File No. 7030-27
Form C28-6379-0

IBM Systems Reference Library

IBM 7030 Data Processing System Bulletin

Loader and BSS Processor

This publication describes the 7030 Master Control Program (MCP) loader and the FORTRAN Binary and Symbolic Subroutine (BSS) processor. The MCP loader is capable of processing relocatable or absolute binary decks. The BSS processor allocates storage, calls subprograms from the FORTRAN library, and completes subprogram cross references. A segmentation program, which provides a method of storage overlay, is included in the BSS processor.

PREFACE

The purpose of this document is to provide a programmer's guide to loader functions. Examples of various deck arrangements are given in a separate section. Appendixes containing card formats and explanations of diagnostic messages are also provided. Considerable detail has been provided in the section on segmentation to assist a reader interested in the depth of the overlay feature of the IBM 7030 system.

The discussion in this bulletin assumes the understanding of the following publications:

Reference Manual, IBM 7030 Data Processing System, Form A22-6530.
Reference Manual, IBM 7030 Master Control Program, Form C22-6678.
Reference Manual, IBM 7030 FORTRAN IV, Form C22-6751.

The segmentation portion of the BSS processor described in this bulletin was programmed by the Naval Weapons Laboratory, Dahlgren, Virginia.

Copies of this and other IBM publications can be obtained at IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D83, PO Box 390, Poughkeepsie, N.Y. 12602

The IBM 7030 system loader consists of two programs: (1) the resume load program of MCP, with the loader as a subroutine, and (2) the Binary and Symbolic Subroutine (BSS) processor. These two programs allow the programmer to submit an input deck containing subprograms written in FORTRAN (which produces relocatable binary cards) or STRAP (which produces relocatable or absolute binary cards, as specified), or to submit a mixture of FORTRAN and STRAP subprograms. The programmer may receive automatic storage allocation by specifying that he wishes to operate in the FORTRAN environment. In this case, the BSS processor assigns storage, processes relocatable and absolute binary input, calls specified routines from the FORTRAN library, and computes all subprogram cross references. The BSS processor gives control to the resume load program and provides information necessary to complete the loading function.

The programmer may specify that he wishes to operate in a non-FORTRAN environment. In this case, the BSS processor is not involved, and control is passed to the resume load program. Storage allocation for relocatable subprograms may be specified by control cards which are meaningful to the loader. No storage allocation is necessary for absolute subprograms, whose locations are specified on the cards.

MCP LOADER

The resume load program of MCP may be used by the problem programmer who wishes to load and execute his program in sections. In a non-FORTRAN environment, this procedure is accomplished with the following calling sequence:

```
B, $MCP
, $RESLD
```

Loading is resumed with the next binary card from the same source as the previous load (system input or disk). Loading is terminated when the next absolute branch card is encountered, and control is given to the address specified on the branch card.

Care should be taken when $RESLD is issued in an I-O fix-up routine. The pseudo-operation does nothing to alter any I-O stacking mode that may be in effect, so that the next section of the job could be given control in the auto-stacked mode. (See Reference Manual, IBM 7030 Master Control Program, Form C22-6678, for a description of the auto-stacked mode.) In this situation the first $RET pseudo-operation issued by the second section would attempt to return control to a location in the first section.

In a FORTRAN environment jobs may be segmented with the use of the segmentation feature of the BSS processor. (Refer to the Segmentation section.)

The resume load program also serves as the final phase of FORTRAN relocatable BSS loading. In this capacity, resume load obtains the loader limits of the problem program from the communication region of MCP (previously set up by the BSS processor) and sets them to extend from the lower limit to the base address of blank common, inhibiting loading into blank common. Resume load then sets the input source to disk, initializes the loader to take information from the relocation tables set up by the BSS processor, and continues the loading function.

In a non-BSS environment, problem program storage is cleared, upon initial entry to resume load, from location 33.0 to the upper limit of the problem program. In a BSS environment, problem program storage is not cleared.

The MCP loader operates as a subroutine of the resume load program. After loading a card, control is returned to resume load by way of normal return (which tries to load another card), branch return (which gives control to the problem program), or error return (which terminates loading and gives a diagnostic message).

The loader handles 18 classes of cards, identified by column 1 punches:

1.	Absolute origin	(7, 8, 9)
2.	Absolute flow	(7, 9)
3.	Absolute branch	(6, 7, 9)
4.	Absolute correction (C)	(12, 3)
5.	Absolute patch (P)	(11, 7)
6.	Absolute dump (D)	(12, 4)
7.	Relocatable data	(6, 7, 8, 9)
8.	Relocatable instruction	(5, 7, 9)
9.	Fortran program	(5, 6, 7, 9)
10.	Common definition	(5, 7, 8, 9)
11.	Fortran branch	(5, 6, 7, 8, 9)
12.	Relocatable correction (K)	(11, 2)
13.	Relocatable patch (A)	(12, 1)
14.	Relocatable dump (Z)	(0, 9)
15.	T	(0, 3)
16.	Super T	(0, 2, 3)
17.	Loader adjustment (O)	(11, 6)
18.	B	(12, 2)

The formats for each of the above cards may be found in Appendix A. The function of each card in the MCP loader is described below:

1. Absolute origin card. Up to 23 half-words of information are loaded, beginning at the origin specified on the card. Bits may be skipped or zeroed before or after loading. The sequence number and check sum are verified.

2. Absolute flow card. Exactly 25 half-words of information are loaded, beginning immediately after the highest location loaded on the previous origin or flow card. If no previous origin is found, a diagnostic message is issued. The sequence number and check sum are verified.

3. Absolute branch card. The sequence number and check sum are verified. The branch address is checked to see if it is within bounds. If there is no branch address, the origin found on the first origin card loaded is used as the branch address. Control is passed, by way of the branch return in resume load, to the branch address.

4. Absolute correction card. Up to four half-words of information are loaded, beginning at the origin specified on the card. If there is no origin on the correction card, the card is interpreted as a continuation card, from which up to four half-words of information are loaded, beginning at the location following the highest location loaded from the previous card. A correction card is ignored by the loader if a 1 punch is found in the decimal point column of the origin. A half-word correction is ignored if a 1 punch is found in the decimal point column for the half-word. The loading address is stepped, although the half-word correction is ignored by the loader (except when the half-word ignored is the last half-word on the card, in which case the loading address is not stepped).

5. Absolute patch card. The instruction at the origin specified on the card is replaced by a branch (or a branch; NOP, depending on the length of the instruction replaced) to a patch area, which starts at the lowest location above all previously loaded locations. The upper limit of the program may have to be extended by the programmer to allow space for the patch area. At the patch area the following is placed: (1) the instruction replaced by the branch to the patch area, (2) the half-words which appear on the patch card, (3) a branch instruction which transfers to the next instruction after the one which was replaced by the branch to the patch area. If there is no origin on the patch card, the card is interpreted as a continuation patch card, from which up to four half-words are loaded beginning at the location following the highest location in the patch area loaded from the previous card.

6. Absolute dump card. The absolute dump card has the same instruction displacement property as the absolute patch card. However, the loader places a calling sequence to the MCP dump routine in the patch area. The first and second half-words on the dump card are inserted as the lower and upper limit dump parameters, and the third half-word is inserted as the format parameter. The dump calling sequence is followed by a branch to the instruction that follows the one which appears at the origin on the dump card, in the same manner as a patch. More than one dump request may be given for the same location, by placing the lower limit for the second request as the fourth half-word on the card and continuing the upper limit and format, and subsequent dump requests, on continuation absolute dump cards (characterized by having no origin on the card). (See dump card format in Appendix A.)

7. Relocatable data card. The relocatable data card contains data information (up to 744 bits) and a count of the number of bits to be skipped or zeroed before or after loading. Data may be loaded into problem program storage or into a common block in a FORTRAN environment. (Common blocks are meaningful to the loader only in the FORTRAN environment.) The card is checked to insure that the data will be loaded within bounds. The origin of the subprogram and the origins of common blocks are taken from the relocation table in a FORTRAN environment; the origin of the subprogram in a non-FORTRAN environment is taken from the preceding loader adjustment card.

8. Relocatable instruction card. A variable amount of information is loaded. (The amount of information that can be punched on a card depends on the number of relocation bits necessary to describe the instruction on the card.) The origin of the subprogram is taken from the relocation table in a FORTRAN environment and from the previous loader adjustment card in a non-FORTRAN environment. The absolute origin is computed by adding the subprogram origin to the relative origin on the card. Addresses are computed from the relocation bits on the card, using the relocation table for loading bases.

9. FORTRAN program card. The FORTRAN program card furnishes information to the BSS processor regarding the size of blank common and names of entry points. When the MCP loader encounters a FORTRAN program card, it takes the origin of the subprogram and the number of cards in the subprogram from the relocation table and computes the number of relocation bits to describe the number of named commons within the subprogram. A FORTRAN program card is accepted by the MCP loader only in a FORTRAN environment.

10. Common definition card. The common definition card contains common names and sizes, information used by the BSS processor. A common definition card is not accepted by the MCP loader in a non-FORTRAN environment, and is ignored in a FORTRAN environment.

11. FORTRAN branch card. The FORTRAN branch card is used to terminate the input phase of the BSS processor. It is ignored by the MCP loader.

12. Relocatable correction (K) card. Up to three half-words are loaded, beginning at the effective origin formed by relocating the relative origin on the card with respect to the subprogram or the named common specified. Each half-word is relocated according to the relocation columns on the card. (See also card formats - Appendix A.) If there is no origin on the correction card, the card is interpreted as a continuation card, from which up to three half-words of information are loaded, beginning at the location following the highest location loaded from the previous card. A correction card is ignored by the MCP loader if a 1 punch is found in the decimal point of the origin. A half-word correction is ignored if a 1 punch is found in the decimal point column for that half-word. The loading address is stepped whether or not the half-word correction is ignored by the loader (except if the half-word ignored is the last half-word on the card). It should be noted that a relocatable correction card at an origin higher than the highest origin in the subprogram has the effect of resetting the patch area to the highest origin encountered.

