

March 1969

THE DESIGN AND IMPLEMENTATION OF A DISC SOFTWARE SYSTEM*

by

Peter H. Lipman

Artificial Intelligence Group

Technical Note No. 5

SRI Project 146561 w.o. 037

*Submitted to the Department of Computer Science of Stanford University in partial fulfillment of the requirements for the master's degree.

CONTENTS

I	INITIAL CONFIGURATION.	1
	A. Input-Output Software System.	1
	B. Disc Hardware	2
	C. Controller Operation.	2
	D. Other Controller Functions.	3
II	LOW-LEVEL SOFTWARE DESIGN DECISIONS.	4
	A. Design Goals.	4
	B. Disc Allocation	4
	C. Disc Assignment Tables.	6
III	DISC EXECUTIVE SOFTWARE.	9
	A. User File Directory and Command Recognition	9
	B. Getting Access to Disc Files.	10
	C. Local vs. Global Directories.	11
	D. Size Limitation of Local File Directory	13
	E. Other Features of the Disc Exec	14
IV	GENERAL COMMENTS ON THE DESIGN OF THE DISC SOFTWARE SYSTEM .	15
	A. Disc Hardware Features that Were Not Used	15
	B. Conclusion.	16
	<u>Appendix</u> --DESCRIPTION OF VARIOUS TABLES AND FORMATS	18

I INITIAL CONFIGURATION

This paper describes in detail the design and implementation of the disc software system for the Artificial Intelligence Group's SDS 940 time-sharing system at Stanford Research Institute. The purpose of this project was to change the version of the SDS 940 time-sharing system being used from one which kept permanent user files on magnetic tape, a painfully slow file storage and access technique, to one which kept user files on disc. The purpose of the project described here was to design, implement, and interface new disc software to the existing time-sharing system, rather than to write a free-standing disc system. This section describes the original software and hardware foundation upon which the disc software system was built.

A. Input-Output Software System

The new disc software had to be interfaced to the input-output (I/Ø) portion of the existing time-sharing system. Let us examine some of the features of that system.

The user does I/Ø operations to "files" which he "opens" by handing a "file name" to the appropriate system routine. He calls on one routine to open a file for input, and another to open one for output. The file name carries with it the information as to what I/Ø device is being referenced. Bracketing characters such as / and ' are used to surround the file name, and these denote to the system what device the information is to be sent to or gotten from. When a user opens a file, he gets back a unique number, which he then references when he does I/Ø operations. At this point, from the user's point of view, all files are the same. He can transfer one word to or from the file by the WIØ (word input-output) system call, or he can transfer a whole block of data by specifying a word count, starting core address, and calling the BIØ (block input-output) system routine.

The most relevant feature of this I/Ø system is that once the file is open, the same system routines are called to perform I/Ø operations on the file and to close it. From the user's point of view, even the opening routines are the same; he just hands them different file names.

B. Disc Hardware

The disc software design is partially determined by the characteristics of the Bryant disc and controller hardware (model 4061). The disc has an array of heads that can move over 256 discrete positions called tracks. The present configuration has 7 physical platters and is expandable to 12 platters. One of the 14 surfaces is used by Bryant for timing information, and the remaining 13 surfaces are available for data storage. The smallest addressable area is a 256-word sector. There are 44 such sectors on each surface for each track. The controller allows a transfer of from 1 to 2048 words and automatically increments the disc address to get the next sequential one when necessary. When writing the disc, the controller fills any incompleting sector with zeros.

C. Controller Operation

The controller is a mildly clever processor in its own right. It uses a fixed-core address as its "location counter" and has a repertoire of three "instructions." (Each of these instructions is assumed to take three consecutive computer words, so after execution of an instruction the "location counter" is bumped by three.) The three controller instructions are:

- (1) Transfer. The three words of the transfer instruction are interpreted as starting disc address, word count, and starting core address, along with some control information. This instruction causes the disc transfer to be performed as specified.
- (2) Disconnect. The disconnect command causes the controller to "halt."
- (3) Branch. The branch command causes the controller to accept the first word as a core address and to put it into its location counter.

(See Sec. A-2 of the Appendix for the formats of these instructions.) To start the controller in motion, you give it one of two kinds of go commands: go-chain or go-no-chain. Go-chain sets the controller

in motion, executing commands until it hits a disconnect instruction. Go-no-chain starts the controller in a mode such that it disconnects after the first transfer.

D. Other Controller Functions

The SDS-940 is a hardware-paged machine with 2048-word pages. The boundaries of these core pages must be treated properly by an I/O transfer. Thus, unless it is specifically told to, the disc controller will not allow a transfer to cross a page boundary. In the address word of the transfer instruction there is room to specify that a page boundary may be crossed and what the new page is. If this information is specified, the controller correctly handles the crossover; if not, it stops the transfer at the crossover point and issues an error interrupt.

