer, the block of three records, to the external device.

The CLOSE operation automatically writes the last block of data, which may validly contain fewer records than the blocking factor specifies.

Fixed-Length and Variable-Length Records

Records and blocks may be fixed in length, where each has the same length throughout the entire file, or variable in length, where the length of each record and the blocking factor are not predetermined. There are five formats:

1. *Fixed, unblocked:* one record of fixed length per block
2. *Fixed, blocked:* more than one fixed-length record per block
3. *Variable, unblocked:* one variable-length record per block
4. *Variable, blocked:* more than one variable-length record per block
5. *Undefined:* contents of no defined format (Not all systems support this format.)

MAGNETIC TAPE STORAGE

The magnetic tape used in a computer system is similar to the tape used by conventional audiotape recorders; both use a similar coating of metallic oxide on flexible plastic, and both can be recorded and erased. Its large capacity and its reusability make tape an economical storage medium.

Data records on tape are usually, but not necessarily, stored sequentially, and a program that processes the records starts with the first record and reads or writes each record consecutively.

The main users of tape are installations such as department stores and utilities that require large files that they process sequentially. Many installations use disk for most general processing and use tape for backing up the contents of the disk master files at the end of each workday. Consequently, if it is necessary to rerun a job because of errors or damage, backup tapes are always available.

Characteristics of Tape

The most common width of a reel of magnetic tape is 1/2 inch, and its length ranges from 200 feet to the common 2,400 feet, with lengths as long as 3,600 feet. A tape drive records data as magnetic bits on the oxide side of the tape.

Storage format. Data is stored on tape according to tracks. The tape in Fig. 17-2 shows nine horizontal tracks, each of which represents a particular bit position. Each vertical set of 9 bits constitutes a byte, of which 8 bits are for data and 1 bit is for parity.

```
     ____________________________
4    0 0 0 ...
6    0 0 0 ...
0    1 1 1 ...
1    0 1 0 ...
2    1 1 0 ...
P    1 0 1 ...   (parity track)
3    0 0 1 ...
7    0 0 0 ...
5    0 0 0 ...
     ____________________________
     | | |
     bytes
```

Figure 17-2 Data on tape.

As you can see, the tracks for each of the bits are not in the expected sequence. The tracks for bits 4 and 5, the least used, are in the outer area where the tape is more easily damaged. The first byte, on the left, would appear in main storage as follows:

Bit value:	1 0 1 0 0 0 0 0 1
Bit number:	0 1 2 3 4 5 6 7 P

Storage density. Tape density is measured by the number of stored characters, or bytes, per inch (bpi), such as 800, 1,600, or 6,250 bpi. Therefore, a 2,400-foot reel with a recording density of 1,600 bpi could contain 46 million bytes, which is equal to over a half-million 80-byte records.

Double-density tape stores data on 18 tracks, representing 2 bytes for each set of 18 vertical bits.

Tape speed. Tape read/write speeds vary from 36 to 200 or more inches per second. Thus a tape drive that reads 1,600 bpi records at 200 inches per second would be capable of reading 320,000 bytes per second. Other high-speed cartridge drives transfer data at up to 3 million bytes per second.

Tape markers. A reflective strip, called a load point marker, located about 15 feet from the beginning of a tape reel, indicates where the system may begin reading and writing data. Another reflective strip, an end-of-tape marker, located about 14 feet from the end of the reel, warns the system that the end of the reel is near and that the system should finish writing data. Both the load point marker and the end-of-tape marker are on the side of the tape opposite the recording oxide.

Tape File Organization

A file or data set on magnetic tape is typically stored in sequence by control field or key, such as inventory number. For compatibility with disks, a reel of tape is know as a *volume*. The simplest case is a one-volume file, in which one file is entirely and exclusively stored on one reel (volume).

An extremely large file, known as a *multivolume file*, requires more than one reel. Many small files may be stored on a *multifile volume*, one after the other, although you may have to rewrite the entire reel just to update one of the files.

Unblocked and Blocked Tape Records

As an example of the effect of blocking records on tape, consider a file of 1,000 records each 800 bytes long. Tape density is 1,600 bytes per inch, and each IBG is 0.6 inches. How much space does the file require given (a) unblocked records and (b) a blocking factor of 5? Calculate the size of a record of 800 bytes as 800 ÷ 1,600 = 0.5 inches.

(a) Unblocked records

One block = one record = 800 bytes	
Length of one block = 800 bytes/1,600 bpi =	0.5″
Length of one IBG =	0.6
Space required for one block	1.1″
Space required for file = 1,000 blocks × 1.1″ = 1,100″	

(b) Blocked records

One block = five records = 4,000 bytes	
Length of one block = 4,000 bytes/1,600 bpi =	2.5″
Length of one IBG =	0.6
Space required for one block	3.1″
Space required for file = 200 blocks × 3.1″ = 620″	

As can be seen, the blocked records require considerably less space because there are fewer IBGs.

Standard Labels

Under the various operating systems, tape reels require unique identification. Each reel, and each file on a reel, usually contains descriptive standard labels supported by the operating systems (1) to uniquely identify the reel and the file for each program that processes it and (2) to provide compatibility with other IBM systems and (to some degree) with systems of other manufacturers.

Installations typically use standard labels. Nonstandard labels and unlabeled tapes are permitted but are not covered in this text. The two types of standard labels are volume and file labels. Figure 17-3 illustrates standard labels for one file on a volume, a multivolume file, and a multifile volume. In the figure, striped lines indicate IBGs, and TM (for tape mark) is a special marker that the system writes to indicate the end of a file or the end of the reel.

Volume Labels

The volume label is the first record after the load point marker and describes the volume (reel). The first 3 bytes contain the identification VOL. Although some systems support more than one volume label, this text describes only the common situation of one label.

On receipt of a new tape reel, an operator uses an IBM utility program to write a volume label with a serial number and a temporary header file label. When

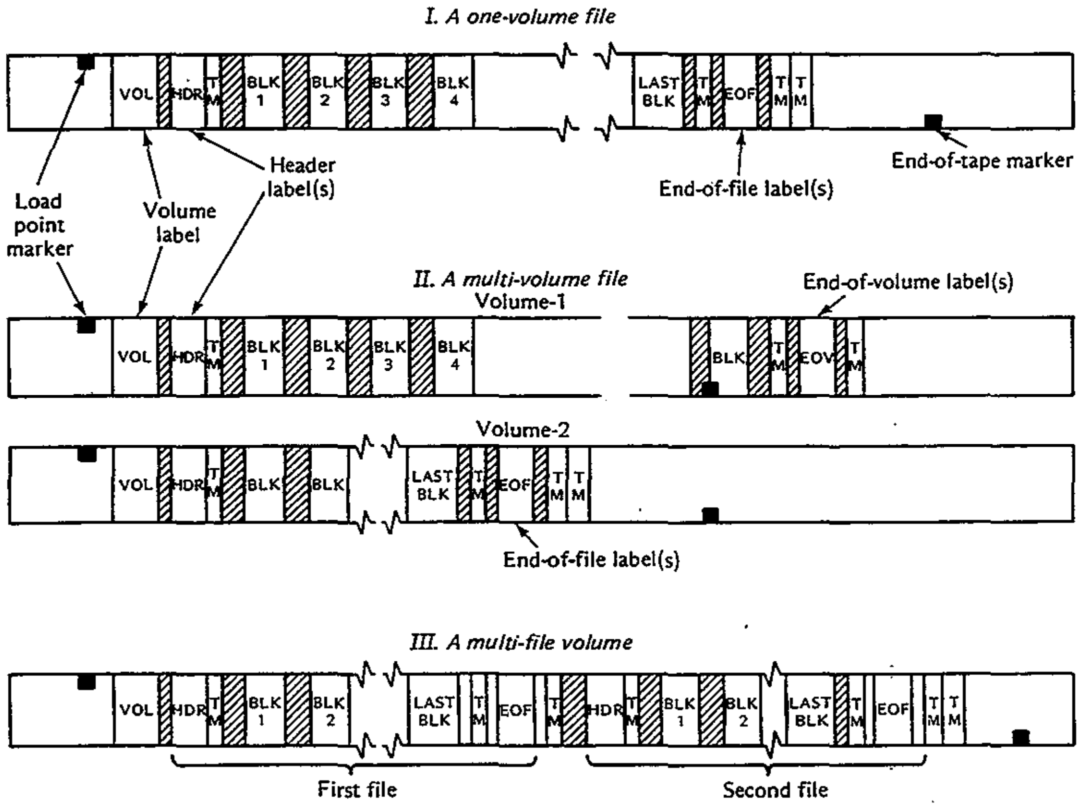


Figure 17-3 Magnetic tape standard labels.

subsequently processing the reel, the system expects the volume label to be the first record. It checks the tape serial number against the number supplied by the job control command, TLBL under DOS or DD under OS.

The following describes each field in the 80-byte standard volume label:

POSITIONS	NAME	DESCRIPTION
01–03	Label identifier	Contains VOL to identify the label.
04	Volume label number	Some systems permit more than one volume label; this field contains their numeric sequence.
05–10	Volume serial number	The permanent unique number assigned when the reel is

POSITIONS	NAME	DESCRIPTION
		received. (The number also becomes the file serial number in the header label.)
11	Volume security code	A special security code, supported by OS.
12–41	Unused	Reserved.
42–51	Owner's identification	May be used under OS to identify the owner's name and address.
52–80	Unused	Reserved.

File Labels

A tape volume contains a file of data, part of a file, or more than one file. Each file has a unique identification to ensure, for example, that the system is processing the correct file and that the tape being used to write on is validly obsolete. Two file labels for each file, a header label and a trailer label, provide this identification.

Header label. A header label precedes each file. If the file requires more than one reel, each reel contains a header label, numbered from 001. If a reel contains more than one file, a header label precedes each file.

The header label contains HDR in the first 3 bytes, the file identification (such as CUSTOMER RECORDS), the date the file may be deleted, and so forth. The system expects a header label to follow the volume label immediately and checks the file identification, date, and other details against information supplied by job control.

OS supports two header labels, HDR1 and HDR2, with the second label, also 80 bytes, immediately following the first. Its contents include the record format (fixed, variable, or undefined), block length, record length, and density of writing on the tape.

Trailer label. A trailer label is the last record of every file. (OS supports two trailer labels.) The first 3 bytes contain EOV if the file requires more than one reel and the trailer label is the end of a reel but not end of the file. The first 3 bytes contain EOF if the trailer label is the end of the file.

The trailer label is otherwise identical to the header label except for a block count field. The system counts the blocks as it writes them and stores the total in the trailer label. Subsequently, when reading the reel, the system counts the blocks and checks its count against the number stored in the trailer label.

The following describes each field in the standard file label for both header and trailer labels.

POSITIONS	NAME	DESCRIPTION
01–03	Label identifier	Contains HDR if a header label, EOF if the end of a file, or EOV if the end of a volume.
04	File label number	Specifies the sequence of file labels for systems that support more than one. OS supports two labels each for HDR, EOF, and EOV.
05–21	File identifier	A unique name that describes the file.
22–27	File serial number	The same identification as the volume serial number for the first or only volume of the file.
28–31	Volume sequence number	The sequence of volume numbers for multivolume files. The first volume for a file contains 0001, the second 0002, and so on.
32–35	File sequence number	The sequence of file numbers for multifile volumes. The first file in a volume contains 0001, the second 0002, and so on.
36–39	Generation number	Each time the system rewrites a file, it increments the generation number by 1 to identify the edition of the file.
40–41	Version number of generation	Specifies the version of the generation of the file.
42–47	Creation date	The year and day when the file was written. The format is byyddd, where b means blank.

POSITIONS	NAME	DESCRIPTION
48–53	Expiration date	The year and day when the file may be overwritten. The format is byyddd, where b means blank.
54	File security code	A special security code used by OS.
55–60	Block count	Used in trailer labels for the number of blocks since the previous header label.
61–73	System code	An identification for the operating system.
74–80	Unused	Reserved.

IOCS FOR MAGNETIC TAPE

The system (IOCS for DOS and data management for OS) performs the following functions for input and for output.

Reading a Tape File

The processing for reading a tape file is as follows:

1. *Processing the Volume Label.* On OPEN, IOCS reads the volume label and compares its serial number to that on the TLBL or DD job control entry.
2. *Processing the Header Label.* IOCS next reads the header label and checks that the file identification agrees with that on the job control entry to ensure that it is reading the correct file. For a multivolume file, the volume sequence numbers are normally in consecutive, ascending sequence.
3. *Reading Records.* The GET macro reads records, specifying either a workarea or IOREG. If the tape records are unblocked, each GET reads one record (a block) from tape into storage. If records are blocked, IOCS performs the required deblocking.
4. *End-of-Volume.* If IOCS encounters the end-of-volume label before the end-of-file (meaning that the file continues on another reel), IOCS checks that the block count is correct. It rewinds the reel, opens a reel on an alternate tape drive, checks the labels, and resumes reading this new reel.
5. *End-of-File.* Each GET operation causes IOCS to transfer a record to the workarea. Once every record has been transferred and processed and you

attempt to perform another GET, IOCS recognizes an end-of-file condition. It then checks the block count, (usually) rewinds the reel, and transfers control to your end-of-file address designated in the DTFMT or DCB macro. You should now CLOSE the tape file. To attempt further reading of a rewound tape file, you must perform another OPEN.

Writing a Tape File

The processing for writing a tape file is as follows:

1. *Processing the Volume Label.* On OPEN, IOCS checks the volume label (VOL) and compares its serial number to the serial number (if any) on the job control entry.
2. *Processing the Header Label.* IOCS next checks the header label for the expiration date. If this date has passed, IOCS backspaces the tape and writes a new header (HDR) over the old one, based on data in job control. If this is a multivolume file, IOCS records the volume sequence number for the volume. It then writes a tape mark.
3. *Writing Records.* If the tape records are unblocked, each PUT writes one record (a block) from tape into storage. If records are blocked, IOCS performs the required blocking.
4. *End-of-Volume.* If IOCS detects the end-of-tape marker near the end of the reel, it writes an EOV trailer label, which includes a count of all blocks written, followed by a tape mark. Since the reflective marker is on the opposite side of the tape, data may be recorded through its area. If an alternate tape drive is assigned, IOCS opens the alternate volume, processes its labels, and resumes writing this new reel.
5. *End-of-File.* When a program closes the tape file, IOCS writes the last block of data, if any. The last block may contain fewer records than the blocking factor specifies. IOCS then writes a tape mark and an EOF trailer label with a block count. Finally, IOCS writes two tape marks and deactivates the file from further processing.

DISK STORAGE

A direct access storage device (DASD), which includes magnetic disk storage and the less common drum storage, is a device that can access any record on a file directly. Diskettes, a common and familiar storage medium on micro- and minicomputers, store data in a similar manner. This section describes the details of the larger magnetic disk devices used in data processing installations.

Each disk storage device contains a number of thin circular plates (or disks)

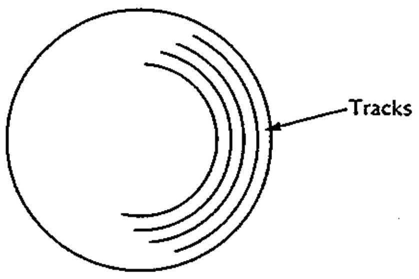


Figure 17-4 Disk surface and tracks.

stacked one on top of the other. Both sides of each plate (except the outer top and bottom on some devices) have a coat of ferrous oxide material to permit recording. As Fig. 17-4 shows, each disk contains circular tracks for storing data records as magnetized bits. Each track contains the same number of bits (and bytes) because the bits are spaced more closely together on the innermost tracks.

The disks are constantly rotating on a vertical shaft. As Fig. 17-5 shows, the disk device has a set of access arms that move read/write heads from track to track. The heads read data blocks from a disk track into main storage and write data blocks from main storage onto a disk track. Because the disks spin continually, the system has to wait for a required data block to reach the read/write heads.

Disk storage devices permit processing of records both sequentially and randomly (directly). As a result, programs can read unsorted records from a transaction file and use them to randomly update matching master records on disk. Disk storage therefore facilitates online processing where users can at any time make inquiries into a file and can enter transactions for updating as they occur.

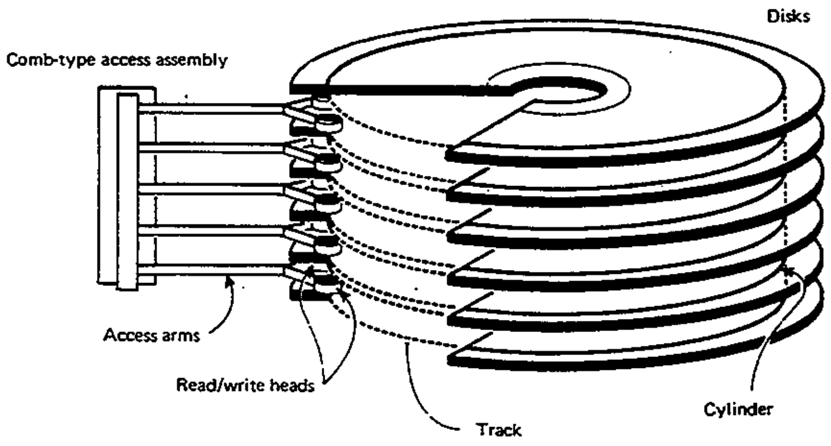


Figure 17-5 Disk read/write mechanism.

Disk Format

The amount of data that a disk device can store varies considerably by model, ranging from small disks with a few million bytes to large disks with more than one billion bytes. Some disk models use fixed-length sectors on each track to store one or more records; the system addresses a record by disk number, track number, and sector number. On other disk models, tracks are not sectored, and records may be of almost any length; the system addresses records by disk surface number and track number.

Like magnetic tape, disk storage contains gaps between one block of data and the next, but the size of the gap is greater on the outermost tracks and smaller on innermost tracks. You may also store records on disk as unblocked or blocked. However, because of the fixed capacity of a disk track, the optimum blocking factor depends on the record length and track capacity. Special formulas are available for calculating optimum blocking factors for different disk devices.

As a simplified example, consider a file containing 1,000-byte records and a disk track with a capacity of 10,000 bytes. If the blocking factor is 5, one block is 5,000 bytes and you can store two blocks (ten records) on a track. If the blocking factor is 6, one block is 6,000 bytes and a track has space for only one block (six records).

The storage of data on disk begins with the top outermost track (track 0) and continues consecutively down, surface by surface, through to the bottom outermost track. Storage of data then continues with the next inner set of tracks (track 1), starting with the top track through to the bottom track. The set of vertical tracks is known as a *cylinder*. As a result, for sequential processing the system reduces access motion of the read/write heads: It reads and writes blocks, for example, on track 5 of every surface (cylinder 5) before moving the arm to cylinder 6.

DISK ARCHITECTURE

The two main types of IBM disk devices are count-key-data (CKD) architecture and fixed-block architecture (FBA).

CKD Architecture

In this design, records and blocks may be of almost any length, subject to limitations of the disk device. A count (C) area contains the block size and an optional key (K) area contains the key of the last record in the block, both of which precede the actual data (D) area; hence CKD.

If a disk contains 20 surfaces, the outer set of tracks (all track 0) is called cylinder 0, the next inner vertical set of tracks is cylinder 1, the next is cylinder 2, and so forth. If the device contains 200 sets of tracks, there are 200 cylinders numbered 0 through 199, each with 20 tracks.

Examples of disk devices using CKD architecture include IBM models 3330, 3340, 3350, and 3380.

The basic format for a track on a CKD device is

Index Point	Home Address	Track Descriptor Record (R0)	Data Record (R1)	Data Record (R2)
(a)	(b)	(c)	(d)	

(a) *Index Point.* The index point tells the read/write device that this point is the physical beginning of the track.

(b) *Home Address.* The home address tells the system the address of the track (the cylinder, head, or surface number) and whether the track is primary, alternate, or defective.

(c) *Track Descriptor Record (R0).* This record stores information about the track and consists of two separate fields: a count area and a data area. The count area contains 0 for record number and 8 for data length and is otherwise similar to the count area described next for data record under item (d). The data area contains 8 bytes of information used by the system. The track descriptor record is not normally accessed by user programs.

(d) *Data Record Formats (R1 through Rn).* The users' data records, or technically, blocks, consist of the following:

Address Marker	Count Area	Key Area (optional)	Data Area

The I/O control unit stores the 2-byte address marker before each block of data, which it uses subsequently to locate the beginning of data.

The count area includes the following:

- An identifier field that provides the cylinder and head number (like that in the home address) and the sequential block number (0–255) in binary, representing R0 through R255. (The track descriptor record, R0, contains 0 for record number.)
- The key length (to be explained shortly).
- The data length, a binary value 0 through 65,535 that specifies the number of bytes in the data area field (the length of your data block). For end-of-file, the system generates a last dummy record containing a length of 0 in this field. When the system reads the file, the zero length indicates that there are no more records.
- The optional key area contains the key, or control field, for the records in the file, such as part number or customer number. The system uses the key area to locate records randomly. If the key area is omitted, the file is said

Device	CAPACITY				SPEED		
	Bytes per track	Tracks per cylinder	Number of cylinders	Total bytes	Ave. seek time (ms)	Ave. rot'l delay (ms)	Data rate KB/sec.
3340-1	8368	12	348	35,000,000	25	10.1	885
3340-2	8368	12	696	70,000,000			
3344	8368	12	4 × 696	280,000,000			
3330-1	13030	19	404	100,000,000	30	8.4	806
3330-11	13030	19	808	200,000,000			
3350	19069	30	555	317,500,000	25	8.4	1200
3375	35616	12	2 × 959	819,738,000	19	10.1	1859
3380	47476	15	2 × 885	1,260,500,000	16	8.3	3000

Figure 17-6 Capacity table for CKD devices.

to be formatted without keys and is stored as count-data format. The key length in the count area contains 0. If the file is formatted with keys, it is stored as count-data format. The key length in the count area contains the length of the key area.

- The data area contains the users' data blocks, in any format, such as unblocked or blocked and fixed or variable length. The system stores as many blocks on a track as possible, usually complete and intact on a track. A record overflow feature permits the overlapping of a record from one track to the next.

Figure 17-6 provides the capacities and speeds of a number of IBM CKD devices.

Under normal circumstances, you won't be concerned with the home address, the track descriptor record, or the address marker, count area, and key area portions of the data record field. You simply provide appropriate entries in your file definition macros and job control commands.

Fixed-Block Architecture

In this design, the recording tracks contain equal-length blocks of 512 bytes, although your records and blocks need not fit a sector exactly.

Figure 17-7 provides the details for two disk models using fixed-block architecture, the 3310 and 3370.

Device	Bytes per Block	Blocks per Track	Number of Cylinders	Tracks per Cylinder	Total Bytes
3310	512	32	358	11	64,520,192
3370	512	62	2 x 750	12	571,392,000

Figure 17-7 Capacity table for FBA devices.

DISK CAPACITY

Knowing the length of records and the blocking factor, you can calculate the number of records on a track and on a cylinder. Knowing the number of records, you can also calculate the number of cylinders for the entire file. Based on the values in Fig. 17-8, the formula for the number of blocks of data per track is

$$\text{Blocks per track} = \frac{\text{track capacity}}{\text{overhead} + C + KL + DL}$$

In the formula, C is a constant overhead value for keyed records, KL means key length, and DL is data (block) length. These values vary by disk device, as shown in Fig. 17-8.

Device	Maximum Capacity (bytes)	One Data Block	Key Overhead	Track Capacity

3330	13,030	135 + C + KL + DL	C = 0 when KL = 0
			C = 56 when KL ≠ 0
				13,165

3340	8,368	167 + C + KL + DL	C = 0 when KL = 0
			C = 75 when KL ≠ 0
				8,535

3350	19,069	185 + C + KL + DL	C = 0 when KL = 0
			C = 82 when KL ≠ 0
				19,254

Figure 17-8 Track capacity table.

The following two examples illustrate.

Example 1. Device is a 3350, records are 242 bytes, five records per block (block size = 1,210), and formatted without keys:

$$\text{Blocks per track} = \frac{19{,}254}{185 + (5 \times 242)} = \frac{19{,}254}{1{,}395} = 13.8$$

$$\begin{aligned}\text{Records per track} &= \text{blocks per track} \times \text{blocking factor}\\ &= 5 \times 13 = 65\end{aligned}$$

Example 2. Same as Example 1, but formatted with keys (key length is 12):

$$\text{Blocks per track} = \frac{19{,}254}{185 + 82 + 12 + 1{,}210} = \frac{19{,}254}{1{,}489} = 12.93$$

$$\text{Records per track} = 5 \times 12 = 60$$

Note that a disk stores a full block, not a fraction of one. Therefore, even if you calculate 13.8 or 12.9 blocks per track, the disk stores only 13 or 12 blocks, respectively.

To determine the number of records on a cylinder, refer to Fig. 17-6, which discloses that a 3350 has 30 tracks per cylinder. Based on Example 1 where the number of records per track is 65, a cylinder on the 3350 could contain 65 × 30 = 1,950 records.

Using these figures, you can now calculate how much disk storage a file of, say, 100,000 of these records would require. Based on the figure of 1,950 records per cylinder, the file would require 100,000 ÷ 1,950 = 51.28 cylinders.

DISK LABELS

Disks, like magnetic tape, also use labels to identify a volume and a file. The system reserves cylinder 0, track 0 for standard labels, as Fig. 17-9 shows. The following describes the contents of track 0:

Record 0: The track descriptor, R(0) record.

Records 1 and 2: If the disk is SYSRES, which contains the operating system, certain devices reserve R(1) and R(2) for the initial program load (IPL) routine. For all other cases, R(1) and R(2) contain zeros.

Record 3: The VOL1 label. OS supports more than one volume label, from R(3) through R(10).

Record 4 through the end of the track: The standard location for the volume table of contents (VTOC). The VTOC contains the file labels for the files on the device. Although you may place a VTOC in any cylinder, its standard location is cylinder 0, track 0.

Volume Labels

The standard volume label uniquely identifies a disk volume. A 4-byte key area immediately precedes the 80-byte volume data area. The volume label is the fourth record (R3) on cylinder 0. The 80 bytes are arranged like a tape volume label, with one exception: Positions 12–21 are the "data file directory," containing the starting address of the VTOC.

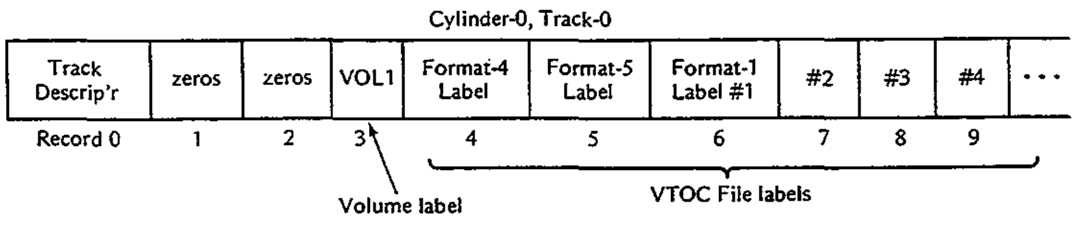


Figure 17-9 Disk volume layout.

File Labels

File labels identify and describe a file, or data set, on a volume. The file label is 140 bytes long, consisting of a 44-byte key area and a 96-byte file data area. Each file on a volume requires a file label for identification. In Fig. 17-9, all file labels for a volume are stored together in the VTOC. There are four types of file labels:

1. The format 1 label is equivalent to a file label on tape. The format 1 label differs, however, in that it defines the actual cylinder and track addresses of each file's beginning and end (its extent). Further, a file may be stored intact in an extent or in several extents in the same volume. Format 3 is used if a file is scattered over more than three extents.
2. The format 2 label is used for indexed sequential files.
3. The format 3 label is stored if a file occupies more than three extents.
4. The format 4 label is the first record in the VTOC and defines the VTOC for the system.

The format 1 file label contains the following information:

POSITION	NAME	DESCRIPTION
01–44	File identification	Unique identifier consisting of file ID, optional generation number, and version number, separated by periods.
45	Format identifier	'1' for format 1.
46–51	File serial number	Volume serial number from the volume label.
52–53	Volume sequence number	Sequence number if the file is stored on more than one volume.
54–56	Creation date	ydd (binary): y = year (0–99) and dd = day (1–366).
57–59	Expiration date	Same format as creation date.
60	Extent count number	Number of extents for this file on this volume.
61	Bytes used in last block of directory	Used by OS.
62	Unused	Reserved.
63–75	System code	Name of the operating system.
76–82	Unused	Reserved.
83–84	File type	Code to identify if SD (sequential),

POSITION	NAME	DESCRIPTION
		DA (direct), or IS (indexed) file organization.
85	Record format	Used by OS.
86	Option codes	ISAM—indicates if master index is present and type of overflow area.
87–88	Block length	ISAM—length of each block.
89–90	Record length	ISAM—length of each record.
91	Key length	ISAM—length of key area.
92–93	Key location	ISAM—position of key within the record.
94	Data set indicators	SD—indicates if last volume.
95–98		Used by OS.
99–103	Last record pointer	Used by OS.
104–105	Unused	Reserved.
106	Extent type	Descriptors for the first or only extent for the file.
107	Extent sequence number	
108–111	Extent lower limit	
112–115	Extent upper limit	
116–125		Descriptors for a second extent.
126–135		Descriptors for a third extent.
136–140	Pointer	Address of the next label.

KEY POINTS

- Sequential file organization provides only for sequential processing of records. Indexed and direct organization provides for both sequential and random processing of records.
- At the beginning of the tape reel is a volume label, which identifies the reel being used. Immediately preceding each file on the tape is a header label, which contains the name of the file and the date the file was created. Following the header label are the records that comprise the data file. The last record is a trailer label, which is similar to the header label but also contains the number of blocks written on the reel.

- To keep track of all the files it contains, a disk device uses a special directory (volume table of contents, VTOC) at the beginning of its storage area. The directory includes the names of the files, their locations on disk, and their present status.
- If you define a tape or disk field as packed on an IBM system, the field contains two digits per byte plus a half-byte for the sign.
- The set of vertical tracks on a disk device is known as a cylinder.
- An interblock gap (IBG) separates each block of data from the next on tape and disk. The length of an IBG on tape is 0.3 to 0.6 inches depending on the device, and the length of an IBG on disk varies by device and by track location. The IBG defines the start and end of each block of data and provides space for the tape when the drive stops and restarts for each read or write.
- Blocking of records helps conserve space on storage devices and reduces the number of input/output operations. The number of records in a block is known as the blocking factor.
- The system reads an entire block into the computer's storage and transfers one record at a time to the program.
- All programs that process a file should use the same record length and blocking factor.
- Records and blocks may be fixed in length, where each has the same length throughout the entire file, or variable in length, where the length of each record and the blocking factor are not predetermined.
- The two main types of disk devices are count-key-data (CKD) architecture, which stores records according to count, key, and data area, and fixed-block architecture (FBA), which stores data in fixed-length blocks.

PROBLEMS

17-1. Distinguish the differences among sequential, direct, and indexed sequential organization methods.

17-2. Explain each of the following: (a) tape density; (b) tape markers; (c) IBG; (d) fixed length and variable length; (e) blocking factor.

17-3. Give an advantage and a disadvantage of increasing the blocking factor.

17-4. What is the purpose of each of the following: (a) volume label; (b) header label; (c) trailer label?

17-5. Distinguish between each of the following: (a) EOV and EOF on a trailer label; (b) a multifile volume and a multivolume file; (c) volume sequence number and file sequence number.

17-6. What is the advantage of disk storage over magnetic tape?

17-7. Based on Fig. 17-6, how many bytes can be stored on a cylinder for each device listed?

17-8. Why does a disk device store data vertically by cylinder rather than by tracks across a surface?

17-9. Explain the purpose of (a) the home address; (b) the track descriptor record; (c) the key area.

17-10. What is the difference between a CKD disk and an FBA disk?

17-11. What are the purpose, location, and contents of the VTOC?

17-12. What is the disk equivalent of the magnetic tape header label; that is, what is its location and how does it differ?

17-13. Assume disk device 3350, record length 300 bytes, and six records per block. Based on the data in Fig. 17-8, calculate the number of records that a track can store for the following: (a) records formatted without keys; (b) records formatted with keys, key length = 10.