13. Relocatable patch (A) card. The effective origin is formed by relocating the relative origin on the card with respect to the subprogram origin. The instruction at the effective origin is replaced by a branch (or a branch; NOP, depending on the length of the instruction replaced) to a patch area at the end of the subprogram. The size of the subprogram is extended when storage allocation takes place in BSS, so that storage for relocatable patch cards is automatically provided in a FORTRAN environment. In a

non-FORTRAN environment, however, no space is provided for relocatable patch cards, and the limit must be extended by the programmer. It may be desirable to extend the size of a subprogram or to force the patch area to begin at a location higher than the highest location loaded by the subprogram. This extension may be accomplished by a relocatable correction card at a relative origin higher than the highest location within the subprogram. At the patch area, up to three half-words which appear on the card are placed after the half or full-word instruction replaced by the branch or the branch; NOP. The instructions making up the patch are followed by a branch instruction that transfers to the next instruction after the one that was replaced by the branch to the patch area. If there is no origin on the patch card, the card is interpreted as a continuation patch card, from which up to three half-words are loaded, beginning at the location following the highest location in the patch area loaded from the previous card. Each half-word from a relocatable patch or continuation relocatable patch card is relocated according to the relocation columns on the card. (See also card formats - Appendix A.)

14. Relocatable dump (Z) card. The effective origin is formed by relocating the relative origin on the card with respect to the subprogram specified (or the named common, although it seems unlikely that the origin would be relocated with respect to named common). The relocatable dump card has the same instruction displacement property as the relocatable patch card. However, the loader places in the patch area a calling sequence to the MCP dump routine, and inserts parameters from the card in the same manner as from an absolute dump card, with the additional feature that the parameters are relocated according to the relocation columns on the card. (See also card formats - Appendix A.) It should be noted that, like the relocatable correction and patch cards, only three half-words are allowed. Additional dump requests at the same location may be given by using continuation relocatable dump cards, characterized by the absence of a relative origin on the card. Storage allocation for the relocatable dump card is similar to that for the relocatable patch card; i.e., storage is automatically allocated in a FORTRAN environment only.

15. T card. No information is loaded from a T card. When the next binary card is encountered, its sequence number is not checked. Instead, the sequence counter is reset to that sequence number. T cards are printed by either the MCP loader or the BSS processor.

16. Super T Card. A Super T card has the effect of stopping sequence checking for the entire job. Super T cards are printed by either the MCP loader or the BSS processor.

17. Loader adjustment (O) card. In a FORTRAN environment, the loader adjustment card has no function in the MCP loader. In a non-FORTRAN environment, however, a FORTRAN program card is not accepted, and the loader adjustment card is used to set the origin for the subprogram that follows it. The origin on the card is checked to see if it is within bounds.

18. B card. A B card is ignored by the MCP loader.

The Binary and Symbolic Subroutine (BSS) processor operates as a preprocessor to the MCP loader in a FORTRAN environment. Its function is to create a relocation table, from which the MCP loader obtains all the information necessary to assign storage areas dynamically to FORTRAN relocatable subprograms. The BSS processor may run as the last link in the FORTRAN chain of processors, or, if the subprograms are all previously compiled, it may run as a single processor. In the latter case, it is called by the following TYPE card:

```
B      TYPE,GO,FORTRAN
```

Input to the BSS processor consists of a collection of subprograms, each of which begins with a FORTRAN program card. Input is from the disk if the BSS processor is operating as the final phase of the FORTRAN chain of processors, or from system input if the BSS processor is running as a single processor. In the latter case, the BSS processor places the subprograms on the disk, so that the MCP loader, in both cases, finds the input on the disk. The last subprogram must be followed by a FORTRAN branch card.

The BSS processor scans the cards in each subprogram and computes the amount of storage to be occupied by the program, including storage for relocatable patches and subprogram extension. Three intermediate tables are set up, containing:

1. Information from FORTRAN program cards.
2. Information about blank and named common.
3. Names of transfer vectors to be filled in. (See card types for details of information taken from each.)

If calls are made to library subprograms, the library routines are fetched from the library and scanned, and information from these cards is used to create entries in the three intermediate tables. The relocation table is created from the three intermediate tables. Absolute origins for subprograms, blank common, and named common are computed. The storage allocation is made as follows: blank common occupies the highest block of storage and named commons, the next highest blocks; subprograms are assigned consecutive blocks starting from address 33.0, in the order in which they are encountered by the BSS processor. Library subprograms are assigned higher locations than non-library subprograms, in the order in which calls to the library routines are encountered; and there may be unused storage between the highest subprogram and the lowest common block.

Card Order

The BSS processor requires a rigid ordering of cards within each subprogram:

A. One or more FORTRAN program cards.

B. One or more FORTRAN common definition cards.

C. A loader adjustment (O) card (if present).

D. One relocatable binary instruction card, containing an index word whose count field specifies the length of the transfer vector (number of distinct entry points to routines called by the subprogram).

E. One or more relocatable binary data cards, containing the transfer vector (A8 names of the entry points outside the subprogram to which transfers are made in the subprogram).

F. Subprogram deck (relocatable or absolute).

G. Correction, patch, or dump cards (relocatable or absolute).

No FORTRAN common definition cards are present unless there is a common defined. If the count field of the index word on the first relocatable binary instruction card is zero, no relocatable binary data card containing the transfer vector is expected to be present.

K, A, and Z cards are associated only with the subprogram which they follow. K, A, and Z cards may be intermixed, but K cards should precede those A or Z cards at the same location. Continuation K, A, and Z cards must immediately follow the cards which they continue.

The function of each card accepted by the BSS processor is described below. (The column 1 codes for the various card types are listed under the MCP loader.)

1. FORTRAN program card. The FORTRAN program card contains the entry point names and their relative origins in the subprogram. (See binary card formats in Appendix A.) The first FORTRAN program card contains, in addition, the size of blank common. The BSS processor sets up an entry in the first intermediate table (Figure 1) upon encountering a FORTRAN program card. Pointers to the second and third intermediate tables (Figures 2 and 3), and the entry point names and their origins relative to the subprogram, are placed in the first intermediate table. When the entry point name *MAIN* is encountered, its origin is placed in MCP's communication region, and is used as the location to which control in the problem program will be passed by the MCP loader.

The subprogram size on the FORTRAN program card is either (a) the size computed at assembly time (if STRAP generates the card) or (b) all 1's (if FORTRAN generates the card). The BSS processor computes the amount of space occupied by the subprogram in storage. The FORTRAN program card is also a signal to the BSS processor to place the computed size of the previous subprogram in that

First Intermediate Table

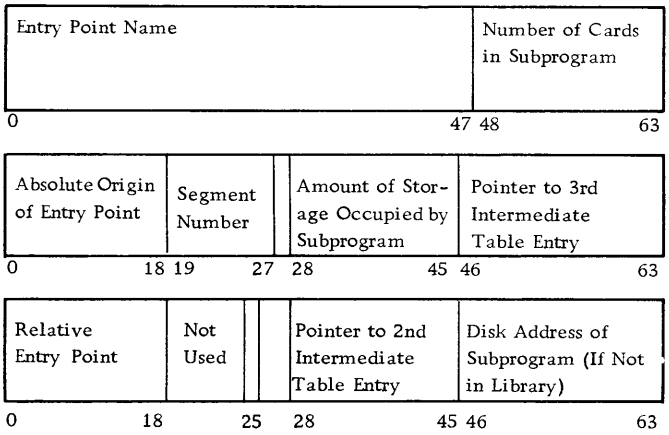


FIGURE 1. ENTRY IN THE FIRST INTERMEDIATE TABLE

Notes: For an absolute subprogram, the origin specified on the loader adjustment card will be placed in the first 19 bits of the second word.

Bit 25 of the third word is set to 1 if the entry is for a library routine.

subprogram's entry in the first intermediate table. If there are no entry points in a subprogram it must be a data subprogram, and the appropriate indication is made in the second intermediate table. The size of blank common is determined by the BSS processor to be the largest blank common size declared by any subprogram, and is updated, if necessary, in the entry for blank common in the second intermediate table.

2. FORTRAN common definition card. The FORTRAN common definition card contains the names and sizes of the named commons used by the subprogram. (See binary card formats in Appendix A.) If two or more subprograms declare the same named common, the largest size declared is assigned to that named common. The names and sizes of named commons are placed by the BSS processor in the second intermediate table, when the common definition card containing this information is encountered.

3. Loader adjustment card. The loader adjustment card is a signal to the BSS processor to allocate space for the following subprogram starting at the absolute origin specified on the card. (See binary card formats in Appendix A.) If there is a conflict between storage previously allocated (i.e., to a subprogram physically preceding the subprogram containing the loader adjustment card), the BSS processor indicates that an error has occurred. Subsequent subprograms are allocated space higher in storage than the one containing the loader adjustment card. The additional stipulation is made that every absolute binary subprogram must contain either a loader adjustment card or a T card to reset the sequence counter.

Second Intermediate Table

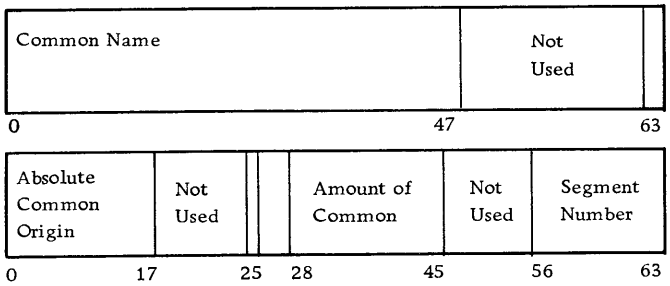


FIGURE 2. ENTRY IN THE SECOND INTERMEDIATE TABLE

Note: Bit 25 of the second word is set to 1 if the entry is for a block data subprogram common.

