B U G S

Brown University Graphics System

Doctor Memory[1]

An Eidetic Memory System

Russell W. Burns
John Stockenberg
Andries van Dam

The Brown University Graphics Project

Division of Applied Mathematics

Box F

Brown University

Providence, Rhode Island 02912

September 15, 1976

---

[1]This research is being supported by the National Science Foundation Grant GJ-28401X, the Office of Naval Research, Contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics; Principal Investigator Andries van Dam.

TABLE OF CONTENTS

1 Introduction.....2
 1.1 Why Eidetic Memory?.....2
 1.2 Why Not?.....2
 1.3 The Implementation.....3
 1.4 Facilities Overview.....4

2 Doctor Memory User interface.....6
 2.1 Error Handling.....6
 2.2 File Creation.....6
 2.3 Connecting a File to Doctor Memory.....7
 2.4 Creating an Area.....8
 2.5 Retrieving an area.....9
 2.6 Pointer Manipulation.....10
 2.7 Releasing an area.....10
 2.8 Disconnecting a File.....11

3 Utility Programs.....12
 3.1 DOCTOR: File Maintainence Utility.....12
  3.1.1 Compress.....12
  3.1.2 Extend.....12
  3.1.3 Copy.....12
  3.1.4 Unload.....13
  3.1.5 LOAD.....13
 3.2 FILEBUG: Doctor Memory File Debugger.....13
  3.2.1 ORIGIN <fileid> ['SEGMENT'].....14
  3.2.2 DISPLAY <key> [<offset> [<length>]].....14
  3.2.3 STORE <key> <offset> <data> [...].....14
  3.2.4 GETAREA <key> <length> [<fill byte>].....14
  3.2.5 DELETE <key>.....15
  3.2.6 LOGIN <fileid> <filename> <filetype>.....15
  3.2.7 LOGOUT [<fileid>].....15
  3.2.8 ROLLOUT [<fileid> [<segment number>]].....15
  3.2.9 GO.....15
  3.2.10 END.....16

4 Doctor Memory Disk Organization.....17

5 Doctor Memory Core Organization.....21

6 Doctor Memory and Programs.....23

Abstract

Doctor Memory is a software package designed to operate on the Meta 4A as part of the Level 0 extended machine. Doctor Memory provides access to disk records in an associative manner, through the use of extended instructions. This operation is a primitive sort of virtual memory, suitable to small scale computor systems. A knowledge of the Meta 4 Level 0 Extended Machine is assumed.

Keywords: Data Structure, Memory Management.

1 INTRODUCTION

1.1 WHY EIDETIC MEMORY?

Doctor Memory is intended for a user who has complex data structures which he wishes to maintain for multiple runs and which cannot reasonably be expressed in a sequential form, or which cannot be squeezed into core storage.

Unfortunately most meaningful graphics applications fall into one of these categories. The data involved in the pictoral representation of, for instance, a piping system is radically different from the data needed for a stress analysis of that system. The idea of variable length records and a higher level of symbolic access to data is geared towards this user. The envisioned applications will involve large data bases with a high degree of interrelation among elements of varying lengths.

1.2 WHY NOT?

The 1444 disk on the BUGS configuration is designed for sequential access to 640 byte records. Any variation on that size requires blocking, unblocking, worrying about sector boundaries and head position, and other hair splitting. The preliminary disk management imbedded in the GMS operating system for BUGS is designed for handling sequential and random access to fixed length records. Mapping from a logical record address to a sector/offset address is a simple arithmetic function. Care still must be taken in splitting records across sector boundaries, as 2 sectors must be accessed for a split record. If records are not split, space is wasted, and the formula must be ammended to account for this waste.

The Read and Write Buffer routines of GMS perform this blocking and deblocking of records, while a read/write sector routine provides disk syncronization, and LEVEL0 provides the actual seeking, reading and writing.

