File No. S360-32
Form C28-6505-2

IBM Systems Reference Library

IBM System/360 Basic Programming Support

Basic Utilities

360P-UT-017, -018, -019, -020

This publication contains all the information needed to make use of the Basic Programming Support Basic Utility programs provided for users of IBM System/360. The utility programs provided are:

Absolute Loader
Relocating Loader
Dump Program
Input/Output Support Package

These programs are designed to load assembled programs into main storage; to provide listings of the contents of storage; and to provide routines for accessing input/output devices. The functions and possible modifications of each program are described in detail.

PREFACE

This publication provides all the information users need in order to use the IBM System/360 Basic Programming Support Basic Utility programs.

To fully understand the material presented in this manual, the reader should be familiar with the contents of the following publications:

IBM System/360 Basic Programming Support Basic Assembler Language, Form C28-6503

IBM System/360 Principles of Operation, Form A22-6821

In addition, it is suggested that the reader be familiar with the following publication: IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.

This publication is arranged in sections which correspond to the major parts of the utility program package. Also included are appendixes that discuss:

- Program segment relocation and linkage, and
- A program capable of creating self-loading loaders.

Illustrations, charts, and examples are provided throughout to provide the clearest possible presentation.

MAJOR REVISION (June 1965)

This revision, which supersedes the previous edition, represents Version 2 of the subject programs. The changes are marked by a vertical bar to the left of the revised text; revised illustrations are denoted by the symbol • to the left of the caption.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

CONTENTS

CHARTS

FIGURES

TABLES

Every installation requires programs to perform such common functions as loading an assembled program into storage or providing a listing of the contents of storage. To save the programmer the time and effort required to write and modify this type of program as job requirements change, IBM makes utility programs available to its customers.

IBM SYSTEM/360 BASIC PROGRAMMING SUPPORT BASIC UTILITY PROGRAMS

The four utility programs provided are: the absolute loader, the relocating loader, the dump program, and the input/output support package.

Absolute Loader

The absolute loader loads program segments (the output of an assembly is called a program segment; a program may be composed of one or more segments) into storage at the addresses assigned to them by the assembler and transfers control to a program segment for execution; it also allows the user to make corrections or additions to the program segments at load time.

Relocating Loader

The relocating loader can load program segments into storage at locations other than those assigned by the assembler; it completes linkage among the segments so that one program segment may refer to another; it allows corrections or additions to be made to the program segments at load time; and it transfers control to one of the loaded segments for execution.

Dump Program

The dump program provides a listing of the contents of all or part of storage, the general registers, and floating-point registers (or any combination of these). The program will edit the listing to fit any of eight basic formats, which are described in "Output Formats." The dump program is available in a single-phase symbolic version, a single-phase assembled version, a single-phase self-loading version (self-loading dump), a symbolic two-phase version, an assembled version of phase 1, and a self-loading version of phase 2.

Input/Output Support Package

The input/output support package consists of a modular set of subroutines which enable the user to utilize input/output devices. (A module in the input/output support package is a logical sequence of coding which either sets up or executes one I/O function.) These are routines to read or punch a card, write on the message or printer device, sense information from a device, single space on the message or printer device, skip to channel one on the printer, read or write tape, write a tapemark, rewind tape, backspace tape a record or file, forward-space tape a record or file, and to read tape backward.

MACHINE REQUIREMENTS

The IBM System/360 Basic Programming Support Basic Utility programs require the following minimum machine configuration:

1. IBM System/360 with 8,192 bytes of storage,
2. One IBM 2540 or 1442 Card Reader-Punch,
3. Standard instruction set,
4. IBM 1403 OR 1443 Printer, or the IBM 1052 Printer-Keyboard if the dump program is being used,

The user's input/output configuration determines what routines he can use from the input/output support package.

MAIN STORAGE REQUIREMENTS

The following is an approximation of how much storage each of the utility programs will occupy. (The user should also take in

to account that locations 0-127 should be added when figuring available storage.)

Program	Bytes of Storage Space
Absolute Loader	2,580*
Relocating Loader	3,800*
Dump (Phase 1 of 2)	3,100*
Dump (Phase 2 of 2)	6,350**
Dump (single phase)	4,460
Dump (self-loading)	3,000
I/O subroutine	800-2,720***

* In the versions of the absolute and relocating loaders supplied by IBM, there is a 250-byte sequence of coding (Initial Entry Routine) that the loaders use to determine the system's configuration. Since this 250-byte area may be overlaid by a program segment at execution time, it is not included in these approximations.

** Needs minimum of 8K to operate. Uses remainder of 8K as buffer.

*** The bytes of storage occupied by the I/O subroutines depend on the installation's requirements.

If the user selects the modules necessary for his installation from the I/O support package and keeps them resident in storage, the I/O modules can be removed from the program and the programs modified by reassembly to link with the I/O of the installation. If this is done, these approximations would be greatly reduced.

The maximum length program which can be loaded by the relocating loader on an 8K configuration is 4,250 bytes, decreased by 12 bytes for each ESD card in the deck to be loaded. Therefore, the use of the relocating loader is recommended only for users with greater then 8K bytes of storage.

INTRODUCTION

The absolute loader loads program segments into the storage locations assigned by the assembler. (The absolute loader will not overlay itself: any attempt to do so will result in an error wait.) This loader recognizes as input three types of load cards. Two of these, the Text (TXT) and Load End (END) cards, are generated by the assembler; the Replace (REP) card, if needed, must be supplied by the programmer. The absolute loader will also accept program segments intended for use by the relocating loader, with the following exceptions:

1. All other cards, including the load cards recognized only by the relocating loader -- the Set Location Counter (SLC), Include Segment (ICS), External Symbol Dictionary (ESD), Relocation List Dictionary (RLD), and Load Terminate (LDT) cards - are ignored. Information meaningful only to the relocating loader in the Text, Replace, and Load End cards is also ignored.

2. Linkage with another program segment is not supplied. If one program segment must refer to instructions or data in a separate program segment, absolute addresses must be used.

3. Two or more program segments can be loaded one after the other if all END cards are removed except the END card after the last program segment.

The absolute loader is available as follows:

1. a symbolic deck.

2. a self-loading, nonrelocatable deck (assembled in lower storage).

3. a self-loading, nonrelocatable deck (assembled in high storage for an 8K configuration).

If the user wants to employ the self-loading deck, he may have to make the following changes to the END card in the self-loading deck:

1. He must punch (in hexadecimal notation) the address of the input device into card columns 17-20, if the address of the input device is different from the address that the loader is to be loaded from. If it is not different, he may leave it blank.

2. If he desires to use a message or printer device for error indications, he must punch (in hexadecimal notation) the address of his typewriting device into card columns 21-24. If there is no typewriter, he must punch the address of the printer. If he leaves these columns blank, the error indications will only be displayed on the console.

ABSOLUTE LOADER FUNCTIONS

The functions of the absolute loader and the cards associated with each function are listed in Table 1.

Table 1. Absolute Loader Functions

Function	Card
Loading: Places the instructions and/or constants of a program segment into the storage locations assigned by the assembler.	One or more Text cards containing the instructions and/or constants of the user's program segment, and their assigned starting address.
Correcting: Allows changes or additions to the instructions and/or constants within the program segment at load time.	One or more Replace cards containing corrections altering the program segment.
Transferring Control: Ends loading of the program segment and transfers control to some location within the program segment.	Load End card containing an address within the program segment to which control will be transferred.

PROGRAM SEGMENT SEQUENCE

A program segment ready to be loaded includes at least two types of cards: Text cards and a Load End card. A Replace card is inserted by the programmer only if he desires to change and/or add to the program segment at load time.

Figure 1 shows a program segment with a Replace card inserted by the programmer, ready for loading by the absolute loader. (The figure is read from the bottom up.)

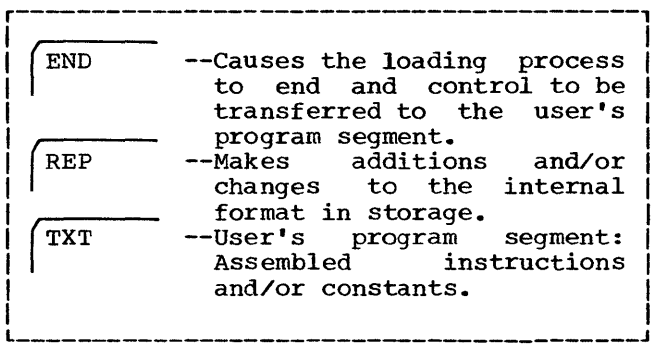


Figure 1. The Sequence of a Program Segment Ready to Be Loaded by the Absolute Loader

CARD FORMATS

The three types of load cards recognized by the absolute loader are defined in detail in the following sections. The function of each card is stated briefly, with any other information pertinent to its use. The card formats are shown in tabular form, with each field of the card explained.

In most cases, values in load cards produced by the assembler are represented in IBM extended card code; for example, the decimal value 20 -- represented in one byte as 0001 0100 -- becomes an 11-9-4 punch in one card column. In contrast, the programmer uses the more convenient hexadecimal code if Replace cards are used. The hexadecimal equivalent of decimal 20 is 14; this is a 1 punch and a 4 punch in two successive card columns, representing the contents of one byte. (Tables for conversion from decimal to hexadecimal are in Appendix B of IBM System/360 Basic Programming Support Basic Assembler Language, Form C28-6503.)

TEXT CARD

The Text card contains, in extended card code, the following:

1. The starting address in storage where the assembled instructions and constants of the user's program segment are to be inserted.

2. The number of bytes of information contained in the card.

3. The text itself; that is, the assembled instructions and/or constants contained in the card.

Each Text card may contain a maximum of 56 bytes of text. Table 2 defines the contents of the Text card fields.

Table 2. Text Card

Column	Contents
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	The starting address, in extended card code, where the information on the card is to be loaded into storage.
9-10	Blank.
11-12	Number, in extended card code, of bytes of text to be loaded from the card.
13-14	Blank.
15-16	Information for the relocating loader. The content of these columns is ignored by the absolute loader.
17-72	From 1 to 56 bytes of text -- instructions and/or constants assembled in extended card code.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

REPLACE CARD

The Replace card is supplied by the programmer, and must be placed in the program segment following the Text cards ~~(or preceding the RLD cards)~~. Both assembled instructions and constants may be changed or additions made. However, all changes and additions must be punched in hexadecimal code.

The programmer cannot replace a two-byte instruction with a four-byte instruction through the load program. In order to replace a two-byte instruction with a four-byte instruction, he must either reassemble his source program or patch; that is, replace the incorrect or old entry with a branch instruction to some storage location into which the replacement will be loaded. Replacement must be made byte for byte.

Table 3 defines the contents of the Replace card fields.

Table 3. Replace Card

Column	Contents
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	REP. Identifies the type of load card.
5-6	Blank.
7-12	Address, in hexadecimal, of the area to be replaced. It must be right-justified in these columns, and unused leading columns filled in with zeros. The address must specify a half-word boundary.
13-16	Blank.
17-70	A maximum of 11 four-digit hexadecimal fields, separated by commas, each replacing one previously loaded half-word (two bytes). The last field must not be followed by a comma.
71-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

LOAD END CARD

The Load End card ends the loading process and causes control to be transferred to some location within the program segment. If a location is not specified in the END card, control is transferred to the first location in storage loaded into from a TXT card (or REP card, if there are no TXT cards) above 127 decimal, or 7F hexadecimal. After control is transferred, the system operates in the Supervisor state, disabled for all interruptions, except a machine check interrupt; see "Input/Output Support Package" for a discussion of interruptions. Table 4 defines the contents of the Load End card fields.

Table 4. Load End Card

Column	Contents
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	END. Identifies the type of load card.
5	Blank.
6-8	Address, in extended card code, of a point in the program segment to which control is to be transferred at load end. If the END card did not specify a point in the program segment to which control is to be transferred, this field will contain blanks and control will be transferred to the first location in storage above location 127 decimal, or 7F hexadecimal, into which data is loaded from a TXT card (or REP card, if one precedes the TXT cards).
9-14	Blank.
15-16	Information for the relocating loader. The content of these columns is ignored by the absolute loader.
17-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

LOADER USE OF I/O SUPPORT PACKAGE

The absolute loader uses selected modules of the I/O support package to read cards or card images from tape. These routines can be used by the programmer by employing the coding sequence (with absolute addresses) discussed in "Input/Output Support Package."

RESIDENT LOADER CONSIDERATIONS

The name of the first instruction in the absolute loader is: LOAD1. If this location is branched to (either from the console or directly from a program segment in storage), another program segment can be loaded without preceding it by another absolute loader. The user may obtain the absolute address of LOAD1 by referring to his listing.

INTRODUCTION

The distinguishing feature of the relocating loader is its ability to relocate program segments and to complete linkage between the segments. (For a detailed discussion on how the relocating loader accomplishes this, see Appendix A.) It also has a storage mapping facility which will provide, on the message device indicated on the END card, the name of each segment and entry point and its assigned location. The relocating loader recognizes eight types of load cards. Four of these are generated by the assembler: the External Symbol Dictionary card (ESD), Text card (TXT), Relocation List Dictionary card (RLD), and the Load End card (END). The other four cards are supplied by the programmer: the Set Location Counter card (SLC) Include Segment card (ICS), Replace card (REP), and Load Terminate card (LDT).

The relocating loader protects itself and the Reference Table (REFTBL) from being overlaid when input is in relocatable form. The Reference Table is a list of 12-byte entries (a maximum of 253 entries) built by the loader; it contains the names and entry points of a program segment along with their present internal location and the relocation factor. When an attempt is made to overlay the loader or the Reference Table an error wait results. (For a discussion of codes and operator actions on any error waits mentioned in this manual, see IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.) When the relocating loader is requested to function as an absolute loader, it does not protect the Reference Table, and the Reference Table can be overlaid.

LOADING CAPACITY

The Relocating Loader available from IBM is set for a maximum storage size of 8K. To modify the Relocating Loader source deck, designed for residence in lower storage, for a storage size greater than 8K it is necessary to alter the constant TOP as described prior to the constant in the listing (or to 131071 for 128K). The source deck should then be assembled and a new loader generated using the LDRGEN program. For further information about loader options and modifications and how to use the Loader Generator Program, refer to the IBM/360 Basic Programming Support, Operating Guide for Basic Assembler and Utilities, Form C28-6557.

The relocating loader is available as follows:

1. A symbolic deck.

2. A self-loading, nonrelocatable deck (assembled in lower storage) for an 8K configuration.

3. A self-loading, nonrelocatable deck (assembled in higher storage) for an 8K configuration.

If the user wants to employ the self-loading deck, he may have to modify the END card in the self-loading deck as follows:

1. Punch (in hexadecimal notation) the address of the input device into card columns 17-20, if the address of the input device is different from the address that the loader is to be loaded from. If it is not different, he may leave it blank.

2. If he desires to use a message or printer device for error indications, he must punch (in hexadecimal notation) the address of his typewriting or printing device into card columns 21-24. If there is no typewriter or printer, he must punch the address of the printer. If he leaves these columns blank, the error indications will only be displayed on the console.

Finally, the relocating loader contains its own location counter (LOCCT); LOCCT determines where program segments will be loaded. LOCCT is set to a constant value during an initial program-loading procedure. Once LOCCT is set, it is subsequently incremented by the number of bytes indicated on an ESD Type 0 card (see "ESD Type 0 (Program Name)"). It may also be incremented by the length indicated on an ICS card (see "Include Segment Card") or set by an SLC card (see "Set Location Counter Card").

UNIQUE RELOCATING LOADER FUNCTIONS

The relocating loader has not only the three functions of the absolute loader (that is, loading, correcting, and transferring control), but also the unique capabilities described in Table 5, by function and the associated control cards.

CARD FORMATS

The eight types of load cards recognized by the relocating loader are described in detail in the following sections. The function of each card is stated briefly, with any special considerations in its use. The card format is shown in tabular form, and each field of the card is explained.

Particular attention has been given to those cards that the programmer supplies (the Set Location Counter, Include Segment, Replace, and Load Terminate cards) and to those cards whose function is closely related to other cards.

SET LOCATION COUNTER CARD

The Set Location Counter card sets the loader location counter in one of three ways:

1. Any absolute address, specified as a hexadecimal number punched in card columns 7-12.

2. Any symbolic address already defined as a program name or entry point. This is specified by a symbolic name punched in card columns 17-22.

3. If there is both a hexadecimal address and a symbolic name, the absolute address (converted to binary) will be added to the internal address assigned to the symbolic name, and the resulting sum will be the address to which the loader's location counter is set. To illustrate this, we will assume that in card columns 7-12 of the Set Location Counter card, 00007F was punched; also that there is a symbolic address called GAMMA and that GAMMA is at storage location 000100 (hexadecimal). The absolute address in card columns 7-12 will be added to the internal address assigned to GAMMA, giving a sum of 00017F. It is at this location in storage that the loader's location counter will be set. (See "Note" under "Include Segment Card.")

If there are blanks in both card columns 7-12 and 17-22, there will be an error wait. If the programmer wishes to use only the symbolic address, he must leave the absolute field blank (or all zeros); if he wishes to use only the absolute address, he must leave the symbolic field blank.

Table 5. Unique Relocating Loader Functions

Functions	Cards
Relocating. Can place the instructions and constants of a program segment into storage locations other than those assigned by the assembler; that is, relocate them.	Set Location Counter (SLC), Include Segment (ICS), External Symbol Dictionary (ESD, type 0), Text (TXT), Replace (REP).
Linkage. Loads two or more program segments one after the other, and completes linkage among them, so that one program segment may refer to constants and/or instructions within another program segment. (Makes any changes necessary in evaluating address constants which are used by the program segment.	External Symbol Dictionary (ESD types 1 and 2), Relocation List Dictionary (RLD), Replace (REP).
Transferring Control. Ends loading and causes control to be transferred according to the priority noted in the discussion of the Load Terminate card.	Load Terminate (LDT) and Load End (END).
Note: The function of the Replace card is essentially the same as in the absolute loader. The Load End card remains an essential part of each program segment, but is subordinate in function to the Load Terminate card.	

In the absence of an initial SLC card, LOCCT is set to the first location available for loading above 127 decimal or 7F hexadecimal.

Table 6 defines the contents of the Set Location Counter card.

Table 6. Set Location Counter Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	SLC. Identifies the type of load card.
5-6	Blank.
7-12	Address in hexadecimal (to be added to the value of the symbol, if any, in columns 17-22). The address must be right-justified in these columns, and unused leading columns filled in with zeros.
13-16	Blank.
17-22	Symbolic name, whose internal assigned location will be used by the loader. The symbol must be left-justified in these columns. If left blank, the address in the absolute field is used.
23-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

INCLUDE SEGMENT CARD

If program segment A is to be loaded, and it makes reference to a program segment named B, the relocating loader requires that the location of segment B must be already established. This requirement may be satisfied in one of two ways:

1. Load segment B first, or
2. If segment B has not been loaded, the programmer must precede segment A with an Include Segment (ICS) card. This card will define segment B by name and length.

Assuming that segment B has not been loaded but has been defined by name and length, the loader then includes segment B in its Control Dictionary and reserves an area of storage for it. (The Control Dictionary is comprised of the Reference Table and the External Symbol Identification (ESID) Table. The ESID Table contains pointers to the entries in the Reference Table that refer to the current program segment.) When the loader subsequently encounters reference to segment B, the actual location of B is already known.

When segment B is loaded, it is placed into the storage area reserved for it. The programmer must specify in the ICS card a value not less than the actual length of segment B (the length of segment B is not retained by the loader and so overlay checks are neither made nor verified). However, if another segment to be loaded, C, makes reference to another entry point within program segment B, then the assembled instructions and constants of B must either be loaded before segment C, or defined for C through an ICS card.

Entry points other than those already established (by an ENTRY assembler instruction) can be established in the same manner. To establish this type of entry point, the programmer takes the following steps:

1. He provides an SLC card that sets the location counter to the desired address. See item 3 under "Set Location Counter Card."
2. He provides an ICS card that indicates a program segment with a length of zero.

Note: Program segments are loaded only on double-word boundaries. The loader automatically makes this adjustment before loading any given segment according to the following criteria:

1. If the ICS card denotes a symbol of length 0, no adjustment is made to LOCCT, and the symbol is placed in REFTBL with the current value of LOCCT assigned to it.
2. If the ICS card denotes a symbol with a length greater than 0, then the following operations occur:
   a. LOCCT is adjusted to the next double-word boundary (if necessary).

b. the symbol goes into REFTBL with the value of LOCCT.

c. the length of the symbol is added to the value of LOCCT, and LOCCT is set to the resulting sum.

Table 7 defines the contents of the Include Segment card fields.

Table 7. Include Segment Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ICS. Identifies the type of load card.
5-16	Blank.
17-22	Name of segment, left-justified in these columns.
23-24	Blank.
25-28	Length (in bytes) in hexadecimal notation of the program segment. This must not be less than the actual length of the segment. (This may be 0 if the ICS card is used to add entry points other than for defining program segments.) The number must be right-justified in these columns, and unused leading columns filled in with zeros.
29-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

EXTERNAL SYMBOL DICTIONARY CARD (ESD)

ESD Type 0 (Program Name)

The External Symbol Dictionary card, Type 0, defines the name of the program segment. The program name is also an entry point to the segment. It is produced by the assembler when it encounters a START instruction. If the START instruction does not specify a program name or if there was no START card, BLANKS will be placed in the loader's Control Dictionary and will define the "name" of that program segment.

The assembler assigns an External Symbol Identification number of 01 (ESID 01) to the program segment. This number is used by the loader as a control (in the Control Dictionary) to the Reference Table. It is at this time, that is, when the loader is processing the ESD (Type 0) card, that the loader computes the segment's relocation factor. The relocation factor is the difference between the address where the program segment is loaded and the address where it was assembled. The loader saves the relocation factor in the Reference Table. The ESID 01 appears in the ESD Type 0, all ESDs Type 1, TEXT, RLD, and the Load End (END) cards produced by the assembler.