4. T cards. The only BSS processor function of a T card is to reset the sequence counter, except for two significant card types: define or define immediate cards, and NODE cards. The define and define immediate cards provide the ability to associate and dissociate named commons. The BSS processor reacts to these cards in the following manner: for a define card (see card format in Appendix A), the second common name appearing on the define card replaces the name of the first common named, in the second intermediate table. The restriction is made that the define card must not appear in the deck before the first common definition card declaring the name of the named common which is to be replaced. The define card may be used, then, to handle two distinct named commons as one. The effect of the

Third Intermediate Table

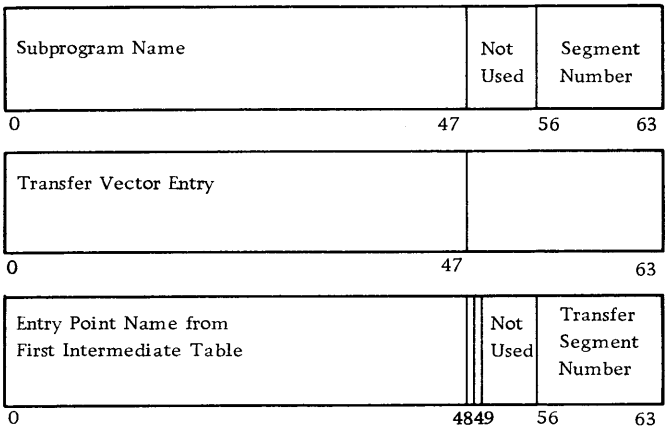


FIGURE 3. ENTRY IN THE THIRD INTERMEDIATE TABLE

Notes: Bits 48 and 49 of the third word are used for verifying legality of transfers between segments (segmentation feature).

The second word is replaced by a branch; NOP to the absolute transfer address.

define immediate card is opposite to that of the define card. A define immediate card is interpreted so that the second common named replaces the name of the first common named only within the subprogram in which the card appears. The second common named may be a new name, not occurring anywhere else in the deck. A define immediate card may be used, then, to dissociate two named commons intended to be dissimilar.

NODE cards are a signal to the BSS processor that the job uses the overlay feature of the system. (Refer to the Segmentation section.)

5. Relocatable binary instruction card. These cards are scanned by the BSS processor only to compute the size of each subprogram, except for the first relocatable binary instruction card, which must contain an index word in the first 64 bits. The length of the transfer vector (number of entry points to routines called by the subprogram) is contained in the count field of this index word.

6. Relocatable binary data card. These cards are scanned by the BSS processor only to compute the size of each subprogram, except for the data cards immediately following the first relocatable binary instruction card. The BSS processor saves the transfer vector from these data cards and makes an entry for each name in the third intermediate table.

7. Absolute binary cards and absolute correction, patch, and dump cards. These cards are ignored by the BSS processor.

8. Relocatable correction, patch, and dump cards (K, A, Z cards). These cards are scanned by the BSS processor in order to compute the size of each subprogram. A relocatable correction card containing an origin higher than the highest location used by the subprogram itself has the effect of extending the subprogram to the origin specified. Relocatable patch and dump cards cause the subprogram to be extended to allow space for the additional instructions or dump calling sequence.

9. FORTRAN branch card. The FORTRAN branch card is used to signal end-of-deck to the BSS processor. The job may not be run in the absence of such a card.

10. B cards. IOD cards and REEL cards cause the amount of storage available to the problem program to be reduced by the BSS processor, which allocates necessary space to IOD tables. All other B cards are ignored by the BSS processor.

11. Super T card. A super T card is a signal to the BSS processor to stop sequence checking on all binary cards for the entire job. Otherwise, the BSS processor verifies sequence numbers and computes check sums of all binary cards.

Upon finishing the scan of the input deck (encountering a FORTRAN branch card), the BSS processor tries to match the transfer names in the third intermediate table with the entry points in the first intermediate table. If a name is not found, the BSS processor tries to fetch that subprogram from the FORTRAN library on disk, and proceeds to set up tables for library material in the same manner as for the input deck. The search for matching transfer names continues until all entries in the third intermediate table are linked to the first intermediate table.

At this point, the BSS processor has all the information it needs to construct the relocation tables and allocate storage for the MCP loader. The BSS processor performs the following functions in connection with storage allocation:

1. Computes the upper limit of storage available for the job, the size of the relocation table, and the origin of blank common.
2. Places the origin of blank common in the MCP communication region.
3. Computes the absolute origins for the subprograms and places them in the first intermediate table.
4. Eliminates commons declared to have identical names and reserves space for the largest of the identically named commons.
5. Computes absolute named common origins and places them in the second intermediate table.
6. Checks for common and program overlap of space allocated.

The BSS processor constructs the relocation table from the three intermediate tables. One entry in the relocation table is created for each subprogram, and contains the following information: (1) subprogram origin, (2) number of named commons used, (3) absolute origins of named commons, (4) length of transfer vector (number of branches to other subprograms), (5) transfer vector (actual branch instructions to absolute storage locations), (6) pointer to the next entry in the relocation table, and either (7) address of the subprogram on the disk and the number of cards (if the subprogram is not a library subprogram), or (8) the entry point name (if the subprogram is a library subprogram).

The BSS processor leaves the relocation table in storage, beginning at the base address of blank common. The BSS processor completes its function by placing the main entry point address in the MCP communication region, sorting the relocation table entries so that library subprograms are loaded in the order in which they appear on the disk, and printing out a mapping of the storage allocation scheme. (See example 4.)

Control is passed from the BSS processor to MCP and then to the MCP loader, which loads the subprograms from the disk into storage, using information from the relocation tables.

SEGMENTATION

The function of the segmentation portion of the BSS processor is to provide a method of subprogram overlay for FORTRAN jobs too large to fit into available storage.

In order to use segmentation, the set of subprograms making up the input deck is first divided into groups (called segments) of subprograms. In front of each group of subprograms should be placed a NODE card which contains its segment number (any non-zero two-digit number) and the segment number of another segment to which the first segment listed on the card is attached. (See Appendix A for NODE card format.) One segment, called the primary segment, remains in storage at all times, and should contain the "main" subprogram (designated by having an entry point called *MAIN*), and the FIOD subprogram. The NODE card which precedes the primary segment should have as its second segment number 00, since it is attached to no other segment. All library routines are assigned storage in the primary segment.

Upper storage is allocated in a particular way in a job which requires segmentation. Since the relocation table (Figure 4) must remain in storage (except for the primary segment) during the execution of the object program, so that the table may be available for loading bases when another segment is called in, the relocation table is assigned storage higher than blank common. A node table (Figure 5), which contains information necessary for segmentation, extends from a storage location immediately above the relocation table to the top of storage available to the problem program.

Storage for named common is reserved below blank common. Whether a named common is available when a given segment is in storage depends on the following:

1. The given segment must be contained in all groups of segments making up a storage load that refer to that named common.
2. The given segment must have been assigned the highest position in storage, consistent with (1), that is not higher than the position of any segment which refers to that named common.(See Figure 6.)

Block data is associated with the segment that contains that block data subprogram. Therefore, all other segments that reference that block data must be in the same group of segments that contains that subprogram and must be allocated storage at a higher level.

Storage for named commons associated with segments which are assigned lower storage locations are assigned higher storage locations than named commons associated with segments which are assigned higher storage locations.

Relocation Table

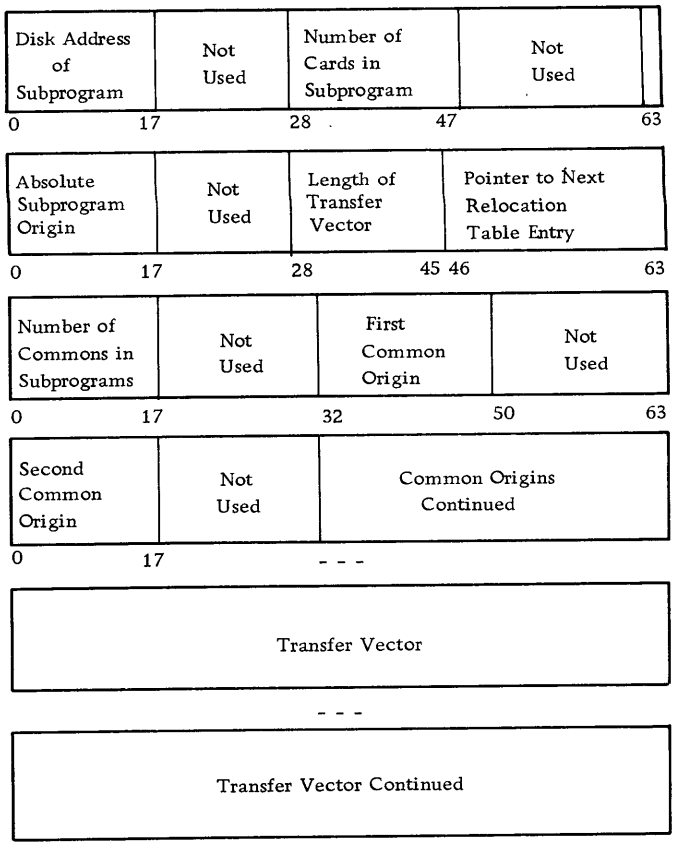


FIGURE 4. ENTRY IN THE RELOCATION TABLE

Notes: Bit 63 of the first word is set to 1 if the entry is for a library subprogram.

The first 48 bits of the first word contain the entry point name if the entry is for a library subprogram.