This system maintains one buffer per active file, providing only one sector buffering and leaving no provision for buffering according to usage or buffer sharing among files. Since record identifiers are directly related to their

sequential order in the file, squeezing out unused records destroys the integrity of any intrafile pointers which identify subsequent records.

Garbage collection in a sequentially addressed file, i.e. a file in which the "address" of a record is dependent upon its relative position in the file, is neccessarily application dependent. Any physical movement of a record potentially invalidates "pointers" in other records. In order to correct these references, a detailed knowledge of the data structures involved is neccessary. Freearea management in a file with variable length records is a book keeping nightmare which should be available as a system function rather than a continual re-invention of the wheel by each application programmer.

The disk access currently available is used largely for program storage. Data files are kept in a sequential order, since the most common access is serial. Most programs must build and modify their data structures in core.

1.3 THE IMPLEMENTATION

These considerations lead to a re-examination of the method of bulk storage access and memory management. A "virtual" memory system seemed to be the most sensible approach, but the neccessity of accessing large address mapping tables for each core access would add unreasonable execution and system overhead, and the 16 bit wide address space made the gains of such an addressing scheme minimal.

The solution to this problem was to add a further refinement to the existing address space. This space is accessed in an associative manner, with each address specifying an area, a variable length logical unit, rather than a particular location in memory. Each area is identified by a 16 bit key. The high order 5 bits of this key specify the file which the user wishes to access, and the low order 11 bits specify the area within that file.

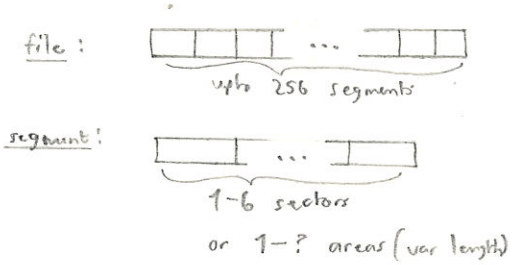
A user may identify one or more of his general purpose registers to be pointers into this virtual address space. The address mapping tables are accessed when the user issues one of the Doctor Memory extended instuctions, the specified area is fetched from disk if neccessary, and the register is initialized to point to the start of that area. If the data management system determines that the real address of that area must be changed, the user's register(s) will be updated

to reflect that change. Addressing within an area works according to the normal base/displacement scheme. [The user may modify his base register to point to anywhere within the area, and the system will maintain that address.]?

1.4 FACILITIES OVERVIEW

For the purposes of this paper a file is defined as a collection of data identified by an external name (e.g. WORKFILE DATA). A segment is defined as a physical unit of the file which is read or written at the same time, i.e. some integral multiple of a disk sector. There are up to 256 segments in a file. An area is defined as a logical unit to the user. Areas are physically independent, although they may contain links to other areas. An area is isomorphic to a record in the sense of a datum retrieved from bulk storage by an external name, but it has an area of core associated with it. This area actually is the record, rather than being a copy of it. The maximum size of an area is equal to the segment size. The minimum size of an area is defined by a function of the segment size.

The abstraction of the area identifier allows intrafile pointers to be independent of the relative position of the areas. Garbage collection can be made an automatic incremental function by allowing areas to migrate within and between segments in the file. The 16 bit key allows interfile pointers to retain their integrity through garbage collection. If two files contain links to one another, these links will still point to the same areas, even though their physical position on the disk or their relative position within the files may have changed.



This system has basically eight user interface points, which are implemented as supervisor calls (LOGIN and LOGOUT[2]) and extended instuctions:

Name	Function
LOGIN	Identify a file to be managed by Dcctor Memory. It establishes a symbolic identifier for the file, and a mode of access.
GETAREA	This function allocates a new area. The key may be specified by the user, or generated by the filing system. In any case, the length of the area is specified in the calling sequence. The address and rounded length of the area and the associated key are returned to the caller.
RETRIEVE	The caller wishes to access the existing area identified by the given key. The address and length of the area are returned to the caller.
COPYBASE	The caller wishes to point an additional register at the specified area.
MOVEBASE	The caller wishes to point another register at the specified area and discontinue use of the original register.
NOT IMPLEMENTED LOCK	The user requires that the specified area must remain in its current location. It may be neither moved nor rolled back to secondary storage.
RELEASE	The area specified by the caller is no longer needed by the caller and may be returned to bulk storage or deleted.
LOGOUT	The caller no longer wishes to access the specified file.