18

SEQUENTIAL FILE ORGANIZATION

OBJECTIVE

To cover sequential file organization and its processing requirements.

In this chapter, you examine sequential file organization for DOS and OS and learn how to create and read such files. You will also examine the definition and processing of variable-length records.

The processing of sequential files involves the same imperative macros used up to now: OPEN, CLOSE, GET, and PUT. IOCS (data management) handles all the necessary label processing and blocking and deblocking of records. Other than job control commands, the only major difference is the use of blocked records.

An installation has to make a (perhaps arbitrary) choice of a blocking factor when a file is created, and all programs that subsequently process the file define the same blocking factor. A program may also define one or more I/O buffers; if records are highly blocked, a second buffer involves more space in main storage with perhaps little gained in processing speed.

CREATING A TAPE FILE

The first two examples create a tape file for DOS and OS. The programs accept input data from the system reader and write four records per block onto tape.

For both programs, OPEN checks the volume label and header label, and CLOSE writes the last block (even if it contains fewer than four records) and writes a trailer label.

DOS Program to Create a Tape File

The DOS DTFMT file definition macro defines a magnetic tape file. You define a DTFMT macro with a unique name for each tape input or output file that the program processes. The parameters that you code are similar to those for the DTFCD and DTFPR macros covered earlier.

In Fig. 18-1, the program reads records into RECDIN and transfers required fields to a tape workarea named TAPEWORK. The program then writes this workarea to a tape output file named FILOTP. Based on the BLKSIZE entry in the DTFMT, IOCS blocks four records before physically writing the block onto tape. Thus for every four input records that the program reads, IOCS writes one block of four records onto tape.

```
         PRINT ON,NODATA,NOGEN
PROG18A  START
         BALR  3,0                     INITIALIZE BASE REGISTER
         USING *,3
         OPEN  FILEIN,FILEOTP          ACTIVATE FILES
         GET   FILEIN,RECDIN           READ 1ST RECORD
A10LOOP  BAL   9,B10PROC
         GET   FILEIN,RECDIN           READ NEXT
         B     A10LOOP

*              E N D - O F - F I L E
A90EOF   CLOSE FILEIN,FILEOTP          DE-ACTIVATE FILES
         EOJ                           NORMAL END-OF-JOB

***            M A I N  P R O C E S S I N G
B10PROC  MVC   ACCTTPO,ACCTIN          MOVE INPUT FIELDS
         MVC   NAMETPO,NAMEIN          *  TO WORK AREA
         MVC   ADDRTPO,ADDRIN          *
         PACK  BALNTPO,BALNIN          *
         MVC   DATETPO,DATEIN          *
         PUT   FILEOTP,TAPEWORK        WRITE TAPE WORKAREA
         BR    9                       RETURN

*              D E C L A R A T I V E S
FILEIN   DTFCD DEVADDR=SYSIPT,         INPUT FILE               +
               IOAREA1=IOARIN1,                                 +
               BLKSIZE=80,                                      +
               DEVICE=2540,                                     +
               EOFADDR=A90EOF,                                  +
               TYPEFLE=INPUT,                                   +
               WORKA=YES
```

Figure 18-1 Program: writing a tape file under DOS.

```
IOARIN1  DC     CL80' '                            INPUT BUFFER 1
RECDIN   DS     0CL80                              INPUT AREA:
CODEIN   DS     CL02                               01-02  RECORD CODE
ACCTIN   DS     CL06                               03-08  ACCOUNT NO.
NAMEIN   DS     CL20                               09-28  NAME
ADDRIN   DS     CL40                               29-68  ADDRESS
BALNIN   DS     ZL06'0000.00'                      69-74  BALANCE
DATEIN   DS     CL06'DDMMYY'                       75-80  DATE
FILEOTP  DTFMT  BLKSIZE=360,                       TAPE FILE            +
                DEVADDR=SYS025,                                         +
                FILABL=STD,                                             +
                IOAREA1=IOARTPO1,                                       +
                IOAREA2=IOARTPO2,                                       +
                RECFORM=FIXBLK,                                         +
                RECSIZE=90,                                             +
                TYPEFLE=OUTPUT,                                         +
                WORKA=YES
IOARTPO1 DS     CL360                              TAPE BUFFER-1
IOARTPO2 DS     CL360                              TAPE BUFFER-2

TAPEWORK DS     0CL90                              TAPE WORK AREA:
ACCTTPO  DS     CL06                               01-06  ACCOUNT NO.
NAMETPO  DS     CL20                               07-26  NAME
ADDRTPO  DS     CL40                               27-66  ADDRESS
BALNTPO  DS     PL04                               67-70  BALANCE
DATETPO  DS     CL06                               71-76  DATE
         DC     CL14' '                            77-90  RESERVED

         LTORG
                =C'$$BOPEN '
                =C'$$BCLOSE'
                =A(FILEIN)
                =A(RECDIN)
                =A(FILEOTP)
                =A(TAPEWORK)
         END    PROG18A
```

Figure 18-1 (continued)

The following explains the DTFMT entries:

BLKSIZE=360 means that each block to be written from the IOAREA is 360 bytes long, based on four records at 90 bytes each.

DEVADDR=SYS025 denotes the logical address of the tape device that is to write the file.

FILABL=STD indicates that the tape file contains standard labels, as described in Chapter 17.

IOAREA1 and **IOAREA2** are the two IOCS buffers, each defined with the same length (360) as BLKSIZE. If your blocks are especially large, you may omit defining a second buffer to reduce program size.

RECFORM=FIXBLK defines output records as fixed-length and blocked. Records on tape and disk may also be variable-length or unblocked.

RECSIZE=90 means that each fixed-length record is 90 bytes in length, the same as the workarea.

TYPEFLE=OUTPUT means that the file is output, that is, for writing only. Other options are INPUT and WORK (for a work file).

WORKA=YES means that the program is to process output records in a workarea. In this program, TAPEWORK is the workarea and has the same length as RECSIZE, 90 bytes. Alternatively, you may code IOREG and use the macro PUT FILEOTP with no workarea coded in the operand.

The DTFMT file definition macro for tape input requires an entry EOFADDR=address to indicate the name of the routine where IOCS links on reaching the end of the tape file.

OS Program to Create a Tape File

For OS, you define a DCB macro with a unique name for each tape input or output file that the program processes. The parameters that you code are similar to those for the DCB macros covered earlier.

In Fig. 18-2, the program reads records into RECDIN and transfers required fields to a tape workarea named TAPEWORK. The program then writes this workarea to a tape output file named FILOTP. Based on the BLKSIZE entry in job control, the system blocks four records before physically writing the block onto tape. Thus for every four input records that the program reads, the system writes one block of four records onto tape.

```
//GO.TAPEOT  DD DSNAME=TRFILE,DISP=(NEW,PASS),UNIT=3420,          +
               DCB=(BLKSIZE=360,RECFM=FB,DEN=3)
                                             <------ DD for tape
                                                     output data set
//GO.SYSIN   DD *         <--- DD for input file
PROG18B  START
         SAVE  (14,12)
         BALR  3,0
         USING *,3
         ST    13,SAVEAREA+4
         LA    13,SAVEAREA
         OPEN  (FILEIN,(INPUT),FILEOTP,(OUTPUT))
         GET   FILEIN,RECDIN            READ 1ST RECORD

***            M A I N   P R O C E S S I N G
A10LOOP  MVC   ACCTTPO,ACCTIN           MOVE INPUT FIELDS TO TAPE
         MVC   NAMETPO,NAMEIN           *    WORK AREA
         MVC   ADDRTPO,ADDRIN           *
         PACK  BALNTPO,BALNIN           *
         MVC   DATETPO,DATEIN           *
         PUT   FILEOTP,TAPEWORK         WRITE WORK AREA ONTO TAPE
         GET   FILEIN,RECDIN            READ NEXT RECORD
         B     A10LOOP

*              E N D - O F - F I L E
A90EOF   CLOSE (FILEIN,,FILEOTP)
         L     13,SAVEAREA+4
         RETURN (14,12)
```

Figure 18-2 Program: writing a tape file under OS.

```
*                D E C L A R A T I V E S
FILEIN   DCB     DDNAME=SYSIN,             DCB FOR INPUT DATA SET    +
                 DEVD=DA,                                            +
                 DSORG=PS,                                           +
                 EODAD=A90EOF,                                       +
                 MACRF=(GM)

RECDIN   DS      0CL80                     INPUT RECORD AREA:
CODEIN   DS      CL02                      01-02  RECORD CODE
ACCTIN   DS      CL06                      03-08  ACCOUNT NO.
NAMEIN   DS      CL20                      09-28  NAME
ADDRIN   DS      CL40                      29-68  ADDRESS
BALNIN   DS      ZL06'0000.00'             69-74  BALANCE
DATEIN   DS      CL06'DDMMYY'              75-80  DATE

FILEOTP  DCB     DDNAME=TAPEOT,            DCB FOR TAPE DATA SET     +
                 DSORG=PS,                                           +
                 LRECL=90,                                           +
                 MACRF=(PM)

TAPEWORK DS      0CL90                     TAPE WORK AREA:
ACCTTPO  DS      CL06                      01-06  ACCOUNT NO.
NAMETPO  DS      CL20                      07-26  NAME
ADDRTPO  DS      CL40                      27-66  ADDRESS
BALNTPO  DS      PL04                      67-70  BALANCE(PACKED)
DATETPO  DS      CL06                      71-76  DATE
         DC      CL14' '                   77-90  RESERVED

SAVEAREA DS      18F                       REGISTER SAVE AREA
         LTORG
         END     PROG18B
```

Figure 18-2 (continued)

The DD job commands for the files appear first in the job stream and provide some entries that could also appear in the DCB. This common practice enables users to change entries without reassembling programs. The DD entries for the tape file, TAPEOT, are as follows:

DSNAME=TRFILE provides the data set name.

DISP=(NEW,PASS) means that the file is new (to be created) and is to be kept temporarily.

UNIT=3420 provides the tape drive model.

BLKSIZE=360 means that each block to be written from the IOAREA is 360 bytes long, based on four records at 90 bytes each.

RECFM=FB defines output records as fixed-length and blocked. Records on tape and disk may also be variable-length (V) or unblocked.

DEN=3 indicates tape density as 1,600 bpi. (DEN=2 would mean 800 bpi.)

The following explains the DCB entries:

DDNAME=TAPEOT relates to the same name in the the DD job control command:

```
//GO.TAPEOT ...
```

DSORG=PS defines output as physical sequential.

LRECL=90 provides the logical record length for each record.

MACRF=(PM) defines the type of output operation as put and move from a workarea. MACRF=(PL) would allow you to use locate mode to process records directly in the buffers.

The DCB file definition macro for tape input requires an entry EOFADDR=address to indicate the name of the routine where IOCS links on reaching the end of the tape file.

Also, another DCB entry, EROPT, provides for an action if an input operation encounters problems. The options are as follows:

=ACC	Accept the possibly erroneous block of data.
=SKP	Skip the data block entirely and resume with the next one.
=ABE	Abend (abnormal end of program execution), the standard default if you omit the entry.

ACC and SKP can use a SYNAD entry for printing an error message and continue processing. If the error message routine is named R10TPERR, the DCB coding could be

```
EROPT=SKP,
SYNAD=R10TPERR
```

Since the use of ACC and SKP may cause invalid results, it may be preferable for important production jobs to use ABE (or allow it to default). See the OS supervisor manuals for other DCB options.

CREATING A SEQUENTIAL DISK FILE

The next two examples create a disk file for DOS and OS. The programs accept input data from the system reader and write four records per block onto disk.

For both programs, OPEN checks the disk label, and CLOSE writes the last data block (even if it contains fewer than four records) and writes a last dummy block with zero length.

DOS Program to Create a Sequential Disk File

The DOS file definition macro that defines a sequential disk file is DTFSD. The parameters that you code are similar to those for the DTFMT macro.

The program in Fig. 18-3 reads the tape records from the file created in Fig. 18-1 and transfers required fields to a disk workarea named DISKWORK. The program then writes this workarea to a disk output file named SDISK. Based on

```
  1          PRINT ON,NODATA,NOGEN
  2 PROG18B  START
  3          BALR  3,0
  4          USING *,3
  5          OPEN  TAPE,SDISK
 14          GET   TAPE,TAPEIN            READ 1ST RECORD
 20 A10LOOP  BAL   9,B10PROC
 21          GET   TAPE,TAPEIN            READ NEXT RECORD
 27          B     A10LOOP

 29 *              M A I N  P R O C E S S I N G
 30 B10PROC  MVC   ACCTDKO,ACCTIN         MOVE FIELDS TO DISK
 31          MVC   NAMEDKO,NAMEIN         *    WORK AREA
 32          MVC   ADDRDKO,ADDRIN         *
 33          ZAP   BALNDKO,BALNIN         *
 34          MVC   DATEDKO,DATEIN         *
 35          PUT   SDISK,DISKWORK         WRITE WORK AREA
 41          BR    9

 43 *              E N D - O F - F I L E
 44 A90END   CLOSE TAPE,SDISK
 53          EOJ

 57 *              D E C L A R A T I V E S
 58 TAPE     DTFMT BLKSIZE=360,           TAPE FILE              +
                   DEVADDR=SYS025,                               +
                   EOFADDR=A90END,                               +
                   ERROPT=IGNORE,                                +
                   FILABL=STD,                                   +
                   IOAREA1=IOARTPI1,                             +
                   RECFORM=FIXBLK,                               +
                   RECSIZE=090,                                  +
                   TYPEFLE=INPUT,                                +
                   WORKA=YES
 96 IOARTPI1 DS    CL360                  INPUT TAPE BUFFER
 98 TAPEIN   DS    0CL90                  TAPE INPUT AREA:
 99 ACCTIN   DS    CL6                    *  ACCOUNT NO.
100 NAMEIN   DS    CL20                   *  NAME
101 ADDRIN   DS    CL40                   *  ADDRESS
102 BALNIN   DS    PL4                    *  BALANCE
103 DATEIN   DS    CL6'DDMMYY'            *  DATE
104          DS    CL14                   *  UNUSED
106 SDISK    DTFSD BLKSIZE=368,           DISK FILE              +
                   DEVADDR=SYS015,                               +
                   DEVICE=3380,                                  +
                   IOAREA1=IOARDK,                               +
                   RECFORM=FIXBLK,                               +
                   RECSIZE=90,                                   +
                   TYPEFLE=OUTPUT,                               +
                   VERIFY=YES,                                   +
                   WORKA=YES
172 IOARDK   DS    CL368                  DISK BUFFER

174 DISKWORK DS    0CL90                  DISK WORK AREA:
175 ACCTDKO  DS    CL06                   *  ACCOUNT NO.
```

Figure 18-3 Program: writing a sequential disk file under DOS.

```
NAMEDKO  DS     CL20                         *  NAME
ADDRDKO  DS     CL40                         *  ADDRESS
BALNDKO  DS     PL04                         *  BALANCE
DATEDKO  DS     CL06                         *  DATE
         DC     CL14' '                      *  RESERVED
         LTORG
                =C'$$BOPEN '
                =C'$$BCLOSE'
                =A(TAPE)
                =A(TAPEIN)
                =A(SDISK)
                =A(DISKWORK)
         END    PROG18B

// EXEC LNKEDT
// TLBL    TAPE,'CUST REC TP',0,100236
// ASSGN   SYS015,DISK,VOL=SVSE03,SHR
// DLBL    SDISK,'CUSTOMER RECORDS SD',0,SD
// EXTENT  SYS015,ATMP70,1,0,3,4
```

Figure 18-3 (continued)

the BLKSIZE entry in the DTFMT and DTFSD, the system both reads and writes blocks of four records, although the two blocking factors need not be the same.

The following explains the DTFSD entries:

BLKSIZE=368 means that the blocksize for output is 360 bytes (4 x 90) plus 8 bytes for the system to construct a count field. You provide for the extra 8 bytes only for output; for input, the entry would be 360.

DEVICE=3380 means that the program is to write blocks on a 3380 disk device.

VERIFY=YES tells the system to reread each output record to check its validity. If the record when reread is not identical to the record that was supposed to be written, the system rewrites the record and performs another reread. If the system eventually cannot perform a valid write, it may advance to another area on the disk surface. Although this operation involves more accessing time, it helps ensure the accuracy of the written records.

DEVADDR, IOAREA1, RECFORM, RECSIZE, TYPEFLE, and **WORKA** are the same as for previous DTFs. You omit the FILABL entry because disk labels must be standard.

If you omit the entry for DEVADDR, the system uses the SYSnnn address from the job control entry.

OS Program to Create a Sequential Disk File

For OS, you define a DCB macro with a unique name for each disk input or output file that the program processes. The parameters that you code are similar to those for the DCB macros covered earlier.

The program in Fig. 18-4 reads the tape records from the file created in Fig.

```
//GO.TAPEIN DD DSNAME=TRFILE,DISP=(OLD,PASS),UNIT=3420,                  +
               DCB=(BLKSIZE=360,RECFM=FB,DEN=3)
//GO.DISKOT DD DSNAME=&TEMPDSK,DISP=(NEW,PASS),UNIT=3380,SPACE=(TRK,10), +
               DCB=(BLKSIZE=360,RECFM=FB)

PROG18D  START 0
         SAVE  (14,12)
         BALR  3,0
         USING *,3
         ST    13,SAVEAREA+4
         LA    13,SAVEAREA
         OPEN  (TAPE,(INPUT),SDISK,(OUTPUT))
         GET   TAPE                     READ 1ST TAPE RECORD

***            M A I N  P R O C E S S I N G
A10LOOP  MVC   TAPEIN,0(1)              MOVE FROM TAPE BUFFER
         MVC   ACCTDKO,ACCTIN           MOVE TAPE FIELDS TO DISK
         MVC   NAMEDKO,NAMEIN           *    WORK AREA
         MVC   ADDRDKO,ADDRIN           *
         ZAP   BALNDKO,BALNIN           *
         MVC   DATEDKO,DATEIN           *
         PUT   SDISK,DISKWORK           WRITE WORK AREA ONTO DISK
         GET   TAPE                     READ NEXT TAPE RECORD
         B     A10LOOP

***            E N D - O F - F I L E
A90END   CLOSE (TAPE,,SDISK)
         L     13,SAVEAREA+4
         RETURN (14,12)

***            D E C L A R A T I V E S
TAPE     DCB   DDNAME=TAPEIN,           TAPE INPUT DATA SET          +
               DSORG=PS,                                             +
               EODAD=A90END,                                         +
               LRECL=90,                                             +
               MACRF=(GL)

TAPEIN   DS    0CL90                    TAPE INPUT AREA:
ACCTIN   DS    CL06                     *  ACCOUNT NO.
NAMEIN   DS    CL20                     *  NAME
ADDRIN   DS    CL40                     *  ADDRESS
BALNIN   DS    PL04                     *  BALANCE (PACKED)
DATEIN   DS    CL06'DDMMYY'             *  DATE
         DS    CL14                     *  UNUSED

SDISK    DCB   DDNAME=DISKOT,           DISK OUTPUT DATA SET         +
               DSORG=PS,                                             +
               LRECL=90,                                             +
               MACRF=(PM)

DISKWORK DS    0CL90                    DISK WORK AREA:
ACCTDKO  DS    CL06                     *  ACCOUNT NO.
NAMEDKO  DS    CL20                     *  NAME
ADDRDKO  DS    CL40                     *  ADDRESS
BALNDKO  DS    PL04                     *  BALANCE (PACKED)
DATEDKO  DS    CL06                     *  DATE
         DC    CL14' '                  *  RESERVED FOR EXPANSION

SAVEAREA DS    18F                      REGISTER SAVE AREA
         LTORG
         END   PROG18D
```

Figure 18-4 Program: writing a sequential disk file under OS.

18-2 and transfers required fields to a disk workarea named DISKWORK. The program then writes this workarea to a disk output file named SDISK. Based on the BLKSIZE entry in job control, the system both reads and writes blocks of four records, although the two blocking factors need not be the same.

The DD entries for the disk file, DISKOT, are as follows:

DSNAME=&TEMPDSK provides the data set name.

DISP=(NEW,PASS) means that the file is new and is to be kept temporarily.

UNIT=3380 provides the disk drive model.

SPACE=(TRK,10) allocates ten tracks for this file.

BLKSIZE=360 means that each block to be written from the buffer is 360 bytes long, based on four records at 90 bytes each.

RECFM=FB defines output records as fixed-length and blocked. Records on disk may also be variable-length (V) or unblocked.

The following explains the DCB entries:

DDNAME=DISKOT relates to the same name in the the DD job control command:

```
//GO.DISKOT ...
```

DSORG=PS defines output as physical sequential.

LRECL=90 provides the logical record length for each record.

MACRF=(PM) defines the type of output operation as put and move from a workarea. MACRF=(PL) would allow you to use locate mode to process records directly in the buffers.

The DCB file definition macro for disk input requires an entry EOFADDR=address to indicate the name of the routine where the system links on reaching the end of the disk file.

VARIABLE-LENGTH RECORDS

Tape and disk files provide for variable-length records, either unblocked or blocked. The use of variable-length records may significantly reduce the amount of space required to store a file. However, beware of trivial applications in which variations in record size are small or the file itself is small, because the system generates overhead that may defeat any expected savings.

A record may contain one or more variable-length fields or a variable number of fixed-length fields.

1. *Variable-Length Fields.* For fields such as customer name and address that vary considerably in length, a program could store only significant characters

and delete trailing blanks. One approach is to follow each variable field with a special delimiter character such as an asterisk.

The following example illustrates fixed-length name and address of 20 characters each, compressed into variable length with an asterisk replacing trailing blanks:

Fixed length:	`Norman Bates........Bates Motel.........`
Variable length:	`Norman Bates*Bates Motel*`

To find the end of the field, the program may use a TRT instruction to scan for the delimiter. Another technique stores a count of the field length immediately preceding each variable-length field. For the preceding record, the count for the name would be 12 and the count for the address would be 11:

```
|12|Norman Bates|11|Bates Motel|
```

2. *Variable Number of Fixed-Length Fields.* Records may contain a variable number of fields. For example, an electric utility company may maintain a large file of customer records with a fixed portion containing the customer name and address and optional subrecords for their electric account, natural gas account, and budget account.

VARIABLE-LENGTH RECORD FORMAT

Immediately preceding each variable-length record on tape or disk is a 4-byte record control word (RCW) that supplies the length of the record. Immediately preceding each block is a 4-byte block control word (BCW) that supplies the length of the block. As a consequence, both records and blocks may be variable length. You have to supply a maximum block size into which the system is to fit as many records as possible.

Unblocked Records

Variable-length records that are unblocked contain a BCW and an RCW before each block. Here are three unblocked records:

```
|BCW|RCW|record 1|...|BCW|RCW record 2|...|BCW|RCW record 3|
```

Suppose that three records are to be stored as variable-length unblocked. Their lengths are 310, 260, and 280 bytes, respectively:

Field:	BCW	RCW	record		BCW	RCW	record		BCW	RCW	record
Length:	4	4	310		4	4	260		4	4	280
Contents:	318	314	record 1		268	264	record 2		288	284	record 3

The RCW contains the length of the record plus its own length of 4. Since the first record has a length of 310, its RCW contains 314. The BCW contains the length of the RCW(s) plus its own length of 4. Since the only RCW contains a length of 314, the BCW contains 318.

Blocked Records

Variable-length records that are blocked contain a BCW before each block and an RCW before each record. The following shows a block of three records:

```
|BCW|RCW|record 1|RCW|record 2|RCW|record 3
```

Suppose that the same three records with lengths of 310, 260, and 280 bytes are to be stored as variable-length blocked and are to fit into a maximum block size of 900 bytes:

Field:	BCW	RCW	record	RCW	record	RCW	record
Length:	4	4	310	4	260	4	280
Contents:	866	314	record 1	264	record 2	284	record 3

The length of the block is the sum of one BCW, the RCWs, and the record lengths:

Block control word:	4 bytes
Record control words:	12
Record lengths:	+850
Total length:	866 bytes

The system stores as many records as possible in the block up to (in this example) 900 bytes. Thus a block may contain any number of bytes up to 900, and both blocks and records are variable length. The system automatically handles all blocking, unblocking, and control of BCWs.

Your BLKSIZE entry tells the system the maximum block length. For example, if the BLKSIZE entry in the preceding example specified 800, the system would fit only the first two records in the block, and the third record would begin the next block.

Programming for Variable-Length Records

Although IOCS performs most of the processing for variable-length records, you have to provide the record length. The additional programming steps are concerned with the record and block length:

Record length. As with fixed-length records, a program may process variable-length records in a workarea or in the buffers (I/O areas). You define the workarea as the length of the largest possible record, including the 4-byte record control word. When creating each record, calculate and store the record length in the record control word field. This field must be 4 bytes long, with the contents in binary format, as

```
VARRCW  DS      F
```

DOS uses only the first 2 bytes of this field.

Block length. You define the I/O area as the length of the largest possible block, including the 4-byte block control word. On output, IOCS stores as many complete records in the block as will fit. IOCS performs all blocking and calculating of the block length. On input, IOCS deblocks all records, similar to its deblocking of fixed-length records.

Sample Program: Reading and Printing Variable-Length Records

Consider a file of disk records that contains variable-length records, with fields defined as follows:

01–04	Record length
05–09	Account number
10–82	Variable name and address

To indicate the end of a name, it is immediately followed by a delimiter, in this case a plus sign (hex '4E'). Another delimiter terminates the next field, the address, and a third terminates the city. Here is a typical case:

```
JP Programmer+1425 North Basin Street+Kingstown+
```

The program in Fig. 18-5 reads and prints these variable-length records. Note that in the DTFSD, RECFORM=VARBLK specifies variable blocked. The program reads each input record and uses TRT and a loop to scan each of the three variable-length fields for the record delimiter. It calculates the length of each field and uses EX to move each field to the output area. The program also checks for the absence of a delimiter.

Output would appear as

```
JP Programmer
1425 North Basin Street
Kingstown
```

```
         PRINT ON,NODATA,NOGEN
PROG18C  START
         BALR  3,0
         USING *,3
         OPEN  FILEIDK,FILEOPR
         GET   FILEIDK,WORKAREA         READ 1ST RECORD

***            M A I N   P R O C E S S I N G
A10LOOP  BAL   5,B10SCAN                SCAN
         GET   FILEIDK,WORKAREA         READ RECORD
         B     A10LOOP
*              E N D - O F - F I L E
A90EOF   CLOSE FILEIDK,FILEOPR          TERMINATE
         EOJ

*              P R O C E S S   V A R I A B L E   R E C O R D
B10SCAN  LA    6,IDENTIN                ADDR OF INPUT IDENT
         LR    7,6                      ESTABLISH ADDRESS OF
         AH    7,RECLEN                    END OF RECORD
         SH    7,=H'9'
         MVC   PRINT+10(5),ACCTIN       MOVE ACCOUNT TO PRINT
B20      TRT   0(73,6),SCANTAB          SCAN FOR DELIMITER
         BZ    B30                      *  NO DELIMITER FOUND
         LR    4,1                      SAVE ADDR OF DELIMITER
         SR    1,6                      CALC. LENGTH OF FIELD
         BCTR  1,0                      DECREMENT LENGTH BY 1
         EX    1,M10MOVE                MOVE VAR LENGTH FIELD
         MVI   CTLCHPR,WSP1
         PUT   FILEOPR,PRINT            PRINT, SPACE 1
         MVC   PRINT,BLANKPR            CLEAR PRINT AREA
         LA    6,1(0,4)                 INCREMENT FOR NEXT FIELD
         CR    6,7                      PAST END OF RECORD?
         BL    B20                      *  NO  - SCAN NEXT
*                                       *  YES - END
B30      MVI   CTLCHPR,WSP2
         PUT   FILEOPR,PRINT            PRINT 3RD LINE
         BR    5                        RETURN

M10MOVE  MVC   PRINT+20(0),0(6)         MOVE VAR FIELD TO PRINT

*              D E C L A R A T I V E S
SCANTAB  DC    78X'00'                  TRT TABLE:
         DC    X'4E'                    * DELIMITER POSITION
         DC    177X'00'                 * REST OF TABLE

FILEIDK  DTFSD BLKSIZE=300,             DISK FILE               +
               DEVICE=3380,                                     +
               DEVADDR=SYS025,                                  +
               EOFADDR=A90EOF,                                  +
               IOAREA1=IOARDKI1,                                +
               IOAREA2=IOARDKI2,                                +
               RECFORM=VARBLK,                                  +
               TYPEFLE=INPUT,                                   +
               WORKA=YES
         DS    0H                       ALIGN ON EVEN BOUNDARY
IOARDKI1 DS    CL300                    BUFFER-1 DISK FILE
IOARDKI2 DS    CL300                    BUFFER-2 DISK FILE

*                                       INPUT AREA:
         DS    0H                       * ALIGN EVEN BOUNDARY
```

Figure 18-5 Program: printing variable-length records.

```
WORKAREA DS     0CL82                       * MAX. RECORD + LENGTH
RECLEN   DS     H                           * 2-BYTE RECORD LENGTH
         DC     H'0'                        * 2 BYTES UNUSED IN DOS
ACCTIN   DS     CL05                        * ACCOUNT NUMBER
IDENTIN  DS     CL73                        * AREA FOR VAR. NAME|ADDR

FILEOPR  DTFPR BLKSIZE=133,                 PRINTER FILE          +
               CTLCHR=YES,                                        +
               DEVADDR=SYSLST,                                    +
               DEVICE=3203,                                       +
               IOAREA1=IOARPR1,                                   +
               IOAREA2=IOARPR2,                                   +
               WORKA=YES
IOARPR1  DC     CL133' '                    BUFFER-1 PRINT FILE
IOARPR2  DC     CL133' '                    BUFFER-2 PRINT FILE

WSP1     EQU    X'09'                       CTL CHAR: PRINT, SPACE 1
WSP2     EQU    X'13'                       *         PRINT, SPACE 2

BLANKPR  DC     C' '
PRINT    DS     0CL133                      PRINT AREA
CTLCHPR  DS     XL1                         *
         DC     CL132' '                    *
         LTORG
                =C'$$BOPEN '
                =C'$$BCLOSE'
                =A(FILEIDK)
                =A(WORKAREA)
                =A(FILEOPR)
                =A(PRINT)
                =H'9'
         END    PROG18C
```

Figure 18-5 (continued)

The DTFSD omits RECSIZE because IOCS needs to know only the maximum block length. For OS, the DCB entry for variable blocked format is RECFM = VB.

You could devise some records and trace the logic of this program step by step.

KEY POINTS

- Entries in a program file definition macro should match the job control commands.
- The block size for a file must be a multiple of record size, and all programs that process the file must specify the same record and block size.
- For variable-length files, the workareas and buffers should be aligned on an even boundary. When creating the file, you calculate and store the record length, whereas the system calculates the block length. Your designated maximum block size must equal or exceed the size of any record.

PROBLEMS

18-1. For blocked disk or tape records, under what circumstances would it be advisable to define only one buffer for the file?

18-2. Revise the program in Fig. 18-1 or 18-2 for six records per block and the use of locate mode.

18-3. Revise the file definition macro entries and I/O areas in Fig. 18-3 or 18-4 for the following. Input records are 90 bytes long and have six records per block. Output records have three records per block, to be loaded on a 3350 disk device as SYS017. Assemble and test.

18-4. Revise the job control for Fig. 18-3 for the following: The filename is DISKOUT, the file ID is ACCTS.RECEIVABLE, retention is 30 days, to be run on SYS017, serial number 123456, using a 3380 on cylinder 15, track 0 for 15 tracks.

18-5. Revise the job control for Figure 18-4 for the following. The filename is DISKOUT, the file ID is ACCTS.RECEIVABLE, retention is 30 days, to be run on SYS017, serial number 123456, using a 3380 on cylinder 15, track 0 for 15 tracks.

18-6. Code the file definition macro for DTFMT. The input file name is TAPFLIN, record size is fixed-length 500 bytes, the blocking factor is 5, on SYS030, two buffers, use of a workarea, and standard labels. The end-of-file address is X10EOF.

18-7. Code the file definition macro for DCB. The input file name is TAPFLIN, record size is fixed-length 500 bytes, the blocking factor is 5, use of a workarea, and standard labels. The end-of-file address is X10EOF.

