File Number S360-20
Form C28-6503-6

**BPS**

**IBM** **Systems Reference Library**

# IBM System/360
# Basic Programming Support
# Basic Assembler and Basic Utility Programs (Card)
# Specifications and Operating Guide

This reference publication is arranged in six major sections to describe these programs:

| Name | Program Number |
|---|---|
| Basic Assembler | 360P-AS-021 |
| Absolute Loader | 360P-UT-017 |
| Input/Output Support Package | 360P-UT-018 |
| Dump Program | 360P-UT-019 |
| Relocating Loader | 360P-UT-020 |

The first section provides a description of the Basic Assembler language and the Basic Assembler program. Features concerned with the planning and writing of source programs are emphasized. The functions and possible modifications of each of the basic utility programs are described in the next major section. Also included is a discussion of program segment relocation and linkage. The input to and output from the Basic Assembler program and procedures for running assembly jobs are described in the third major section. The operating procedures for the utility programs are presented in the fourth major section. Program waits and operator messages appear in the fifth major section, followed by a sample problem in the last major section.

The reader should be familiar with the material in the IBM System/360 Principles of Operation, Form A22-6821.

The titles and abstracts of related publications are listed in the IBM System/360 Bibliography, Form A22-6822.

Some functions described in this manual require the use of an absolute address. Users of these programs can obtain the appropriate absolute address by referring to the writeup, supplied with the Program Material List, entitled "Attachment 1 - Special Information."

# PREFACE

Basic Assembler Language is a symbolic programming language for the IBM System/360. Basic Assembler Program translates source programs (symbolic language) into machine-language programs. The first section of this manual contains all information required for writing IBM System/360 programs. This includes the rules for writing source statements, a description of assembler instructions, and a list of machine instructions represented in the language.

Basic utility programs (described in the second section) load assembled programs into main storage, provide listings of the contents of storage, and provide routines for accessing input/output devices. The relocating loader relocates other programmers' subroutines, and establishes linkage among them. The Loader Generator Program (LDRGEN) regenerates loader program decks into a form suitable for direct loading into storage.

Operating information and techniques for the Basic Assembler appear in the third section. The Assembler has two phases. Phase 1 partially processes source programs, which are read from punched cards or magnetic tape. Phase 2 completes the processing to produce object programs in punched cards or on magnetic tape.

Operating information and techniques for the basic utility programs are provided in the fourth section. The Single-Phase Dump program produces a listing of the contents of the registers and/or storage areas defined by the user's program. The Two-Phase Dump program produces card or tape records (Phase 1) and listings (Phase 2) of the contents of the registers and/or storage areas defined by the user's program. The Absolute and Relocating Loaders load assembled programs (from cards or tape) into storage for execution.

A program wait (fifth section) occurs whenever the Basic Assembler or basic utility programs must communicate with the operator. A program wait is indicated by the wait light on the system control panel. The coded message can be displayed on the system control panel or can be printed on the output device. The message indicates the program being executed when the wait occurred, the reason for the wait, and the operator action required.

A Card Assembler and Utilities Sample Problem is provided (sixth section) to test the Basic Assembler and Basic Utility Programs (Card) supplied by IBM to the user.

The I/O subroutines are supplied by IBM in symbolic deck form. The other utility programs and the Assembler Program are supplied in assembled deck form but can also be obtained in symbolic form as optional material. The LDRGEN is available only in symbolic form as optional material. This is indicated in the corresponding sections of the manual.

Readers should be familiar with the IBM System/360 and have an understanding of the storage-addressing scheme, data formats, and machine-instruction formats and functions. This information can be found in the publication IBM System/360 Principles of Operation, Form A22-6821.

Seventh Edition, August 1967

This edition, Form C28-6503-6, is a major revision of, and obsoletes C28-6503-5 and Technical Newsletters N24-5174 and N24-5210. This manual also incorporates the information from and obsoletes these publications:

IBM System/360 Basic Programming Support, Basic Utility Programs Specifications, C28-6505-3 and Technical Newsletters N24-5135 and N24-5183

IBM System/360 Basic Programming Support, Basic Assembler and Basic Utility Programs (Card) Operating Guide, C28-6557-3 and Technical Newsletter N24-5198.

This edition contains support for an intermediate storage size (24K) for System/360 Model 30. Notations have also been made to indicate whether programs are available in symbolic or assembled form and whether they are optional material. Changes are indicated by a vertical line to the left of affected text.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Endicott, New York 13760.

The Basic Assembler language is a symbolic programming language for use with the IBM System/360. This language provides programmers with a convenient means of writing machine instructions, designating registers and input/output devices, and specifying the format and addresses of storage areas, data, and constants. All operational capabilities of the IBM System/360 can be expressed in Basic Assembler language programs.

The language features are designed to simplify writing programs for the IBM System/360 by avoiding unnecessary complexity. This reduces program errors and, consequently, the time required to produce a program that is suitable for execution. The language is therefore easier to learn.

Basic Assembler source programs are translated into IBM System/360 machine language object programs by the Basic Assembler (that is, the assembler). In the process of translating programs, the assembler performs certain auxiliary functions, some automatically, others requested by special assembler instructions the programmer writes in his source program.

The assembler is a two-phase program available as non-relocatable assembled self-loading card decks. It is available as optional material in symbolic form for both phases. The assembler has a special operating procedure for use with the IBM 1442-N1 or 2520-B1 Card Read-Punch. During the first phase, the assembler punches information into the source-program deck. Using this information in the second phase, the assembler produces an object program. For systems with tape, a 2540 Card Read-Punch, or a 2501 Card Reader with a 2520 Model B2 or B3 Card Punch, this intermediate information is stored in a tape or card file, rather than the source-program deck. The temporary file then serves as input for the second phase.

## FEATURES OF THE IBM SYSTEM/360 BASIC ASSEMBLER

The most significant features provided by the assembler and its language are summarized in the following paragraphs. This summary does not include all features, nor complete explanations of the features listed. For more detailed descriptions, the reader is referred to subsequent sections.

<u>Mnemonic Operation Codes:</u> Mnemonic operation codes, provided for all machine instructions, are used instead of the more cumbersome internal operation codes of the machine. For example, the Branch-on-Condition instruction can be represented by the mnemonic BC, instead of the machine operation code 01000111. The various machine mnemonic operation codes are presented under the topic <u>Machine Instruction Mnemonics</u>.

<u>Symbolic Referencing of Storage Addresses:</u> Instructions, data areas, register numbers, and other program elements can be referred to by symbolic names instead of actual machine addresses and designations. See the topic <u>Symbols</u>.

<u>Automatic Storage Assignment:</u> The assembler assigns consecutive addresses to program elements as it encounters them. After processing each element, the assembler increments a counter by the number of bytes assigned to that element. This counter indicates the storage location available to the next element. See the topic <u>Location Counter</u>.

<u>Convenient Data Representation:</u> Constants can be specified as decimal digits, alphabetic characters, hexadecimal digits, and storage addresses. The assembler converts the data into a machine format compatible with IBM System/360. This data can be in a form suitable for use in fixed-point and floating-point arithmetic operations. See the topic <u>DC - Define Constant</u>.

<u>Renaming Symbols:</u> A symbolic name can be equated to another symbol so that both refer to the same storage location, general register, etc. This enables the same program item to be referred to by different names in different parts of the program. See <u>EQU - Equate Symbol</u>.

<u>Program Linking:</u> Independently assembled programs to be loaded and executed together may make symbolic references to instructions and data in each other. See the discussion of program link instructions.