Node Table (Segmentation Only)

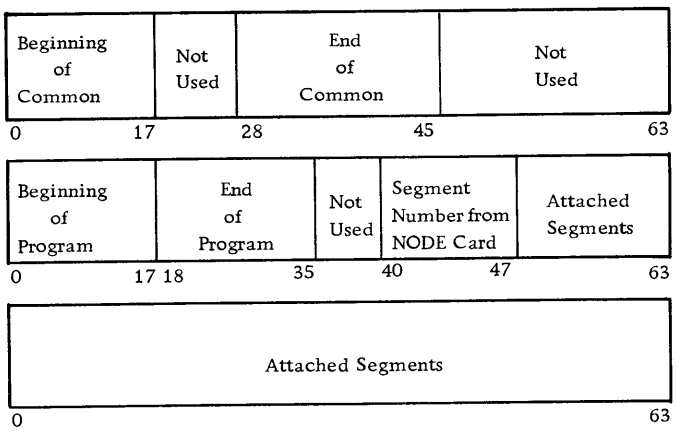


FIGURE 5. ENTRY IN THE NODE TABLE

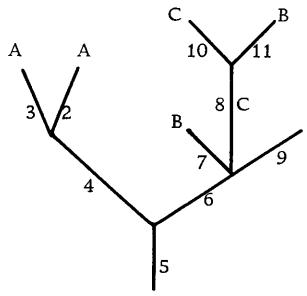


FIGURE 6. TREE STRUCTURE OF A JOB REQUIRING THE SEGMENTATION FEATURE OF THE BSS PROCESSOR

Notes: Segment 5 is the primary segment and contains the subprogram with entry point *MAIN*, all library subprograms, the FIOD subprogram, and the link subprogram. Segment 5 is allocated starting at location 33.0 and remains in storage at all times.

The NODE card in front of segment 5 should contain T NODE, 05,00 in the NODE card format. (See card formats, Appendix A.) The NODE card in front of segment 2 should contain T NODE, 02, 04.

Segments 10 and 11 are allocated storage beginning at the same location, and therefore will not be in storage at the same time. Segment 9 is allocated storage higher than segment 6 in the diagram. Segments 5, 6, 8, and 11, with their associated named commons, may make up one storage load.

Since only segments 2 and 3 use named common A, named common A is associated with segment 4 and is available for storage whenever segment 4 is called in. Named common B is associated with segment 6. Named common C is associated with segment 8.

If A were a block data subprogram contained in segment 3 and referenced by segment 2, an error would be indicated.

Named common B would be allocated space higher in storage than named common C.

The number of segments contained in a single storage load cannot exceed 8. The total number of segments cannot exceed 99.

Some types of transfer are illegal. The general rule is made that no transfer is legal which causes the originating segment to be overlaid. Hence, transfers from a subprogram in one storage load to a subprogram in another storage load are illegal. A transfer from one segment of a storage load to another segment of that storage load is legal provided no transfer is made by the second segment which can cause the originating segment to be overlaid, since the return cannot be executed. In Figure 6, a transfer from segment 6 to segment 4 is illegal. A transfer from segment 11 to segment 8 is illegal if segment 8 attempts to transfer control to segment 10 before the return to segment 11.

The BSS processor determines whether illegal overlay can occur, and gives an error message if an illegal transfer exists. There are two exceptions to the BSS processor checking procedure:

1. No check is made to determine if illegal overlay is caused by a called subprogram which appears in an argument list by way of an EXTERNAL statement.
2. No check is made to see if named commons are overlaid.

A named common cannot be expected to be preserved when the segment with which it is associated has been overlaid.

The total of all subprograms and common associated with the segments of any storage load must not exceed in size the storage available. An error message is given by the BSS processor if the limit is exceeded.

Detail of Segmentation Function

There are two parts to the segmentation feature. The first part operates as a departure from the normal path of the BSS processor, and its function is to set up the node table, which contains information necessary for segmentation, and to form the relocation table, used by the MCP loader. The second part of the segmentation feature is the link program, which operates at execution time, and performs the supervisory function of calling for the proper segment to be loaded by the MCP loader. (Refer to the Link Program section.)

BSS Processor Deviation For Segmentation Function

When the BSS processor has completed its scan of the input deck, the segmentation portion of code is entered if a NODE card had been encountered. The node table, which at this time contains the segment numbers and the segments to which they are attached (taken from the NODE cards), is expanded in the following way: successive scans of the node table are made, picking up the chain of attachment of the segments. (For example, if a table entry shows that segment 5 is attached to segment 2, a search is made to find out to what segment segment 2 is attached.) When the primary segment is encountered (attached to segment number 00), the lower limit for available storage is placed in the node table, and an entry is made in the second intermediate table for the segment number of blank common. A link table mask is set up in the communication region of MCP (Figure 7) to indicate which is the primary segment. This mask is a bit string used by the link program to determine which segments are currently in storage. (See information on the link program.) The scan for attached segments continues until all segments are

traced to the primary segment. If a segment is not attached properly in this phase, an error message is given. A node table entry would show 08, 04, 02, 01, 00, where segment 8 was attached to segment 4, segment 4 was attached to segment 2, etc., at the end of the scan.

The expansion of the node table is followed by the normal scan of the library subprograms called by the input deck.

The remaining functions of the segmentation feature of the BSS processor are divided into 8 phases, described below.

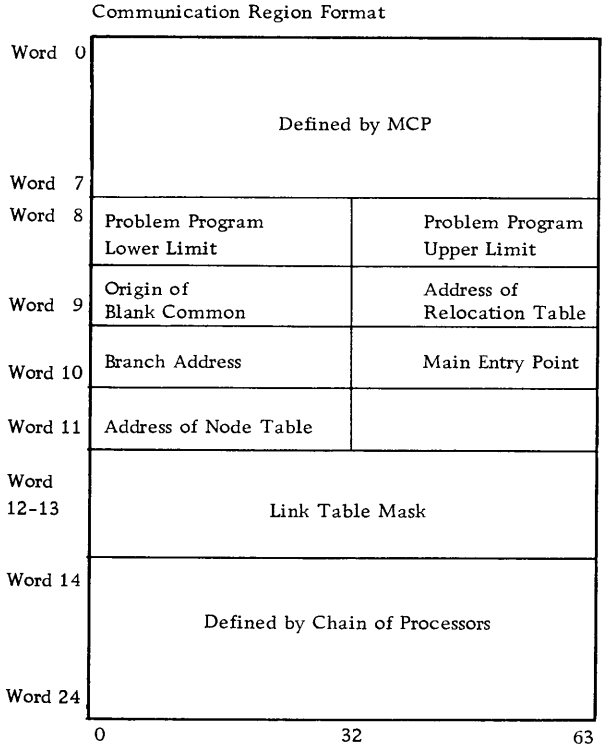


FIGURE 7. MCP COMMUNICATION REGION

Phase 1 -- Computation of storage addresses for the beginning and end of each segment and computation of absolute locations of entry points: A scan is made of the node table for the primary segment entry. This discovery initiates scanning the first intermediate table, picking up all subprograms in the primary segment, and computing absolute origins of entry points in all subprograms of that segment. When all entries in the first intermediate table have been scanned, the storage address of the end of the segment is placed in the node table. (A pointer which contains the address of the sum of the subprogram origin and its length is kept during the scan of the first intermediate table.) The storage address of the beginning of the next segment is placed in the node table. The node table is scanned for that segment number, and the process continues until all origins of entry points are complete, and all storage addresses for the beginning and end of each segment have been computed.

Phase 2 -- Elimination of commons of the same name and adjustment of the segment in which the duplicate appears: A scan of the second intermediate table is made. The segment number of the segment in which the common was declared has been filled in when the common definition card was encountered in the input deck. When two named commons are found which have the same name (but are not necessarily in the same segment), the size of the larger of the two named commons is substituted in the entry in the second intermediate table for the size of the smaller. A search of the node table is initiated for the entries which have the same segment numbers as the segment numbers of the duplicate commons. Pointers are moved across the entries in the node table until a match is made of associated (attached) segment numbers. When the match is found, that segment number is placed in one of the duplicate named common entries in the second intermediate table. The segment number of the duplicate entry is set to zero, and a pointer is set up in the entry that has been eliminated in this way. A check is made for duplicate block data commons and for an attempt to associate a block data common with a segment other than the one in which it occurs. If either of the two conditions occurs, an error message is given.

The scan is continued until all commons of identical names have been handled and the associated segment numbers have been filled in.

Phase 3 -- Computation of absolute common origins: The end of blank common is computed and placed in the node table entry for the primary segment. The second intermediate table is scanned to find all commons associated with the primary segment. The origins of these commons are computed by subtracting the amount of common from the origin of the common previously computed, starting from the base address of blank common. In the primary segment the common origins are lowered, if necessary, to allow space for the relocation table for the primary segment. When all commons associated with a segment have been treated, the beginning and end addresses of common for that segment are placed in the node table. The above process is repeated for each entry in the node table.

A second scan of the second intermediate table is made in order to complete the origins of duplicate commons. The origin of the duplicate common is obtained from the entry of the same name.

Phase 4 -- Checking for overlap of program and common: A scan is made of the node table. If the beginning address of common for any segment is lower than the highest address occupied by a subprogram in that segment, an error message is given. An additional check is made to insure that there is space available for the relocation table for the primary segment.

Phase 5 -- Formation of transfer vectors and checking for legality of initial transfers between segments: Segment numbers of segments transferred to are matched, from the third intermediate table, in the node table. If the desired transfer address is either within the segment or to a segment which will be loaded at a lower storage address, the actual address of the transfer is computed and stored in the third intermediate table entry. If the desired transfer address is to a segment which will be loaded at a higher storage address, a transfer to the link subprogram is stored in the third intermediate table. (See also information about the link subprogram.)