A second function of the controller is a hardware-aided, software-controlled data protection scheme. Essentially, it allows the software to specify a 4-bit class code to be written along with the data for a given 256-word sector. This information is written in an area preceding the data area of the sector. If the appropriate control bit is turned on in the transfer command, then the class bits on the disc are compared with the class bits in the command, and if they don't compare, the transfer is aborted and an error interrupt is issued. If a write operation is specified and no compare is requested, then the class bits from the command are written into the class field.

One further function the controller performs is the sending of status information to the computer. For purposes of this communication, the controller has at its disposal one interrupt and two fixed-core locations. The software can specify whether or not it wants to receive an interrupt at the completion of a transfer. In addition, the interrupt is generated on all error conditions, and an error condition register is stored into one of the fixed-core locations with a specific bit on for each error condition that occurs. The remaining core location is used by the controller to send periodic information about the disc's next angular position and track. This angular position report is constantly being updated while the controller is running, and the information it gives

is sufficiently far enough ahead so that the software has time to perform a transfer into the specified sector without having to wait an extra revolution.

II LOW-LEVEL SOFTWARE DESIGN DECISIONS

The next two sections of this paper will describe in some detail the design and implementation decisions made in order to build the disc system. This section is concerned with the "lower-level" software written to interface the disc to the time-sharing monitor. Section III will deal with the "higher-level" software associated with a portion of the time-sharing system called the exec.

A. Design Goals

The following design goals were established at the beginning of the project:

- (1) From the user's point of view, a disc file should look the same as any other file. This means that the user should be able to hand the name of a disc file to the same system routine that opens tape files or drum files, and then he should be able to do the same device-independent I/O operations to that file.
- (2) All information on the disc had to be safeguarded as much as possible from software or hardware failure. For example, a system crash due to any kind of failure not directly associated with the disc should have absolutely no effect on the disc.
- (3) The system should be fully operational as soon as possible after the disc was. Closely associated with this aim was the desire to avoid massive alteration of the operational portions of the system.

B. Disc Allocation

In order to have disc files appear like all other files, the first task was to build disc software into what is called the sequential

file machinery--i.e., the software that handles the WIØ and BIØ system calls (formally "SYSPOPS," system-programmed operators). Part of the design of this section was straightforward, consisting of replication of the file structure used on the drum files. An index block technique was used, with one disc segment containing a list of disc addresses that point at the data segments of the file. The index block has forward and backward pointers to allow a fully general chaining of index blocks, with the result that there is no limitation on disc file size.

Though the structure of a sequential disc file required no particular thought, the disc allocation algorithm and some closely related routines offered some more complex design problems. One such problem was the development of a method for allocating disc that would not slow down the writing of disc files too disasterously. It was clear that the assignment algorithm was going to require a sufficiently large amount of code so that it was not going to fit into the permanently resident portion of the time-sharing monitor. This left the choice of expanding the resident monitor or putting the code in what is called nonresident monitor, a section of monitor that has to be swapped in when it is needed. The latter technique was by far the more palatable in my judgment, but clearly would add overhead to the running of the assignment algorithm. To further add to the overhead, it became clear that the information as to which disc segments were available to be allocated would itself have to be on the disc. It certainly could not be in resident monitor, and obviously the criterion that the disc be protected from system software or hardware crashes pretty much requires that the information be kept on the disc.

The above comments make one point very clear: assigning disc segments for a sequential disc output file is an expensive proposition. Even if the code could be resident, the disc accesses necessary to find out what disc areas are available would be time-consuming. It was concluded that the way to speed things up was to attempt to minimize the number of times the allocation routine was called on for a given output file by allocating a large number of segments at once, rather than just one, and then building into the close-file mechanism the ability to

deallocate the unused segments. If this could be done optimally, then exactly the right number of segments would be allocated the first time for the whole file, so that the assignment algorithm would be called upon only at open-file time.

Since how to allocate disc for a given output file is totally dependent on how long the file is going to be, the system has to require the user to specify this information. The user calls a higher-level (in the exec) system routine to open files, and that routine is the one that calls the device-oriented opening routine. To allow this executive routine the full flexibility to specify how a disc output file is to be allocated, a parameter is accepted which specifies how many disc segments are to be allocated each time more disc is needed and how many times the user is to be allowed to allocate that many segments. The use of this particular generalization will be discussed in Sec. III-E of the paper.

C. Disc Assignment Tables

One of the design decisions that had to be made was how to keep track of which areas of disc were in use and which were free. A common technique is to have a bit table with one bit for each segment of disc. Jack Armstrong, a systems programmer with whom I consulted throughout the project, objected to the use of bit tables on the grounds that they tended to be very inflexible. For instance, it might be desirable at some future point to keep more information about a disc segment than that it is free or allocated. With this objection in mind, I implemented a generalized bit table with the ability to have up to three bits of information for each disc segment. The number-of-bits-per-table entry is an assembly parameter and is presently set to two (see table in Sec. B of the Appendix).

