File No. 7090-27
Form C28-6331

IBM Systems Reference Library

IBM 7090/7094 Programming Systems
IBJOB Processor
Overlay Feature of IBLDR

This publication supplies the programmer with the information necessary to use the Overlay feature of the Loader, IBLDR. IBLDR is the component of the IBJOB Processor that is responsible for loading jobs. The material contained in this publication is intended for the experienced programmer.

The Overlay feature of IBLDR answers the need for an effective way to run jobs that exceed the capacity of a single core storage load. This is accomplished by having one or more links of a job in core storage at any one time; these links may then be overlaid with other links, as required.

It is assumed that the reader is familiar with the contents of the following publications:

IBM 7090/7094 Programming Systems: IBJOB Processor, Form C28-6275.

IBM 7090/7094 Operating Systems: Basic Monitor (IBSYS), Form C28-6248.

The machine requirements necessary to use the Overlay feature of the Loader are the same as those specified for the 7090/7094 IBJOB Processor.

MAJOR REVISION (May, 1963)
This publication is a major revision of Form J28-6305.
It replaces the previous publication and obsoletes it.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the content of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N.Y.

INTRODUCTION TO OVERLAY

Overlay is a method of core storage utilization by jobs that exceed the capacity of core storage. The programmer divides the job to be executed by the overlay method into links. A link is one or more decks of a job. One of these links, called the main link, is loaded directly into core storage and remains in core storage throughout the execution of a job along with the Overlay subroutine and the tables required for execution. The Loader writes the other links, called dependent links, on some external file. A dependent link is one that is usually in core storage only at the time it is being used. It can be overlaid by other dependent links.

Each deck within a link can be called, or referred to, by the standard MAP CALL pseudo-operation.

All links, except the main link, are written on external files in scatter-load format and have a block size of 464 words.

The files available for link storage may be assigned to either disk storage or magnetic tape by using the methods of system unit assignment described in the publication, IBM 7090/7094 Operating Systems: Basic Monitor (IBSYS), Form C28-6248.

THE OVERLAY STRUCTURE

The MAP CALL pseudo-operation is the only operation that will induce overlay. Therefore, FORTRAN and COBOL statements which result in a MAP CALL to another deck can be used to induce overlay.

An example of the structure of an overlay job is illustrated in Figure 1.

In Figure 1, the vertical lines represent a link, or links, into which the program has been divided. Each link may contain one or more decks. The horizontal lines indicate the logical origin of the links. Link 0 is the main link, and remains in core storage at all times; the other links are stored on some external file.

Overlay can be induced only by the execution of a CALL from a link that is presently in core storage to a link that is not in core storage. When a CALL statement is executed in a link to any of the decks contained in another link, the incoming link replaces the link in core storage that has the same logical origin as the incoming link, and will also overlay all deeper links in the same chain below that logical origin. A chain is a sequence of links in core storage from the deepest link required, through whatever links precede it, to the main link. It is assumed that the normal way of terminating the execution of a deck in a dependent link will be with the RETURN statement.

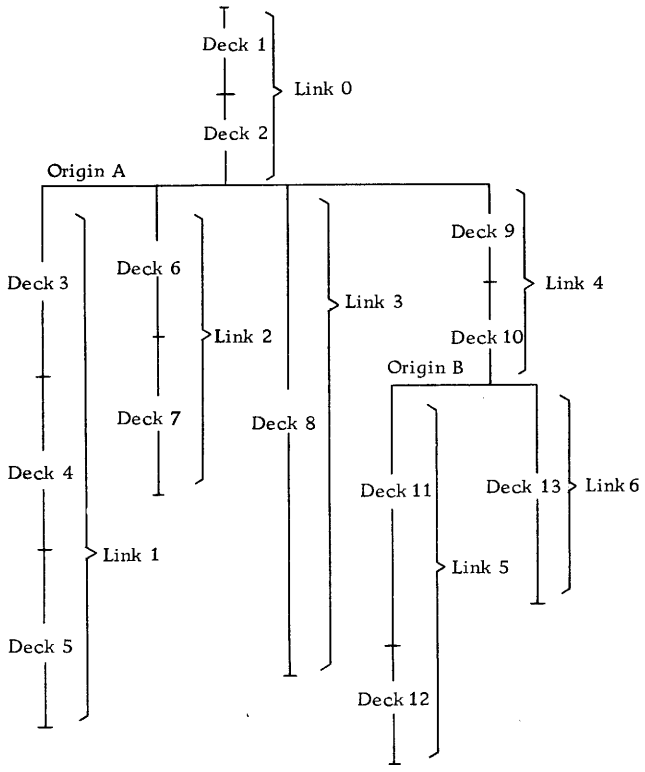


Figure 1

The CALL Statement

The primary rule regarding overlay-including CALL statements is the following:

No deck may either directly or indirectly call for itself to be overlaid.