<u>Relocatable Programs:</u> The assembler produces object programs in a relocatable format; that is, a format that enables

programs to be loaded and executed at storage locations different from those assigned when the programs were assembled.

Assembler Instructions: A set of special instructions for the assembler is included in the language. Some features described in this section are implemented by these instructions. See the topic Assembler Instructions.

Base Register and Displacement Assignment: The programmer can instruct the assembler to assign base registers and compute displacements for symbolic machine addresses. See the discussion of Base Register Instructions.

Program Listings: For every assembly, the assembler can provide a listing of the source program and the resulting object program. A description of the listing format can be found under the topic Program Listing.

Error Checking: Source programs are examined by the assembler for possible errors arising from incorrect usage of the language. Wherever an error is detected, a coded error message (a flag) is printed in the program listing. For card systems without printers, limited error notification is provided. See the topic Error Notification.

Program Reassembly: A special reassembly procedure is provided for programs assembled by the IBM 1442 Model N1 or 2520 Model B1 Card Read-Punch card-operating procedure. This permits partially or completely assembled (and modified) source programs to be reassembled in less time than required for a new assembly. See the topic Reassembly Procedure.

Device Assignment: The assembler has five types of input/output. Four (the assembler, source program, intermediate text, and object code) can use card read punch or tape; the fifth (the listing) can use printer, printer-keyboard, or tape.

> Note: If tape is used for listing, it must be 800 BPI or less. Also, tape may be used for listing only with a Model 40 or larger system because the speed of these systems is sufficient to handle "chain data."

If series 2400 tape drives are available (either seven- or nine-track), one to five drives may be used in the assembly at the user's option. If one tape unit is available, it may be used for any of the five input/output types enumerated above. If two tape units are available, they may be used for any two of the five input/output types, and so on. The user indicates the input/output types by means of "Configuration Cards." Details concerning these cards are found in the section Basic Assembler Operating Procedures.

## COMPATIBILITY WITH OTHER SYSTEM/360 ASSEMBLERS

Programs written in the Basic Assembler language as described in this publication are acceptable to the other Basic Programming Support, Basic Operating System, and Operating System Assemblers, and the 7090/7094 Support Package Assembler. Similarly, any source programs written in these other assembly languages are acceptable to the Basic Assembler if they are compatible to the Basic Assembler. Appendix C, the System/360 Assemblers-Language Features Comparison Chart may be used as a guide for the exchange of source programs between assemblers.

The assembler also accepts programs written for the IBM System/360 Model 20 Basic Assembler, except where differences in machine design have made it necessary to include some instructions in the Model 20 Basic Assembler language that are not contained in the Basic Assembler Language. These instructions are:

BAS BASR CIO HPR SPSW TIOB XIO
Y-type Expression Constants

Note also that the pseudo-registers, zero through three, on the Model 20 are handled differently from the corresponding actual registers on other models of the System/360.

## MACHINE REQUIREMENTS

The assembler operates on an IBM System/360 with the following minimum configuration:

8,192 bytes of storage
Standard Instruction Set
An IBM 1442 Model N1, 2540, or 2520 Model B1 Card Read-Punch; or
An IBM 2501 Card Reader with a 2520 Model B2 or B3 Card Punch

This configuration is for the card-operating procedure for the assembler, providing card intermediate text.

If IBM 2400-series Magnetic Tape Units are available in addition to the equipment required for card intermediate text, the tape-operating procedure may be used to provide tape intermediate text, if desired.

If an IBM 1443 Model N1 or 1403 Printer, or an IBM 1052 Printer-Keyboard is provided, the assembler provides a program listing, complete with error flags, for each assembly. An option is available to list only those statements containing errors. For information concerning this option, refer to Program Listing.

### CARD OR TAPE INTERMEDIATE TEXT

The assembler is a two-phase program. The first phase produces data for use by the second phase. The intermediate data produced by the first phase must be passed on to the second phase via some external storage medium. The storage mediums used are punched cards or magnetic tape. The machine configuration determines which option applies at a particular installation.

## BASIC ASSEMBLER LANGUAGE

### BASIC ASSEMBLER CARD FORMATS

An assembler language source program consists of a sequence of source statements punched into cards, one statement per card. Source programs may also be loaded from tape, in unblocked card-image records. The card columns available for punching source statements vary with the machine configuration (that is, input device, card or tape option) and at the programmer's discretion. (See Figure 1.)

| Source Input Unit | Intermediate Text | Columns Available |
|---|---|---|
| 2540 | tape | 1-71 or 25-71 |
| 2540 | card | 1-47 or 25-71 |
| 2501 | tape | 1-71 or 25-71 |
| 2501 | card | 1-47 or 25-71 |
| 1442-N1 | tape | 1-71 or 25-71 |
| 1442-N1 | card | 25-71 |
| 2520-B1 | tape | 1-71 or 25-71 |
| 2520-B1 | card | 25-71 |
| tape | tape | 1-71 or 25-71 |
| tape | card | 1-47 or 25-71 |

Figure 1. Source Program Column Assignment

1. Columns 1-71 (rather than only columns 1-47) may be used with a 2540 alone, or with a 2501 and a 2520-B2 or B3 input and card intermediate text. The assembler scans all 71 columns of the statement field when obtaining the information required to generate the appropriate object code. However, only the contents of columns 1-47 and 73-80 are included in the program listing produced by the assembler. Columns 1-24 must be blank when using a 1442-N1 or 2520-B1 input and card intermediate text.

2. The use of tape source input and card intermediate text is not recommended for a 1442-N1 or 2520-B1 system. If this option is selected, the assembly proceeds normally, and the source statement does not appear on the listing.

(When tape is used for input, its format is that of 80-byte unblocked records. Each record is equivalent to a card, each byte representing one card column.)

In addition to a source statement, each card may contain an identification sequence number in columns 73-80.

The discussion of card formats assumes that card input and intermediate text are used, and all statements begin in column 1. When card column assignments differ because of statements beginning in column 25, the column numbers associated with the statements beginning in column 25 are placed in parentheses, e.g., 1(25).

The statements may be written on one of two standard coding forms provided by IBM: a "long" form, Form X28-6507 (Figure 2), and a "short" form, Form X28-6506, for IBM Card Read-Punch card-option assemblies (Figure 3).

Each line of the coding form is used for a single statement and/or comments. The information on each line is punched into one card. If a card is completely blank, it is ignored by the assembler. The position numbers on the forms correspond to the card columns.

Space is provided at the top of both coding forms to identify the program and provide instructions for the keypunch operator. None of this information is punched into the statement cards.

Statement Fields

An assembler statement is composed of one to four fields, from left to right: name field, operation field, operand field, and comments field. The identification-sequence field is not part of the statement. The statement fields can be written on the coding form in what basically is a free form. As a convenience, however, the name and operation fields are marked on the coding forms by heavy lines to indicate the maximum length of these fields. Programmers may wish to align the fields at these lines to create a neat and orderly appearance in the program listing.