A great deal of thought went into the organization of the assignment tables, both in terms of where they should be on the disc and how they should be organized internally. If this disc were a fixed head type instead of a movable one, then the question of where the assignment tables should be put would not be relevant. But with a movable head disc it is important to avoid changing tracks any more frequently than necessary.

Thus, for instance, the assignment algorithm was designed to allocate all of a file on the same track to the extent that it was possible. For the same reason it seemed logical to break up the assignment tables into separate tables for each track and keep them on that track. But the size of the assignment table for just one track was so small that it would have been too much of a waste to use the whole sector for it, so as a compromise two assignment tables were put into one sector covering that track and the next one.

Internally the assignment tables were designed to make it easy to assign a sector with a given rotational position. The organization of the disc creates 22 distinct rotational positions (see the description in Sec. A of the Appendix); therefore, each assignment table was divided into 22 subtables. Organized in this way, it was of little difficulty to set up an algorithm which, given the rotational position of one segment, could pick another segment with the next optimal rotational position. By allocating a file in this way, it becomes possible to implement a high-speed BIØ (block input-output transfer). This has not yet been done, but the tools are there and it is partially designed. The way the chaining hardware of the disc controller works, the transfer of n segments could be performed by setting up a list of n commands followed by a disconnect command and telling the controller to start in chained mode. In effect the n transfers may be treated as if they were one, and if the disc addresses are in optimal or nearly optimal relation to each other, then the transfer will be completed in the minimum number of disc revolutions.

Dividing the assignment tables into track-oriented tables and keeping them on or near their track created two basic problems. First of all, when first assigning disc for a file, it is necessary to pick a track to assign from. Since the number of segments to assign is known, it would be ideal to be able to pick a track that has that many segments free. To allow this, a special "room available" table in resident monitor with 3 bits' worth of information per track was designed. If the entry for a given track is zero, then there is no room on that track at all. Otherwise, the entry tells approximately what proportion of the total segments are left. One other small table was needed in resident monitor

for the purpose of keeping track of where on the disc the assignment tables are located and whether or not one is in use. This table has 6 bit entries for each 2 tracks with a 5-bit coded disc address, allowing the flexibility to move an assignment table if a given disc area should go down, and a busy bit signifying that the table is being changed and must not be touched by anyone else. (See Secs. C and D, Appendix, for complete details of these two tables.)

In consultation with Jack Armstrong, it was agreed to assemble into the system only one fixed disc address, and to keep all other information about the disc on the disc. So the concept of the "key table" was introduced and in that "key table" are kept all the disc addresses for the assignment tables (see Sec. G, Appendix). This allows a system-initialize routine to initialize its table of assignment table addresses, and then to initialize its room available table by reading in each assignment table. To facilitate this latter initialization, a counter is kept at the front of each assignment table which keeps count of how many disc segments are available on that track.

This counter is kept in the "preamble" to each assignment table. The preamble consists of three words, the first of which is the track to which the table refers. The second is the number of segments still available on the track, and the third is the "next random subtable." As described previously, each assignment table is broken down into 22 separate subtables, each of which contains all those disc addresses referring to a fixed rotational position. Normally the allocation algorithm requests a subtable based on what subtable the last segment was taken from, but at the beginning of the process it makes little difference what subtable is allocated from. It does make a difference in one respect, though, because it would be undesirable to run out of entries in subtables unevenly. This would tend to make the assigning by rotational position less effective. Ideally, subtables should be depleted evenly. Thus, the concept of the "next random subtable" was used, and when it is of no consequence what subtable is received, the next random one is requested. If this were to be done perfectly, it would probably be best to choose the subtable with the most segments left, but the technique of just incrementing the next

random subtable word and cycling it around seems to be quite adequate, and considerably easier.

III DISC EXECUTIVE SOFTWARE

The exec is essentially a privileged user program, shared by all users on the system, which serves as a command receiver and processor. It is this program that the user communicates with when, for instance, he wants to enter the system, or start up a subsystem such as FORTRAN or LISP. From the point of view of this paper, the relevant feature of the exec is that it is responsible for all file directory handling and the accessing of files by name.

A. User File Directory and Command Recognition

This section describes the exec file-handling system which had to be interfaced to. For each user on the system, the exec has a page (2048 words) which it uses to store information local to that user. A section of this TS (temporary storage) block is used for a file directory. A hash coding technique is used to transform the file name into a pointer to a table that contains the relevant information about the file. The table entry for each file contains three words (see Appendix, Sec. E), in which are stored pointers to the file name, type of information in the file (symbolic, binary, etc.), the medium on which the file is stored (magnetic tape, drum, etc.), where the file is on that medium, and other information depending on the medium.