The following types of CALL statements are valid:

1. A CALL towards a deeper link in the same chain is permissible. This type of CALL may or may not induce overlay, depending on whether or not the link has already been loaded into core storage.
2. A CALL within a link is always permissible; this type of CALL does not induce overlay.

3. A CALL from a deeper link to decks within the same chain of links toward the main link is permissible, provided the called deck or decks which it calls do not cause the originating deck to be overlaid.

If an invalid CALL statement is used, the job will be prevented from execution, unless the NOFLOW option is specified in the variable field of the $IBJOB card.

Since Overlay structure is defined at load time, all overlay-inducing CALL statements must be defined at load time. Therefore, a CALL statement of the form

```
CALL     **
```

where the address is supplied at execution time, cannot be expected to initiate overlay.

Referring to the Overlay structure in Figure 1, the following examples may be given:

1. A CALL from Deck 3 to Deck 4.
   Permitted: A CALL within the same link will never induce overlay.
2. A CALL from Deck 2 to Deck 12.
   Permitted: This CALL from Link 0 to Link 5 will initiate the loading of Links 4 and 5.
3. A CALL from Deck 12 to Deck 9.
   Permitted: This CALL is in the chain of links toward the main link.
4. A CALL from Deck 13 to Deck 3.
   Not Permitted: This CALL would induce the loading of Link 1, thereby overlaying Link 6, which contains the deck in which the CALL statement originated.
5. A CALL from Deck 11 to Deck 9, followed by a CALL from Deck 9 to Deck 13.
   Not Permitted: The CALL from Deck 11 to Deck 9 is valid, but, since Deck 9 contains a CALL to Deck 13, Link 6 would overlay Link 5, which contains the calling deck.

Virtual Control Sections

During the analysis of virtual CALL control sections, virtual control sections other than the CALL type are also checked for validity. Since this type of virtual reference cannot induce overlay of a link but may cause an error if a section that is not in core storage is referenced, the following rules should be considered:

1. A reference to a control section in a deeper link in the same chain is permissible, but may be in error if the deeper link has not been loaded into core storage. If the LOGIC option has been specified on the $IBJOB card, a warning message will be printed.
2. A reference within a link is always permissible.
3. A reference from a deeper link to a control section within the same chain of links toward the main link is always permissible.
4. A reference to a section that is not in the allowable chain of links is not permitted since, by the definition of overlay structure, the section referenced would not be in core storage. The NOFLOW option will permit this type of reference, if desired.

Storage Allocation During Execution

Figure 2 illustrates how the Overlay structure in Figure 1 would be assigned to core storage. Links having the same logical origin will be loaded starting at the same absolute location, unless the programmer has specified an absolute loading address for one or more links.

Library subroutines in the main link will be loaded following the input decks which constitute the main link. The input/output buffers will occupy the unused core storage area between the longest possible link configuration and the highest available core storage location. The FORTRAN COMMON area, if used, will be assigned following the library subroutines.

In Figure 2, the possible configuration of links in core storage at any one time is:

1. Link 0 (main link) only.
2. Link 0 and Link 1.
3. Link 0 and Link 2.
4. Link 0 and Link 3.
5. Link 0 and Link 4.
6. Link 0, Link 4, and Link 5.
7. Link 0, Link 4, and Link 6.