The starting address at which the program segment will be loaded is determined by the following conditions:

1. If the name of the segment defined by the ESD Type 0 card is contained in REFTBL, then the segment is loaded beginning at the location specified in REFTBL and no adjustment of LOCCT is made.

2. If the name of the segment specified in the ESD Type 0 card is not in REFTBL, then the following occur:

   a. LOCCT is adjusted to the next double-word boundary (if necessary).

   b. the segment name is placed in REFTBL with the adjusted value of LOCCT.

   c. the length of the segment is added to the adjusted value of LOCCT, and LOCCT is set to the resulting sum.

   d. the segment is loaded starting at the location specified in REFTBL.

The loader loads only one program segment at a time and does not save the identifying number from one program segment to another. Therefore, there is no conflict in the table when the next segment is assigned the same identifying number; that is, the next program segment loaded may be assigned an identifying number of 01 (ESID 01).

This routine maps the segment's name and its assigned location.

Table 8 defines the contents of the Type 0 External Symbol Dictionary card fields.

Table 8. ESD Card Type 0 (Program Name)

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-14	Blank.
15-16	External Symbol Identification (ESID). Number, in extended card code, assigned to the program segment.
17-22	Program name.
23-24	Blank.
25	Extended card code 12-0-1-8-9 (hexadecimal value of 00), identifying this as a program name card.
26-28	Address, in extended card code, of the first byte of the program segment as assigned by the assembler.
29	Blank.
30-32	Number, in extended card code, of bytes in the program segment.
33-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

ESD Type 1 (Entry Point)

The Type 1 External Symbol Dictionary card defines an entry point within the program segment to which another segment may refer. This card is produced by the assembler when it encounters an ENTRY assembler instruction, one card being produced for each entry point so defined. All ESD Type 1 cards are assigned the same ESID as that of the ESD Type 0 of the same program segment. Duplicate entries will cause a loader error wait. (See the publication IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.) There may not be more than 100 ENTRYs for a given program segment.

To enable reference to an entry point in one program segment, another segment must define it within its own assembly as an external symbol. However, entry points need not be predefined if they are not referenced during the load. This routine maps each entry point and its assigned location.

Table 9 defines the contents of the Type 1 External Symbol Dictionary card.

Table 9. ESD Card Type 1 (Entry Point)

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-16	Blank.
17-22	Name of entry point.
23-24	Blank.
25	Extended card code 12-1-9 (hexadecimal value of 01), identifying this as an entry point card.
26-28	Address, in extended card code, of the entry point as assigned by the assembler.
29-30	Blank.
31-32	External Symbol Identification (ESID). Number, in extended card code, assigned to program segment in which entry points occur.
33-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

ESD Type 2 (External Symbol)

The Type 2 External Symbol Dictionary card points to a name within another program segment, to which this segment may refer. The card is produced by the assembler when it encounters an EXTRN instruction, one card being produced for each external symbol so defined. The assembler assigns each External Symbol a unique ESID. The ESIDs range from 2 through 15 and so there may not be more than 14 in any given program segment.

The ESID is used as a pointer to the Reference Table which includes:

1. The external program segment name or entry point.

2. Its actual internal address.

The same ESID number appears in the RLD card associated with the external symbol.

The loader loads only one program segment at a time. It saves names from one segment to the next, but not identifying numbers. Therefore, there is no conflict in the tables when the sequence of ESIDs reappears. To reference an external symbol, that symbol must be declared an entry point in some other segment (unless it is the name of the program segment).

Table 10 defines the contents of the Type 2 External Symbol Dictionary card fields.

Table 10. ESD Card Type 2 (External Symbol)

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-14	Blank.
15-16	External Symbol Identification (ESID). Sequential number, in extended card code, assigned to external symbol.
17-22	Name of external symbol.
23-24	Blank.
25	Extended card code 12-2-9 (hexadecimal value of 02) identifying this as an external symbol card.
26-28	Extended card code 12-0-1-8-9, 12-0-1-8-9, and 12-0-1-8-9 (hexadecimal value of 000000). An address of 0 is always assigned to External Symbols by the Basic Assembler.
29-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

SUMMARY OF EXTERNAL SYMBOL DICTIONARY CARDS: The External Symbol Dictionary cards are generated by the assembler. There are three types of ESD cards:

1. ESD Type 0 defines the name, starting address, and length of a program segment. It is produced by the assembler when the assembler encounters a START assembler instruction. There is only one ESD Type 0 card produced per program segment; it is assigned an ESID of 01 by the Basic Assembler.

2. ESD Type 1 defines an entry point within the program segment to which another segment may refer. It is produced by the assembler when the assembler encounters an ENTRY assembler instruction. One card is produced for each entry point so defined.

3. ESD Type 2 points to a name within another program segment to which this program segment may refer. It is produced by the assembler when the assembler encounters an EXTRN assembler instruction.

The assembler assigns the external symbol an identifying number of from 2 through 15 (according to the order in which it is encountered among the segment's external symbols).

TEXT CARD

The Text card contains instructions and/or constants of the user's program segment and the starting address at which the first byte of text is to be loaded from the card. Each card contains a maximum of 56 bytes of text, in extended card code.

Table 11 defines the contents of the Text card fields.

Table 11. Text Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	24-bit starting address (in extended card code) in storage where the information from the card is to be loaded.
9-10	Blank.
11-12	Number of bytes (in extended card code) of text to be loaded from the card.
13-14	Blank.
15-16	External Symbol Identification (ESID). Number, in extended card code, assigned to the program segment in which the text occurs.
17-72	A maximum of 56 bytes of instructions and/or constants assembled in extended card code.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

RELOCATION LIST DICTIONARY CARD

The Relocation List Dictionary card (RLD) is produced by the assembler when it encounters a DC instruction or the second operand of a CCW instruction which defines an address as a relocatable symbol or expression. This may be the address of either an internal symbol, which occurs only within the program segment, or of an external symbol belonging to another segment (ESID with an identifying number of from 2 through 15; see "ESD Card Type 2 (External Symbol)."

For example, in program segment A, the programmer wishes to refer to a subroutine, SQRT, in segment B. He defines it as an external symbol:

```
          EXTRN       SQRT
```

Now he may branch to it within his program segment in the following manner:

```
          L           15,ADSQRT
          BALR        14,15
```

Because he does not know what its address will be at load time, he uses a symbolic address:

```
ADSQRT    DC          A(SQRT)
```

In this example, SQRT is an external, relocatable symbol, whose value will change as a result of segment B being relocated. The assembler assigns ADSQRT a value of zero, and when the address for SQRT is defined at load time, this value is added to zero. A segment may contain more than one symbol or expression definable in terms of one relocatable symbol. For example:

```
ADSQRT    DC    A(SQRT)
ADSQR1    DC    A(SQRT+10)
ADSQR2    DC    A(SQRT+20)
```

The RLD card lists addresses for as many as 13 expressions so defined. If there are more than 13 such expressions, other RLD cards associated with the symbol are produced.

Table 12 defines the contents of the Relocation List Dictionary card fields.

Table 12. Relocation List Dictionary Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	RLD. Identifies the type of load card.
5-10	Blank.
11-12	Number, in extended card code, of bytes of information in the variable field (card columns 17-72) of this card. The range is from 8 to a maximum of 56.
13-16	Blank.
17-72	Variable field (in extended card code). Consists of the following subfields: Relocation Header. (Two bytes.) An ESID with a value of from 01 through 15. Whether or not the value is 01 or from 02 through 15 depends on whether the symbol it points to is internal or external to the particular program segment. Position Header. (Two bytes.) The ESID assigned to this program segment. Flag Byte (bits 0 through 3 are not used). This byte contains three items: 1. Size. (Bits 4 and 5.) Two bits which indicate the length (in bytes) of the adjusted address (AA Cell) a. 00 - one-byte cell b. 01 - two-byte cell c. 10 - three-byte cell d. 11 - four-byte cell

(Continued)

Table 12. (Continued)

Column	Contents
	2. Complement Flag. (Bit 6.) When this bit is a one, it means that the value (or address) of the symbol is to be subtracted from the contents of the AA Cell. When this bit is a zero, the value of the symbol is to be added to the contents of the AA Cell. 3. Continuation Flag. (Bit 7.) When this bit is a one, it means that this is one of a series of addresses to be adjusted. When this bit is a zero, this is the only AA Cell to be adjusted or the last in a series using the same Relocation and Position headers. Address. The three-byte address of the location of the AA Cell. The Flag Byte and Address may be repeated for AA Cells as long as the continuation flag bit is on in the current four-byte entry.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

REPLACE CARD

The Replace card is supplied by the programmer, and should be placed in the program segment immediately following the Text cards. Both instructions and constants may be changed and/or additions made. The Replace card must be punched in hexadecimal code.

If additions made by Replace cards increase the length of a program segment, the programmer must place an Include Segment card (which defines the total length of that program segment) at the front of the program segment.

Table 13 defines the contents of the Replace card fields.

Table 13. Replace Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	REP. Identifies the type of load card.
5-6	Blank.
7-12	Starting address, in hexadecimal, of the area to be replaced, as assigned by the assembler. It must be right-justified in these columns, and unused leading columns filled in with zeros.
13-14	Blank.
15-16	External Symbol Identification (ESID). Hexadecimal number assigned to the program segment in which replacement is to be made.
17-70	A maximum of 11 four-digit hexadecimal fields, separated by commas, each replacing one previously loaded half-word (two bytes). The last field must not be followed by a comma.
71-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

LOAD END CARD

The Load End card (END) is produced by the assembler when it encounters the END instruction. This card ends loading of a program segment and may specify a location within the segment to which control is to be transferred.

Table 14 defines the contents of the Load End card fields.

Table 14. Load End Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	END. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the program segment to which control may be transferred at the end of the loading process. See the conditions and priority discussed under Load Terminate card.
9-14	Blank.
15-16	External Symbol Identification (ESID).
17-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

LOAD TERMINATE CARD

The Load Terminate card (LDT) must be placed at the end of the program segment. It has two uses:

1. It is needed to end the loading process.
2. It causes control to be transferred to some location within the segments loaded.

The specific location to which control is transferred is determined through the following order of priority:

1. Control is always transferred to a location specified in a Load Terminate card.
2. If the Load Terminate card does not specify a location, control is transferred to the location specified by the first Load End card encountered during the current loading process.
3. If neither the Load Terminate card nor any of the Load End cards specifies a

location, control is transferred to the first location loaded into from a TXT card (or REP card, if there are no Text cards), above 127 decimal or 7F hexadecimal, of the first program segment loaded.

When control is transferred to the program segment(s) loaded, the system operates in the Supervisor state, disabled for all interruptions except a machine check interrupt; see "Input/Output Support Package" for a discussion of interruptions.

Figure 2 shows a possible sequence of cards, in a series of program segments, ready to be loaded by the relocating loader; it does not show all permissible combinations of load cards. (The figure reads from the bottom.)

OTHER FEATURES

In addition to the relocating loader's basic functions, it can be used for two other operations:

1. To implement a technique that allows execution of programs larger than available storage, that is, an overlaying load procedure.
2. To operate in the same way as the absolute loader.

A description of these operations follows.

Table 15 defines the contents of the Load Terminate card fields.

Table 15. Load Terminate Card

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	LDT. Identifies the type of load card.
5-16	Blank.
17-22	Name of entry point to the program segment, left-justified in these columns. Use of this field is optional.
23-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

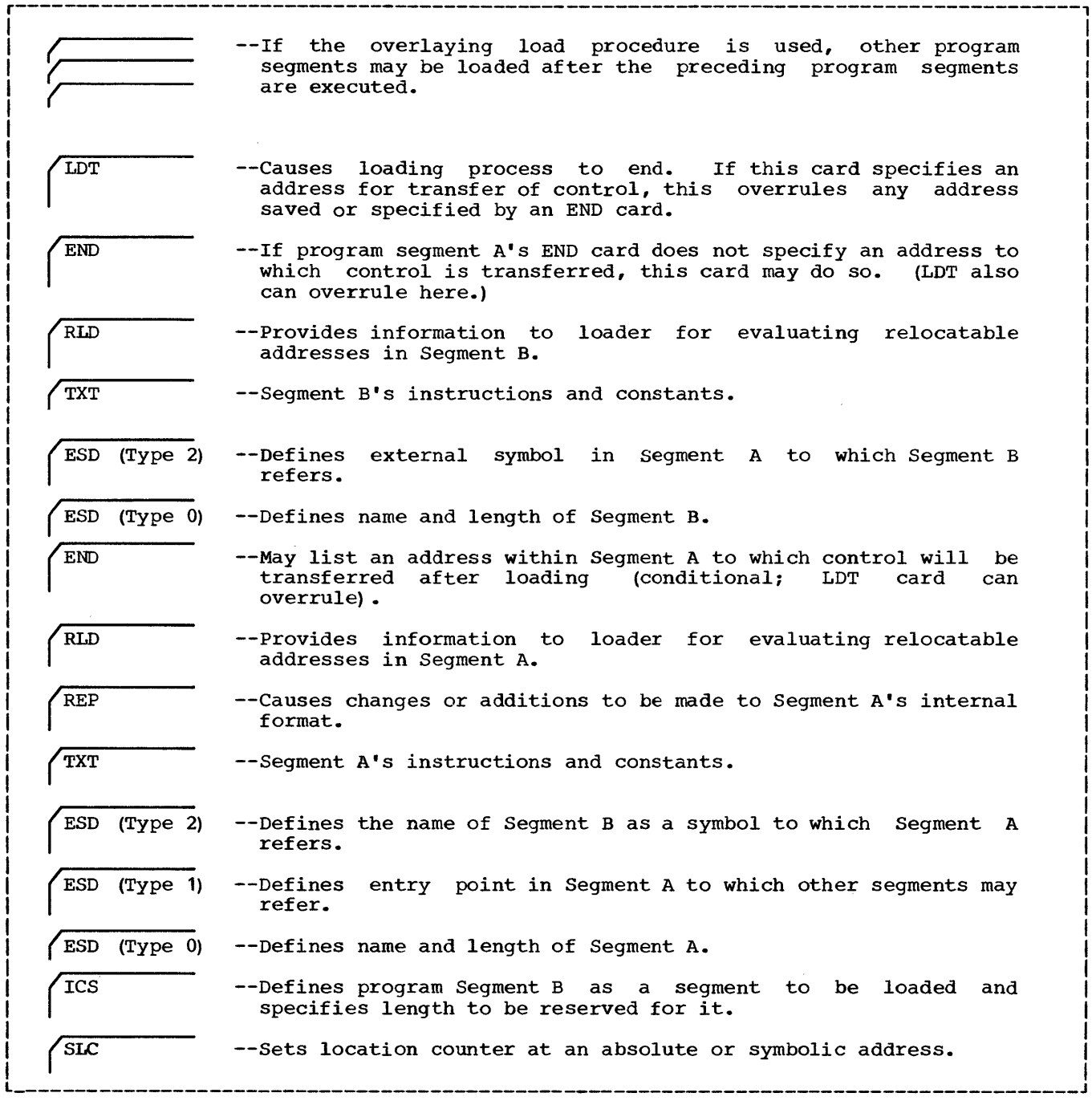


Figure 2. Two Program Segments Ready for Loading by Relocating Loader (This figure reads from bottom to top.)

OVERLAYING LOAD PROCEDURE

The overlaying load procedure allows the programmer to execute programs larger than available storage. The general principle is that once a loaded program segment is no longer needed, another program segment may be loaded over it. The process of overlaying the segments no longer needed with another program segment is continued until all the program segments are executed.

More specifically, the first segments are loaded in the usual manner. The loading procedure would then be interrupted by an LDT card which would transfer control to one of the loaded segments. When the

loaded segment has completed its operations, the program segment would transfer control back to the loader to load the next program segment. The considerations for doing this are described in the next paragraph.

The relocating loader defines, as a built-in entry point, a location named RESUME. If the loader is entered at this location, loading will resume at the location specified in LOCCT, which has not been reset or changed after loading the previous segment; the programmer can reset LOCCT by an SLC card.

The relocating loader may be entered at RESUME by the following coding sequence in the program segment:

```
        EXTRN  RESUME       Define RESUME to
                            the segment
          .
          .
          .
        L      1,RESADD     Load address of
                            RESUME
        BCR    15,1         Branch to RESUME
          .
          .
          .
RESADD  DC     A(RESUME)    Define address of
                            RESUME
```

If the first card the loader encounters is an SLC card which sets LOCCT to the same starting address the previous program segment had occupied, the previous segment will be overlaid. Consider the following example:

A user has a 16K machine. He has inventory records that show:

1. Quantity on hand at the beginning of the month.
2. The number of items sold during the month.
3. The number of items purchased during the month.
4. The minimum re-order figure.

These inventory records occupy 4000 bytes of storage.

He has a program segment to perform each of the following operations:

1. Subtract the number of items sold from the quantity on hand at the beginning of the month; program segment J.
2. Add the number of items purchased; program segment K.
3. Compare the items on hand to the minimum re-order figure and move those items which must be re-ordered to an output buffer area; program segment L.
4. Print a list of the current inventory on hand; program segment M.
5. Print a list of the items to be re-ordered; program segment N.

Each of these five program segments occupies 1500 bytes of storage and the output buffer occupies 250 bytes of storage. Finally, the relocating loader occupies 3800 bytes of storage and the user's I/O routines occupy 1000 bytes of storage.

Since the entire program is larger than available storage, the programmer uses the overlaying load procedure as follows:

1. He loads the loader, the list of his inventory, and the first program segment. He then interrupts the loading procedure with a Load Terminate card, which transfers control to one of the loaded segments; in this case, program segment J, and execution proceeds until all the inventory categories have been processed by this program segment.
2. Program segment J then transfers control to location RESUME, and the next program segment -- program segment K -- is loaded. The first card in program segment K is an SLC card which uses the name of program segment J as the address to which the location counter is to be set. Thus, program segment K would overlay program segment J. In this illustration, the second program segment would overlay the first, which is no longer needed.
3. Control is again transferred to one of the program segments by interrupting the loading procedure with a Load Terminate card, and execution proceeds. the program segments he no longer needs with another program segment until the lists of inventory on

hand and items to be re-ordered are printed (always making sure that he does not attempt to overlay the loader or the other segments).

LOADING IN ABSOLUTE FORM

The relocating loader operates in a manner similar to the absolute loader, if the External Symbol Dictionary card (ESD type 0) is removed from the program segment before load time.

Note: The loader will not record in the Reference Table the presence of a program segment loaded in absolute form. The loader loads one or more segments in absolute form until it encounters a Load Terminate card. (Load End card will not terminate loading.) It also loads program segments in both absolute form (without ESD type 0 cards) and in relocatable form. However, the following limitations apply to this situation:

1. No linkage is provided with any program segment loaded in absolute form. If the programmer wishes to load at the locations assigned by the assembler with linkage to another segment, he must specify the starting address with a Set Location Counter card and must not remove the ESD type 0 card.

2. If two or more program segments are loaded in absolute form, one will overlay the other at all common addresses.

LOADER USE OF I/O SUPPORT PACKAGE

The relocating loader uses selected modules of the I/O Support Package to read cards or card images from tape and, if a writing device (typewriter or printer) is indicated to the loader, storage mapping and error messages will also be written. These routines can be used by the programmer by employing the coding sequence (with absolute addresses) discussed in "Input/Output Support Package."

RESIDENT LOADER CONSIDERATIONS

The name of the first instruction in the relocating loader is: LOAD2. If this location is branched to (either from the console or from a program segment in storage that defines LOAD2 as an EXTRN), another program segment can be loaded without preceding it with another relocating loader.

CAUTION: The user cannot use LOAD2 for an overlaying load procedure, since the Reference Table is destroyed whenever LOAD2 is branched to.

See appendixes for further information about relocation and linkage, and the creation of self-loading loaders.

INTRODUCTION

The dump program is designed to provide a listing of the contents of all or part of storage, the general registers, and the floating-point registers (or any combination of these). To be more specific, at the option of the user, the dump program can produce a listing of any or all of the following:

1. Console listing; that is, a listing of storage locations from zero through 127. This listing includes:
   a. Initial Program Loading PSW: locations 0-7
   b. Initial Program Loading CCW1: locations 8-15
   c. Initial Program Loading CCW2: locations 16-23
   d. External Old PSW: locations 24-31
   e. Supervisor Call Old PSW: locations 32-39
   f. Program Old PSW: locations 40-47
   g. Machine check old PSW: locations 48-55
   h. Input/output Old PSW locations 56-63
   i. CSW: locations 64-71
   j. CAW: locations 72-75
   k. Unused word: locations 76-79
   l. Timer: locations 80-83
   m. Unused word: locations 84-87
   n. External New PSW: locations 88-95
   o. Supervisor call new PSW 96-103
   p. Program New PSW: 104-111
   q. Machine Check New PSW: locations 112-119
   r. Input/output New PSW: locations 120-127
2. The sixteen general registers.
3. The four floating-point registers.[1]
4. All or part of storage.

The listing is printed on the IBM 1403 or 1443 Printer or on the IBM 1052 Printer-Keyboard.

FEATURES

The dump program has the following features:

1. Listings may be taken at any point during execution of the user's program.
2. The user may choose any of eight basic formats for the listing and may include several storage areas in different formats within the same listing.
3. Lengths of the areas to be listed, and, with two of the output formats, the length of the items within the area, may be specified.
4. Request numbering allows the user to provide for several listings in his source program, but to call for only those listings needed during a particular run.
5. Each storage area listed may be assigned an identifying label of eight characters, which will immediately precede the listing of the storage area.

VERSIONS OF THE DUMP PROGRAM

There are two versions of the dump program: the single-phase version and a two-phase version. (See the "Introduction" to this manual for an approximation of the storage required for each of the versions of the dump program.) The single-phase version is available as follows:

---

[1]If the floating-point registers are requested on a machine without the floating-point feature, a program error wait will occur and the program will not continue.