General rules that must be observed when writing statements are:

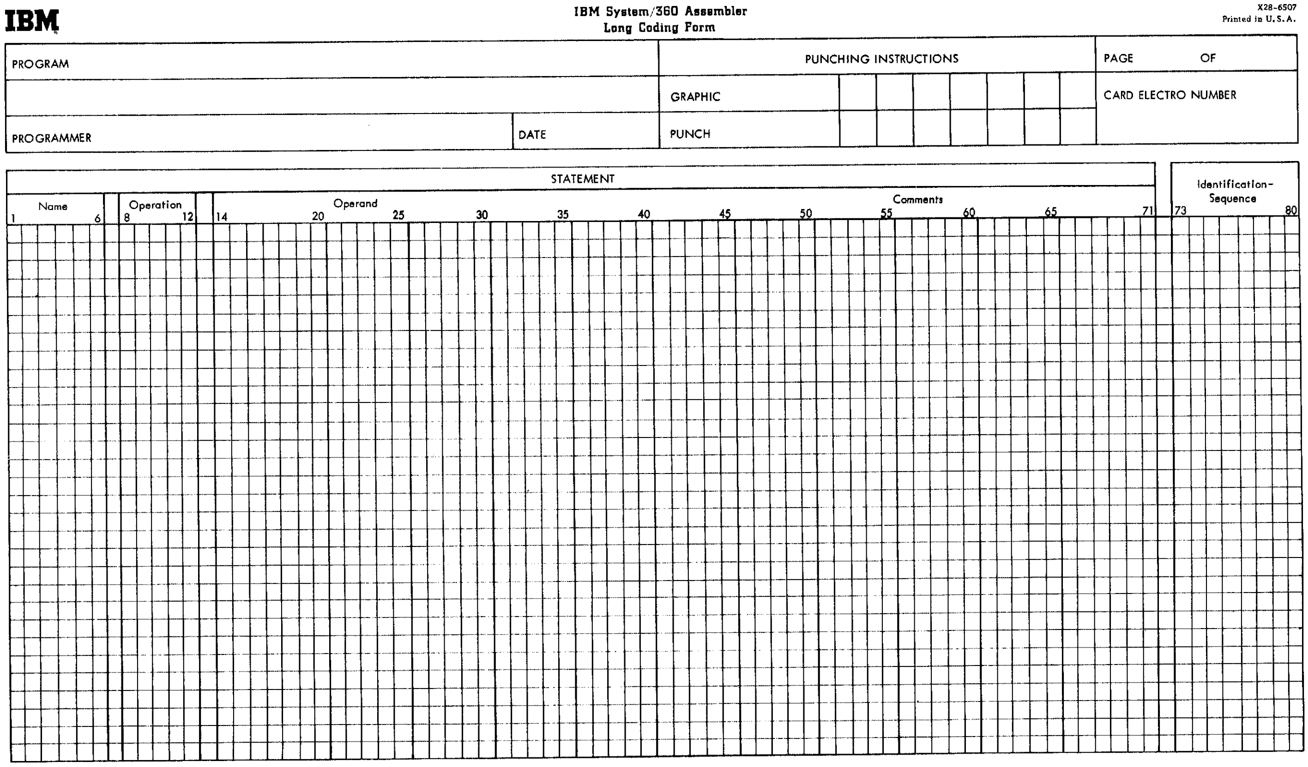
1. The only required field in a statement is the operation field. The other fields are optional, depending on the operation and the programmer's wishes.

2. The fields in a statement must be in order and separated from one another by at least one blank.

3. The name, operation, and operand fields must not contain embedded blanks. A blank may, however, occur in the operand field as a character self-defining value or character constant.

4. Only one statement is allowed to a line; a statement cannot be continued on additional lines.

5. Column 72 must be blank.



IBM

IBM System/360 Assembler
Long Coding Form

X28-6507
Printed in U.S.A.

PROGRAM

PUNCHING INSTRUCTIONS

PAGE OF

GRAPHIC

CARD ELECTRO NUMBER

PROGRAMMER

DATE

PUNCH

STATEMENT

Name 1 6

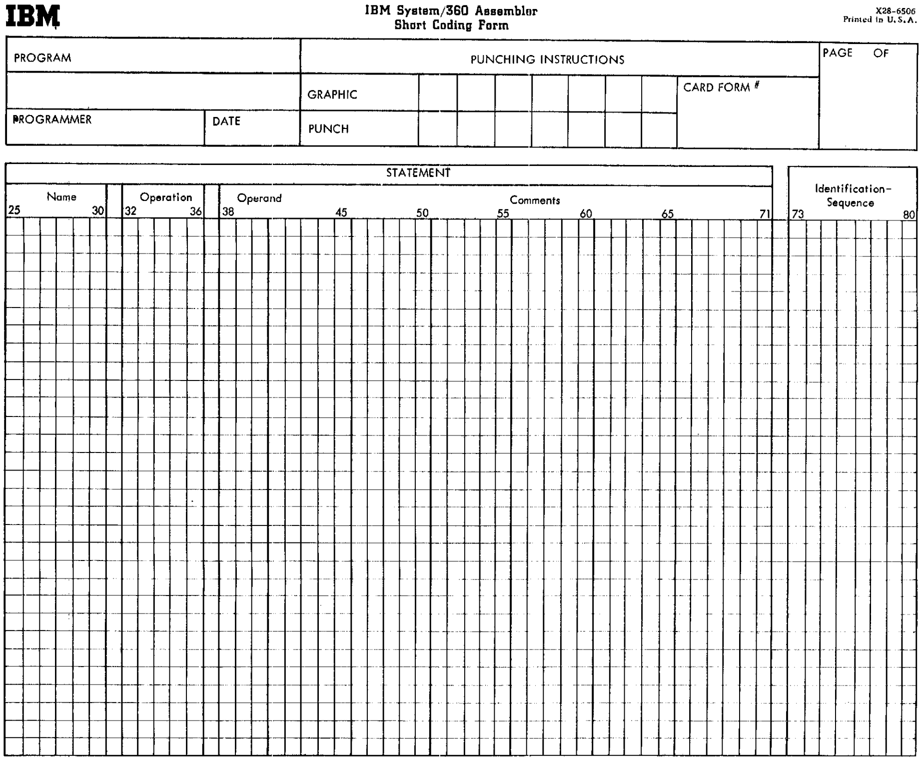
Operation 8 12

Operand 14 20 25 30 35 40 45 50

Comments 55 60 65 71

Identification-Sequence 73 80

Figure 2. IBM System/360 Long Coding Form



IBM

IBM System/360 Assembler
Short Coding Form

X28-6506
Printed in U.S.A.

PROGRAM

PUNCHING INSTRUCTIONS

PAGE OF

GRAPHIC

CARD FORM #

PROGRAMMER

DATE

PUNCH

STATEMENT

Name 25 30

Operation 32 36

Operand 38 45 50

Comments 55 60 65 71

Identification-Sequence 73 80

Figure 3. IBM System/360 Short Coding Form

Name Field: The name field is used to assign a symbolic name to a statement. A name enables other statements to refer to the statement by that name. If a name is given, it must begin in column 1 (25) and must not extend beyond column 6 (30). A name is always a symbol and must conform to the rules for symbols (see the section, Symbols). Figure 4 shows the symbol FIELD2 used as a name. The number of symbols in an assembly is limited. The limit varies with main storage size. For specific information see Symbol Table.

If column 1 (25) is blank, the assembler will assume that the statement has no name. Column 1 (25) is also used to indicate that the card is a comments card (see Comments Field).