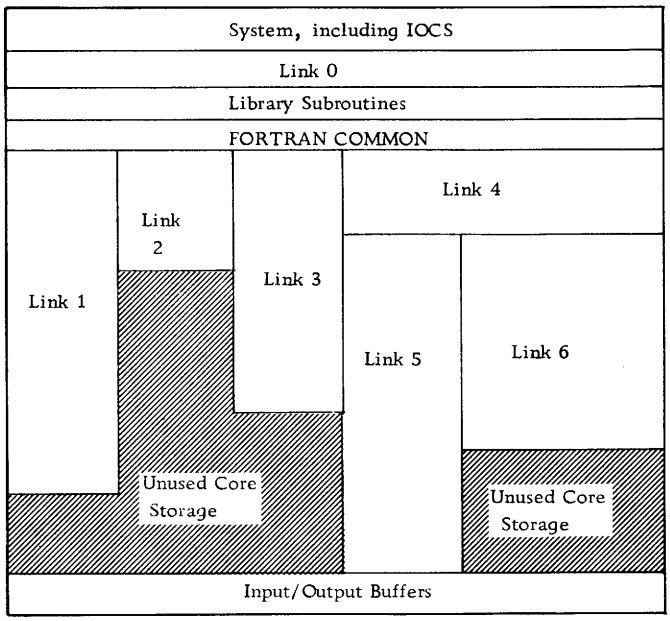


Figure 2

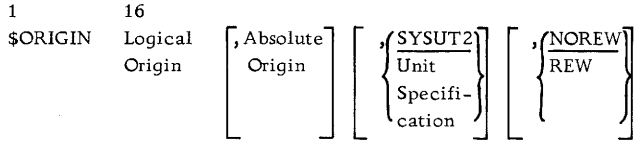
CONTROL CARDS

The Overlay feature of IBLDR primarily involves two control cards: $ORIGIN and $INCLUDE. The $ORIGIN control card is used to specify the logical origin of links, thereby indicating which links can overlay another. The $INCLUDE control card is used to specify that a deck (or any control section) be loaded in some link other than one in which it would normally be assigned.

The order in which options specified on control cards are exercised is not significant, unless otherwise specified.

$ORIGIN Control Card

The logical origins that are specified on $ORIGIN control cards govern the structure of an Overlay deck. The decks appearing first in the program are assigned to the main link, and are usually not preceded by a $ORIGIN control card. However, if a $ORIGIN control card is used to designate the main link, the logical origin specified by this card cannot be used on succeeding $ORIGIN control cards or an error condition will result, since this specifies more than one main link.



This control card initiates an Overlay link for the decks that follow. Decks following the $ORIGIN control card will be assigned to the same link until the occurrence of another $ORIGIN control card, a $ENTRY control card, and/or an end of file.

All pertinent information must be on this card; the $ETC control card may not be used to extend the variable field information.

The following list indicates the options that may be specified on the $ORIGIN control card:

Logical Origin — This field must be the first subfield in the variable field, and must be assigned a value. The value assigned is any arbitrary string of up to six characters, at least one of which is non-numeric and none of which is blank or among the following six special characters: ( ) = , / ' .

Absolute Origin — This field contains from one to five numeric characters specifying an absolute location at which the link is to be loaded. If the number is expressed in octal, an alphabetic O must precede the number. This field is used only if a program requires that a link be loaded at a specific location; it has no effect on the Overlay structure. It merely determines the loading point for this particular link and all links proceeding from it, if their $ORIGIN control cards do not specify an absolute origin.

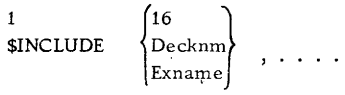
Unit Specification Option — This field specifies the input/output device on which the link is to be written. Any of the following seven system units may be specified:

SYSUT2 (or UT2)	SYSLB4 (or LB4)
SYSUT3 (or UT3)	SYSCK1 (or CK1)
SYSLB2 (or LB2)	SYSCK2 (or CK2)
SYSLB3 (or LB3)	

If the field is omitted, SYSUT2 is assigned. It is assumed that the unit chosen is in READY status and that it will not be used for any purpose other than loading links during job execution.

Rewind Option — NOREW specifies that the input/output unit containing the link is not to be rewound after the link is loaded; REW specifies that it is to be rewound. If this field is omitted, the unit will not be rewound.

$INCLUDE Control Card



This card specifies that the decks and/or the control sections named in the variable field are to be included in the link in which this card appears rather than in the link to which they would normally be assigned.

The subfields of the variable field contain alphameric literals which specify either a deck name (usually a library subroutine) or a real control section name of non-zero length (usually a block of data or coding) to be included in this link.

If a library subroutine is specified, the deck name of the subroutine (and not one of its entry points) must be given. Library subroutines will be automatically placed in the main link so that they will be available to all subsequent links. A library subroutine may, however, be assigned to a dependent link by means of a $INCLUDE control card. A subroutine or control section cannot be loaded in more than one link. If it is called from more than one link, it must be loaded in a link that is available to all calling links.

The following library subroutines may not be specified on a $INCLUDE control card. These subroutines must always be in the main link:

. FPTRP Floating Point Trap Subroutine
. LXCON Execution Control Subroutine
. LOVRY Overlay Link Loading Subroutine

The variable field of a $INCLUDE control card may be extended over more than one card, using either the $ETC control card or another $INCLUDE control card. The $INCLUDE control card may appear immediately following the $ORIGIN control card specifying the link, between the decks within the link, or immediately following the last deck of the link; the only restriction is that it may not appear within a deck in a link.