---

[2]These functions are available to the user at the command level as well.

2 DOCTOR MEMORY USER INTERFACE

2.1 ERROR HANDLING

Doctor Memory is designed with the "friendly" user in mind. Program checks are issued only in the case of "malformed" instructions. A register specification interrupt will be issued if a register to be modified by the system is the MSR or the SFP. Protection violation interrupts will be issued upon attempts to modify a readonly file. An addressing exception will be issued if the user attempts to access a file identifier which is not logged in. Other errors will cause the condition code to be set so the user can initiate his own error recovery.

If a register which is currently connected to an area is specified as a target for a new address, the old area will be freed automatically before the new area is accessed. All registers connected by a procedure are implicitly freed when a RET instruction is encountered.

The LOGOUT supervisor call insures that all areas in the specified file are copied to disk. If a user program terminates abnormally the user may issue the LOGOUT command to insure the integrity of his data.

2.2 FILE CREATION

To create a file, the user should use the LOGIN command. The syntax for the LOGIN command for allocation is as follows:

LOGIN <fileid> <filename> <filetype> (<options>

Where <fileid> is a number between 0 and 31 , specifying the "address" at which the file is to be attached to Doctor Memory. The <filetype> operand may be omitted if the user wishes to conform to the convention of naming the type according to the mode of access. See the discussion of the <mode> option in the section below on "Connecting a File to Doctor Memory".

The <options> field is optional if the filename is fully specified. The options in general control the attributes and access of the file. These options are specified in keyword form (shortest acceptable name underscored):

MODE=U, R, W, or P, for Utility, Readonly, Read/Write, and Program.

SSIZE= segment size, in sectors (number from 1 to 6).

FSIZE= file size, in segments (number from 1 to 256).

RKEY= protection key for read access to file (number from 0 to 15).

WKEY= protection key for write access to file (number from 0 to 15).

CORE= number between 2 and 256 specifying the number of 648 byte buffers to be used by DOCTOR MEMORY.

2.3 CONNECTING A FILE TO DOCTOR MEMORY

Connecting a file to Doctor Memory may be accomplished through the use of the LOGIN command or through the MLOGIN supervisor call. The LOGIN command allows greater flexibility, but MLOGIN is a resident routine with a smaller overhead. The syntax of the LOGIN command is the same for connecting a file as creating it, with the exception that the SSIZE and FSIZE operands are ignored. If the <filetype> is omitted, the type will default according to the mode as follows:

MODE=U, <filetype>:=UTIL

MODE=R or W, <filetype>:=DATA

MODE=P, <filetype>:=DMOD

If the user wishes to connect to a utility file, he may omit the filename and specify merely "LOGIN <fileid> (MODE=U", and optionally the size parameters. The LOGIN command will search for the smallest free utility file which satisfies the space requirements.

The MLOGIN supervisor call is intended for use by routines which need to access specific files. It is passed a parameter list, (see figure 1), specifying the file name and type, the mode of access and the protect key(s). The "address"

at which the file is to be connected may either be specified by the caller or assigned by the system (if the user specifies an address of X'FF'). The available modes are X'00' for utility, X'10' for readonly, X'20' for read/write and X'30' for program. If utility is specified, the real mode of the file will be returned to the user in the mode field of his parameter list.