1. A symbolic deck that may be assembled by the user at the locations he desires and loaded by either the absolute or relocating loader; this deck provides all the facilities listed in "Features."

2. An assembled deck that may be loaded by either the absolute or relocating loader; this version provides all the facilities listed in "Features."

3. A self-loading deck which is assembled beginning at Location 128 decimal; it provides hexadecimal listings only. This deck provides the user with the advantage of dumping storage when his program has come to an unexpected stop. It also allows the user to dump the contents of storage after execution of his program.

The two-phase version is supplied as follows:

1. Phase 1 of the Two Phase Dump is available as (a) a symbolic deck that must be assembled and (b) an assembled deck that may be loaded by either the absolute or relocating loader.

2. Phase 2 is available as (a) a symbolic deck that must be assembled and (b) as a self-loading deck.

Each of the phases is loaded and executed separately.

Thus, this version provides the advantage of conserving storage, since only Phase 1 is resident during execution of the user's program.

The single-phase version program is discussed in the body of this section. The two-phase version is discussed in "Two-Phase Dump," and the self-loading version of the dump program is discussed in "Self-Loading Dump Program."

REQUEST NUMBERS

Two bytes of storage, beginning at symbolic location RTBL, are used by the dump program as binary switches indicating the status of request numbers. The 16 bit positions, beginning with zero in the high-order position, correspond to the 16 possible request numbers -- 0 through F. The presence of a bit indicates that a storage print is to be executed if the user's call parameter includes a request number corresponding to the position of that bit. After assembly, the programmer inserts the desired mask into RTBL by a Replace card.[1] Prior to assembly, he may set the mask in the symbolic deck.

These two bytes are originally defined as DC X'8000'. This indicates that a request specification of zero will result in the execution of a storage print, while the specification of any other request number will cause immediate return to the user's program.

Table 16 defines the fields of a Replace card used for request numbers.

Example

Symbolic locations RTBL and RTBL+1, as originally assembled by DC X'8000', may be illustrated as follows:

```
                Symbolic        Symbolic
                Location        Location
                RTBL            RTBL+1
                ↑               ↑
               ┌-------------------------------┐
Bit            |1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
               |-------------------------------|
Request        |0 1 2 3 4 5 6 7 8 9 A B C D E F|
Number         └-------------------------------┘
```

---

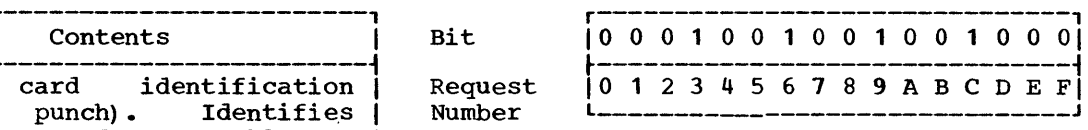
[1] The programmer must refer to his listing and place the absolute address assigned to symbolic location RTBL in card columns 7-12 of the Replace card. He will find this location near the end of his listing.

Table 16. Format of Replace Card for Request Numbers

Column	Contents
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	REP. Identifies the type of load card.
5-6	Blank.
7-12	Starting absolute address in hexadecimal as assigned by the assembler to symbolic location RTBL. The programmer must refer to the assembly listing to get this absolute address. It must be right-justified in these columns, and unused leading columns filled in with zeros.
13-14	Blank.
15-16	External Symbol Identification (ESID 01). Hexadecimal number assigned to the program segment in which the replacement is to be made.
17-20	One four-digit hexadecimal field indicating which of the bit positions in symbolic location RTBL and RTBL plus 1 are to be set to a binary one.
21-72	Blank.
73-80	Not used by the loader. The programmer may leave blank or punch in program identification for his own convenience.

Assume that the programmer finds the absolute address, as assigned by the assembler to symbolic location RTBL, to be a hexadecimal 1388 (5000 decimal); also assume that the request numbers that he wishes are 3, 6, 9, and C.

The programmer punches a hexadecimal 001388 in card columns 7-12 of the Replace card. In columns 17-20, he punches a hexadecimal 1248. After the Replace card has been loaded, the bit positions in hexadecimal locations 1388 and 1389 are:



Now if the user's call parameter includes a request number corresponding to a bit that is on (i.e., 3, 6, 9, or C), a storage print will be taken.

DUMP PROGRAM REQUIREMENTS

SINGLE-PHASE

If the single-phase dump program is being used, the user supplies (by symbolic cards prior to assembly or by a Replace card at object time) the following information to the dump program:

1. The storage capacity of the user's machine.
2. The type of output device to be used.
3. The address of the output device.
4. The address of the IBM 1052 Printer-Keyboard (if one is available for operator messages).

The storage capacity of the user's machine is supplied to the dump program by locating the following card in the dump source program[1]:

```
DSTOPL  DC      AL3(8192)
```

The user takes this card out, and if the operand field does not specify his storage capacity, he must punch a copy of this card (in decimal notation) with the storage capacity of his machine in the operand field, and put it back into the dump source program.

The type of output device that is to be used and its address are supplied to the dump program by locating the following card in the dump source program:

```
OUTDEV  DC      X'zzzzzzzz'
```

---

[1]Note: This and subsequent cards come immediately before the END card in the dump source program. Their relative order cannot be altered.

In the low-order two bytes of the operand field, he must punch the address of the output device; in the high-order two bytes, if the output device is to be the IBM 1403 or 1443 Printer, he punches 0000. For example:

```
OUTDEV  DC      X'0000Addr'
```

If it is the IBM 1052 Printer-Keyboard, he punches 0001. For example:

```
OUTDEV  DC      X'0001Addr'
```

The user then locates the following card in the dump source program:

```
TYPWTR  DC      X'zzzz'
```

If there is an IBM 1052 Printer-Keyboard available for operator messages, he punches its address in the operand field; if there is none, he should punch in the address of another printer. If there is neither, he punches this card as follows:

```
TYPWTR  DC      X'FFFF'
```

The user then puts the card back into the dump source program.

Placing hexadecimal F's in TYPWTR only disables Dump Program operator messages, not those of the I/O routines. There are two methods to disable I/O messages. They are as follows:

1. Prior to assembly remove the "Write Error Message Base Routine," from the I/O portion of the program.

2. At object time, use a Replace card to change the instruction at SAGINW+4 (in the I/O Base Routine - Group 1, Interrogate I/O Interrupt or CC1) back to the same format it had on the assembly listing.

Example

A user has a machine with a storage capacity of 65,536 bytes. He is going to make his listings on the IBM 1403 Printer, which is unit 9 on selector channel 1. He wants his messages written on the IBM 1052 Printer-Keyboard, which is unit 5 on multiplexor channel 0. He would punch the cards as follows:

```
DSTOPL  DC      AL3(65536)
OUTDEV  DC      X'00000109'
TYPWTR  DC      X'0005'
```

CALLING SEQUENCE

When the dump program and the user's program are assembled together, the user calls the dump program with the following sequence of coding:

```
        LA      15,DUMP
        BALR    14,15
```

and follows these instructions with the appropriate DC assembler instructions setting up the call parameter for the listing.

Note: When the dump program and the user's program are assembled separately and the relocating loader is being used, the programmer must define the dump program as an external symbol:

```
        EXTRN   DUMP
```

and he can call it by:

```
        L       15,ADDUMP
        BALR    14,15
```

after having generated an address:

```
ADDUMP  DC      A(DUMP)
```

The rest of the discussion on the calling sequence applies to both loaders.

Control returns to the user's program at the location immediately following the call parameter. The call parameter is one half-word if a print of storage is not desired, and three half-words, if a print of storage is desired. The call parameter specifies the following basic conditions for the listing:

1. The request number of the listing.

2. The options (see the "Introduction" to this section for a list of options) which the listing will include.

If the listing is to include storage, the number of Control List (see "Control List Format") entries and the address of the first entry must be specified. If all of storage is to be listed in 32-bit hexadecimal, the Count field of the call parameter may contain zero, and the Address field will then be ignored (but must not be omitted).

Note: Except for symbolic references, the variable fields of the DC instructions which set up the Call Parameter and Control List are usually coded in hexadecimal.

Table 17 shows the format of the call parameter.

Table 17. Call Parameter Format

1	2 3	8 9	12 13	16 17	24 25	48
Length of Parameter	Not Used	Option	Request Number	Count	Address	
00 or 11		0000 0001 0010 0011 0100 0101 0110 0111	(0-15)$_{10}$ (0-F)$_{16}$	(00-FF)$_{16}$	Address of first entry in the control list	

Bit Positions	Field Name	Significance	Hexadecimal Coding
1-2	Length of parameter	00 indicates a half-word parameter and that no storage is to be dumped. 11 indicates a three half-word parameter and that at least one area of storage is to be dumped.	00 or C0
3-8		Not used.	
9-12	Option	0000 indicates no options are exercised. 0001 print general registers. 0010 print floating-point registers. 0011 print floating-point and general registers. 0100 print console listing. 0101 print console listing and general registers. 0110 print console listing and floating-point registers. 0111 print all options.	0 1 2 3 4 5 6 7
13-16	Request Number	A four-bit hexadecimal number from 0 through F. If the corresponding RTBL bit is a one, the listing is provided; otherwise, control returns immediately to the user's program.	0 through F
17-24	Count	An eight-bit number (when symbolic address constants are used to designate addresses, this number is limited by the maximum number of address constants allowed by the Basic Assembler) which is the total number of entries in the Control List. If this number is 0, all of storage is printed in 32-bit hexadecimal format and the Address field of the call parameter is ignored (but it may not be omitted).	00 through FF
25-48	Address	The 24-bit address of the first entry in the Control List. If symbolic, it is coded separately as: DC AL3 (symbol).	If absolute a 1 to 6 digit number

Examples of the required call-parameter coding follow. (Each example assumes that the corresponding request number has been specified.)

Example 1

```
         LA          15,DUMP
         BALR        14,15
DUMP1    DC          X'0034'
```

where:

00 indicates a half-word call parameter and that no storage is to be dumped.

3 indicates that the floating-point and general registers are to be dumped.

4 is the request number.

In this example, the general and floating-point registers are listed, and control returns to DUMP1 + 2.

Example 2

```
         LA          15,DUMP
         BALR        14,15
DUMP2    DC          X'C00000000000'
```

where:

C0 indicates a three half-word call parameter and that at least one area of storage is to be dumped.

0 since the count field is zero, no options are to be exercised.

0 is the request number.

00 is the control list entry, so all of storage will be listed in 32-bit hexadecimal format. Control returns to location DUMP2 + 6.

000000 is the address field. Since the count field is 0, this field is ignored but may not be omitted.

Example 3

```
         LA          15,DUMP
         BALR        14,15
DUMP3    DC          X'C01A04'
         DC          AL3(LIST)
```

where:

C0 indicates a three half-word call parameter and that at least one area of storage is to be dumped.

1 indicates that the general registers are to be printed.

A is the request number.

04 is the number of Control List entries.

LIST is the address of the first Control List entry.

The general storage registers and the four storage areas specified by the Control List entries beginning at location LIST are to be dumped. Control returns to location DUMP3 + 6.

CONTROL LIST FORMAT

The Control List consists of a maximum of 255 entries. Each entry specifies the following:

1. An area of storage to be listed.
2. How it is to be listed: in what format it is to be listed and the length in bytes of each item to be listed (where not implied by the format).
3. The address of the first byte of the area to be listed.
4. Whether the End Flag field specifies an end address plus 1 location or a count of bytes.
5. Whether or not there is a dump identification label.
6. The size of the area is defined in the End/Count field of the Control List entry either by the address of the last byte plus 1 or by the number of bytes in the area.

If the programmer assigns an identifying eight-byte label to an area, he places the label as the second double-word of the Control List entry. When printed, the label precedes the listed area.

Table 18 shows the format of the Control List Entry.

Table 18. Control List Entry Format

1	2	3 4	5 8	9 32	33 40	41 64	65 128
Label Flag	End Flag	Not Used	Format Code	Starting Address	Length	End/Count	Label
0 or 1	0 or 1		(0-A)$_{16}$	Address of first byte of the area to be listed	(01-10)$_{16}$	Either an End Address +1 Location, or a count in bytes of the area to be listed (see "End Flag").	Optional eight-byte label (2 words)

Bit Positions	Field Name	Significance
1	Label flag	0 indicates that no label is associated with the area. 1 indicates that there is a label associated with the area.
2	End flag	0 indicates that the End/Count field is interpreted as a Count. 1 indicates the End/Count field is interpreted as an end address plus 1.
3-4		Not used.
5-8	Format code	A four-bit hexadecimal number, zero through A, specifying the list format (see "Output Formats").
9-32	Starting Address	The 24-bit address of the first byte of the area to be listed. The area must be properly aligned on a half-word, full-word, or double-word boundary, according to the format requested. If symbolic, it is coded separately as: AL3(symbol).
33-40	Length	An eight-bit number -- 1 through 16 -- specifying the length in bytes of each item. Used only with items of variable length having format codes of 0, 1, 2, or 3. If not used, it may be coded as: 00.
41-64	End/Count	If the End flag is zero, this is the number of bytes to be listed, right-justified. If the End flag is one, this is a 24-bit address of the end of the area plus 1 that is to be listed. If symbolic, it is coded separately as: AL3 (symbol).
65-128	Label	An optional eight-byte label (if less than eight characters, blanks must be included), present only when the Label flag is one.

Examples of the required coding follow.

```
LIST DC X'C8'          Label flag; end
                       flag; format 8
     DC AL3(START)     Starting address
     DC X'00'          Length field is
                       ignored (because
                       format 8
                       is specified)
     DC AL3(END+1)     End address + 1
     DC C'COREDUMP'    Label field

     DC X'88'          Label flag;
                       Count; format 8
     DC AL3(SINE)      Starting address
     DC X'00000200'    Length field is
                       ignored (because
                       format 8 is
                       specified) ; Count
     DC C'SINEDUMP'    Label field

     DC X'42'          No label flag;
                       end flag;
                       format 2
     DC AL3(DATA)      Starting address
     DC X'10'          Length field of 16
     DC AL3(DATA+400)  End address + 1
```

The three list entries above would produce listings of the following:

1. The label COREDUMP, followed by the area from START through END, in hexadecimal half-words with mnemonics.

2. The label SINEDUMP, followed by the 512 bytes starting at SINE, in hexadecimal half-words with mnemonics.

3. The area from DATA through DATA+399, in hexadecimal, each item 16 bytes long.

OUTPUT FORMATS

Listings produced by the dump program contain as many complete items per line as the length of the item permits. In the case of format types 0, 1, 2, and 3 (shown in Table 19), the length of an item is defined by the Length field (bit positions 33-40) of the Control List Entry; in the case of types 4 through A, it is implied by the format.

The dump program has one error message intended for the use of the programmer. This error message, which may be produced by either the single-phase dump or Phase 2 of the two-phase dump, will appear on the listing as follows:

```
DCI   Control List Error...
      This Request Skipped
```

Table 19. Output Formats

Code	Format
0 or 2	Hexadecimal. Each byte is decoded to two hexadecimal digits. Length is as specified in the Length field.
1 or 3	Each byte is printed as an alphabetic or zoned decimal character. Length is as specified in the Length field.
4 or 8	Hexadecimal half-word with mnemonics. Each half-word is decoded to four hexadecimal digits, and interpreted mnemonic operation codes appear beneath each instruction. NOTE: Data whose bit configuration coincides with that of an operation code is also accompanied by a mnemonic. If a bit combination which does not represent a valid mnemonic is encountered, an X will appear below the high-order digit of the address in the left-hand margin.
5	Hexadecimal full-words without mnemonics. Length of each item is four bytes.
6	Short-precision floating-point decimal. Each full-word of binary data is converted to eight decimal digits, with sign and exponent. Negative numbers appear in true form.
7	Long-precision floating-point decimal. Each double-word of binary data is converted to 17 decimal digits, with sign and exponent. Negative numbers appear in true form.
9	Half-word fixed-point decimal. Each half-word of binary data is converted to decimal with a sign. Negative numbers appear in true form.
A	Full-word fixed-point decimal. Each word of data is converted to decimal with a sign. Negative numbers appear in true form.

This message will occur whenever an invalid condition is encountered in the Control List Entry. The error may be caused by a

Call Parameter which does not contain a valid Control List Address.

Finally, when the floating-point formats are used, the printed fraction will not differ by more than one in the low-order position from the exact decimal representation rounded to eight (short-precision) or 17 (long-precision) places.

Table 19 shows the output formats of the dump program. See Figure 3 for a sample listing of each of the output formats. (Note: When a format that prints mnemonics is being used, the user may find the character X beneath the high-order digit of the location specifier and on the same line as the mnemonics. If this occurs, it means that at least one invalid operation code was encountered on that line.)

TWO-PHASE DUMP

As mentioned in "Versions of the Dump Program," the dump program is also available in a two-phase version. These phases are loaded and executed separately to conserve main storage; the first phase produces nonedited data which is used by the second phase to produce listings in the same formats that the single-phase operation does; calling sequence and parameter formats are the same as in the single-phase operation.

The user supplies certain information to the two-phase dump program as he had to do in the single phase dump program. Therefore the user supplies Phase 1 (in the source program or by a Replace card at object time) with the following information:

1. The storage capacity of his machine.
2. The type of device to be used for output.
3. The address of the output device.
4. The address of the IBM 1052 Printer-Keyboard (if one is available for operator messages).

The storage capacity is provided to Phase 1 source program by locating the following card[1]:

```
DSTOPL  DC      AL3(8192)
```

The user takes this card out, and if the operand field does not specify his storage capacity, he must punch a copy of this card (in decimal notation) with the storage capacity of his machine in the operand field, and put it back into the Phase 1 source deck.

The type of output device that is to be used and its address are supplied to the Phase 1 source program by locating the following card:

```
OUTDEV  DC      X'zzzzzzzz'
```

In the low-order two bytes of the operand field, he must punch the address of the output device; in the high-order two bytes, if the output device is to be tape unit, the user punches 0000. For example:

```
OUTDEV  DC      X'0000Addr'
```

If the output device is to be the IBM 2540 Card Reader-Punch, the user punches 0001. For example:

```
OUTDEV  DC      X'0001Addr'
```

If the output device is to be the IBM 1442 Card Reader-Punch, the user punches 0002. For example:

```
OUTDEV  DC      X'0002Addr'
```

The user then locates the following card in the Phase 1 dump source program:

```
TYPWTR  DC      X'zzzz'
```

If there is an IBM 1052 Printer-Keyboard available for operator messages, he punches its address in the operand field; if there is none available, he should punch in the address of another printer. If neither are available, he punches it as follows:

```
TYPWTR  DC      X'FFFF'
```

The user then puts the cards back into the Phase 1 source deck.

Placing hexadecimal F's in TYPWTR only disables Dump Program operator messages, not those of the I/O routines. There are two methods to disable I/O messages. They are as follows:

1. Prior to assembly, remove the "Write Error Message Base Routine" from the I/O portion of the program.
2. At object time, use a Replace card to change the instruction at SAGINW+4 (in the I/O Base Routine - Group 1, Interrogate I/O Interrupt or CC 1) back to

---

[1]Note: The cards to be punched for Phase 1 come immediately before the END card in the Phase 1 source deck. Their relative order cannot be altered.

the same format it had on the assembly listing.

If using the Phase 2 source program, the user must supply (by symbolic changes to the source program or by a Replace card to the assembled relocatable deck at object time) the following:

1. The type of output device to be used and its address.
2. The type of input device to be used and its address.
3. The address of the typewriter (if one is available).

The type of output device that is to be used and its address are supplied to Phase 2 by locating the following card in the Phase 2 source program[1]:

```
OUTDEV  DC      X'zzzzzzzz'
```

In the low-order two bytes of the operand field, he must punch the address of the output device; in the high-order two bytes, if the output is to be printed on the IBM 1403 or 1443 Printer, the user punches 0000. For example:

```
OUTDEV  DC      X'0000Addr'
```

If the output is to be written on the IBM 1052 Printer-Keyboard, the user punches 0001. For example:

```
OUTDEV  DC      X'0001Addr'
```

The input device to be used and its address are supplied to Phase 2 by locating the following card in the Phase 2 source program:

```
INDEV   DC      X'zzzzzzzz'
```

In the low-order two bytes, he must punch the address of the input device; in the high-order two bytes, if the input is to come from tape, the user punches 0000. For example:

```
INDEV   DC      X'0000Addr'
```

If the input is to come from cards, the user punches 0001. For example:

```
INDEV   DC      X'0001Addr'
```

The user then locates the following card in the Phase 2 dump source program:

---

[1]Note: The cards to be punched for Phase 2 come immediately before the END card in the Phase 2 source program. Their relative order cannot be altered.

```
TYPWTR  DC      X'zzzz'
```

If there is an IBM 1052 Printer-Keyboard available for operator messages, he punches its address in the operand field; if there is none available, he should punch in the address of another available printer. If neither are available, he punches it as follows:

```
TYPWTR  DC      X'FFFF'
```

Placing hexadecimal F's in TYPWTR only disables Dump Program operator messages, not those of the I/O routines. There are two methods to disable I/O messages. They are as follows:

1. Prior to assembly, remove the Write Error Message Base Routine from the I/O portion of the program.
2. At object time use a Replace card to change the instruction at SAGINW+4 (in the I/O Base Routine - Group 1, Interrogate I/O Interrupt or CC 1) back to the same format it had on the assembly listing.

If the user wishes to use the self-loading version of Phase 2. (A Phase 2 relocatable assembled deck can not be loaded by either the absolute or the relocating loader on an 8K machine) the following information must be supplied:

1. The type of output device and its address.
2. The type of input device and its address.
3. The address of the IBM 1052 Printer-Keyboard (if one is available for operator messages).