$IBJOB Control Card

An additional Overlay option, FLOW/NOFLOW, may appear on the $IBJOB control card. FLOW (the standard case) specifies that the program will not be executed if a CALL statement exists in a link that will cause itself to be overlaid or if a reference is made to a control section not in the permissible chains of links. NOFLOW specifies that execution will be permitted in this case.

CONTROL CARD USAGE

Figure 3 illustrates how a deck would be set up to produce the program structure given in Figure 4.

In Figure 3, the $ORIGIN control card which first uses logical origin ALPHA immediately follows the main link (decks 1 and 2). All links using logical origin ALPHA, therefore, proceed from the main link. Every new logical origin encountered on a $ORIGIN control card specifies that all links using this logical origin will proceed from the previous link. The $ORIGIN control cards containing the logical origins BETA and GAMMA are placed after the links from which they proceed. In this manner, the $ORIGIN control cards are used to form the Overlay structure.

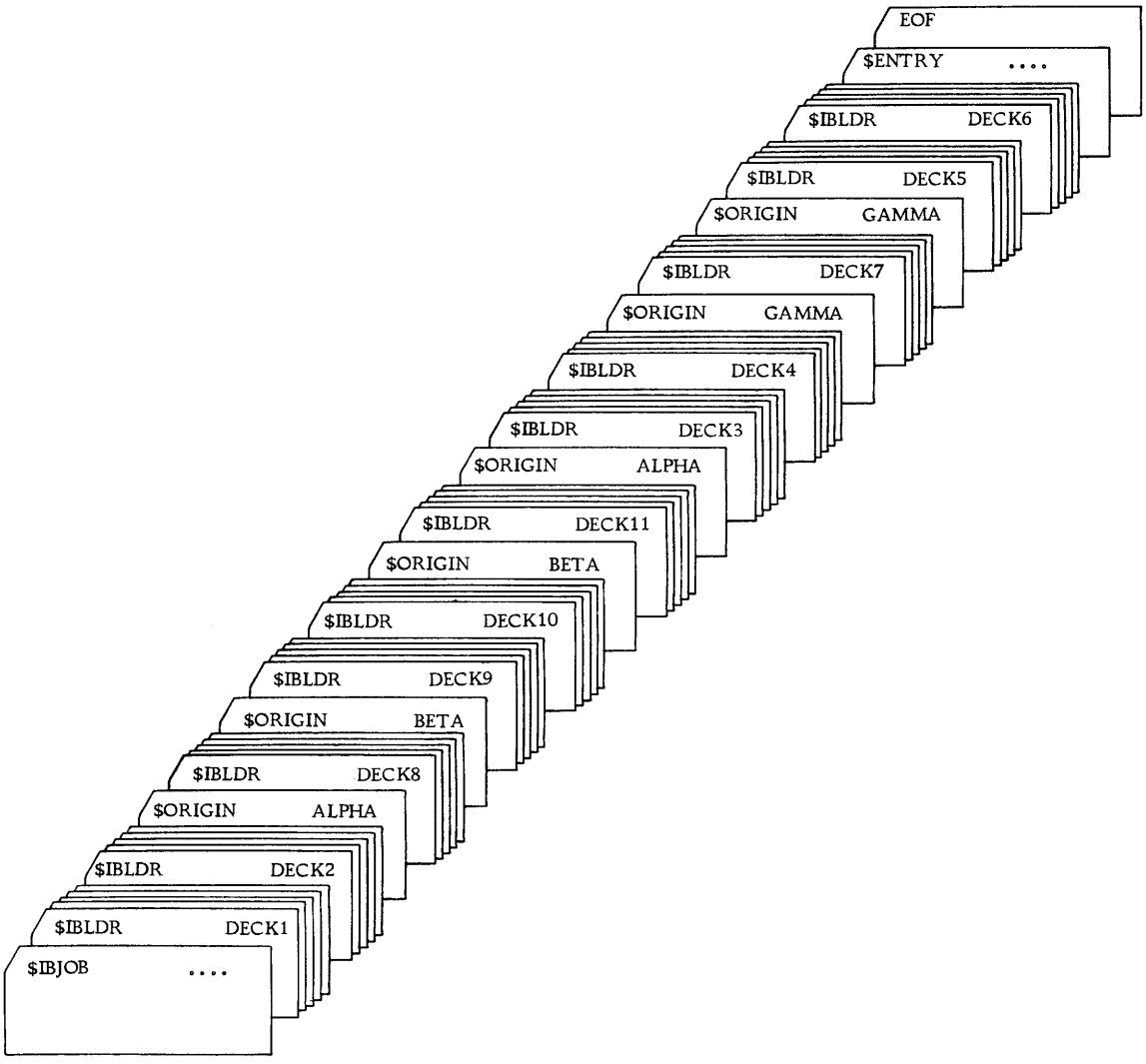


Figure 3

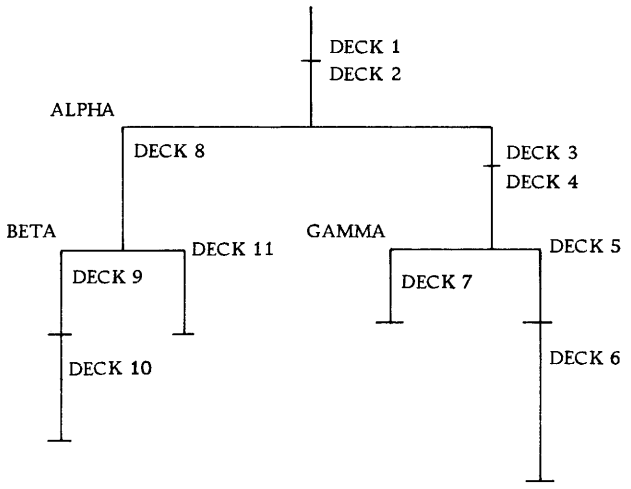


Figure 4

The following examples are given to aid the programmer in the use of Overlay control cards. To include the subroutine FLOG and the control section XYZ in the link which contains Deck 1, the following sequence could be used:

```
$ORIGIN          ALPHA
$INCLUDE         FLOG, XYZ

Deck 1
  .
  .
  .
  .
```

When a deck or section is assigned to a link by means of a $INCLUDE control card, care must be taken that the link incorporating the deck or control section be available to all other links that refer to or call the deck or section.

If a $INCLUDE control card is used to move a block of instructions or data from a deck to some other link, it is possible to cause the external link file to be written in a format which cannot be used efficiently during execution.

For example in the following sequence:

```
$ORIGIN      A, SYSUT3                Link A
$INCLUDE     XYZ
   :
   :
   :
$ORIGIN      B, SYSUT3