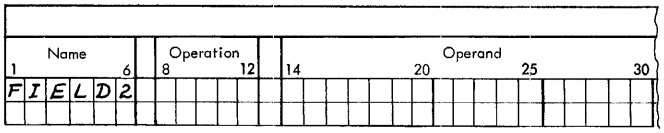


Figure 4. Example of the Name Field

Operation Field: The operation field is used to specify the mnemonic operation code of a machine or assembler instruction. This field may begin in any column to the right of column 1 (25) if the name field is blank. If the name field is not blank, at least one blank must separate the name and operation fields. The operation field may contain any valid mnemonic operation code. The valid machine-instruction mnemonics are listed in Machine Instruction Statements. The valid assembler-instruction mnemonics are listed in the section Assembler Instructions. A valid mnemonic must never exceed five characters. If an invalid mnemonic is specified, the assembler treats the statement as a comments statement and flags an error.

Figure 5 shows the mnemonic for the compare instruction (RR format) used in a statement named TEST. Note that this mnemonic could have been placed in columns 6-7, since this would have satisfied the requirement that at least one blank space separate the fields.

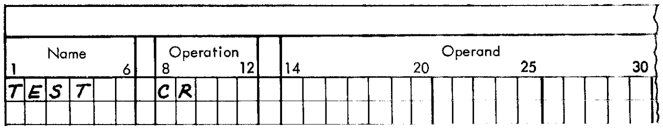
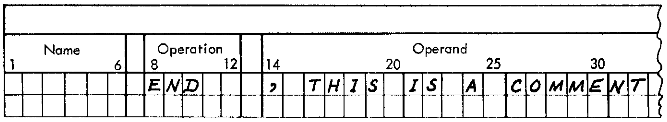


Figure 5. Example of the Operation Field

Operand Field: The operand field provides the assembler with additional information about the instruction specified in the operation field. If a machine instruction has been specified, the operand field contains information required by the assembler to generate the machine instruction. The operand field specifies registers, storage addresses, input/output devices, immediate data, masks, and storage-area lengths. For an assembler instruction, the operand field conveys whatever information the assembler requires for the particular instruction.

The operand field may begin in any column to the right of the operation field, provided at least one blank space separates it from the last character of the mnemonic.

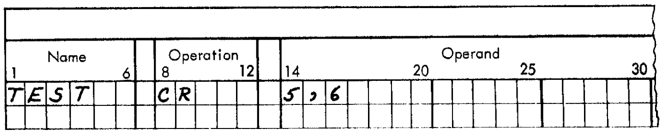
Certain assembler instructions do not require the operand field to be specified. If there is no operand field but there is a comments field, the absence of the operand field must be indicated by a comma, preceded and followed by one or more blanks. Figure 6 illustrates this rule.



```
       END   , THIS IS A COMMENT
```

Figure 6. Example of No Operand Field with Comments

Depending on the instruction, the operand field may be composed of one or more subfields, called operands. Operands must be separated by commas. It must be remembered that a blank delimits the field; thus, a blank must not intervene between operands and commas. Figure 7 is an example of the same compare instruction shown in Figure 5, with its two operands specifying general registers 5 and 6. In Figure 7, as in Figure 5, the fields are separated by more than the minimum number of blank spaces.



```
TEST   CR    5,6
```

Figure 7. Example of the Operand Field

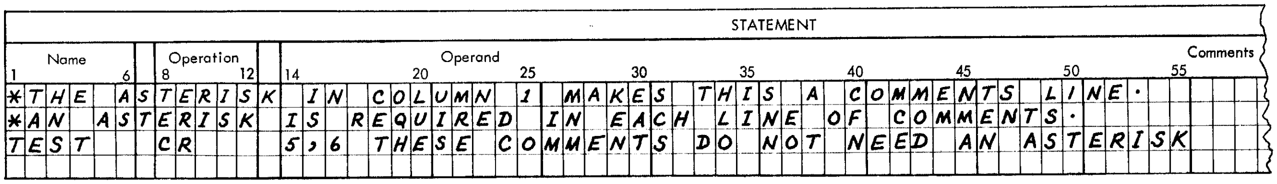
Comments Field: Comments, provided for the convenience of the programmer, permit lines or paragraphs of descriptive information about the program to be inserted into the program listing. Comments appear only in the program listing; they have no effect on the assembled object program. Any valid characters (including blanks) may be used as comments.

The comments field must (1) appear to the right of the operand field, and (2) be preceded by at least one blank. If there is no operand field but there is a comments field, the absence of the operand field must be indicated by a comma, preceded and followed by one or more blanks. The entire statement field can be used for comments by placing an asterisk in column 1 (25); the entire statement will be treated as comments. Column 72, however, must remain blank.

If it is necessary to continue full-card comments on additional lines, each such line must have an asterisk in column 1 (25), as illustrated in Figure 8.

## Identification-Sequence Field

The identification-sequence field may be used for program identification and statement sequence numbers. This field can occupy columns 73-80 only. The information normally is punched in every statement card. The assembler, however, will not check this field. It will merely reproduce the information in the field on the output listing of the program.



```
*THE ASTERISK IN COLUMN 1 MAKES THIS A COMMENTS LINE.
*AN ASTERISK IS REQUIRED IN EACH LINE OF COMMENTS.
TEST   CR    5,6 THESE COMMENTS DO NOT NEED AN ASTERISK
```

Figure 8. Example of the Comments Field

## WRITING BASIC ASSEMBLER STATEMENTS

Language statements are accepted by the assembler only if they conform to the established grammatical rules and vocabulary restrictions presented in this section. The reader can expect that many points not fully explained when first mentioned in this section are subsequently described in detail.

### Character Set

Basically, statements may be written using the following characters:

A through Z
0 through 9
* + - , ( ) ' . blank

The card column punch-combinations that the assembler accepts for these characters are listed below. This list also contains the punches assumed for additional printer graphics, which may be used in comments. The punch combinations accepted by the assembler are those of the Extended Binary Coded Decimal Interchange Code (EBCDIC). Note that the punch combinations for +, (, ), =, and ' are different from those of Binary Coded Decimal (BCD).

| Character | Punch Combination |
|---|---|
| A - I | 12 punch and a 1 - 9 punch, respectively |
| J - R | 11 punch and a 1 - 9 punch, respectively |
| S - Z | 0 (zero) punch and a 2 - 9 punch, respectively |
| 0 - 9 | 0 (zero) - 9, respectively |
| blank | No punches |
| & | 12 |
| / | 0-1 |
| - | 11 |
| . (period) | 12-3-8 |
| $ | 11-3-8 |
| , | 0-3-8 |
| # | 3-8 |
| < | 12-4-8 |
| * | 11-4-8 |
| % | 0-4-8 |
| @ | 4-8 |
| ( | 12-5-8 |
| ) | 11-5-8 |
| ' (single quotation) | 5-8 |
| + | 12-6-8 |
| = | 6-8 |

### Symbols

Symbols are created and used by the programmer for symbolic referencing of storage areas, instructions, input/output units, and registers.

A symbol may contain from one to six characters, in any combination of alphabetic (A through Z) and numeric (0 through 9) characters. The first character must be alphabetic. Special characters and embedded blanks must not be used in symbols. Any violation of these rules is noted by an error flag in the program listing. The symbol will not be used.

The following are valid symbols:

READER
A23456
LOOP2
N
S4

These symbols are invalid:

| | |
|---|---|
| 256B | First character is not alphabetic |
| AREATWO | More than six characters |
| RCD*34 | Contains a special character |

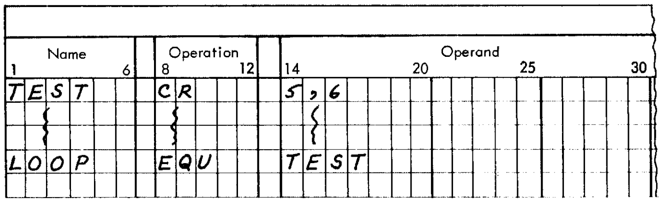
*Defining Symbols:* Symbols are meaningful in statements when used as operands and names. In order for a symbol to be used as an operand, it must be defined somewhere in the program. When a symbol is used as an operand, and therefore defined, the assembler will normally assign certain attributes to it.