18-8. Code the file definition macro for DTFSD. The input file name is DSKFLIN, record size is fixed-length 500 bytes, the blocking factor is 5, on SYS030, disk device 3380, two buffers, and use of a workarea. The end-of-file address is X10EOF.

18-9. Code the file definition macro for DCB. The input file name is DSKFLIN, record size is fixed-length 500 bytes, the blocking factor is 5, disk device 3380, and use of a workarea. The end-of-file address is X10EOF.

18-10. A file contains variable-length records with the following lengths: 326, 414, 502, 384, 293, 504. The maximum block length is 1,200 bytes. Arrange the records in blocks and show RCWs and BCWs.

18-11. Write a program that creates a supplier file on disk from the following input records:

01-05	Supplier number
06-25	Supplier name
26-46	Street
47-67	City
68-74	Amount payable
75-80	Date of last purchase (yymmdd)

Store name, street, and city as variable-length fields, with hex 'FF' as a delimiter after each field. Store the amount payable in packed format.

19

VIRTUAL STORAGE ACCESS METHOD (VSAM)

OBJECTIVE

To explain the design of the virtual storage access method and its processing requirements.

Virtual storage access method (VSAM) is a relatively recent file organization method for users of IBM OS/VS and DOS/VS. VSAM facilitates both sequential and random processing and supplies a number of useful utility programs.

The term *file* is somewhat ambiguous since it may reference an I/O device or the records that the device processes. To distinguish a collection of records, IBM OS literature uses the term *data set*.

VSAM provides three types of data sets:

1. *Key-sequenced Data Set (KSDS).* KSDS maintains records in sequence of key, such as employee or part number, and is equivalent to indexed sequential access method.
2. *Entry-sequenced Data Set (ESDS).* ESDS maintains records in the sequence in which they were initially entered and is equivalent to sequential organization.

Feature	Key-Sequenced	Entry-Sequenced	Relative-Record
Record sequence	By key	In sequence in which entered	In sequence of relative record number
Record length	Fixed or variable	Fixed or variable	Fixed only
Access of records	By key via index or RBA	By RBA	By relative record number
Change of address	Can change record RBA	Cannot change record RBA	Cannot change relative record number
New records	Distributed free space for records	Space at end of data set	Empty slots in data set
Recovery of space	Reclaims space if record is deleted	No delete but can overwrite an old record	Can reuse deleted space

Figure 19-1 Features of VSAM organization methods.

3. *Relative-Record Data Set (RRDS).* RRDS maintains records in order of relative record number and is equivalent to direct file organization.

Both OS/VS and DOS/VS handle VSAM the same way and use similar support programs and macros, although OS has a number of extended features.

Thorough coverage of assembler VSAM would require an entire textbook. However, this chapter supplies enough information to enable you to code programs that create, retrieve, and update a VSAM data set. For complete details, see the IBM Access Methods Services manual and the IBM DOS/VSE Macros or OS/VS Supervisor Services manuals.

CONTROL INTERVALS

For all three types of data sets, VSAM stores records in groups (one or more) of control intervals. You may select the control interval size, but if you allow VSAM to do so, it optimizes the size based on the record length and the type of disk device being used. The maximum size of a control interval is 32,768 bytes.

At the end of each control interval is control information that describes the data records:

Rec-1	Rec-2	Rec-3	. . .	Control Information

A control interval contains one or more data records, and a specified number of control intervals comprise a control area. VSAM addresses a data record by relative byte address (RBA)—its displacement from the start of the data set. Consequently, the first record of a data set is at RBA 0, and if records are 500 bytes long, the second record is at RBA 500.

The list in Fig. 19-1 compares the three types of VSAM organizations.

ACCESS METHOD SERVICES (AMS)

Before physically writing (or "loading") records in a VSAM data set, you first catalog its structure. The IBM utility package, Access Method Services (AMS), enables you to furnish VSAM with such details about the data set as its name, organization type, record length, key location, and password (if any). Since VSAM subsequently knows the physical characteristics of the data set, your program need not supply as much detailed information as would a program accessing an ISAM file.

The following describes the more important features of AMS. Full details are in the IBM OS/VS and DOS/VS Access Methods Services manual. You catalog a VSAM structure using an AMS program named IDCAMS, as follows:

```
OS:     //STEP EXEC PGM=IDCAMS
DOS:    // EXEC IDCAMS,SIZE=AUTO
```

Immediately following the command are various entries that DEFINE the data set. The first group under CLUSTER provides required and optional entries that describe all the information that VSAM must maintain for the data set. The second group, DATA, creates an entry in the catalog for a data component, that is, the set of all control area and intervals for the storage of records. The third group, INDEX, creates an entry in the catalog for a KSDS index component for the handling of the KSDS indexes.

Figure 19-2 provides the most common DEFINE CLUSTER entries. Note that to indicate continuation, a hyphen (-) follows every entry except the last.

Cluster level

```
DEFINE   CLUSTER
        ( NAME(data-set-name) -
         {CYLINDERS(primary[ secondary]) |
          BLOCKS(primary[ secondary]) |                 (choose
          RECORDS( primary[ secondary]) |                one)
          TRACKS(primary[ secondary])} -
         [INDEXED | NONINDEXED | NUMBERED] -            (choose one)
         [KEYS(length offset)] -
         [RECORDSIZE(average maximum)] -
         [VOLUMES(vol-ser[ vol-ser ...])]
```

Data component level

```
           [DATA
           ([CONTROLINTERVALSIZE(size)] -
            [NAME(data-name)] -
            [VOLUMES(vol-ser[ vol-ser ...])]
            )]
```

Index component level

```
           [INDEX
           ([NAME(index-name)] -
            [VOLUMES(vol-ser[ vol-ser ...])]
            )]
```

Figure 19-2 Entries for defining a VSAM data set.

Note:

SYMBOL	MEANING
[ ]	Optional entry, may be omitted
{ }	Select one of the following options
( )	You must code these parentheses
|	"or"

Figure 19-2 (continued)

- DEFINE CLUSTER (abbreviated DEF CL) provides various parameters all contained within parentheses.
- NAME is a required parameter that supplies the name of the data set. You can code the name up to 44 characters with a period after each 8 or fewer characters, as EMPLOYEE.RECORDS.P030. The name corresponds to job control, as follows:

```
OS:     //FILEVS DD DSNAME=EMPLOYEE.RECORDS.P030 ...
DOS:    // DLBL FILEVS,'EMPLOYEE.RECORDS.P030',0,VSAM
```

The name FILEVS in this example is whatever name you assign to the file definition (ACB) in your program, such as

```
            filename ACB DDNAME=FILEVS ...
```

- BLOCKS. You may want to load the data set on an FBA device (such as 3310 or 3370) or on a CKD device (such as 3350 or 3380). For FBA devices, allocate the number of 512-byte BLOCKS for the data set. For CKD devices, the entry CYLINDERS (or CYL) or TRACKS allocates space. The entry RECORDS allocates space for either FBA or CKD. In all cases, indicate a primary allocation for a generous expected amount of space and an optional secondary allocation for expansion if required.
- Choose one entry to designate the type of data set: INDEXED designates key-sequenced, NONINDEXED is entry-sequenced, and NUMBERED is relative-record.
- KEYS for INDEXED only defines the length (from 1 to 255) and position of the key in each record. For example, KEYS (6 0) indicates that the key is 6 bytes long beginning in position 0 (the first byte).
- RECORDSIZE (or RECSZ) provides the average and maximum lengths in bytes of data records. For fixed-length records and for RRDS, the two entries are identical. For example, code (120b120) for 120-byte records.
- VOLUMES (or VOL) identifies the volume serial number(s) of the DASD volume(s) where the data set is to reside. You may specify VOLUMES at

any of the three levels; for example, the DATA and INDEX components may reside on different volumes.

DEFINE CLUSTER supplies a number of additional specialized options described in the IBM AMS manual.

ACCESSING AND PROCESSING

VSAM furnishes two types of accessing, keyed and addressed, and three types of processing, sequential, direct, and skip sequential. The following chart shows the legal accessing and processing by type of organization:

Type	Keyed Access	Addressed Access
KSDS	Sequential Direct Skip sequential	Sequential Direct
ESDS		Sequential Direct
RRDS	Sequential Direct Skip sequential	

In simple terms, *keyed accessing* is concerned with the key (for KSDS) and relative record number (for RRDS). For example, if you read a KSDS sequentially, VSAM delivers the records in sequence by key (although they may be in a different sequence physically).

Addressed accessing is concerned with the RBA. For example, you can access a record in an ESDS using the RBA by which it was stored. For either type of accessing method, you can process records sequentially or directly (and by skip sequential for keyed access). Thus you always use addressed accessing for ESDS and keyed accessing for RRDS and may process either type sequentially or directly. KSDS, by contrast, permits both keyed access (the normal) and addressed access, with both sequential and direct processing.

KEY-SEQUENCED DATA SETS

A key-sequenced data set (KSDS) is considerably more complex than either ESDS or RRDS but is more useful and versatile. You always create ("load") a KSDS in ascending sequence by key and may process a KSDS directly by key or sequentially. Since KSDS stores and retrieves records according to key, each key in the data set must be unique.

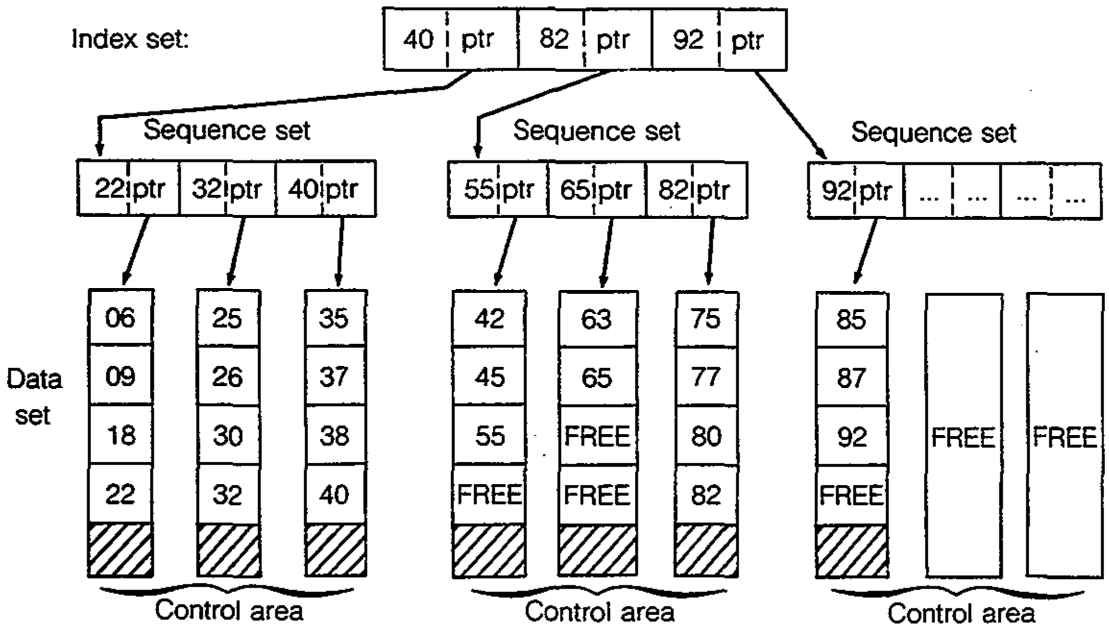


Figure 19-3 Key-sequenced organization.

Figure 19-3 provides a simplified view of a key-sequenced data set. The control intervals that contain the data records are depicted vertically, and for this example three control intervals comprise a control area. A *sequence set* contains an entry for each control interval in a control area. Entries within a sequence set consist of the highest key for each control interval and the address of the control interval; the address acts as a pointer to the beginning of the control interval. The highest keys for the first control area are 22, 32, and 40, respectively. VSAM stores each high key along with an address pointer in the sequence set for the first control area.

At a higher level, an *index set* (various levels depending on the size of the data set) contains high keys and address pointers for the sequence sets. In Fig. 19-3, the highest key for the first control area is 40. VSAM stores this value in the index set along with an address pointer for the first sequence.

When a program wants to access a record in the data set directly, VSAM locates the record first by means of the index set and then the sequence set. For example, a program requests access to a record with key 63. VSAM first checks the index set as follows:

RECORD KEY	INDEX SET	
63	40	Record key high, not in first control area.
63	82	Record key low, in second control area.

VSAM has determined that key 63 is in the second control area. It next examines the sequence set for the second control area to locate the correct control interval. These are the steps:

RECORD KEY	SEQUENCE SET	
63	55	Record key high, not in first control interval.
63	65	Record key low, in second control interval.

VSAM has now determined that key 63 is in the second control interval of the second control area. The address pointer in the sequence set directs VSAM to the correct control interval. VSAM then reads the keys of the data set and locates key 63 as the first record that it delivers to the program.

Free Space

You normally allow a certain amount of free space in a data set for VSAM to insert new records. When creating a key-sequenced data set, you can tell VSAM to allocate free space in two ways:

1. Leave space at the end of each control interval.
2. Leave some control intervals vacant.

If a program deletes or shortens a record, VSAM reclaims the space by shifting to the left all following records in the control interval. If the program adds or lengthens a record, VSAM inserts the record in its correct space and moves to the right all following records in the control interval. VSAM updates RBAs and indexes accordingly.

A control interval may not contain enough space for an inserted record. In such a case, VSAM causes a *control interval split* by removing about half the records to a vacant control interval in the same control area. Although records are now no longer *physically* in key order, for VSAM they are *logically* in sequence. The updated sequence set controls the order for subsequent retrieval of records.

If there is no vacant control interval in a control area, VSAM causes a control area split, using free space outside the control area. Under normal conditions, such a split seldom occurs. To a large degree, a VSAM data set is self-organizing and requires reorganization less often than an ISAM file.

ENTRY-SEQUENCED DATA SETS

An entry-sequenced data set (ESDS) acts like sequential file organization but has the advantages of being under control of VSAM, some use of direct processing, and password facilities. Basically, the data set is in the sequence in which it is created, and you normally (but not necessarily) process from the start to the end of the data set. Sequential processing of an ESDS by RBA is known as addressed access, which is the method you use to create the data set. You may also process ESDS records directly by RBA. Since ESDS is not concerned with keys, the data set may legally contain duplicate records.

Assume an ESDS containing records with keys 001, 003, 004, and 006. The data set would appear as follows:

| 001 | 003 | 004 | 006 | .

You may want to use ESDS for tables that are to load into programs, for small files that are always in ascending sequence, and for files extracted from a KSDS that are to be sorted.

RELATIVE-RECORD DATA SETS

A relative-record data set (RRDS) acts like direct file organization but also has the advantages of being under control of VSAM and offering keyed access and password facilities. Basically, records in the data set are located according to their keys. For example, a record with key 001 is in the first location, a record with key 003 is in the third location, and so forth. If there is no record with key 002, that location is empty, and you can subsequently insert the record.

Assume an RRDS containing records with keys 001, 003, 004, and 006. The data set would appear as follows:

| 001 | ... | 003 | 004 | ... | 006 |

Since RRDS stores and retrieves records according to key, each key in the data set must be unique.

You may want to use RRDS where you have a small to medium-sized file and keys are reasonably consecutive so that there are not large numbers of spaces. One example would be a data set with keys that are regions or states, and contents are product sales or population and demographic data.

You could also store keys after performing a computation on them. As a simple example, imagine a data set with keys 101, 103, 104, and 106. Rather than store them with those keys, you could subtract 100 from the key value and store the records with keys 001, 003, 004, and 006.

VSAM MACRO INSTRUCTIONS

VSAM uses a number of familiar macros as well as a few new ones to enable you to retrieve, add, change, and delete records. In the following list, for macros marked with an asterisk (*), see the IBM DOS/VS or OS/VS Supervisor and I/O Macros manual for details.

- To relate a program and the data:

ACB	(access method control block)
EXLST	(exit list)

- To connect and disconnect a program and a data set:

OPEN	(open a data set)
CLOSE	(close a data set)
TCLOSE*	(temporary close)

- To define requests for accessing data:

RPL	(request parameter list)

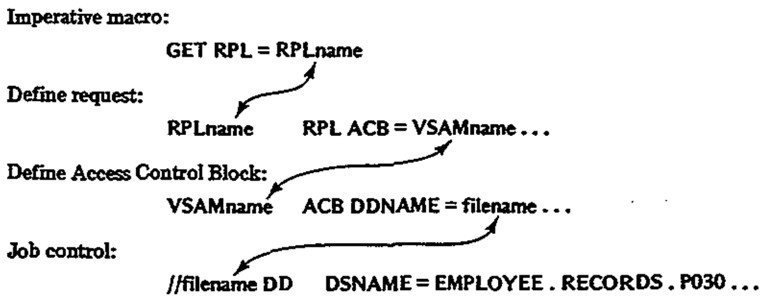
- To request access to a file:

GET	(get a record)
PUT	(write or rewrite a record)
POINT*	(position VSAM at a record)
ERASE	(erase a record previously retrieved with a GET)
ENDREQ*	(end request)

- To manipulate the information that relates a program to the data:

GENCB*	(generate control block)
MODCB*	(modify control block)
SHOWCB	(show control block)
TESTCB*	(test control block)

A program that accesses a VSAM data set requires the usual OPEN to connect the data set and CLOSE to disconnect it, the GET macro to read records, and PUT to write or rewrite records. An important difference in the use of macros under VSAM is the RPL (request for parameter list) macro. As shown in the following relationship, a GET or PUT specifies an RPL macro name rather than a file name. The RPL in turn specifies an ACB (access control block) macro, which in its turn relates to the job control entry for the data set:



```
Imperative macro:
                  GET RPL = RPLname
Define request:
                  RPLname      RPL ACB = VSAMname . . .
Define Access Control Block:
                  VSAMname     ACB DDNAME = filename . . .
Job control:
                  //filename DD    DSNAME = EMPLOYEE . RECORDS . P030 . . .
```

The ACB macro is equivalent to the OS DCB or DOS DTF file definition macros. As well, the OPEN macro supplies information about the type of file organization, record length, and key. Each execution of OPEN, CLOSE, GET, PUT, and ERASE causes VSAM to check its validity and to insert a code into register 15 that you can check. A return code of X'00' means that the operation was successful. You can use the SHOWCB macro to determine the exact cause of the error.

THE ACB MACRO: ACCESS METHOD CONTROL BLOCK

The ACB macro identifies a data set that is to be processed. Its main purpose is to indicate the proposed type of processing (sequential or direct) and the use of exit routines, if any. The DEFINE CLUSTER command of AMS has already stored much of the information about the data set in the VSAM catalog. When a program opens the data set via the ACB, VSAM delivers this information to virtual storage.

Entries for an ACB macro may be in any sequence, and you may code only those that you need. Following is the general format, which you code like a DCB or DTF, with a comma following each entry and a continuation character in column 72. All operands are optional.

```
name      ACB      AM=VSAM,                             +
                   DDNAME=filename,                     +
                   EXLST=address,                       +
                   MACRF=([ADR][,KEY]
                          [,DIR][,SEQ][,SKP]
                          [,IN][,OUT]
                          [,NRM|AIX]),                  +
                   STRNO=number
```

name	The name indicates the symbolic address for the ACB when assembled. If you omit the DDNAME operand from the ACB definition, this name should match the filename in your DLBL or DD job statement.
AM=VSAM	Code this parameter if your installation also uses VTAM; otherwise, the assembler assumes VSAM.
DDNAME	This entry provides the name of your data set that the program is to process. This name matches the filename in your DLBL or DD job statement.
EXLST	The address references a list of your addresses of routines that provide exits. Use the EXLST macro to generate the list, and enter its name as the address. A common use is to code an

	entry for an end-of-file exit for sequential reading. If you have no exit routines, omit the operand.
`MACRF`	The options define the type of processing that you plan. In the following, an underlined entry is a default:

`ADR` | <u>`KEY`</u>	Use ADR for addressed access (KS and ES) and KEY for keyed access (KS and RR).
`DIR` | <u>`SEQ`</u> | `SKP`	DIR provides direct processing, SEQ provides sequential processing, and SKP means skip sequential (for KS and RR).
<u>`IN`</u> | `OUT`	IN retrieves records and OUT permits retrieval, insertion, add-to-end, or update for keyed access and retrieval, update, or add-to-end for addressed access.
<u>`NRM`</u> | `AIX`	The DDNAME operand supplies the name of the data set (or path). NRM means normal processing of the data set, whereas AIX means that this is an alternate index.

	Other MACRF options are RST | <u>NRS</u> for resetting catalog information and <u>NUB</u> | UBF for user buffers.
`STRNO`	The entry supplies the total number of RPLs (request parameter lists) that your program will use at the same time (the default is 1).

ACB also has provision for parameters that define the number and size of buffers; however, the macro has standard defaults.

In the program example in Fig. 19-4, the ACB macro VSMFILOT has only two entries and allows the rest to default. Access is keyed (KEY), processing is sequential (SEQ), and the file is output (OUT). There is no exit list, STRNO defaults to 1, and MACRF defaults to NRM (normal path).

The assembler does not generate an I/O module for an ACB, nor does the linkage editor include one. Instead, the system dynamically generates the module at execute time.

THE RPL MACRO: REQUEST PARAMETER LIST

The request macros GET, PUT, ERASE, and POINT require a reference to an RPL macro. For example, the program in Fig. 19-4 issues the following GET macro:

```
GET  RPL=RPLISTIN
```

The operand supplies the name of the RPL macro that contains the information needed to access a record. If your program is to access a data set in different ways, you can code an RPL macro for each type of access; each RPL keeps track of its location in the data set.

The standard format for RPL is as follows. The name for the RPL macro is the one that you code in the GET or PUT operand. Every entry is optional.

```
RPLname    RPL    AM=VSAM,                +
                  ACB=address,            +
                  AREA=address,           +
                  AREALEN=length,         +
                  ARG=address,            +
                  KEYLEN=length,          +
                  OPTCD=(options),        +
                  RECLEN=length
```

AM	The entry VSAM specifies that this is a VSAM (not VTAM) control block.
ACB	The entry gives the name of the associated ACB that defines the data set.
AREA	The address references an I/O workarea in which a record is available for output or is to be entered on input.
AREALEN	The entry supplies the length of the record area.
ARG	The address supplies the search argument—a key, including a relative record number or an RBA.
KEYLEN	The length is that of the key if processing by generic key. (For normal keyed access, the catalog supplies the key length.)
OPTCD	Processing options are SEQ, SKP, and DIR; request options are UPD (update) and NUP (no update). For example, a direct update would be (DIR,UPD).
RECLEN	For writing a record, your program supplies the length to VSAM, and for retrieval, VSAM supplies the length to your program. If records are variable length, you can use the SHOWCB and TESTCB macros to examine the field (see the IBM Supervisor manual).

THE OPEN MACRO

The OPEN macro ensures that your program has authority to access the specified data set and generates VSAM control blocks.

```
[label]  OPEN  address[,address ... ]
```

The operand designates the address of one or more ACBs, which you may code either as a macro name or as a register notation (registers 2–12); for example:

```
   OPEN VSFILE
or LA   6,VSFILE
   OPEN (6)
```

You can code up to 16 filenames in one OPEN and can include both ACB names and DCB or DTF names. Note, however, that to facilitate debugging, avoid mixing them in the same OPEN. OPEN sets a return code in register 15 to indicate success (zero) or failure (nonzero), which your program can test:

X'00'	Opened all ACBs successfully.
X'04'	Opened all ACBs successfully but issued a warning message for one or more.
X'08'	Failed to open one or more ACBs.

On a failed OPEN or CLOSE, you can also check the diagnostics following program execution for a message such as OPEN ERROR X'6E', and check Appendix K of the IBM Supervisor manual for an explanation of the code.

THE CLOSE MACRO

The CLOSE macro completes any I/O operations that are still outstanding, writes any remaining output buffers, and updates catalog entries for the data set.

```
[label] CLOSE address[,address ... ]
```

You can code up to 16 names in one CLOSE and can include both ACB names and DCB or DTF names. CLOSE sets a return code in register 15 to indicate success or failure, which your program can test:

X'00	Closed all ACBs successfully.
X'04'	Failed to close one or more ACBs successfully.
X'08'	Insufficient virtual storage space for close routine or could not locate modules.

THE REQUEST MACROS: GET, PUT, ERASE

The VSAM request macros are GET, PUT, ERASE, POINT, and ENDREQ. For each of these, VSAM sets register 15 with a return code to indicate success or failure of the operation, as follows:

`X'00'`	Successful operation.
`X'04'`	Request not accepted because of an active request from another task on the same RPL. End-of-file also causes this return code.
`X'08'`	A logical error; examine the specific error code in the RPL.
`X'0C'`	Uncorrectible I/O error; examine the specific error code in the RPL.

The GET Macro

GET retrieves a record from a data set. The operand specifies the address of an RPL that defines the data set being processed. The entry may either (1) cite the address by name or (2) use register notation, any register 2–12, in parentheses. You may use register 1; its use is more efficient, but GET does not preserve its address.

```
1. GET   RPL=RPLname
2. LA    reg,RPLname
   GET   RPL=(reg)
```

The RPL macro provides the address of your workarea where GET is to deliver an input record. Register 13 must contain the address of a savearea defined as 18 fullwords.

Under sequential input, GET delivers the next record in the data set. The OPTCD entry in the RPL macro would appear, for example, as OPTCD=(KEY,SEQ) or OPTCD=(ADR,SEQ). You have to provide for end-of-file by means of an EXLST operand in the associated ACB macro; see Fig. 19-4 for an example.

For nonsequential accessing, GET delivers the record that the key or relative record number specifies in the search argument field. The OPTCD entry in the RPL macro would appear, for example, as OPTCD=(KEY,SKP) or OPTCD=(KEY,DIR), or as an RBA in the search argument field, as OPTCD=(ADR,DIR).

You also use GET to update or delete a record.

The PUT Macro

PUT writes or rewrites a record in a data set. The operand of PUT specifies the address of an RPL that defines the data set being processed. The entry may either (1) cite the address by name or (2) use register notation, any register 2–12, in parentheses. You may use register 1; its use is more efficient, but PUT does not preserve its address.

```
1. PUT   RPL=RPLname
2. LA    reg,RPLname
   PUT   RPL=(reg)
```

The RPL macro provides the address of your workarea containing the record that PUT is to add or update in the data set. Register 13 must contain the address of a savearea defined as 18 fullwords.

To create (load) or extend a data set, use sequential output. The OPTCD entry in the RPL macro would appear, for example, as OPTCD=(SEQ or SKP). SKP means "skip sequential" and enables you to start writing at any specific record.

For writing a KSDS or RRDS, if OPTCD contains any of the following, PUT stores a new record in key sequence or relative record sequence:

`OPTCD=(KEY,SKP,NUP)`	Skip, no update
`OPTCD=(KEY,DIR,NUP)`	Direct, no update
`OPTCD=(KEY,SEQ,NUP)`	Sequential, no update

Note that VSAM does not allow you to change a key in a KSDS (delete the record and write a new one). To change a record, first GET it using OPTCD=UPD, change its contents (but not the key), and PUT it, also using OPTCD=UPD. To write a record in ESDS, use OPTCD=(ADR, . . .).

The ERASE Macro

The purpose of the ERASE macro is to delete a record from a KSDS or an RRDS. To locate an unwanted record, you must previously issue a GET with an RPL specifying OPTCD=(UPD. . .).

```
[label]  ERASE  RPL=address or =(register)
```

For ESDS, a common practice is to define a delete byte in the record. To "delete" a record, insert a special character such as X'FF'; all programs that process the data set should bypass all records containing the delete byte. You can occasionally rewrite the data set, dropping all deletes.

THE EXLST MACRO

If your ACB macro indicates an EXLST operand, code a related EXLST macro. EXLST provides an optional list of addresses for user exit routines that handle end-of-file and error analysis. All operands in the macro are optional.

`[label]`	`EXLST`	`AM=VSAM,`
		`EODAD=address,`
		`LERAD=address,`
		`SYNAD=address`
			+
			+
			+

When VSAM detects the coded condition, the program enters your exit routine. Register 13 must contain the address of your register savearea. For example, if you are reading sequentially, supply an end-of-data address (EODAD) in the EXLST macro—see the ACB for VSMFILIN in Fig. 19-4.

Here are explanations of the operands for EXLST:

VSAM	Indicates a VSAM control block.
EODAD	Supplies the address of your end-of-data routine. You may also read sequentially backward, and VSAM enters your routine when reading past the first record. The request return code for this condition is X'04'.
LERAD	Indicates the address of the routine that analyzes logical errors that occurred during GET, PUT, POINT, and ERASE. The request return code for this condition is X'08'.
SYNAD	Provides the address of your routine that analyzes physical I/O errors on GET, PUT, POINT, ERASE, and CLOSE. The request return code for this condition is X'0C'.

Other operands are EXCPAD and JRNAD.

THE SHOWCB MACRO

The original program in Fig. 19-4 contained an error that caused it to fail on a PUT operation. The use of the SHOWCB macro in the error routine for PUT (R30PUT) helped determine the actual cause of the error.

The purpose of SHOWCB is to display fields in an ACB, EXLST, or RPL. Code SHOWCB following a VSAM macro where you want to identify errors that VSAM has detected. The SHOWCB in the PUT error routine in Fig. 19-4 is as follows:

```
         SHOWCB  RPL=RPLISTOT,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
         ...
FDBKWD   DC      F'0'
```

AREA	Designates the name of a fullword where VSAM is to place an error code.
FIELDS	Tells SHOWCB the type of display; the keyword FDBK (feedback) causes a display of error codes for request macros.
LENGTH	Provides the length of the area in bytes.

On a failed request, VSAM stores the error code in the rightmost byte of the fullword area. These are some common error codes:

08	Attempt to store a record with a duplicate key.
0C	Out-of-sequence or duplicate record for KSDS or RRDS.

10	No record located on retrieval.
1C	No space available to store a record.

Your program can test for the type of error and display a message. For nonfatal errors, it could continue processing; for fatal errors, it could terminate.

The original error in Fig. 19-4 was caused by the fact that the RPL macro RPLISTOT did not contain an entry for RECLEN; the program terminated on the first PUT error, with register 15 containing X'08' (a "logical error"). Insertion of the SHOWCB macro in the next run revealed the cause of the error in FDBKWD: 00006C. Appendix K of the IBM Supervisor manual explains the error (in part) as follows: "The RECLEN value specified in the RPL macro was [either] larger than the allowed maximum [or] equal to zero. . . ." Coding a RECLEN operand in the RPL macro solved the problem, and the program then executed through to normal termination. One added point: Technically, after each SHOWCB, you should test register 15 for a successful or failed operation.

SAMPLE PROGRAM: LOADING A KEY-SEQUENCED DATA SET

The program in Fig. 19-4 reads records from the system reader and sequentially creates a key-sequenced data set. A DEFINE CLUSTER command has allocated space for this data set as INDEXED (KSDS), with three tracks, a 4-byte key starting in position 0, and an 80-byte record size. The program loads the entire data set and closes it on completion. For illustrative (but not practical) purposes, it reopens the data set and reads and prints each record.

The PUT macro that writes records into the data set is:

```
PUT RPL=RPLISTOT
```

RPLISTOT defines the name of the ACB macro (VSMFILOT), the address of the output record, and its length. Although the example simply duplicates the records into the data set, in practice you would probably define various fields and store numeric values as packed or binary.

The ACB macro defines VSMFILOT for keyed accessing, sequential processing, and output. The DDNAME, VSAMFIL, in this example relates to the name for the data set in the DLBL job control entry (DD under OS).

For reading the data set, the GET macro is

```
GET RPL=RPLISTIN
```

RPLISTIN defines the name of the ACB macro (VSMFILIN), the address in which GET is to read an input record, and the record length.

