TECHNICAL MANUAL NO 12

ABSOLUTE OVERLAYS

UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

TECHNICAL MANUAL NO 12

ABSOLUTE OVERLAYS

Edited Lisha Kayrooz

MNT-12
1 May 1974

This manual has been authorized by
the Director of the Computer Centre.

CONTENTS

1. ABSOLUTE OVERLAYS - INTRODUCTION

This document describes the structure and usage of an absolute overlay system. A subroutine OVERLAY is provided in the standard library of subroutines which can be used from a FORTRAN or MACRO program to call the overlays.

1.1 OVERLAY STRUCTURE

An overlaid program consists of a permanent segment and one or more overlaid segments. Each segment can contain one or more FORTRAN or MACRO subroutines or functions, but the MAIN program must be in the permanent segment if FORTRAN is used.

The program overlay structure may conveniently be described in terms of a tree structure as shown in Figure 1.1. There are a number of alternative branches in the tree, each representing one combination of coexisting segments in core. Communication can be carried out directly along branches, but only indirectly between branches through common branch links. The root of the tree is the permanent segment.

Each overlay segment or link has three attributes:

(a) Overlay Number

A decimal integer representing the order of the overlay at creation time, that is, the order in which they are presented to the Linking Loader. This number may be used in calling subroutine OVERLAY to indicate which particular overlay is to be called. In Figure 1.1, the overlay numbers are in roman numerals.

(b) Overlay Area

An octal digit indicating the topological area of core into which the overlay segment is to be loaded. The overlay areas are numbered consecutively from 1 along any branch of the tree. Area 1 is the first overlay area after the permanent area. In Figures 1.1 and 1.2 the overlay areas

are shown in arabic numerals. An example of possible core usage is shown in Figure 1.2 and illustrates that the actual position of an area depends on those below it in the branch. The total core area used depends on the length of the longest branch.

(c) Overlay Name

An optional name with a maximum of five ASCII characters which may be used as an alternative to the overlay number when calling the overlay. The name may be the same as that of a subroutine within the overlay.