In order to test the legality of the transfers, the structures of the node table entries for the two segments involved in the transfer are examined (that is, the entry for the segment from which the transfer is made and the entry for the segment to which the transfer is made). If it is possible to match the segment number from which the transfer is made with one of the associated segment numbers in the node table entry for the segment to which the transfer is made, the transfer may be legal. The transfer may also be legal if it is possible to match the segment number to which the transfer is made with one of the associated segment numbers in the node table entry for the segment from which the transfer is made. However, if neither of the two above conditions are met, then the transfer is illegal. In Figure 8, the legality of transfers of the following kinds are tested by this phase: (1) a transfer from segment 1 to segment 4 may be legal; (2) a transfer from segment 4 to segment 1 may be legal; (3) a transfer from segment 3 to segment 4 is illegal; (4) a transfer from segment 2 to segment 3 is illegal.

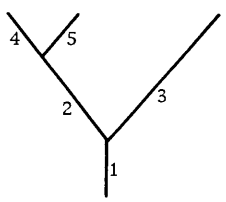


FIGURE 8. EXAMPLE OF TRANSFERS BETWEEN SEGMENTS

Phase 6 -- Further testing for legality of transfers between segments: A scan of the third intermediate table is made. The entries are marked in such a way that each direction of transfer is unique. With successive comparisons of the segment numbers of the entries in the third intermediate table with segment numbers and their associated segment numbers in the entries in the node table, in a manner similar to the procedure described in phase 5, the compatibility of the transfers is tested. A transfer to a segment at a lower storage address is illegal in this phase if the return cannot be made because another transfer would cause the original segment to be overlaid.

Phase 7 -- Formation of relocation table: The base address of the relocation table is computed. (Note: A job which uses the segmentation feature will not have its relocation table starting at the same storage location as the base address of blank common. Refer to discussion of storage allocation under Segmentation.) A check is made to see if the relocation table is too large. Using the information in the three intermediate tables, the relocation table entries for each segment are filled in. The address of the relocation table is placed in the communication region of MCP. (See Figure 5.)

Phase 8 -- Transfer of the node table: The base address of the node table is computed. (The node table is originally located in lower storage, but its final location is at the top of available storage.) A verification is made that the relocation table is lower in storage than the computed address for the node table. The first 64 bits of each entry in the node table are not transferred to the upper address.

The BSS processor gives up control after reordering the library subprogram entries in the relocation table and printing out the mapping of storage allocation.

The Link Program

The segmentation feature of the BSS processor places into the transfer vector a branch to the link program, if the call is to a subprogram in a segment which will occupy a higher storage position than the segment from which the call is made.

The communication region of MCP is set up by the BSS processor to contain a link table mask. Each bit which is turned on in the mask is an indication to the link program that the segment designated by that bit is currently in storage. (See Figure 7.)

Control is initially passed to the link program (after the loading of the primary segment, but before execution has begun). The only function of this entry

to the link program is to pick up the communication region, with the link table mask, from MCP. The link program then gives control to the object program at the main entry point.

The link program receives control again through the transfer vector, as described above. The transfer vector contains, in addition to the branch to the link program, the absolute address to which the transfer is requested and the segment number of the segment to which the transfer is made.

The link program determines from the link table mask whether the required segment is currently in storage. If the segment is already in storage, the link program passes control to the absolute address contained in the transfer vector. If the desired segment is not currently in storage, the link program searches the node table entry for that segment number. The link program calls for the loading of each of the segments in that node table entry until all the segments in that entry are loaded. Control is then passed to the required address.

The following examples illustrate various deck arrangements and job environments which are acceptable to the BSS processor and the MCP loader. Example 1 is a job for which the assembly program will produce both relocatable and absolute output. The deck arrangement of example 1, shown in Figure 9, would be loaded by the MCP loader. The BSS processor would be used to handle storage allocation for example 2, however, whose deck arrangement is shown in Figure 10. Example 3 shows the use of SUBTYPE cards and NODE cards in a job which uses the segmentation feature of the BSS processsor. Example 4 illustrates the core storage allocation of all subprogram entry points and commons, provided by the BSS processor in a FORTRAN environment.

Example 1. Non-BSS Deck with Both Relocatable and Absolute Subprograms.

```
B       JOB,NON'BSS MIXED DECK

B       TYPE,COMPILGO,STRAP

        PUNREL
        ORIGIN,(8)41.
START   B,$MCP
        ,$SPR
        ,MESSG1
        ,1.
        B,ABSLUTE
        B,$MCP
        ,$ABEOJ
ABSLUTE SYN,(8)66.
MESSG1  (A8)DD(8U),J NOW IN MAIN SUBPROGRAM8
        DRZ(8U),14

        PUNNOR
        SLC,(8)66.
        B,$MCP
        ,$SPR
        ,MESSG2
        ,1.
        B,RELOCAT
        B,$MCP
        ,$ABEOJ
RELOCAT SYN,(8)103.
MESSG2  (A8)DD(8U),J NOW IN ABSL SUBPROGRAM8
        DRZ(8U),14

        PUNREL
        ORIGIN,(8)103.
        B,$MCP
        ,$SPR
        ,MESSG3
        ,1.
        B,END
END     SYN,(8)130.
MESSG3  (A8)DD(8U),J NOW IN RELOCATABLE SUB8

        PUNNOR
        SLC,(8)130.
        B,$MCP
        ,$SPR
        ,MESSG4
        ,1.
        B,$MCP
        ,$EOJ
MESSG4  (A8)DD(8U),J END OF PROGRAM8
        DRZ(8U),15
        END,(8)41.
```

Example 2. BSS Deck with Subprograms in Both FORTRAN and STRAP

```
B       JOB,BSS MIXED DECK

B       TYPE,COMPILGO,FORTRAN

T       SUBTYPE,FORTRAN
      COMMON/BILL/A(10),C(10)
      DIMENSION D(10)
      N=10
      PRINT 100
      DO 20 I=1,N
 20   A(I)=I
      CALL SUB1(N)
      DO 21 I=1,N
 21   D(I)=STSUB1(C(I))
      SUM=0.0
      DO 22 I=1,N
 22   SUM=SUM+ABSSUB(D(I))
      PRINT 99,SUM
      RETURN
 99   FORMAT (6HOSUM =,E25.14)
 100  FORMAT (26HOTHIS IS THE MAIN PROGRAM.)
      END

T       SUBTYPE,STRAP
        PUNREL
        SEM,92,114,116
        PUNFPC,LAST
ABSSUB  ENTER,ENTPT
        XW,0
ENTPT   B,1.($15) * RETURN
LAST    DR(N),0
        END

T       SUBTYPE,FORTRAN
      SUBROUTINE SUB1(N)
      COMMON/JOE/B(10)
      COMMON/BILL/A(10),C(10)
      PRINT 100
      DO 90 I=1,N
 90   B(I)=A(I)*A(I)
      NN=N
      CALL SUB2(NN,SQSUM)

      DO 91 I=1,N
 91   C(I)= A(I)/SQSUM
      RETURN
 100  FORMAT(25HOTHIS IS SUBROUTINE SUB1.)
      END

T       SUBTYPE,STRAP
        PUNREL
        SEM,92,114,116
        PUNFPC,LAST
STSUB1  ENTER,SUBEP
        XW,0
SUBEP   B,1.($15) * RETURN
LAST    DR(N),0
        END

T       SUBTYPE,FORTRAN
      SUBROUTINE SUB2(NN,SQSUM)
      COMMON/JOE/B(10)
      SUM=0.0
      PRINT 100
      DO 90 I=1,NN
 90   SUM=SUM+B(I)
      SQSUM=SQRT(SUM)
      RETURN
 100  FORMAT (25HOTHIS IS SUBROUTINE SUB2.)
      END
```