The ACB macro defines VSMFILIN for keyed access, sequential processing, and input. The DDNAME, VSAMFIL, relates to the name for the data set in

```
IDCAMS  SYSTEM SERVICES

   DELETE (VSAMFIL.ABEL) CLUSTER PURGE
IDC0550I ENTRY (C) VSAMFIL.ABEL DELETED
IDC0550I ENTRY (D) VSAMFIL.DATA DELETED
IDC0550I ENTRY (I) VSAMFIL.INDEX DELETED
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

   DEFINE CLUSTER (NAME(VSAMFIL.ABEL) -
                  TRACKS(3) -
                  VOLUME(SVSE03) -
                  INDEXED -
                  KEYS(4 0) -
                  RECORDSIZE(80 80) ) -
             DATA (NAME(VSAMFIL.DATA) ) -
             INDEX (NAME(VSAMFIL.INDEX) )

IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0

// OPTION LINK,PARTDUMP,NOXREF,LOG
   ACTION NOMAP
// EXEC ASSEMBLY,SIZE=256K

  3          PRINT NOGEN,NODATA
  4 *              M A I N   P R O C E S S I N G
  5 *              ----------------------------
  6 PROGVSM  START
  7          BALR  12,0                      INITIALIZE
  8          USING *,12                        BASE REG &
  9          LA    13,VSAMSAVE                 VSAM SAVEAREA
 10          OPEN  FILEIN,VSMFILOT
 19          LTR   15,15                     SUCCESSFUL OPEN?
 20          BNZ   R10OPEN                     NO - TERMINATE
 21          GET   FILEIN,VSMREC             READ 1ST RECORD
 28 A10LOOP  BAL   6,B10LOAD                 CREATE FILE
 29          GET   FILEIN,VSMREC             READ NEXT
 35          B     A10LOOP

 37 A80EOF   CLOSE FILEIN,VSMFILOT
 46          LA    13,VSAMSAVE
 47          OPEN  FILEPRT,VSMFILIN
 56          LTR   15,15                     SUCCESSFUL OPEN?
 57          BNZ   R10OPEN                     NO -- TERMINATE
 58          BAL   6,C10PRINT                READ & PRINT VSAM FILE
 60 A90EOF   CLOSE FILEPRT,VSMFILOT
 69          EOJ                             NORMAL TERMINATION

 73 *              L O A D   V S A M   F I L E
 74 *              -------------------------
 75 B10LOAD  PUT   RPL=RPLISTOT              WRITE VSAM RECORD
 82          LTR   15,15                     SUCCESSFUL WRITE?
 83          BNZ   R30PUT                     NO --ERROR
 84          BR    6                         RETURN

 86 *              R E A D   &   P R I N T   V S A M   F I L E
 87 *              ---------------------------------------
 88 C10PRINT GET   RPL=RPLISTIN
 95          LTR   15,15                     SUCCESSFUL READ?
 96          BNZ   R40GET                      NO - TERMINATE
```

Figure 19-4 Loading a key-sequenced data set.

```
         MVC   PRREC,VSMREC
         PUT   FILEPRT,PRINT                 PRINT RECORD
         B     C10PRINT

*              E R R O R   R O U T I N E S
*              -------------------------
R10OPEN  MVI   ERRCDE,C'O'                   OPEN ERROR
         B     R90DUMP
R30PUT   MVI   ERRCDE,C'P'                   PUT ERROR
         ST    15,SAVE15
         SHOWCB RPL=RPLISTOT,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
         CLOSE FILEIN,VSMFILOT
         B     R90DUMP
R40GET   MVI   ERRCDE,C'G'                   GET ERROR
         ST    15,SAVE15
         SHOWCB RPL=RPLISTIN,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
         CLOSE FILEPRT,VSMFILOT
R90DUMP  EQU   *
         PDUMP ERRCDE,PRINT+133
         EOJ                                 ABNORMAL TERMINATION

*              -------------------------
*              D E C L A R A T I V E S
*              -------------------------
FILEIN   DEFIN A80EOF                        DEFINE INPUT FILE
FILEPRT  DEFPR                               DEFINE PRINTER FILE
VSMFILOT ACB   DDNAME=VSAMFIL,               DEFINE VSAM O/P FILE    +
               MACRF=(KEY,SEQ,OUT)

RPLISTOT RPL   ACB=VSMFILOT,                 RPL FOR VSMFILOT        +
               AREA=VSMREC,                                          +
               AREALEN=80,                                           +
               RECLEN=80,                                            +
               OPTCD=(KEY,SEQ,NUP)

VSMFILIN ACB   DDNAME=VSAMFIL,               DEFINE VSAM I/P FILE    +
               MACRF=(KEY,SEQ,IN),                                   +
               EXLST=EOFDCB

EOFDCB   EXLST EODAD=A90EOF                  EOF EXIT FOR VSAM I/P
RPLISTIN RPL   ACB=VSMFILIN,                 RPL FOR VSMFILIN        +
               AREA=VSMREC,                                          +
               AREALEN=80,                                           +
               OPTCD=(KEY,SEQ,NUP)

VSAMSAVE DS    18F                           VSAM SAVEAREA
ERRCDE   DC    X'00'                         ERROR CODE
SAVE15   DS    F
FDBKWD   DC    F'0'
VSMREC   DS    0CL80                         INPUT/OUTPUT RECORD
RECKEY   DS    CL04                          *
         DS    CL76                          *

PRINT    DS    0CL133                        PRINT RECORD
         DC    X'09'                         *
PRREC    DC    CL80' '                       *
         DC    CL52' '                       *
         LTORG
               =C'$$BOPEN '
               =C'$$BCLOSE'
               =CL8'IKQVTMS'
```

Figure 19-4 (continued)

```
                  =CL8'$$BPDUMP'
                  =A(ERRCDE,PRINT+133)
                  =A(FILEIN)
                  =A(VSMREC)
                  =A(RPLISTOT)
                  =A(RPLISTIN)
                  =A(FILEPRT)
                  =A(PRINT)
             END  PROGVSM
// EXEC LNKEDT,SIZE=128K

// DLBL VSAMFIL,'VSAMFIL.ABEL',,VSAM
// EXTENT SYS008,SVSE03
// ASSGN SYS008,X'303'
// EXEC ,SIZE=128K

Output:-

   0034AES PROCESSORS            (Output from
   0047MICROTEL INDUSTRIES        printing contents
   0065ACE ELECTRONICS            of loaded data set)
```

Figure 19-4 (continued)

the DLBL job control entry. Note that there is an ACB and RPL macro for both input and output, but both ACB macros specify the same DDNAME: VSAMFIL.

Error routines are for failures on OPEN, GET, and PUT. These rather primitive routines supply an error code and the contents of the declaratives; in practice, you may want to enlarge these routines. If you fail to provide error routines, your program may crash with no clear cause.

During testing, you may have changed the contents of a VSAM data set and now want to reload (re-create) the original data set. Except for updating with new keys, VSAM does not permit overwriting records in a data set. You have to use IDCAMS to DELETE and again DEFINE the data set as follows:

```
DELETE(data-set-name) CLUSTER PURGE ...
DEFINE CLUSTER(NAME(data-set-name) -
...
```

Loading an ESDS

To convert the program in Fig. 19-4 from KSDS to ESDS, change DEFINE CLUSTER from INDEXED to NONINDEXED and delete the KEYS and INDEX entries. Change the ACB MACRF from KEY to ADR, and change the RPL OPTCD from KEY to ADR—that's all.

KEYED DIRECT RETRIEVAL

Key-sequenced data sets provide for both sequential and direct processing by key. For direct processing, you must supply VSAM with the key of the record to be accessed. If you use a key to access a record directly, it must be the same length as the keys in the data set (as indicated in the KEYS operand of DEFINE CLUSTER), and the key must actually exist in the data set. For example, if you request a record with key 0028 and there is no such record, VSAM returns an error code in register 15.

Using the data set in Fig. 19-4, assume that a program is to access records directly. A user enters record key numbers via a terminal, and the program is to display the record on the screen. In this partial example, the RPL macro specifies the name (ARG) of the key to be in a 4-byte field named KEYFLD. These are the specific coding requirements for the ACB, RPL, and GET macros:

```
VSMFILE  ACB  DDNAME=name,                    +
              MACRF=(KEY,DIR,IN)

RPLIST   RPL  ACB=VSMFILE,                    +
              AREA=DCBREC,                    +
              AREALEN=80,                     +
              ARG=KEYFLD,                     +
              OPTCD=(KEY,DIR,NUP)
KEYFLD   DS   CL4
DCBREC   DS   CL80
         ...
        [Accept a key number from the terminal]
         MVC  KEYFLD,keyno
         GET  RPL=RPLIST
         LTR  15,15
         BNZ  error
        [Display the record on the screen]
```

For updating a KSDS record, change the MACRF from IN to OUT, and change the OPTCD from NUP to UPD. GET the record, make the required changes to it (but not the key!), and PUT the record using the same RPL.

SORTING VSAM FILES

You can sort VSAM records into either ascending or descending sequence. You must first use DEFINE CLUSTER to allocate a vacant data set (NONINDEXED) for SORT to write the sorted data set. Here is a typical SORT specification:

```
// EXEC SORT,SIZE=256K
     SORT FIELDS=(1,4,CH,A,9,4,PD,D)
     RECORD TYPE=F,LENGTH=(150)
     INPFIL VSAM
     OUTFIL ESDS
     END
/*
```

SORT causes the SORT program to load into storage and begin execution.

SORT FIELDS defines the fields to be sorted, indicated by major control to minor, from left to right. In this example, the major sort field begins in position 1 (the first position), is 4 bytes long, is in character (CH) format, and is to be sorted in ascending (A) sequence. The minor sort field begins in position 9, is 4 bytes long, is in packed (PD) format, and is to be sorted in descending (D) sequence. The example could be a sort of departments in ascending sequence, and within each department are employee salaries in descending sequence.

RECORD TYPE indicates fixed (F) length and record length (150 bytes).

INPFIL informs SORT that the input file is VSAM; SORT can determine the type of data set from the VSAM catalog.

OUTFIL defines the type of output file, in this case entry-sequenced. This entry should match the DEFINE CLUSTER for this data set, NONINDEXED.

Job control commands for SORTIN and SORTOUT provide the names of the data sets. Since job control varies by operating system and by installation requirements, check with your installation before attempting the SORT utility.

VSAM UTILITY PRINT

IDCAMS furnishes a convenient utility program named PRINT that can print the contents of a VSAM, SAM, or ISAM data set. The following provides the steps for OS and for DOS:

```
OS:     //STEP EXEC PGM=IDCAMS
         PRINT INFILE(filename) CHARACTER or HEX or DUMP
        /*
DOS:    // EXEC IDCAMS,SIZE=256K
         PRINT INFILE(filename) CHARACTER or HEX or DUMP
        /*
```

The options for PRINT indicate the format of the printout, in character, hexadecimal, or both (DUMP prints hex on the left and character format on the right).

INFILE(filename) matches the name in the OS DD or DOS DLBL job statement with any valid filename as long as the two are identical. The DD or DLBL statement notifies VSAM which data set is to print.

PRINT lists KSDS and ISAM data sets in key sequence and lists ESDS, RRDS, and SAM data sets in physical sequence. You can also print beginning and ending at a specific record.

KEY POINTS

- A key-sequenced data set (KSDS) maintains records in sequence of key, such as employee or part number, and is equivalent to indexed sequential access method.
- An entry-sequenced data set (ESDS) maintains records in the sequence in which they were initially entered and is equivalent to sequential organization.
- A relative-record data set (RRDS) maintains records in order of relative record number and is equivalent to direct file organization.
- For the three types of data sets, VSAM stores records in groups (one or more) of control intervals. At the end of each control interval is control information that describes the data records.
- Before physically writing (loading) records in a VSAM data set, you must first catalog its structure. Access method services (AMS) enables you to furnish VSAM with such details about the data set as its name, organization type, record length, key location, and password (if any).
- VSAM furnishes two types of accessing, keyed and addressed, and three types of processing, sequential, direct, and skip sequential.
- The most common errors in processing VSAM data sets occur because of the need to match definitions in the program, job control, and the cataloged VSAM data set.
- The data-set-name in job control (such as CUSTOMER.INQUIRY) must agree with the NAME(data-set-name) entry in DEFINE CLUSTER. This name is the only one by which VSAM recognizes the data set. VSAM relates the ACB DDNAME in the program to the job control name and the job control name to the data-set-name.
- If a data set is cataloged as KSDS, ESDS, or RRDS, each program must access it accordingly.
- For KSDS, the length and starting position of the key in a record must agree with the KEYS entry in DEFINE CLUSTER and, for direct input, with the defined ARG in the OPTCD.
- Every program that references the data set defines the fields with identical formats and lengths in the same positions; the actual field names need not be identical. You may define as character any input field in a record that the program does not reference. The simplest practice is to catalog all record definitions in the assembler source library and COPY the definition into the program during assembly.
- After each OPEN, CLOSE, GET, PUT, and SHOWCB, test register 15 for success or failure, and use SHOWCB (as well as TESTCB) as a debugging aid.

PROBLEMS

19-1. What are the three types of VSAM data sets, and how do they differ?

19-2. Explain control interval, control interval split, and RBA.

19-3. Assume a KSDS that contains records with keys in two control areas as follows:

Control area 1:	360, 373, 385
	390, 412, 415
Control area 2:	420, 475, 480
	512, 590, 595

What are the contents of (a) the two sequence sets; (b) the index set?

19-4. What is the program that catalogs the structure of a VSAM data set, and what are its three component levels?

19-5. Code DEFINE CLUSTER for the data-set-name CUSTOMER.FILE, assuming 20 blocks, ESDS, and 100-byte records.

19-6. Code job control (OS DD and DOS DLBL) for the data set in Problem 19-5 with filename CUSTVS.

19-7. Code the ACB macro named CUSVSIN for the data set in Problem 19-6 for addressed sequential input and an EXLST macro named EOFCUS for end-of-file.

19-8. Code the RPL macro named RPLCUSIN for the ACB in Problem 19-7 with an input area named CUSVSREC.

19-9. Code the GET macro to read the data set in Problem 19-8.

19-10. Write a program that creates a KSDS supplier file from the following input records:

```
01-05   Supplier number
06-25   Supplier name
26-46   Street
47-67   City
68-74   Amount payable
75-80   Date of last purchase (yymmdd)
```

Store the amount payable in packed format.

20

INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

OBJECTIVE

To explain the design of indexed sequential access method and its processing requirements.

Indexed sequential access method (ISAM) is available in many variations on microcomputers, minicomputers, and mainframes, although the preferred method under DOS/VS and OS/VS is the newer VSAM.

A significant way in which ISAM (and other nonsequential file organization methods) differs from sequential organization is that the record keys in an indexed file must be unique; this is a system requirement, not just a programming practice. Consequently, an indexed file is typically a master file. Also, there is a clear difference between updating a sequential file and updating an indexed file. When you update a sequential file, you rewrite the entire file; this practice leaves the original file as a convenient backup in case the job must be rerun. When you update an indexed file, the system rewrites records in the file directly in place, thereby providing no automatic backup file. To create a backup, you periodically copy the file onto another device.

The flexibility of indexed sequential access method is realized at some cost in both storage space and accessing time. First, the system requires various levels of indexes to help locate records in the file. Second, the system stores new, added records in special reserved overflow areas.

Check that your system supports ISAM before attempting to use it.

CHARACTERISTICS OF INDEXED SEQUENTIAL FILES

ISAM initially stores records sequentially and permits both sequential and random processing. The features that provide this flexibility are indexes to locate a correct cylinder and track and keys to locate a record on a track.

Keys

A key is a record control field such as customer number or stock number. Records in an indexed file are in sequence by key to permit sequential processing and to aid in locating records randomly, and blocks are formatted with keys. That is, ISAM writes each block immediately preceded by the highest key within the block, namely, the key of the last or only record in the block. The key is usually also embedded within each data record, as normal.

Unblocked Records

This is the layout of keys for unblocked records:

key 201		record 201		key 205		record 205		key 206		record 206

The records could represent, for example, customer numbers, and the keys could be for customer numbers 201, 205, and 206. In this example, the key is 3 characters long and the data record is the conventional size. Under unblocked format, a key precedes each block containing one record.

Blocked Records

This is the layout of keys for blocked records based on the preceding unblocked example:

key 206		record 201	record 205	record 206

Under blocked format, the key for the last record in the block, 206, precedes the block.

ISAM automatically handles this use of keys, and when you perform a read operation, the system delivers the block, not the separate key, to main storage.

Indexes

To facilitate locating records randomly, ISAM maintains three levels of indexes on disk: track index, cylinder index, and an optional master index.

Track index. When ISAM creates a file, it stores a track index in track 0 of each cylinder that the file uses. The track index contains the highest key number for each track on the cylinder. For example, if track 4 on cylinder 12 contains records with keys 201, 205, 206, and 208, the track index contains an entry for key 208 and a reference to cylinder 12, track 4. If a disk device has ten tracks per cylinder, there are ten key entries for each track index, in ascending sequence.

Cylinder index. When ISAM creates a file, it stores a cylinder index on a separate cylinder containing the highest key for each cylinder. For example, if the file is stored on six cylinders, the cylinder index contains six entries.

Master index. An optional master index facilitates locating an appropriate cylinder index. This index is recommended if the entries in the cylinder index exceed four cylinders—a very large file.

PROCESSING AN INDEXED FILE

Consider a small indexed file containing 14 records on cylinder 5, with tracks 1 and 2 containing five records and track 3 containing four. This area is known as the *prime data area.* Track 1, for example, contains records with keys 205, 206, 208, 210, and 213. Assume that records are suitably blocked.

TRACK	DATA RECORDS ON CYLINDER 5
1	205 206 208 210 213
2	214 219 220 222 225
3	226 227 230 236 unused

Track 0 of cylinder 5 contains the track index, with an entry indicating the high key for each track. The track index entries specify that the highest keys on cylinder 5, tracks 1, 2, and 3 are 213, 225, and 236, respectively:

```
track index   key cylinder track  key cylinder track  key cylinder track
  /            \      \   /        \      \   /        \      \   |
track 0       | 213     0501 |    | 225     0502 |    | 236     0503 |
```

The cylinder index contains an entry for each cylinder that contains data, indicating the high key for each cylinder. In this case, the only index entry is key 236 on cylinder 5 (the track number is not important in this index):

$$\text{Cylinder index} \qquad |\ \overbrace{236}^{\text{key}}\ \overbrace{0500}^{\text{cylinder}}\ |$$

As an example of processing, a program has to locate randomly a record with key 227. The read statement directs the system to perform the following steps:

1. Check the cylinder index (assuming no master index), comparing key 227 against its first entry, 236. Since 227 is lower, the required record should be on cylinder 5.
2. Access the track index in cylinder 5, track 0, comparing key 227 successively against each entry: 213 (high), 225 (high), and 236 (low). According to the entry for 236, the required record should be on cylinder 5, track 3.
3. Check the keys on track 3; find key 227 and deliver the record to the program's input area. If the key and the record do not exist, ISAM signals an error.

As you can see, locating a record randomly involves a number of additional processing steps, although little extra programming effort is required. Even more processing steps are involved if a new record has to be added. If ISAM has to insert the record within the file, it may have to "bump" a record into an overflow area.

Overflow Areas

When a program first creates a file, ISAM stores the records sequentially in a prime data area. If you subsequently add a new record, ISAM stores it in an overflow area and maintains links to point to it.

There are two types of overflow areas: cylinder and independent:

1. For a *cylinder overflow area*, each cylinder has its own overflow track area. ISAM reserves tracks on the same cylinder as the prime data for all of its overflow records stored on a specific cylinder. The advantage of cylinder overflow is that less disk seek time is required to locate records on a different cylinder. The disadvantage is an uneven distribution of overflow records: Some of the overflow cylinders may contain many records, whereas other overflow cylinders may contain few or none.
2. For an *independent overflow area*, ISAM reserves a number of separate cylinders for all overflow records in the file. The advantage is that the distribution of overflow records is unimportant. The disadvantage is in the additional access time to locate records in the overflow area.

A system may adopt both types: the cylinder overflow area for initial overflows and the independent overflow area in case cylinder overflow areas overflow.

In our most recent example, adding a record with key 209 causes ISAM to bump record 213 from track 1 into an overflow area, move 210 in its place, and insert 209 in the place vacated by 210. The following assumes a cylinder overflow area in track 9:

TRACK	DATA RECORDS ON CYLINDER 5	
1	205 206 208 209 210	prime data area
2	214 219 220 222 225	
3	226 227 230 236 unused	
...		
9	213	overflow area

The track index now becomes 210, with a pointer (not shown) to key 213 in the overflow area:

```
                    key cylinder track   key cylinder track   key cylinder track
track index         | 210     0501 |     | 225     0502 |     | 236     0503 |
```

Reorganizing an Indexed File

Because a large number of records in overflow areas cause inefficient processing, an installation can use a program periodically to rewrite or reorganize the file. The program simply reads the records sequentially and writes them into another disk area. ISAM automatically follows its indexes for the input file and delivers the records sequentially from the prime and overflow areas. It stores all the output records sequentially in the new prime data area and automatically creates new indexes. At this time, the program may drop records coded for deletion.

PROCESSING DOS INDEXED SEQUENTIAL FILES

Since ISAM automatically handles indexes and overflow areas, little added programming effort is involved in the use of indexed files. There are four approaches to processing:

1. *Load or Extend.* The initial creation of an ISAM file is known as loading. Once a file is loaded, you may extend it by storing higher-key records at the end of the file.
2. *Adding Records.* New records have keys that do not currently exist on the file. You have to insert or add these records within the file.

3. *Random Retrieval.* To update an ISAM file with data (such as sales and payments on customer records), you use the key to locate the master record randomly and rewrite the updated record.
4. *Sequential Processing.* If you have many records to update and the new transactions are in sequence, you can sequentially read, change, and rewrite the ISAM master.

Load or Extend a DOS ISAM File

Loading a file creates it for the first time, and extending involves storing records at the end. Input records must be in ascending sequence by a predetermined key, and all keys must be unique. For load and extend, you code the usual OPEN and CLOSE to activate and deactivate the file. Figure 20-1 uses sequential input records to load an ISAM file named DISKIS. The new macros for this purpose are SETFL, WRITE, ENDFL, and DTFIS.

Name	Operation	Operand
[label]	SETFL	filename
[label]	WRITE	filename,NEWKEY
[label]	ENDFL	filename

Let's examine the imperative macros and the DTFIS file definition macro.

SETFL (set file load mode). SETFL initializes an ISAM file by preformatting the last track of each track index. The operand references the DTFIS name of the ISAM file to be loaded. In Fig. 20-1, SETFL immediately follows the OPEN macro.

WRITE. The WRITE macro loads a record into the ISAM file. Operand 1 is your DTFIS filename, and operand 2 is the word NEWKEY. You store the key and data area in a workarea (named ISAMOUT in Fig. 20-1). DTFIS knows this area through the entry WORKL=ISAMOUT. For the WRITE statement, ISAM checks that the new key is in ascending sequence. ISAM then transfers the key and data area to an I/O area (named IOARISAM in the figure and known to DTFIS by IOAREAL=IOARISAM). Here ISAM constructs the count area:

WORKL=ISAMOUT:		key	data
IOAREAL=IOARISAM:	count	key	data

ENDFL (end file load mode). After all records are written and before the CLOSE, ENDFL writes the last data block (if any), an end-of-file record, and any required index entries.

```
  1          PRINT ON,NODATA,NOGEN
  2 PROG20A  START
  3          BALR  3,0
  4          USING *,3
  5          OPEN  DISKSD,DISKIS
 14          SETFL DISKIS                    SET ISAM LIMITS
 20          TM    DISKISC,B'10011000'       ANY SETFL ERRORS?
 21          BO    R10ERR                       YES - ERROR ROUTINE
 22          GET   DISKSD,SDISKIN            GET 1ST SEQ'L RECORD

 29 *              M A I N   P R O C E S S I N G
 30 A10LOOP  MVC   ISKEYNO,ACCTIN            SET UP KEY NUMBER
 31          MVC   ISRECORD,SDISKIN          SET UP ISAM DISK RECORD
 32          WRITE DISKIS,NEWKEY             WRITE ISAM RECORD
 37          TM    DISKISC,B'11111110'       ANY WRITE ERRORS?
 38          BO    R10ERR                       YES - ERROR ROUTINE
 39          GET   DISKSD,SDISKIN            GET NEXT SEQ'L RECORD
 45          B     A10LOOP                      NO  - CONTINUE

 47 *              E N D - O F - F I L E
 48 A90END   ENDFL DISKIS                    END ISAM FILE LIMITS
 60          TM    DISKISC,B'11000001'       ANY ENDFL ERRORS?
 61          BO    R10ERR                       YES - ERROR ROUTINE
 62          CLOSE DISKSD,DISKIS             NORMAL TERMINATION
 71          EOJ

 75 *              D I S K   E R R O R   R O U T I N E S
 76 R10ERR   EQU   *                         DISK ERROR
 77 *         .                              RECOVERY ROUTINES
 78 *         .
 79          CLOSE DISKSD,DISKIS             ABNORMAL TERMINATION
 88          EOJ

 92 *              D E C L A R A T I V E S
 93 DISKSD   DTFSD BLKSIZE=360,              SEQUENTIAL DISK INPUT    +
                   DEVADDR=SYS015,                                    +
                   EOFADDR=A90END,                                    +
                   DEVICE=3340,                                       +
                   IOAREA1=IOARSD1,                                   +
                   RECFORM=FIXBLK,                                    +
                   RECSIZE=90,                                        +
                   TYPEFLE=INPUT,                                     +
                   WORKA=YES
154 IOARSD1  DS    CL360                     SEQ'L DISK BUFFER-1

156 SDISKIN  DS    0CL90                     SEQ'L DISK INPUT AREA
157 ACCTIN   DS    CL06                      *    KEY
158          DS    CL84                      *    REST OF RECORD

160 DISKIS   DTFIS CYLOFL=1,                 INDEXED SEQ'L LOAD       +
                   DEVICE=3340,                                       +
                   DSKXTNT=2,                                         +
                   IOAREAL=IOARISAM,                                  +
                   IOROUT=LOAD,                                       +
                   KEYLEN=6,                                          +
                   KEYLOC=1,                                          +
                   NRECDS=3,                                          +
                   RECFORM=FIXBLK,                                    +
                   RECSIZE=90,                                        +
                   VERIFY=YES,                                        +
                   WORKL=ISAMOUT
209 IOARISAM DS    CL284                     ISAM BUFFER AREA
```

Figure 20-1 Program: loading a DOS ISAM file.

```
ISAMOUT  DS     0CL96                    ISAM WORKAREA
ISKEYNO  DS     CL06                     *    KEY LOCATION
ISRECORD DS     CL90                     *    DATA AREA

         LTORG
                =C'$$BOPEN '
                =C'$$BSETFL'
                =C'$$BENDFL'
                =C'$$BCLOSE'
                =A(DISKIS)
                =A(DISKSD)
                =A(SDISKIN)
         END    PROG20A
```

Figure 20-1 (continued)

The DTFIS Macro

The maximum length for an ISAM filename is 7. In Fig. 20-1, the DTFIS entries for the file being loaded are as follows:

CYLOFL=1 gives the number of tracks on each cylinder to be reserved for each cylinder overflow area (if any).

DEVICE=3340 is the disk device containing the prime data area or overflow area.

DSKXTNT=2 provides the number of extents that the file uses: one for each data extent and one for each index area and independent overflow area extent. The program in Fig. 20-1 has one extent for the prime data area and one for the cylinder index.

IOAREAL=IOARISAM provides the name of the ISAM I/O load area. The symbolic name, IOARISAM, references the DS buffer area. For loading blocked records, you calculate the field length as

Count area (8) + key length (6) + block length (90 × 3) = 284

IOROUT=LOAD tells the assembler that the program is to load an ISAM file.

KEYLEN=6 gives the length of each record's key.

KEYLOC=1 tells ISAM the starting location of the key in the record, where 1 is the first position.

NRECDS=3 provides the number of records per block.

RECFORM=FIXBLK indicates fixed, blocked record format.

RECSIZE=90 gives the length of each record.

VERIFY=YES tells the system to check the parity of each record as it is written.

WORKL=ISAMOUT gives the name of your load workarea, which is a DS defined elsewhere in the program. For blocked records, you calculate the field length as

$$\text{Key length } (6) + \text{data area } (90 \times 3) = 284$$

For unblocked records, you would calculate the field length as

Count area (8) + key length + "sequence link field" (10) + record length

Status Condition

On execution, ISAM macros may generate error conditions, which you may test. After each I/O operation, ISAM places its status in a one-byte field named filenameC. For example, if your DTFIS name is DISKIS, ISAM calls the status byte DISKISC. Following is a list of the 8 bits in filenameC that the system may set when loading an ISAM file:

BIT	LOAD STATUS ERROR CONDITION
0	Any uncorrectable disk error except wrong length record.
1	Wrong length record detected on output.
2	The prime data area is full.
3	SETFL has detected a full cylinder index.
4	SETFL has detected a full master index.
5	Duplicate record—the current key is the same as the one previously loaded.
6	Sequence error—the current key is lower than the one previously loaded.
7	The prime data area is full, and ENDFL has no place to store the end-of-file record.

The program in Fig. 20-1 uses TM operations to test DISKIS after execution of the macros SETFL, WRITE, and ENDFL. After SETFL, for example, TM tests whether bits 0, 3, and 4 are on. If any of the conditions exist, the program executes an error routine (not coded) where the program may isolate the error and issue an error message.

The job control commands also vary. First, the DLBL job entry for "codes" contains ISC, meaning indexed sequential create, and second, there is an EXTENT command for both the cylinder index and the data area.

Random Retrieval of an ISAM File

The main purpose of organizing a file as indexed sequential is to facilitate the random accessing of records. For this, there are a number of special coding requirements. The program in Fig. 20-2 randomly retrieves records in the file created in Fig. 20-1. The program reads a file of modification records in random sequence, with changes to the ISAM master file. For each modification record, the program uses the account number (key) to locate the correct ISAM record, corrects it, and then updates the record on the ISAM file.

ISAM macros for random retrieval. The new macros for random retrieval are

Name	Operation	Operand
[label]	READ	filename,KEY
[label]	WAITF	filename
[label]	WRITE	filename,KEY

READ causes ISAM to access a required record from the file. Operand 1 contains the DTFIS filename, and operand 2 contains the word KEY. You store the key in the field referenced by the DTFIS entry KEYARG. In Fig. 20-2, KEYARG=KEYNO. For each modification record processed, the program transfers the account key number to KEYNO.