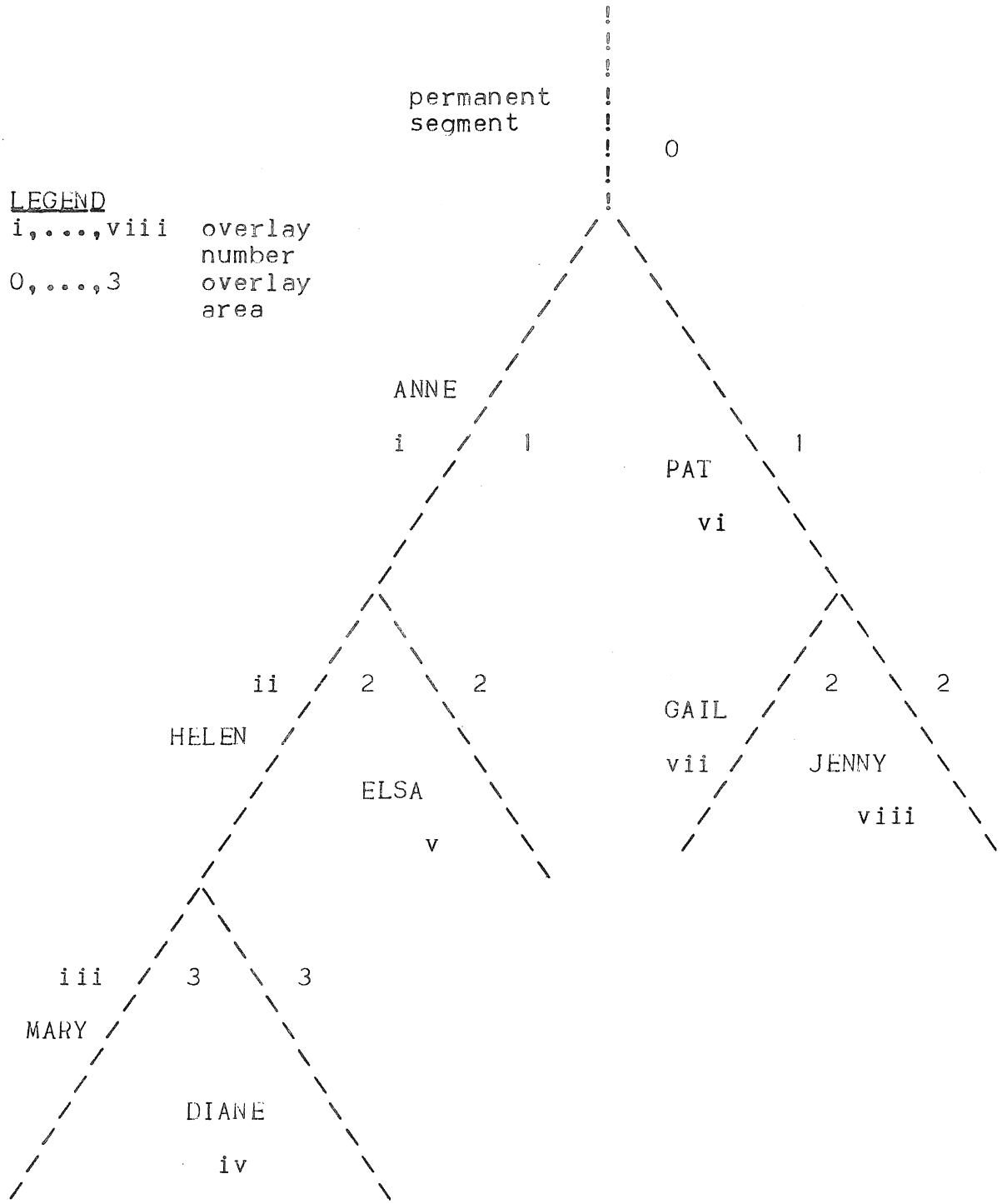

Figure 1.1 Tree Representation of Program Overlay Structure

Number	Area	Name
i	1	ANNE
ii	2	HELEN
iii	3	MARY
iv	3	DIANE
v	2	ELSA
vi	1	PAT
vii	2	GAIL
viii	2	JENNY

Table 1-1 Summary of Overlay Attributes for Figure 1.1

Throughout this description reference is made to overlays higher or lower in a branch. This refers to the area numbers which commence at 1 from the permanent segment. Thus, in figure 1.1, ANNE is lower in the branch than ELSA since ANNE's area is 1 and ELSA's is 2.

