ISAM FOR UTS

F. Haney
D. Heying
R. Sharpe

Distribution: D. Cota
R. Spinrad

AD-72-0868
November 10, 1972

I. SUMMARY

- The addition of ISAM file capability to UTS will require 12 man-months of effort, total, for design, implementation and checkout.
- The work will take six months.
- UTS is organized to permit clean addition of the ISAM file access method. Each access method is implemented by a self-contained content manager. Several access methods have been added recently at a cost of a few man-months effort each.
- UTS ISAM will be equivalent to XOS ISAM in performance.
- UTS ISAM will provide all functional capabilities available in XOS ISAM.
- UTS ISAM will provide automatic increase in file size (extends), as required, a feature not available in XOS ISAM.

II. INTRODUCTION

The ISAM facility in UTS is a functional adjunct to the conventional multi-level keyed file capability which provides superior performance and requires less direct access storage for a special class of file manipulation. Current multi-level keyed files support all of the functional capabilities of ISAM (except for incidental differences involving the location of the key field and maximum key length). Enhanced performance is provided for sequential access of ISAM files by combining the "master-index" (of conventional keyed files) with the data. Thus, ISAM combines the space and sequential access speed advantages of consecutive files with the random access by keys of keyed files.

A new file organization, I for ISAM, is provided as a new modular (isolatable) facility in UTS. This new file organization may be accessed either sequentially or randomly (by key). ISAM files must be created sequentially (sorted-key order), but may be updated randomly (by key).

The KEYED files do not constrain keys to be within the data record and (currently) constrain key length to 31 bytes. The ISAM files require keys to be contained within the data record, but allow keys to be up to 255 bytes in length.

III. FUNCTIONAL DESCRIPTION OF ISAM FOR UTS

The ISAM file organization may be accessed via the standard file CALs M:READ, M:WRITE and M:DELREC. Extensions are provided in three distinct system areas:

- A modular file access routine for ISAM supplements the current set (individual modules currently manage consecutive, keyed, and random file organizations).
- The user control language is incidentally extended in batch (CCI) and on-line (TEL) to reflect the new organization (indexed) and the key length and key position within the data record.

- System procedures (for Meta Symbol) are expanded to include the new organization and the key length and key position within the data record.

Changes to the system procedures, CCI and TEL are only cosmetic additions to existing parameterized facilities and hence are essentially trivial. The only significant development is concentrated in the provision of a new, modular access routine.

The control language extensions to CCI (Batch) and TEL (on-line) are as follows:

```
TEL
      !SET dcb params;INDEXED;BLKL=lll;KEYP=mmm;KEYL=nnn
CCI
      !ASSIGN dcb params;INDEXED,(BLKL,lll),(KEYP,mmm),(KEYL,nnn)
```

INDEXED is a new organization and supplements the current set which are CONSECutive, KEYED and RANDOM.

The value associated with KEYP indicates the key position relative to byte 0 of the data record, while the value associated with KEYL indicates the key length from 1 to 255 bytes. The value of BLKL indicates block length* in bytes and will be rounded up to an integral number of 2048 byte pages for system use.

The system procedures (reflected in the Metasymbol SYSTEM BPM) are extended as follows:

```
M:DCB  dcb,(INDEXED),(KEYP,mmm),(KEYL,nnn),(BLKL,lll)
M:OPEN  dcb,(INDEXED),(KEYP,mmm),(KEYL,nnn),(BLKL,lll)
```

No cosmetic changes are required for the procedures:

M:READ	(read record)
M:WRITE	(write record)
M:DELREC	(delete record)

* The block length is assumed to be 2048 bytes (the current system standard) if unspecified. This value allows user blocking factor control up to 8192 bytes; logical records, however, can be as large as user memory as contrasted to the XOS limit of 32,767 bytes.

The KEYL (key length) value of INDEXED would be treated exactly as KEYM (key max. length) is now for KEYED files.

The index will automatically "prefer" RAD and the data will automatically "prefer" DISC, however, the space for each is allocated dynamically and no user file space specification is required for public files. No special utility is required for "reorganization" to eliminate overflow blocks; a standard PCL (peripheral control language) COPY will move an INDEXED file to any device, including OVER itself, to effect reorganization. Also, a standard LIMIT card specification for FPOOL (file pool) and IPOOL (index pool) will provide (and allow) space to hold high-level index blocks in core and minimize redundant (wasteful) reloading of index blocks.

The ISAM file space may be (optionally) preallocated with a suitable specification of the RSTORE parameter (available via SET, ASSIGN, M:DCB and M:OPEN).

IV. PERFORMANCE

There are several dimensions of performance which can be examined relative to ISAM in UTS. First the data structures in ISAM data blocks are approximately equivalent to those in UTS consecutive files. Thus, the read/write CPU time for UTS consecutive files (about 1.7ms) is an upper bound on the CPU time required per read/write in ISAM files.

The second major factor to consider in performance is elapsed time due to I/O accesses. There are several subfactors to consider here. First, the No-wait I/O capability that exists in UTS will be available to allow users to overlap CPU and compute time. To consider other factors, a tabular list of differences between Keyed and ISAM files will be illustrative.

Feature	Advantage ISAM	Advantage Keyed
1. Variable (Large) Block Sizes vs. fixed Block Size	Sequential Processing Requires fewer accesses to secondary storage due to larger block sizes.	Random Processing requires less channel and program wait time due to smaller amount of extraneous data transferred. Less main memory required to process file.
2. Pre-Allocation vs. Dynamic Allocation	Data kept proximate thus requiring less arm movement; File Deleting is faster.	Wasted Secondary Space kept to minimum.
3. Key in data vs. Keys separate from data	Fewer I/O accesses for sequential processing. Less storage occupied if Key is naturally part of data.	Restructuring required much less frequently. Files may be created with Keys in any order. Easier to update with better representation of updates in the index structure.