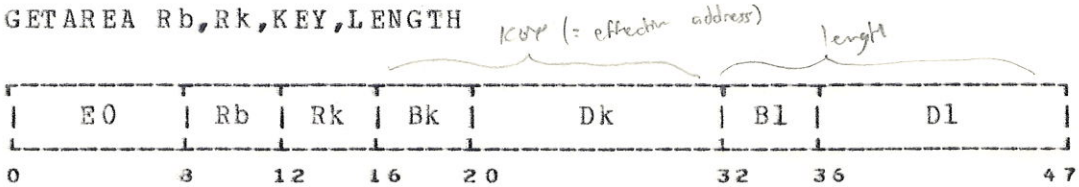
```
*****
*****     LOGIN SUPERVISOR CALL PARAMETER LIST
*****
*****
LOGPARM   DSECT     ,                    DUMMY SECTION FOR PARMS
LOGWCH    DS        A                    WAIT CONTROL HALFWORD
LOGFNPT   DS        A                    ==> FILENAME, FILE TYPE
LOGFNUM   DS        X                    FILE NUMBER (OR X'FF'
*****                              ... IF TO BE SYSTEM ASSIGNED)
LOGPMODE  DS        X                    MODE OF FILE
LOGCORE   DS        X                    NUMBER OF SECTOR
BUFFERS REQUESTED
LOGKEY    DS        X                    PROTECT KEY
```

Figure 1 - Login SVC parameter list.

2.4 CREATING AN AREA

An area may be created through the use of the GETAREA instruction:

```
GETAREA Rb,Rk,KEY,LENGTH
```



Results, normal: An area will be created with the specified key and length; the high order 5 bits of Bk should contain the required file identifier; if the low

order 11 bits of the KEY effective address are zero, a key will be assigned to the area by the system; in any case, Rb will be set to point to this new area, the full 16 bit key will be returned in Rk, the length of the area will be returned in register 2, and the MSR will be set to "success". If Rk is 0, 2 or 15, the key will not be returned; if Rb is 2, the length will not be returned.

Results, abnormal: If there is not enough room in the file for the new area, the MSR will be set to "not success".

Program Checks: addressing, protection, register specification.

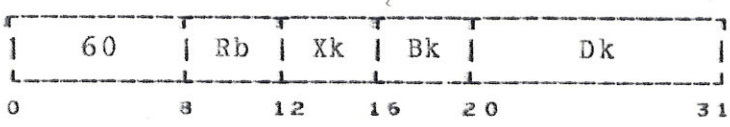
length in bytes

2.5 RETRIEVING AN AREA

To retrieve an existing area, use the RETRIEVE instruction:

RETRIEVE Rb,KEY

KEY (= effective address)



Results, normal: Rb will be set to the start of the area, the length of the area will be returned in register 2 and the MSR will be set to "success"; if Rb is register 2, the length will not be returned.

Results, abnormal: If the specified area was not found, the MSR will be set to "not success".

Program Checks: addressing, register specification.

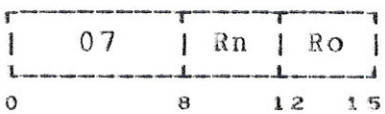
2.6 POINTER MANIPULATION

Registers connected to areas through RETRIEVE may be modified by the user to point anywhere within that area. Doctor Memory will maintain those pointers even if the area is moved or removed to secondary storage. Pointers to or within areas may be manipulated with two special instructions in addition to the normal ones:

COPYBASE Rn,Ro

Results: Rn will be pointed to the same area and offset as Ro.
Program checks: register specification.

MOVEBASE Rn,Ro

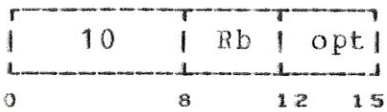


Results: Rn will be pointed to the same area and offset as Ro; the system will no longer maintain the address in Ro.
Program checks: register specification.

2.7 RELEASING AN AREA

To disconnect a register from an area, use the RELEASE instruction:

RELEASE Rb,(<options>)



Where <options> is a parenthesized list (shortest acceptable form underscored, default first in pair):

NOEXPRESS/EXPRESS - indicates whether or not the request should be handled ASAP (bit 12).

NODELETE/DELETE - indicates whether or not the area is to be deleted from the file (bit 13).