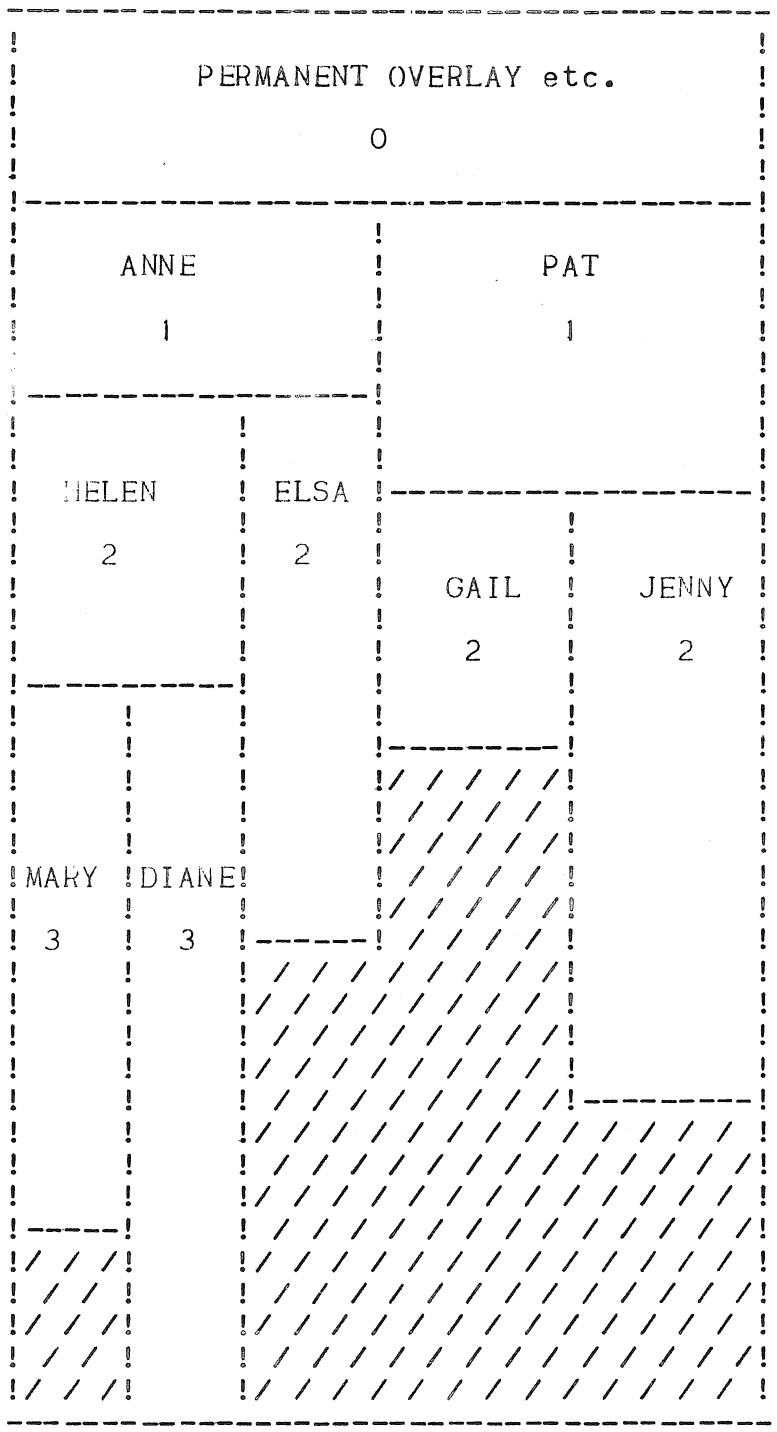

Figure 1.2 Example of Possible Core Usage

1.2 OVERLAY CREATION BY THE USER

The user indicates to the system the routines to be included in each overlay. Basically, the Loader is given a list of files to load with separators inserted in the list at the beginning of each overlay indicating the overlay area and name.

The order of the overlays is important as this determines the core layout. Each branch must be completed in an orderly fashion as described more fully in section 1.3.

The user must indicate where each overlay starts. This is done with a loader string that indicates the overlay area and its name.

The standard system loader is the Digital 5-series loader. To use the overlay system the special overlay loader OVLOAD should be used to load the main program and the necessary overlay routines.

The appropriate switches for this loader are:

/nZ is the 'overlay' switch.
n is the 'area' number. (assumed to be 1 if /Z).

/M type out a map (/1M at the start causes symbol listing).

/G finish loading and exit to the monitor.

/E execute after loading.

/F forces library search. (needed before a map).

Following the order given in table 1-1, a suitable command string might read:

```
.R OVLOAD                          ; run the overlay loader.
(*/1M)                             ; if a symbol map is required.
*MAINPG,FSTSUB                     ; permanent segment of files,
*SNDSUB                            ; MAINPG, FSTSUB &SNDSUB.
                                   ; more main segment routines.
*ANNE=/1ZTRDSUB                    ; overlay 1 named ANNE,
                                   ; comprising file TRDSUB.
*HELEN=/2ZFTHSUB                   ; overlay 2 named HELEN,
                                   ; comprising file FTHSUB.
*MARY=/3ZDISUB                     ; overlay 3 named MARY,
                                   ; comprising file DISUB.
*DIANE=/3ZDESUB,FFASUB             ; overlay 4 named DIANE,
                                   ; comprising files DESUB and
                                   ; FFASUB, to go into overlay
                                   ; area 3 when called.
*/G
```

In order to guard against typing errors, it is wise to keep the overlay loader strings in a separate file. This file can be referenced by the overlay loader using an indirect command.