A symbol is defined when used as the name of a statement. When the assembler finds a symbol in the name field, it will assign an address-value attribute and a length attribute to the symbol. The address value is the storage address of the leftmost byte of the field allotted to the statement; the length is the number of bytes in the storage field named by the symbol. This length is called the implied length associated with the symbol. The convenience of having implied lengths becomes apparent in the discussion of the symbolic format of machine instructions in the SS format.

A symbol defined in this manner is normally called a *relocatable* symbol. That is, the address value of the symbol changes if the program is loaded at a location other than its assembled location.

Symbols can be assigned arbitrary absolute values by use of the EQU assembler instruction. These values may designate registers, input/output units, immediate data, etc. They can also specify actual storage addresses such as permanently allocated interrupt locations. Symbols so defined are termed absolute symbols since their values are fixed and will not change because of program location.

Previously Defined Symbols: Sometimes the programmer will desire to give an alternate name to a previously defined symbol. "Previously defined" means that the symbol has appeared as the name of some statement prior to being used in the operand field of another statement. Figure 9 shows how the symbol TEST, defined in the first statement, is given an alternate name.



| Name 1 6 | Operation 8 12 | Operand 14 20 25 30 |
|---|---|---|
| TEST | CR | 5,6 |
| LOOP | EQU | TEST |

Figure 9. Example of Coding with Previously Defined Symbols

External and Entry-Point Symbols: Symbols are normally defined in the same program in which they are used as operands. It is possible, however, to define a symbol in one program, use it in another program assembled independently of the first, and then execute both programs together. Such a symbol is called an "external symbol" when it is used as an operand. The symbol is termed an "entry-point symbol" in the program in which it is defined. The address value of the entry-point symbol is assigned to the external symbol when both programs are loaded by the relocating loader.

Before using an external symbol or defining an entry-point symbol, the programmer must indicate to the assembler which symbols are external and which are entry points. The ENTRY and EXTRN assembler instructions are provided for this purpose. Both instructions are described in Assembler Instructions.

External symbols are always relocatable. They are subject to certain usage restrictions that are discussed later in this publication.

General Restrictions on Symbols: The following restrictions are in addition to those imposed elsewhere in the discussion of symbols:

1. A symbol may appear only once in a program as the name of a statement. If a symbol is used as a name more than once, only the first usage will be recognized. Each subsequent usage of the symbol as a name will be ignored and noted with an error flag in the program listing.
2. The number of symbols that may be defined in a program is restricted, depending on the machine's storage size. These restrictions are explained in detail in the section The Symbol Table.
3. A symbol must always be defined as having a positive value not exceeding 65,535. Any symbol whose definition is contrary to this rule will not be used, and the statement in which it appears will be flagged as an error.

## The Location Counter

The assembler maintains a counter (the Location Counter) used to assign consecutive storage addresses to program statements. It always points to the current address. After each machine instruction is processed, the Location Counter is incremented by the number of bytes assigned to that instruction. Certain assembler instructions also cause the Location Counter to be incremented, others do not affect it.

The programmer can set and change the Location Counter by using the START and ORG assembler instructions described in Assembler Instructions.

Location Counter Overflow: The maximum value of the Location Counter is 65,535, a 16-bit value. If a program being assembled causes the Location Counter to be incremented beyond 65,535, the assembler will retain only the rightmost 16 bits in the counter and continue the assembly, checking for any other source program errors. No object program is produced. The assembler can, however, provide a listing of the entire source program. The statement causing the overflow is flagged in the listing.

Program References: The programmer may refer to the current value of the Location Counter at any place in a program by using an asterisk as an operand. The asterisk represents the location of the first byte currently available. The use of an asterisk in a machine-instruction statement is the same as giving the statement a name and then using that name as an operand in the same statement. Note that the asterisk has a different address value each time it is used. The asterisk has a length attribute of 6, except in an EQU statement where the length attribute is 1. An asterisk used as an operand is considered a relocatable symbol.

## Self-Defining Values

The ability to represent an absolute value symbolically is an advantage in cases where the value will be referred to repeatedly. However, it is equally necessary to have a convenient means of specifying an actual machine value or a bit configuration without having to go through the procedure of equating it to a symbol and using the symbol. The assembler language provides this facility through the self-defining value, which can be a decimal, hexadecimal, or character representation.

Self-defining values may be used to specify such program elements as immediate data, masks, registers, addresses, and address increments. The type of representation selected (decimal, hexadecimal, or character) will depend on what is being specified. The use of a self-defining value is quite distinct from the use of data constants specified by the DC assembler instruction and by literal operands. When a self-defining value is used in a machine-instruction statement, its value is assembled into the instruction. When a data constant is specified in a machine instruction, its address is assembled into the instruction.

Decimal: A decimal self-defining value is an unsigned number from one through six decimal digits. A decimal self-defining value of more than six digits is not valid. The acceptable decimal digits are 0 through 9. Some examples are:

```
  7     4092      0007
147      128    199860
```

The assembler imposes additional restrictions on decimal self-defining values, depending on their use. For example, a decimal self-defining value designating a general register should be from 0 through 15; one designating a core storage address should not exceed the size of available storage.

Hexdecimal: A hexadecimal self-defining value is an unsigned number of from one to six hexadecimal digits, enclosed in single quotation marks, and preceded by the letter X. Hexadecimal self-defining values of more than six digits are not valid.

Each hexadecimal digit converts to a four-bit value. The hexadecimal digits, and their bit patterns are:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0000 | 4 | 0100 | 8 | 1000 | C | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | D | 1101 |
| 2 | 0010 | 6 | 0110 | A | 1010 | E | 1110 |
| 3 | 0011 | 7 | 0111 | B | 1011 | F | 1111 |

The following are examples of hexadecimal self-defining values:

```
X'25'    X'B'     X'12FA1E'
X'F4F'   X'00CD'  X'00E0'
```

A table for converting decimal values to hexadecimal is provided in Appendix B.

Character: A character self-defining value is a single character, enclosed in single quotation marks, and preceded by the letter C. A character self-defining value may be a blank or any combination of punches in a single card column that translates into the 8-bit IBM Extended Binary Coded Decimal Interchange Code (EBCDIC). There are 256 such combinations. Appendix A is a table of these combinations, their interchange codes, and, where applicable, their printer graphics. A single quotation mark used as a character self-defining value, or an ampersand, is represented as two single quotation marks, or two ampersands, enclosed in single quotation marks, thus: C'''' or C'&&'

Examples of character self-defining values are:

```
C'/'     C'#'     C'.'
C'B'     C'2'     C' ' (blank)
```

The same value can frequently be represented by any one of the three types of self-defining values. Thus, the decimal self-defining value 196 can be expressed in hexadecimal as X'C4' and as a character C'D'. The selection of a particular type of value is left to the programmer. Decimal self-defining values, for example, might be used for actual addresses or register and input/output unit numbers, hexadecimal self-defining values for masks, and character self-defining values for immediate data.

## Expressions

The term "expression" refers to symbols or self-defining values used as operands, either alone or in some arithmetic combination. Expressions are used to specify the various fields of machine instructions and as the operands of assembler instruction statements.