The user supplies this information by taking out the END card from the self-loading deck of Phase 2 of the Two-Phase Dump and punching this card as follows:

Columns 17-20	The address of the output device, printer, or IBM 1052 printer-Keyboard, that to be used.
Column 21	0 if a printer is to be used, or 1 if an IBM 1052 Printer-Keyboard is to be used.
Columns 22-25	The address of the input device that is to be used.
Column 26	0 if the input is to come from tape, or

1 if the input is to come from cards.

Columns 27-30 The address of the IBM 1052 Printer-Keyboard, if one is available for operator messages. If none is available, he must punch it as: FFFF.

I/O error messages are only displayed on the console during error waits when the self-loading deck supplied by IBM is used.

A user with a machine larger than 8K can make more efficient use of Phase 2 of the Two-Phase Dump by altering the source program for residence in higher storage and increasing the buffer size. (Both of the preceding are noted on the assembly listing.) The assembled deck can then be loaded by either the absolute or relocating loader.

Phase 1 is resident in storage during execution of the user's program. It occupies much less storage than the single-phase dump program and it may be called as often as necessary during the execution of the user's program.

The output of Phase 1 is in Text (TXT) card format (formats of Text cards are discussed in the sections on both the absolute and relocating loaders); when Phase 2 is loaded at the termination of the job (or at the end of the day), all of storage is available for its use.

1. Sequence

   a. Phase 1 dumps the contents of storage and/or registers, according to the options listed in the "Introduction," onto DMP and TXT cards, or as card images on tape. Storage is dumped on loader TXT cards or as card images on tape. (The TXT cards produced by Phase 1 can be loaded by either the Absolute or Relocating Loaders; thus, if the user programs a routine to reset the general registers and locations 0-127, and the I/O devices are repositioned, a checkpoint procedure can be facilitated.) Phase 1 does not rewind tape.

   b. At the conclusion of the user's program or at the end of the day, Phase 2 is loaded. Phase 2 initially rewinds tape. It reads the output of Phase 1, and produces listings identical with those of the single-phase program.

2. Phase 1 output

   a. DUMP (DMP) cards (or card images on tape) identified by a 12-3-9 punch in card column one. These cards contain the call parameter, locations 0-127, and the contents of the general registers (and floating-point registers, if requested).

   b. DUMP (DMP) cards for each entry in the Control List.

   c. TEXT (TXT) cards containing the data in all storage areas specified in the Control List. These cards are identified by a 12-2-9 punch in card column one.

Note: Output from Phase 1 will go into stacker one on the 1442 Card Reader-punch and into the zero stacker on the punch side if the 2540 Card Reader-Punch is being used. These cards must be loaded in the same order that they were produced by Phase 1.

SELF-LOADING DUMP PROGRAM

Both the single-phase dump program and the two-phase dump program are executed with the user's program. The self-loading deck may be executed if an unexpected stop occurs. It can be loaded directly into storage to dump the contents of storage. The self-loading deck is assembled beginning at location 128 decimal; since it is not loaded by the loaders, it cannot be altered or patched by a Replace card. It will provide a listing in full-word hexadecimal format of the following:

1. The console.
2. The general registers. Only the contents of registers 0, 7, 9, 10, and 15 remain intact after the loading process.
3. The contents of storage from the first address after the location of the self-loading dump program through the full capacity of the user's machine.

The user supplies the following to the self-loading dump program:

1. The type of output device and its address.
2. The storage capacity of his machine.
3. The IBM 1052 Printer-Keyboard address (if one is available for operator messages).

The user supplies this information by taking out the END card from the self-loading deck and punching it as follows:

Columns 17-20	The address of the printer or IBM 1052 Printer-Keyboard that is to be used.
Column 21	0 if a printer is to be used, or 1 if an IBM 1052 Printer-Keyboard is to be used.
Columns 22-26	The storage capacity (in hexadecimal) of his machine, right-justified in these columns with a zero in the high-order column.
Columns 27-30	The address of the IBM 1052 Printer-Keyboard, if one is available for operator messages. If none is available, he must punch it as: FFFF.

```
    LOC 0        0004000A00000600          LOC 8        0200060060000050          LOC 16       0200046020000050
OLD PSWS         0106004000F12345    000000035001EF84    0000000000000000    0000000000000000    8006000A0000000F
CSW   0001EEE80C000000           CAW   0001EEE0        00000000        TIMER  00000000          00000054
NEW PSWS         000000000001ED92    000400000001ED50    000400000001EDDC    000400000001EDDC    000000000001EE6C

    FPR 0   .12345678901234567 E 01   .12345678901234567 E-01   .12345678901234567 E 75   .12345678901234567 E-77

    GPR 0   00000000   11111111   22222222   33333333   44444444   55555555   66666666   77777777
    GPR 8   88888888   99999999   AAAAAAAA   BBBBBBBB   CCCCCCCC   DDDDDDDD   EEEEEEEE   FFFFFFFF

        C000L001
000000  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
000016  16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B

        C000L002
000032  0001  0002  0003  0004  0005  0006  0007  0008  0009  000A  000B  000C  000D  000E  000F  0010
000052  0011  0012  0013  0014  0015  0016  0017  0018  0019  001A  001B  001C  001D  001E  001F  0020

        C000L003
000072  000001  000002  000003  000004  000005  000006  000007  000008  000009  00000A  00000B  00000C
000096  00000D  00000E  00000F  000010  000011  000012  000013  000014  000015  000016  000017  000018

        C005
00011C  00000001  00000002  00000003  00000004  00000005  00000006  00000007  00000008  00000009  0000000A
000138  0000000B  0000000C  0000000D  0000000E  0000000F  00000010  00000011  00000012  00000013  00000014

        C001L001
000160  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
000188  C D E F G H I J K L M N O P Q R S T U V W X Y Z A B C D

        C001L005
0001D0  ABCDE  FGHIJ  KLMNO  PQRST  UVWXY  ZABCD  EFGHI  JKLMN  OPQRS  TUVWX  YZABC  DEFGH  IJKLM  NOPQR
000216  STUVW  XYZAB  CDEFG  HIJKL  MNOPQ  RSTUV  WXYZA  BCDEF  GHIJK  LMNOP  QRSTU  VWXYZ  ABCDE  FGHIJ

        C004
00025C  0580  4890  8046  1A98  48A0  8046  1A89  48B0  8046  1ABA  47F0  805A  48C0  8058  48E0  803A
          BALR  LH          AR    LH          AR    LA          AR    BC          LH          LH

00027C  14FE  43EE  8048  42EC  803E  88F0  0004  46C0  801A  58F0  8042  58E0  803E  07F1  000F  0000
x         NR    IC          STC         SRL         BCT         L           L           BCR

        C010
00029C  2147483647    2147483648-   1234567890-   1234567890    1234567890    1234567890    1234567890    0000000000
0002BC  0000000001    0000000002    0000000003    0000000004    0000000005    0000000006    0000000007    0000000008

        C009
0002DC  32767  32768-  12345  12345  12345  12345  12345  12345  12345  12345  12345  12345  12345  12545-
0002F2  12345  12345-  12345  12345  12345  12345  12345  12345  12345  12345  12345  12345  12345  12345-

        C006
000304  .12345678 E 01   .12345678 E-01   .12345678 E 75   .12345678 E-77   .12345678 E 10   .12345678 E 10
00031C  .00000000 E 00   .00000000 E 00   .00000000 E 00   .00000000 E 00   .00000000 E 00   .00000000 E 00

        C007
000330  .12345678901234567 E 75   .12345678901234567 E 00   .12345678901234567 E-77   .12345678901234567 E 00
000350  .00000000000000000 E 00   .12345678901234567 E 26   .12345678901234567 E 00   .12345678901234567 E 00
```

Console Listing

Floating-Point Registers

General Registers

Hexadecimal, one-byte length

Hexadecimal, two-byte length

Hexadecimal, three-byte length

Full-word unsigned hexadecimal

Characters, one-byte length

Characters, five-byte length

Hexadecimal, with mnemonics

Full-word fixed-point decimal

Half-word fixed-point decimal

Short-precision floating-point decimal

Long-precision floating-point decimal

Note: Main storage addresses are in left-hand margin; format of each listing is preceded by a label. Formats are identified by inserts in right-hand margin.

Figure 3. Example of Storage Print Listing

INTRODUCTION

The Input/Output Support Package consists of a modular set of subroutines which enable the user to operate input/output devices. (A module in the Input/Output Support Package is a logical sequence of coding which either sets up or executes one I/O function.) There are three types of modules in the I/O Support Package; they are:

1. Required modules. These modules must always be present when the I/O Support Package is used.

2. Optional Modules. These modules need not be present to perform the basic functions of the I/O Support Package, but can be included to expand the facilities of the basic functions. (Note: the user physically selects the modules that are required and the others that he desires from the decks supplied by IBM; see "How the I/O Support Package is Supplied".)

3. Entry modules. These modules support certain functions of a given I/O device, for example, to read a card or write tape.

FORMAT OF PRESENTATION

Each of the three types of modules that constitute the I/O Support Package is discussed separately in the following order:

1. Required modules.

2. Optional modules.

3. Entry modules.

The discussions under these three headings provide the following information:

- The listing group heading for each module is noted in the discussions. The listing of the I/O Support Package provided by IBM groups all the modules under headings which correspond to the function of that module; for example, the entry modules are grouped under the heading "I/O Call Entry Group".

- There is a set of charts provided at the end of the I/O Support Package which illustrates the relationships of all the modules. The discussions point out which chart the reader should go to for a graphic illustration of module relationships. When selecting the modules to be used for a given application, the user is strongly urged to make frequent reference to these charts and to the listing of the I/O Support Package provided by IBM. (When using the charts, the user should find the name of the module that he desires and then follow the arrow that leads from that module, taking all branches, and include every other module that the flow line intersects.)

After the entry modules have been described, their functions explained, and requirements for their use defined, the following sections are presented:

1. Calling the entry modules. This section tells what information the user's program must supply to call the entry modules.

2. Direct Linkage. This section explains a method of coding to call the entry modules when the I/O Support Package and user's program are assembled together.

3. Indirect Linkage. This section explains a method of coding to call the entry modules when the I/O Support Package and user's program are assembled separately.

The remainder of this section shows how to organize the selected modules and presents considerations for card-only and limited card-tape installations, followed by the charts which show the relationships of the entire package.

Because of the modularity of the I/O Support Package, the reader will find many relationships and dependencies among the routines. Therefore, he is urged to first read through the entire section and become familiar with the general principles that govern the use of the I/O Support Package.

HOW THE I/O SUPPORT PACKAGE IS SUPPLIED

The I/O Support Package is supplied as follows:

1. A symbolic deck, which contains the entire I/O Support Package; the user may select those modules that suit his particular needs.

2. An assembled deck which contains the entire I/O Support Package.

3. An assembled deck for use with the IBM 1052 Printer-Keyboard, the 2540 and 1442 Card Reader-Punch, and the 1403 or 1443 Printer (card-only installations); this version does not contain the entire I/O Support Package. For information on this version, see "Card-Only Installations."

4. An assembled deck for use with all the devices noted in 3, above, and the IBM 2400 Series Magnetic Tape Units (card-tape installations); this version does not contain the entire I/O Support Package. For information on this version, see "Card-Tape Installations."

PREREQUISITE CONSIDERATIONS

To understand the following discussions, the reader must be familiar with the following information:

1. The symbolic names of the entry modules and a brief description of their functions and limitations.

2. The symbolic names assigned by the I/O Support Package to the general registers. (These names may be used in place of actual register numbers).

The following is a list of the subroutine entry modules. These modules support certain functions of a given device and are subject to the limitations of the device involved. The user is cautioned that no check is made to ensure that the calling sequence (see "Calling the Entry Modules" and "Direct Linkage") for the entry modules conforms to the specifications for the particular device. For this reason, the user should be throughly familiar with these specifications as they are explained in the reference manuals for the various I/O devices.

The subroutine entry modules are as follows:

SRDCW Read a card; wait.[1]

SWMSW Write a message; wait.

SPRTW Print a line; wait.

SPUC Punch n columns; no wait; this entry is only for a device whose punch address differs from the reader address (IBM 2540 Card Reader-Punch).

SRTPW Read tape; wait.

SPCR Punch n columns; no wait; this entry is only for a device whose punch address is identical with the reader address (IBM 1442 Card Reader-Punch).

SSNSW Sense information from the designated device; wait.

SCTLW Issue specified control command; wait.

SPCMW Single-space the message unit; wait.

SPCPW Single-space the printer unit; wait.

SKIPW Printer skip to channel one; wait.

SPCRW Punch n columns; wait; this entry is only for a device whose punch address is identical with the reader address (IBM 1442 Card Reader-Punch).

SPUCW Punch n columns; wait; this entry is only for a device whose punch address differs from the reader address (IBM 2540 Card Reader-Punch).

SWTPW Write tape; wait.

SRWD Rewind tape; no wait.

SWTMW Write a tapemark; wait.

SBSRW Backspace one physical record; wait.

SBSF Backspace file; no wait.

SFSRW Forward-space one physical record; wait.

---

[1] Wherever "wait" occurs, it indicates that control does not return to the user's program until the device reaches the end of the operation, including all mechanical motion.

SFSF Forward-space file; no wait.

SBRTW Backward read tape record; wait.

Note: These subroutine entry modules may be used in any combination; however, since they are oriented to function and not to device, it is possible that some function of a given device may not be supported. For example, no combination of the entry modules will enable the user to read from the IBM 1052 Printer-Keyboard.

The general registers are referred to by symbolic names in the I/O Support Package. (Note: the user's program may use the actual register numbers if it is so desired.) The following is a list of the symbolic names used in this section equated to their corresponding actual register assignments:

```
SREGR    EQU    0
SREGZ    EQU    1
SREGA    EQU    2
SREGN    EQU    3
SREGL    EQU    4
SREGE    EQU    5
SLUBRG   EQU    6
SREGC    EQU    7
SREGS    EQU    8
SREGP    EQU    9
```

If these symbolic names are used by the user's program, they must be defined at assembly time; if the I/O Support Package is assembled with the user's program, the I/O Support Package supplies equivalence statements (see "Direct Linkage"); if the user's program is assembled separately, these names must be defined within the user's program (see "Indirect Linkage"). The I/O Support Package saves and restores these registers. All discussions in this section use the symbolic names of the general registers.

REQUIRED SUBROUTINE MODULES

The discussion of the required subroutine modules will deal with the following points:

1. The significance of the required modules.
2. The names and the group under which they can be found on the listing provided by IBM.
3. Considerations about the individual module.
4. Use of the required modules.

The reader is urged to refer to the charts at the end of the I/O section for the relationship of the other parts of the I/O Support Package to the required modules. The relationship of the required modules is illustrated on Chart AA.

SIGNIFICANCE OF THE REQUIRED MODULES

The required modules are the foundation of the I/O Support Package; they must always be included whenever the I/O Support Package is used, regardless of what entry or optional modules are selected by the user.

NAMES AND LISTING GROUP

Table 20 gives the names of the required modules and their associated modules; it also gives the group name under which they can be found on the listing provided by IBM.

Table 20. Names and Listing Group of the Required and Their Associated Modules

Names	Listing Group
Primary Call Entry Table	I/O Call Entry Group
Secondary Call Entry Table	I/O Call Entry Group
I/O Base Routine Part 1	I/O Base Routine - Group 1
I/O Base Routine Part 2	I/O Base Routine - Group 2
Multiple Unit Device-Address Routine	I/O Base Routine - Group 2
Command Operation Modifiers Routine	I/O Base Routine - Group 2
Initial New PSW Set Up Routine	I/O Base Routine - Group 2

PRELIMINARY CONSIDERATIONS

Before these modules are explained, the reader must be familiarized with the following basic considerations:

1. Each entry module must have such information as device address; this type of information is not supplied from within the entry module. To point out where an entry module obtains this information, we may divide all the entry modules into two types: "primary" and "secondary." (This is only an illustrative distinction; such a distinction will not be found in a listing of the entry modules.)

Table 21 shows which entry modules may be considered primary and which secondary.

The main difference between the primary and secondary entry modules is that the secondary entry modules are dependent on the primary call modules. The paragraph following Table 21 explains this dependence.

Table 21. Primary and Secondary Entry Modules

Primary Entry Modules	Secondary Entry Modules
SRDCW read card	SPCR punch rdr
SWMSW write msg	SSNSW
SPRTW print line	SCTLW
SPUC punch	SPCMW
SRTPW read tape	SPCPW
	SKIPW
	SPCRW
	SPUCW
	SWTPW
	SRWD
	SWTMW
	SBSRW
	SBSF
	SFSRW
	SFSF
	SBRTW

The primary entry modules are provided with the information they need to address an I/O device by the Primary Call Entry Table module. This module contains the address of the primary entry module, the device unit address (Note: The user must initially supply the addresses of his devices to the Primary Call Entry Table), and a space for an exceptional condition return address. The secondary entry modules have a similar table, the Secondary Call Entry Table; however, this table only provides the address of the secondary entry module. The unit device address and the space for an exceptional condition return address are obtained from the Primary Call Entry Table.

2. The reader will find, by referring to his listing, that what has been called I/O Base Routine - Part 1 in Table 20 consists of four modules. The names of these four modules are:

   - I/O Interrupt Entry
   - Set Up Return
   - Initiate I/O Action
   - Interrogate I/O Interrupt or Condition Code 1

   Because of their functions, they will be referred to as if there were only two modules: I/O Initiator and Interrupt Analyzer.

3. The reader will also find that what has been referred to in Table 20 as I/O Base Routine Part 2, consists of 2 modules. Their names are:

   - Save Entry Registers and Initialize CCW and CAW
   - I/O Operations Control Constants

   They are referred to as: Housekeeping and Constants area.

4. The following three routines are special cases:

   - Multiple Unit Address-Device Routine
   - Command Operation Modifiers Routine
   - New PSW Set Up Routine

   They are special cases since, under certain conditions, the I/O Support Package could be used without them. These conditions are explained in the section immediately following.

USE OF THE REQUIRED MODULES

The following discussion explains each of the required modules and considerations for their use.

Primary Call Entry Table

This table consists of primary entry module addresses, device addresses, and a space for the exceptional condition return address. This module must always be included. For example, if a primary entry module is used, the user must include:

1. The primary entry module itself, for example, SRDCW.
2. The Primary Call Entry Table.

In organizing the I/O Support Package, the Primary Call Entry Table (SINTRY) is placed first.

Secondary Call Entry Table

This table consists of the addresses of the secondary entry modules. If a secondary entry module is used, the user must include the following:

1. The secondary entry module itself, for example, SKIPW.
2. The associated primary entry module (if any): although the secondary module performs its own specific set-up functions, it branches to its associated primary entry module for all common functions. For example, the SKIPW module sets up the command parameters and the skip command, then branches to the SPRTW module which sets the printer reference and branches to the Initiate I/O portion of the I/O Base Routine. (Table 22, which appears later in the text, lists these associations.)

The Secondary Call Entry Table (SNTRY2) follows SINTRY when organizing the I/O Support Package.

I/O BASE Routine - Part 1

This part of the I/O Base Routine consists of the following:

- The I/O initiator
- The interrupt analyzer

The I/O Base Routine - Part 1 follows the SNTRY2 in organizing the I/O Support Package.

Note: All other selected modules should follow the I/O Base Routine - Part 1 and precede the I/O Base Routine - Part 2 when organizing the I/O Support Package.

I/O Base Routine - Part 2

This part of the I/O Base Routine consists of the following:

1. Housekeeping - This module must follow all other modules added after the I/O Base Routine - Part 1 and precede the constants area.
2. Constants - This area of constants must follow the housekeeping and precede all other I/O Base Routine - Group 2 modules.

Multiple Unit Device-Address Routine

When the user is employing a class of device for which the unit address changes from call to call, the Multiple Unit Address-Device Routine is required. Each time there is a new device address, this address must be loaded right-justified into the high-order 16 bits of register SREGN. When this module is present, these bits are always interpreted as a new device address. Therefore, if this module is present and a new device address is not being used, these bits should be set to zero. See "Direct Linkage" for the procedures and precautions that must be taken. This routine follows I/O Base - Part 2 when organizing the I/O Support Package.

Command Operation Modifiers Routine

When the user wishes to employ any command operation modifiers, he must use the Command Operation Modifiers Routine. He must also place the 5-bit modifier pattern in the high-order bits of register SREGA. Any such bits will be inserted in the CCW for the current call. If this module is present but modifiers are not desired, these bits must be set to zero. See "Direct Linkage" for procedures and precautions that must be taken.

This routine follows the Multiple Unit Address-Device Routine, when organizing the I/O Support Package.

New PSW Set Up Routine

When the user does not have his own routine to set up new PSWs, this routine is required. It follows the Command Operation Modifiers Routine when organizing the I/O Support Package.

SUMMARY OF REQUIRED MODULES

This section has provided a verbal description of the significance, names, group listing, and the use of the required I/O Support Package modules. The reader should refer to Chart AA for a graphic representation of these modules and their relationship to the other modules in the I/O Support Package.

OPTIONAL SUBROUTINE MODULES

The next group of modules to be discussed are the optional subroutines. These modules are not required for the basic uses of the I/O Support Package; they enable the user to expand the basic capabilities of the package.

The reader will note that if he wishes to select a module to perform a particular function, the module he selects may require the presence of one or more other modules. For this reason, the charts should be used along with the verbal descriptions. The following is the format of presentation in this section:

1. The names and functions of all of the optional subroutine modules will be presented, grouped according to the heading under which they appear on the listing provided by IBM. If, within any group, the name of a module is indented, this signifies that the module requires the presence of the last module whose name is not indented. For example, the format:

   UE BASE Routine
   UE Printer Routine

   signifies that the UE Printer Routine requires the presence of the UE Base Routine. Other first level requirements will be noted in the discussion of individual routines. However, the reader is cautioned that these discussions are intended only as an aid to understand the routines, not to point out all dependencies. Dependencies are illustrated on the charts at the end of the I/O section; the chart reference for each group is noted next to the name of the group.