Example:

The file MYOV.CMD contains the following:

```
MAIN,FSTSUB,SNDSUB
ANNE=/1ZTRDSUB
HELEN=/2ZFTHSUB
MARY=/3ZDISUB
DIANE=/3ZDESUB,FFASUB
/F/M
/G
```

To create the overlays:

```
.R OVLOAD

*MYOV@                             Searches for .CMD file first,
                                   then for null extension.
```

Note that the overlay area name can be omitted if desired.
e.g. The line

```
HELEN=/2ZFTHSUB
```

could be replaced by
=/2ZFTHSUB

in the above file MYOV.CMD.

SAVING FILES THAT USE OVERLAYS

Users can SAVE overlay programs by following the procedure outlined below. (The success of the operation will depend on the size of the two files produced and the user's file storage limits).

(a) Load the main program and its overlays into core

e.g. Assume a main program called TEST1 with two overlay files TEST2 and TEST3, both to be overlayed to area 1.

```
.R OVLOAD
*TEST1
*BBB=/1ZTEST2
*CCC=/1ZTEST3
*/G
```

(b) Save the program

```
.SAVE MYPRG<cr>
```

At this stage, there will be two files created on the user's area
(i) MYPRG.SAV
(ii) BBB.ABS

Note that the ABS file takes the name of the first overlay area after the permanent (area=0) section. In the example above this is BBB. If there is no name provided in the loader string, the default name of OVFIL will be used.

The program can now be referred to by name MYPRG. For example, to run the program, type

```
.RUN MYPRG
```

The subroutine OVERLAY looks for the .ABS file as follows:

1. Firstly, the .ABS is looked for on any ppn passed across in the arguments to the first call to OVERLAY. This usage is not recommended for future programs but is included for compatibility with that provided in earlier systems.

2. Secondly, it is looked for on the ppn from which the .SAV file came if run from a .SAV file.

3. Thirdly, it is looked for on the jobs ppn.

4. Finally, it is looked for on device SYS:

[Note: To implement 2, subroutine OVERLAY is given a starting - address OVENT and must be loaded after a FORTRAN main program (i.e. a program with starting address "MAIN."). This works automatically when OVERLAY is obtained from the Fortran library.]

Note that it is the overlay area that is indicated in the string. The overlay number is the number of the overlay in the command string and this has been included in the example for clarity by putting it after a comment indicator (;).

1.3 OVERLAY CREATION BY THE SYSTEM

Each overlay command indicates to the Linking Loader the overlay area and name to be used. On receipt of this command a library search is carried out so that any library routines mentioned in the last or old overlay, which do not already exist lower in the branch, can be included.

An entry is then created in an overlay table within the subroutine OVERLAY, containing the name and area information given. The overlay number is the number of the entry in the overlay table.

If the new overlay area number is greater than the old, the new overlay is loaded into core immediately above the old overlay. However, one or more old overlays must be overwritten if the new area is less than or equal to the old area. In this case, the core images of these old overlays are written out on an overlay file and no further changes can be made to them. Information about size and position of these overlays is placed in the overlay table for later use by OVERLAY. The next location to be loaded is then reset so that the new overlay is placed in the

area previously used by the old overlay(s).

For example, in Figures 1.1 and 1.2, consider the case when overlay ELSA is reached. The old overlay at this time is DIANE with an area of 3 compared with ELSA's 2. Therefore DIANE is written out. Now the old overlay is HELEN with area 2 and this is again written out. Now the old overlay is ANNE with area 1 and ELSA can be added immediately above it.

Note that this writing out process is done during creation and it is at this stage that the overlay order is important as it determines the core usage and the possible interconnections between overlays. During actual execution of the program, the order of usage can be quite different and no writing out of an old overlay is done when a new overlay is brought in by OVERLAY. The areas into which the overlays are brought are fixed permanently at creation time.