NOALL/ALL - indicates whether or not all references to that area (i.e. all registers connected to the area with RETRIEVE, COPYBASE, MOVEBASE, etc.) are to be disconnected (bit 14).

WRITE/NOWRITE - indicates whether or not the area should be written to disk (bit 15).

Results: Rb will have been disconnected from the area; the area will have been deleted if requested.

Program checks: protection, register specification.

2.8 DISCONNECTING A FILE

A file may be disconnected from Doctor Memory by using the LOGOUT command or the MLOGOUT supervisor call. The syntax of the LOGOUT command is a follows:

```
LOGOUT <fileid>
```

The MLOGOUT supervisor call expects register 2 to contain a number between 0 and 31, designating the file to be disconnected.

3 UTILITY PROGRAMS

3.1 DOCTOR: FILE MAINTAINENCE UTILITY

The DOCTOR utility program allows the basic file maintainence facilities which the user might need. The format of the command is as follows:

DOCTOR <function> <parameters> ...

3.1.1 COMPRESS

The COMPRESS function is intended to allow users to free up space allocated for a file which is not needed. The format of the COMPRESS command is as follows:

DOCTOR COMPRESS <filename> <filetype>

At the completion of the command processing, the file will be compacted into the smallest possible amount of space, and any excess will be freed.

3.1.2 EXTEND

The EXTEND function is intended to allow users to extend the size of a file which is full:

DOCTOR EXTEND <filename> <filetype> <number of segments to extend>

3.1.3 COPY

The COPY function is intended to allow the user to create another copy of an already existing file:

DOCTOR COPY <filename> <filetype> <newname> <newtype>

3.1.4 UNLOAD

The UNLOAD function converts a Dr. Memory file to standard GMS format:

DOCTOR UNLOAD <filename> <filetype>

The original file is erased, and a new file with the same name is created. The new file is in sequential format. The first record of the file is the image of the LOGIN command neccessary to re-create the file. The other records contain the key and length as the first four bytes of the record, followed by the data from the area. The record length of the new file will be the larger of the size of the largest area or the length of the LOGIN command.

The new file may be shipped to the 360 using CHARON for backup purposes or manipulated with the editor. It may be re-constituted by using the LOAD function.

3.1.5 LOAD

The LOAD function is the converse of the UNLOAD function:

DOCTOR LOAD <filename> <filetype>

The LOAD function will create a Dr. Memory file from a sequential file in the unloaded format described above.

3.2 FILEBUG: DOCTOR MEMORY FILE DEBUGGER

FILEBUG is an extension to FUDD to allow a user to examine and modify his Doctor Memory data structure. It may be entered by the FILEBUG command to FUDD. FILEBUG is a non-resident ccmmand, but it will remain in core until it receives an 'END' subcommand. Upon entry, FILEBUG will print 'FILEBUG HERE ...' and request a sub-command. The subcommands are described below.

3.2.1 ORIGIN <FILEID> ['SEGMENT']

Function:

The ORIGIN subcommand sets the default file for subsequent operations. Whenever a key is specified whose 5 highorder bits are zero, the default file number is or-ed into these bits to form the final key. E.G. if the user has set the origin file to 1 and requests key 6, the key used will be X'0806'. If the 'SEGMENT' operand is specified, the <key> operands for 'DISPLAY' and 'STORE' will be treated as segment addresses rather than keys. This operand is intended only for system debugging and should be avoided by the normal user.

3.2.2 DISPLAY <KEY> [<OFFSET> [<LENGTH>]]

Function:

The DISPLAY subcommand allows the user to print all or part of an area. The default offset is zero, and the default length is the length of the area.

3.2.3 STORE <KEY> <OFFSET> <DATA> [...]

Function:

The STORE subcommand allows the user to modify data in an area. The default offset is zero. Up to four 16 bit fields may be specified.