The above advantages in favor of ISAM will be provided by UTS ISAM. The number of accesses will be reduced to the same as XOS ISAM for sequential processing because data blocks will be variable sized and no index need be consulted. The arm motion will be the same for UTS ISAM as for XOS ISAM if pre-allocation is requested. If pre-allocation is not requested, groups of several thousand bytes will be allocated, keeping the data relatively close together while retaining the space efficiency of dynamic allocation. In addition, any UTS ISAM file can be increased in size without explicit request. Deleting files will be as fast in UTS ISAM as in XOS ISAM as a concise record is kept of all space in the file.

V. DESIGN APPROACH

The UTS file management system provides a convenient vehicle for addition of new access methods. Much the same approach would be used for adding ISAM as was used in adding the Random, Consecutive, and ANS tape access methods. This approach is represented graphically in Figure V.I.

Figure V.I shows that user programs request system services via CAL's which are uniform across access methods. CAL decoding transfers control to the appropriate monitor routine. If the request is an open or close request the common catalog is consulted and the data set is prepared for accessing. If the request is an access request (read, write, position, etc.), the appropriate content manager is called to do any access-method-dependent processing. Each of the content managers will call IOQ for any physical data transfers required. They will also call the File Space Allocator for any needs to allocate secondary storage.

ISAM, an independent module, would be added as a content manager in the same manner as the above mentioned content managers were added. A Content Manager Environment Simulator is available to assist in the development of the above mentioned content managers. This simulator, which was used in development of consecutive AM and ANS AM, provides the interfaces depicted in the chart (namely CAL decodes interface, physical I/O interface, and Allocation interface). Thus, new content managers can be debugged as user programs from a terminal and then integrated into the system as a working entity.

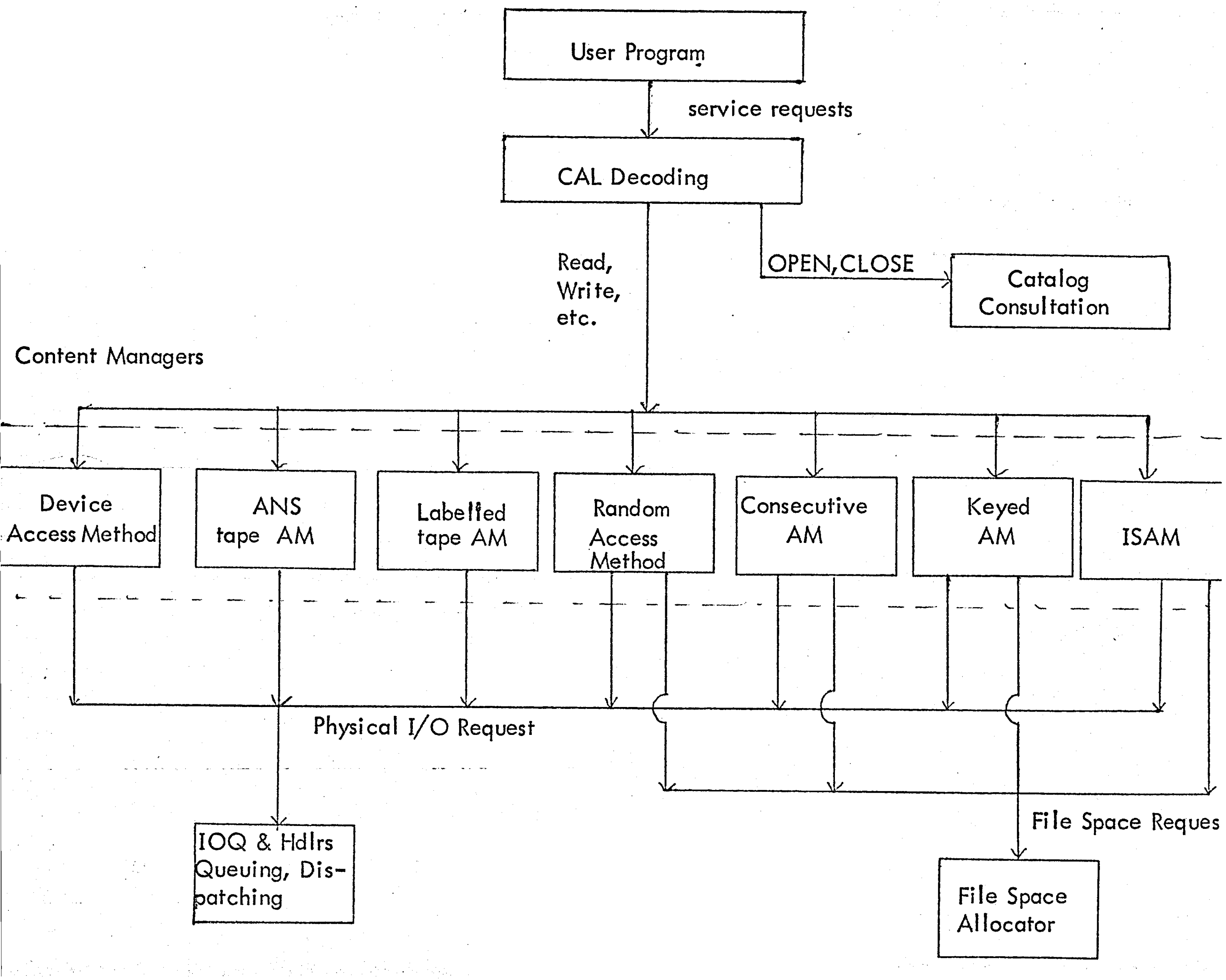
User Program
service requests
CAL Decoding
Read, Write, etc.
OPEN, CLOSE
Catalog Consultation
Content Managers
Device Access Method
ANS tape AM
Labelled tape AM
Random Access Method
Consecutive AM
Keyed AM
ISAM
Physical I/O Request
File Space Reques
IOQ & Hdlrs Queuing, Dispatching
File Space Allocator

FIGURE V.I

The ISAM structure would be built as outlined in Appendix B. Block sizes would be variable with secondary storage space used determined by rounding block sizes to the next multiple of 2048 bytes. The complexity of the structure is estimated as approximately 30-40% greater than that of UTS consecutive files. Thus, the amount of new code required for the content manager is estimated at approximately 900 words. Sequential processing of ISAM files will not require any accesses for index blocks.