2. This part also presents some practical functions that a user might select and lists the modules that are required for this function. Here also the reader should refer to the charts for second-level dependencies.

LISTING GROUP, NAMES, AND FUNCTIONS

The following pages provide the user with a brief explanation of the functions of the optional modules and their first-level requirements.

Unit Exceptional Condition (UEC) Group

Chart: DD

UE Base Routine: This routine is entered when an exceptional condition indication occurs. It directs control to the UE Specific Unit Base Routine; if that module is not present, it directs control to Set Up Unit Exception Return Address routine. If only the UE Base Routine is present, an error wait will ensue.

UE Specific Unit Base Routine: This routine enables the attachment of other routines that provide for specific reactions to a UEC on a given device. If the UE Printer Routine is present, control passes to that routine; if not, control returns to the UE Base Routine to check for the exit to the Set Up UE Return Address Routine.

Set Up Unit Exception Return Address: This routine will return control to the address specified in register SREGL. (See "Direct Linkage.")

UE Printer Routine: This routine determines if the UEC originated from the printer; if not, control returns directly to the UE Base Routine; if it did, this routine issues a Skip-To-Channel 1 instruction to the printer. (This is used to restore the printer to a line 1 position on the next page.) Control then returns to the UE Base Routine to check for the exit to the Set Up UE Return Address Routine.

I/O Base Routine - Group 1

Condition Code 1 Unit Identity Display: This routine places the current device address and device identification in the I/O Old PSW. (Chart AA)

Minor Interrupt Conditions Base Routine: This routine makes it possible to check for incorrect record length, program control interrupt, and/or attention bits. For any one of these indications, it branches to the appropriate routine, namely, Incorrect Length Record Indication Base Routine, Program Control Interrupt Base Routine, Attention Base Routine (each of these three routines requires the presence of the Minor Interrupt Conditions Base Routine). If these indications are not found, or if the appropriate module is not present, control is directed to the Interrupt Analyzer portion of the I/O Base Routine. (Chart BB)

Incorrect Length Record Indication Base Routine: This routine checks for an incorrect length record: if there is one, it branches to the Interrupt Analyzer portion of the I/O Base Routine; if not, it branches back to the Minor Interrupt Conditions Base Routine to check for a PCI indication.

Program Control Interrupt Base Routine: This routine checks for a program control interrupt: if there is one, it branches to the Interrupt Analyzer portion of the I/O Base Routine; if not, it branches back to the Minor Interrupt Conditions Base Routine to check for an Attention indication.

Attention Base Routine: This routine checks for an attention bit: if there is one, it branches to the Interrupt Analyzer portion of the I/O Base Routine; if not, it branches back to the Minor Interrupt Conditions Base Routine.

Issue Internal Call Routine: This routine is required for the operation of the following four optional modules: Internal Unit Sense Routine, Write Error Message Base Routine, Tape Retry Routine, UE Printer Routine. Each of these routines uses the Issue Internal Call Routine to save the current registers, set the internal call switch on, save the current I/O Old PSW and CSW, branch to the internal call entry, and restore, after the internal call, all the locations saved. (Chart BB)

Internal Unit Sense Routine: This routine also requires the presence of the SSNSW entry module. It saves the current general registers and branches to the Issue Internal Call Routine. When the internally called sense routine is completed, it restores the registers and I/O Old PSW and returns control to the calling routine.

Write Error Message Base Routine: This routine also requires the presence of the SWMSW entry module and the Condition Code 1 Unit Identity Display Routine. If the interrupt device is the message unit, this routine loads a wait-state PSW. If it is not, an error message will be written on the appropriate unit and the routine will then load a wait-state PSW.

Write Error Routine - Expansion 1: This routine also requires the presence of the Write Error Message Base Routine and the Binary-to-Hex Conversion into Image Routine. This routine causes the I/O Old PSW and the CSW to be written, in addition to the information provided by the Write Error Message Base Routine.

Binary-to-Hex Conversion into Image Routine: This routine converts binary bytes into two hexadecimal characters each and sets the characters in the indicated field.

Write Error Routine - Expansion 2: This routine also requires the presence of the Write Error Message Routine - Expansion 1, and the Internal Unit Sense Routine. This routine causes the six sense bytes transmitted by the device to be written, in addition to the information provided by the Write Error Message Base Routine and Write Error Message Routine - Expansion 1.

Save and Restore External New PSW: This routine saves the current External New PSW and replaces it with an External New PSW to repeat the I/O operation with channel, external, and machine check interrupts disabled. This routine requires the presence of the New PSW Set Up Base Routine (see the discussion of this module under "Required Subroutine Modules") to which it returns control. (Chart CC)

External Interrupt Base Routine: This routine determines if the interrupt is a console, timer, or external signal interrupt. If it is a console interrupt, it branches to the Initiate I/O Action portion of the I/O Base Routine; otherwise, it branches to the Interrupt Analyzer portion of the I/O Base Routine. Note: The function of this routine is to provide exits for user-supplied routines that handle timer and external signal interrupts. (Chart AA)

Unit Check Group

Chart: CC

Unit Check Base Routine: This routine will branch to the Unit Check Tape Routine when a unit check has occurred. If the Unit Check Tape Routine is not present, an error wait will ensue, unless the unit check was due to sensing a channel 9 on the printer. In this case, the unit check will be ignored, unless the user inserts his own routine.

Unit Check Tape Routine: This routine also requires the presence of the Internal Unit Sense Routine, Internal Call Routine, Tape Entry Base Routine, Tape Backspace Record Entry Routine, and Tape Forward Space Record Routine. This routine checks the device address of the source of the unit check against that of the tape device. If the source was not a tape unit, control returns to the Unit Check Base Routine; if it was, a sense command is issued to the tape unit and the sense bits are interrogated. If the sense bits indicate that the operation may be retried (and is not a data check), another attempt is made. If the new attempt is successful, processing continues. If the new attempt is unsuccessful, and the maximum number of retries have been made, control is transferred to the Interrupt Analyzer portion of the I/O Base Routine. If the sense bits indicate that a data check is present, control is transferred to the Tape Retry Base Routine; if not, or if the Tape Retry Base Routine is not present, it branches to the Interrupt Analyzer. If the sense bits indicate that the attempt may not be retried, control is transferred to the Interrupt Analyzer.

Tape Retry Routine: This routine also requires the presence of the Unit Check Tape Routine and the Control Entry module (SCTLW). This routine tries to perform the original I/O call until it is successful or until the maximum number (as specified by IBM standards) of retries has occurred. If the maximum number of retries has occurred, it branches to the Tape Read Retry Routine or the Tape Write Retry Routine or, if the proper routine is not present, to the Interrupt Analyzer portion of the I/O Base Routine.

Tape Read Retry Routine - Backspace Cleaner: This routine requires the presence of the Unit Check Tape Routine, the Tape Tape Retry Base Routine, and the Internal Unit Sense Routine. This routine performs the backspace cleaner operation by backspacing four records (or to load point, if fewer than four records have been previously read), then forward spacing to the position of the tape at the entrance to the routine. The routine then branches to re-issue the original call, if the maximum number of backspace cleaner operations has not been performed. If the maximum number of backspace cleaner operations has been performed, the routine branches to the Interrupt Analyzer portion of the I/O Base Routine.

Tape Write Retry Routine - Erase Forward: This routine requires the presence of the Unit Check Tape Routine, the Tape Retry Routine, and the Rewind Entry Routine (SRWD). This routine performs the erase forward operation and branches to re-issue the original call, if the maximum number of operations has not been performed. If the maximum number of operations has been performed, the routine branches to the Interrupt Analyzer portion of the I/O Base Routine.

I/O Call Entry Group

Chart: EE

Locate SINTRY Table Unit Block: This routine sets symbolic register SLUBRG with the proper device unit block address.

Sense Entry Locate SINTRY Table Block Exit: This routine also requires the presence of the SSNSW entry module. It will effect a branch from the SSNSW routine to the Locate SINTRY Table Unit Block routine.

Control Entry Locate SINTRY Table Block Exit: This routine also requires the presence of the SCTLW entry module. It will effect a branch from the SCTLW routine to the Locate SINTRY Table Unit Block routine.

PRACTICAL USES OF THE OPTIONAL ROUTINES

This section describes some situations in which the user would select optional routines. The situations are ordered so that the routines required follow the same order in which they were described under "Listing Group, Names, and Functions."

The discussions in this section provide more details about the optional routines, but should be supplemented by referring to

the charts, since the discussions do not reflect all module requirements that a routine might have.

If the user wishes to note and take any action in his own program on an exceptional condition indication, the following modules (Chart DD) must be included:

- UE Base Routine
- Set Up Exception Return Address

The return address to the routine in his program which is concerned with the exceptional condition indication must be loaded into register SREGL, as explained in "Direct Linkage." Whatever is in register SREGL is used as the return address.

If the user wishes to note and take action on the printer for an exceptional condition when an automatic Skip-to-Channel 1 has occurred, the following modules (Chart DD) must be included:

- UE Base Routine
- UE Specific Unit Base Routine
- UE Printer Routine
- Issue Internal Call Routine
- Set Up Exception Return Address

If, after an error wait resulting from a condition code 1, the user would like to provide for displaying the address of the I/O unit responsible, the following module (Chart AA) must be included:

- Condition Code 1 Unit Identity Display

If the user wishes to provide to check for an incorrect length record, a program control interrupt, or attention bits, the following modules (Chart BB) must be included:

- Minor Interrupt Conditions Base Routine
- Incorrect Length Record Indication Base Routine
- Program Control Interrupt Base Routine
- Attention Base Routine

If information is to be sensed by one of the selected modules, the following modules (Chart BB) must be included:

- Issue Internal Call Routine
- The SSNSW entry module

The set of sense bytes transmitted by the device will be stored in the symbolic locations in the SSNSW routine, starting at SNSA. (SNSA is the symbolic name of a six-byte area which is defined by an ENTRY instruction in the I/O Support Package. If the user defines SNSA as an EXTRN in his program, the information stored there can be made available to his program.) The user must refer to the reference manuals of the particular I/O device for information about sense bytes.

If, before an error wait occurs, the user would like to have the three identifying characters from the address portion of the current PSW written on the message device, the following modules (Chart BB) must be included:

- Write Error Message Base Routine
- Issue Internal Call Routine
- SWMSW Entry Module
- Condition Code 1 Unit Identity Display

If the user would like to further amplify this and also have the I/O Old PSW and CSW written on the message unit, he must include the four modules listed immediately above, plus the following:

- Write Error Routine - Expansion 1
- Binary-to-Hex Conversion into Image Routine

The user can expand the scope of this option to write sense information from the device that was being operated when the interrupt occurred by including the modules listed immediately above and the following:

- Write Error Routine - Expansion 2
- Internal Unit Sense Routine

If the user wishes to save the current External New PSW to repeat the I/O operation with channel, external, and machine check interrupts disabled, the following modules (Chart AA) must be included:

- New PSW Set Up Base Routine
- Save and Restore External New PSW

If the user wishes to provide for servicing console interrupts under the control of an I/O subroutine and still permit the attachment of other routines to service timer and/or external signal caused interrupts, the following module (Chart CC) must be included:

- External Interrupt Base Routine

If the user wishes to provide for tape error retries, the following modules (Chart CC) must be included:

- Unit Check Base Routine
- Unit Check Tape Routine
- Tape Retry Base Routine
- Tape Read Retry Routine (if reading tape)
- Tape Write Retry Routine (if writing tape)

If the user is employing either the control (SCTLW) or the sense (SSNSW) entry and he does not want to load the location of the device address into register SLUBRG, the following module (Chart EE) must be included:

- Locate SINTRY Table Unit Block

(The device address must appear in the high-order 16 bits of register SREGN.) If the user is employing the Locate SINTRY Table Unit Block with the SSNSW entry, he must include the following module:

- Sense Entry Locate SINTRY Table Block Exit

If the user is employing the Locate SINTRY Table Unit Block with the SCTLW entry, he must include the following module:

- Control Entry Locate SINTRY Table Block Exit

If the user wishes to interface properly with SEREP (System's Environment Recording Edit and Print), he must include the following modules:

- Issue Internal Call Routine
- Internal Unit Sense Routine
- New PSW Set Up Base Routine
- Unit Check Base Routine

SUMMARY OF OPTIONAL ROUTINES

Although the optional routines are not necessary for the basic use of the I/O Support Package, they do expand the capabilities of it. To fully understand their use, the reader should use the charts along with this section.

SUMMARY OF I/O ENTRY MODULES

Charts: EE and FF

The functions of each of the I/O entry modules are summarized in this section. If any entry module requires the presence of a module other than the ones previously defined, it will be pointed out in the discussion of that module. Finally, to avoid repetition while describing the entry modules, error halts and checking for a busy device are discussed in the following two paragraphs.

DETECTION OF ERROR CONDITIONS

The detection of an error condition may follow execution of an I/O subroutine. Some subroutines provide for a number of retries, if an error prevents successful completion of the subroutine. In all cases, if a subroutine cannot be completed successfully because of an error condition, processing halts and information pertaining to the error will appear on the operator's system console. The operator may then choose to retry through Console Interrupts, and thereby retry the routine, or he may wish to load SEREP to obtain diagnostic information. (For a complete discussion of error messages and operator actions, see IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.)

CHECK FOR BUSY DEVICE

Every device has a busy bit (the busy bit is located in bit position 7 in the word in SINTRY that contains the device address), which is set after initiation of any operation on that device; when the operation is completed, this bit is set back to zero. The programmer may want to test this bit before issuing another I/O call to the same device. Figure 4 shows a coding sequence for an object program by which the programmer can locate and test the busy bit.

In using the entry modules which have no wait for the completion of the I/O operation, testing this bit is especially important before moving new information into the output area.

When operations that do not wait for device end have been accepted by the channel, control returns to the user's program at the instruction following the calling

```
            EXTRN   SINTRY              Define Primary Call Entry Table
             .
             .
             .
            L       SREGZ,PITBAD        Load address of SINTRY
            TM      QPUC+4(SREGZ),1     Test to see if busy bit is on
            BC      1,xxx               Branch if busy
             .
             .
             .
PITBAD      DC      A(SINTRY)           Define address of SINTRY
QPUC        EQU     36                  Specify the displacement of the punch
                                        entry module from SINTRY
```

Note: The displacement of device addresses from SINTRY is obtained by adding 4 to the displacement of the associated primary entry module from SINTRY: thus, to obtain the displacement from SINTRY of the punch device, 4 is added to the displacement from SINTRY of the SPUC entry module.

Figure 4. Coding in User's Program to Test Busy Bit

sequence. When it is completed, an interrupt occurs and the busy bit is set to zero. If no error was detected, control then returns to the user's program at the point where the interrupt occurred.

FUNCTIONS OF THE I/O ENTRY MODULES

In the following discussions, it is understood that control returns to the user's program at the instruction following the calling sequence, that is, the byte following the BALR instruction. In the entry modules that wait for completion of the I/O operation (all entries whose symbolic names end in W), control does not return until completion; in the others, control returns after successful initiation of the I/O operation. If the user wants to provide for an exceptional condition return address, register SREGL must be loaded as described in "Direct Linkage" and the modules specified in "Optional Subroutine Modules" must be included. Finally, the number of bytes to be transmitted (that is, the number of bytes the programmer loads into register SREGN) must not exceed the capacity of that device, nor can it be zero, since this is an invalid byte count to the channel.

Read A Card (SRDCW)

The number of columns specified in register SREGN are read into the area specified by register SREGA.

Write a Message (SWMSW)

The number of bytes specified in register SREGN are typed by the IBM 1052 Printer-Keyboard.

Print n Columns (SPRTW)

The number of columns specified in register SREGN are written on one line.

Punch n Columns (SPUC)

The number of columns specified in register SREGN are punched. This punch entry is for use only with units which have individual punch addresses, such as the IBM 2540 Card Reader-Punch.

Read Tape n Bytes (SRTPW)

The number of bytes specified in register SREGN are read into the area specified by register SREGA. (Minimum record length is 12 bytes.)

Note: The use of this entry requires the presence of the Tape Entry Base Routine module.

Punch n Columns (SPCR)

The number of columns specified in register SREGN are punched. This entry is for use only with dual service units whose read and punch addresses are identical (IBM 1442 Card Reader-Punch).

Note: The IBM 1442 Card Reader-Punch does not advance cards automatically from the punch station; therefore, whenever it is necessary to move a card from the punch station, the user must include a dummy read-a-card calling sequence to eject the card when punching is completed, or the Command Operation Modifier Routine.

Sense (SSNSW)

To determine the status of an I/O device, a sense instruction is issued to the unit designated by symbolic register SLUBRG (General Register 6), which must contain the location of the device address cell in the Primary Call Entry Table. The set of sense bytes transmitted by the device is stored in symbolic location SNSA. The number of bytes transmitted is determined by the device, the maximum being six. The user is referred to the reference manuals of the particular I/O devices for interpretations of sense bytes. See "Sense Entry Example."

Issue Specified Control Command (SCTLW)

A control command for the operation specified through the Command Operation Modifier Routine is issued to the control device whose address is specified in register SLUBRG. See "Control Entry Example."

This entry requires the following conditions.

1. The Command Operation Modifier Routine, to specify the operation of the control command, must be included.
2. SLUBRG must contain the location of the device address at the time of entry to this routine; or the Locate SINTRY Table Unit Block Routine and Control Entry Locate SINTRY Table Block Exit module must be included, and the high-order 16 bits of register SREGN must contain the device address as it appears in SINTRY.

Single Space Message Unit (SPCMW)

A line consisting of one blank character is written on the message unit. No data parameters are necessary.[1]

Single Space Printer (SPCPW)

A line consisting of one blank character is written on the printer. No data parameters are necessary.[1]

Printer Skip to Channel One (SKIPW)

A control commmand initiating a Skip-to-Carriage Tape One is issued to the printer. No data parameters are necessary.[1]

Punch n Columns (SPCRW)

The number of columns specified in register SREGN are punched. This punch entry is for use only with dual service units whose read and punch addresses are identical (IBM 1442 Card Reader-Punch).

Note: The IBM 1442 Card Reader-Punch does not advance cards automatically from the punch station; therefore, whenever it is necessary to move a card from the punch station, the user must include a dummy read-a-card calling sequence to eject the card when punching is completed, or the Command Operation Modifier Routine.

Punch n Columns (SPUCW)

The number of columns specified in register SREGN are punched. This punch entry is for use only with units which have individual punch addresses, such as the IBM 2540 Card Reader-Punch.

---

[1]Note: Although the data registers need not be loaded for the operations so noted, the specifications (noted in "Direct Linkage") for using the Multiple Unit Address-Device Routine and Command Operations Modifiers Routine must be adhered to.

Write Tape n bytes (SWTPW)

The number of bytes specified by register SREGN are written from the area specified by register SREGA. (Minimum record length is 18 bytes.)

Note: This entry requires the Tape Entry Base Routine.

Rewind (SRWD)

The tape is rewound. When the rewind has been initiated, control returns to the user's program at the instruction following the calling sequence. No data parameters are necessary.[1]

Note: This entry requires the Tape Entry Base Routine.

Write Tape Mark (SWTMW)

A tape mark is written on the specified tape. No data parameters are necessary.[1]

Note: This entry requires the Tape Entry Base Routine.

Backspace Record (SBSRW)

The appropriate tape is backspaced over the physical record. (A tape mark is recognized as one physical record.) No data parameters are necessary.[1]

Note: This entry requires the Tape Entry Base Routine.

Backspace File (SBSF)

The appropriate tape is backspaced over the first tape mark encountered. No data parameters are necessary.[1]

Note: This entry requires the Tape Entry Base Routine.

Forward Space Record (SFSRW)

The appropriate tape is spaced forward one physical record. No data parameters are necessary.[1]

Note: This entry requires the presence of the Tape Entry Base Routine.

Forward Space File (SFSF)

The appropriate tape is spaced forward over the first tape mark encountered. No data parameters are necessary.[1]

Note: This entry requires the presence of the Tape Entry Base Routine.

Backward Read Tape Record (SBRTW)

The number of bytes specified in register SREGN are read in backward motion into the area specified by register SREGA. (Minimum record length is 12 bytes.)

CAUTION: The address in register SREGA for this routine should be the last address that is to be read into, rather than the starting address. (The user is referred to the reference manuals for the appropriate tape units for a discussion of reading in backward motion.)

Note: This entry requires the presence of the Tape Entry Base Routine.

Table 22 shows the required modules for each of the entries. The following considerations should be remembered when reading this table:

1. All required routines must be present.
2. No optional routines are included in the table.

ORGANIZATION OF THE SUBROUTINE MODULES

Once the user has selected all the modules he requires, he must then organize them in the following sequence:

1. He places the Primary and Secondary Call Entry Tables first in the deck.

---

[1]Note: Although the data registers need not be loaded for the operations so noted, the specifications (noted in "Direct Linkage") for using the Multiple Unit Address-Device Routine and Command Operations Modifiers Routine must be adhered to.

2. Then, he places the part of the I/O Base Routine that contains the I/O Initiator and the Interrupt Analyzer.

3. He may then place, in any order, all the other modules he has selected, as long as all ORG statements follow any symbol they refer to.

4. He places the second part of the I/O Base Routine, which contains the I/O Base Routine's general housekeeping and constants area.

5. If any of the following modules are selected, they would come last in the deck: Multiple Unit Address-Device Routine, Command Operation Modifiers Routine, New PSW Set Up Routine.

Figures 5 and 6 show two possible organizations of modules. The figures read from the bottom to the top.

Note: The user may follow the order he finds in examining the assembly listing of the modules as they were received from IBM.