3.2.4 GETAREA <KEY> <LENGTH> [<FILL BYTE>]

Function:

The GETAREA subcommand allows the user to create a new area. If a zero key is specified, a key will be assigned by the system. The low order byte of the <fill byte> operand will be used to initialize the contents of the area. The default fill byte is X'00'.

3.2.5 DELETE <KEY>

Function:

The DELETE subcommand allows the user to delete an area.

3.2.6 LOGIN <FILEID> <FILENAME> <FILETYPE>

Function:

The LOGIN subcommand allows the user to connect a file to Dr. Memory. The file will be logged in in utility mode, and the origin will be set to this file.

3.2.7 LOGOUT [<FILEID>]

Function:

The LOGOUT subcommand allows the user to disconnect a file from Dr. Memory. The default <fileid> is the current origin.

3.2.8 ROLLOUT [<FILEID> [<SEGMENT NUMBER>]]

Function:

The ROLLOUT subcommand allows the user to insure that all his areas are written to disk. If a segment number is specified, only that segment will be written out, else all the segments of of the file currently in core will be written to disk. The default <fileid> is the current origin.

3.2.9 GO

Function:

The GO subcommand returns the user to the FUDD command environment, but maintains the origin and keeps FILEBUG in core.

3.2.10 END

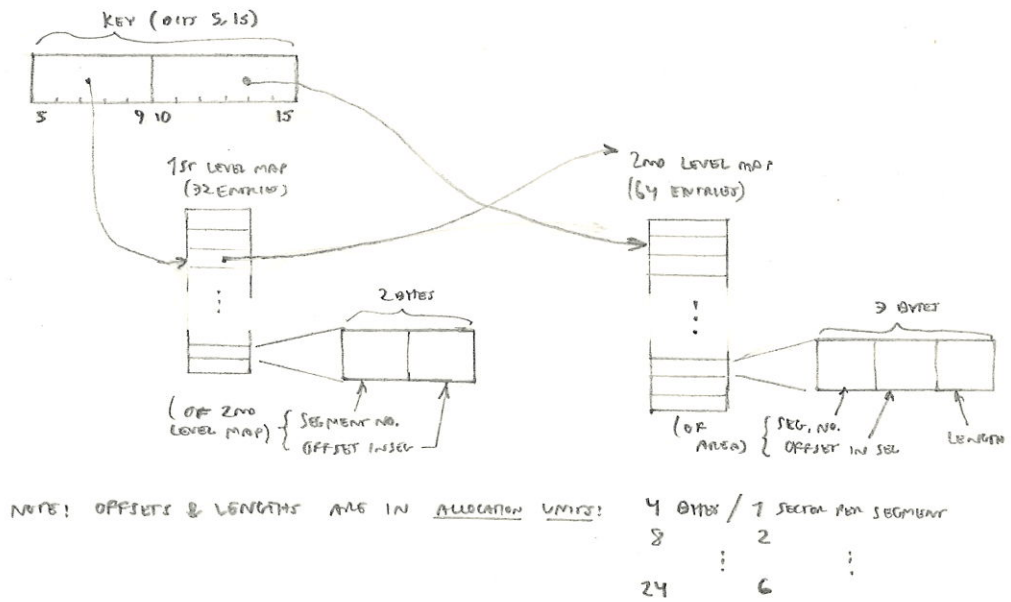
Function:

The END subcommand returns the user to the FUDD command environment and deletes FILEBUG from core.

4 DOCTOR MEMORY DISK ORGANIZATION

The first area of any file, (see figure 2), is reserved for the use of the access method mapping routines, and the GMS file handler. The first sixteen bytes are used by the filing system to indicate file size and organization. The rest of this area is used by the access method tc maintain login information, a free area map and to translate keys into segment/offset addresses. The login infomation is a four byte field containing the segment size of the file (1 byte), the default mode and protect key of the file. The free area map contains the amount of free space available on any given segment. It consists of 256 one byte entries, one per segment. The remainder of the first segment contains the first level index map, containing 32 entries, one for each of the 32 first level keys. Each of these entries contains the segment number and offset to a second level table.