Expressions are classified as either simple or compound, relocatable or absolute. Unless otherwise qualified, the term "expression" hereinafter implies any expression, simple or compound, relocatable or absolute.

A simple expression is a single unsigned symbol (including the asterisk used as the Location Counter value) or a single unsigned self-defining value used as an operand. The following are simple expressions:

```
FIELD2        2          C'R'
X'BF'         *          ALPHA
```

A compound expression is a combination of two or, at most, three simple expressions, connected to each other by arithmetic operators. The recognized operators are + (plus), - (minus), and * (asterisk), denoting, respectively, addition, subtraction, and multiplication. The following are compound expressions:

```
N+14*256       ENTRY-OVER
FIELD+X'2D'    *+GAMMA-200
```

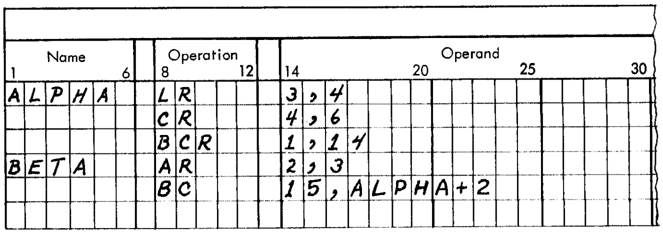
Note that an asterisk is used for the Location Counter (*+GAMMA-200) and as an operator (N+14*256), but cannot be used in succession to denote the two in the same expression. The following example is invalid:

```
**5
```

A compound expression must not contain either two simple expressions or two operators in succession, nor may it begin with an operator. The following examples violate these rules and, therefore, are invalid:

```
AREAX'C'       -DELTA+256
FIELD+-10      +FIELD-10
```

Relative Addressing: Relative addressing is a technique of addressing instructions and data areas by designating their location in relation to the Location Counter or to some symbolic location. This type of addressing is always in bytes, never in words or instructions. In the sequence of instructions shown in Figure 10, the location of the CR machine instruction can be expressed as ALPHA+2 or BETA-4, because all mnemonics in this example are for 2-byte instructions in the RR format except the last, which is in the RX format. The expression *+3 specifies an address that is three bytes greater than the current value of the Location Counter.



```
ALPHA  LR   3,4
       CR   4,6
       BCR  1,14
BETA   AR   2,3
       BC   15,ALPHA+2
```

Figure 10. Example of Relative Addressing

Attributes of Expressions: The assembler separately evaluates each expression in the operand field. An expression is terminated by a comma, a left or right parenthesis, or a blank, depending on what the expression specifies (see section Machine Instruction Statements). The evaluation procedure is as follows:

1. Each simple expression is given its numerical value.
2. Arithmetic operations are performed from left to right, with multiplication before addition and subtraction. Thus, A+B*C is evaluated as A+(B*C) and not (A+B)*C.
3. The arithmetic result becomes the value attribute of the expression.

In addition to computing the value attribute of an expression, the assembler also determines its length attribute. For a compound expression, the length attribute is the same as the implied length attribute of its leftmost simple expression. If the leftmost simple expression in an expression is a self-defining value, the implied length attribute of that expression is one byte. If it is an asterisk, the implied length attribute is six bytes.

Absolute and Relocatable Expressions: An expression is absolute if its value is unaffected by program relocation. An absolute expression either:

1. Contains only absolute symbols or self-defining values.
2. Is of the following forms (where R is a relocatable symbol, and A is an absolute symbol or self-defining value):

```
R-R      R-R+A    R-R-A    A+R-R
R-A-R    A-R+R    R+A-R
```

Although the address values of both relocatable symbols are subject to change when the program is loaded, the difference between their values is constant; that is, absolute.

An expression is relocatable if its value changes upon program relocation, for example, when the value of an expression changes by N if the program was loaded N bytes away from its assembled location. Relocatable expressions must conform to the following rules:

1. A relocatable expression must contain either one or three relocatable symbols. If there are three relocatable symbols, one (and only one) must be preceded by the minus (-) operator. If only one relocatable symbol is present, it must not be preceded by the minus operator.
2. A relocatable symbol may not be multiplied. That is, it must not be preceded or followed by the asterisk (*) operator.

The following examples illustrate absolute and relocatable expressions. R represents relocatable symbols; A, absolute symbols.

Absolute Expressions:
R-R+5
A+14*C'H'
2048
A*A

Relocatable Expressions:
R+2
R-8*A
R-R+R
*-X'FB2'
R-A

The following expressions are invalid for the reasons listed:

| | |
|---|---|
| R+R<br>R+R-A | Contain two relocatable symbols. |
| R*A | Relocatable symbol is multiplied. |
| R+R+R | No minus operator. |
| A-R | Single relocatable symbol is preceded by a minus operator. |
| R-R-R | Two minus operators. |

Restrictions: The following restrictions apply to all expressions. Additional limitations are imposed where pertinent in this publication.

1. An expression can have a negative value only when it is an absolute expression specifying an address constant using the DC assembler instruction.
2. An expression containing an external symbol may not contain any other relocatable symbols. For the purpose of evaluating such an expression, the value of the external symbol at assembly time is zero; the symbol is revalued when the program is loaded.
3. If an expression is used as the operand of a machine instruction statement, any self-defining values within it must not exceed 4095. Instructions containing self-defining values exceeding 4095 are set to zero. The operation code remains unchanged.
4. The maximum value of an expression is 65,535. If an expression exceeding this maximum value is used in a machine instruction statement, the entire instruction except for the operation code is set to zero. If that expression is used in an assembler instruction statement, the action taken depends on the instruction.

Note: The maximum value of each individual term in the operand field of USING, ORG, END, EQU, CCW (second operand), and DC (A) assembler instructions must not exceed 16,777,215. The maximum value of an entire expression in an operand field of a USING, ORG, END, or EQU instruction is, however, 65,535. The maximum value of an entire expression in the operand field of a DC (A) or CCW (second operand) instruction is 16,777,215.

## MACHINE INSTRUCTION STATEMENTS

The assembler language provides for the symbolic representation of all machine instructions. The symbolic format of these instructions varies with the machine format. There are five basic machine formats: RR, RX, RS, SI, and SS. Within each basic format, further variations are possible.

Machine instructions are automatically aligned by the assembler on half-word boundaries. Any byte skipped because of alignment is set to zero. Such situations arise when data is inserted into the instruction string, as in a calling sequence.

Any machine instruction statement may be given a name which other assembler statements can use. The value attribute of such a name is the address of the leftmost byte assigned to the assembled instruction. The length attribute of the name depends on the basic machine format as follows:

| Basic Machine Format | Implied Length Attribute (in Bytes) |
|---|---|
| RR | 2 |
| RX | 4 |
| RS | 4 |
| SI | 4 |
| SS | 6 |

### Instruction Format

Figure 11 shows each basic machine format followed by its corresponding symbolic operand field formats and mnemonic operation codes. The numbers in the basic machine formats are the bit sizes of the field.

Figure 12 identifies the field codes used in Figure 11 and contains pertinent information for specifying the fields in machine instruction statements. The following are additional points that must be considered:

1. If no indexing is used in an RX instruction and the base register (B2) is present, the X2 field must be written as a zero. If not written as a zero, the base register is assembled as an index register (X2). If indexing is used, and the base register is implied, the base register field may be omitted.

2. If the field or fields enclosed in parentheses are omitted, the parentheses (and the comma between them) may also be omitted.