Table 22. Module Relationships

Entry Module	Type	Additional Modules Required
SRDCW	Primary	Primary Call Entry Table.
SWMSW	Primary	Primary Call Entry Table.
SPRTW	Primary	Primary Call Entry Table.
SPUC	Primary	Primary Call Entry Table.
SRTPW	Primary	Primary Call Entry Table, Tape Entry Base Routine.
SPCR	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SRDCW.
SSNSW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Symbolic register SLUBRG must contain the location of the unit device address of the Primary Call Entry Table module.
SCTLW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Command Operation Modifier Routine, and either SLUBRG must contain the control device unit block address or the following two modules must be included: Locate SINTRY Table Unit Block Routine, Control Entry Locate SINTRY Table Block Exit.
SPCMW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SWMSW entry module.
SPCPW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SPRTW entry module.
SKIPW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SPRTW entry module.
SPCRW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SRDCW entry module.
SPUCW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, SPUC entry module.
SWTPW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SRWD	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SWTMW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SBSRW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SBSF	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SFSRW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SFSF	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.
SBRTW	Secondary	Primary Call Entry Table, Secondary Call Entry Table, Tape Entry Base Routine.

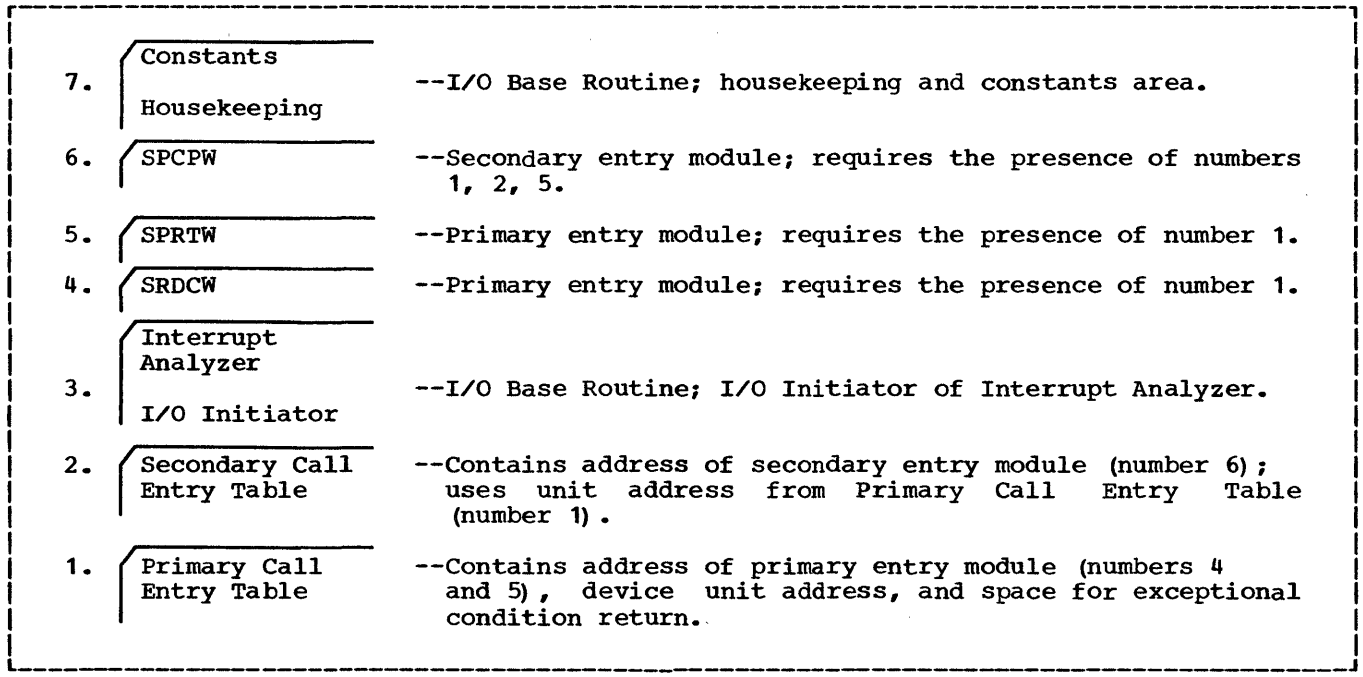


Figure 5. Organization of Subroutine Modules without Optional Routines (Read from bottom to top.)

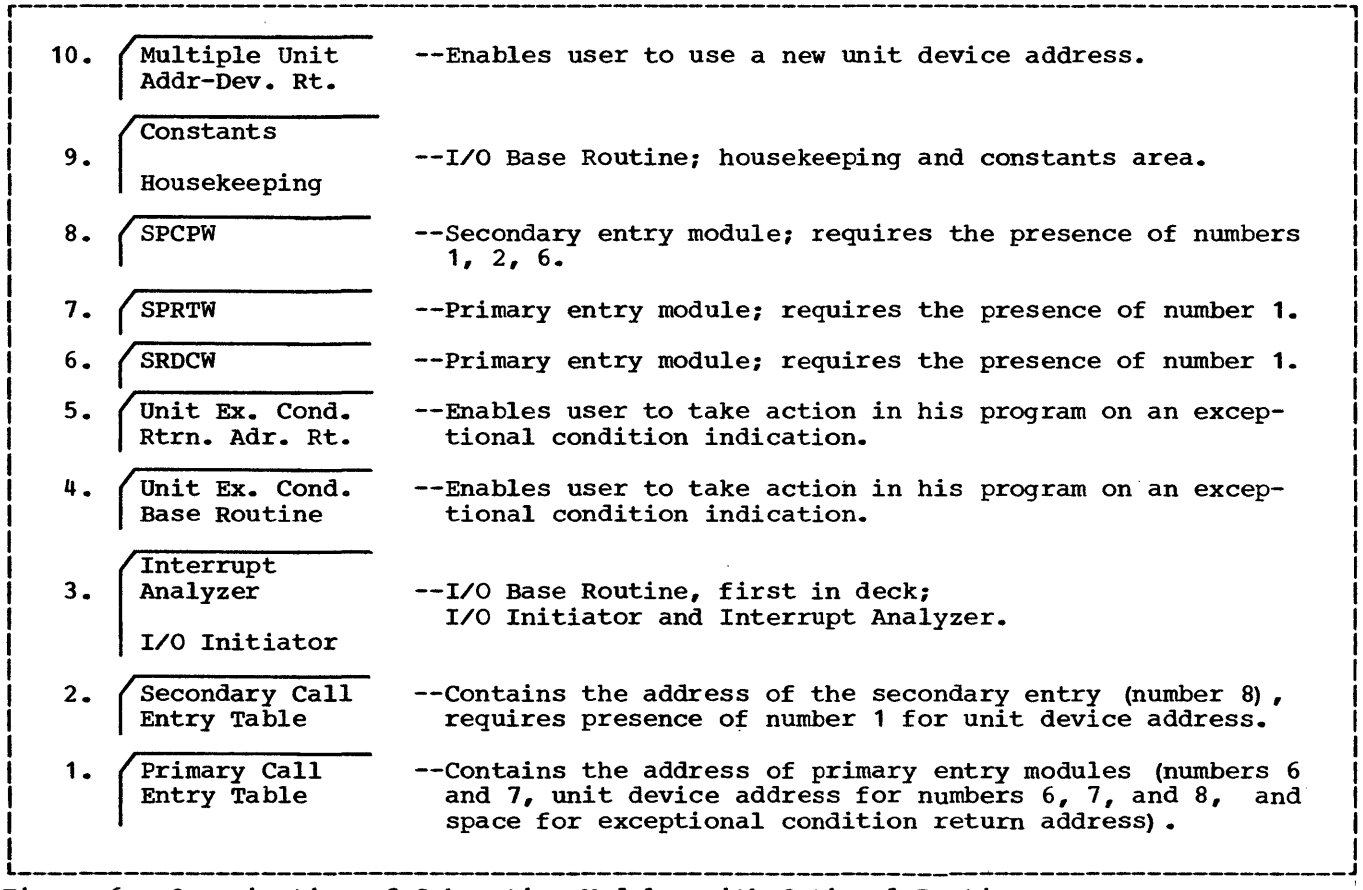


Figure 6. Organization of Subroutine Modules with Optional Routines (Read from bottom to top.)

CALLING THE ENTRY MODULES

There are two possible methods of calling the entry modules: directly and indirectly. Direct linkage may be used only when the selected modules of the symbolic I/O routines are assembled with the user's program. (In this case, the pertinent ENTRY instructions may be removed.) Indirect linkage must be used when the selected modules, in assembled form (either as supplied by IBM or those separately assembled by the user) are not assembled with the user's program. The indirect method may also be used when the selected modules are assembled with the user's program. Since the indirect method may be used in both instances, it is the preferred method.

With either of these methods, the selected entry module is called by loading the following information into general registers and transferring control by a BALR instruction:

1. The address of the I/O entry module.
2. The address of the I/O area.
3. The number of bytes to be processed.

Note: Each installation must initially supply the addresses of its I/O devices to the Primary Call Entry Table. This may be done by changing the symbolic cards prior to assembly, or by Replace cards at execution time.

Wherever an exceptional condition may occur, another general storage register (SREGL) is loaded with the return address to the routine in the user's program that uses the unit exceptional condition; for example, End of File.

DIRECT LINKAGE

The user may employ any coding sequence which provides all the information specified in "Calling the Entry Modules." (Examples of the coding follow this section.) One possible coding sequence when the user's program and I/O Support Package are assembled together is as follows:

```
LA      SREGZ,xxxx
LA      SREGA,yyyy
LA      SREGN,n
LA      SREGL,zzzz
BALR    SREGR,SREGZ
```

The following is an explanation of this coding sequence.

LA SREGZ,xxxx

Load the address of the desired entry module; for example, SRDCW.

where:

SREGZ is the general register which is loaded with the address of the desired entry module: General Register 1.

xxxx is the address of the desired entry module, for example, SRDCW, SPCR, etc.

LA SREGA,yyyy

Load the address of the first byte of data to be processed.

where:

SREGA is the general register which is loaded with the address of the first byte to be processed: General Register 2.

yyyy is the address of the first byte of data to be processed.

CAUTION: To employ any command operation modifiers, the Command Operation Modifiers Routine must be included. The user must also place the 5-bit modifier pattern in the high-order bits of register SREGA. Any such bits will be inserted in the CCW for the current call.

If the Command Operation Modifiers Routine is being employed, this instruction may be replaced by the following coding in the user's program:

```
              L     SREGA,MOBITS
              .
              .
              .
              DS    0F
MOBITS        DC    X'mm'                *
              DC    AL3(YYYY)
```

* One byte containing the modifier bit pattern.

No check is made for the validity or applicability of any such modifier bits found in register SREGA. Any future action or corrective measures for conditions produced by the user-supplied modifiers may not exist in the I/O Support Package. (See the reference manuals for the particular I/O device for bit pattern data.) Finally, the I/O Support Package always interrogates the high-order 5 bits of register SREGA;

therefore, the user should be certain that they are set to zeros if the Command Operation Modifiers Routine is present but is not being used in the current call.

LA SREGN,n

Load the number of bytes (may not exceed 65,535) to be processed.

where:

SREGN is the general register which is loaded with the number of bytes of data to be processed: General Register 3.

n is the number of bytes of data to be processed.

The high-order 16 bits of this register may be used to hold the address of a new device which was not specified in the source program. (The original device address is supplied by the programmer to SINTRY in his I/O package source program. Corrections to the device address in SINTRY may be made at assembly time by symbolic card changes, at load time by Replace cards, and at execution time by manual stores from the console.) This may be done by including the Multiple Unit Address-Device Routine and loading the new address into the high-order 16 bits of register SREGN by the following coding in the user's program:

```
                L     SREGN,DEVADR
                .
                .
                .
                DS    0F
DEVADR          DC    X'Addr'
                DC    H'n'
```

However, when the Multiple Unit Address-Device Routine is present, any bits found in the high-order 16 bits of register SREGN are always interpreted as a device address. Therefore, when an alternate device address is not going to be used, the programmer should be certain these bits are set to zeros.

LA SREGL,zzzz

Load the return address to that routine in the user's program which uses an exceptional condition indication. (If the user desires this option, he must include the modules specified for it under "Optional Subroutine Modules.")

where:

SREGL is the general register which is loaded with the return address to that point in the user's program which uses an exceptional condition indication: General Register 4.

zzzz is the address in the user's program that uses the exceptional condition indication.

If the modules specified for an exceptional condition return address in "Optional Subroutine Modules" are present, and if the exceptional condition indication is not significant, this register should contain the normal return address to the current call. It need not be loaded for other routines. If an exceptional condition occurs and these modules are not present, an error wait will ensue.

BALR SREGR,SREGZ

Branch and Link.

where:

SREGR is the general register which is loaded with the return address to the user's program, making linkage possible: General Register 0.

EXAMPLE OF DIRECT LINKAGE

The following is an example of the coding in the user's program that is assembled with the I/O Support Package. The first set of coding uses symbolic register names; the second set uses the actual register numbers. Both sets assume the following:

1. All required routines are present.
2. The area INFORM is defined in the user's program.
3. The routine beginning at CHKRT notes the occurrence of an exceptional condition and the appropriate modules are present.
4. The user wishes to write on the IBM 1052 Printer-Keyboard.
5. 32 bytes are to be written beginning from INFORM.

Coding with symbolic register names:

```
LA    SREGZ,SWMSW    Load address of the routine to write a message.
LA    SREGA,INFORM   Load address of the first byte of the area to be written from.
LA    SREGN,32       Load number of bytes to be written.
LA    SREGL,CHKRT    Load address of routine in user's program that uses exceptional condition indication.
BALR  SREGR,SREGZ    Branch and Link.
```

Coding with actual register numbers:

```
LA      1,SWMSW
LA      2,INFORM
LA      3,32
LA      4,CHKRT
BALR    0,1
```

INDIRECT LINKAGE

As was pointed out, the preceding coding sequence may be used only when the I/O Support Package is assembled with the user's program. When the I/O Support Package is not assembled with the user's program, he must use a different sequence of coding (this sequence may also be used when the I/O Support Package is assembled with the user's program).

If the user's program and I/O Support Package are not assembled together, the user must employ the call entry tables to produce the entry linkage. The starting address of the Primary Call Entry Table is symbolic name SINTRY; the starting address of the Secondary Call Entry Table is symbolic name SNTRY2. Figure 7 shows the construction of the Primary Call Entry Table and Figure 8 shows the construction Secondary Call Entry Table.

```
SINTRY   DS     0D              Define starting address of table
         DC     A(SRDCW)        Read card and wait
SUTAB    EQU    *               Define first device entry
SCRDR    DC     A(10) = 00A     Card reader address
         DC     A(0)            Area for unit exceptional condition return
                                address
*
         DC     A(SWMSW)        Write message and wait
STYPR    DC     A(9)  = 009     Typewriter address
         DC     A(0)            Area for unit exceptional condition return
                                address
*
         DC     A(SPRTW)        Print a line and wait
SPRTR    DC     A(11) = 00B     Printer address
         DC     A(0)            Area for unit exceptional condition return
                                address
         DC     A(SPUC)         Punch
SPNCH    DC     A(13) = 00D     Punch address
         DC     A(0)            Area for unit exceptional condition return
                                address
*                                 Note: This unit block used only
*                                 for punch whose unit address
*                                 differs from the card reader.
*
         DC     A(SRTPW)        Read tape record and wait
STAP     DC     A(180)          Tape address
         DC     A(0)            Area for unit exceptional condition return
                                address
*
*
SDUMD    DC     A(0)            Dummy entry - termination
         DC     A(61440)        Dummy entry - termination
*
```

Figure 7. Primary Call Entry Table

```
SNTRY2   EQU     *              Define starting address of the table
         DC      A(SPCR)        Punch (reader)
         DC      A(SSNSW)       Sense 6 bytes
         DC      A(SPCMW)       Typewriter single space
         DC      A(SPCPW)       Printer single space
         DC      A(SKIPW)       Printer skip-to-channel 1
         DC      A(SPCRW)       Punch (reader) and wait
         DC      A(SPUCW)       Punch and wait
         DC      A(SWTPW)       Write tape record and wait
         DC      A(SRWD)        Rewind tape
         DC      A(SWTMW)       Write tape mark and wait
         DC      A(SBSRW)       Backspace tape record and wait
         DC      A(SBSF)        Backspace tape file
         DC      A(SFSRW)       Forward space tape record and wait
         DC      A(SFSF)        Forward space tape file
         DC      A(SBRTW)       Read tape record backward and wait
         DC      A(SCTLW)       Issue control command
```

Figure 8. Secondary Call Entry Table

In order to use the entry tables, the user must first define them in his object program. He does this as follows:

```
EXTRN   SINTRY
EXTRN   SNTRY2
```

The next step is to load the address of the desired entry module into a general register. These tables reveal two facts pertinent to loading this address:

1. It takes two instructions to load this address. The address of the table is first loaded into a general register. The second instruction uses this general register to load the address of the desired entry module.

2. Each of the locations in the tables that contain the address of an entry module is displaced from the starting address of the table by a certain number of bytes. Therefore, to load the address of any entry module from the entry tables, the coding sequence must reflect the displacement of that location in the entry table which contains the address of the desired entry module. This displacement may be defined by the use of Equate (EQU) instructions in the user's program. Figure 9 shows the exact displacement for all of the entry modules. The reader should note that he may use any symbolic name for the entry modules in the EQU instructions, as long as he does not use their actual symbolic name; that is, he may not use SRDCW, SPCPW, etc., as a symbolic name, (if he did, there would be duplicate symbols).

1. Displacement of Primary Entry Module Addresses from SINTRY

Entry Address	Operation	Bytes from SINTRY
QRDCW	EQU	0
QWMSW	EQU	12
QPRTW	EQU	24
QPUC	EQU	36
QRTPW	EQU	48

2. Displacement of Secondary Entry Module Address from SNTRY2

Entry Address	Operation	Bytes from SNTRY2
QPCR	EQU	0
QSNSW	EQU	4
QPCMW	EQU	8
QPCPW	EQU	12
QKIPW	EQU	16
QPCRW	EQU	20
QPUCW	EQU	24
QWTPW	EQU	28
QRWD	EQU	32
QWTMW	EQU	36
QBSRW	EQU	40
QBSF	EQU	44
QFSRW	EQU	48
QFSF	EQU	52
QBRTW	EQU	56
QCTLW	EQU	60

Figure 9. Displacement in Entry Tables

Thus, what the user must effectively do is add the displacement to the address of SINTRY or SNTRY2. Once the user has established the addresses of SINTRY and SNTRY2 by:

```
PITBAD   DC     A(SINTRY)
SETBAD   DC     A(SNTRY2)
```

and the displacement from these addresses of that location in the table that contains the address of the desired entry module (for example, the routine to print a line), by:

```
QPRTW    EQU    24
```

he can then load the address of this routine by the following two instructions:

```
L  SREGZ,PITBAD          Load the address of
                         SINTRY
L  SREGZ,QPRTW(SREGZ)    Load the contents
                         of the location
                         SINTRY+24, in this
                         case the address of
                         the SPRTW entry
                         module
```

These two instructions replace and serve the same purpose as:

```
LA       SREGZ,xxxx
```

which was the first instruction in the coding sequence noted in "Direct Linkage." All the other instructions in that sequence, that is:

```
LA       SREGA,yyyy
LA       SREGN,n
LA       SREGL,zzzz
BALR     SREGR,SREGZ
```

remain the same, if the register assignments are equated as in the I/O Support Package, and all specifications which were described there also apply when they are used as part of the linkage format for a user's program that was assembled separately from the I/O Support Package.

Figure 10 is an example of the linkage format for a user's program that was assembled separately from the I/O Support Package. The following assumptions are made in this example:

1. The only entry modules desired are SPCRW and SWMSW.

2. Symbolic register names are used. Note: The user may employ the actual register numbers, if it is so desired.

SENSE ENTRY EXAMPLE

This section provides a coding example of the SSNSW entry. The following assumptions are made in this example:

1. All required modules are present.

2. Information is to be sensed from the printer.

3. The I/O Package was assembled separately from the user's program.

4. The user will load the unit reference into register SLUBRG.

5. The SSNSW entry will transmit the sensed data to the area defined by the I/O Support Package - beginning at symbolic location SNSA, which must be defined by an EXTRN in the user's program. SNSA is defined as an ENTRY in the I/O Support Package.

Figure 11 illustrates the coding in the user's program.

If the user did not want to load the unit reference into register SLUBRG, he would do the following:

1. Include the Locate SINTRY Table Unit Block and Sense Entry Locate SINTRY Table Block Exit modules.

2. Place the address of the device in the high-order 16 bits of register SREGN; this may not be a new device address.

```
            EXTRN   SINTRY              Define Primary Call Entry Table
            EXTRN   SNTRY2              Define Secondary Call Entry Table
              .
              .
              .
            L       SREGZ,PITBAD        Load the address of SINTRY
            L       SREGZ,QWMSW(SREGZ)  Load the contents of the Location SINTRY+12,
                                          in this case the address of SWMSW
            LA      SREGA,yyyy          Load address of first byte to be processed
            LA      SREGN,n             Load the number of bytes to be processed
            LA      SREGL,zzzz          Load exceptional condition return address
            BALR    SREGR,SREGZ         Branch and Link
              .
              .
              .
            L       SREGZ,SETBAD        Load the address of SNTRY2
            L       SREGZ,QPCRW(SREGZ)  Load the contents of the location SNTRY2+20,
                                          in this case, the address of SPCRW
            LA      SREGA,yyyy          Load address of first byte to be processed
            LA      SREGN,n             Load the number of bytes to be processed
            BALR    SREGR,SREGZ         Branch and Link
              .
              .
              .
  QWMSW     EQU     12                  Specify the displacement (as shown in
                                          Figure 9) of the location in SINTRY which
                                          contains the address of the desired entry
                                          module, in this case, SWMSW
  QPCRW     EQU     20                  Specify the displacement (as shown in
                                          Figure 9) of the location in SNTRY2 which
                                          contains the address of the desired entry
                                          module, in this case, SPCRW
  SREGZ     EQU     1                   Equate SREGZ to general register 1
  SREGA     EQU     2                   Equate SREGA to general register 2
  SREGN     EQU     3                   Equate SREGN to general register 3
  SREGL     EQU     4                   Equate SREGL to general register 4
  SREGR     EQU     0                   Equate SREGR to general register 0
              .
              .
              .
  PITBAD    DC      A(SINTRY)           Define the address of SINTRY
  SETBAD    DC      A(SNTRY2)           Define the address of SNTRY2
```

Note: If the EXTRN instructions are removed, this coding would also serve when the object program and I/O Support Package are assembled together.