$IBLDR       DECK 1                   Link B
$IBLDR       DECK 2
             (contains section XYZ)
   :
   :
   :
```

the instructions or data in section XYZ which are to become part of link A will not be encountered by the Loader for processing until after link A and a portion on link B have been written onto SYSUT3. Therefore, on SYSUT3, the information in section XYZ will be isolated from the main portion of link A. If SYSUT3 is a tape file, some tape will have to be spaced over when loading link A in order to load the XYZ portion. This situation can usually be avoided by specifying a unique unit for the storing of the link which contains the $INCLUDE control card.

It should be noted that the condition above occurs only in the special case where the section of the $INCLUDE control card is internal to some deck and contains text. This condition will not occur when assigning library subroutines or control sections which do not contain text to other links by means of the $INCLUDE control card.

Overlay Logic Messages

If the LOGIC option has been specified on the $IBJOB control card, all CALLS and references to sections or entry points in different links are tabulated. The following are examples of this tabulation:

```
LINK 5 DECK 'XXXXXX' CALLS TO LINK 4 DECK 'YYYYYY'
SECTION 'ENTRY1' UPWARD CALL.
```

This message specifies a CALL exists up the chain of links toward the main link.

```
LINK 3 DECK 'A' CALLS TO LINK 7 DECK 'B' SECTION 'ENT'
DOWNWARD CALL (VIA TRANSFER VECTOR)
```

This message specifies that the CALL may induce the loading of a link.

```
LEVEL = 3
LINK 4 DECK 'C' CALLS TO LINK 5 DECK 'G' SECTION 'START'
IMPROPER CALL
```

This message specifies that the CALL is not within the allowable chain of links.

```
LINK 7 DECK 'CALIF' CALLS TO LINK 8 DECK 'K' SECTION
'ENTRY2' IMPROPER CALL (NOFLOW OPTION ALLOWS)
```

This message is the same as message 3 but execution will be allowed.

```
LINK 0 DECK 'MNP' REFERS TO LINK 3 DECK 'CK2' SECTION
'TABLE'. (MAY CAUSE ERROR IF LINK 3 NOT LOADED.)
```

This message warns of a possible error condition.