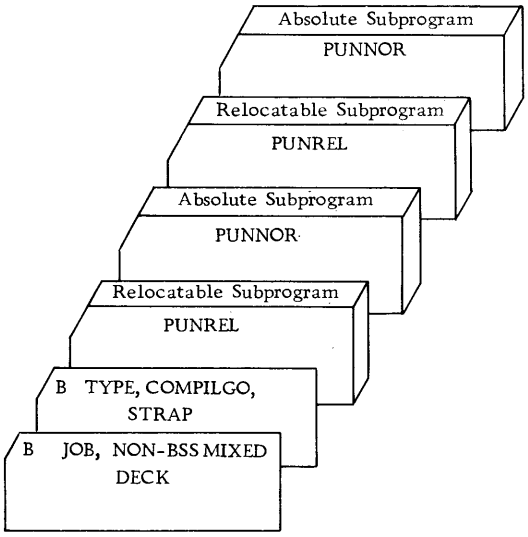


FIGURE 9. STRAP JOB WITH SUBPROGRAMS IN BOTH RELOCATABLE AND ABSOLUTE

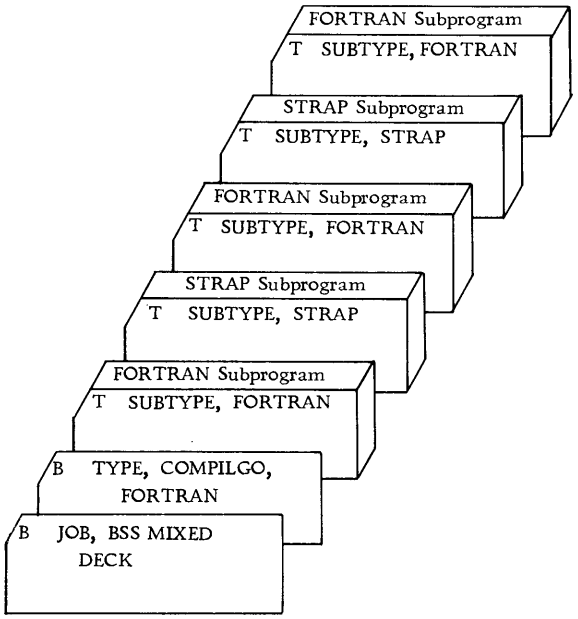
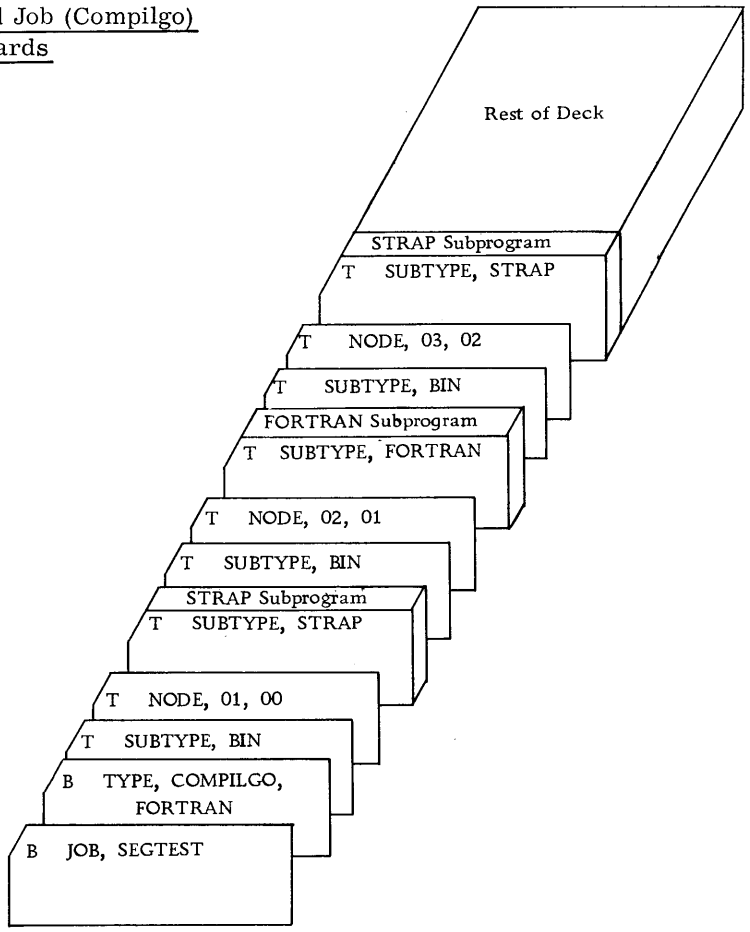


FIGURE 10. FORTRAN JOB WITH SUBPROGRAMS WRITTEN IN BOTH STRAP AND FORTRAN

Example 3. BSS Segmented Job (Compilgo) Illustrating use of NODE Cards



Example 4. Typical BSS Storage Map

SUBPROGRAMS

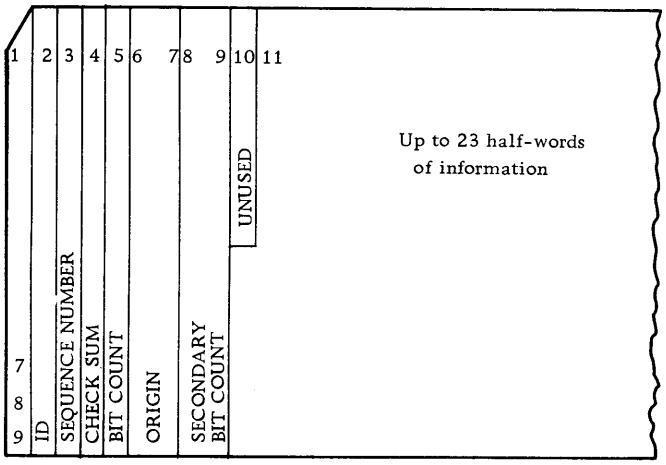
NAME	ORIGIN	REL. E. P.	SIZE	LIB.	SEG. NO.
MAIN	000041	000010.0	000745	NO	
STEP	001006	000022.0	000441	NO	
RANDM	001447	000000.0	000007	NO	
POLY	001456	000022.0	000346	NO	
DERIV	002024	000022.0	000271	NO	
VALUE	002315	000022.0	000134	NO	
FISIP*	002451	000001.0	000007	YES	
IOCS*	002460	000002.0	003037	YES	
SPTXT*	002460	001124.0	E. P.	YES	
UST*	005517	000001.0	000002	YES	

COMMONS

NAME	ORIGIN	AMOUNT	SEG. NO.
	213713	000014	
DIFF	213703	000010	
A	213535	000146	
STE	213527	000006	
POL	213363	000144	

APPENDIX A. CARD FORMATS

Absolute Origin Card

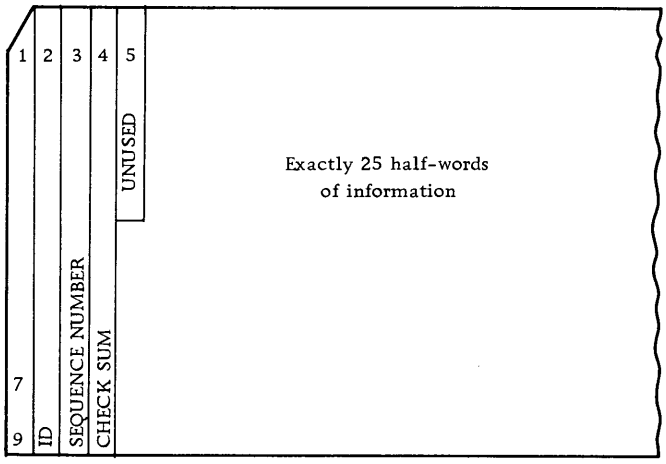


Secondary Bit Count (columns 8-9)

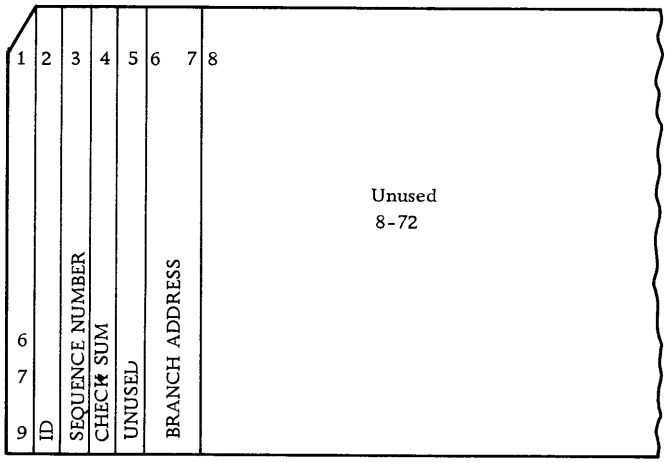
Bits to be zeroed/skipped before/after loading as determined from 5.0, 5.1:

5.0	0-skip	1-zero
5.1	0-before	1-after

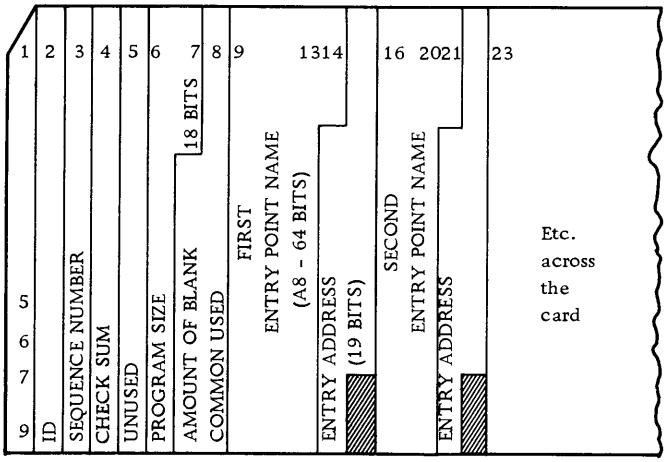
Absolute Flow Card



Absolute Branch Card

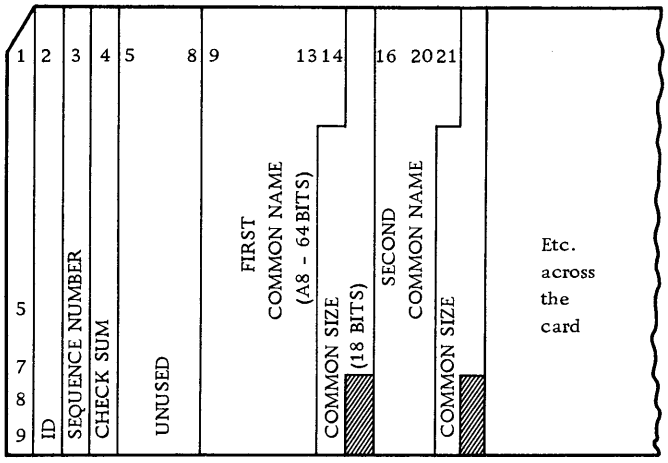


FORTRAN Program Card



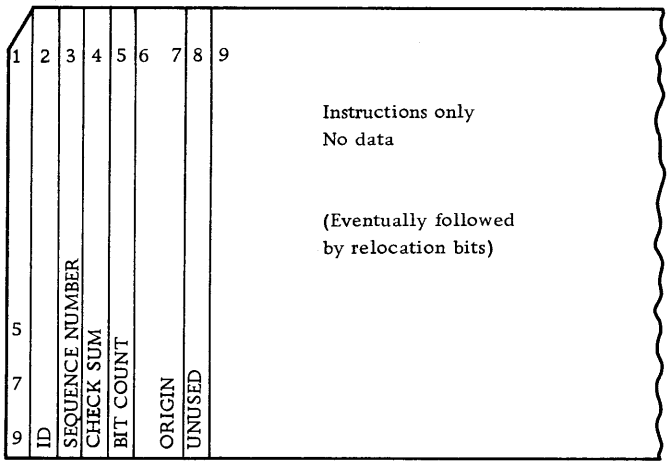
Columns 6, 7, and 8 are blank on any subsequent continuation program cards.