Neither IOQ nor the File Space Allocator would require any modification as they are sufficiently general to handle the requirements of ISAM. CAL decoding and Catalog routines would be modified to recognize the new access method.

ISAM would be added to UTS as an option available to programs whose data are a good fit to the ISAM structure - i.e., they can avail themselves of the potential performance gains and can tolerate the concomitant decrease in flexibility. Keyed files would be retained for their greater flexibility and function and (in some cases) superior performance

VI. <u>COSTS</u>

These cost estimates are developed by considering the size and complexity of the tasks involved and by comparing this development with similar ones which have been completed recently.

The major development areas are:

1) The ISAM access method itself
2) The automatic recovery interface to assure no loss of updates in the event of a crash
3) Changes to the JCL decoding in CCI and TEL to provide for new parameters

Changes are <u>not required</u> to file backup and PCL utility processors since organization information is accurately carried on file to tape and tape to file operations.

Costs are:

Design, development, debugging, and technical documentation	- 12mm
Availability time after start of project including integration with a new system version	- 6 months

These costs do not include standard overhead burden.

Similar projects completed in the last two years which are of similar complexity and scope are:

	development	elapsed
Random files in E00 BPM	3mm	4 months
True consecutive files in C00 UTS	4mm	7 months
ANS Tape facilities including both the access method and label validation	24mm	10 months

APPENDICES

The following two appendices show:

- Functional Description of UTS Keyed files as per the UTS Reference Manual.
- Functional Description of XOS ISAM as per the XOS Reference Manual.

The keyed file description is presented here merely to indicate the similarity to ISAM and to show that ISAM contains no major function or architectural aspect that is not presently supported in UTS.

APPENDIX A

UTS KEYED FILES

APPENDIX D. FILE ORGANIZATION

A file is an organized collection of information that may only be created, modified, or deleted through the Monitor system. A file has one base name but may have other names synonymous with it.

Information is retrieved from a file by specifying the file name, password, account, and the desired record within the file.

The Monitor maintains a directory of accounts that have files which are maintained between jobs. This is called an Account Directory, and contains, with each account number, an address of a directory of files (termed a File Directory) for that account. A File Directory contains, with each file name, an address of a table containing file attributes and disc locations for that file. The table is called a File Information Table. To summarize, the Monitor has a single Account Directory, which in turn points to a File Directory for each account. Each File Directory, in turn, points to a File Information Table (FIT) for each file.

Each file has associated with it (in the FIT) information controlling who may access it and how it may be accessed. A password and a list of which accounts may read or update the file is recorded. Protection from unauthorized disclosure is attained by checking the information carried with the file against the information supplied by the user.

Changes to the file are allowed or disallowed based on the user's password and account. No accidental changes can occur.

A file may be shared among several users providing that none of them updates the file or attempts to replace the file.

A job cannot create a file in an account other than its own.

FILE ORGANIZATION

KEYED FILES

Keyed files are those in which each record has an identifying key associated with it. A key consists of a byte string, the first byte of which states the number of bytes in the string. The contents of each byte may be a binary number or a character.

As the file is being created, a master index is also created with an entry for each keyed record in the file. The entry contains such information as the key, disc address of the record, size of the record, and position of the record within the blocking buffer.

The records are automatically packed into blocking buffers with the last portion of the last record extending into another buffer as necessary. If the record is large, it is written directly from the user's area instead of being packed into a buffer. Keyed files may be accessed by direct or sequential access.

CONSECUTIVE FILES

Consecutive files are files whose records are organized in a consecutive manner; i.e., the user is aware of no identifying keys associated with the records. The records may only be accessed sequentially.

As with Keyed files, a master index is created along with the file. The master index contains information similar to that for keyed files. The key will be a three-byte 'dummy' key, created by the Monitor, but transparent to the user. As each new record is created in a consecutive file, the Monitor binarily increments the last dummy key to obtain a new dummy key.

The records in consecutive files are blocked identically to keyed files.

MULTI-LEVEL INDEX STRUCTURES

A multi-level index structure is a collection of hierarchical levels of index blocks, where the entries in a higher level point to index blocks at the next lower level and the entries in the lowest level (called level 0) point to data records. This is best illustrated by an example as shown in Figure D-1.

Both keyed and consecutive files have level 0 index blocks. Only keyed files can have a multi-level index structure. The multi-level structure is initially built during a CLOSE if a keyed file has more than three level 0 index blocks.

In the example shown in Figure D-1, the keyed file has

- 15,570 records and the keys at level 0 point to these data records. Based on an 11-byte maximum key length, there are 40 keys in each level 0 block and 127 keys in each higher-level block.

- 390 index blocks at level 0, four index blocks at level 1, and one index block at level 2. The next higher-level is built if the last level has more than three index blocks.

Each entry in a higher-level index block contains the disc address of an index block at the next lower level, and the key of the first key in that block.

The multi-level index structure can considerably speed up the direct access of a large keyed file, at only a small cost of secondary storage space. Since the keys are ordered in ascending sequence, at most it would take three index block accesses to locate a data record as shown in the example. Without the higher-level index structure, it could take up to 390 index block accesses.