3. If the value of an absolute expression exceeds the maximum value (stated in Figure 12) for a field, the entire instruction is set to zero except for the operation code; the statement is then flagged in the program listing. This does not apply to the displacement field.

4. If the value of a displacement field exceeds 4095, only the rightmost 12 bits are used; the listing is then flagged.

5. If the programmer writes an absolute expression specifying a displacement and does not specify a base register, the assembler places zero in the base-register field. The same applies to the index register.

6. If any invalidity in the operand field (other than those listed above) prevents correct evaluation of an expression, the entire instruction except for the operation code is set to zero, and the statement is flagged. Such invalidities would include undefined symbols, use of relocatable expressions when absolute expressions are called, etc.

| Basic Machine Format | | | | | | | | Assembler Operand Field Format | Applicable Instructions |
|---|---|---|---|---|---|---|---|---|---|
| Bits | 8 | 4 | 4 | 4 | 12 | 4 | 12 | | |
| RR | Op Code | R1 | R2 | N/A | N/A | N/A | N/A | R1,R2 | All RR instructions except BCR,SPM,SVC |
| | Op Code | M1 | R2 | N/A | N/A | N/A | N/A | M1,R2 | BCR |
| | Op Code | R1 | 0 | N/A | N/A | N/A | N/A | R1 | SPM |
| | Op Code | I | | N/A | N/A | N/A | N/A | I | SVC |
| RX | Op Code | R1 | X2 | B2 | D2 | N/A | N/A | R1,D2(X2,B2) | All RX instructions except BC |
| | Op Code | M1 | X2 | B2 | D2 | N/A | N/A | M1,D2(X2,B2) | BC |
| RS | Op Code | R1 | R3 | B2 | D2 | N/A | N/A | R1,R3,D2(B2) | BXH,BXLE,LM,STM |
| | Op Code | R1 | 0 | B2 | D2 | N/A | N/A | R1,D2(B2) | All shift instructions |
| SI | Op Code | I2 | | B1 | D1 | N/A | N/A | D1(B1),I2 | All SI instructions except LPSW,SSM,HIO,<br>SIO,TIO,TCH |
| | Op Code | 0 | 0 | B1 | D1 | N/A | N/A | D1(B1) | LPSW,SSM,HIO,<br>SIO,TIO,TCH,TS |
| SS | Op Code | L1 | L2 | B1 | D1 | B2 | D2 | D1(L1,B1),D2(L2,B2) | PACK,UNPK,MVO,<br>AP,CP,DP,MP,SP,<br>ZAP |
| | Op Code | L | | B1 | D1 | B2 | D2 | D1(L,B1),D2(B2) | NC,OC,XC,CLC,<br>MVC,MVN,MVZ,TR,<br>TRT,ED,EDMK |

Figure 11. Machine Instruction Statement Formats

## Implied Base Registers and Displacements

The assembler has the facility for assigning base registers and computing displacements for symbolic storage addresses. This is accomplished by programmer specification of a symbolic address through the use of a relocatable symbol. This implies that the assembler is to select the base register and displacement. Before this can be done, however, the programmer must indicate to the assembler the contents and number of the general registers available for base registers. The USING and DROP instructions described in the section Base Register Instructions, convey this information.

Base registers and displacements can be implied for RX, RS, SI, and SS instructions. For example, the operands of an RS instruction can be specified as

```
R1, R3, S2
```

where S2 represents a symbolic address (i.e., a relocatable symbol) that the assembler will separate into a displacement (D2) and base register (B2).

To specify addresses in this manner, the programmer must observe these rules:

1. The base register instructions (USING and DROP) must be used as described in this publication (see Base Register Instructions).
2. The symbolic address must be represented by a simple or compound relocatable expression.
3. A base register must not be written. An explicit base register will cause the assembler to treat the storage address as a displacement, and an error will result because a displacement must always be an absolute expression. An explicit index register may be used, however, in the usual manner.

In the following example, the relocatable expression FIELD, with an address value of 7400 (decimal), is used in a machine instruction; assume that the assembler has been told that general register 12 contains 4096 (decimal) and is available as a base register.

```
ST    4,FIELD
```

The assembled machine instruction (in hexadecimal) would be as follows, the value of D2 being the difference between 7400 and 4096.

| Operation Code | R1 | X2 | B2 | D2 |
|---|---|---|---|---|
| 50 | 4 | 0 | C | CE8 |

If the instruction was ST 4,FIELD(2), the assembled machine instruction would differ from the previous example only in that the content of the X2 field would be 2 rather than zero.

Reference Summary for Operand Fields

| Field Code | Code Represents | Field Bit Size | Expression | |
|---|---|---|---|---|
| | | | Allowable Types | Maximum Values |
| R1,R2,R3 | General or floating-point register | 4 | Simple absolute | 15 |
| M1 | Mask | 4 | Simple absolute | 15 |
| D1,D2 | Displacement | 12 | Simple or compound absolute | 4095 |
| B1,B2 | Base register | 4 | Simple absolute | 15 |
| X2 | Index register | 4 | Simple absolute | 15 |
| L1,L2 | Length | 4 | Simple absolute | 16* |
| L | Length | 8 | Simple absolute | 256* |
| I2,I | Immediate | 8 | Simple absolute | 255 |

* These are maximum values for length fields allowed in assembler statements; the values assembled for the instruction length fields are one less than these values.

Figure 12. Operand Field Summary

### Implied and Explicit Lengths

The length field in SS instructions can be implied or explicit. An implied length is the length attribute of either the absolute expression specifying the displacement or the relocatable expression specifying the symbolic address, whichever is written in the statement. The length attribute of a compound expression is the implied length of its leftmost simple expression.

An explicit length, by contrast, is written by the programmer in the statement as a simple absolute expression. If a length is explicit, it overrides the implied length associated with the displacement or symbolic address.

Regardless of how the length is specified (implied or explicit), if it exceeds the values indicated in Figure 12 for the L, L1, and L2 fields, the entire assembled instruction, except the operation code, will be set to zero.

Note that the length, whether implied or explicit, is always an effective length. That is, it is one more than the value inserted into the length field of the assembled machine instruction. In the case where an explicit length of zero is specified, the assembler assumes an effective length of one. Thus, a zero is inserted in the length field of the assembled instruction.

The reference summary in Figure 12 is for use with the figure showing the machine instruction formats (Figure 11). For each explicit operand format in column 1, any of the corresponding implied operand formats in columns 2, 3, or 4 can be substituted in order to specify an implied length or an implied base register and displacement, or both.

Reference Summary for Implied Operands

| Basic Machine Format | Explicit Base Registers and Displacement | | Implied Base Registers and Displacement | |
|---|---|---|---|---|
| | Explicit Length (1) | Implied Length (2) | Explicit Length (3) | Implied Length (4) |
| RX | D2(X2,B2) | N/A | S2(X2) | N/A |
| RS | D2(B2) | N/A | S2 | N/A |
| SI | D1(B1) | N/A | S1 | N/A |
| SS | D1(L1,B1) | D1(,B1) | S1(L1) | S1 |
| SS | D2(L2,B2) | D2(,B2) | S2(L2) | S2 |
| SS | D1(L,B1) | D1(,B1) | S1(L) | S1 |
| SS | D2(B2) | N/A | S2 | N/A |

The S1 and S2 fields are relocatable expressions or absolute expressions representing values up to 4095; all other fields are absolute expressions. Where the S1 and S2 fields are absolute expressions, base register zero is implied.