Common Definition Card



If the number of named commons is greater than that which will fit on a single card, additional cards will be punched in the same format.

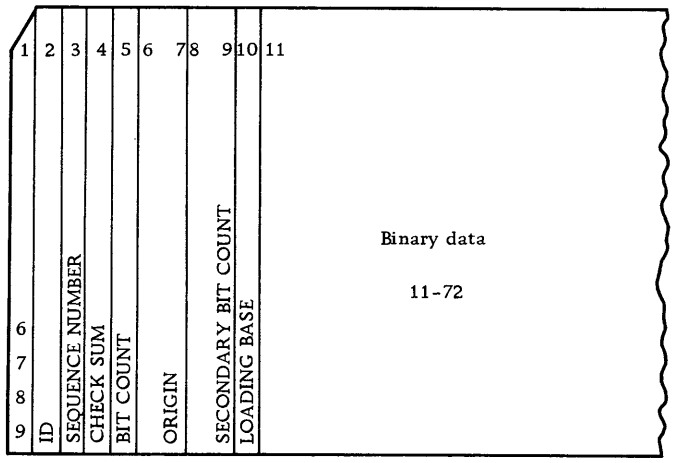
Relocatable Binary Instruction Card



Relocation Bits (describing a half-word field)

0	No relocation
1 0	Relocation
1 0 0 . .	First 18 bits (address)
1 0 1 . .	Last 18 bits (refill)
1 0 . . 0	As lower address
1 0 . . 1 . .	As upper address
1 0 . . 1 0	With respect to blank common
1 0 . . 1 1 i	With respect to named common number i (length of i sub-field is determined by number of named commons)

Relocatable Binary Data Card



No relocation bits on this card.

Loading Base (column 10)

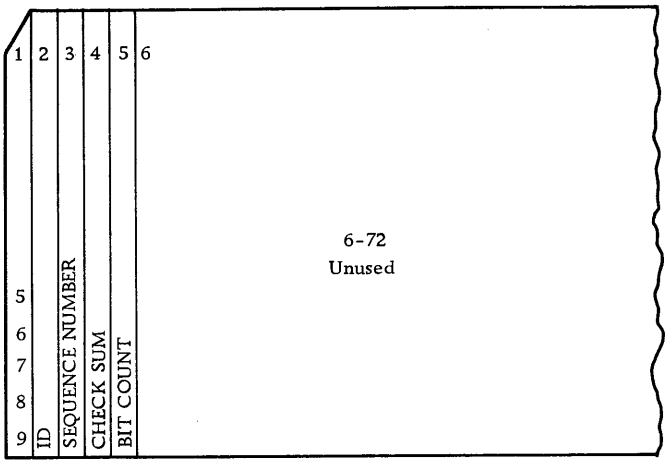
0 - Program Data
1 - 1st Named Common
2 - 2nd Named Common, etc.

Secondary Bit Count (columns 8-9)

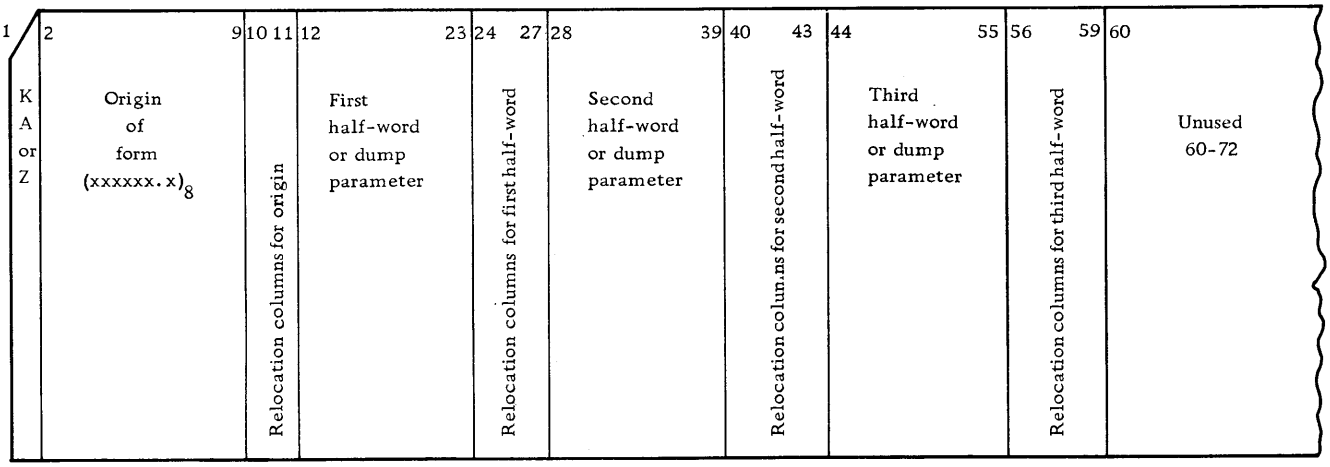
Bits to be zeroed/skipped before/after loading as determined from 5.0, 5.1:

5.0	0-skip	1-zero
5.1	0-before	1-after

FORTRAN Branch Card



Relocatable K, A, or Z Card (Octal - Hex)



Relocation columns for origin (columns 10-11)

00 - relocate with respect to the subprogram origin.

xx - relocate with respect to named common number xx, where xx is a two-digit octal number. (Named commons are numbered in the order in which they appear within the subprogram.)

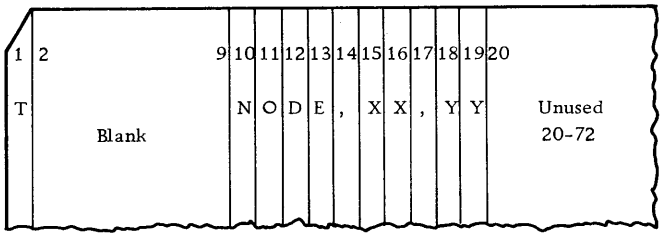
Relocation columns for each half-word (columns 24-27, 40-43, 56-59)

----	No relocation
P---	Relocate with respect to the subprogram origin
C---	Relocate with respect to common origin
P0--	Relocate the leftmost 18 bits with respect to the subprogram origin
P1--	Relocate the rightmost 18 bits with respect to the subprogram origin
C0--	Relocate the leftmost 18 bits with respect to common origin
C1--	Relocate the rightmost 18 bits with respect to common origin
C0xx or C1xx	Relocate leftmost or rightmost 18 bits, respectively, with respect to named common number xx. (If xx = 00, relocate with respect to blank common.)

Note: Blanks in significant columns will be considered to be 0's. The number of the named common must be right-justified in the field in which it appears. The format of continuation K, A, Z cards is the same as above, except that columns 2-11 should be blank. Dump parameters are the same as for absolute dump cards.

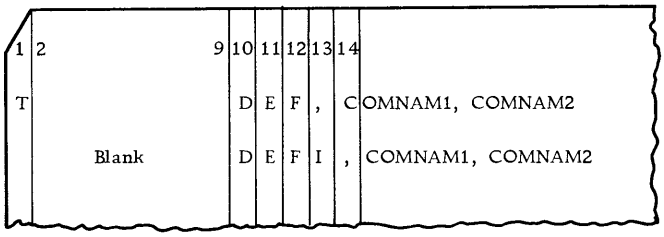
T Cards

1. Node Card



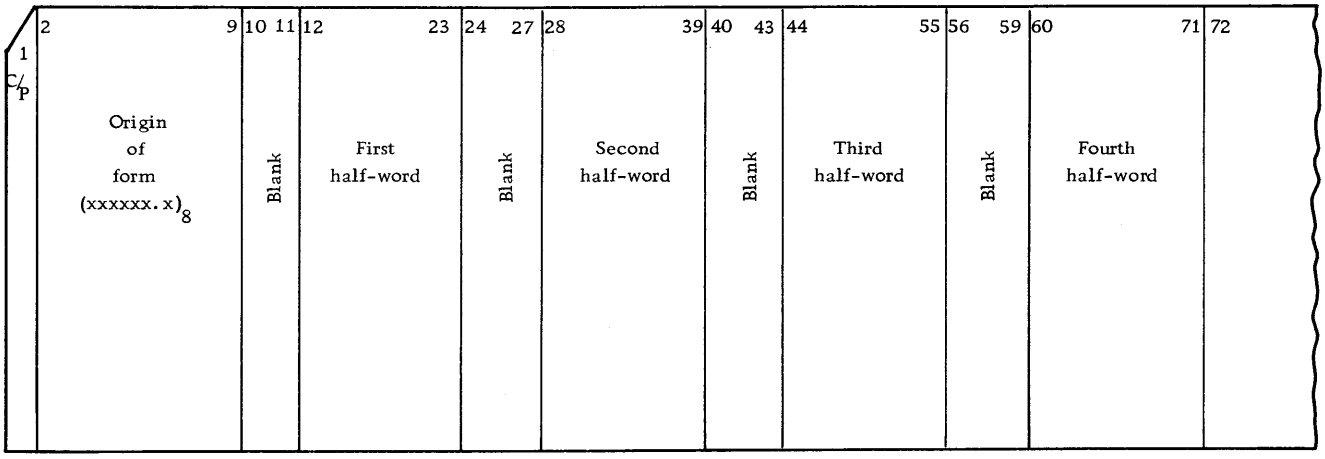
XX is the two-digit segment number for the group of subprograms following this node card and preceding the next node card. YY is a two-digit segment number of the segment to which the current segment, XX, is to be attached. The value of YY should be 00 on the node card for the primary segment.