WAITF allows completion of a READ or WRITE operation before another is attempted. Since a random retrieval reads and rewrites the same record, you must ensure that the operation is finished. Code WAITF anywhere following a READ or WRITE and preceding the next READ or WRITE.

```
         PRINT ON,NODATA,NOGEN
PROG20B  START
         BALR  3,0
         USING *,3
         OPEN  FILEIN,DISKIS
         GET   FILEIN,RECDIN              READ 1ST INPUT RECORD

*              M A I N  P R O C E S S I N G
A10LOOP  MVC   ISKEYNO,ACCTIN             SET UP KEY NUMBER
         READ  DISKIS,KEY                 READ ISAM RANDOMLY
         TM    DISKISC,B'11010101'        ANY READ ERROR?
         BO    R10ERR                          YES - ERROR ROUTINE
         WAITF DISKIS                     COMPLETE READ OPERATION
         MVC   ACCTDKO,ACCTIN             MOVE FIELDS
         MVC   NAMEDKO,NAMEIN             *  TO DISK
         MVC   ADDRDKO,ADDRIN             *  WORKAREA
         PACK  BALNDKO,BALNIN             *
         MVC   DATEDKO,DATEIN             *
         WRITE DISKIS,KEY                 WRITE NEW ISAM RECORD
         TM    DISKISC,B'11000000'        ANY WRITE ERROR?
         BO    R10ERR                     *  YES - ERROR ROUTINE
         GET   FILEIN,RECDIN              READ NEXT INPUT RECORD
         B     A10LOOP                    *  NO  - CONTINUE
```

Figure 20-2 Program: random retrieval of a DOS ISAM file.

```
*                 E N D - O F - F I L E
A90END   CLOSE  FILEIN,DISKIS            TERMINATION
         EOJ

*                 D I S K   E R R O R   R O U T I N E S
R10ERR   EQU    *                        DISK ERROR
*        .                               RECOVERY ROUTINES
         B      A90END

*                 D E C L A R A T I V E S
FILEIN   DTFCD  DEVADDR=SYSIPT,          INPUT FILE           +
                IOAREA1=IOARIN1,                              +
                BLKSIZE=80,                                   +
                DEVICE=2540,                                  +
                EOFADDR=A90END,                               +
                TYPEFLE=INPUT,                                +
                WORKA=YES
IOARIN1  DS     CL80                     INPUT BUFFER 1

RECDIN   DS     0CL80                    INPUT AREA:
CODEIN   DS     CL02                     *  RECORD CODE '01'
ACCTIN   DS     CL06                     *  ACCOUNT NO.
NAMEIN   DS     CL20                     *  NAME
ADDRIN   DS     CL40                     *  ADDRESS
BALNIN   DS     ZL06'0000.00'            *  BALANCE
DATEIN   DS     CL06'DDMMYY'             *  DATE

DISKIS   DTFIS  CYLOFL=1,                ISAM RANDOM RETRIEVAL +
                DEVICE=3340,                                   +
                DSKXTNT=2,                                     +
                IOAREAR=IOARISAM,                              +
                IOROUT=RETRVE,                                 +
                KEYARG=ISKEYNO,                                +
                KEYLEN=6,                                      +
                KEYLOC=1,                                      +
                NRECDS=3,                                      +
                RECFORM=FIXBLK,                                +
                RECSIZE=90,                                    +
                TYPEFLE=RANDOM,                                +
                VERIFY=YES,                                    +
                WORKR=ISAMOUT
IOARISAM DS     CL270                    ISAM BUFFER AREA

ISAMOUT  DS     0CL90                    ISAM WORKAREA:
ISKEYNO  DS     CL06                     *  KEY AREA
ACCTDKO  DS     CL06                     *  ACCOUNT NO.
NAMEDKO  DS     CL20                     *  NAME
ADDRDKO  DS     CL40                     *  ADDRESS
BALNDKO  DS     PL04                     *  BALANCE
DATEDKO  DS     CL06                     *  DATE
         DC     CL14' '                  *  RESERVED

         LTORG
                =C'$$BOPEN '
                =C'$$BCLOSE'
                =A(FILEIN)
                =A(RECDIN)
                =A(DISKIS)
         END    PROG20B
```

Figure 20-2 (continued)

WRITE rewrites an ISAM record. Operand 1 is the DTFIS filename, and operand 2 is the word KEY, which refers to your entry in KEYARG.

The DTFIS Macro

In Fig. 20-2, the DTFIS entries for the random retrieval include these:

IOAREAR=IOARISAM provides the name of the ISAM I/O retrieval area. The symbolic name, IOARISAM, references the DS retrieval area for unblocked records. For blocked records, the buffer size is

Record length (including keys) × blocking factor

For unblocked records, the buffer size is:

Key length + "sequence link field" (10) + record length

TYPEFLE=RANDOM means that the system is to retrieve records randomly by key. Other entries are SEQNTL for sequential and RANSEQ for both random and sequential.

WORKR=ISAMOUT gives the name of your retrieval workarea.

Status Condition

The status byte for add and retrieve is different from load. Following is a list of the 8 bits in filenameC that the system may set:

BIT	ADD AND RETRIEVE STATUS CONDITION
0	Any uncorrectable disk error except wrong length record.
1	Wrong length record detected on an I/O operation.
2	End-of-file during sequential retrieval (not an error).
3	The requested record is not in the file.
4	The ID given to SETFL for SEQNTL is outside the prime data limits.
5	Duplicate record—an attempt to add a record that already exists in the file.
6	The cylinder overflow area is full.
7	A retrieval operation is trying to process an overflow record.

The program in Fig. 20-2 uses TM operations to test DISKIS after execution of the macros READ and WRITE. Once again, the program would isolate the error and issue an error message.

Sequential Reading of an ISAM File

Sequential reading of an ISAM file involves the use of the SETL, GET, and ESETL macros. SETL (Set Low) establishes the starting point of the first record to be processed. Its options include these:

- Set the starting point at the first record in the file:

```
SETL filename,BOF
```

- Set the starting point at the record with the key in the field defined by the DTFIS KEYARG entry:

```
SETL filename,KEY
```

- Set the starting point at the first record within a specified group. For example, the KEYARG field could contain "B480000" to indicate all records with key beginning with B48:

```
SETL filename,GKEY
```

The ESETL macro terminates sequential mode and is coded as ESETL, filename.

DTFIS entries include these:

IOAREAS=buffername for the name of the buffer area. You calculate the buffer size just as you do for random retrieval.

IOROUT=RETRVE to indicate sequential retrieval.

TYPEFLE=SEQNTL or **RANDOM** for sequential or random retrieval.

KEYLOC=n to indicate the first byte of the key in a record, if processing begins with a specified key or group of keys and records are blocked.

To delete a record, you may reserve a byte in the record and store a code in it. A practice is to use the first byte to match OS requirements. Subsequently, your program may test for the code when retrieving records and when reorganizing the file.

PROCESSING OS INDEXED SEQUENTIAL FILES

Processing ISAM files under OS is similar to DOS processing, except that QISAM (queued indexed sequential access method) is used for sequential processing and BISAM (basic indexed sequential access method) is used for random processing.

The Delete Flag

Under OS, the practice is to reserve the first byte of each record with a delete flag, defined with a blank when you create the file. You also code OPTCD=L in the DCB macro or the DD command. When you want to delete a record, store X'FF' in this byte. QISAM subsequently will not be able to retrieve the record. QISAM automatically drops the record when the file is reorganized.

Let's examine some features of OS ISAM processing.

Load an ISAM File

The OS imperative macros concerned with loading an ISAM file are the conventional OPEN, PUT, and CLOSE. DCB entries are as follows:

DDNAME	Name of the data set.
DSORG	IS for indexed sequential.
MACRF	(PM) for move mode or (PL) for locate mode.
BLKSIZE	Length of each block.
CYLOFL	Number of overflow tracks per cylinder.
KEYLEN	Length of the key area.
LRECL	Length of each record.
NTM	Number of tracks for the master index, if any.
OPTCD	Options required, such as MYLU in any sequence: M establishes a master index (or omit M). Y controls use of cylinder overflow areas. I controls use of an independent area. L is a delete flag to cause bypassing records with X'FF' in the first byte. U (for fixed length only) establishes the track index in main storage.
RECFM	Record format for fixed/variable and unblocked/blocked: F, FB, V, and VB.
RKP	Relative location of the first byte of the key field, where 0 is the first location. (For variable-length records, the value is 4 or greater.)

Sequential Retrieval and Update

Under OS, sequential retrieval and update involve the OPEN, SETL, GET, PUTX, ESETL, and CLOSE macros. Once the data set has been created with standard labels, many DCB entries are no longer required. DDNAME and DSORG=IS are still used, and the following entries are available:

`MACRF=(entry)`	The entries are (GM) or (GL) for input (PM) or (PL) for output (GM,SK,PU) if read and rewrite in place, where S is use of SETL, K is key or key class, and PU is use of PUTX macro
`EODAD=eofaddress`	Used for input, if reading to end-of-file.
`SYNAD=address`	Requests optional error checking.

The SETL macro. SETL (Set Low address) establishes the first sequential record to be processed anywhere within the data set. The general format is the following:

Name	Operation	Operand
[label]	SETL	dcb-name,start-position,address

The start-position operand has the following options:

B	Begin with the first record in the data set. (Omit operand 3 for B or BD.)
K	Begin with the record with the key in the operand 3 address.
KC	Begin with the first record of the *key class* in operand 3. A key class is any group of keys beginning with a common value, such as all keys H48xxxx. If the first record is "deleted," begin with the first non-deleted record.
I	Begin with the record at the actual device address in operand 3.

BD, KD, KDH, KCD, and ID cause retrieval of only the data portion of a record.

Here are some examples of SETL to set the first record in a file named DISKIS, using a 6-character key:

- Begin with the first record in the data set:

```
         SETL     DISKIS,B
```

- Begin with the record with the key 012644:

```
         SETL     DISKIS,K,KEYADD1
```

```
PROG20C  START
         SAVE  (14,12)
         BALR  3,0
         USING *,3
         ST    13,SAVEAREA+4
         LA    13,SAVEAREA

         OPEN  (ISFILE)
         SETL  ISFILE,B                        START 1ST RECORD OF DATA SET
         TIME
         ST    1,TODAY
         SP    TODAY,=P'5000'                  CALC DATE 5 YEARS AGO
         GET   ISFILE                          GET 1ST RECORD

A10LOOP  CP    26(3,1),TODAY                   5 YEARS OR OLDER?
         BNL   A20                             *    NO  - BYPASS
         MVI   0(1),X'FF'                      *    YES - SET DELETE CODE
         PUTX  ISFILE                          *    RE-WRITE RECORD
A20      GET   ISFILE                          GET NEXT
         B     A10LOOP                         LOOP

A90EOF   ESETL ISFILE                          END-OF-FILE
         CLOSE (ISFILE)
         L     13,SAVEAREA+4
         RETURN (14,12)

SAVEAREA DS    18F
TODAY    DS    F                               TODAY'S DATE: 00YYDDD+
IOAREA   DS    CL100                           DISK IO AREA

ISFILE   DCB   DDNAME=INDEXDD,                                         +
               DSORG=IS,                                               +
               EODAD=A90EOF,                                           +
               MACRF=(GL,S,PU)
         LTORG
         END   PROG20C
```

Figure 20-3 Program: sequential retrieval of an OS ISAM file.

- Begin with the first record of the key class 012:

```
              SETL     DISKIS,KC,KEYADD2
              ...
KEYADD1       DC       C'012644'          6-character key
KEYADD2       DC       C'012',XL3'00'     3-character key class
```

The ESETL macro, used as ESETL dcb-name, terminates sequential retrieval. If there is more than one SETL, ESETL must precede each one.

The program in Fig. 20-3 reads an ISAM file sequentially and inserts a delete code in any record that is more than five years old. The TIME macro delivers the standard date from the communication region as packed 00yyddd+, and the date in the record (positions 26–28) is in the same format. The PUTX macro rewrites an obsolete record with a delete byte in the first position.

KEY POINTS

- The indexed system writes a key preceding each block of records. The key is that of the highest record in the block.
- The track index, cylinder index, and master index help the system locate records randomly.
- The track index is in track 0 of each cylinder of the file and contains the highest key number for each track of the cylinder.
- The cylinder index is on a separate cylinder and contains the key number of the highest record on the cylinder.
- The optional master index is recommended if the cylinder index exceeds four cylinders in size.
- The master index facilitates locating keys in the cylinder index, the cylinder index facilitates locating keys in the track index, and the track index facilitates locating the track containing the required record.
- ISAM creates a file sequentially in a prime data area. Subsequent additions of higher keys append to the end, and additions of lower keys cause records to bump into an overflow area.
- Cylinder overflow areas reserve tracks on a cylinder for all overflows in that cylinder. This method reduces disk access time.
- Independent overflow areas reserve separate cylinders for overflows from the entire file. This method helps if there is an uneven distribution of overflow records—that is, many overflow records in some cylinders and few or none in others.

PROBLEMS

20-1. What is the purpose of (a) the master index; (b) the cylinder index; (c) the track index?

20-2. An indexed file contains three records per block. For a block containing records with keys 542, 563, and 572, what is the key for the block?

20-3. An indexed file contains unblocked records on cylinder 8 beginning with track 1. Assuming four records per track, show the organization of the records on the tracks for keys 412, 413, 415, 417, 419, 420, 424, 425, 432, 433.

20-4. For the file in Problem 20-3, show the contents of (a) the track index; (b) the cylinder index.

20-5. For the file in Problem 20-3, show the records on the tracks if a program adds a record with key 422. Assume that track 20 handles overflow records.

20-6. What are the two overflow areas and their advantages and disadvantages? Under what circumstances would you recommend use of both types of overflow areas?

20-7. What is the normal procedure to remove records from an overflow area into proper sequence in the prime data area?

20-8. What is the common method for deleting records from an indexed file for (a) DOS; (b) OS?

20-9. What are the different ways to process an ISAM file? What is the difference between extending and adding records?

20-10. Write a program that creates an ISAM supplier file on disk from the following input records:

01-05	Supplier number
06-25	Supplier name
26-46	Street
47-67	City
68-74	Amount payable
75-80	Date of last purchase (yymmdd)

Store the amount payable in packed format.

21

OPERATING SYSTEMS

OBJECTIVE

To introduce basic operating systems for DOS and OS and job control requirements.

This chapter introduces material that is suitable for more advanced assembler programming. The first section examines general operating systems and the various support programs. Subsequent sections explain the functions of the program status word and the interrupt system. Finally, there is a discussion of input/output channels, physical IOCS, and the input/output system.

These topics provide an introduction to systems programming and the relationship between the computer hardware and the manufacturer's software. A knowledge of these features can be a useful asset when serious bugs occur and when a solution requires an intimate knowledge of the system.

In an installation, one or more systems programmers, who are familiar with the computer architecture and assembler language, provide support for the operating system. Among the software that IBM supplies to support the system are language translators such as assembler, COBOL, and PL/I and utility programs for cataloging and sorting files.

OPERATING SYSTEMS

Operating systems were developed to minimize the need for operator intervention during the processing of programs. An operating system is a collection of related programs that provide for the preparation and execution of a user's programs. The system is stored on disk, and part of it, the supervisor program, is loaded into the lower part of main storage.

You submit job control commands to tell the system what action to perform. For example, you may want to assemble and execute a source program. To this end, you insert job control commands before and after the source program and submit it as a job to the system. In simple terms, the operating system performs the following steps:

1. Preceding the source program is a job control command that tells the operating system to assemble a program. The system loads the assembler program from a disk library into storage and transfers control to it for execution.
2. The assembler reads and translates the source program into an object program and stores it on disk.
3. Another job control command tells the system to link-edit the object program. The system loads the linkage editor from a disk library into storage and transfers control to it for execution.
4. The linkage editor reads and translates the object program, adds any required input/output modules, and stores it on disk as an executable module.
5. Another job control command tells the system to execute the executable module. The system loads the module into storage and transfers control to it for execution.
6. The program executes until normal or abnormal termination, when it returns processing control to the system.
7. A job command tells the system that this is the end of the job, since a job may consist of any number of execution steps. The system then terminates that job and prepares for the next job to be executed.

Throughout the processing, the system continually intervenes to handle all input/output, interrupts for program checks, and protecting the supervisor and any other programs executing in storage.

IBM provides various operating systems, depending on users' requirements, and they differ in services offered and the amount of storage they require. These are some major operating systems:

DOS	Disk Operating System	Medium-sized systems
DOS/VSE	Disk Operating System	Medium-sized systems with virtual storage
OS/VS1	Operating System	Large system

OS/VS2	Operating System	Large system
OS/MVS	Operating System	Large system

Systems Generation

The manufacturer typically supplies the operating system on reels of magnetic tape, along with an extensive set of supporting manuals. A systems programmer has to tailor the supplied operating system according to the installation's requirements, such as the number and type of disk drives, the number and type of terminals to be supported, the amount of processing time available to users, and the levels of security that are to prevail. This procedure is known as *systems generation*, abbreviated as *sysgen*.

Operating System Organization

Figure 21-1 shows the general organization of Disk Operating System (DOS), on which this text is largely based. The three main parts are the control program, system service programs, and processing programs.

Control Program

The control program, which controls all other programs being processed, consists of initial program load (IPL), the supervisor, and job control. Under OS, the functions are task management, data management, and job management.

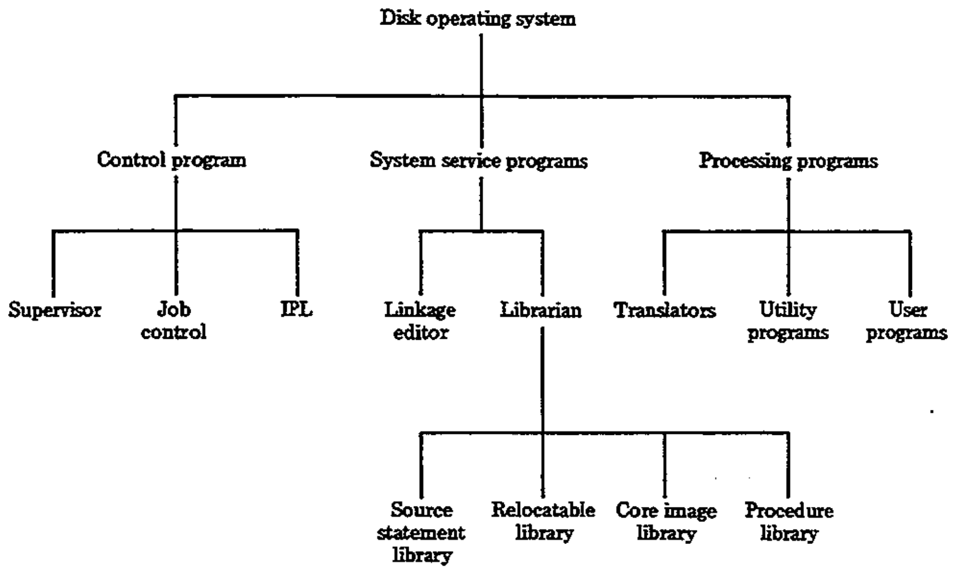


Figure 21-1 Disk operating system organization.

IPL is a program that the operator uses daily or whenever required to load the supervisor into storage. On some systems, this process is known as booting the system.

Job control handles the transition between jobs run on the system. Your job commands tell the system what action to perform next.

The supervisor, the nucleus of the operating system, resides in lower storage, beginning at location X'200'. The system loads user (problem) programs in storage following the supervisor area, resulting in at least two programs in storage: the supervisor program and one or more problem programs. Only one is executing at any time, but control passes between them.

The supervisor is concerned with handling interrupts for input/output devices, fetching required modules from the program library, and handling errors in program execution. An important part of the supervisor is the input/output control system (IOCS), known under OS as data management.

Figure 21-2 (not an exact representation) illustrates the general layout of the supervisor in main storage. Let's examine its contents.

1. *Communication Region.* This area contains the following data:

LOCATION	CONTENTS
00–07	The current date, as mm/dd/yy or dd/mm/yy
08–11	Reserved
12–22	User area, set to zero when a JOB command is read to provide communication within a job step or between job steps
23	User program status indicator (UPSI)
24–31	Job name, entered from job control
32–35	Address: highest byte of problem program area
36–39	Address: highest byte of current problem program phase
40–43	Address: highest byte of phase with highest ending address
44–45	Length of label area for problem program

2. *Channel Scheduler.* The channels provide a path between main storage

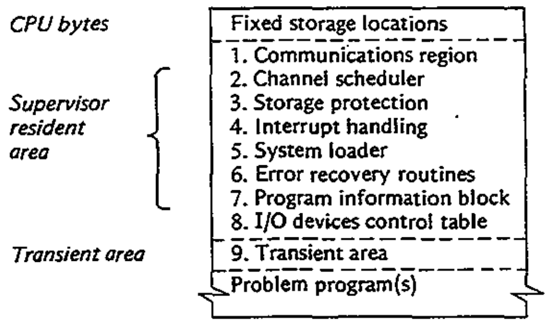


Figure 21-2 Supervisor areas.

and the input/output devices for all I/O interrupts and permit overlapping of program execution with I/O operations. If the requested channel, control unit, and device are available, the channel operation begins. If they are busy, the channel scheduler places its request in a queue and waits until the device is available. The channel notifies the scheduler when the I/O operation is complete or that an error has occurred.

3. *Storage Protection.* Storage protection prevents a problem program from erroneously moving data into the supervisor area and destroying it. Under a multiprogramming system, this feature also prevents a program in one partition from erasing a program in another partition.

4. *Interrupt Handling.* An interrupt is a signal that informs the system to interrupt the program that is currently executing and to transfer control to the appropriate supervisor routine. A later section on the program status word covers this topic in detail.

5. *System Loader.* The system loader is responsible for loading programs into main storage for execution.

6. *Error Recovery Routines.* A special routine handles error recovery for each I/O device or class of devices. When an error is sensed, the channel scheduler invokes the required routine, which attempts to correct the error.

7. *Program Information Block (PIB).* The PIB contains information tables that the supervisor needs to know about the current programs in storage.

8. *I/O Devices Control Table.* This area contains a table of I/O devices that relate physical unit addresses (X'nnn') with logical addresses (SYSxxx).

9. *Transient Area.* This area provides temporary storage for less used routines that the supervisor loads as required, such as OPEN, CLOSE, DUMP, end-of-job handling, some error recovery, and checkpoint routines.

System Service Programs

System service programs include the linkage editor and the librarian.

Linkage editor. The linkage editor has two main functions:

1. To include input/output modules. An installation catalogs I/O modules in the system library (covered next). When you code and assemble a program, it does not yet contain the complete instructions for handling input/output. On completion of assembly, the linkage editor includes all the required I/O modules from the library.
2. To link together separately assembled programs. You may code and assemble a number of subprograms separately and link-edit these subprograms into

one executable program. The linkage editor enables data in one subprogram to be recognized in another and facilitates transfer of control between subprograms at execution time.

Librarian. The operating system contains libraries on a disk known as SYSRES to catalog both IBM programs and the installation's own commonly used programs and subroutines. DOS/VS supports four libraries:

1. The source statement library (SSL) catalogs as a book any program, macro, or subroutine still in source code. You can use the assembler directive COPY to include cataloged code into your source program for subsequent assembling.
2. The relocatable library (RL) catalogs frequently used modules that are assembled but not yet ready for execution. The assembler directs the linkage editor to include I/O modules automatically, and you can use the INCLUDE command to direct the linkage editor to include your own cataloged modules with your own assembled programs.
3. The core image library (CIL) contains phases in executable machine code, ready for execution. The CIL contains, for example, the assembler, COBOL, PL/I, and other translator programs, various utility programs such as LINK and SORT, and your own production programs ready for execution. To request the supervisor to load a phase from the CIL into main storage for execution, use the job control command // EXEC phasename.
4. The procedure library (PL) contains cataloged job control to facilitate automatic processing of jobs.

The OS libraries vary by name according to the version of OS, but basically the OS libraries equivalent to the DOS source statement, relocatable, and core image are, respectively, source library, object library, and load library, and they serve the same functions.

Processing Programs

Processing programs are cataloged on disk in three groups:

1. Language translators that IBM supplies with the system include assembler, PL/I, COBOL, and RPG.
2. Utility programs that IBM supplies include such special-purpose programs as disk initialization, copy file-to-file, and sort/merge.
3. User-written programs that users in the installation write and that IBM does not support. All the programs in this text are user-written programs.

For example, the job command // EXEC ASSEMBLY causes the system to load the assembler from the CIL into an available area ("partition") in storage and begins assembling a program. The job command // OPTION LINK directs the assembler to write the assembled module on SYSLNK in the relocatable library.

Once the program is assembled and stored on SYSLNK, the job command // EXEC LNKEDT tells the linkage editor to load the module from SYSLNK into

FIXED STORAGE LOCATIONS

AREA, dec.	Hex addr	EC only	Function
0– 7	0		Initial program loading PSW, restart new PSW
8– 15	8		Initial program loading CCW1, restart old PSW
16– 23	10		Initial program loading CCW2
24– 31	18		External old PSW
32– 39	20		Supervisor Call old PSW
40– 47	28		Program old PSW
48– 55	30		Machine-check old PSW
56– 63	38		Input/output old PSW
64– 71	40		Channel status word (see diagram)
72– 75	48		Channel address word (0–3 key, 4–7 zeros, 8–31 CCW address)
80– 83	50		Interval timer
88– 95	58		External new PSW
96–103	60		Supervisor Call new PSW
104–111	68		Program new PSW
112–119	70		Machine-check new PSW
120–127	78		Input/output new PSW
132–133	84		CPU address assoc'd with external interruption, or unchanged
132–133	84	X	CPU address assoc'd with external interruption, or zeros
134–135	86	X	External interruption code
136–139	88	X	SVC interruption (0–12 zeros, 13–14 ILC, 15:0, 16–31 code)
140–143	8C	X	Program interrupt (0–12 zeros, 13–14 ILC, 15:0, 16–31 code)
144–147	90	X	Translation exception address (0–7 zeros, 8–31 address)
148–149	94		Monitor class (0–7 zeros, 8–15 class number)
150–151	96	X	PER interruption code (0–3 code, 4–15 zeros)
152–155	98	X	PER address (0–7 zeros, 8–31 address)
156–159	9C		Monitor code (0–7 zeros, 8–31 monitor code)
168–171	A8		Channel ID (0–3 type, 4–15 model, 16–31 max. IOEL length)
172–175	AC		I/O extended logout address (0–7 unused, 8–31 address)
176–179	B0		Limited channel logout (see diagram)
185–187	B9	X	I/O address (0–7 zeros, 8–23 address)
216–223	D8		CPU timer save area
224–231	E0		Clock comparator save area
232–239	E8		Machine-check interruption code
248–251	F8		Failing processor storage address (0–7 zeros, 8–31 address)
252–255	FC		Region code*
256–351	100		Fixed logout area*
352–383	160		Floating-point register save area
384–447	180		General register save area
448–511	1C0		Control register save area
512†	200		CPU extended logout area (size varies)

*May vary among models: see system library manuals for specific model

†Location may be changed by programming (bits 8–28 of CR 15 specify address)

Figure 21-3 Fixed storage locations.

storage, to complete addressing, and to include I/O modules from the RL. Assuming that there was no job command to catalog it, the linkage editor writes the linked phase in the CIL in a noncatalog area. If the next job command is // EXEC with no specified phasename, the supervisor loads the phase from the noncatalog area into storage for execution. The next program that the linkage editor links overlays the previous one in the CIL noncatalog area.

The job command // OPTION CATAL instead of // OPTION LINK tells the system both to link the program and to catalog the linked phase in the catalog area of the CIL. You normally catalog production programs in the CIL and for immediate execution use the job command // EXEC phasename.

FIXED STORAGE LOCATIONS

As mentioned earlier, the first X'200' bytes of storage are reserved for use by the CPU. Figure 21-3 lists the contents of these fixed storage locations.

MULTIPROGRAMMING

Multiprogramming is the concurrent execution of more than one program in storage. Technically, a computer executes only one instruction at a time, but because of the fast speed of the processor and the relative slowness of I/O devices, the computer's ability to service a number of programs at the same time makes it appear that processing is simultaneous. For this purpose, an operating system that supports multiprogramming divides storage into various *partitions* and is consequently far more complex than a single-job system.

The number and size of partitions vary according to the requirements of an installation. One job in each partition may be subject to execution at the same time, although only one program is actually executing. Each partition may handle jobs of a particular nature. For example, one partition handles relatively short jobs of high priority, whereas another partition handles large jobs of lower priority.

The job scheduler routes jobs to a particular partition according to its class. Thus a system may assign class A to certain jobs, to be run in the first partition.

In Fig. 21-4, the job queue is divided into four classes, and main storage is divided into three user partitions. Jobs in class A run in partition 1, jobs in classes B and C run in partition 2, and jobs in class P run in partition 3.

Depending on the system, storage may be divided into many partitions, and a job class may be designated to run in any one of the partitions. Also, a partition may be designated to run any number of classes.

When an operator uses the IPL procedure to boot the system, the supervisor is loaded from the CIL into low storage. The supervisor next loads job control from the CIL into the various partitions. The supervisor then scans the system readers and terminals for job control commands.

When a job completes processing, the job scheduler selects another job from

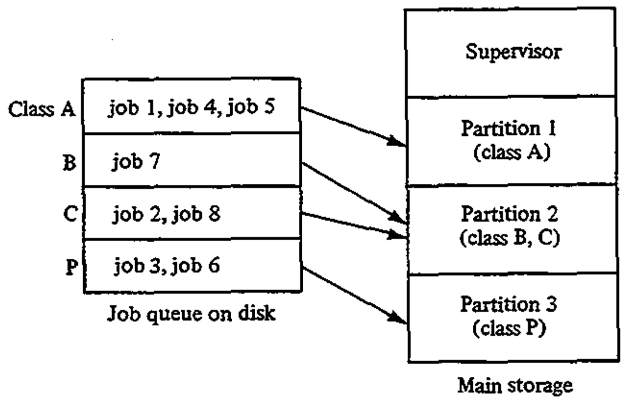


Figure 21-4 Job queue and partitions.

the queue to replace it. For example, if partition 1 is free, the job scheduler in Fig. 21-4 selects from the class A queue either the job with the highest priority or, if all jobs have the same priority, the first job in the queue.

The system has to provide a more or less equitable arrangement for processing jobs in each partition. Under time slicing, each partition is allotted in turn a time slice of so many milliseconds of execution. Control passes to the next partition when the time has expired, the job is waiting for an I/O operation to complete, or the job is finished.

VIRTUAL STORAGE

In a multiprogramming environment, a large program may not fit entirely in a partition. As a consequence, both DOS/VS and OS/VS support a virtual storage system that divides programs into segments of 64K bytes, which are in turn divided into pages of 2K or (usually) 4K bytes. On disk, the entire program is contained as pages in a page data set, and in storage VS arranges a page pool for as much of the program as it can store, as shown in Fig. 21-5. As a consequence, a program that is 100K in size could run in a 64K partition. If the executing program references an address for a part of the program that is not in storage, VS swaps an unneeded page into the page data set on disk and pages in the required page from

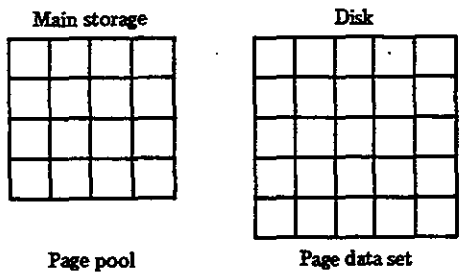


Figure 21-5 Page pool.

disk into the page pool in storage. (Actually, VS swaps onto disk only if the program has not changed the contents of the page.) The 16 control registers handle much of the paging operations.

Since a page from disk may map into any page in the pool, VS has to change addresses; this process is known as dynamic address translation (DAT).

When running a realtime application such as process control, a data communications manager, or an optical scan device, you may not want VS to page it out. It is possible to assign an area of nonpageable (real) storage for such jobs or use a "page fix" to lock certain pages into real storage.

PROGRAM STATUS WORD: PSW

The PSW is a doubleword of data stored in the control section of the CPU to control an executing program and to indicate its status. The two PSW modes are *basic control* (*BC*) *mode* and *extended control* (*EC*) *mode*. A 0 in PSW bit 12 indicates BC mode, and a 1 indicates EC mode. EC mode provides an extended control facility for virtual storage.

One of the main features of the PSW is to control the state of operation. Users of the system have no concern with certain operations such as storage management and allocation of I/O devices, and if they were allowed access to every instruction, they could inadvertently access other users' partitions or damage the system. To provide protection, certain instructions, such as Start I/O and Load PSW, are designated as privileged.

The PSW format is the same in only certain positions for each mode. Figure 21-6 illustrates the two modes, in which the bits are numbered 0 through 63 from left to right. Some of the more relevant fields are explained next.

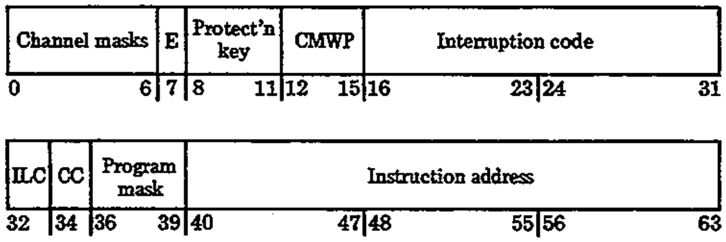
Bit 14: Wait state. When bit 14 is 0, the CPU is in running state executing instructions. When bit 14 is 1, the CPU is in wait state, which involves waiting for an action such as an I/O operation to be completed.

Bit 15: State. For both modes, 0 = supervisor state and 1 = problem state. When the computer is executing the supervisor program, the bit is 0 and all instructions are valid. When in the problem state, the bit is 1 and privileged instructions cannot be executed.