CALLS and references to and from decks within the same link are not tabulated.

Messages pertaining to improper calls and references are always printed if the FLOW option has been specified. If NOFLOW has been specified, the message will not be printed (unless LOGIC is requested as explained above).

For CALL errors of a more serious nature, a message such as the following may be printed.

```
LEVEL = 3
IMPROPER CALL STRUCTURE WILL CAUSE A CALLING DECK
"ABC" LINK 9 TO BE OVERLAID BY LINK 7 BEFORE RETURN IS
MADE FROM A CALL TO ENTRY POINT "BEGIN". THE CALL
FLOW, IN REVERSE ORDER IS:
    DECK 'SCAN' LINK 7, IS CALLED BY
    DECK 'MAS' LINK 2, IS CALLED BY
    DECK 'ABC' LINK 9
```

This message tabulates the flow of CALLS, in reverse order, from the deck which would actually cause the improper overlay back to the original calling deck. In this example, the CALL flow is from link 9 to link 2 to link 7. This message is printed only if the FLOW option is specified on the $IBJOB control card.

$ORIGIN Control Card Error Messages

The messages are preceded by the offending control card.

Error Severity = 1, Minor errors

1. OPTIONS OTHER THAN ABSOLUTE ORIGIN IGNORED ON MAIN LINK $ORIGIN CARD. Unit specification and NOREW or REW options have no meaning for the main link.
2. PARAMETER 'XXXXXX' NOT RECOGNIZED. IGNORED

Error Severity = 3, NOGO errors

1. ILLEGAL CHARACTER. REMAINDER OF CARD IGNORED
   The card contains one of the characters ( ) = / '
2. IMPROPER FORMAT. Leading, trailing or multiple field separators in variable field.
3. NO SYMBOLIC ORIGIN SPECIFIED. Blank variable field or unit specification or REW/NOREW option appears starting in col. 16.
4. IMPROPER SYMBOLIC ORIGIN. Logical Origin is all numeric or greater than 6 characters.

$INCLUDE Control Card Error Messages

The messages are preceded by the offending control card.

Error Severity = 3, NOGO errors

1. ILLEGAL CHARACTER, REMAINDER OF CARD IGNORED
   The card contains one of the characters ( ) = / '
2. IMPROPER FORMAT. Leading, trailing or multiple field separators in the variable field.
3. SECTION OR DECK 'XXXXXX' HAS BEEN SPECIFIED TO BE ASSIGNED TO MORE THAN ONE LINK. Section or deck appears on more than one $INCLUDE card.
4. UNDEFINED SECTION OR DECK NAME 'XXXXXX'. Section or deck is not defined in object deck or subroutine library.
5. LEVEL = 3
   SECTION NAME BEGINNING '******' IS GREATER THAN 6 CHARACTERS.
   IGNORED

If the entry point specified on a $ENTRY control card is not in the main link, the message ENTRY POINT SPECIFIED NOT IN MAIN LINK is printed.

Termination of Loading

If an error of severity level 3 on a $ORIGIN or $INCLUDE card is detected, loading will be terminated after all control cards are examined.

Level = 4 LOADING TERMINATED DUE TO IMPROPERLY DEFINED OVERLAY STRUCTURE.

APPENDIX: THE CALL TRANSFER VECTOR

During loading, an analysis is made of all CALL statements in the program. If a CALL statement will induce overlay, the transfer address of the CALL statement is modified to refer to a transfer vector of the form

```
PFX     entry point, , link number
TXI     . LOVRY
```

For example, the statement

```
CALL    . SUBPR
```

may be modified by the Loader to

```
CALL    . TV001
```

and, at location . TV001, the Loader will generate the two words

```
PFX     SUBPR , link number
TXI     . LOVRY
```

This transfer vector is constructed by IBLDR and is stored with the object program in a generated control section called . LVEC. During execution, if the called deck has been loaded into core storage, PFX was set to TSL and a transfer would now be made to the entry point. If the called deck has not been loaded into core storage, PFX would be set to TXH, and a transfer made to the linkloading subroutine, . LOVRY. This subroutine will load the required links and reset all the transfer vector words involved to properly indicate the load status of the links.

If LOGIC is specified on the IBJOB card, the absolute location of . LVEC will be indicated in the logic listing.

Printed in U.S.A. C28-6331

IBM
International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, New York