An alternative way of numbering and presenting the overlays for the example is shown in Figure 1.3.

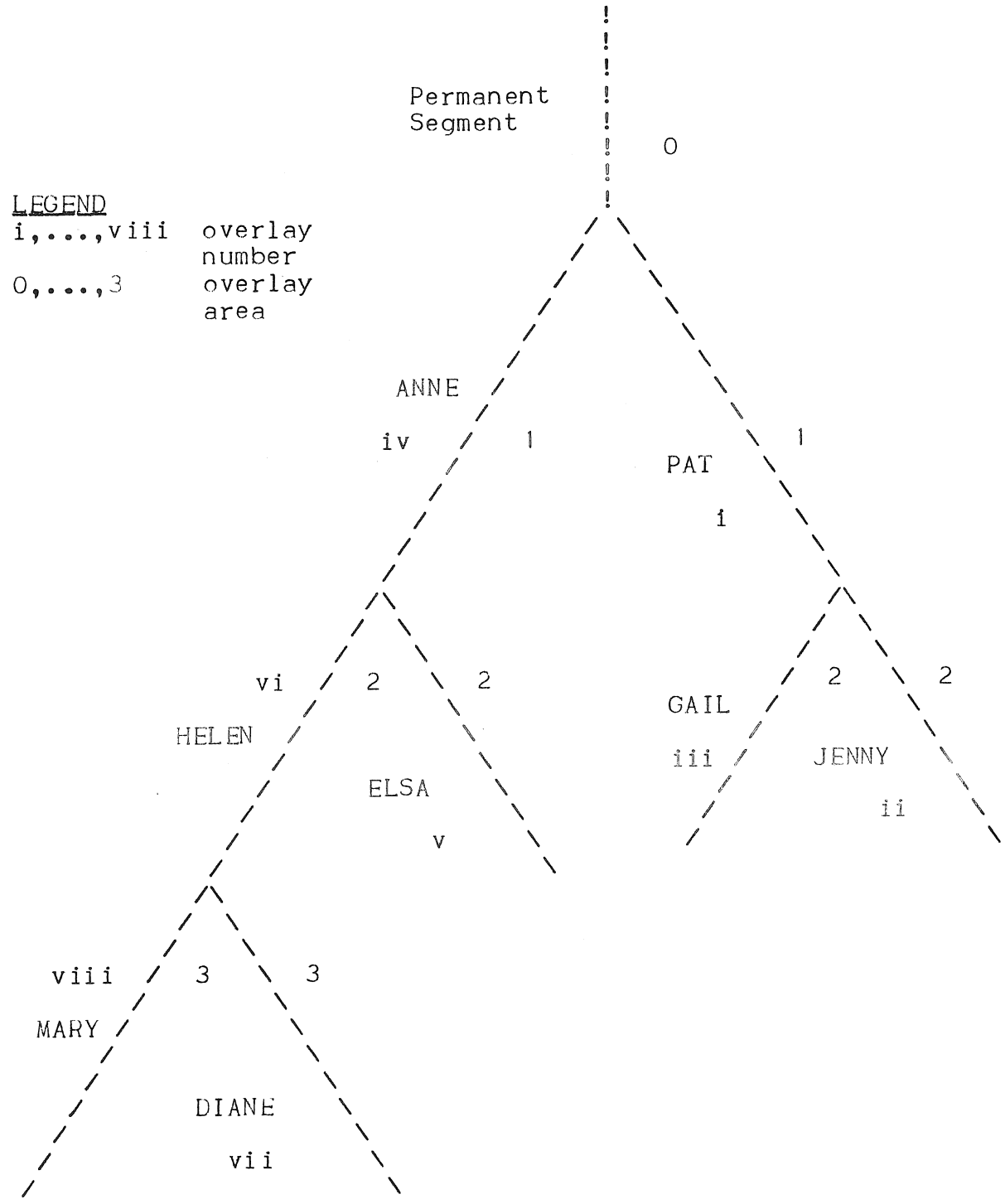

Figure 1.3 Alternative Representation of Figure 1.1

Number	Area	Name
i	1	PAT
ii	2	JENNY
iii	2	GAIL
iv	1	ANNE
v	2	ELSA
vi	2	HELEN
vii	3	DIANE
viii	3	MARY