Bits 16–31: Program interrupt code (BC mode). When a program interrupt occurs, the computer sets these bits according to the type. The following list shows the interrupt codes in hex format:

0001	Operation exception
0002	Privileged operation exception
0003	Execute exception
0004	Protection exception
0005	Addressing exception

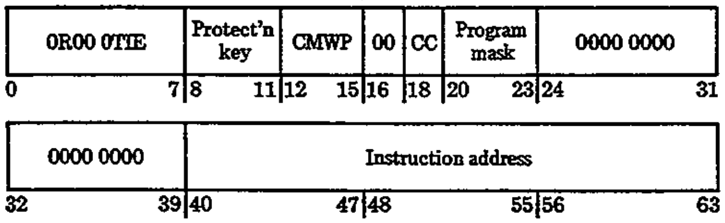
PROGRAM STATUS WORD (BC Mode)



0–5 Channel 0 to 5 masks
6 Mask for channel 6 and up
7 (E) External mask
12 (C–0) Basic control mode
13 (M) Machine-check mask
14 (W–1) Wait state
15 (P–1) Problem state

32–33 (ILC) Instruction length code
34–35 (CC) Condition code
36 Fixed-point overflow mask
37 Decimal overflow mask
38 Exponent underflow mask
39 Significance mask

PROGRAM STATUS WORD (EC Mode)



1 (R) Program event recording mask
5 (T–1) Translation mode
6 (I) Input/output mask
7 (E) External mask
12 (C–1) Extended control mode
13 (M) Machine-check mask
14 (W–1) Wait state

15 (P–1) Problem state
18–19 (CC) Condition code
20 Fixed-point overflow mask
21 Decimal overflow mask
22 Exponent underflow mask
23 Significance mask

Figure 21-6 PSW format for BC and EC modes.

0006	Specification exception
0007	Data exception
0008	Fixed-point overflow exception
0009	Fixed-point divide exception
000A	Decimal overflow exception
000B	Decimal divide exception
000C	Exponent overflow exception
000D	Exponent underflow exception
000E	Significance exception
000F	Floating-point divide exception

0010	Segment translation exception
0011	Page translation exception
0012	Translation specification exception
0013	Special operation exception
0040	Monitor event
0080	Program event (may be combined with another code)

Bits 34–35: Condition code. BC mode only; the condition code under EC mode is in bits 18–19. Comparisons and certain arithmetic instructions set this code.

Bits 40–63: Instruction address. This area contains the address of the next instruction to be executed. The CPU accesses the specified instruction from main storage, decodes it in the control section, and executes it in the arithmetic/logic section. The first 2 bits of a machine instruction indicate its length. The CPU adds this length to the instruction address in the PSW, which now indicates the address of the next instruction. For a branch instruction, the branch address may replace the PSW instruction address.

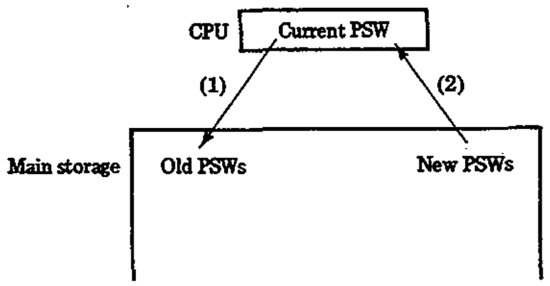
INTERRUPTS

An interrupt occurs when the supervisor has to suspend normal processing to perform a special task. The six main classes of interrupts are as follows:

1. *Program Check Interrupt.* This interrupt occurs when the computer cannot execute an operation, such as performing arithmetic on invalid packed data. This is the common type of interrupt when a program terminates abnormally. Appendix B lists the various types of program interrupts.
2. *Supervisor Call Interrupt.* A problem program may issue a request for input/output or to terminate processing. A transfer from the problem program to the supervisor requires a supervisor call (SVC) operation and causes an interrupt.
3. *External Interrupt.* An external device may need attention, such as the operator pressing the request key on the console or a request for communications.
4. *Machine Check Interrupt.* The machine-checking circuits may detect a hardware error, such as a byte not containing an odd number of on bits (odd parity).
5. *Input/Output Interrupt.* Completion of an I/O operation making a unit available or malfunction of an I/O device (such as a disk head crash) cause this interrupt.
6. *Restart Interrupt.* This interrupt permits an operator or another CPU to invoke execution of a program.

The supervisor region contains an interrupt handler for each type of interrupt. On an interrupt, the system alters the PSW as required and stores the PSW in a fixed storage location, where it is available to any program for testing.

The PSW discussed to this point is known as the current PSW. When an interrupt occurs, the computer stores the current PSW and loads a new PSW that controls the new program, usually the supervisor. The current PSW is in the control section of the CPU, whereas the old and new PSWs are stored in main storage, as the following indicates:



The interrupt replaces the current PSW in this way. (1) It stores the current PSW into main storage as the old PSW, and (2) it fetches a new PSW from main storage, to become the current PSW. The old PSW now contains in its instruction address the location following the instruction that caused the interrupt. The computer stores PSWs in 12 doubleword locations in fixed storage; 6 are for old PSWs and 6 are for new PSWs, depending on the class of interrupt:

	OLD PSW	NEW PSW
Restart	0008	0000
External	0024	0088
Supervisor call	0032	0096
Program old PSW	0040	0104
Machine check	0048	0112
Input/output	0056	0120

Let's trace the sequence of events following a supervisor interrupt. Assume that the supervisor has stored in the six new PSWs the address of each of its interrupt routines. (The old PSWs are not required yet.) Remember also that when an instruction executes, the computer updates the instruction address and the condition code in the current PSW as required.

1. A program requests input from disk. The GET or READ macro contains a supervisor call to link to the supervisor for input/output. This is a supervisor interrupt.

2. The instruction address in the current PSW contains the address in the program immediately following the SVC that caused the interrupt. The CPU stores this current PSW in the old PSW for supervisor interrupt, location 32. The new PSW for supervisor interrupt, location 96, contains supervisor state bit = 0 and the address of the supervisor interrupt routine. The CPU moves this new PSW to the current PSW and is now in the supervisor state.
3. The PSW instruction address contains the address of the supervisor I/O routine, which now executes. The channel scheduler requests the channel for disk input.
4. To return to the problem program, the supervisor loads the old PSW from location 32 back into the current PSW. The instruction links to the PSW instruction address, which is the address in the program following the original SVC that caused the interrupt. The system switches the PSW from supervisor state back to problem state.

In the event of a program check interrupt, the computer sets its cause on PSW bits 16–31, the program interrupt code. For example, if the problem program attempts arithmetic on invalid data, the computer senses a data exception and stores X'0007' in PSW bits 16–31. The computer then stores the current PSW in old PSW location 0040 and loads the new PSW from 0104 into the current PSW. This PSW contains the address of the supervisor's program check routine, which tests the old PSW to determine what type of program check caused the interrupt.

The supervisor displays the contents of the old PSW in hexadecimal and the cause of the program check (data exception), flushes the interrupted program, and begins processing the next job. Suppose that the invalid operation is an MP at location X'6A320'. Since MP is 6 bytes long, the instruction address in the PSW and the one printed will be X'6A326'. You can tell from the supervisor diagnostic message that the error is a data exception and that the invalid operation immediately precedes the instruction at X'6A326'.

CHANNELS

A channel is a component that functions as a separate computer operated by channel commands to control I/O devices. It directs data between devices and main storage and permits attaching a great variety of I/O devices. The more powerful the computer model, the more channels it may support. The two types of channels are multiplexer and selector.

1. *Multiplexer channels* are designed to support simultaneous operation of more than one device by interleaving blocks of data. The two types of multiplexer channels are byte-multiplexer and block-multiplexer. A byte-multiplexer channel typically handles low-speed devices, such as printers and terminals.

A block-multiplexer can support higher-speed devices, and its ability to interleave blocks of data facilitates simultaneous I/O operations.

2. *Selector channels*, no longer common, are designed to handle high-speed devices, such as disk and tape drives. The channel can transfer data from only one device at a time, a process known as burst mode.

Each channel has a 4-bit address coded as in the following example:

CHANNEL	ADDRESS	TYPE
0	0000	byte-multiplexer
1	0001	block-multiplexer
2	0010	block-multiplexer
3	0011	block-multiplexer
4	0100	block-multiplexer
5	0101	block-multiplexer
6	0110	block-multiplexer

A *control unit*, or controller, is required to interface with a channel. A channel is basically device-independent, whereas a control unit is device-dependent. Thus a block-multiplexer channel can operate many type of devices, but a disk drive control unit can operate only a disk drive. Figure 21-7 illustrates a typical configuration of channels, control units, and devices.

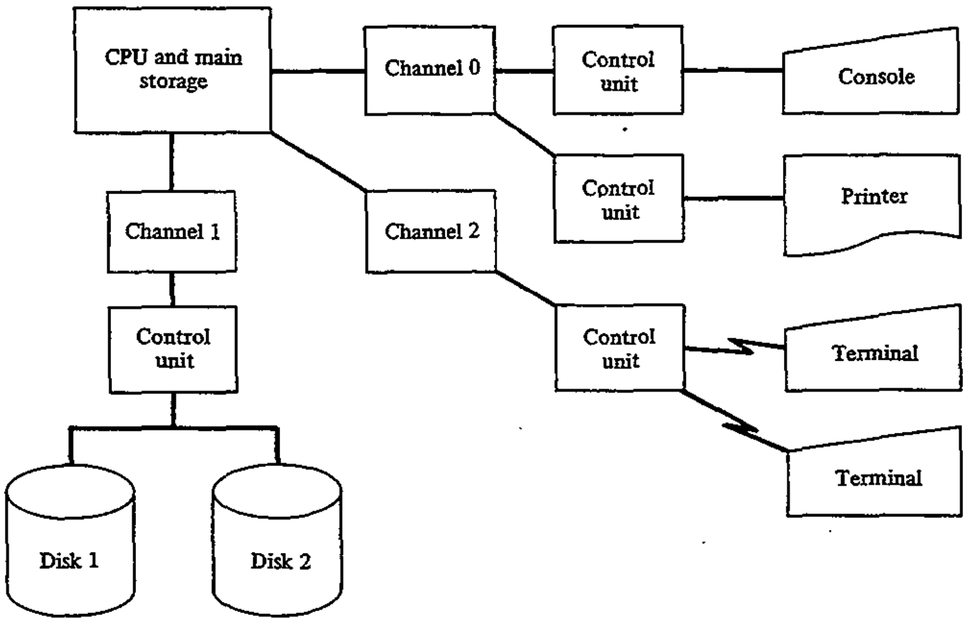


Figure 21-7 Channels, control units, and devices.

For example, a computer uses a multiplexer channel to connect it to a printer's control unit. The control unit has a 4-bit address. Further, each device has a 4-bit address and is known to the system by a physical address. The device address is therefore a 12-bit code that specifies:

DEVICE	CODE
Channel	0CCC
Control unit	UUUU
Device	DDDD

If the printer's device number is 1110 (X'E') and it is attached to channel 0, control unit 1, then to the system its physical address is 0000 0001 1110, or X'01E'. Further, if two disk devices are numbered 0000 and 0001 and they are both attached to channel 1, control unit 9, their physical addresses are X'190' and X'191', respectively. This physical address permits the attaching of 2^8, or 256 devices.

Symbolic Assignments

Although the supervisor references I/O devices by their physical numbers, your programs use symbolic names. You may assign a symbolic name to any device temporarily or (more or less) permanently, and a device may have more than one symbolic name assigned. The operating system uses certain names, known as system logical units, that include the following:

SYSIPT	The terminal, system reader, or disk device used as input for programs
SYSRDR	The terminal, system reader, or disk device used as input for job control for the system
SYSIN	The system name to assign both SYSIPT and SYSRDR to the same terminal, system reader, or disk device
SYSLST	The printer or disk used as the main output device for the system
SYSPCH	The device used as the main unit for output
SYSOUT	The system name to assign both SYSLST and SYSPCH to the same output device
SYSLNK	The disk area used as input for the linkage editor
SYSLOG	The console or printer used by the system to log operator messages and job control statements
SYSRES	The disk device where the operating system resides
SYSRLB	The disk device for the relocatable library
SYSSLB	The disk device for the system library

In addition, you may reference programmer logical units, SYS000–SYSnnn.

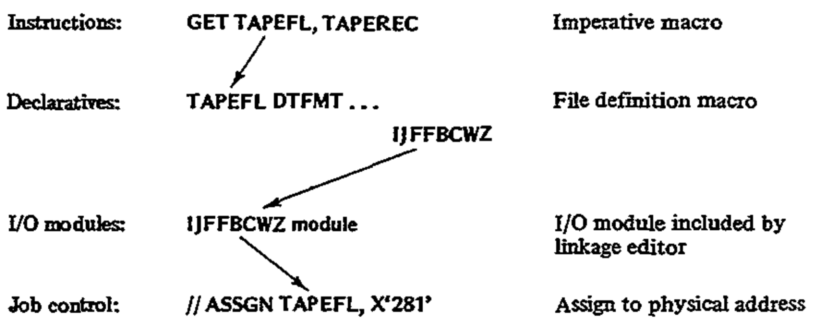
For example, you may assign the logical address SYS025 to a disk drive with physical address X'170'. The supervisor stores the physical and logical addresses in an I/O devices control table in order to relate them. A simplified table could contain the following:

I/O DEVICE	PHYSICAL ADDRESS	LOGICAL UNITS
Reader	X'00C'	SYSIPT, SYSRDR
Printer	X'00E'	SYSLST.
Disk drive	X'170'	SYSLNK, SYSRES, SYS025
Tape drive	X'280'	SYS031, SYS035

A reference to SYSLST is to the printer, and a reference to SYSLNK, SYSRES, or SYS025, depending on its particular use, is to disk device X'170'. You may assign a logical address permanently or temporarily and may change logical addresses from job to job. For instance, you could use an ASSGN job control command to reassign SYS035 for a program from a disk device X'170' to another disk device X'172'.

I/O LOGIC MODULES

Consider a program that reads a tape file named TAPEFL. The program would require a DTFMT or DCB file definition macro to define the characteristics of the file and tape device to generate a link to an I/O logic module. The assembler determines which particular logic module, based on (1) the kind of DTF and (2) the specifications within the file definition, such as device number, an input or output file, the number of buffers, and whether processing is in a workarea (WORKA) or a buffer (IOREG). In the following example, the assembler has generated a logic module named IJFFBCWZ (the name would vary depending on specifications within the DTFMT).



When linking a program, the linkage editor searches for addresses in the external symbol dictionary that the assembler generates. For this example, the ESD would contain entries at least for the program name and IJFFBCWZ. The linker accesses the named module cataloged on disk (provided it was ever cataloged) and includes it at the end of the assembled object program. One role of a system programmer is to define and catalog these I/O modules.

On execution of the program, the GET macro links to the specified file definition macro, DTFMT. This macro contains the address of the I/O logic module at the end of the object program where the linker included it. The module, combined with information from the DTFMT, contains all the instructions necessary to notify the supervisor as to the actual type of I/O operation, device, block size, and so forth.

The only remaining information is to determine which tape device; the supervisor derives it from the job control entry, which in this example assigns X'281' as the physical address. The supervisor then (at last) delivers the physical request for input via a channel command.

For example, the printer module, PRMOD, consists of three letters (IJD) and five option letters (abcde), as IJDabcde. The options are based on the definitions in the DTFPR macro, as follows:

a	RECFORM: FIXUNB (F), VARUNB (V), UNDEF (U)
b	CTLCHR: ASA (A), YES (Y), CONTROL (C)
c	PRINTOV=YES and ERROPT=YES (B), PRINTOV=YES and ERROPT not specified (Z), plus 14 other options
d	IOAREA2: defined (I), not defined (Z)
e	WORKA: YES (W), YES and RDONLY=YES (V), neither specified (Z)

A common printer module for IBM control character, two buffers, and a workarea would be IJDFYZIW. For one buffer, the module is IJDFYZZW.

PHYSICAL IOCS

Physical IOCS (PIOCS), the basic level of IOCS, provides for channel scheduling, error recovery, and interrupt handling. When using PIOCS, you write a channel program (the channel command word) and synchronize the program with completion of the I/O operation. You must also provide for testing the command control block for certain errors, for checking wrong-length records, for switching between I/O areas where two are used, and, if records are blocked, for blocking and deblocking.

PIOCS macros include CCW, CCB, EXCP, and WAIT.

Channel Command Word (CCW)

The CCW macro causes the assembler to construct an 8-byte channel command word that defines the I/O command to be executed.

Name	Operation	Operand
[label]	CCW	command-code,data-address,flags,count-field

- command-code defines the operation to be performed, such as 1 = write, 2 = read, X'09' = print and space one line.
- data-address provides the storage address of the first byte where data is to be read or written.
- flag bits determine the next action when the channel completes an operation defined in a CCW. You can set flag bits to 1 to vary the channel's operation (explained in detail later).
- count-field provides an expression that defines the number of bytes in the data block that is to be processed.

Command Control Block (CCB)

You define a CCB macro for each I/O device that PIOCS macros reference. The CCB comprises the first 16 bytes of most generated DTF tables. The CCB communicates information to PIOCS to cause required I/O operations and receives status information after the operation.

Name	Operation	Operand
blockname	CCB	SYSnnn,command-list-name

- blockname is the symbolic name associated with the CCB, used as an old PSW for the EXCP and WAIT macros.
- SYSnnn is the symbolic name of the I/O device associated with the CCB.
- command-list-name is the symbolic name of the first CCW used with the CCB.

Execute Channel Program (EXCP)

The EXCP macro requests physical IOCS to start an I/O operation, and PIOCS relates the blockname to the CCB to determine the device. When the channel and the device become available, the channel program is started. Program control then returns to your program.

Name	Operation	Operand
[label]	EXCP	blockname *or* (1)

The operand gives the symbolic name of the CCB macro to be referenced.

The WAIT Macro

The WAIT macro synchronizes program execution with completion of an I/O operation, since the program normally requires its completion before it can continue execution. (When bit 0 of byte 2 of the CCB for the file is set to 1, the WAIT is completed and processing resumes.) For example, if you have issued an EXCP operation to read a data block, you now WAIT for delivery of the entire block before you can begin processing it.

Name	Operation	Operand
[label]	WAIT	blockname *or* (1)

The operand gives the symbolic name of the CCB macro to be referenced.

CCW Flag Bits

You may set and use the flag bits in the CCW as follows:

- Bit 32 (chain data flag), set by X'80', specifies *data chaining*. When the CCW has processed the number of bytes defined in its count field, the I/O operation does not terminate if this bit is set. The operation continues with the next CCW in storage. You may use data chaining to read or write data into or out of storage areas that are not necessarily adjacent.

  In the following three CCWs, the first two use X'80' in the flag bits, operand 3, to specify data chaining. An EXCP and CCB may then reference the first CCW, and as a result, the chain of three CCWs causes the contents of an 80-byte input record to be read into three separate areas in storage: 20 bytes in NAME, 30 bytes in ADDRESS, and 30 bytes in CITY.

```
DATCHAIN CCW  2,NAME,X'80',20     Read 20 bytes into NAME, chain.
         CCW  ,ADDRESS,X'80',30   Read 30 bytes into ADDRESS, chain.
         CCW  ,CITY,X'00',30      Read 30 bytes into CITY, terminate.
```

- Bit 33 (chain command flag), set by X'40', specifies *command chaining* to enable the channel to execute more than one CCW before terminating the I/O operation. Each CCW applies to a separate I/O record.

  The following set of CCWs could provide for reading three input blocks, each 100 bytes long:

```
COMCHAIN CCW  2,INAREA,X'40',100     Read record-1 into
                                     INAREA, chain.
         CCW  2,INAREA+100,X'40',100 Read record-2 into
                                     INAREA+100, chain.
         CCW  2,INAREA+200,X'00',100 Read record-3 into
                                     INAREA+200, terminate.
```

- Bit 34 (suppress length indication flag), set by X'20', is used to suppress an error indication that occurs when the number of bytes transmitted differs from the count in the CCW.
- Bit 35 (skip flag), set by X'10', is used to suppress transmission of input data. The device actually reads the data, but the channel does not transmit the record.
- Bit 36 (program controlled interrupt flag), set by X'08', causes an interrupt when this CCW's operation is complete. (This is used when one supervisor SIO instruction executes more than one CCW.)
- Bit 37 (indirect data address flag), as well as other features about physical IOCS, is covered in the IBM Principles of Operation manual and the appropriate supervisor manual for your system.

Sample Physical IOCS Program

The program in Fig. 21-8 illustrates many of the features of physical IOCS we have discussed. It performs the following operations:

- At initialization, prints three heading lines by means of command chaining (X'40').
- Reads input records one at a time containing salesman name and company.
- Prints each record.
- Terminates on reaching end-of-file.

Note that the program defines a CCB/CCW pair for each type of record, and the EXCP/WAIT operations reference the CCB name—INDEVIC for the reader, OUTDEV1 for heading lines, and OUTDEV2 for sales detail lines. Each CCB contains the name of the I/O device, SYSIPT or SYSLST, and the name of an associated CCW: INRECD, TITLES, and DETAIL, respectively.

```
  LOC  OBJECT CODE       STMT    SOURCE STATEMENT
                            1           PRINT NODATA,NOGEN
000000                      2 PIOCSPRG START 0                       INITIALIZE
000000 0530                 3           BALR  3,0                    *
                            4           USING *,3                    *

                            6           EXCP  OUTDEV1                PRINT TITLES
                           10           WAIT  OUTDEV1                *

                           17 A10OREAD EXCP  INDEVIC                 READ RECORD
                           21           WAIT  INDEVIC                *

00002A D501 3076 3332      28           CLC   RECORD(2),=C'/*'      END FILE?
000030 4780 305C           29           BE    A900END                  YES
000034 D213 32B8 3076      31           MVC   SURNOUT,SURNAME        LOAD
00003A D213 32CD 308A      32           MVC   GIVENOUT,GIVENAME     * PRINT
000040 D21D 32E2 309E      33           MVC   COMPOUT,COMPANY       * LINE

                           35           EXCP  OUTDEV2                PRINT
                           39           WAIT  OUTDEV2                *
00005A 47F0 3014           45           B     A100READ               RETURN

                           47 A900END  EOJ                           END OF JOB

                           51 *         ----------------------
                           52 *         D E C L A R A T I V E S
                           53 *         ----------------------
                           54 INDEVIC  CCB   SYSIPT,INRECRD         I/P DEVICE
000070 0200007820000050    65 INRECRD  CCW   X'02',RECORD,X'20',80

000078                     67 RECORD   DS    0CL80                  I/P RECORD
000078                     68 SURNAME  DS    CL20                   *
00008C                     69 GIVENAME DS    CL20                   *
0000A0                     70 COMPANY  DS    CL40                   *

                           72 OUTDEV1  CCB   SYSLST,TITLES          O/P DEVICE

0000D8 8B0000D860000001    84 TITLES   CCW   X'8B',*,X'60',1
0000E0 110000F840000085    85          CCW   X'11',PRIMARY,X'40',133
0000E8 1900017D40000085    86          CCW   X'19',SECONDRY,X'40',133
0000F0 1100020200000085    87          CCW   X'11',TERTIARY,X'00',133

0000F8                     89 PRIMARY  DS    0CL133                 TITLE #1
0000F8 4040404040404040    90          DC    CL37' '
00011D E340D640D7404040    91          DC    CL16'T O P   S A L E'
00012D E240D440C540D540    92          DC    CL80'S M E N   O F'

00017D                     94 SECONDRY DS    0CL133                 TITLE #2
00017D 4040404040404040    95          DC    CL34' '
00019F E340C840C5404040    96          DC    CL14'T H E   W E S'
0001AD E340C540D940D540    97          DC    CL14'T E R N   R E'
0001BB C740C940D640D540    98          DC    CL71'G I O N'

000202                    100 TERTIARY DS    0CL133                 TITLE #3
000202 4040404040404040   101          DC    CL26' '
00021C E2E4D9D5C1D4C540   102          DC    CL21'SURNAME'
000231 C7C9E5C5D540D5C1   103          DC    CL21'GIVEN NAME'
000246 C3D6D4D7C1D5E840   104          DC    CL65'COMPANY'
```

Figure 21-8 Program: physical IOCS.

```
                              106 OUTDEV2  CCB    SYSLST,OUTRECRD    O/P DEVICE
000297 00
000298 090002A020000085 118 OUTRECRD CCW    X'09',DETAIL,X'20',133

0002A0                    120 DETAIL   DS     0CL133             DETAIL
0002A0 4040404040404040 121            DC     CL26' '            * LINE
0002BA                    122 SURNOUT  DS     CL20
0002CE 40                 123          DC     CL01' '
0002CF                    124 GIVENOUT DS     CL20
0002E3 40                 125          DC     CL01' '
0002E4                    126 COMPOUT  DS     CL30
000302 4040404040404040 127            DC     CL35' '

000328                    129          LTORG  ,
000328 000000C8           130                 =A(OUTDEV1)
00032C 00000060           131                 =A(INDEVIC)
000330 00000287           132                 =A(OUTDEV2)
000334 615C               133                 =C'/*'
                          134          END    PIOCSPRG

     Output:-
                    T O P   S A L E S M E N   O F

                 T H E   W E S T E R N   R E G I O N

          SURNAME              GIVEN NAME           COMPANY

          RUTH                 GEORGE HERMAN        LASER CORP.
          JOHNSON              WALTER               AMX ELECTRONICS
          COLLINS              EDDIE                B M I
          COBB                 TYRUS RAYMOND        AUDIO SHACK
          SPEAKER              TRIS                 PACKLETT HEWARD
          SIMMONS              AL                   VIDEO DUMP
          SISLER               GEORGE               COMPUTER HEAP
          WAGNER               HANS                 DIGITAL CORP.
```

Figure 21-8 (continued)

KEY POINTS

- Systems generation (sysgen) involves tailoring the supplied operating system to the installation's requirements, such as the number and type of disk drives, the number and type of terminals to be supported, the amount of process time available to users, and the levels of security that are to prevail.
- The control program, which controls all other programs being processed, consists of initial program load (IPL), the supervisor, and job control. Under OS, the functions are task management, data management, and job management.
- Initial program load (IPL) is a program that the operator uses daily or whenever required to load the supervisor into storage. The system loader is responsible for loading programs into main storage for execution.
- The supervisor resides in lower storage, beginning at location X'200'. The

supervisor is concerned with handling interrupts for input/output devices, fetching required modules from the program library, and handling errors in program execution.

- Channels provide a path between main storage and the input/output devices and permit overlapping of program execution with I/O operations. The channel scheduler handles all I/O interrupts.
- Storage protection prevents a problem program from erroneously moving data into the supervisor area and destroying it.
- An interrupt is a signal that informs the system to interrupt the program that is currently executing and to transfer control to the appropriate supervisor routine.
- The source statement library (SSL) catalogs as a book any program, macro, or subroutine still in source code.
- The relocatable library (RL) catalogs frequently used modules that are assembled but not yet ready for execution.
- The core image library (CIL) contains phases in executable machine code, ready for execution.
- Multiprogramming is the concurrent execution of more than one program in storage. An operating system that supports multiprogramming divides storage into various partitions. One job in each partition may be subject to execution at the same time, although only one program is actually executing.
- The PSW is stored in the control section of the CPU to control an executing program and to indicate its status. The two PSW modes are basic control (BC) mode and extended control (EC) mode.
- Certain instructions such as Start I/O and Load PSW are privileged to provide protection against users' accessing the wrong partitions.
- An interrupt occurs when the supervisor has to suspend normal processing to perform a special task. The supervisor region contains an interrupt handler for each type of interrupt.
- A channel is a component that functions as a separate computer operated by channel commands to control I/O devices. It directs data between devices and main storage and permits the attachment of a variety of I/O devices. The two types are multiplexer and selector.
- The operating system uses certain names, known as system logical units, such as SYSIPT, SYSLST, and SYSLOG. Programmer logical units are referenced as SYS000–SYSnnn.
- Physical IOCS (PIOCS), the basic level of IOCS, provides for channel scheduling, error recovery, and interrupt handling. When using PIOCS, you write a channel program (the channel command word) and synchronize the program with completion of the I/O operation.
- The CCW macro causes the assembler to construct an 8-byte channel command word that defines the I/O command to be executed.

PROBLEMS

21-1. What is the purpose of an operating system?
21-2. Where is the supervisor located in storage?
21-3. What is a sysgen?
21-4. What is the purpose of the supervisor transient area?
21-5. Where is the channel scheduler and what is its function?
21-6. In which libraries are the following stored (a) phase; (b) module; (c) book?
21-7. What are the two main functions of the linkage editor?
21-8. Explain the role of partitions and the job scheduler.
21-9. What is dynamic address translation?
21-10. What do the first 512 bytes of main storage contain?
21-11. What are the two modes and the two states of the PSW?
21-12. Where in the PSW (the name and bit positions) is the next sequential instruction located?
21-13. What are the classes of interrupts and their causes?
21-14. What is the purpose of channels? What are the two types and their differences?
21-15. A printer, number 1101, is attached to control unit 0010 and a multiplexer channel. What is the printer's physical address in hex?
21-16. Distinguish between physical address and logical address.
21-17. What are system logical units and programmer logical units?
21-18. Revise a simple program and substitute physical IOCS for input/output.

APPENDIX

A

HEXADECIMAL-DECIMAL CONVERSION

This appendix provides the steps in converting between hexadecimal and decimal formats. The first section shows how to convert hex A7B8 to decimal 42,936 and the second section shows how to convert 42,936 back to hex A7B8.

CONVERTING HEXADECIMAL TO DECIMAL

To convert hex number A7B8 to a decimal number, start with the leftmost hex digit (A), continuously multiply each hex digit by 16, and accumulate the results. Since multiplication is in decimal, convert hex digits A through F to decimal 10 through 15.

First digit, A (10):	10
Multiply by 16:	× 16
	160
Add next digit, 7:	+ 7
	167
Multiply by 16:	× 16
	2672
Add next digit, B (11):	+ 11
	2683
Multiply by 16:	× 16
	42,928
Add next digit, 8:	+ 8
Decimal value	42,936

You can also use the conversion table in Fig. A-1. For hex number A7B8, think of the rightmost digit (8) as position 1, the next digit to the left (B) as position 2, the next digit (7) as position 3, and the leftmost digit (A) as position 4. Refer to the figure and locate the value for each hex digit:

For position 1 (8), column 1 equals	8
For position 2 (B), column 2 equals	176
For position 3 (7), column 3 equals	1,792
For position 4 (A), column 4 equals	40,960
Decimal value:	42,936

CONVERTING DECIMAL TO HEXADECIMAL

To convert decimal number 42,936 to hexadecimal, first divide the original number 42,936 by 16; the remainder becomes the rightmost hex digit, 6. Next divide the new quotient 2,683 by 16; the remainder, 11 = B, becomes the next hex digit to the left. Develop the hex number from the remainders of each step of the division. Continue in this manner until the quotient is zero.

DIVISION	QUOTIENT	REMAINDER	HEX	
42,936 ÷ 16	2,683	8	8	(rightmost)
2,683 ÷ 16	167	11	B	
167 ÷ 16	10	7	7	
10 ÷ 16	0	10	A	(leftmost)

Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

Figure A-1

You can also use Fig. A-1 to convert decimal to hexadecimal. For decimal number 42,936, locate the number that is equal or next smaller. Note the equivalent hex number and its position in the table. Subtract the decimal value of that hex digit from 42,936, and locate the difference in the table. The procedure works as follows:

	DECIMAL		HEX
Starting decimal value:	42,936		
Subtract next smaller number:	−40,960	=	A000
Difference:	1,976		
Subtract next smaller number:	− 1,792	=	700
Difference:	184		
Subtract next smaller number:	− 176	=	B0
Difference:	8	=	8
Final hex number:			A7B8

APPENDIX

B

PROGRAM INTERRUPTS

A program interrupt occurs when a program attempts an operation that requires special attention. These are the program interrupts, listed by hex code number.

1. Operation Exception

The CPU has attempted to execute an invalid machine operation, such as hexadecimal zeros. Possible causes: (a) missing branch instruction, and the program has entered a declarative area; (b) the instruction, such as a floating-point operation, is not installed on the computer; (c) during assembly, an invalid instruction has caused the assembler to generate hexadecimal zeros in place of the machine code. For a 6-byte instruction, such as MVC with an invalid operand, the assembler generates 6 bytes of hex zeros. At execute time, the computer tries to execute the zero operation code, causing an operation exception. (Since the computer attempts to execute 2 bytes at a time, the system may generate three consecutive operation exceptions.) See also the causes for an addressing exception.