2. Define and Define Immediate Cards



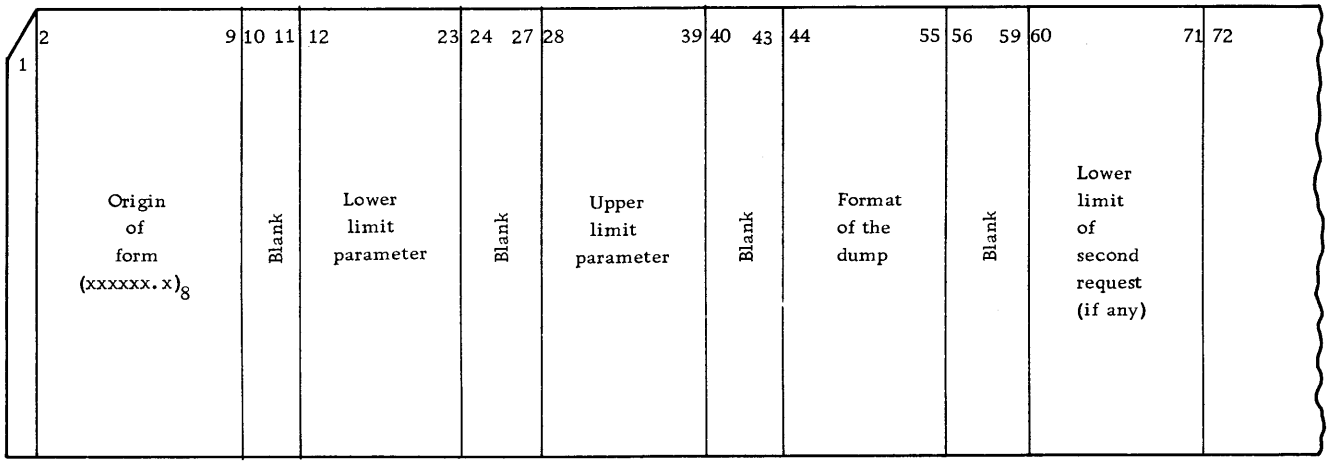
Note: All other T cards may have columns 2-72 used for anything. The format of a super T card is S and T multi-punched in column 1, but the card is printed by the loader as an ST. Columns 2-72 are ignored.

Absolute Correction/Patch Cards (Octal-Hex)



Each half-word should be of the form xxxxxx.xx HH.

Absolute Dump Card



Format of the dump (columns 44-55)

The following formats are available:

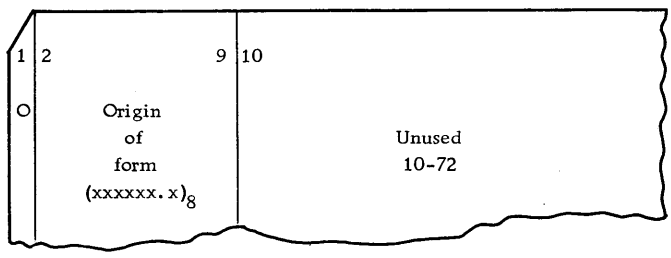
0.0	Octal - hex with panel
0.40	Octal - hex without panel
1.0	Floating-point with panel
1.40	Floating-point without panel
2.0	Index word with panel
2.40	Index word without panel
17.0	Octal - hex with panel, no mnemonics
17.40	Octal - hex without panel, no mnemonics

The termination of dump requests is determined by bit 24 in the format half-word.

Bit 24 = 0	No more requests follow
Bit 24 = 1	More requests follow

If more than one dump request is given for the same location, the lower limit for the second request begins in column 60. The upper limit and format of the second request are continued on a second D card in column 12. Columns 2-11 are left blank on this and all subsequent dump continuation cards.

Loader Adjustment Card



APPENDIX B. REJECT MESSAGES

MCP Loader

MCP loader diagnostics are usually of the following form:

LOD. REJ. CD. XXXX ID YYYYYYYY
(Reason)
LAST CORRECT CARD LOADED IS SEQUENCE NO. XXXX ID YYYYYYYY

where:

1. XXXX is the sequence number taken from column 3 of the binary card.
2. YYYYYYYY is the identification taken from columns 73-80 of the binary card.
3. (Reason) is one of the following:

CHECK SUM ERROR
SEQUENCE ERROR
ILLEGAL TYPE OF CARD
ILLEGAL NON-BSS FUNCTION (I.e., a card was encountered in a non-BSS environment which may only be used in a BSS environment.)
TRIED TO LOAD OUTSIDE LIMITS (This message is also given if the address specified on the branch card is outside the problem program limits. Normally, it means that a binary, C, P, D, K, A, or Z card, if loaded, would have gone outside the boundaries specified on the LIM card. The programmer can probably correct the error by extending the limits.)
C, P, D, K, A, or Z CD PUNCH ERR (There is an illegal character in one of the fields.)
NO ORIGIN ON FIRST C OR K CARD
NO ORIGIN IN FIRST CARD (In an absolute deck, this message may mean that the first card was a flow card. A non-BSS relocatable deck without a loader adjustment card will also cause the above error message.)

4. If the last correct card loaded was in card code, the entire card is printed, instead of the sequence number and identification.

Other messages given by the loader are:

JOB CANNOT BE RUN BECAUSE OF INPUT DIFFICULTIES (An uncorrectable UK was encountered in the reading of this deck by MCP.)

NO BRANCH CARD IN CURRENT DECK--JOB REJECTED

BSS Processor

The following error messages may be given by the BSS processor:

1. JOB TERMINATED DUE TO SEQUENCE (OR CHECK SUM) ERROR. LAST CORRECT CARD IS SEQ. NO. X ID Y IN Z SUBPROGRAM. (X is the sequence number from column 3, Y is the identification in column 4, and Z is the subprogram name.)
2. JOB REJECTED - COMMON AND PROGRAM OVERLAP. (The total size of subprograms and commons is greater than the size of available storage.)
3. BSS TERMINATED JOB - PROGRAMMER ERROR (The above message appears on the console only, for any of the errors below.)

JOB TERMINATED - (REASON) (This message appears on the output listing, where (reason) is one of the following:)

T.V. XX INCOMPLETE IN YY SUBPROGRAM (Entry point XX cannot be found, either in the input deck or in the FORTRAN library. YY is the name of the subprogram in which the transfer vector appears. A missing subprogram may have to be added to the input deck.)

TOO MANY NAMED COMMONS IN YY SUBPROGRAM (The number of entries in the second intermediate table has exceeded available storage.)

TOO MANY TRANSFER VECTORS IN YY SUBPROGRAM (The number of entries in the third intermediate table has exceeded available storage.)

ILLEGAL TYPE OF CARD IN YY SUBPROGRAM (The column 1 code is not acceptable to the BSS processor.)

RELOCATION TABLES ARE TOO LARGE (The number of entries in the relocation table has exceeded available storage.)

INCORRECT CARD ORDER IN YY SUBPROGRAM (See Card Order in section on BSS processor.)

4. ILLEGAL DISK INPUT TO BSS LOADER
5. LOADER ADJUSTMENT HAS LOW ORIGIN (The origin specified on the loader adjustment card would cause this subprogram, if loaded, to overlay a subprogram previously loaded. The origin on the loader adjustment card should be changed.)
6. REPEATED UK INTERRUPTS FROM DISK
7. EPGK INTERRUPT FROM DISK
8. COMMON XX WAS NOT IN THE SYMBOL TABLE WHEN THE DEFINE CARD WAS ENCOUNTERED IN SUBPROGRAM YY (The first named common specified on the define card has not yet been declared on a common definition card in the input deck.)
9. THE 50TH DEFINE CARD HAS BEEN ENCOUNTERED. THE REST (IF ANY) WILL BE IGNORED (No more than 50 significant define cards may be used.)

10. NO ENTRY POINT WITH NAME *MAIN* - JOB WILL NOT BE RUN (There is no main subprogram in the input deck.)

The following additional error messages may appear in a job which uses the segmentation feature:

1. ERROR - SEGMENTS NOT PROPERLY ATTACHED (A segment has been declared which is not attached directly or indirectly to the primary segment. The programmer may have to re-organize his overlay.)
2. ERROR - SEGMENTS HAVE TOO MANY LEVELS (Segments may have no more than eight levels. The programmer should re-organize his overlay.)
3. ERROR - ORIGIN CARD OVERLAYS PREVIOUS CODING. ORIGIN IGNORED (The origin specified on the loader adjustment card would cause this subprogram, if loaded, to overlay coding which should not be overlaid. The origin specified on the loader adjustment card should be changed.)
4. ERROR - SEG END IN COMMON TABLE NOT IN NODE TBL (An out-of-phase condition exists in the BSS processor.)
5. ERROR - DUPLICATE DATA COMMONS (Two block data subprograms have declared named commons of the same name. Unique names should be assigned.)
6. ERROR - DATA COMMON NOT IN PROPER SEGMENT (The data common was not declared in the segment with which it is associated by the segmentation logic. The block data subprogram should be placed in another segment.)
7. ERROR - MAIN SEGMENT NOT FOUND (An out-of-phase condition exists in the BSS processor.)
8. ERROR - THE PROGRAM EXCEEDS STORAGE IN SEG XX (The total amount of storage used in segment XX for subprograms and commons is larger than the amount of storage available. Segments should be re-arranged.)
9. ERROR - ILLEGAL TRANSFER BETWEEN SEGMENTS (See illegal types of transfers in Phase 5 of section on Segmentation.)
10. ERROR - TOO MUCH STORAGE USED BY RELOCATION TABLE
11. ERROR IN LOCATION OF RELOCATION TABLE (An out-of-phase condition exists in the BSS processor.)

Printed in U.S.A. C28-6379-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601