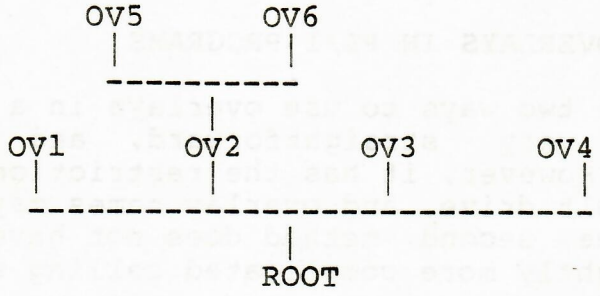
Appendix E

This appendix describes several additional features incorporated into LINK-80 and LIB-80 in release versions later than 1.0, including extensions to process run-time overlays, and controls for location of source, intermediate, and destination files. Use of the automatic PL/I-80 library search "request item" is included, along with a description of new command line error reporting formats. Additional LIB-80 facilities are also included for deleting or replacing various modules in a subprogram library.

E.1.0. OVERLAYS

LINK may be used to produce a simple tree structure of overlays as shown in the diagram below:



In addition to producing ROOT.COM and ROOT.SYM files, LINK will produce an OVL file and a SYM file for each overlay specified in the command line. The OVL file consists of a 256-byte header containing the load address and length of the overlay, followed by the absolute object code. The origin of an overlay is the highest address of the module below it on the 'tree' rounded up to the next 128-byte boundary. The stack and free space for the PL/I program will be located at the top of the highest overlay linked, rounded up to the next 128-byte boundary. This address is written to the console upon completion of the entire link and is patched into the root module in the location '?MEMRY'. The SYM file contains only those symbols which have not been declared in another module lower in the 'tree'.

The following restrictions must be observed when producing a system of overlays with PL/I-80 and LINK:

Each overlay has one entry point by which it is entered. This entry point is assumed by the overlay manager to be at the base (load address) of the overlay.

No upward references are allowed from a module to an entry point in an overlay higher on the tree, other than the main entry point of the overlay as described in 1. Downward references to entry points in overlays lower on the tree or in the root module are allowed.

The overlays are not relocatable. Hence the root module must be a COM file.

Common blocks (Externals in PL/I) which are declared in one module may not be initialized by a module higher in the tree. Any attempt to do so will be ignored by LINK.

Overlays may be nested to a depth of 5 levels.

The default buffer located at 80H is used by the overlay manager, so user programs should not depend on data stored in this buffer.

E.1.1. USING OVERLAYS IN PL/I PROGRAMS

There are two ways to use overlays in a PL/I program. The first method is very straightforward, and will suffice for most applications. However, it has the restrictions that all overlays must be on the default drive, and overlay names may not be determined at run-time. The second method does not have these restrictions, and involves a slightly more complicated calling sequence.

To use the first method, an overlay is simply declared as an entry constant in the module where it is referenced. As an entry constant, it may have parameters declared in a parameter list. The overlay itself is simply a PL/I procedure, or group of procedures. For example, the following program is a root module having one overlay:

```
root: procedure options (main);
   declare ovl entry (char (15));
   put skip list ('root');
   call ovl ('overlay 1');
   end root;
```

The overlay OV1.PLI appears as follows:

```
ovl:  procedure (c);
      declare c char (15);
      put skip list (c);
      end ovl;
```

Note that if parameters are passed to an overlay, it is the programmer's responsibility to ensure that the number and type of the parameters are the same in the calling program and the overlay itself.

To link these two programs into an overlay system, the following link command would be used:

```
LINK ROOT(OVL)
```

(The command line syntax for linking overlays is described in detail in a later section.)

LINK will produce four files from this command: ROOT.COM, ROOT.SYM, OVL.OVL and OVL.SYM. When ROOT.COM is executed, it will first put the message 'root' out at the console. The 'call ovl' statement will transfer control to the overlay manager. The overlay manager loads the file OVL.OVL from the default drive at the proper location above ROOT.COM and transfers control to it, passing the char (15) parameter in the normal manner. The overlay then executes, producing the message 'overlay 1' at the console. It then returns directly to the statement following the 'call ovl' in root.pli, and execution continues from that point.

Using this method, if the overlay manager determines that the requested overlay is already in memory, the overlay will not be reloaded before control is transferred to it. There are several important notes regarding this first overlay method:

The name associated with the overlay in the call and entry statements is the actual name of the OVL file loaded by the overlay manager, so the two names must agree. Since symbol names are truncated to 6 characters in the REL file produced by PL/I-80, the names of the OVL files must be limited to 6 characters.

The name of the entry point to an overlay (the name of the procedure) need not agree with the name used in the calling sequence. The same name should be used to avoid confusion.

The overlay manager will only load overlays from the default drive (the drive which was the default drive when execution of the root module began, regardless of any changes to the default drive which may have occurred since then).