The mechanisms the exec uses to enter names into the file directory and look them up are part of a general string manipulation package which resides in the resident monitor and is available to any user program. There are two separate string lookup mechanisms which return the pointer to the three-word entry associated with that string. One accepts the full string (the whole file name in the case of the exec wanting to look up a file name), and the other accepts one character at a time of the string and as soon as the string is uniquely defined types the rest of it. The latter lookup mechanism is called command recognition, and is used extensively in the system.

The user can specify to the exec which of the above lookup mechanisms to use when looking up a file name. He can pass the whole file name to the exec, or if the file is to be an input file, he can ask to have the name command recognized. In specifying the name of an output file, special bracketing characters are used to signify the medium on which the file is to be written. (They used to be ' for magnetic tape files and / for drum files; now they are ' for disc files, # for magnetic tape file, and / for drum files.) This technique is only necessary when the file name is that of a new file. If the user wishes to overwrite an old file, then the name is already in the directory and so is the medium on which the file is stored. A file name without bracket characters may be used. If it is the name of a new file, then it goes onto the default medium (used to be magnetic tape, now it is the disc). An unbracketed file name may be command recognized for output as well as input.

B. Getting Access to Disc Files

In designing the disc software extension to the exec, the technique of giving a user access to his tape files proved to be a useful model. When the user wants access to his files on a given tape, he issues a command to the exec, specifying a tape unit and a user name. The exec then reads the central file directory (a directory of all the files on the tape) off the front of the specified tape and makes a file directory entry for each file belonging to the specified user.

The same sort of technique seemed to make sense for disc files. The analogue to the tape central file directory would be a disc file directory kept on disc. Of course a separate directory for each user name was used instead of one huge directory. (The directory format is described in Appendix Sec. F.) When the user enters the system, he specifies a user name and also a charge number. The user name is mapped to a user number, and the number is used as an index into a table within the "key table" (see description in Appendix Sec. G), and the disc address of the first segment of the user's disc directory is extracted. This directory is read in and entries are made in the TS block directory for all those disc files that have the same charge number as the one specified. By making TS block file directory entries for disc files, all the old

mechanisms for opening files by name worked with very little alteration.

C. Local vs. Global Directories

In discussions with Jack Armstrong and two other system programmers, Chuck Kirkley and Bill Duvall, they expressed their dissatisfaction with a couple of restrictions that they felt were unnecessary. Their arguments were concerned with two separate issues which tended to get badly intertwined. This section concerns itself with the argument over local versus global file directories. The global directory approach requires that each time a disc file is accessed, the system look up the file name in the disc file directory on the disc and get the relevant information needed to open it. One obvious argument against this and for the local directory technique is that looking up the file is a considerably faster process in a local directory since no disc accesses are required. However, an argument for it is that with a global directory that everyone accesses you can have a neat way to access someone else's disc file. Clearly, looking up someone else's file is no more difficult than looking up your own. Secondly, the problems of several users' accessing the same file are neatly taken care of by a global directory. This is a real horror in a local directory system. If two users have local directory entries for the same file and one deletes the file, the system must somehow delete the file from the other user's local directory too. In a global directory, the entry goes away, and the next time someone tries to access it, the file just isn't there.

Despite the handicaps of a local directory, it was decided to stick with it, the major reason being to preserve command recognition of disc files. As noted previously, I wanted disc files to look like all other files to the user, and it seemed unacceptable to command recognize some and not others. Chuck Kirkley came up with an argument to change the command recognition technique so that local directory drum files and global directory disc files could be command recognized in the same way. As we mulled over the techniques, it became apparent that major overhaul of much of the file handling system would be required, and this was beyond the scope of the project.

In order to take care of the intricate problems associated with two or more users having local file directory entries for the same disc file, I set down the rule that a user may only have local file directory entries for disc files under his user name and bearing his charge number. Also, no single disc file may be shared in the sense that the same copy of the file is pointed to by two separate disc directory entries. That is, the only way the same file can possibly be in two local directories at once is if the two users are entered under the same user name and charge number. With this restriction, the technique for fixing up one user's local directory after the other user overwrites or deletes a file is fairly easy. Whenever a user does something to change a local directory entry for a disc file, the exec also changes the entry on the disc directory so that it is always up to date. After the exec does this, it scans a table that tells who is on under what charge number and turns on a warning bit for any other user with the same name and charge number as yours. Looking at the mechanism from the other viewpoint, every time the exec is asked to access a disc file for you, it checks to see if your warning bit has been set. If it has, it reads in the disc file directory and makes any necessary corrections to your local directory.