Associated with this map is a 32 bit first level free map. A one bit indicates that the associated second level map is full and can contain nc new keys.



```
*****
*****     SEGMENT ZERO CONTAINS THE BASIC FILE HANDLING
*****     INFORMATION FOR THE FILE TO BE PROCESSED
*****
SEG0      DSECT  ,                    DSECT FOR SEGMENT ZERO
GMSHEAD   DS     CL16 .               GMS FILE HEADER
          ORG    GMSHEAD
GMSRECLN  DS     H .                  RECORD LENGTH
GMSRECCT  DS     H .                  RECORD COUNT
GMSSECCT  DS     H .                  SECTOR COUNT
GMSDATE   DS     CL6 .                DATE OF CREATION
*****     UNUSED 4 BYTES
          ORG
LOGINFO   DS     CL4 .                LOGIN INFORMATION
          ORG    LOGINFO
LOGSEGSZ  DS     X .                  SEGMENT SIZE IN SECTORS
LOGMODE   DS     X .                  DEFAULT MODE
LOGPROT   DS     X .                  PROTECT KEY
*****     UNUSED 1 BYTE
          ORG
FREEMAP   DS     CL256 .              FREE AREA MAP
*****                                 (ONE BYTE/SEGMENT)
FREEKEY   DS     BL.32 .              FREE KEY BIT MASK
$1STMAP   DS     32H .                FIRST LEVEL INDEX
          ORG    $1STMAP
MAPENTRY  DS     2C .                 INDIVIDUAL MAP ENTRY
          ORG    MAPENTRY
MAPSEG    DS     X .                  RELATIVE SEGMENT NUMBER
MAPOFF    DS     X .                  OFFSET WITHIN SEGMENT
           ORG
SEGOLEN   EQU    *-SEG0 .             LENGTH OF SEGMENT ZERO
```

Figure 2 - Segment zero format.

The second level tables, (see figure 3), will be acquired as needed by the filing system; they will be constucted in free space in the file. In order to minimize the amount of space needed for these tables, the relative byte address and length will be kept in allocation units dependent on the segment size. The lengths are expressed in four byte increments for a segment size of one sector (640 bytes), eight byte increments for a segment size of two sectors and so on. The second level table consists of 64 three byte entries consisting of the relative segment number, offset and length in allocation units.

Accessing an area will neccessitate indexing into the first level table to find the appropriate second level table, which will then be fetched. The remainder of the key will determine the index into the second level table, and the proper segment will be loaded into core. The user's register will then be pointed to the required offset into that segment and his register table updated appropriately.

An attempt will be made to keep free space on any given segment contiguous. Lengths defined by the user will be rounded up to the basic units of the space allocation algorithm. Area migration may be used to minimize fragmentation and to place records which are used together on the same segment or adjacent segments.

```
*****
*****     THE SECOND LEVEL INDEX SEGMENT(S) CONTAIN
*****     THE RELATIVE SEGMENT ADDRESSES AND LENGTHS
*****     OF THE AREAS IN THE FILE
*****
KEYSEG    DSECT  ,                    DSECT FOR KEY SEGMENT
KEYTABLE  DS     64CL3 .              SECOND LEVEL KEY TABLE
          ORG    KEYTABLE
KEYENTRY  DS     CL3 .                ACTUAL KEY ENTRY
          ORG    KEYENTRY
AREASEG   DS     X .                  RELATIVE SEGMENT NUMBER OF
AREA
AREAOFF   DS     X .                  OFFSET WITHIN SEGMENT
AREALEN   DS     X .                  LENGTH OF AREA
          ORG
KEYSGLEN  EQU    *-KEYSEG .           LENGTH OF SECOND LEVEL TABLE
```

Figure 3 - Second Level Index Segment.