Table 1-2 Summary of Overlay Attributes for Figure 1.3

2. USE OF OVERLAYS

2.1 CALLING OVERLAYS

When the program is executed, no overlays are in core and an overlay must be brought into core before the routines in the overlay can be used. To call overlay 4 in the example, we could use one of the following statements:

(a) CALL OVERLAY(4)

(b) J=4
CALL OVERLAY(J)

(c) CALL OVERLAY('DIANE')

(d) NAME='DIANE'
CALL OVERLAY(NAME)

DIANE will not be loaded if it is already in core.

Note that the call by name can only be used if the overlays have been named. If any overlay is unnamed its name entry in the overlay table will be zero.

When a call to load an overlay is made, overlays lower in the branch will also be loaded if they are not already in core. For example, HELEN and ANNE would be loaded if one of the above calls to DIANE were made from the permanent segment and if they were not already in core. Note that MARY, ELSA, PAT, GAIL and JENNY would be marked as not being in core.

When an overlay is loaded by a call to OVERLAY, its core image will be returned to its state at creation time. Therefore, values of variables set within the overlay during execution are lost when reloaded.

2.2 COMMUNICATION BETWEEN OVERLAYS

Because of the structure of the overlay system, there may be upwards and downwards communication between routines in a branch. Thus, routines in the permanent segment can call routines in any overlay and vice versa. In the example, routines in ANNE can call those in DIANE and vice versa, but those in DIANE cannot call those in ELSA. DIANE and ELSA can only pass data to each other via some common arguments from ANNE, or through a COMMON area located in ANNE or the permanent segment as described later in section 3.2.

3. ACCESSIBILITY OF OTHER AREAS

3.1 LIBRARY ROUTINES

Since a library search is carried out at the end of each overlay and the permanent segment, library routines are placed in overlays as necessary. For example, if SIN were used by HELEN, it would be placed there if not already mentioned in ANNE or the permanent segment. DIANE and MARY could both refer to HELEN's SIN. However, a second copy would be placed in ELSA if SIN was required there as well.

3.2 COMMON

Blank COMMON, named COMMON and BLOCK DATA can be thought of in the same way as library routines. They are placed in the first overlay in the branch that mentions them and are then available to overlays further up the branch. Thus, a COMMON defined in a permanent segment can be used by all overlays. However, a COMMON first defined in HELEN is available to HELEN, MARY and DIANE only and will be reset to its original values every time HELEN is reloaded. Furthermore, if this COMMON is mentioned in ELSA, it will be an entirely different one to that provided in HELEN.

3.3 DATA STATEMENTS

DATA statements can give subtly different effects on reloading since they can refer to something in another area. It must be remembered that DATA statements are set up at creation time and not execution time. For example, suppose blank COMMON is defined in the permanent segment and suppose HELEN contains DATA statements that refer to blank COMMON. In this case, reloading of HELEN will not reset the data as they were set at creation time. However, if HELEN's DATA statements referred to variables in HELEN or to a COMMON that was first mentioned in HELEN, then they will be reset every time HELEN is reloaded as they have been placed in the core image overlay that is reloaded.

4. ERRORS IN OVERLAYS

4.1 ERRORS DETECTED DURING CREATION

?MISSING OVERLAY SUBROUTINE
...There was no overlay call in the permanent segment.

?OVERLAY TABLE FULL
...More than 20 overlays.

?MISSING OVERLAY AREA
...Numbering of overlay areas incorrect.

4.2 ERRORS DURING LOADING OF AN OVERLAY

All error messages generated by subroutine OVERLAY will indicate the name or number of the overlay that is being called and the octal address of the call. In addition, one of the following messages will be output:

OVERLAY NUMBER INCORRECT
...A call to OVERLAY with a number 0 or greater than 20. Note that a negative number is interpreted as a name.