Certainly the worst restriction here is the limitation on sharing files across user names and even across charge numbers for the same user name. Part of this restriction has been lifted by allowing a read-only access to another user's file (with his permission) by specifying the user name surrounded by parentheses followed by the complete file name. This form of file accessing is effectively a global file directory technique. Allowing write access this way presents two serious difficulties. The first is philosophical. It isn't at all clear whether a person charging to one charge number should be allowed to accrue charges to another. This is exactly what would happen if, for instance, he were allowed to overwrite another user's file with one twice as long.

The other difficulty has to do with disc file charging. Overwriting a file is essentially writing a new file and deleting the old at close-file time. The delete mechanism returns the number of disc segments deleted. The close output file mechanism returns the number of disc

segments used by the file. It is these pieces of information that are going to be used to tell how much change in disc storage has occurred for a given user under a given charge number. This net change will be saved along with other charging information. This works well when the user overwrites files for his own charge number, but it falls apart if he overwrites a file under another charge. At this point no serious thought has been given to how this problem can be solved. But it is clear from the above two arguments that it would be quite reasonable to allow write access to files under another user name but with the same charge number.

The read-only access technique provided a very simple way to set up a library of programs available to everyone. The user refers to such a program by typing ()'FILE', and the routine that handles this form of access maps the null user name to the name for the library.

D. Size Limitation of Local File Directory

The second point that caused a great deal of discussion and dissatisfaction boiled down to the fact that the local file directory as implemented in the system was too small. It only holds about 48 files, and many of the bigger users are constantly running out of room. The solution is obvious, but will require a considerable amount of work to effect. Unfortunately, the file directory cannot be expanded without moving it from the TS block. There is just not enough room left in this area to increase the size of the file directory by any reasonable amount. The solution is to move the file directory out of the TS block to an area of the drum. This way it could be of indefinite length. One drawback is that there would be increased overhead in accessing the local file directory since it would be out on the drum.

Expansion of the local directory is probably inevitable, but it wasn't done in the initial work for the disc exec. To alleviate the problem of a user having more files on his disc directory than will fit in the local directory, a scheme of group codes for disc files was designed. This is simply a way to partition the disc file directory. Each file can be given one or more group codes and then a group code number can be specified by a command and the files with that number are

brought into the local directory. This scheme is a nice generalized way of segmenting a large number of files into logical groups. I feel that the whole concept of group codes would be valid even if the local directories had been expanded, and I am convinced that when this expansion does finally take place, group codes will still be a useful tool. I should add at this point that I have already made initial plans to implement group code mnemonic names to replace the numbers. This is an obvious extension which the users will no doubt prefer.

E. Other Features of the Disc Exec

As noted earlier, one of the main functions of the exec is to provide the ability to access files by name. When a user opens a file for output in this system, he is supposed to specify an approximation of how long a file he will write. Because this is often impossible to know at file-open time, subsystems like FORTRAN II have never correctly opened new tape files, but have been able to write drum files because the length parameter is ignored. I decided to use the length parameter correctly with disc files and as a result I will eventually have to clean up various subsystems which do not specify length. The correct way to do this, it seems to me, is to allow the user to specify the length if he knows it, and if he doesn't, allow him to request that it be as long as possible. It actually proved useful to allow a couple of default lengths for disc files, the first specified by 0 allowing a file with the same maximum length as drum files (roughly 37K), the other specified by -1 allowing some mammoth size file. This parameter has not yet been implemented, and the maximum size hasn't been chosen. One other length specification was suggested by Jack Armstrong and is a nice extension allowing the user to specify directly how he wants the file assigned. He specifies a length with the sign bit on, the number of segments to be allocated each time through the allocation algorithm in the low 10 bits, and the maximum number of times he is allowed to go through the allocation algorithm in the next 10 bits.

This last option shows the format that is passed to the monitor routine specifying precisely how the file is to be allocated. If a real length is given, then it is converted into this format, but only if the

total number of segments exceeds the number of segments on a track is more than one entry into the assignment algorithm specified. If the user specifies either 0 or -1, then a request will be passed on to the monitor to allocate only a few segments at a time, but to allow an appropriately large number of times to reenter the allocation algorithm. The point of only allocating a small number of segments at a time on an unknown length file is that it is not reasonable to tie up a large portion of disc unless there is a reasonable probability that it will be used.

Aside from providing the user with the ability to access disc files by name, the disc exec code is also responsible for all operations on the user's disc file directory. In keeping with the philosophy that the disc should be independent of the rest of the system, the disc directory is always kept up to date. One feature that I built into the disc directory is worthy of comment. Every time a file is written, the data and time are recorded in the disc directory and a special bit is turned on, signifying that the file needs to be backed up. This backup bit was designed to be used by a privileged user program which scans disc directories looking for files with that bit on, copies the file to tape, and shuts the bit off. There would be no particular difficulty in having the system periodically start up such a program and thus automatically back up disc files, but for the present I plan to operate the program manually. I envision running full disc dumps (in a file-by-file format) on a weekly basis with dumps of only those files that were changed or added once a day.