The user has no control over the initial creation of the multi-level index structure but he can specify when and if the higher-level structure should be rebuilt. This can

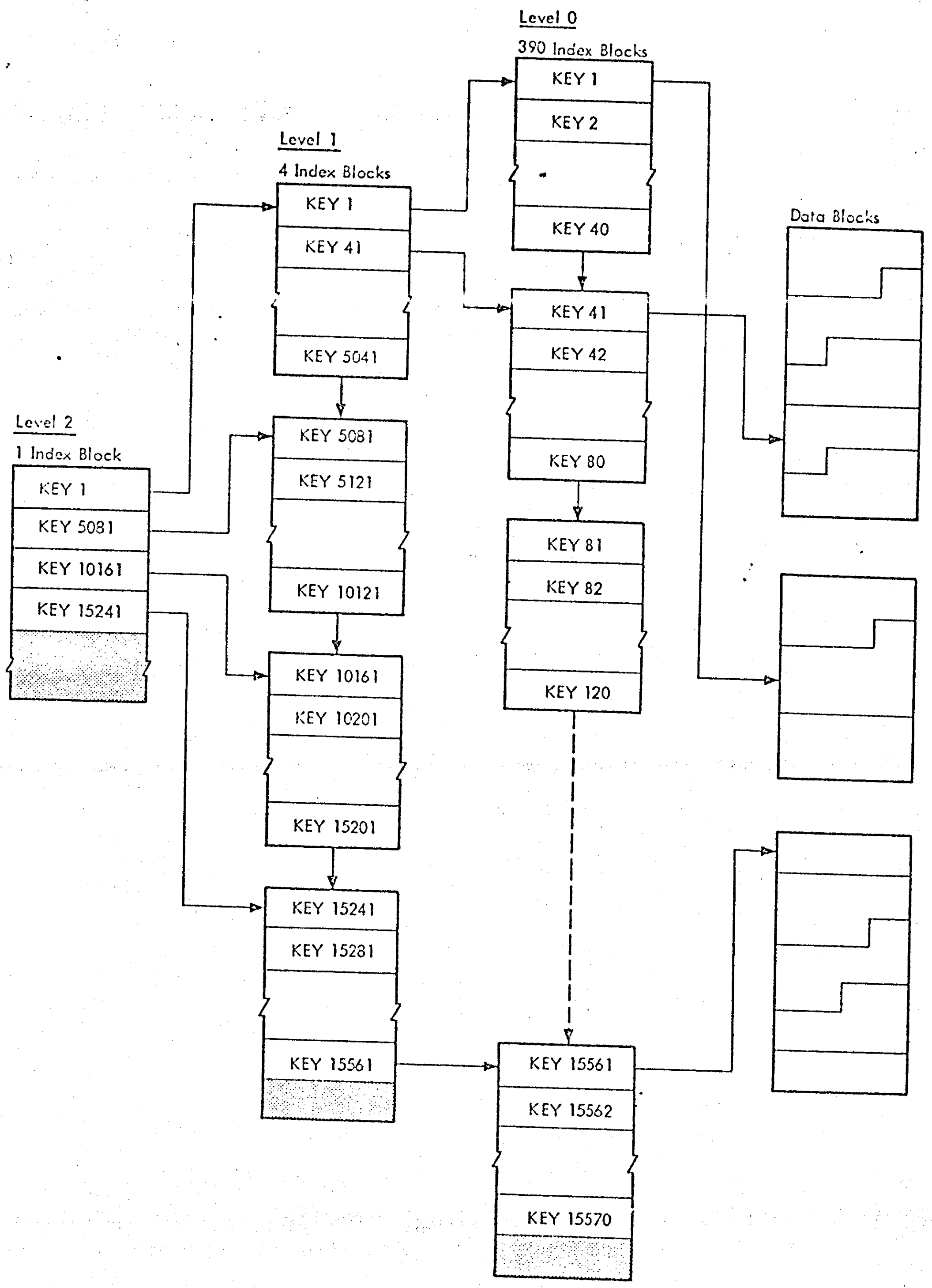

Figure D-1. Example of Multi-Level Index Structure specified by using the NEWX option on the !ASSIGN control command or the M:OPEN and M:DCB procedures.

The space required to hold a given file can be estimated by applying the following rules:

DATA BLOCKS

1. Each data block contains 2048 bytes.
2. Each data granule contains one data block.
3. Each data block is compact, except that all records start on word boundaries.
4. Each record or record segment (if a record resides in more than one data block) has a level 0 index entry associated with it.

LEVEL 0 INDEX BLOCKS

1. Each index block contains 1024 bytes.
2. Each index granule contains two index blocks.
3. Each index block is compact except that 12 bytes are preempted and spare space may be reserved at user request.
4. Each index entry occupies KEYM plus 14 bytes.

HIGHER-LEVEL INDEX BLOCKS

1. Each higher-level index block contains 2048 bytes.
2. Each higher-level index granule contains one higher-level index block.
3. Each higher-level index block is compact except that 12 bytes are reserved.
4. Each higher-level index entry occupies KEYM plus five bytes.

The following two examples show the cost to build the multi-level index structure, i.e., disc accesses to build it and disc storage required to contain it, and the saving in time when accessing it.

Example 1

Number of records	=	40,000	
Record size	=	60 bytes	
Key size (KEYM)	=	3 bytes	
Spare space	=	.1	
Data granules	=	$\frac{(40,000)(60)}{2048}$	= 1172
Keys/Level 0 Index block	=	$\frac{(1024-12-102)}{17}$	= 53
(1024 x .1 = 102; 14 + 3 = 17 KEYM)			
Level 0 Index blocks	=	$\frac{40,000}{53}$	= 758
Level 0 Index granules	=	379 (RAD or disc)	
(758 ÷ 2 = 379)			
Level 1 Index blocks	=	$\frac{758}{(2048-12)/8}$	= 3
(KEYM + 5 = 8)			
Level 1 Index granules	=	3	

This file requires a total of 1554 granules of storage of which three are required to store the multi-level index. It would cost 761 disc accesses to build the structure when the file is closed. With the multi-level structure, each random record fetch requires 3-2/3 device accesses, whereas without it each fetch would be 254 accesses.