The names of the overlays are fixed - the source program must be edited, recompiled and relinked to change the names of the overlays.

No non-standard PL/I statements are needed (the program is transportable to other systems).

In some applications it is useful to have greater flexibility with overlays, such as the ability to load overlays from different drives, or the ability to determine the name of an overlay at run-time, say from the keyboard or from a disk file. This is accomplished using a second overlay method.

In this case, an explicit entry point into the overlay manager must be declared in the PL/I program as follows:

```
declare ?ovlay entry (char (10), fixed (1));
```

The first parameter is a character string specifying the name of the overlay to load and an optional drive code in the standard CP/M format 'd:filename'. The second parameter is the load flag. If the load flag is 1, the overlay manager will load the specified overlay whether or not it is already in memory. If the load flag is 0, the overlay will only be loaded if it is not already in memory.

The 'call ?ovlay' statement tells the overlay manager to load the requested overlay, if needed. The overlay manager returns to the calling program, which must then perform a dummy call to execute the overlay just processed by the overlay manager. This allows a parameter list to be passed to the overlay.

The example shown in the first method above would appear as follows:

```
root: procedure options (main);
   declare ?ovlay entry (char (10), fixed (1));
   declare dummy entry (char (15));
   declare name char (10);
   put skip list ('root');
   name = 'OV1';
   call ?ovlay (name, 0);
   call dummy ('overlay 1');
   end root;
```

OV1.PLI would be the same as before.

At run-time the overlay manager would load OV1.OVL from the default drive, since that is the current value of the variable 'name', and then return to the calling program (in this case, root). At this point, the argument 'overlay 1' would be set up according to the PL/I-80 parameter passing conventions. The 'call dummy' transfers control to the overlay manager, which would simply tranfer control to the base address of the overlay whose name was just processed. When OV1 is finished, it returns to the statement following the 'call dummy' statement. Note that while in the example above, 'name' was set to 'OV1' in an assignment statement, the overlay name could have been supplied as a character string derived from some other source,

such as the operator's keyboard. Several important points must be observed when using the second overlay technique:

A drive code may be specified so overlays may be loaded from drives other than the default drive. If no drive is specified, the default drive is used as described in Method 1.

Since the name of the overlay is specified in the character string (and not by the entry symbol), it may be up to 8 characters in length.

If there are any parameters in the dummy call following the 'call ?ovlay', they must agree in number and type with the parameters in the procedure declaration in the overlay.

E.1.2. SPECIFYING OVERLAYS IN THE COMMAND LINE

The syntax for specifying overlays is similar to that for linking without overlays, except that each overlay specification is enclosed in parentheses. An overlay specification may be in one of the following forms:

```
link root(ov1)

link root(ov1,part2,part3)

link root(ov1=part1,part2,part3)
```

The first command produces the file OV1.OVL from a file OV1.REL, while the second command produces the OV1.OVL file from OV1.REL, PART2.REL, and PART3.REL. In the last case, the OV1.OVL file is produced from PART1.RLE, PART2.REL, and PART3.REL.

Note that a left parenthesis, which indicates the start of a new overlay specification, also indicates the end of the group preceding it. In other words, the following command line is invalid and will be flagged as an error:

```
LINK ROOT(OV1),MOREROOT
```

All files to be included at any point on the 'tree' must appear together, without any intervening overlay specifications. Thus the following command is valid:

```
LINK ROOT,MOREROOT(OV1)
```

Any filename in the command line may be followed by a number of link switches enclosed in square brackets, as described in the LINK-80 Operator's Guide. Note that the overlay specifications are not set

off from the root module or from each other with commas. Spaces may be used to improve readability.

Nesting of overlays is indicated in the command line by nesting parentheses. The following command line could be used to link the overlay system shown on the first page of the overlay description:

```
LINK ROOT (OV1) (OV2 (OV5) (OV6)) (OV3) (OV4)
```

E.1.3. SAMPLE LINK EXECUTION

In the following sample link operation, notice that OV1 is flagged as an undefined symbol. LINK is simply indicating that OV1 has not been defined in the current module, so it is assumed to be either the name of an overlay or a dummy entry point to an overlay. When linking overlays, each entry variable which refers to an overlay (by actual name or a dummy entry) will appear as an undefined symbol. No symbols other than these actual or dummy overlay entry points should be undefined.

```
A>LINK ROOT(OV1)
LINK 1.1

PLILIB   RQST   ROOT     0100   /SYSIN/  1A15   /SYSPRI/ 1A3A

UNDEFINED SYMBOLS:

OV1

ABSOLUTE     0000
CODE SIZE    18BC (0100-19BB)
DATA SIZE    02A9 (1A90-1D38)
COMMON SIZE  00D4 (19BC-1A8F)
USE FACTOR     4E

LINKING OV1.OVL

PLILIB   RQST

ABSOLUTE     0000
CODE SIZE    0024 (1D80-1DA3)
DATA SIZE    0002 (1DA4-1DA5)
COMMON SIZE  0000
USE FACTOR     09

MODULE TOP   1E00
```

```
A>ROOT

root
overlay 1
End of Execution
A>
```

E.1.4. RUN-TIME ERROR MESSAGES

The overlay manager may produce one of the following error messages:

ERROR (8) OVERLAY, NO FILE d:filename.OVL
The indicated file could not be found.