IV GENERAL COMMENTS ON THE DESIGN OF THE DISC SOFTWARE SYSTEM

A. Disc Hardware Features that Were Not Used

In the initial design phase of this project a great deal of thought was given to the hardware features of the disc and controller. I spent quite a bit of time talking with Jack Armstrong about how to use two of the more unusual features, the class code protection scheme and the angular position report.

As described earlier (Sec. I-D), the class protection hardware essentially allows the software to write out a protection code with each 256-word sector. Then, if a certain bit is on in the disc transfer

command, the controller will require that the class code on the disc match the class code specified in the command. If such a comparison is requested and the class fields do not compare, the transfer does not take place. This facility of the disc hardware just does not seem to fit into a time-sharing application. In this system at least, users never deal with disc addresses directly. It is always the system itself which specifies a physical disc address. With this level of protection already built into the file handling scheme, the class protection technique seemed redundant.

The angular position report (APR) hardware seems, at first glance, to provide a very nice ability. It gives the system the ability to know a millisecond or so ahead of time what disc sectors it can transfer into with minimum latency. Unfortunately, it was necessary to allocate disc ahead of time--that is, before the user has said to write on it. Dynamically allocating disc at write time would be very nice, but I cannot see doing it on this system. One place that the APR hardware could be used effectively would be in emptying a queue of disc requests in optimal order instead of the order in which the requests were queued. This has not been done for two reasons. First, it is not at all clear that a time-sharing environment like ours can keep the disc busy enough to ever have more than one or two requests waiting on the disc queue at any given time. Secondly, reordering transfer requests is nontrivial. You must be very wary to honor a write request out of a given core area before honoring the read request, originally queued behind it, into that same core area.

B. Conclusion

In this concluding section I will examine the extent to which this system meets the three main criteria discussed earlier (Sec. II-A). The first criterion was that disc files should look the same as other files to the user. I feel that this was accomplished quite satisfactorily. Disc files are a completely natural extension to the system and can be operated on by the user in exactly the ways he is used to. The only problem at this point is that several subsystems such as FORTRAN II need to have a minor alteration made to them so that they specify a

length parameter when opening output files.

The second criterion was that the disc should be as self-sufficient as possible in the sense that system crashes should not leave the disc in bad shape. By taking care to keep the disc assignment tables and file directories constantly up to date on the disc, this has also been accomplished fairly successfully. Some extra care was taken in the operation of overwriting old files to further crash-protect the user and the disc. Overwriting a disc file is actually the two distinct operations of writing a new file and deleting the old. The delete and the updating of the disc file directory are all done at close-file time, so that if the system crashes in the middle of writing an old file, the old copy is still there. I took a great deal of time trying to organize critical operations so that if they were crashed in process, the least possible harm would be done. The most serious error from the point of view of disc allocation, for instance, is that a segment be marked free when in fact it is in use by someone. This would result in someone else having the segment assigned to him, and the second person would then overwrite some of the first person's data. To avoid this, the delete operation, for instance, marks the file deleted in the local and disc file directories before it actually deallocates the disc segments. Thus, the worst that happens if a crash occurs is that some disc will be allocated but not used by anyone. This is easily remedied. The same situation will exist if a disc file was being written, since disc will have been allocated, but directory entries are not made until close-file time.

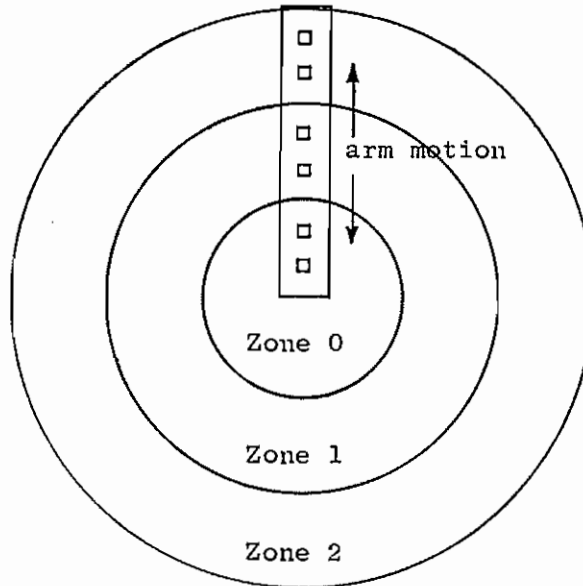
The final criterion I set for myself was that the disc software system should be operational as shortly as possible after the disc was ready. The system is in use at this writing and the disc has not been accepted yet. There are many improvements to be made--for instance, the high-speed BIØ which I discussed needs to be implemented. It is typical of such a systems development project, I think, that there will always be improvements and new features that need to be written. But I feel justified in saying that the disc system that is up and running right now is a very reasonable system to use from the user's standpoint and certainly a vastly better one than the tape-oriented version which was run for so long.