Example 2

Number of data records	=	maximum for each device (see below).	
Record size	=	1024 bytes	
Key size (KEYM)	=	15 bytes	
Spare space	=	0	
Keys/Index block	=	$\frac{(1024-12)}{29}$	= 34
Keys/higher-level Index block	=	$\frac{(2048-12)}{20}$	= 101

Item	7232 RAD	7242 Disk Packs
Number of data records.	6144	24000
Level 0 granules.	91	353
Level 1 granules.	2	7
Level 2 granules.		1

The cost to build the multi-level structure in the 7242 example is 714 device accesses. Without the multi-level structure a random fetch could take 707 device accesses in the worst case; with it, four accesses are required.

The reader can easily see that the cost of storing the multi-level index structure is trivial and the one time cost to build it can be insignificant for a large file which will be read or updated frequently.

The following refinements of the build and rebuild logic have been made to better accommodate on-line system usage.

1. If the higher-level structure is being built on-line, only the first three level 1 blocks will be built, thus limiting the amount of time taken to build the higher-level structure on-line.
2. The higher-level structure will never be rebuilt on-line.
3. When a record is accessed in a file for which only a partial higher-level occurs, and the record is beyond the end of the current higher-level, then level 1 is extended as the search for the desired key takes place.

RECORD BLOCKING

The system will automatically block records for keyed and consecutive files in 512-word blocks to provide more efficient use of disc space. The user has no knowledge of this blocking and, when reading, will receive the appropriate record within the block and not the entire block.

When updating a keyed file, the user may rewrite a record in a size larger or smaller than the original record size. If necessary, the Monitor will allocate additional disc space to accommodate the larger size.

A write with a 0 byte count will result in a master index entry for the record with fields in the entry pertaining to disc address, record size, and displacement into the blocking buffer all set to zero.

RANDOM FILES

Random files provide an organization for those users desiring to manage their own files or who do not wish to incur the overhead imposed by system file management. Random organization differs from keyed and consecutive organization as follows:

1. A Random file is simply a collection of contiguous granules on the specified device type. The number of granules is specified at the time the file is opened (and may not be expanded after it has been opened). If the requested number of granules are not available contiguously, an abnormal code (major code X'01', subcode X'0B') is returned to the user and the file is not opened.
2. The user must specify a relative starting granule number with each read or write and a byte count (the default byte count in the DCB may be used). If the starting granule number does not fall between 0 and the total number of granules allocated at "OPEN" -1, inclusive, an error code of X'42' is then returned to the user. If the byte count exceeds granule size, the operation will continue in the next contiguous granule(s) until all requested bytes have been transferred. The system will return the next available relative granule number to the user (in the KBUF field of the DCB) at the completion of each read/write. If there are not sufficient granules to accommodate the specified byte count, an error code (major code X'57', subcode X'44') is returned to the user and the actual number of bytes transmitted is placed in the RWS and ARS fields of the DCB.
3. Each write/read consumes the entire specified gr The contents of the granule includes no system in tion. Management of the user's data is the respons bility of that user.
4. Function has the following meaning for Random fil when any random file is opened it is first checked fo existence.
   - If the file does not exist and Function is IN or INOUT, an abnormal code of X'03' is given. I the file does not exist and OUT or OUTIN is specified, a new Random file is allocated unless the associated account number differs from the use account number (in this case, the file will not be opened and an abnormal code of X'14' will be returned).
   - If the file does exist, the user is checked for app priate access permission (read/write account n bers, password), and an abnormal code X'14' is returned if there is a violation. If there is n violation, the user may proceed to read (unless opened OUT) or write (unless opened IN). If file is opened OUT or OUTIN, the function is changed to INOUT. Note that the user may write in a granule in which he has already written, and may also read a granule in which he has not wri

Thus, the Monitor provides allocation of granules, security checks and normal I/O queuing service and clean up. T user is responsible for record management.

FILE ACCESS

DIRECT ACCESS

Direct access may be used only on files with keyed organization.

OUTPUT FILES

When a WRITE is given, a key must be specified. The keys do not need to be given in a sorted order. They will be ordered as they are stored on disc.

Unlike sequential output files, a WRITE never causes forward information to be deleted.

Reading is not allowed.

SCRATCH FILES

A scratch file is identical to an output file, except that reading is permitted before the file is closed. As for output files, a key must be specified on each Write. The keyed record is merged into the file.

A Read may or may not specify a key. If a key is specified, a search is made of the file until the key is found and the record is then read. If the key is not

APPENDIX B

XOS ISAM FILE

the case of direct-access media the actual physical disposition of information is determined solely by the system and thus is transparent to the user. (The use of the basic direct access method, BDAM, implies no file structure or organization whatever, and can only be used with private or nonstandard disk packs.)

The four possible file organizations, and the media to which they apply, are

- Sequential (C) – all media.
- Indexed-Sequential (I) – direct-access only.
- Partitioned (P) – direct-access only.
- Direct (D) – direct-access only.

Although, in general, each file organization corresponds to a particular access method for file creation, several access methods may apply for subsequent access to a file of given organization. For example, a partitioned file can be read by the assisted sequential, assisted partitioned, virtual sequential, and virtual direct access methods.

SEQUENTIAL (C) ORGANIZATION

The sequential file organization permits sequential access to the records or blocks of a file. It is created by either ASAM or VSAM, and is the only organization applicable to nonmagnetic device files as well as to files on magnetic media.