2. Privileged-Operation Exception

An attempt has been made to execute a privileged instruction that only the supervisor is permitted to execute. Possible causes: See operation (1) and addressing

(5) exceptions. Since there are many causes, it may be necessary to take a hexadecimal dump of the program to determine the contents of I/O areas and other declaratives to discover at what point during execution the error occurred.

3. Execute Exception

An attempt has been made to use the EX instruction on another EX instruction.

4. Protection Exception

A storage protection device prevents programs from erroneously moving data into the supervisor area or other partitions. Such attempts (for example, by MVC and ZAP) cause the computer to signal the error. Possible causes: (a) the program has erroneously loaded data into one of its base registers; (b) improper explicit use of a base register.

5. Addressing Exception

The program is attempting to reference an address that is outside available storage. Possible causes: (a) a branch to an address in a register containing an invalid value; (b) an instruction, such as MVC, has erroneously moved a data field into program instructions; (c) improper use of a base register, for example, loaded with a wrong value; (d) a BR instruction has branched to an address in a register and the wrong register was coded or its contents were changed.

6. Specification Exception

The program has violated a rule of an instruction. (a) For any type of operation, an attempt has been made to execute or branch to an instruction that does not begin on an even storage address (possibly an incorrect base register). (b) For packed operations DP and MP, a multiplier or divisor exceeds 8 bytes, or the length of the operand 1 field is less than or equal to that of operand 2. (c) For binary operations D, DR, M, MR, SLDA, SLDL, SRDA, and SRDL, the instruction does not reference an even-numbered register. (d) A floating-point operation does not reference register 0, 2, 4, or 6, or an extended-precision instruction does not reference a proper pair of registers, 0 and 2 or 4 and 6. (e) CLCL or MVCL does not reference an even-numbered register.

7. Data Exception

An attempt has been made to perform arithmetic on an invalid packed field. (a) For AP, CP, CVB, DP, ED, EDMK, MP, SP, SRP, or ZAP, the digit or sign positions contain invalid data. Possible causes: An input field contains blanks or other nondigits that pack invalidly; failure to pack, or an improper pack; an AP has added to an accumulator that was not initialized with valid packed data; improper use of relative addressing; an MVC has erroneously destroyed a packed field; improper explicit use of a base register. (b) The multiplicand field for an MP is too short. (c) The operation fields for AP, CP, DP, MP, SP, or ZAP overlap improperly due to incorrect use of relative addressing.

8. Fixed-Point Overflow Exception

A binary operation (A, AH, AR, LCR, LPR, S, SH, SR, SLA, or SLDA) has caused the contents of a register to overflow, losing a leftmost significant digit. The maximum value that a register can contain is, in decimal notation, +2,147,483,647.

9. Fixed-Point Divide Exception

A binary divide (D or DR) or a CVB has generated a value that has exceeded the capacity of a register. A common cause for divide operations is dividing by a zero value. The maximum value that a register can contain is, in decimal notation, +2,147,483,647.

A. Decimal-Overflow Exception

The result of a decimal packed operation (AP, SP, SRP, or ZAP) is too large for the receiving field. Solution: Redefine the receiving field so that it can contain the largest possible value, or perform a right shift to reduce the size of the value.

B. Decimal-Divide Exception

The generated quotient/remainder for a DP operation is too large for the defined area. Possible causes: (a) failure to follow the rules of DP; (b) the divisor contains a zero value.

C. Exponent-Overflow Exception

A floating-point arithmetic operation has caused an exponent to overflow (exceed +63).

D. Exponent-Underflow Exception

A floating-point arithmetic operation has caused an exponent to underflow (less than −64).

E. Significance Exception

A floating-point add or subtract has caused a zero fraction. All significant digits are lost, and subsequent computations may be meaningless.

F. Floating-Point Divide Exception

A floating-point operation has attempted a division using a zero divisor.

In each case, the system issues an error message, giving the type of program interrupt and the address where the interrupt occurred. Sometimes an error causes a program to enter a declarative area or another invalid area outside the program. (The computer may even find a valid machine code there.) In debugging, determine how the program arrived at the invalid address. In many cases, a dump of the program's registers and storage area is essential in tracing the cause of the error.

Another common error, though not a program interrupt, is generated by the operating system: INVALID STATEMENT. The system is attempting to read an invalid job control command. A common cause is a program that has terminated before reading all its data in the job stream, and the system is trying to read its remaining data records as job commands. Possible causes are (a) missing branch instructions causing the program inadvertently to enter its end-of-file routine; (b) branching to the end-of-file routine on an error condition without flushing remaining records in the job stream.

APPENDIX

C

ASSEMBLER INSTRUCTION SET

MACHINE INSTRUCTIONS

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Add (c)	AR	1A	RR	R1,R2
Add (c)	A	5A	RX	R1,D2(X2,B2)
Add Decimal (c)	AP	FA	SS	D1(L1,B1),D2(L2,B2)
Add Halfword (c)	AH	4A	RX	R1,D2(X2,B2)
Add Logical (c)	ALR	1E	RR	R1,R2
Add Logical (c)	AL	5E	RX	R1,D2(X2,B2)
AND (c)	NR	14	RR	R1,R2
AND (c)	N	54	RX	R1,D2(X2,B2)
AND (c)	NI	94	SI	D1(B1),I2
AND (c)	NC	D4	SS	D1(L,B1),D2(B2)
Branch and Link	BALR	05	RR	R1,R2
Branch and Link	BAL	45	RX	R1,D2(X2,B2)
Branch on Condition	BCR	07	RR	M1,R2
Branch on Condition	BC	47	RX	M1,D2(X2,B2)
Branch on Count	BCTR	06	RR	R1,R2
Branch on Count	BCT	46	RX	R1,D2(X2,B2)
Branch on Index High	BXH	86	RS	R1,R3,D2(B2)
Branch on Index Low or Equal	BXLE	87	RS	R1,R3,D2(B2)
Clear I/O (c,p)	CLRIO	9D01	S	D2(B2)
Compare (c)	CR	19	RR	R1,R2
Compare (c)	C	59	RX	R1,D2(X2,B2)
Compare and Swap (c)	CS	BA	RS	R1,R3,D2(B2)
Compare Decimal (c)	CP	F9	SS	D1(L1,B1),D2(L2,B2)
Compare Double and Swap (c)	CDS	BB	RS	R1,R3,D2(B2)
Compare Halfword (c)	CH	49	RX	R1,D2(X2,B2)
Compare Logical (c)	CLR	15	RR	R1,R2
Compare Logical (c)	CL	55	RX	R1,D2(X2,B2)
Compare Logical (c)	CLC	D5	SS	D1(L,B1),D2(B2)
Compare Logical (c)	CLI	95	SI	D1(B1),I2
Compare Logical Characters under Mask (c)	CLM	BD	RS	R1,M3,D2(B2)
Compare Logical Long (c)	CLCL	0F	RR	R1,R2
Convert to Binary	CVB	4F	RX	R1,D2(X2,B2)
Convert to Decimal	CVD	4E	RX	R1,D2(X2,B2)
Diagnose (p)		83		*Model-dependent*
Divide	DR	1D	RR	R1,R2
Divide	D	5D	RX	R1,D2(X2,B2)
Divide Decimal	DP	FD	SS	D1(L1,B1),D2(L2,B2)
Edit (c)	ED	DE	SS	D1(L,B1),D2(B2)
Edit and Mark (c)	EDMK	DF	SS	D1(L,B1),D2(B2)
Exclusive OR (c)	XR	17	RR	R1,R2
Exclusive OR (c)	X	57	RX	R1,D2(X2,B2)

MACHINE INSTRUCTIONS (Contd)

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Exclusive OR (c)	XI	97	SI	D1(B1),I2
Exclusive OR (c)	XC	D7	SS	D1(L,B1),D2(B2)
Execute	EX	44	RX	R1,D2(X2,B2)
Halt I/O (c,p)	HIO	9E00	S	D2(B2)
Halt Device (c,p)	HDV	9E01	S	D2(B2)
Insert Character	IC	43	RX	R1,D2(X2,B2)
Insert Characters under Mask (c)	ICM	BF	RS	R1,M3,D2(B2)
Insert PSW Key (p)	IPK	B20B	S	
Insert *Storage* Key (p)	ISK	09	RR	R1,R2
Load	LR	18	RR	R1,R2
Load	L	58	RX	R1,D2(X2,B2)
Load Address	LA	41	RX	R1,D2(X2,B2)
Load and Test (c)	LTR	12	RR	R1,R2
Load Complement (c)	LCR	13	RR	R1,R2
Load Control (p)	LCTL	B7	RS	R1,R3,D2(B2)
Load Halfword	LH	48	RX	R1,D2(X2,B2)
Load Multiple	LM	98	RS	R1,R3,D2(B2)
Load Negative (c)	LNR	11	RR	R1,R2
Load Positive (c)	LPR	10	RR	R1,R2
Load PSW (n,p)	LPSW	82	S	D2(B2)
Load Real Address (c,p)	LRA	B1	RX	R1,D2(X2,B2)
Monitor Call	MC	AF	SI	D1(B1),I2
Move	MVI	92	SI	D1(B1),I2
Move	MVC	D2	SS	D1(L,B1),D2(B2)
Move Long (c)	MVCL	0E	RR	R1,R2
Move Numerics	MVN	D1	SS	D1(L,B1),D2(B2)
Move with Offset	MVO	F1	SS	D1(L1,B1),D2(L2,B2)
Move Zones	MVZ	D3	SS	D1(L,B1),D2(B2)
Multiply	MR	1C	RR	R1,R2
Multiply	M	5C	RX	R1,D2(X2,B2)
Multiply Decimal	MP	FC	SS	D1(L1,B1),D2(L2,B2)
Multiply Halfword	MH	4C	RX	R1,D2(X2,B2)
OR (c)	OR	16	RR	R1,R2
OR (c)	O	56	RX	R1,D2(X2,B2)
OR (c)	OI	96	SI	D1(B1),I2
OR (c)	OC	D6	SS	D1(L,B1),D2(B2)
Pack	PACK	F2	SS	D1(L1,B1),D2(L2,B2)
Purge TLB (p)	PTLB	B20D	S	
Read Direct (p)	RDD	85	SI	D1(B1),I2
Reset Reference Bit (c,p)	RRB	B213	S	D2(B2)
Set Clock (c,p)	SCK	B204	S	D2(B2)
Set Clock Comparator (p)	SCKC	B206	S	D2(B2)

MACHINE INSTRUCTIONS (Contd)

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Set CPU Timer (p)	SPT	B208	S	D2(B2)
Set Prefix (p)	SPX	B210	S	D2(B2)
Set Program Mask (n)	SPM	04	RR	R1
Set PSW Key from Address (p)	SPKA	B20A	S	D2(B2)
Set Storage Key (p)	SSK	08	RR	R1,R2
Set System Mask (p)	SSM	80	S	D2(B2)
Shift and Round Decimal (c)	SRP	F0	SS	D1(L1,B1),D2(B2),I3
Shift Left Double (c)	SLDA	8F	RS	R1,D2(B2)
Shift Left Double Logical	SLDL	8D	RS	R1,D2(B2)
Shift Left Single (c)	SLA	8B	RS	R1,D2(B2)
Shift Left Single Logical	SLL	89	RS	R1,D2(B2)
Shift Right Double (c)	SRDA	8E	RS	R1,D2(B2)
Shift Right Double Logical	SRDL	8C	RS	R1,D2(B2)
Shift Right Single (c)	SRA	8A	RS	R1,D2(B2)
Shift Right Single Logical	SRL	88	RS	R1,D2(B2)
Signal Processor (c,p)	SIGP	AE	RS	R1,R3,D2(B2)
Start I/O (c,p)	SIO	9C00	S	D2(B2)
Start I/O Fast Release (c,p)	SIOF	9C01	S	D2(B2)
Store	ST	50	RX	R1,D2(X2,B2)
Store Channel ID (c,p)	STIDC	B203	S	D2(B2)
Store Character	STC	42	RX	R1,D2(X2,B2)
Store Characters under Mask	STCM	BE	RS	R1,M3,D2(B2)
Store Clock (c)	STCK	B205	S	D2(B2)
Store Clock Comparator (p)	STCKC	B207	S	D2(B2)
Store Control (p)	STCTL	B6	RS	R1,R3,D2(B2)
Store CPU Address (p)	STAP	B212	S	D2(B2)
Store CPU ID (p)	STIDP	B202	S	D2(B2)
Store CPU Timer (p)	STPT	B209	S	D2(B2)
Store Halfword	STH	40	RX	R1,D2(X2,B2)
Store Multiple	STM	90	RS	R1,R3,D2(B2)
Store Prefix (p)	STPX	B211	S	D2(B2)
Store Then AND System Mask (p)	STNSM	AC	SI	D1(B1),I2
Store Then OR System Mask (p)	STOSM	AD	SI	D1(B1),I2
Subtract (c)	SR	1B	RR	R1,R2
Subtract (c)	S	5B	RX	R1,D2(X2,B2)
Subtract Decimal (c)	SP	FB	SS	D1(L1,B1),D2(L2,B2)
Subtract Halfword (c)	SH	4B	RX	R1,D2(X2,B2)
Subtract Logical (c)	SLR	1F	RR	R1,R2
Subtract Logical (c)	SL	5F	RX	R1,D2(X2,B2)
Supervisor Call	SVC	0A	RR	I
Test and Set (c)	TS	93	S	D2(B2)
Test Channel (c,p)	TCH	9F00	S	D2(B2)
Test I/O (c,p)	TIO	9D00	S	D2(B2)
Test under Mask (c)	TM	91	SI	D1(B1),I2
Translate	TR	DC	SS	D1(L,B1),D2(B2)
Translate and Test (c)	TRT	DD	SS	D1(L,B1),D2(B2)
Unpack	UNPK	F3	SS	D1(L1,B1),D2(L2,B2)
Write Direct (p)	WRD	84	SI	D1(B1),I2
Zero and Add Decimal (c)	ZAP	F8	SS	D1(L1,B1),D2(L2,B2)

Floating-Point Instructions

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Add Normalized, Extended (c,x)	AXR	36	RR	R1,R2
Add Normalized, Long (c)	ADR	2A	RR	R1,R2
Add Normalized, Long (c)	AD	6A	RX	R1,D2(X2,B2)
Add Normalized, Short (c)	AER	3A	RR	R1,R2
Add Normalized, Short (c)	AE	7A	RX	R1,D2(X2,B2)
Add Unnormalized, Long (c)	AWR	2E	RR	R1,R2
Add Unnormalized, Long (c)	AW	6E	RX	R1,D2(X2,B2)
Add Unnormalized, Short (c)	AUR	3E	RR	R1,R2
Add Unnormalized, Short (c)	AU	7E	RX	R1,D2(X2,B2)
Compare, Long (c)	CDR	29	RR	R1,R2
Compare, Long (c)	CD	69	RX	R1,D2(X2,B2)
Compare, Short (c)	CER	39	RR	R1,R2
Compare, Short (c)	CE	79	RX	R1,D2(X2,B2)
Divide, Long	DDR	2D	RR	R1,R2
Divide, Long	DD	6D	RX	R1,D2(X2,B2)
Divide, Short	DER	3D	RR	R1,R2
Divide, Short	DE	7D	RX	R1,D2(X2,B2)
Halve, Long	HDR	24	RR	R1,R2
Halve, Short	HER	34	RR	R1,R2
Load and Test, Long (c)	LTDR	22	RR	R1,R2
Load and Test, Short (c)	LTER	32	RR	R1,R2
Load Complement, Long (c)	LCDR	23	RR	R1,R2
Load Complement, Short (c)	LCER	33	RR	R1,R2
Load, Long	LDR	28	RR	R1,R2

Floating-Point Instructions (Contd)

NAME	MNEMONIC	OP CODE	FORMAT	OPERANDS
Load, Long	LD	68	RX	R1,D2(X2,B2)
Load Negative, Long (c)	LNDR	21	RR	R1,R2
Load Negative, Short (c)	LNER	31	RR	R1,R2
Load Positive, Long (c)	LPDR	20	RR	R1,R2
Load Positive, Short (c)	LPER	30	RR	R1,R2
Load Rounded, Extended to Long (x)	LRDR	25	RR	R1,R2
Load Rounded, Long to Short (x)	LRER	35	RR	R1,R2
Load, Short	LER	38	RR	R1,R2
Load, Short	LE	78	RX	R1,D2(X2,B2)
Multiply, Extended (x)	MXR	26	RR	R1,R2
Multiply, Long	MDR	2C	RR	R1,R2
Multiply, Long	MD	6C	RX	R1,D2(X2,B2)
Multiply, Long/Extended (x)	MXDR	27	RR	R1,R2
Multiply, Long/Extended (x)	MXD	67	RX	R1,D2(X2,B2)
Multiply, Short	MER	3C	RR	R1,R2
Multiply, Short	ME	7C	RX	R1,D2(X2,B2)
Store, Long	STD	60	RX	R1,D2(X2,B2)
Store, Short	STE	70	RX	R1,D2(X2,B2)
Subtract Normalized, Extended (c,x)	SXR	37	RR	R1,R2
Subtract Normalized, Long (c)	SDR	2B	RR	R1,R2
Subtract Normalized, Long (c)	SD	6B	RX	R1,D2(X2,B2)
Subtract Normalized, Short (c)	SER	3B	RR	R1,R2
Subtract Normalized, Short (c)	SE	7B	RX	R1,D2(X2,B2)
Subtract Unnormalized, Long (c)	SWR	2F	RR	R1,R2
Subtract Unnormalized, Long (c)	SW	6F	RX	R1,D2(X2,B2)
Subtract Unnormalized, Short (c)	SUR	3F	RR	R1,R2
Subtract Unnormalized, Short (c)	SU	7F	RX	R1,D2(X2,B2)

c. Condition code is set.
n. New condition code is loaded.
p. Privileged instruction.
x. Extended precision floating-point.

EXTENDED MNEMONIC INSTRUCTIONS[†]

Use	Extended Code* (RX or RR)	Meaning	Machine Instr.* (RX or RR)
General	B or BR	Unconditional Branch	BC or BCR 15,
	NOP or NOPR	No Operation	BC or BCR 0,
After Compare Instructions (A:B)	BH or *BHR*	Branch on A High	BC or BCR 2,
	BL or *BLR*	Branch on A Low	BC or BCR 4,
	BE or *BER*	Branch on A Equal B	BC or BCR 8,
	BNH or *BNHR*	Branch on A Not High	BC or BCR 13,
	BNL or *BNLR*	Branch on A Not Low	BC or BCR 11,
	BNE or *BNER*	Branch on A Not Equal B	BC or BCR 7,
After Arithmetic Instructions	BO or *BOR*	Branch on Overflow	BC or BCR 1,
	BP or *BPR*	Branch on Plus	BC or BCR 2,
	BM or *BMR*	Branch on Minus	BC or BCR 4,
	BNP or *BNPR*	Branch on Not Plus	BC or BCR 13,
	BNM or *BNMR*	Branch on Not Minus	BC or BCR 11,
	BNZ or *BNZR*	Branch on Not Zero	BC or BCR 7,
	BZ or *BZR*	Branch on Zero	BC or BCR 8,
After Test under Mask Instruction	BO or *BOR*	Branch if Ones	BC or BCR 1,
	BM or *BMR*	Branch if Mixed	BC or BCR 4,
	BZ or *BZR*	Branch if Zeros	BC or BCR 8,
	BNO or *BNOR*	Branch if Not Ones	BC or BCR 14,

*Second operand not shown; in all cases it is D2(X2,B2) for RX format or R2 for RR format.

[†]For OS/VS and DOS/VS: source: GC33-4010.

EDIT AND EDMK PATTERN CHARACTERS (in hex)

20—digit selector	40—blank	5C—asterisk
21—start of significance	4B—period	6B—comma
22—field separator	5B—dollar sign	C3D9—CR

CONDITION CODES

Condition Code Setting	0	1	2	3
Mask Bit Value	8	4	2	1
General Instructions				
Add, Add Halfword	zero	<zero	>zero	overflow
Add Logical	zero, no carry	not zero, no carry	zero, carry	not zero, carry
AND	zero	not zero	—	—
Compare, Compare Halfword	equal	1st op low	1st op high	—
Compare and Swap/Double	equal	not equal	—	—

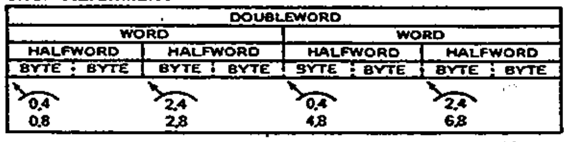
CONDITION CODES (Contd)

Compare Logical	equal	1st op low	1st op high	—
Exclusive OR	zero	not zero	—	—
Insert Characters under Mask	all zero	1st bit one	1st bit zero	—
Load and Test	zero	<zero	>zero	—
Load Complement	zero	<zero	>zero	overflow
Load Negative	zero	<zero	—	—
Load Positive	zero	—	>zero	overflow
Move Long	count equal	count low	count high	overlap
OR	zero	not zero	—	—
Shift Left Double/Single	zero	<zero	>zero	overflow
Shift Right Double/Single	zero	<zero	>zero	—
Store Clock	set	not set	error	not oper
Subtract, Subtract Halfword	zero	<zero	>zero	overflow
Subtract Logical	—	not zero, no carry	zero, carry	not zero, carry
Test and Set	zero	one	—	—
Test under Mask	zero	mixed	—	ones
Translate and Test	zero	incomplete	complete	—
Decimal Instructions				
Add Decimal	zero	<zero	>zero	overflow
Compare Decimal	equal	1st op low	1st op high	—
Edit, Edit and Mark	zero	<zero	>zero	—
Shift and Round Decimal	zero	<zero	>zero	overflow
Subtract Decimal	zero	<zero	>zero	overflow
Zero and Add	zero	<zero	>zero	overflow
Floating-Point Instructions				
Add Normalized	zero	<zero	>zero	—
Add Unnormalized	zero	<zero	>zero	—
Compare	equal	1st op low	1st op high	—
Load and Test	zero	<zero	>zero	—
Load Complement	zero	<zero	>zero	—
Load Negative	zero	<zero	—	—

CONDITION CODES (Contd)

Load Positive	zero	—	>zero	—
Subtract Normalized	zero	<zero	>zero	—
Subtract Unnormalized	zero	<zero	>zero	—
Input/Output Instructions				
Clear I/O	no oper in progress	CSW stored	chan busy	not oper
Halt Device	interruption pending	CSW stored	channel working	not oper
Halt I/O	interruption pending	CSW stored	burst op stopped	not oper
Start I/O, SIOF	successful	CSW stored	busy	not oper
Store Channel ID	ID stored	CSW stored	busy	not oper
Test Channel	available	interruption pending	burst mode	not oper
Test I/O	available	CSW stored	busy	not oper
System Control Instructions				
Load Real Address	translation available	ST entry invalid	PT entry invalid	length violation
Reset Reference Bit	R=0, C=0	R=0, C=1	R=1, C=0	R=1, C=1
Set Clock	set	secure	—	not oper
Signal Processor	accepted	stat stored	busy	not oper

CNOP ALIGNMENT



APPENDIX

D

DOS AND OS JOB CONTROL

This appendix provides some typical examples of job control under DOS and OS.

DOS JOB CONTROL

Here is an example of conventional job control to assemble and execute a program under DOS:

`// JOB jobname`	Jobname may be 1–8 characters.
`// OPTION DUMP,LIST,LOG,XREF`	DUMP: Print contents of storage on abnormal execute error (or NODUMP). LIST: List the assembled program (or NOLIST). LOG: Print the job control statements (or NOLOG). XREF: Print a cross-reference of symbolic names after the assembly (or NOXREF).
`   ACTION MAP`	Print a map of the link-edited program (or NOMAP).

```
// EXEC ASSEMBLY                  Load assembler and begin assembly.
     ... (source program here)
/*                                End of assembly.
// EXEC LNKEDT                    Perform link edit.
// EXEC                           Load linked module into storage; begin
                                  execution.
     ... (input data here)
/*                                End of input data.
/&                                End of job; return to supervisor.
```

Larger DOS systems provide for cataloging commonly used job control on disk in the procedure library. The preceding example of job control could be cataloged, for example, to provide for automatic assembly, link edit, and execute through the use of only a few job commands, as follows:

```
* $$ JOB jobname                  Jobname may be 1-8 characters.
// EXEC PROC=ASSEMBLY             Cataloged procedure ASSEMBLY con-
                                  tains assembly, link-edit, and execute job
                                  commands.
     ... (source program here)

/*                                End of assembly.
     ... (input data here)

/*                                End of input data.
/&                                End of job.
* $$ EOJ
```

DOS Job Control for Magnetic Tape

The job commands for magnetic tape are similar to those for the system reader and printer. However, tape files require additional information on a TLBL job command to provide greater control over the file.

```
// TLBL filename,'file-ID',date,file-serial-no.,volume-sequence
no.,file-sequence-no.,generation-no.,version-no.
```

filename	Name of the DTFMT, the only required entry.
'file-ID'	The file identifier in the file label, 1–17 characters.
retention date	One of two formats for output files: (1) yy/ddd, the date of retention; e.g., 95/030 tells the system to retain the file until Jan. 30, 1995; (2) dddd, a retention period in days.
file serial number	1–6 characters, the volume serial number for the first or only volume of the file.

volume sequence number	1–4 digits for the volume number in a multi-volume file.
file sequence number	1–4 digits for the file number in a multifile volume.
generation number	1–4 digits for the generation number.
version number	1–2 digits for the version number.

If you omit any of the last four entries, the system assumes 1 if output and ignores if input.

DOS Job Control for Direct Access Storage Devices

Each extent (disk area) for a disk file requires two job control commands, DLBL and EXTENT, equivalent to the magnetic tape TLBL job command. Note that you may store a file on more than one extent. DLBL and EXTENT follow the LNKEDT command, coded as follows:

```
// EXEC LNKEDT
// DLBL filename...
// EXTENT symbolic-unit...
```

Here are details for the DLBL and EXTENT commands:

```
// DLBL filename,'file-ID',date,codes
```

filename	Name of the DTFSD, 1–7 characters.
'file-ID'	1–44 characters, between apostrophes. This is the first field of the format 1 label. You can code the file ID and optionally generation and version number. If you omit this entry, the system uses the filename.
retention date	One of two formats for output: (1) dddd = retention period in days; (2) yy/ddd = date of retention; e.g., 95/030 means retain file until January 30, 1995. If you omit this entry, the system assumes 7 days.
codes	Type of file label: SD is sequential disk; ISC is index sequential create; ISE is index sequential extend; DA is direct access. If you omit this entry, the system assumes SD.

```
// EXTENT symbolic-unit,serial-no.,type,sequence-no.,relative-
track,number-of-tracks,split-cylinder-track
```

symbolic unit	The symbolic unit SYSnnn for the file. If you omit this entry, the system assumes the unit from the preceding EXTENT, if any.

serial number	The volume serial number for the volume. If you omit this entry, the system uses the number from the preceding EXTENT, if any.
type	The type of extent, where 1 is data area with no split cylinder; 2 is independent overflow area for IS; 4 is index area for IS; 8 is data area, split cylinder. If omitted, the system assumes type 1.
sequence number	The sequence number (0–255) of this extent in a multiextent file. Not required for SD and DA, but if used, the extent begins with 0. For IS with a master index, the number begins with 0; otherwise IS files begin with extent 1.
relative track	1–5 digits to indicate the sequential track number, relative to 0, where the extent begins. The formula to calculate the relative track is $\text{RT} = \text{tracks per cylinder} \times \text{cylinder number} + \text{track number}$ Example for a 3350 (30 tracks/cylinder), on cylinder 3, track 4: $\text{RT} = (30 \times 3) + 4 = 94$
number of tracks	1–5 digits to indicate the number of tracks allocated for the file on this extent.
split cylinder track	Digits 0–19 to signify the upper track number for split cylinders in SD files. (There may be more than one SD file within a cylinder.)

OS JOB CONTROL

There are different versions of OS job control language. The following illustrates one version, providing for assembly, link edit, and execution of test data. The program uses the system reader and a printer file, both of which require a DD (data definition) job command.

```
//jobname JOB [optional account#,acctg-information,programmer-name]

//stepname EXEC ASMGCLG       Use ASMG to assemble, with no execute.
                   └┼┼┼──────Level of assembler, e.g., F or G.
                    └┼┼──────Compile (assemble).
                     └┼──────Link-edit the assembled program.
                      └──────Go, or execute the linked program.
//ASM.SYSIN DD *              The * means that the source program
                              immediately follows in the job stream.
```

```
    ...(source program here)
/*                            End of assembly.
//GO.SYSUDUMP DD SYSOUT=A     Causes printing of execution error
                              diagnostics.
//GO.printername DD SYSOUT=A  Data definition for printer in program
                              DCB. (A is class of output for printer.)
//GO.readername DD *          Data definition for system reader.  (*
                              indicates that input data immediately follows.)
    ...(input data here)
/*                            End of input data.
//                            Optional entry for end of job.
```

Descriptions of the OS EXEC and DD commands follow. As a convention:

- Braces { } indicate a choice of one entry.
- Brackets [ ] indicate an optional entry from which you may choose one entry or none.
- Parentheses ( ), where they appear, must be coded.

The OS EXEC Command

The general format for the OS EXEC command is the following:

```
                         ⎧ PGM=programname                          ⎫
                         ⎪ PGM=*.stepname.ddname                    ⎪
//[stepname] EXEC        ⎨ PGM=*.stepname.procstepname.ddname       ⎬
                         ⎪ [PROC=]procedure-name                    ⎪
                         ⎩ [other options]                          ⎭
```

Other options for EXEC include ACCT (accounting information), COND, DPRTY (for MVT), PARM (parameter), RD (restart definition), REGION (for MVT), ROLL (for MVT), and TIME (to assign CPU time limit for a step).

```
ACCT[.procstepname]=(accounting information)

                      ⎡(code,operator)                         ⎤
COND[.procstepname]=  ⎢(code,operator,stepname)                ⎥
                      ⎣(code,operator,stepname,procstepname)   ⎦

DPRTY[.procstepname]=(value1,value2)

PARM[.procstepname]=value

RD[.procstepname]=R or RNC or NC or NR

REGION[.procstepname]=(valueK[,value1K])
```

```
ROLL[.procstepname]=({YES | NO} {,YES | ,NO})

TIME[.procstepname]=(mins,secs)
```

The OS DD Command

The DD (data definition) command defines the name and property of each device that the program requires. Its general format is the following:

```
//ddname      DD   operand
   procstepname.ddname
```

The operand for DD permits a variety of options, as follows:

```
[*   ]                                              Define a data set in the
[DATA]                                              input stream.

[DCB=(attributes)                             ]
[DCB=(dsname[,attributes])                    ]     Completion of data
[DCB=(*.ddname[,attributes])                  ]     control block.
[DCB=(*.stepname.ddname[,attributes])         ]
[DCB=(*.stepname.procstep.ddname[,attributes])]

[DDNAME=ddname]                                     Postpones definition of
                                                    the data set.

[      ([NEW][,DELETE  ][,DELETE  ])]               Assigns status,
[      ([OLD][,KEEP    ][,KEEP    ])]               disposition, and
[DISP= ([SHR][,PASS    ][,CATLG   ])]               conditional disposition
[      ([MOD][,CATLG   ][,UNCATLG ])]               of the data set.
[      (     [,UNCATLG ]           )]

[        {dsname                           }]
[        {dsname(areaname)                 }]
[        {dsname(membername)               }]
[        {dsname(generation#)              }]       Abbreviated as DSN.
[DSNAME= {&&dsname                         }]       Assign name to new or
[        {&&dsname(areaname)               }]       existing data set.
[        {&&dsname(membername)             }]
[        {*.ddname                         }]
[        {*.stepname.ddname                }]
[        {*.stepname.procstepname.ddname   }]

[FCB=(image-id [,ALIGN ])]                          Forms control for 3211
[              [,VERIFY])]                          printer.
```

`[LABEL=([data set seq#][parameters])`	[Label information (see below).
`[SPACE=(parameters)]`	Allocate space on disk for a new data set (see below).
`[SYSOUT=(classname[,programname][,form#])[OUTLIM=no.]]`	Route a data set through the output job stream.
`[UNIT=(parameters)]`	Unit information.
`[VOLUME=(parameters)]`	Also VOL. Provide information about the volume (see below).