Appendix

DESCRIPTION OF VARIOUS TABLES AND FORMATS

A. Disc and Controller Hardware

1. Disc Layout

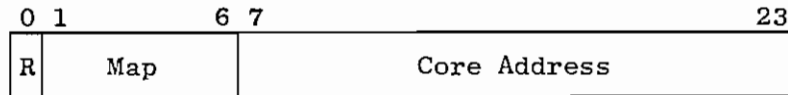
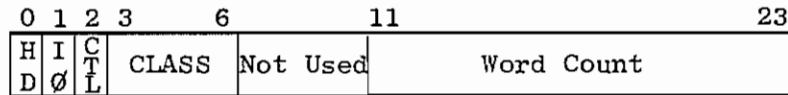
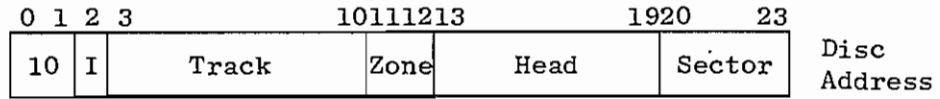


A disc surface is divided into 3 zones which are written at different densities. There are 2 heads in each zone. All heads are attached to an arm which moves in and out along the radius of the disc and has 256 discrete positions called tracks. The heads do not move with respect to each other. Zone 0 has 3 256-word sectors around. Zone 1 has 8, and Zone 2 has 11. This gives a total of 22 rotational positions on the disc.

Throughout the system disc addresses are broken down into track, zone, head, and sector, where the "head" is actually a combination of physical head and surface. With 13 surfaces and 2 heads in a zone, the "head" takes on values between 0 and 25. This proves to be a useful notation since the number of such "heads" (26) is the number of sectors on a track with the same rotational position.

2. Disc Controller Instruction Format

a. Perform Disc Transfer



I = Interrupt on completion if on.

HD = Write headers if on. Header words in front of each sector have track and sector addresses in them. These are written only during maintenance.

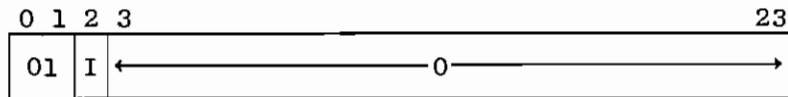
IØ = Read if off, write if on.

CTL = Check class field against class written on disc (Sec. I-D).

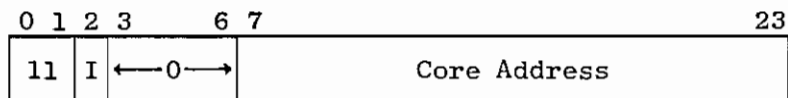
R = Enable crossing page boundary.

MAP = High-order bits to be used when page boundary is crossed.

b. Halt Instruction

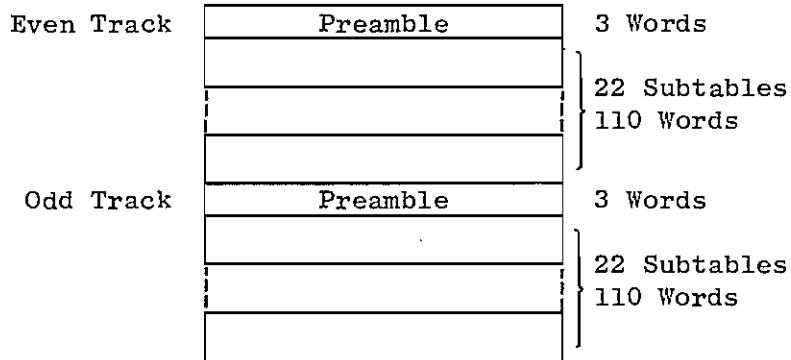


c. Branch Instruction



B. Disc Assignment Tables

1. Format of the Table for Two Tracks



2. Preamble

The preamble consists of a 3-word header for each track-oriented table (see Sec. II-C):

Track
No. of Free Segments
Next Random Subtable

3. Subtables

Each subtable is 5 words long and contains 2 bit entries for each of the 26 disc segments in that rotational position on the given track. The rest of the room in the 5 words is reserved for expansion to the full complement of 46 segments (23 surfaces x 2 heads per zone; see Sec. A in Appendix). There is also room to expand to 3 bit entries, but not quite to full complement (maximum of 40 entries). To allow expansion of the assignment tables beyond 20 surfaces using 3 bit entries, it would be necessary to put a single assignment table in a sector instead of putting two tables together.

4. Entry Values

- 0 = segment not in use
- 1 = segment in use
- 2 = this code not used now
- 3 = segment physically bad (surface defect).

C. Disc Room Available Time

This is a 32-word resident monitor table containing a 3-bit entry per track which gives a rough estimate of how much room is left in the assignment table.