Depending upon file media restrictions, any of the three record formats, F, V, or U, are allowed with use of the assisted sequential access method. Although a sequential file can be written or read at the logical-record level by ASAM, it can be read (or written) only at the block level by VSAM.

Existing sequential files on magnetic tape are always extendable – at the cost of losing any subsequent files on the same volume. On direct-access media, they may be extended up to the limits of the possible space allocation; also individual records may be deleted, or modified if the record length is not changed.

INDEXED-SEQUENTIAL (I) ORGANIZATION

The indexed-sequential file organization permits either direct access to individual logical records identified by record key, or sequential access to records in ascending order of their keys, starting with a specified record. A record key is a data item within the record body, provided by the user, which serves to uniquely identify the record. The location of a record specified by key is determined (by the system) via an index mechanism that is constructed and maintained by the system as part of the file.

Indexed-sequential organization is applicable only to direct-access media. Either F- or V-format records are allowed. An indexed-sequential file is created using the assisted indexed access method (AIAM).

The indexed-sequential organization is shown schematically in Figure 6-9. The file is composed of data blocks, index blocks, and (possibly) overflow blocks. Upon creation, the file will consist of one or more data blocks, and at least one index block. The index may be multilevel, as illustrated in Figure 6-9 (1st and 2nd level index blocks). The number of index blocks and number of levels thereof is a function of block size, number of data blocks, and record-key length (as described below).

Prior to file creation, the user must request allocation of sufficient file space to allow for all of the data, index, and overflow blocks that may eventually be needed. The method for calculating this space requirement is described below under "Space Allocation". Also prior to creation, he must describe to the system both the beginning byte position, relative to byte 0 of the record body, and the length of the record key by means of the DCB parameters KYP and KYL respectively.

During file creation, the user must create the record keys and write the logical records in ascending record-key order (binary collating sequence); if a record is presented out of ascending key order it is not accepted and an abnormal condition occurs.

For each base data block written, a <u>record-index entry</u> is automatically created. It is composed of the record key corresponding to that of the last record in the data block and a pointer to the beginning of that block. The record-index entries are blocked as are user records, and the set of these blocks constitute the <u>first-level</u> index.

For each first-level index block written an <u>index entry</u> is created. It is composed of the record key corresponding to that of the last record-index entry in the first-level index block and a pointer to the beginning of that block. These index entries are blocked, similarly, into the second-level index.

Given enough data blocks, the above process applies recursively with third, fourth, ..., 255th level indices produced. In general, at least one (partial) index block exists at any level when two or more blocks exist at the next lower level – including the "data block level".

Overflow data blocks are created if, during subsequent updating, either inserted or lengthened records cause original records to be "pushed down" beyond the boundary of a data block. The resulting overflow is automatically moved to an overflow block which is linked between the two data blocks as shown in Figure 6-9. Two or more overflow blocks can be linked between two data blocks in this manner. Note that overflow blocks do not appear explicitly in the index, and are undesirable from the viewpoint of access speed and storage space utilization. A utility processor, REORGI, is provided to effect a reorganization of overflowed indexed sequential files. (See the XDS Utilities Reference Manual.