5 DOCTOR MEMORY CORE ORGANIZATION

The main memory control blocks of Doctor Memory serve as an interface between LEVEL0 and LEVEL1 modules, and to allow speedy access to often used areas and segments. A root pointer to these blocks is in the third word of the disk UCB. This root pointer indicates the address of Doctory Memory's OFFICE, (see figure 4). This block contains first the information for LEVEL0 to access its name table, then information about the free area maintained by DRPD. Finally, the rest of the block contains the name table itself.

Each name table entry corresponds to a file identifier under the control of Doctor Memory. It contains the name and mode for DRPD, the sector starting address of the file, the length of segments in bytes, a pointer to a GMS Active File Table entry for the file, and a lock word.

```
*****
*****     THIS AREA IS GETMAINED AT THE TIME OF THE FIRST LOGIN
*****     AND FREED AT THE LAST LOGOUT. IT CONTAINS INFORMATION
*****     COMMON TO ALL PHASES OF DOCTOR MEMORY AND DRPD. THE
*****     THIRD WORD OF THE DISK UCB POINTS TO THIS AREA.
*****
OFFICE    DSECT  ,                        DSECT FOR RESIDENT AREA
OFFICEHD  DC     A(FILEMAP) ==> TO MAP OR DUMMY ELEMENT
          DC     A(32) NUMBER OF ENTRIES
MAPEDB    DS     CL8 EDB TO SEARCH BY NAME AND MODE
FREELIST  DS     A ==> TO START OF FREE AREA
FREELEN   DS     H LENGTH OF SAME
DRPDEND   DS     A ==> END OF DRPD AREA
DRPDSTRT  DS     A .                        ==> START OF DRPD AREA
DRPDLENG  DS     A .                        LENGTH OF SAME
          DS     A .                        UNUSED
FILEMAP   DS     32CL16 .                   NAME TABLE
          ORG    FILEMAP
MAPSELEC  DS     CL2 .                      SELECTION WORD
          ORG    MAPSELEC
DRNAME    DS     X .                        FILE NAME
DRMODE    DS     X .                        FILE MODE OF ACCESS
RO        EQU    16 .                       READ ONLY INDICATOR
FILESTRT  DS     A .                        SECTOR ADDRESS OF START OF
FILE
FILECNT   DS     A .                        SECTOR COUNT
FILESIZE  DS     A .                        LENGTH OF SEGMENT IN BYTES
FILESTAT  DS     A .                        STATUS BITS
FILEGMS   DS     A .                        ==> AFT FOR FILE
FILEUNIT  DS     A .                        SEGMENT SIZE MULTIPLIER
FILELOCK  DS     A.                       FILE LOCK WORD
*****                                   (LOCKED BY EXTENDED OP ROUTINES)
          ORG
OFFICELN  EQU    *-OFFICE
```

Figure 4 - Doctor Memory's OFFICE Layout.

6 DOCTOR MEMORY AND PROGRAMS

One of the purposes of the 16 bit area identifier is to allow programs to be dynamically relocated with a minimum change in programming convention. A "binder" program can translate external references to a form usable at run time, directly replacing 16 bit VCON's with 16 bit keys. The programs to be accessed are placed in a program library in executable form by the binder.

Certain programming conventions must be adhered to in order for successful relocation of programs:

1. Programs must be serially re-usable .
2. Programs must be self-relocating, i.e. they cannot contain data (e.g. address constants) whose value is dependent upon program location.
3. There will be some maximum size limit placed on CSECTs to be re-located, perhaps 1280 bytes. – 1 sector

The facilities for managing the program counter exist, but the flow of control through calls and returns has not yet been established.

PRES: Direct readout. Dr. Memory, [UHCLEM].

CLEM: Thank you, thank you. Now Doctor - I'm speaking to you, Doctor ...

DR M: MMMMMMMMMM?

CLEM: Something the leprechauns asked me when I was a sprout in Indiana has always puzzled me. Doctor? Question. Evaluate. Why does the porridge bird lay his egg in the air?

I Think We're All Bozos on this Bus
Firesign Theater