Figure 10. Example of Indirect Linkage

```
          EXTRN    SINTRY        Define primary call table
          EXTRN    SNTRY2        Define secondary call table
          EXTRN    SNSA          Define sense area
          .
          .
          .
          L        1,SETBAD      Load address of SNTRY2
          L        1,(4,1)       Load address of Sense Entry
          L        6,PITBAD      Load address of SINTRY
          LA       6,28(6)       Load address of the unit address cell, in this case,
                                 that of the printer
          BALR     0,1           Branch and Link to I/O
          .
          .
          .
PITBAD    DC       A(SINTRY)     Define the address of SINTRY
SETBAD    DC       A(SNTRY2)     Define the address of SNTRY2
```

Figure 11. Sense Entry Coding Example

CONTROL ENTRY EXAMPLE

This section provides a coding example of the SCTLW entry. The following assumptions are made in this example:

1. All required modules are present.

   Note: The SCTLW entry requires the presence of the Command Operation Modifiers routine.

2. The user wants to provide for an immediate space of 3 on the printer; the modifier bit pattern for this is 00011.

3. The user will load the unit reference into register SLUBRG (register 6).

4. The I/O Support Package was assembled with the user's program.

Figure 12 illustrates the coding in the user's program.

If the user did not want to load the unit reference into register SLUBRG, he would do the following:

1. Include the Locate SINTRY Table Unit Block and Control Entry Locate SINTRY Table Block Exit modules.

```
          LA       SREGZ,SCTLW   Load the address of the SCTLW entry
          LA       SLUBRG,SPRTR  Load the address in SINTRY of the unit address
                                 cell, in this case, the printer
          LA       SREGA,MOBITS  Load the modifier bit pattern into the high-order
                                 five bits of register SREGA
          LA       SREGN,1       Load a number to ensure that an invalid byte
                                 count of zero is not in register SREGN
          LA       SREGL,EXCPAD  Load the address of the user's routine that
                                 handles an exceptional condition indication
          BALR     SREGR,SREGZ   Branch and Link
          .
          .
          .
          DC       0F            Align on full-word boundary
MOBITS    DC       X'18'         Define the modifier bit pattern; this particular
                                 pattern has the bit configuration 00011000;
                                 it is placed in the high-order byte of register
                                 SREGA and the high-order five bits are interpreted
                                 as the modifier bit pattern. For other modifier
                                 bit patterns, the user is referred to the reference
                                 manual of the particular I/O device
          DC       AL3(0)        The address portion is not significant
```

Figure 12. Control Entry Coding Example

2. Place the address of the device in the high-order 16 bits of register SREGN; this may not be a new device address.

CARD-ONLY INSTALLATIONS

An assembled version of the I/O Support Package is provided for card-only installations. It includes the following entry routines:

SRDCW
SWMSW
SPRTW
SPUC and SPUCW
SPCR and SPCRW
SSNSW
SKIPW

All requirements specified in "Input/Output Support Package" apply to card-only installations with the following limitations:

1. Only the modules required for card I/O routines will be included.

2. The only option provided is for an exceptional condition return address. The modules required for this option will be included.

3. Modules required for SEREP interface are included.

This version supports the following I/O devices:

- One IBM 2540 and/or 1442 Card Reader-Punch
- One IBM 1052 Printer-Keyboard
- One IBM 1403 or 1443 Printer

CARD-TAPE PACKAGE

Another assembled version of the I/O Support Package is the card-tape package. It includes the following entry routines:

SRDCW
SWMSW
SPRTW
SPUC and SPUCW
SPCR and SPCRW
SSNSW
SKIPW
STPBKW
SRTPW
SWTPW
SRWD
SWTMW
SBSRW
SBSF
SFSRW
SCTLW

This version supports the following I/O devices:

1. One IBM 2540 and/or 1442 Card Reader-Punch.

2. One IBM 1052 Printer-Keyboard.

3. One IBM 1403 or 1443 Printer.

4. Any number of IBM 2400 Series Magnetic Tape Units.

All the requirements specified in the "Input/Output Support Package" apply to this card-tape package, with the following limitations:

1. Only the modules required to support these entries will be included.

2. There are four optional facilities supplied with this version:

   - Exceptional Condition Routines
   - Tape Retry on Error Routines
   - Multiple Unit Address-Device Routine
   - Command Operation Modifiers Routine

3. Modules required for SEREP interface and tape error recovery are included.

CHARTS OF MODULE RELATIONSHIPS

These charts are intended to give the reader a view of the dependencies among the modules of the I/O Support Package. The general approach to these charts is as follows: all the modules outside the required group may be used independently, but the user must follow the flow lines from the module he selects back to the required modules and include every module

the flow line intersects. More specifically, when the user selects any module, he should follow the arrow from that module, taking all branches, and incorporate in his deck all the modules encountered.

The reader should understand the following criteria for using these charts:

1. The name (as it appears on the listing of the I/O Support Package supplied by IBM) of each module is contained in process blocks. Also in the process block is the symbolic starting address of the module; for example, Issue Internal Call (listing name of the routine), SNTCL (symbolic starting address of that routine).

2. The listing group name is also contained on the charts; the reader will find these group names in the verbal discussions of the I/O Support Package.

3. Above each block containing the name of a module, there are a series of codes designed to aid the user when selecting modules from the I/O Support Package. The fields of the code are separated by commas. The following is an explanation of these codes:

   a. The first two digits are the identifying number of the module; these digits are found in columns 76 and 77 of the symbolic decks; for example, above the block that contains the name SINTRY, the first two digits are: 10. These digits -- 10 -- appear in every card of the SINTRY module (in the symbolic deck) in columns 76 and 77.

   b. The second field shows the number of bytes (these are the initial release figures and are subject to change) the particular module occupies; for example, above the block that contains the name of the Attention Base Routine, the following appears in the first two fields: 3J,12,...; where 3J is the identifying number and 12 (decimal) indicates that this module occupies 12 bytes.

   c. After the field that indicates the bytes occupied by the module, there are a series of codes that indicate which modules are used by the basic utility programs (this is intended as an aid to the user who desires to select his modules from the I/O Support Package and make his own resident I/O). Table 23 defines these codes.

Table 23. Chart Codes for Basic Utility Programs

Code	Significance
ALL	This module is used by all the basic utility programs.
C	This module is provided with the Card-Only version of the I/O Support Package.
D1	This module is used by Phase 1 of the Two-Phase Dump Program.
D2	This module is used by Phase 2 of the Two-Phase Dump Program.
DS	This module is used by the single phase Dump Program.
DT	This module is used by both phases of the Two-Phase Dump Program.
L	This module is used by the Absolute and Relocating Loaders.
T	This module is provided with the Card-Tape version of the I/O Support Package.

For example, the codes above the block that contains the name of the SKIPW entry module (J3,36,T,C,D2,DS) are interpreted as follows:

J3 Identifying number; this number appears in all cards of the symbolic version of the SKIPW module in columns 76 and 77.

36 This module occupies 36 bytes in storage.

T This module is provided with the Card-Tape version of the I/O Support Package.

C This module is provided with the Card-Only version of the I/O Support Package.

D2 This module is used by Phase two of the Two-Phase Dump Program.

DS This module is used by the single phase Dump Program.

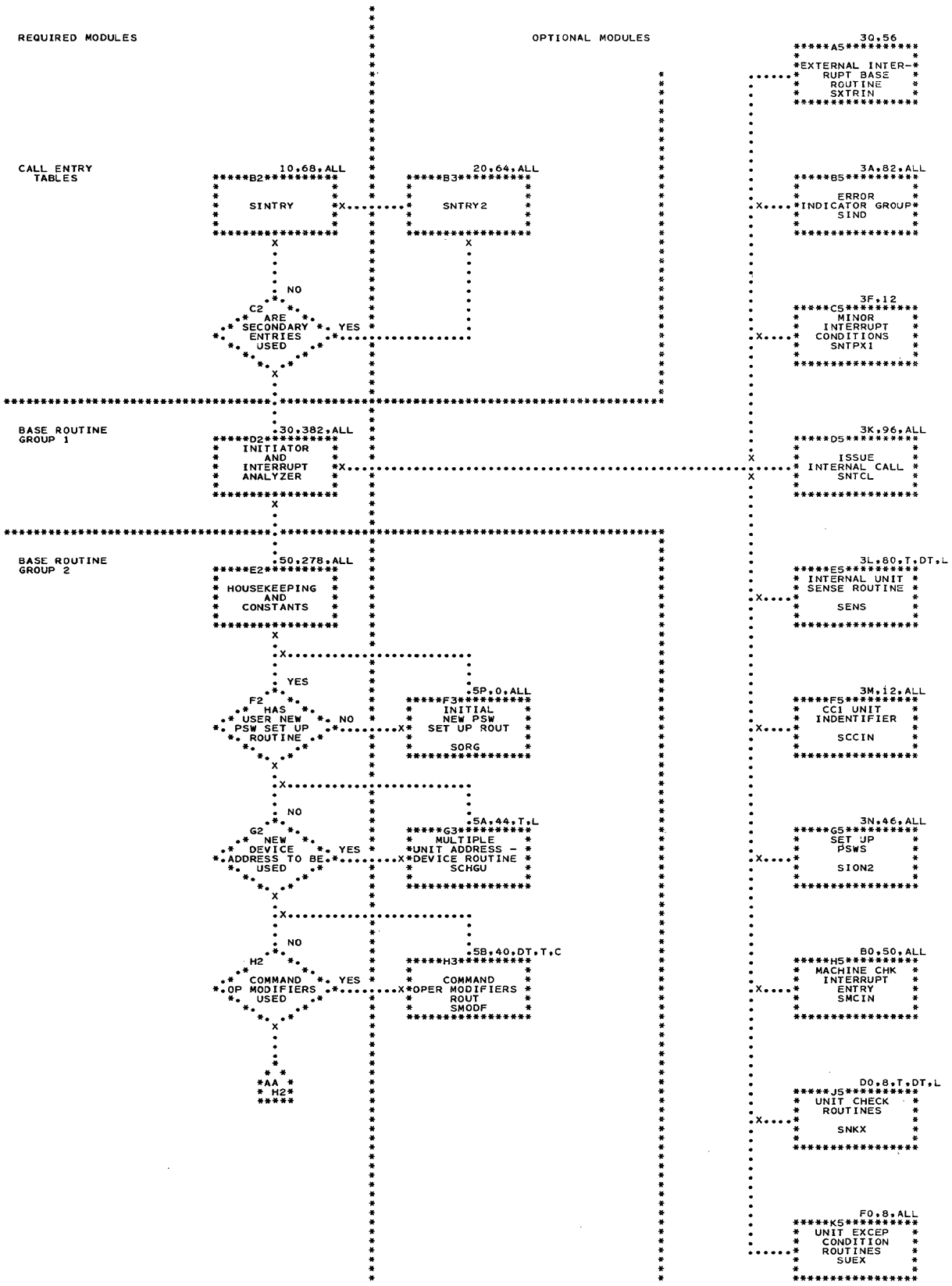


Chart AA. Required Modules and Interrupt Action Modules

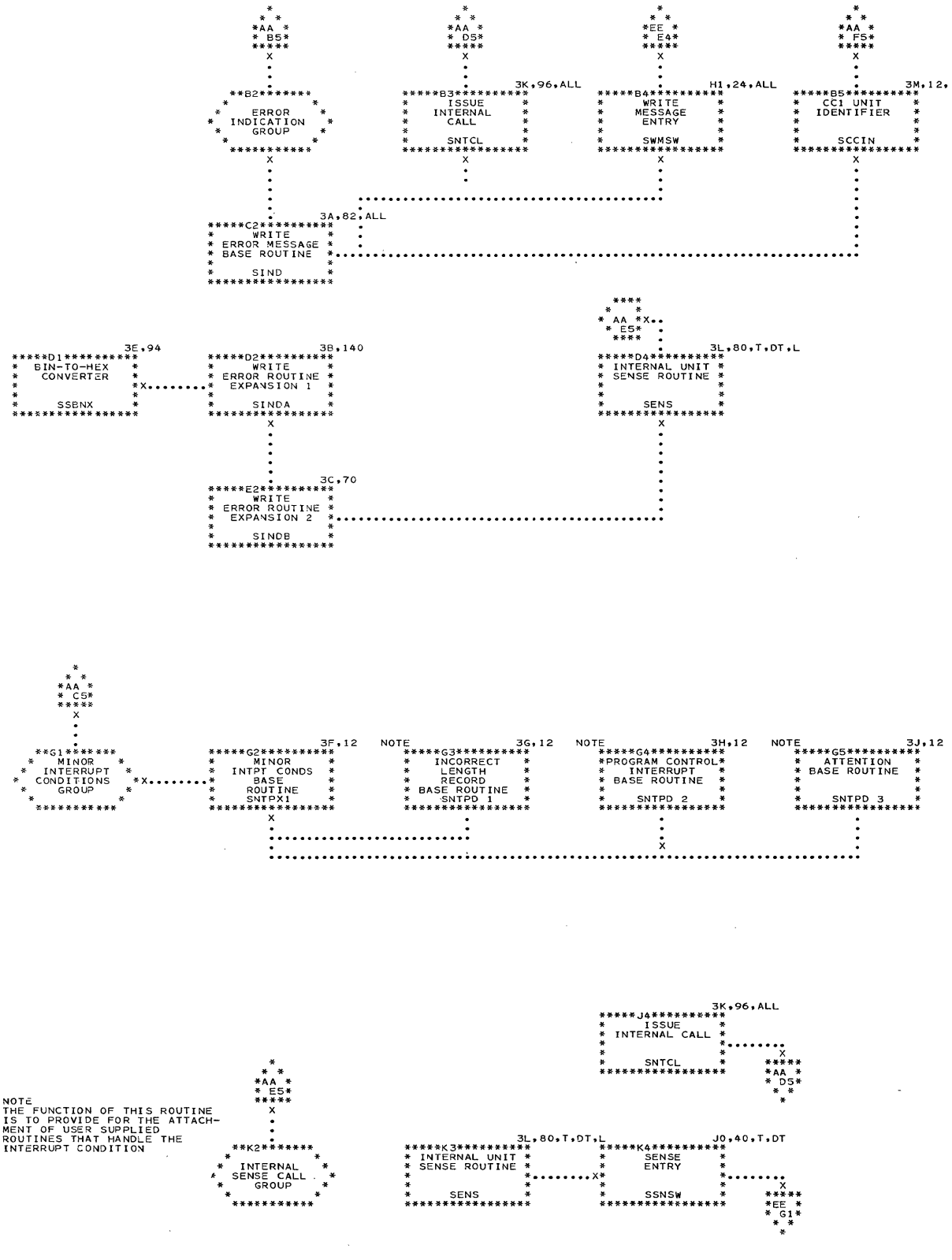


Chart BB. I/O Base Routine - Group 1 Optional Modules

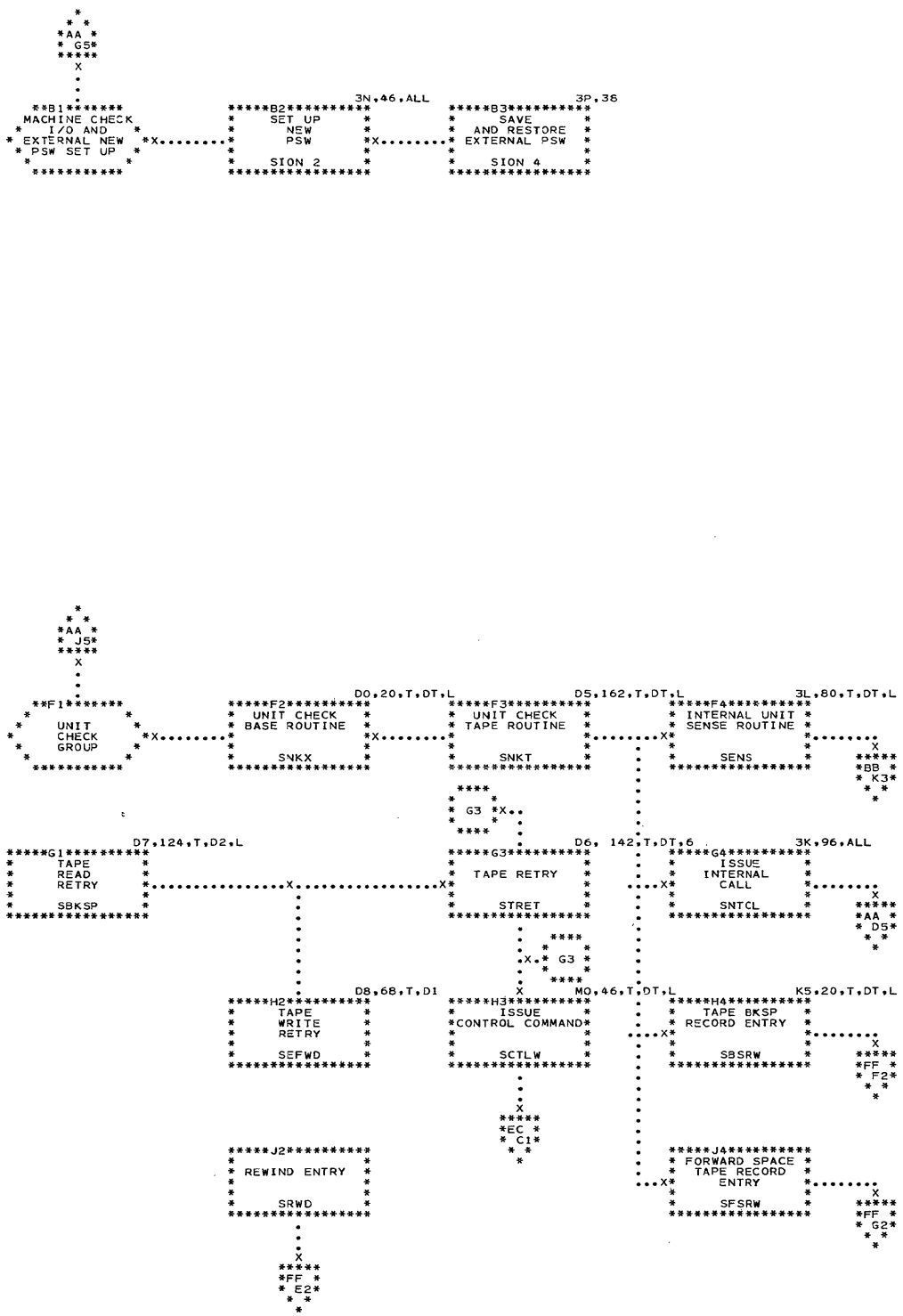


Chart CC. I/O Base Routine - Group 1 PSW Routines, Machine Check Group, Unit Check Group