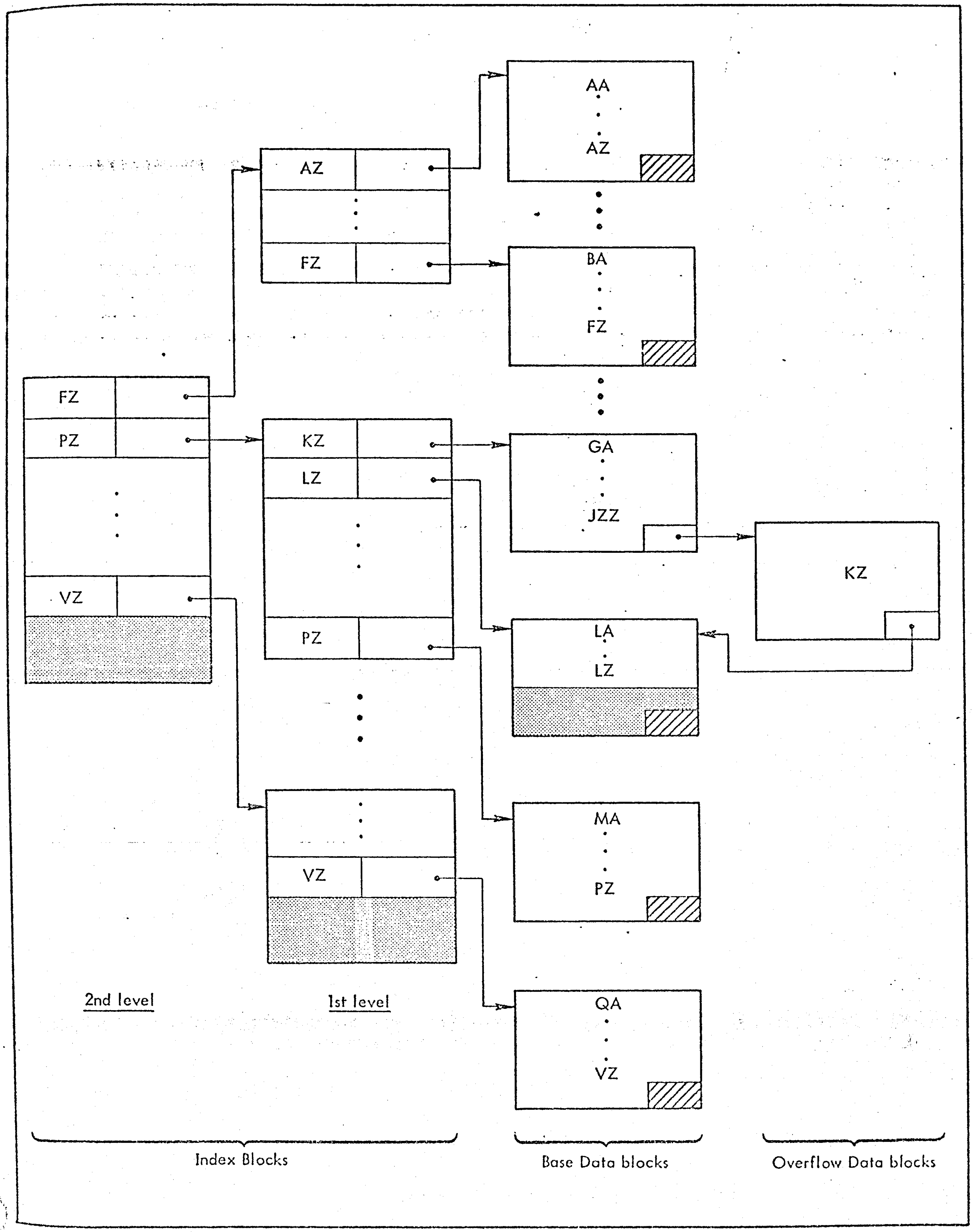

Figure 6-9. Indexed-Sequential Organization

At the end of the file creation process, the system automatically inserts a dummy record having the maximum possible key value (X'FF...F'). This permits subsequent insertion of records with keys greater than that of the last record originally written, effectively allowing file extension. The dummy last record cannot be accessed by the user program, however.

Care must be taken that the key field does not overlap the deletion control character (byte 0), if the latter is specified; a program abort on file opening will occur if KYP = 0 (default value 0) in this case.

If the ICY (index copy) option of the M:OPEN procedure is specified, the system will automatically copy the index portion of an existing file to a temporary file in secondary storage. This will generally result in faster direct-access processing time, especially if the secondary-storage media is appreciably faster than the media upon which the entire file resides, e.g., RAD vs. disk pack.

PARTITIONED (P) ORGANIZATION

The partitioned organization permits either sequential access to the records of a file, or direct positioning to the beginning of a named partition of a file for subsequent sequential access to the records thereof. This organization is essentially an arrangement of a sequential file into uniquely locatable subfiles.

A partitioned file is created with the assisted partitioned access method (APAM). It is applicable only to direct-access media. Either F or V record format may be utilized.

The partitioned organization is shown schematically in Figure 6-10. Note that the user's data blocks are preceded by, and possibly interspersed with, system-constructed partition key (name) and a pointer to the first logical record in the associated partition. The directory blocks are, however, transparent to the user's program as they are not accessible via assisted access methods. For example, if either APAM or ASAM is used to read a partitioned file, they will "skip over" the directory blocks. However, the user must, prior to file creation, request allocation of sufficient file space to allow for both data and directory blocks. The method for calculating and specifying this space requirement is described below under "Space Allocation".

To create a partitioned file, the user must begin by assigning a partition key (with the M:STOW procedure); that is, the file must contain at least one partition. During creation, the user may create as many partitions as desired. In addition to principal (i.e., first-assigned) partition keys, the user may assign synonym keys, i.e., aliases of a given partition name. The keys may be up to 255 bytes in length.

During subsequent processing of the file, synonym keys may be added, any key may be deleted, and new partitions created. In addition, existing records may be delete or be modified if the record length is not changed.

It is important to note that when reading a partitioned file (either with APAM or ASAM), the system does not detect an end-of-partition condition: the user may read to end-of-file, across partition boundaries, whether starting from a partition boundary or from beginning-of-file. A partitio key locates the beginning of a partition, but not the end of the preceding one: therefore a partitioned file may be given a hierarchical, or "nested", structure by the appropriate ordering of subsumed partitions. (End-of-partition may, of course, be signaled by a user datum detected by the program, e.g., a zero-length record in V-format.)

The pointer portion of a directory entry contains the relative block number of the block in which the associated partition begins, and the byte displacement of that partition's first record. (Synonym entries contain, in addition, a synonym indicator.) The partition keys are sorted, when necessary, and maintained in ascending order of key value within the directory block chain.

DIRECT (D) ORGANIZATION

The direct organization permits direct access to blocks of a file by relative block number (in relation to the beginning of the file, block 0), via the VDAM access method only. It is an "unmanaged" organization relative to the C, I, and P organizations.

A direct-organization file is composed of blocks of BKL defined (or default 1024-byte) length, and transmission must begin on a block boundary. However, the length of the data actually transmitted is specified in the M:READ or M:WRITE procedure by the transfer-length (TRL) option (default = 1 block). The length of data transmitted may be less than the block length, or may extend over several contiguous blocks, but it is limited by the maximum-transfer-length (MXL) parameter of the DCB associated with the file.

No block header is created in D organization; no logical record structure within the block is recognized by VDAM.

Files created by VDAM are accessable by VSAM and also by ASAM using U record format.

TEMPORARY AND PERMANENT FILES

Files on magnetic media can be either temporary or permanent files. (The distinction is not relevant for nonmagnetic device files, for a number of reasons.) In principle, a permanent file is one that continues to exist in a retrievable form after the execution of the job that creates it; a temporary file does not. That is, a permanent for file reference, only the STS option – with OLD or MOD specified – need be added.

SPACE ALLOCATION

Space is allocated for the creation of a new disk or RAD file according to the specified or default values of the SIZ parameter of the !ASSIGN control command (or of the M:ASSIGN procedure, if used). The syntax of the SIZ parameter is described in Chapter 3.

The meaning and effect of the SIZ parameter values vary according to the organization of the file to be created. They are described for each organization in the following subsections.