ERROR (9) OVERLAY, DRIVE d:filename.OVL
An invalid drive code was passed as a parameter to ?ovlay.

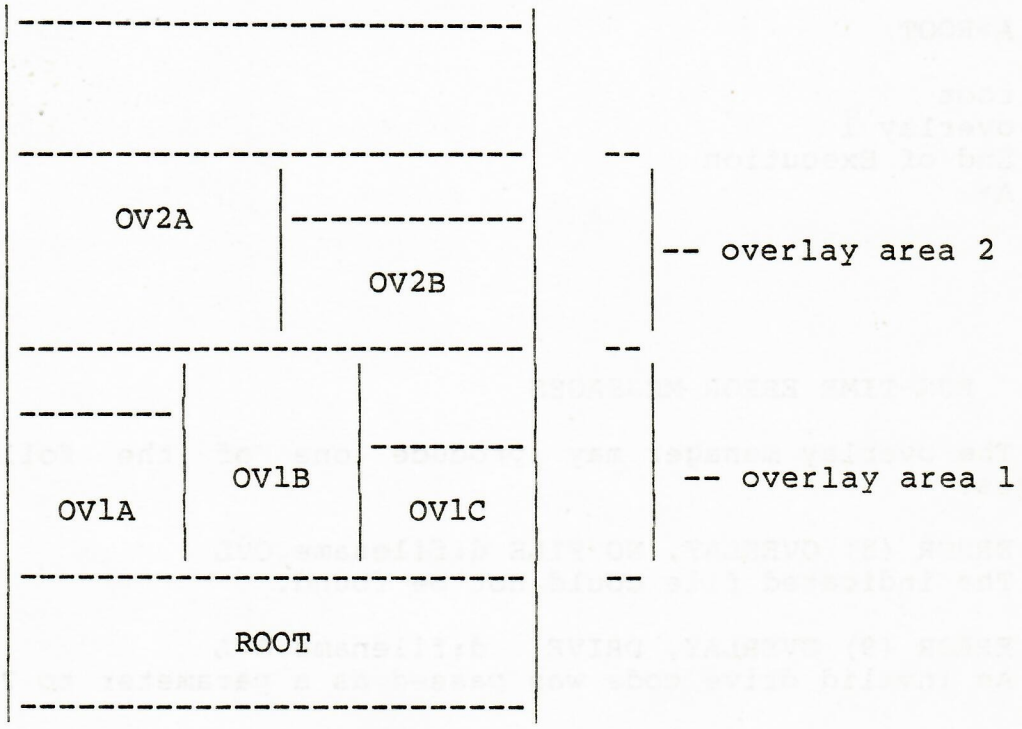
ERROR (10) OVERLAY, SIZE d:filename.OVL
The indicated overlay would overwrite the PL/I stack and/or free space if it were loaded.

ERROR (11) OVERLAY, NESTING d:filename.OVL
Loading the indicated overlay would exceed the maximum nesting depth.

ERROR (12) OVERLAY, READ d:filename.OVL
Disk read error during overlay load, probably caused by premature EOF.

E.1.5. OTHER OVERLAY SYSTEMS

A system of overlays may also be produced which is not a tree structure, but rather contains a number of separate overlay areas, as shown in the figure below:



In such a system, the root module can reference any of the overlays. An overlay may reference entry points in the root module or the main entry point of any overlay which is not in the same overlay area.

Linking a system of overlays as shown above is done in a number of steps. One link must be performed for each overlay area, since the address of the top of the overlay area must be supplied to LINK when linking the next higher overlay area. For example, the command

```
LINK ROOT (OV1A)(OV1B)(OV1C)
```

generates the three overlays in overlay area 1, and indicates the top address of the module. This address is supplied as the load address in the next command:

```
LINK ROOT (OV2A[Lmod top]) (OV2B [Lmod top])
```

This command creates the overlays for overlay area 2 at the appropriate address. Note that the overlay area which is the highest in memory should be linked last, since the module top address is always written into the root module at the end of the link.

At some point after the entire system has been linked, it may be desirable to relink only one overlay, which may not be at the top overlay area. This may be done using the $OZ switch to prevent generation of a root module which would contain an erroneous ?MEMRY value.