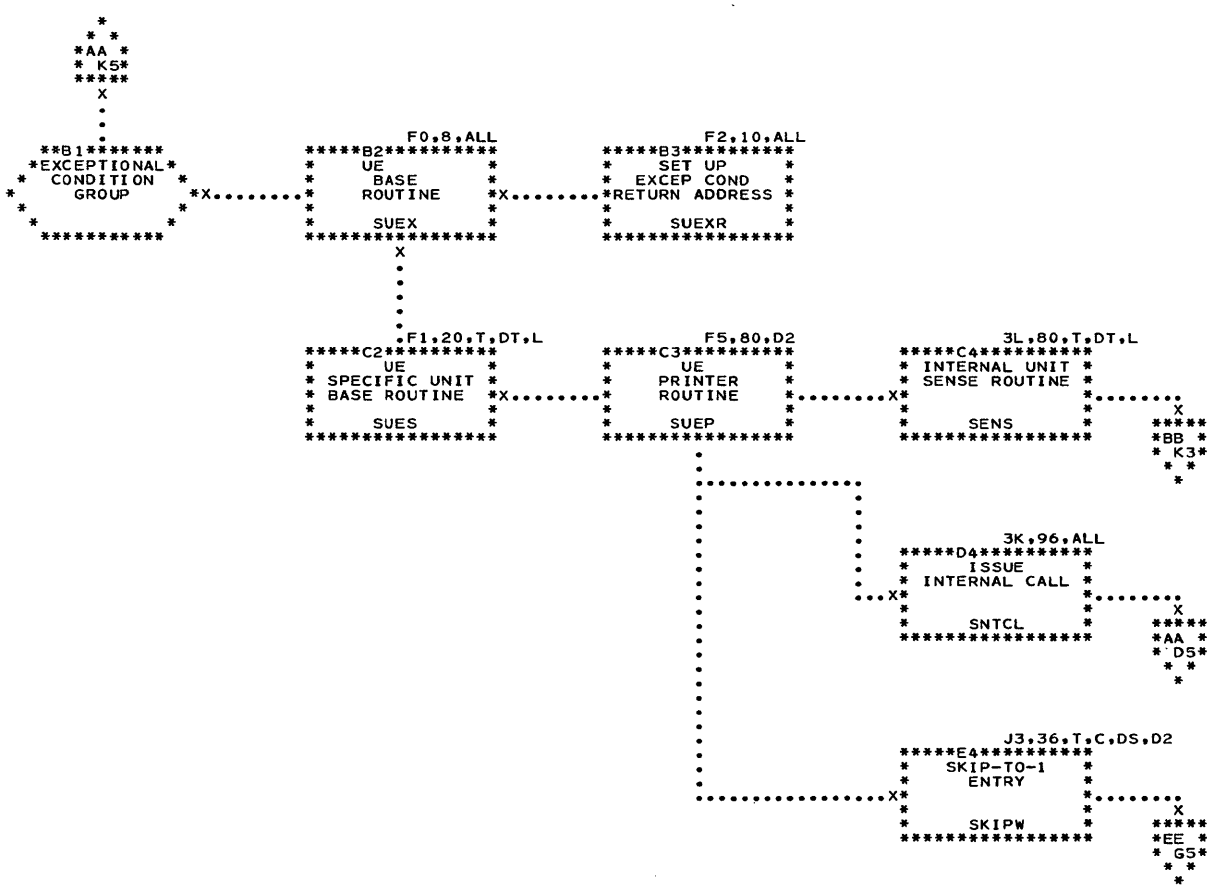


Chart DD. Unit Exceptional Condition Group

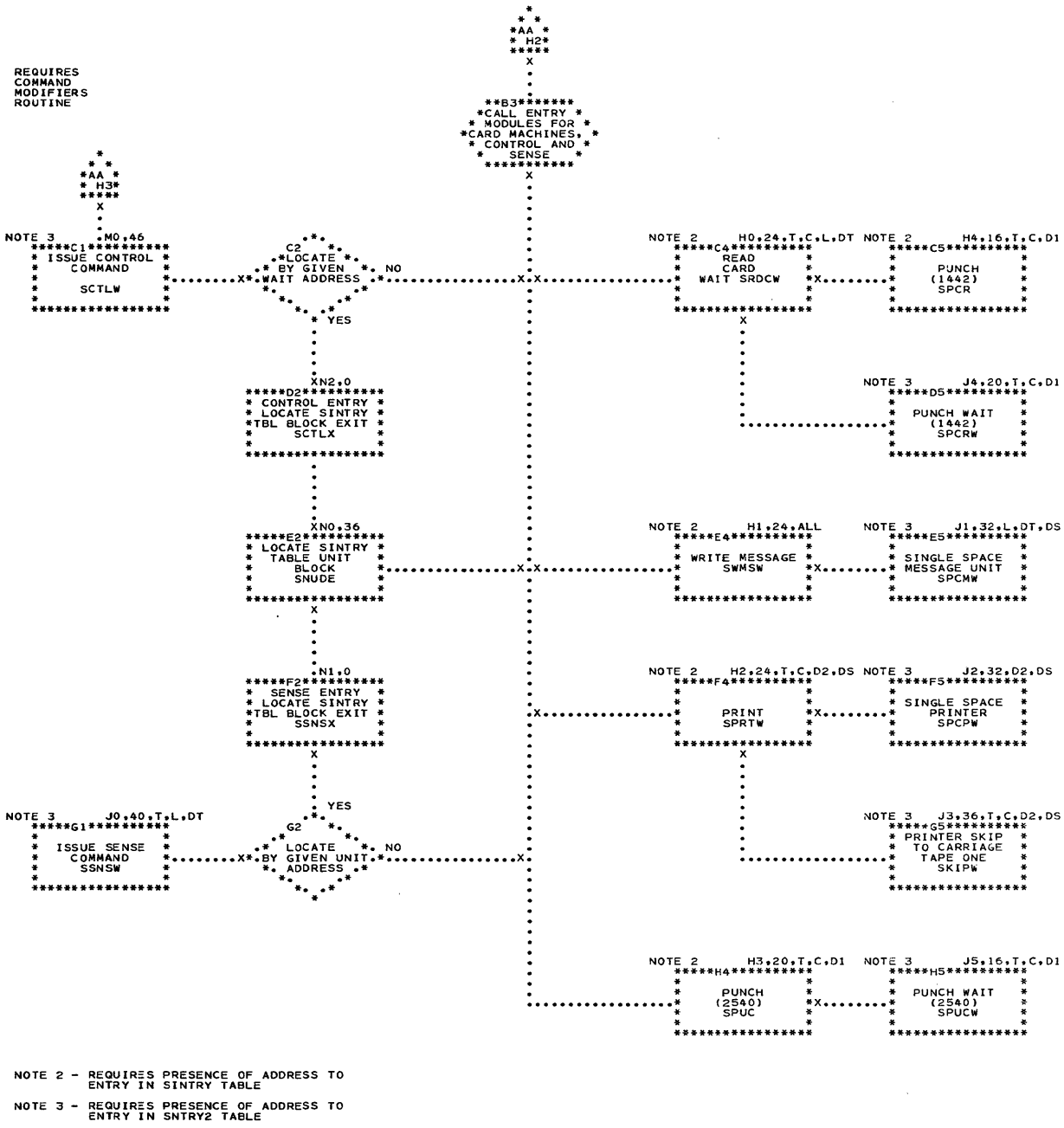
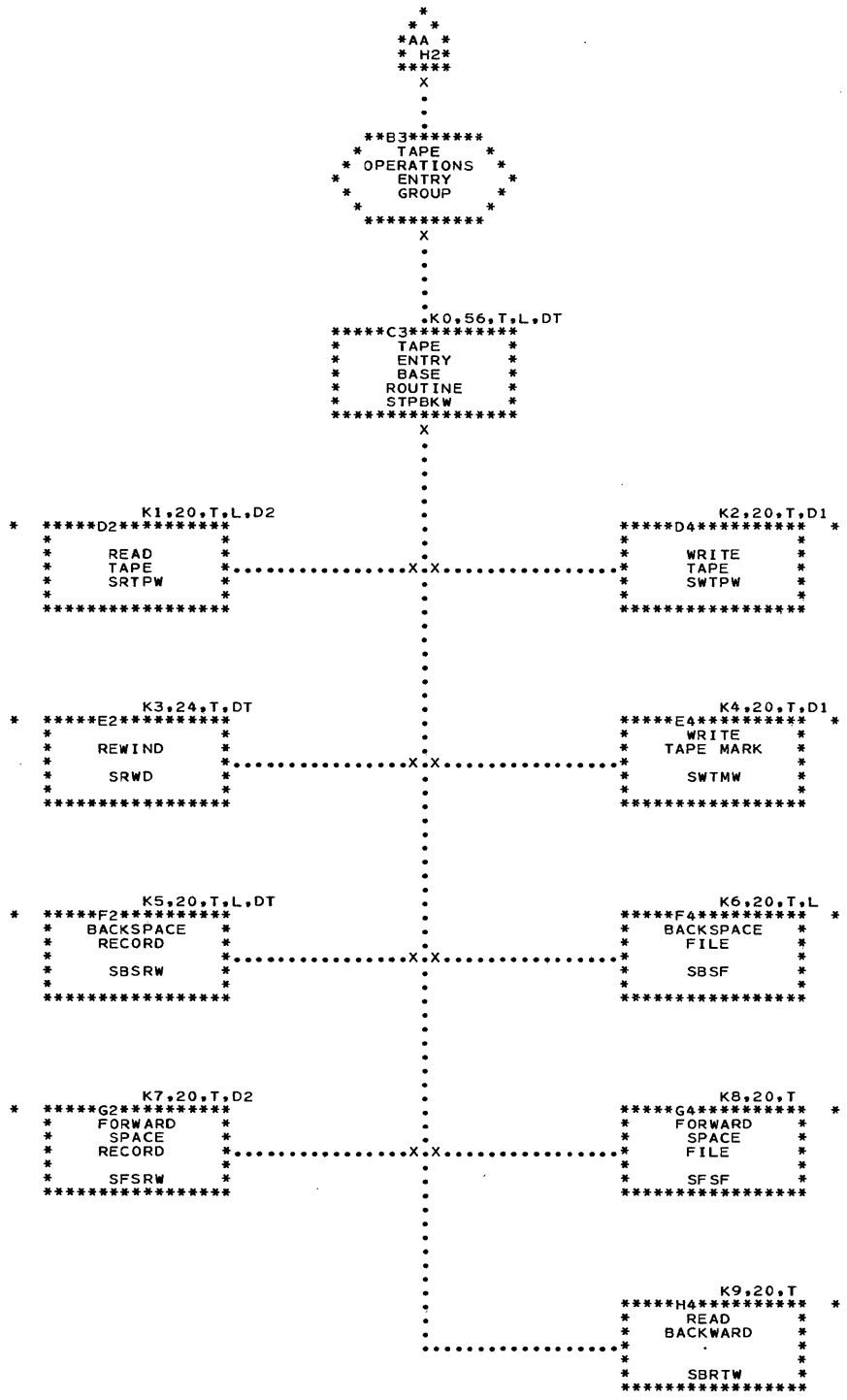


Chart EE. I/O Call Entry Group Modules for Non-Tape, Sense, and Control Operations



* REQUIRES PRESENCE OF ADDRESS TO ENTRY IN SNTRY 2 TABLE

Chart FF. I/O Call Entry Group Modules for Tape Operations

The programmer often finds it necessary to use subroutines and other program segments that he himself did not produce. In most cases, the programmer knows the calling sequence of these routines; however, the assembled location or the size of these routines usually is not known. In using the relocating loader, the question of size may or may not be of concern to the programmer (depending on the storage capacity of his machine) and the question of assembled addresses is of no concern, since the loader will load and set up linkage between these various routines.

When relocating program segments and establishing linkage among them, the relocating loader must calculate certain information during the loading process.

The loader receives the information to answer these questions from the load cards that it encounters during loading. Some of the information that the loader receives must be saved for later use during the loading process. The information that is saved is placed in the Control Dictionary, which is composed of two tables, one called the Reference Table and the other the External Symbol Identification Table.

The External Symbol Identification Table is contained in the loader itself. The Reference Table is built downward from the highest available storage address (location 8191 in the low version released by IBM), each entry (a maximum of 253 entries) consisting of 12 bytes. The Reference Table is protected from being overlaid when input to the loader is in relocatable form. However, during an absolute load, the Reference Table is not protected and may be overlaid.

The information required by the loader answers the following questions:

1. What are the names (program name, entry points, and external symbols) by which this segment may communicate with other program segments, and what are the actual addresses of these names? A program segment (or subroutine) may be referenced by other program segments: if the segment which is referenced is in storage at load time, the address of the segment is already established; if it is not in storage at load time, the name and entry points must be defined to the loader by an ICS card (and SLC card, if necessary). (These assigned addresses are kept by the loader in the Reference Table.)

2. What address constants within the assembled segment would change value as a result of this segment or another segment being relocated? During the loading process, the loader is notified that adjustments are to be made within this program segment by the ESD cards (types 0 and 2). It is told how and where these adjustments are to be made by the RLD cards.

3. What is the relocation factor; that is, what is the difference between the assembled address of the segment and the address where loading will begin? This factor must be added to or subtracted from the assembled address of the program name and any other entry point to the segment, and the assembled address in all Text and Replace cards.

Example

In order to illustrate, step by step, how the loader accomplishes relocation and linkage, we will assume that there are two program segments to be loaded, SEGA and SEGB.

SEGA refers to two subroutines in SEGB called SQRT and LINK. SEGA defines SQRT and LINK as external symbols by these assembly instructions:

```
SEGA    START  128
        EXTRN  SQRT
        EXTRN  LINK
```

During execution, SEGA can branch to these external subroutines, thus:

```
        L      15, ADSQRT
        BALR   14,15
               .
        L      15,ADLINK
        BALR   14,15
```

Address constants are generated for them in this manner:

```
ADSQRT  DC     A(SQRT)
ADLINK  DC     A(LINK)
```

SEGB refers to SEGA by its program name, which is an entry point.

SEGB must define SEGA as an external symbol:

```
        EXTRN  SEGA
```

and generate an address constant:

```
ADSEGA  DC     A(SEGA)
```

to allow a branch and link operation.

Note that SEGA does not yet have the actual addresses it needs of SEGB, nor does SEGB have the address of SEGA. These addresses will not be assigned until load time. The ESD and RLD cards produced by the assembler for each segment provide the information the loader needs to complete linkage.

To illustrate the use of the relocation factor (see point 2 on the preceding page), the example in Figure 13 assumes that SEGA was assembled at storage location 500 and has a length of 200 bytes; that SEGB was assembled at storage location 400 and has a length of 100 bytes; finally, it assumes that the programmer desires to load the segments beginning at location 1000. Note carefully that this procedure requires a Set Location Counter card to set the initial loading location to 1000. Also note that since SEGB refers to SEGA by name, an Include Segment card is also necessary to establish the location and length of SEGA before it is loaded.

Figure 13 illustrates the loading process. It shows how each card is generated from the user's source deck, through assembler operations, to assembler output and onto load time. Finally, the figure illustrates the appearance of storage after loading. The five columns of Figure 13 are read left to right following the flow noted in the previous two sentences.

Each card is referred to by its three-letter mnemonic: SLC, ICS, ESD, and so forth.

Other abbreviations used in Figure 13 are:

ESID	for	External Symbol Identification
LOCCT	for	Location Counter
REFTBL	for	Reference Table
ESIDTBL	for	External Symbol Identification Table

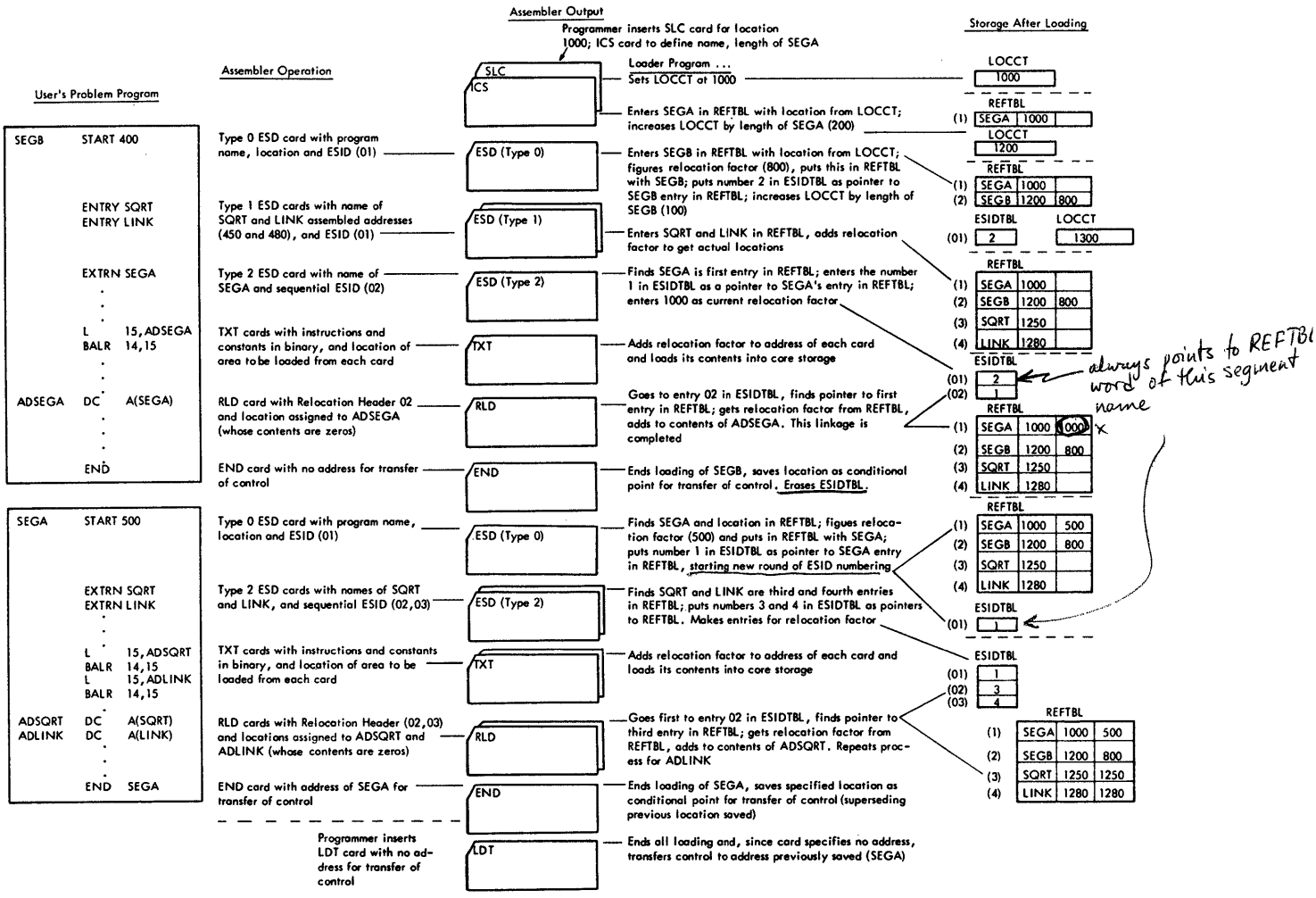


Figure 13. Example of the Loading Process

APPENDIX B: SELF-LOADING PROGRAM GENERATOR (LDRGEN)

LDRGEN is a program designed to regenerate loader program decks into a form suitable for direct loading into storage. Furthermore, since neither the absolute nor relocating loader is provided in a form that can be relocated, LDRGEN can be used by an installation to cause the loaders to occupy locations in storage other than the locations they occupy in the versions released by IBM.

REQUIREMENTS FOR USING LDRGEN

LDRGEN is provided only in symbolic form and therefore must be assembled by the user. Similarly, the absolute and relocating loaders must be assembled at the locations desired by the user. Prior to assembly of the LDRGEN program, the user must provide LDRGEN with the address of the output device: he does this by means of an Equate instruction that he inserts into the LDRGEN deck immediately before the END card. It is coded as:

```
OUTPUT   EQU    (address of the output
                 device in hexadecimal
                 or its equivalent
                 decimal notation)
```

The assembled loader deck and LDRGEN programs can be loaded into storage by the absolute or the relocating loader.

CAUTION: the versions of the loaders released by IBM occupy low- or high-storage locations on an 8K configuration. Since it is necessary to load the assembled relocatable decks of both LDRGEN and the loader being regenerated, care must be taken to ensure that neither of these will overlay the loader loading them. In other words, all must fit in the storage of the machine, remembering that the self-loading loader occupies predetermined locations and the loader being generated must occupy the locations where its residence is desired.

The loader program must declare the following information to LDRGEN:

1. The lowest storage address occupied by the loader; this address shall be called ALPHA.

2. The loader initial entry point; this address shall be called BETA.

3. The availability of an area of at least 160 bytes for the temporary residence of the bootstrap routine; this area shall be called IOTA. (The address IOTA must be on a double-word boundary.) IOTA should not be included within the loader (that is, between ALPHA and OMEGA); it should be adjacent to the loader. This may be coded as:

   ```
   IOTA     EQU    *-160
   ```

   to reside below the loader or as:

   ```
   IOTA     EQU    *
   ```

   to reside above the loader.

4. The highest storage address plus 1 occupied by the loader; this address shall be called OMEGA.

PROVIDING ADDRESSES

As was pointed out, LDRGEN is loadable by either the absolute or the relocating loader. Both loaders define ALPHA, BETA, IOTA, and OMEGA by ENTRY assembler instructions; therefore, these addresses are supplied to LDRGEN in one of two ways, depending on whether the absolute or the relocating loader was used to load LDRGEN.

If the absolute loader was used, the addresses are supplied to LDRGEN by Replace cards:

Assigned to	Into LDRGEN at Location
ALPHA	ALPHAA
ALPHA	ALPHAB
BETA	BETAA
IOTA	IOTAA
OMEGA	OMEGAA

If the relocating loader was used to load LDRGEN, the linkage is supplied through ENTRY assembler instructions. LDRGEN defines these addresses through EXTRN assembler instructions. IOTA should be designated by the loader program as a buffer area. This area is temporarily occupied by the bootstrap routine, but it is available to the object program at execution time.

Finally, LDRGEN provides the facility of producing duplicate decks; there is a halfword in LDRGEN called CON. This location is originally assembled with a value of one. However, if the user desires more than one copy of his deck, he may change the value in CON, by a Replace card, to any desired value. The value in CON will be decremented by one after each copy of the deck is made and will continue to make copies of the deck until the value in CON is reduced to zero.

SEQUENCE OF OPERATIONS

The following is the sequence of operations of LDRGEN.

1. It calculates the difference between ALPHA and OMEGA; this gives LDRGEN the size of the object program it will write.

2. It adjusts the bootstrap (160 bytes) address to the designated area -- IOTA.

3. It issues a write command for:

   a. One 24-byte card containing the IPL record (Initial Program Loading PSW, Initial Program Loading CCW1, Initial Program Loading CCW2),

   b. Two 80-byte cards containing the bootstrap routine,

   c. A series of 80-byte records, of which the first 72 bytes are text, containing the loader program in a form suitable for direct loading into storage (that is, the contents of ALPHA through ALPHA+71, ALPHA+72 through ALPHA+143, etc.). These cards will be sequenced in columns 77-80.

4. After the entire program has been regenerated, it writes an END card using the address of BETA as the initial entry point to the loader.

5. It examines the count to see if duplicate decks are to be written. If there are duplicate decks to be made, the sequence of operations begins again at item 3.

INDEX

READER'S COMMENTS

IBM System/360 Basic Programming Support

Title: Basic Utilities

360P- UT-017, -018, -019, -020

Form: C28-6505-2

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?

___ As an introduction to the subject ___ For additional knowledge

Other _____

fold

Please check the items that describe your position:

___ Customer personnel ___ Operator ___ Sales Representative

___ IBM personnel ___ Programmer ___ Systems Engineer

___ Manager ___ Customer Engineer ___ Trainee

___ Systems Analyst ___ Instructor Other _____

Please check specific criticism(s), give page number(s), and explain below:

___ Clarification on page(s)

___ Addition on page(s)

___ Deletion on page(s)

___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL

No Postage Necessary if Mailed in U.S.A.

staple

fold

fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

old

fold

IBM

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