Note that no new space allocation can be made for a disk/RAD file that is to be rewritten, i.e., a file replacing an existing identically-named one will occupy the same space allocated to the original file. (Status OLD; Output processing mode.)

CONSECUTIVE ORGANIZATION

Value1 of the SIZ parameter specifies, in quanta of 8K bytes, the initial amount of space to be allocated to the new file. Value2 specifies the size of the increments to be added to the file in case of either overflow of the initial allocation during creation, or extension of the file during subsequent status-MOD, Output-mode processing.

If all of the space specified by value1 is not available on the first of a series of volumes specified, the remainder will be allocated on succeeding volume(s).

INDEXED-SEQUENTIAL ORGANIZATION

For an indexed-sequential file:

value1 specifies, in quanta, the space to be allocated for base data blocks.

value2 specifies in quanta, the space to be allocated for (1) the creation of index blocks, and (2) overflow blocks.

Note that unlike C and P organization files, the entire and final amount of file space – including possible overflow space – is allocated at file creation time; no subsequent extensions are allowed.

If value2 is omitted or given a zero value, the system reserves index-block space as if all of the file were to be allocated for base data blocks (less index space), and does not allow any space for overflow.

METHOD OF CALCULATING THE NUMBER OF INDEX BLOCKS REQUIRED

Since the indexed-sequential organization is relatively complex, a method is presented for the calculation of the number of index blocks that will be needed (assuming that the substantive file grows to full size), given a specified amount of space for base data blocks (value1 – value2 X 8,192 bytes).

Preliminary Definitions. BKL defines the block length of base data blocks, overflow data blocks, and index blocks, in bytes. Since blocks of an indexed-sequential file, regardless of record format, contain a 4-byte block header and a 4-byte linkage field, the usable block size, b, is defined as follows:

$$b = BKL - 8$$

KYL defines the record key length, in bytes.

Both BKL and KYL are specified in the DCB corresponding to the file to be created.

Values to be computed:

N_0, the number of base data blocks.

E_x, the number of index entries per index block.

N_1, the number of first-level index blocks.

.

.

.

N_i, the number of i^{th} level index blocks.

Equations. Assuming that (value 1-value 2) > 0, the value N_0 is derived as follows:

$$N_0 = \text{integer portion of} \left[\frac{(\text{value 1-value 2})}{BKL}\right] \times 8192$$

Any index, entry is composed of a record key plus a 3-byte pointer to a late block. Therefore, the index-entry length, l, is given by

$$l = KYL + 3$$

and the number of entries per index block, E_x, by

$$E_x = \text{integer} \left[\frac{b}{l}\right]$$

The number of n^{th} level index blocks, N_n, is given by the number of base data blocks divided by the number of entries in an index block, the result being rounded upwards to an integer, i.e.,

$$N_1 = \text{integer}\left[\frac{N_o}{E_x}\right]$$

Similarly

$$N_2 = \text{integer}\left[\frac{N_1}{E_x}\right]$$

$$\vdots$$

$$N_i = \text{integer}\left[\frac{N_{i-1}}{E_x}\right]$$

Therefore, the total number of index blocks, N, that will be created is

$$N = \sum_i N_i$$

and the total amount of space that will be reserved therefore, is in bytes

N x BLK

The excess of value2 x 8,192 over the amount of space derived above will be available for overflow blocks. By making a preliminary estimate of value 1 and value 2, based on the prediction size of the data portion of the file, and then performing the calculations described above, the values of value 1 and value 2 may be adjusted to produce the most efficient allocation.

Caution: Since blocks on direct-access media always being on a sector boundary and take up as many full sectors as are required to accommodate the block length, the user must carefully relate BKL and the sector size of the device involved in order not to waste direct-access media space. Specifically, if BKL is not equal to of a multiple of sector size, the equation given above for deriving N_o, and thus the entire calculation, is invalidated.

PARTITIONED ORGANIZATION

Value1 of the SIZ parameter specifies, in quanta, the initial amount of space to be allocated to the new file. Value2 specifies the size of the increments to be added to the file in case of either overflow during creation, or extension during subsequent status-MOD, Output-mode processing.

To arrive at appropriate SIZ parameter values for a partitioned file the user should note that

1. The directory entries and the data records are kept in separate blocks and that in both cases the effective block length is BKL-8.
2. To compute the number of partition key entries a block can contain, E_d, one must consider that the length of each entry is, in bytes

   KYL + 5

   Thus

$$E_d = \text{integer}\left[\frac{b}{KYL + 5}\right]$$

The probable maximum number of partition keys, both principal and synonym, to be stowed is therefore a factor in estimating the best allocation.

DIRECT ORGANIZATION

Value1 is the total amount of space, in quanta, to be allocated to the direct-organization file. Value2 is not significant for this organization; i.e., a direct-organization file is not extendable beyond its creation-time allocation.

RULE FOR ALLOCATION OF MULTIVOLUME DIRECT-ACCESS FILES

For all files necessitating parallel mounting, i.e., multivolume direct-access files (I, P, or D organization), the amount of space specified by value1 of the SIZ parameter must be available exclusively on the volumes specified for mounting or the allocation request will be refused.

PASSWORD PROTECTION

When creating a file that is to be password protected, or when accessing a file that has been password protected, an X1-class abnormal return occurs at OPEN time. The abnormal return routine must detect abnormal code X'19' and must then load registers 6 and 7 with a value representing the password before executing an M:RETURN. If the file is being created, the password is entered into the HDR3 label. If an existing file is not being accessed, then the value is compared with the file's password in the HDR3 label; and if the values are identical, processing continues normally. When the passwords do not match, the job step is aborted.

At file creation time, the user informs the system that a password is to be applied to the file by means of the PAS option of the PRT (protection) field of the !ASSIGN command.