The following describes in detail the parameters for LABEL, UNIT, and VOLUME.

```
                        ┌,SL ┐
                        │,SUL│
                        │,AL │┌,PASSWORD┐┌,IN ┐     ┌EXPDT=yymmdd┐
LABEL=( [dataset seq#]  │,AUL││        ││    │[,]  │            │ )*
                        │,NSL│└,NOPWREAD┘└,OUT┘     └RETPD=nnnn  ┘
                        │,NL │
                        └,BLP┘

        ┌TRK      ┐          ┌,secondary┐┌,directory┐  ┌,RLSE┐┌,CONTIG┐
SPACE=( │CYL      │ (,primary│          ││          │) │     ││,MXIG  │[,ROUND] )
        └blocksize┘          └,         ┘└,index    ┘  └,    ┘└,ALX   ┘

SPACE=(ABSTR,(primary qty, address┌,directory┐))
                                  └,index    ┘

       ┌unit-address┐┌,count┐
UNIT=( │device-type ││,P    │[,DEFER][,SEP=(ddname1, ...)] )
       └group-name  ┘└,     ┘

UNIT=AFF=ddname

VOLUME= ( [PRIVATE]┌,RETAIN┐┌,volseq#┐[,volcount][,]┌SER=serial#,...┐ )
(or VOL)           └,      ┘└,       ┘              │REF=dsname     │
                                                    └REF=*.ddname   ┘
```

*EXPDT is expiration date and RETPD is retention period.

For REF, other entries are

```
REF=*.stepname.ddname
REF=*.stepname.procstepname.ddname
```

Other DD operands include these:

`DUMMY`	Bypass I/O on a data set under BSAM and QSAM.
`DYNAM`	Request dynamic allocation under MVT with TSO.
`AFF=ddname`	Request channel separation.
`OUTLIM=number`	Limit the number of logical records to be included in an output data set.
`SPLIT=operand`	Assign space for a new data set on a disk device and to share cylinders.
`SUBALLOC=operand`	Request part of the space on a disk device that the job assigned earlier.
`TERM=TS`	Inform the system that data is transferring to or from a timesharing terminal.

APPENDIX

E

SPECIAL MACROS: INIT, PUTPR, DEFIN, DEFPR, EOJ

This appendix describes the special macros INIT, PUTPR, DEFIN, DEFPR, and EOJ used at the beginning of this text to handle program initialization and input/output. The macros are simple to implement and to use, and anyone is free to catalog them. Beginners often have trouble coding the regular full macros, making punctuation and spelling errors and omitting entries. The use of macros such as the ones in this appendix can avert a lot of initial coding errors and can free beginners to concentrate on programming logic.

The INIT macro, which is used for initializing base register addressing, requires versions for both DOS and OS, shown in Figs. E-1 and E-2, respectively. A further recommended refinement could include the DOS STXIT or OS SPIE macro for error recovery.

```
         MACRO
&INITZE  INIT
&INITZE  BALR  3,0                  LOAD BASE REGISTER 3
         USING *,3,4,5              ASSIGN BASE REGS 3,4 & 5
         LA    5,2048               LOAD X'800' (1/2 OF X'1000')
         LA    4,2048(3,5)          LOAD BASE REG 4
         LA    5,2048(4,5)          LOAD BASE REG 5
         MEND
```

Figure E-1 The DOS INIT macro.

```
          MACRO
&INITZE   INIT
&INITZE   SAVE  (14,12)            SAVE REGS FOR SUPERVISOR
          BALR  3,0
          USING 3,4,5
          ST    13,SAVEAREA+4      SAVE ADDRESSES FOR RETURN
          LA    13,SAVEAREA            TO SUPERVISOR
          LA    5,2048             LOAD X'800' (1/2 OF X'1000')
          LA    4,2048(3,5)        LOAD BASE REG 4
          LA    5,2048(4,5)        LOAD BASE REG 5
          B     SAVEAREA+18*4
          SPACE
SAVEAREA  DS    18F                SAVE AREA FOR INTERRUPTS
          MEND
```

Figure E-2 The OS INIT macro.

```
          MACRO
&WRITE    PUTPR &FILE,&PRAREA,&CTLCHR
          LCLC  &CTL
.*
.VALAREA AIF   ('&CTLCHR' NE 'WSP1').NEXT1   PRINT & SPACE 1?
&CTL      SETC  'X''09'''
          AGO   .NEXT9
.*
.NEXT1    AIF   ('&CTLCHR' NE 'WSP2').NEXT2   PRINT & SPACE 2?
&CTL      SETC  'X''11'''
          AGO   .NEXT9
.*
.NEXT2    AIF   ('&CTLCHR' NE 'WSP3').NEXT3   PRINT & SPACE 3?
&CTL      SETC  'X''19'''
          AGO   .NEXT9
.*
.NEXT3    AIF   ('&CTLCHR' NE 'SP1').NEXT4    SPACE 1, NO PRINT?
&CTL      SETC  'X''0B'''
          AGO   .NEXT9
.*
.NEXT4    AIF   ('&CTLCHR' NE 'SP2').NEXT5    SPACE 2, NO PRINT?
&CTL      SETC  'X''13'''
          AGO   .NEXT9
.*
.NEXT5    AIF   ('&CTLCHR' NE 'SP3').NEXT6    SPACE 3, NO PRINT?
&CTL      SETC  'X''1B'''
          AGO   .NEXT9
.*
.NEXT6    AIF   ('&CTLCHR' NE 'SK1').NEXT7    SKIP TO NEW PAGE?
&CTL      SETC  'X''8B'''
          AGO   .NEXT9
.*
.NEXT7    AIF   ('&CTLCHR' NE 'WSP0').NEXT8   PRINT & SPACE 0?
&CTL      SETC  'X''01'''
          AGO   .NEXT9
.*
.NEXT8    MNOTE 1,'INVALID PRINT CONTROL - DEFAULT TO WSP1'
&CTL      SETC  'X''09'''
.*
.NEXT9    ANOP
&WRITE    MVI   &PRAREA,&CTL               MOVE CTL CHAR TO PRINT
          PUT   &FILE,&PRAREA              *    & PRINT
.NEXT10   ANOP
          MEND
```

Figure E-3 The PUTPR macro.

The PUTPR macro, shown in Fig. E-3, generates two instructions, of the form:

```
MVI PRINT,X'nn'      Insert control character
PUT PRTR,PRINT       Print line
```

If the control character is invalid, the macro instruction defaults to write and space one line.

The DEFIN macro defines the system reader and assume the use of a workarea for input. (That is, you code GET filename,workarea.) The macro usefully checks the validity of the supplied end-of-file address. The DOS version, shown in Fig. E-4, generates a DTFCD, whereas the OS version, shown in Fig. E-5, generates a DCB. The particular entries may vary by installation.

```
         MACRO
&FILEIN  DEFIN &EOF
         AIF   (T'&EOF EQ 'I' OR T'&EOF EQ 'M').A10  EOF ADDRESS VALID?
         MNOTE 1,'EOF ADDRESS NOT DEFINED'   NO -
&EOF     CLOSE &FILEIN                   *     GENERATE EOF ROUTINE
         EOJ
.A10     ANOP
&FILEIN  DTFCD BLKSIZE=80,               DEFINE INPUT FILE          +
               DEVADDR=SYSIPT,                                      +
               DEVICE=2540,                                         +
               EOFADDR=&EOF,                                        +
               IOAREA1=INBUFF1,                                     +
               IOAREA2=INBUFF2,                                     +
               TYPEFLE=INPUT,                                       +
               WORKA=YES
         SPACE
INBUFF1  DC    CL80' '                   INPUT BUFFER-1
INBUFF2  DC    CL80' '                   INPUT BUFFER-2
         MEND
```

Figure E-4 The DOS DEFIN macro.

```
         MACRO
&FILEIN  DEFIN &EOF
         AIF   (T'&EOF EQ 'I' OR T'&EOF EQ 'M').A10  EOF ADDRESS VALID?
         MNOTE 1,'EOF ADDRESS NOT DEFINED'   NO -
&EOF     CLOSE &FILEIN                   *     GENERATE EOF ROUTINE
         EOJ
.A10     ANOP
&FILEIN  DCB   DDNAME=SYSIN,             DEFINE INPUT FILE          +
               DEVD=DA,                                             +
               DSORG=PS,                                            +
               EODAD=&EOF,                                          +
               MACRF=(GM)
         MEND
```

Figure E-5 The OS DEFIN macro.

```
         MACRO
&PRFILE  DEFPR
&PRFILE  DTFPR BLKSIZE=133,                  DEFINE OUTPUT FILE        +
               CTLCHR=YES,                                             +
               DEVADDR=SYSLST,                                         +
               DEVICE=3203,                                            +
               IOAREA1=PRBUFF1,                                        +
               IOAREA2=PRBUFF2,                                        +
               WORKA=YES
         SPACE
PRBUFF1  DC    CL133' '                      OUTPUT BUFFER-1
PRBUFF2  DC    CL133' '                      OUTPUT BUFFER-2
         SPACE
         MEND
```

Figure E-6 The DOS DEFPR macro.

```
         MACRO
&PRFILE  DEFPR
&PRFILE  DCB   DDNAME=SYSPRINT,                                        +
               DEVD=DA,                                                +
               DSORG=PS,                                               +
               RECFM=FBSM,                                             +
               MACRF=(PM)
SYSPRINT EQU   &PRFILE
         ENTRY SYSPRINT
         MEND
```

Figure E-7 The OS DEFPR macro.

```
         MACRO
&LABEL   EOJ
&LABEL   L     13,SAVEAREA+4              END-OF-JOB
         RETURN (14,12)                   RETURN TO SUPERVISOR
         MEND
```

Figure E-8 The OS EOJ macro.

The DEFPR macro defines the printer and assumes the use of a workarea for output. (That is, you code PUT filename,workarea.) The DOS version, shown in Fig. E-6, generates a DTFPR, whereas the OS version, shown in Fig. E-7, generates a DCB. The particular entries may vary by installation.

DOS already has a simple EOJ macro. The OS EOJ macro, shown in Fig. E-8, generates the load savearea and return and ties in with the OS INIT macro.

APPENDIX

F

EBCDIC CODE REPRESENTATION

CODE TRANSLATION TABLE

Dec.	Hex	Instruction (RR)	Graphics and Controls BCDIC	EBCDIC(1)	ASCII	EBCDIC Card Code	Binary
0	00			NUL	NUL	12-0-1-8-9	0000 0000
1	01			SOH	SOH	12-1-9	0000 0001
2	02			STX	STX	12-2-9	0000 0010
3	03			ETX	ETX	12-3-9	0000 0011
4	04	SPM		PF	EOT	12-4-9	0000 0100
5	05	BALR		HT	ENQ	12-5-9	0000 0101
6	06	BCTR		LC	ACK	12-6-9	0000 0110
7	07	BCR		DEL	BEL	12-7-9	0000 0111
8	08	SSK			BS	12-8-9	0000 1000
9	09	ISK			HT	12-1-8-9	0000 1001
10	0A	SVC		SMM	LF	12-2-8-9	0000 1010
11	0B			VT	VT	12-3-8-9	0000 1011
12	0C			FF	FF	12-4-8-9	0000 1100
13	0D			CR	CR	12-5-8-9	0000 1101
14	0E	MVCL		SO	SO	12-6-8-9	0000 1110
15	0F	CLCL		SI	SI	12-7-8-9	0000 1111
16	10	LPR		DLE	DLE	12-11-1-8-9	0001 0000
17	11	LNR		DC1	DC1	11-1-9	0001 0001
18	12	LTR		DC2	DC2	11-2-9	0001 0010
19	13	LCR		TM	DC3	11-3-9	0001 0011
20	14	NR		RES	DC4	11-4-9	0001 0100
21	15	CLR		NL	NAK	11-5-9	0001 0101
22	16	OR		BS	SYN	11-6-9	0001 0110
23	17	XR		IL	ETB	11-7-9	0001 0111
24	18	LR		CAN	CAN	11-8-9	0001 1000
25	19	CR		EM	EM	11-1-8-9	0001 1001
26	1A	AR		CC	SUB	11-2-8-9	0001 1010
27	1B	SR		CU1	ESC	11-3-8-9	0001 1011
28	1C	MR		IFS	FS	11-4-8-9	0001 1100
29	1D	DR		IGS	GS	11-5-8-9	0001 1101
30	1E	ALR		IRS	RS	11-6-8-9	0001 1110
31	1F	SLR		IUS	US	11-7-8-9	0001 1111
32	20	LPDR		DS	SP	11-0-1-8-9	0010 0000
33	21	LNDR		SOS	! !	0-1-9	0010 0001
34	22	LTDR		FS	"	0-2-9	0010 0010
35	23	LCDR			#	0-3-9	0010 0011
36	24	HDR		BYP	$	0-4-9	0010 0100
37	25	LRDR		LF	%	0-5-9	0010 0101
38	26	MXR		ETB	&	0-6-9	0010 0110
39	27	MXDR		ESC	'	0-7-9	0010 0111

CODE TRANSLATION TABLE (Contd)

Dec.	Hex	Instruction (RX)	Graphics and Controls BCDIC	EBCDIC(1)	ASCII	EBCDIC Card Code	Binary
40	28	LDR			(	0-8-9	0010 1000
41	29	CDR			)	0-1-8-9	0010 1001
42	2A	ADR		SM	*	0-2-8-9	0010 1010
43	2B	SDR		CU2	+	0-3-8-9	0010 1011
44	2C	MDR			,	0-4-8-9	0010 1100
45	2D	DDR		ENQ	-	0-5-8-9	0010 1101
46	2E	AWR		ACK	.	0-6-8-9	0010 1110
47	2F	SWR		BEL	/	0-7-8-9	0010 1111
48	30	LPER			0	12-11-0-1-8-9	0011 0000
49	31	LNER			1	1-9	0011 0001
50	32	LTER		SYN	2	2-9	0011 0010
51	33	LCER			3	3-9	0011 0011
52	34	HER		PN	4	4-9	0011 0100
53	35	LRER		RS	5	5-9	0011 0101
54	36	AXR		UC	6	6-9	0011 0110
55	37	SXR		EOT	7	7-9	0011 0111
56	38	LER			8	8-9	0011 1000
57	39	CER			9	1-8-9	0011 1001
58	3A	AER			:	2-8-9	0011 1010
59	3B	SER		CU3	;	3-8-9	0011 1011
60	3C	MER		DC4	<	4-8-9	0011 1100
61	3D	DER		NAK	=	5-8-9	0011 1101
62	3E	AUR			>	6-8-9	0011 1110
63	3F	SUR		SUB	?	7-8-9	0011 1111
64	40	STH		Sp Sp	@	no punches	0100 0000
65	41	LA			A	12-0-1-9	0100 0001
66	42	STC			B	12-0-2-9	0100 0010
67	43	IC			C	12-0-3-9	0100 0011
68	44	EX			D	12-0-4-9	0100 0100
69	45	BAL			E	12-0-5-9	0100 0101
70	46	BCT			F	12-0-6-9	0100 0110
71	47	BC			G	12-0-7-9	0100 0111
72	48	LH			H	12-0-8-9	0100 1000
73	49	CH			I	12-1-8	0100 1001
74	4A	AH		¢ ¢	J	12-2-8	0100 1010
75	4B	SH	.	. .	K	12-3-8	0100 1011
76	4C	MH	⌑)	< <	L	12-4-8	0100 1100
77	4D		[	((	M	12-5-8	0100 1101
78	4E	CVD	<	+ +	N	12-6-8	0100 1110
79	4F	CVB	‡	| |	O	12-7-8	0100 1111

CODE TRANSLATION TABLE (Contd)

Dec.	Hex	Instruction and Format	Graphics and Controls				EBCDIC Card Code	Binary
			BCDIC	EBCDIC(1)		ASCII		
80	50	ST	& +	&	&	P	12	0101 0000
81	51					Q	12-11-1-9	0101 0001
82	52					R	12-11-2-9	0101 0010
83	53					S	12-11-3-9	0101 0011
84	54	N				T	12-11-4-9	0101 0100
85	55	CL				U	12-11-5-9	0101 0101
86	56	O				V	12-11-6-9	0101 0110
87	57	X				W	12-11-7-9	0101 0111
88	58	L	.			X	12-11-8-9	0101 1000
89	59	C				Y	11-1-8	0101 1001
90	5A	A		!	!	Z	11-2-8	0101 1010
91	5B	S	$	$	$	[	11-3-8	0101 1011
92	5C	M	*	*	*	\	11-4-8	0101 1100
93	5D	D	]	)	)	]	11-5-8	0101 1101
94	5E	AL	;	;	;	¬ ^	11-6-8	0101 1110
95	5F	SL	Δ	¬	¬	_	11-7-8	0101 1111
96	60	STD	-	-	-	`	11	0110 0000
97	61		/	/	/	a	0-1	0110 0001
98	62					b	11-0-2-9	0110 0010
99	63					c	11-0-3-9	0110 0011
100	64					d	11-0-4-9	0110 0100
101	65					e	11-0-5-9	0110 0101
102	66					f	11-0-6-9	0110 0110
103	67	MXD				g	11-0-7-9	0110 0111
104	68	LD				h	11-0-8-9	0110 1000
105	69	CD				i	0-1-8	0110 1001
106	6A	AD		¦		j	12-11	0110 1010
107	6B	SD	,	,	,	k	0-3-8	0110 1011
108	6C	MD	% (	%	%	l	0-4-8	0110 1100
109	6D	DD	γ	_	_	m	0-5-8	0110 1101
110	6E	AW	\	>	>	n	0-6-8	0110 1110
111	6F	SW	⧻	?	?	o	0-7-8	0110 1111
112	70	STE				p	12-11-0	0111 0000
113	71					q	12-11-0-1-9	0111 0001
114	72					r	12-11-0-2-9	0111 0010
115	73					s	12-11-0-3-9	0111 0011
116	74					t	12-11-0-4-9	0111 0100
117	75					u	12-11-0-5-9	0111 0101
118	76					v	12-11-0-6-9	0111 0110
119	77					w	12-11-0-7-9	0111 0111
120	78	LE				x	12-11-0-8-9	0111 1000
121	79	CE		`		y	1-8	0111 1001
122	7A	AE	ƀ	:	:	z	2-8	0111 1010
123	7B	SE	# =	#	#	{	3-8	0111 1011
124	7C	ME	@ '	@	@	¦	4-8	0111 1100
125	7D	DE	:	'	'	}	5-8	0111 1101
126	7E	AU	>	=	=	~	6-8	0111 1110
127	7F	SU	√	"	"	DEL	7-8	0111 1111
128	80	SSM -S					12-0-1-8	1000 0000
129	81			a	a		12-0-1	1000 0001
130	82	LPSW -S		b	b		12-0-2	1000 0010
131	83	Diagnose		c	c		12-0-3	1000 0011
132	84	WRD SI		d	d		12-0-4	1000 0100
133	85	RDD		e	e		12-0-5	1000 0101
134	86	BXH		f	f		12-0-6	1000 0110
135	87	BXLE		g	g		12-0-7	1000 0111
136	88	SRL		h	h		12-0-8	1000 1000
137	89	SLL		i	i		12-0-9	1000 1001
138	8A	SRA					12-0-2-8	1000 1010
139	8B	SLA RS			{		12-0-3-8	1000 1011
140	8C	SRDL			≤		12-0-4-8	1000 1100
141	8D	SLDL			(		12-0-5-8	1000 1101
142	8E	SRDA			+		12-0-6-8	1000 1110
143	8F	SLDA			+		12-0-7-8	1000 1111
144	90	STM					12-11-1-8	1001 0000
145	91	TM SI		j	j		12-11-1	1001 0001
146	92	MVI		k	k		12-11-2	1001 0010
147	93	TS -S		l	l		12-11-3	1001 0011
148	94	NI		m	m		12-11-4	1001 0100
149	95	CLI SI		n	n		12-11-5	1001 0101
150	96	OI		o	o		12-11-6	1001 0110
151	97	XI		p	p		12-11-7	1001 0111
152	98	LM -RS		q	q		12-11-8	1001 1000
153	99			r	r		12-11-9	1001 1001
154	9A						12-11-2-8	1001 1010
155	9B				}		12-11-3-8	1001 1011

CODE TRANSLATION TABLE (Contd)

Dec.	Hex	Instruction (SS)	Graphics and Controls				EBCDIC Card Code	Binary
			BCDIC	EBCDIC(1)		ASCII		
156	9C	SIO, SIOF			¤		12-11-4-8	1001 1100
157	9D	TIO, CLRIO S			)		12-11-5-8	1001 1101
158	9E	HIO, HDV			±		12-11-6-8	1001 1110
159	9F	TCH			■		12-11-7-8	1001 1111
160	A0				-		11-0-1-8	1010 0000
161	A1			~	°		11-0-1	1010 0001
162	A2			s	s		11-0-2	1010 0010
163	A3			t	t		11-0-3	1010 0011
164	A4			u	u		11-0-4	1010 0100
165	A5			v	v		11-0-5	1010 0101
166	A6			w	w		11-0-6	1010 0110
167	A7			x	x		11-0-7	1010 0111
168	A8			y	y		11-0-8	1010 1000
169	A9			z	z		11-0-9	1010 1001
170	AA						11-0-2-8	1010 1010
171	AB				└		11-0-3-8	1010 1011
172	AC	STNSM SI			┌		11-0-4-8	1010 1100
173	AD	STOSM			[		11-0-5-8	1010 1101
174	AE	SIGP -RS			≥		11-0-6-8	1010 1110
175	AF	MC -SI			●		11-0-7-8	1010 1111
176	B0				0		12-11-0-1-8	1011 0000
177	B1	LRA -RX			1		12-11-0-1	1011 0001
178	B2	See below			2		12-11-0-2	1011 0010
179	B3				3		12-11-0-3	1011 0011
180	B4				4		12-11-0-4	1011 0100
181	B5				5		12-11-0-5	1011 0101
182	B6	STCTL RS			6		12-11-0-6	1011 0110
183	B7	LCTL			7		12-11-0-7	1011 0111
184	B8				8		12-11-0-8	1011 1000
185	B9				9		12-11-0-9	1011 1001
186	BA	CS RS					12-11-0-2-8	1011 1010
187	BB	CDS			┘		12-11-0-3-8	1011 1011
188	BC				┐		12-11-0-4-8	1011 1100
189	BD	CLM			]		12-11-0-5-8	1011 1101
190	BE	STCM RS			≠		12-11-0-6-8	1011 1110
191	BF	ICM			—		12-11-0-7-8	1011 1111
192	C0		?	{			12-0	1100 0000
193	C1		A	A	A		12-1	1100 0001
194	C2		B	B	B		12-2	1100 0010
195	C3		C	C	C		12-3	1100 0011
196	C4		D	D	D		12-4	1100 0100
197	C5		E	E	E		12-5	1100 0101
198	C6		F	F	F		12-6	1100 0110
199	C7		G	G	G		12-7	1100 0111
200	C8		H	H	H		12-8	1100 1000
201	C9		I	I	I		12-9	1100 1001
202	CA						12-0-2-8-9	1100 1010
203	CB						12-0-3-8-9	1100 1011
204	CC			ʃ			12-0-4-8-9	1100 1100
205	CD						12-0-5-8-9	1100 1101
206	CE			Y			12-0-6-8-9	1100 1110
207	CF						12-0-7-8-9	1100 1111
208	D0		!	}			11-0	1101 0000
209	D1	MVN	J	J	J		11-1	1101 0001
210	D2	MVC	K	K	K		11-2	1101 0010
211	D3	MVZ	L	L	L		11-3	1101 0011
212	D4	NC	M	M	M		11-4	1101 0100
213	D5	CLC	N	N	N		11-5	1101 0101
214	D6	OC	O	O	O		11-6	1101 0110
215	D7	XC	P	P	P		11-7	1101 0111
216	D8		Q	Q	Q		11-8	1101 1000
217	D9		R	R	R		11-9	1101 1001
218	DA						12-11-2-8-9	1101 1010
219	DB						12-11-3-8-9	1101 1011
220	DC	TR					12-11-4-8-9	1101 1100
221	DD	TRT					12-11-5-8-9	1101 1101
222	DE	ED					12-11-6-8-9	1101 1110
223	DF	EDMK					12-11-7-8-9	1101 1111
224	E0		‡	\			0-2-8	1110 0000
225	E1						11-0-1-9	1110 0001
226	E2		S	S	S		0-2	1110 0010
227	E3		T	T	T		0-3	1110 0011
228	E4		U	U	U		0-4	1110 0100
229	E5		V	V	V		0-5	1110 0101
230	E6		W	W	W		0-6	1110 0110
231	E7		X	X	X		0-7	1110 0111

CODE TRANSLATION TABLE (Contd)

Dec.	Hex	Instruction and Format	Graphics and Controls BCDIC	EBCDIC(1)	ASCII	EBCDIC Card Code	Binary
232	E8		Y	Y	Y	0-8	1110 1000
233	E9		Z	Z	Z	0-9	1110 1001
234	EA					11-0-2-8-9	1110 1010
235	EB					11-0-3-8-9	1110 1011
236	EC			⑂		11-0-4-8-9	1110 1100
237	ED					11-0-5-8-9	1110 1101
238	EE					11-0-6-8-9	1110 1110
239	EF					11-0-7-8-9	1110 1111
240	F0	SRP	0	0	0	0	1111 0000
241	F1	MVO	1	1	1	1	1111 0001
242	F2	PACK	2	2	2	2	1111 0010
243	F3	UNPK	3	3	3	3	1111 0011

CODE TRANSLATION TABLE (Contd)

Dec.	Hex	Instruction (SS)	Graphics and Controls BCDIC	EBCDIC(1)	ASCII	EBCDIC Card Code	Binary
244	F4		4	4	4	4	1111 0100
245	F5		5	5	5	5	1111 0101
246	F6		6	6	6	6	1111 0110
247	F7		7	7	7	7	1111 0111
248	F8	ZAP	8	8	8	8	1111 1000
249	F9	CP	9	9	9	9	1111 1001
250	FA	AP		|		12-11-0-2-8-9	1111 1010
251	F8	SP				12-11-0-3-8-9	1111 1011
252	FC	MP				12-11-0-4-8-9	1111 1100
253	FD	DP				12-11-0-5-8-9	1111 1101
254	FE					12-11-0-6-8-9	1111 1110
255	FF			EO		12-11-0-7-8-9	1111 1111

APPENDIX

G

SUMMARY OF ASSEMBLER DECLARATIVES

Here is a list of the assembler data types for defining DC and DS declaratives.

Type	*Format*	*Implied length*	*Maximum length*	*Alignment*	*Truncation/ padding*
A	address	4	4	word	left
B	binary digits	—	256	byte	left
C	character[1]	—	256	byte	right
D	floating-point — — long	8	8	doubleword	right
E	floating-point — — short	4	8	word	right
F	fixed-point binary	4	8	word	left
H	fixed-point binary	2	8	halfword	left
L	floating-point — — extended	16	16	doubleword	right
P	packed decimal	—	16	byte	left
Q	symbol naming a DXD or DSECT[2]	4	4	word	left
S	address in base/ displacement format	2	2	halfword	—
V	external defined address	4	4	word	left
X	hexadecimal digits[1]	—	256	byte	left
Y	address	2	2	halfword	left
Z	zoned decimal	—	16	byte	left

[1]For DS, C and X type declaratives may have a defined length up to 65,535. [2]Q-type declaratives are available only for F-level Assembler.

APPENDIX

H

SUMMARY OF ASSEMBLER DIRECTIVES

Here is a list of the various assembler directives in each general category. Directives marked with an asterisk (*) are available only under OS/VS or VM.

Program sectioning and linking

COM	Identify beginning of a common control section.
CSECT	Identify start or resumption of a control section.
CXD*	Cumulative length of an external dummy section.
DSECT	Identify start or resumption of a dummy control section.
DXD*	Define an external dummy section.
ENTRY	Identify an entry point, referenced in another assembly.
EXTRN	Identify an external symbol, defined in another assembly.
START	Define start of the first control section in a program.
WXTRN	Identify a weak external symbol (suppresses search of libraries).

Base register assignment

DROP	Discontinue use of a base register.
USING	Indicate sequence of base registers to use.

Listing control

EJECT	Start assembled listing on next page.
PRINT	Control assembled listing (operands are ON/OFF, GEN/NO-GEN, and DATA/NODATA).
SPACE	Space *n* lines in the assembled listing.
TITLE	Provide a title at the top of each page of listing.

Program control

CNOP	Conditional no-operation (see next section).
COPY	Copy code from an assembler source library.
END	Signal end of an assembly module.
EQU	Equate name or number to a symbol.
ICTL	Define the format of following source statements.
ISEQ	Start or end sequencing of source input statements.
LTORG	Begin the literal pool.
OPSYN*	Equate a name operation code with an operand op code.
ORG	Set the location counter.
POP*	Recover status of PRINT/USING directives saved by last PUSH.
PUNCH	Provide output on cards.
PUSH*	Save current PRINT/USING status.
REPRO	Reproduce the following card.

Macro definition

MACRO	Begin a macro definition.
MEND	Terminate a macro definition.
MEXIT	Exit from a macro definition.
MNOTE	Display a macro note.

Conditional assembly

ACTR	Set loop counter for conditional assembly.
AGO	Branch to sequence symbol.
AIF	Conditional branch to sequence symbol.
ANOP	Assembly no-operation.
GBLA	Define global SETA symbol.
GBLB	Define global SETB symbol.
GBLC	Define global SETC symbol.
LCLA	Define local SETA symbol.
LCLB	Define local SETB symbol.

LCLC	Define local SETC symbol.
SETA	Set an arithmetic variable symbol.
SETB	Set a binary variable symbol.
SETC	Set a character variable symbol.

Conditional No-Operation (CNOP)

The purpose of CNOP is to enable you to align instructions on integral boundaries. You would most likely use CNOP where you have defined local declaratives at the end of a subroutine and want to ensure that the first instruction for the next subroutine begins on an even boundary.

There are six variations on CNOP, depending on whether you want alignment based on fullword or doubleword boundaries. Operand 2 designates fullword (4) or doubleword (8) alignment. Operand 1 determines the particular location in the fullword or doubleword. To force the correct alignment, CNOP generates from one to three NOP instructions, each 2 bytes long.

Fullword alignment

`CNOP 0,4`	On fullword boundary
`CNOP 2,4`	On address aligned on halfword boundary in middle of aligned fullword

Doubleword alignment

`CNOP 0,8`	On doubleword boundary
`CNOP 2,8`	On second halfword immediately following doubleword boundary
`CNOP 4,8`	On fullword boundary in middle of aligned doubleword
`CNOP 6,8`	On fourth halfword boundary in aligned doubleword

A common requirement for alignment on a fullword boundary is simply CNOP 0,4. If the assembler location counter was at X'762', this CNOP would generate one NOP so that the following instruction begins at X'764'. Note, however, that if the location counter is at an odd-numbered address, the assembler forces normal alignment before processing the CNOP.

Relevant IBM reference manuals

GC33-4010 OS/VS-DOS/VS-VM/370 Assembler Language

GC24-3414 DOS Assembler Language

APPENDIX I

ANSWERS TO SELECTED PROBLEMS

Chapter 1

1-4. (a) 7; (c) 25.
1-5. (a) 110; (c) 10010.
1-6. (a) A; (c) 12; (e) 20.
1-7. (a) B; (c) 12; (e) 1A.
1-11. (a) 64 × 1,024 = 65,536.
1-18. (a) binary = 1111 0101; hex = F5.
1-19. (a) 370 F3F7F0 11110011 11110111 11110000
(c) Sam E28194 11100010 10000001 10010100 (lowercase)
1-21. (a) PAT D7C1E3 11010111 11000001 11100011

Chapter 2

2-1. (a) A unit of data, such as employee number or rate of pay.
2-2. (a) The instruction that a computer executes.
2-5. (a) The program as written in symbolic language, prior to assembly.