OVERLAY NOT IN TABLE
...The name in the overlay call does not exist, perhaps the overlay was not named at creation time.

INCORRECT NUMBER OF ARGUMENTS

ERROR READING OVERLAY FILE

DISK UNAVAILABLE

OVERLAY FILE NOT PRESENT
...Trouble whilst trying to read the overlay file.

OVERLAY WILL OVERWRITE CALLER
...For example, if DIANE trys to load ANNE or ELSA.

4.3 UNDETECTED ERRORS

The user can cause errors that are undetected by the system and which will result in unpredictable results.

A call to a routine in an overlay that has not previously been loaded by a CALL OVERLAY or which has been overwritten by a later call, will obviously be incorrect.

A call to a routine in another branch can result if, for example, HELEN calls a routine that is incorrectly placed in ELSA. This leads to unpredictable results at creation time.

5. EXAMPLE

The following FORTRAN example illustrates the use of overlays.

file MAINPG.F4 :

```
      A1=0.0
      A2=0.0
      A3=0.0
C
C
C******   READ RADIUS AND NUMBER
C
C
 5    READ 10, A1, NUM
 10   FORMAT (F10.4, I1)
      IF (A1 .EQ. 0.0) GO TO 50
C
C
C******  NUM = 1 (CIRCLE), 2 (CYLINDER), 3 (SPHERE)
C
C
      IF (NUM - 2) 1, 2, 3
C
C
C******  CIRCLE
C******         PERIMETER
C******         AREA
C
C
 1    CALL OVERLAY ('CIRCL')
      CALL CIR (A1,A2,A3)
      PRINT 11
 11   FORMAT (1H1/ 8H CIRCLE://35H  RADIUS PERIMETER  AREA)
      GO TO 40
C
C
C******  CYLINDER
C******           SURFACE AREA
C******           VOLUME
C
C
 2    CALL OVERLAY ('CYLIN')
      CALL CIR (A1,A2,A3)
      CALL CYL (A1,A2,A3)
      PRINT 22
 22   FORMAT (1H1/10H CYLINDER:/ 11H    HEIGHT/
     1 35H AND RADIUS  SURF. AREA     VOLUME)
```

```
      GO TO 40
C
C
C******  SPHERE
C******          SURFACE AREA
C******          VOLUME
C
C
 3    CALL OVERLAY ('SPHER')
      CALL SPH (A1,A2,A3)
      PRINT 33
 33   FORMAT (1H1/ 8H SPHERE://35H  RADIUS SURF. AREA
     1 VOLUME)
C
C
 40   PRINT 45, A1,A2,A3
 45   FORMAT (1H, F10.4, 2F12.4)
      GO TO 5
 50   STOP
      END
```

file CIRCL.F4 :

```
      SUBROUTINE CIR (RAD, PER, ARE)
      PER = 2.0*3 14159*RAD
      ARE = 3.14159*RAD*RAD
      END
```

file CYLIN.F4 :

```
      SUBROUTINE CYL (HGT, SAR, VOL)
      SAR = SAR*(HGT+HGT)
      VOL = VOL*HGT
      END
```

file SPHER.F4 :

```
      SUBROUTINE SPH (RAD, SAR, VOL)
      SAR = 4.0*3.14159*RAD*RAD
      VOL = (4.0*3.14159*RAD**3)/3.0
      END
```

When this program is loaded, as follows,

```
.R OVLOAD
*MAINPG
*=/1ZCIRCL
*=/2ZCYLIN
*=/1ZSPHER
*/G

.SAVE MAINPG
```

and then run with the following data,

```
5.0   1
5.0   2
5.0   3
0.0
```

the results are as follows :

```
 CIRCLE:

     RADIUS    PERIMETER        AREA
     5.0000      31.4159     78.5397

 CYLINDER:
     HEIGHT
 AND RADIUS   SURF. AREA     VOLUME
     5.0000    314.1590    392.6987

 SPHERE:

     RADIUS   SURF. AREA     VOLUME
     5.0000    314.1590    523.5983
```