It is the responsibility of the programmer to ensure that none of the overlays overlap, and that no overlay attempts to reference

another overlay in the same overlay area.

E.1.6. THE LINK-80 "$" SWITCH

The '$' switch is used to control the source and destination devices under LINK-80. The general form of the switch is:

$td

where 't' is a type and 'd' is a drive specifier. There are five types:

C - console

I - intermediate

L - library

O - object

S - symbol

The drive specifier may be a letter in the range 'A' thru 'P' corresponding to one of sixteen logical drives, or one of the following special characters:

X - console

Y - printer

Z - byte bucket

$Cd - Console

Messages which normally appear at the console may be directed to the list device ($CY) or may be suppressed ($CZ). Once $CY or $CZ has been specified, $CX may be used later in the command line to redirect console messages to the console device.

$Id - Intermediate

Intermediate files generated by LINK are normally placed on the default drive. The $I switch allows the user to specify another drive to be used by LINK for intermediate files.

$Ld - Library

LINK normally searches on the default drive for library files

which are automatically linked because of a request item in a REL file. The $L switch instructs LINK to search the specified drive for these library files.

$Od - Object

LINK normally generates an object file on the same drive as the first REL file in the command line, unless an output file with an explicit drive is included in the command. The $O switch instructs LINK to place the object file on the drive specified by the character following the $O, or to suppress the generation of an object file if the character following the $O is a 'Z'.

$Sd - Symbol

LINK normally generates a symbol file on the same drive as the first REL file in the command line, unless an output file with an explicit drive is included in the command. The $S switch instructs LINK to place the symbol file on the drive specified by the character following the $S, or to suppress the generation of a symbol file if the character following the $S is a 'Z'.

'td' character pairs following a '$' must not be separated by commas. The entire group of $ switches is set off from any other switches by a comma, as shown below:

```
LINK PART1[$SZ,$OD,$LB,Q],PART2

LINK PART1[$SZODLB,Q],PART2

LINK PART1[$SZ OD LB],PART2[Q]
```

The three command lines above are equivalent.

The $I switch specifies the drive to be used for intermediate files during the entire link operation. The other '$' switches may be changed in the command line. The value of a '$' switch will remain in effect until it is changed as the command line is processed from left to right. This is generally useful only when linking overlays. For example:

```
LINK ROOT (OV1[$SZCZ])(OV2)(OV3)(OV4[$SACX])
```

will suppress the SYM files and console output generated when OV1, OV2 and OV3 are linked. When OV4 is linked, the SYM file will be placed on drive A: and the console output will be sent to the console device.

The NR and NL switches used in LINK 1.0 to suppress the recording and listing of the symbol table are not recognized by LINK 1.1, since $SZ and $CZ can be used to perform these functions.

E.1.7. THE REQUEST ITEM

Version 1.1 of PL/I-80 uses the request item (a specific bit pattern in a REL file) to indicate to LINK that the PLILIB is to be searched. This is also how the Microsoft compilers link their run-time libraries. When LINK processes a library request, it first searches for an IRL file with the specified filename. If there is no IRL file, it searches for a REL file of that name. Failing in both searches, the error message

NO FILE: filename.REL

is produced, and LINK aborts. Libraries requested in this manner will appear in the symbol table listed at the console with a value of 'RQST'.

E.1.8. COMMAND LINE ERRORS

The error messages 'FILE NAME ERROR' and 'INVALID SYNTAX' are no longer generated. Instead, when a command line error of any kind is detected the command tail is echoed up to the point where the error occurred, followed by a question mark. For example:

```
LINK A, B, C; D
A, B, C;?

LINK LONGFILENAME
LONGFILEN?
```

E.1.9. ADDITIONAL LIB-80 FACILITIES

Modules in a library may be deleted or replaced in a single command. The names of the modules to be affected are enclosed in angle brackets immediately following the name of the source file containing the modules. The following examples demonstrate the use of this feature.

```
lib newlib=oldlib<mod1>

lib newlib=oldlib<mod1=file1>

lib newlib=oldlib<mod1=>

lib newlib=oldlib<mod1,mod2=file2,mod3=>
```

In the first case, a new library NEWLIB.REL is created which is the same as OLDLIB.REL except that the module MOD1 is replaced by the

contents of the file MOD1.REL. This form should be used if the name of the module being replaced is the same as the filename of the REL file replacing the module.

In the second case, the module MOD1 is replaced by the contents of the file FILE1.REL in the new library NEWLIB.REL. This form is used to replace a module when the name of the module is not the same as the name of the file which is to replace it. Note that this form must be used if the filename has more than 6 characters, since module names in the REL file are truncated to 6 characters.

When the third command is used, NEWLIB.REL is created from OLDLIB.REL without the module MOD1.

The last command form demonstrates that a number of replace and/or delete instructions may be included within the angle brackets.