- 0 \Rightarrow no free segments left
- 1 \Rightarrow 0 < free \leq 1/7 total
- 2 \Rightarrow 1/7 < free \leq 2/7 total
- ⋮
- 7 \Rightarrow 6/7 < free \leq 7/7 total.

D. Disc Assignment Table Pointer Table (DISATP)

This is a 32-word resident monitor table containing a 6-bit entry for each 2 tracks. This table provides a coded disc address for the sector which holds the 2 assignment tables and also a busy bit to signify that the assignment tables associated with that entry are being changed and must not be accessed.

BSY	Trk	Zn	HD	Sec
-----	-----	----	----	-----

The disc address of the segment on which the assignment tables are stored is calculated as follows:

$$\text{TRACK} \leftarrow \text{TRK1} + \text{TRK}$$

where TRK1 is the position of the entry
in the table times 2

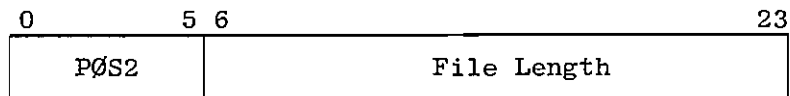
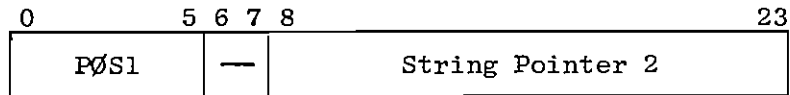
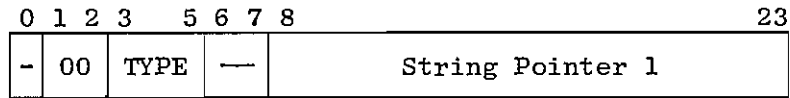
$$\text{ZONE} \leftarrow \text{ZN} + 1$$

$$\text{HEAD} \leftarrow \text{HD}$$

$$\text{SECTOR} \leftarrow \text{SEC} .$$

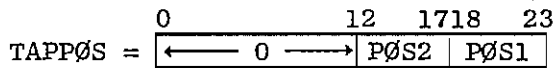
E. TS Block File Directory Formats

1. Tape File

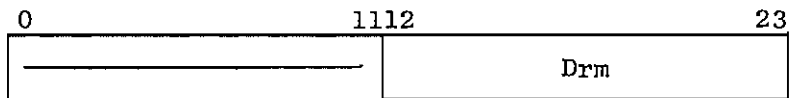
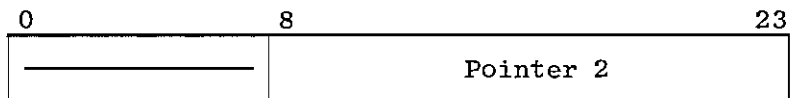
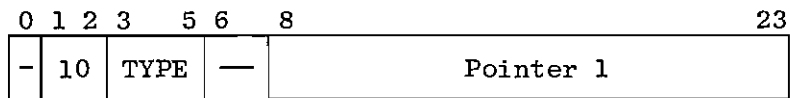


TYPE = type of information in file

PØS1 and PØS2 form the file's tape position



2. Drum File



Drm = drum address of first index block right shifted 2 bits.

0	3	4		23
Charge Number				

0	4	5	8	9	10	11	12	14	15	23
		NGRP	R Ø	P R I	-	TYPE		End Name		

File Name Terminated by 136B Character

Group Code Words

DEL = file deleted if on

BU = file needs to be backed up if on

Yr = year value added to 69

NGRP = number of words of group code following file name

RØ = file read-only if on (not implemented)

PRI = file private if on (may not be read by others)

End Name = pointer to first word after file name.

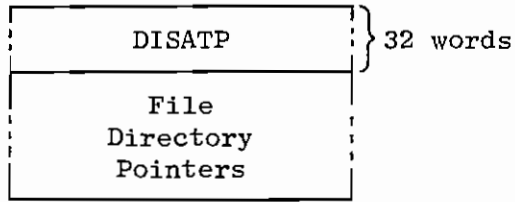
3. Group Code Format

A group code is 6 bits; 4 group codes are packed in each group code word. The code 77B is the null code used to fill partially full group code words. Valid group codes are 0 through 62. A file may have a maximum of 60 group codes (maximum of 15 words of group codes). If NGRP = 0, then the file has group code 0.

Group codes are totally transparent to those users who do not need them.

G. Key Table

This is the only disc address assembled into the system. Its contents are essentially all the other disc addresses the system needs.



DISATP is exactly the disc assignment table pointer table which is put into resident monitor. (Sec. D, Appendix).

The file directory pointer table is a table indexed by user number (mapped from user name) containing the disc address of the first segment of the disc file directory for that user. If the entry has the sign bit on, then there is no directory for that user.