Figure 13. Implied Operand Field Summary

## Machine Instruction Mnemonics

Figure 14 contains an alphabetical listing of the mnemonics of all the machine instructions and their operand field formats. The column headings in the list are:

1. Mnemonic Code: This column contains the mnemonic operation code for the machine instruction.
2. Instruction: This column contains the name of the instruction associated with the mnemonic.
3. Operation Code: This column contains the hexadecimal equivalent of the actual machine operation code.
4. Basic Machine Format: This column contains the basic machine format of the instruction:

   RR, RS, RX, SI, or SS.

5. Operand Field Format: This column shows the explicit symbolic format of the operand field for the particular mnemonic.

Appendix D provides a table for conversion of hexadecimal operation codes to their associated mnemonic codes.

| Mnemonic Code | Instruction | Operation Code | Basic Machine Format | Operand Field Format |
|---|---|---|---|---|
| A | Add | 5A | RX | R1,D2(X2,B2) |
| AD | Add Normalized, Long | 6A | RX | R1,D2(X2,B2) |
| ADR | Add Normalized, Long | 2A | RR | R1,R2 |
| AE | Add Normalized, Short | 7A | RX | R1,D2(X2,B2) |
| AER | Add Normalized, Short | 3A | RR | R1,R2 |
| AH | Add Half-Word | 4A | RX | R1,D2(X2,B2) |
| AL | Add Logical | 5E | RX | R1,D2(X2,B2) |
| ALR | Add Logical | 1E | RR | R1,R2 |
| AP | Add Decimal | FA | SS | D1(L1,B1),D2(L2,B2) |
| AR | Add | 1A | RR | R1,R2 |
| AU | Add Unnormalized, Short | 7E | RX | R1,D2(X2,B2) |
| AUR | Add Unnormalized, Short | 3E | RR | R1,R2 |
| AW | Add Unnormalized, Long | 6E | RX | R1,D2(X2,B2) |
| AWR | Add Unnormalized, Long | 2E | RR | R1,R2 |
| BAL | Branch and Link | 45 | RX | R1,D2(X2,B2) |
| BALR | Branch and Link | 05 | RR | R1,R2 |
| BC | Branch on Condition | 47 | RX | M1,D2(X2,B2) |
| BCR | Branch on Condition | 07 | RR | M1,R2 |
| BCT | Branch on Count | 46 | RX | R1,D2(X2,B2) |
| BCTR | Branch on Count | 06 | RR | R1,R2 |
| BXH | Branch on Index High | 86 | RS | R1,R3,D2(B2) |
| BXLE | Branch on Index Low or Equal | 87 | RS | R1,R3,D2(B2) |
| C | Compare Algebraic | 59 | RX | R1,D2(X2,B2) |
| CD | Compare, Long | 69 | RX | R1,D2(X2,B2) |
| CDR | Compare, Long | 29 | RR | R1,R2 |
| CE | Compare, Short | 79 | RX | R1,D2(X2,B2) |
| CER | Compare, Short | 39 | RR | R1,R2 |
| CH | Compare Half-Word | 49 | RX | R1,D2(X2,B2) |
| CL | Compare Logical | 55 | RX | R1,D2(X2,B2) |
| CLC | Compare Logical | D5 | SS | D1(L,B1),D2(B2) |
| CLI | Compare Logical Immediate | 95 | SI | D1(B1),I2 |
| CLR | Compare Logical | 15 | RR | R1,R2 |
| CP | Compare Decimal | F9 | SS | D1(L1,B1),D2(L2,B2) |
| CR | Compare Algebraic | 19 | RR | R1,R2 |
| CVB | Convert to Binary | 4F | RX | R1,D2(X2,B2) |
| CVD | Convert to Decimal | 4E | RX | R1,D2(X2,B2) |
| D | Divide | 5D | RX | R1,D2(X2,B2) |
| DD | Divide, Long | 6D | RX | R1,D2(X2,B2) |
| DDR | Divide, Long | 2D | RR | R1,R2 |
| DE | Divide, Short | 7D | RX | R1,D2(X2,B2) |
| DER | Divide, Short | 3D | RR | R1,R2 |
| DP | Divide Decimal | FD | SS | D1(L1,B1),D2(L2,B2) |
| DR | Divide | 1D | RR | R1,R2 |
| ED | Edit | DE | SS | D1(L,B1),D2(B2) |
| EDMK | Edit and Mark | DF | SS | D1(L,B1),D2(B2) |
| EX | Execute | 44 | RX | R1,D2(X2,B2) |
| HDR | Halve, Long | 24 | RR | R1,R2 |
| HER | Halve, Short | 34 | RR | R1,R2 |
| HIO | Halt I/O | 9E | SI | D1(B1) |

Figure 14. Machine Instruction Mnemonics (Part 1 of 3)

| Mnemonic Code | Instruction | Operation Code | Basic Machine Format | Operand Field Format |
|---|---|---|---|---|
| IC | Insert Character | 43 | RX | R1,D2(X2,B2) |
| ISK | Insert Storage Key | 09 | RR | R1,R2 |
| L | Load | 58 | RX | R1,D2(X2,B2) |
| LA | Load Address | 41 | RX | R1,D2(X2,B2) |
| LCDR | Load Complement, Long | 23 | RR | R1,R2 |
| LCER | Load Complement, Short | 33 | RR | R1,R2 |
| LCR | Load Complement | 13 | RR | R1,R2 |
| LD | Load, Long | 68 | RX | R1,D2(X2,B2) |
| LDR | Load, Long | 28 | RR | R1,R2 |
| LE | Load, Short | 78 | RX | R1,D2(X2,B2) |
| LER | Load, Short | 38 | RR | R1,R2 |
| LH | Load Half-Word | 48 | RX | R1,D2(X2,B2) |
| LM | Load Multiple | 98 | RS | R1,R3,D2(B2) |
| LNDR | Load Negative, Long | 21 | RR | R1,R2 |
| LNER | Load Negative, Short | 31 | RR | R1,R2 |
| LNR | Load Negative | 11 | RR | R1,R2 |
| LPDR | Load Positive, Long | 20 | RR | R1,R2 |
| LPER | Load Positive, Short | 30 | RR | R1,R2 |
| LPR | Load Positive | 10 | RR | R1,R2 |
| LPSW | Load PSW | 82 | SI | D1(B1) |
| LR | Load | 18 | RR | R1,R2 |
| LTDR | Load and Test, Long | 22 | RR | R1,R2 |
| LTER | Load and Test, Short | 32 | RR | R1,R2 |
| LTR | Load and Test | 12 | RR | R1,R2 |
| M | Multiply | 5C | RX | R1,D2(X2,B2) |
| MD | Multiply, Long | 6C | RX | R1,D2(X2,B2) |
| MDR | Multiply, Long | 2C | RR | R1,R2 |
| ME | Multiply, Short | 7C | RX | R1,D2(X2,B2) |
| MER | Multiply, Short | 3C | RR | R1,R2 |
| MH | Multiply Half-Word | 4C | RX | R1,D2(X2,B2) |
| MP | Multiply Decimal | FC | SS | D1(L1,B1),D2(L2,B2) |
| MR | Multiply | 1C | RR | R1,R2 |
| MVC | Move Characters | D2 | SS | D1(L,B1),D2(B2) |
| MVI | Move Immediate | 92 | SI | D1(B1),I2 |
| MVN | Move Numerics | D1 | SS | D1(L,B1),D2(B2) |
| MVO | Move with Offset | F1 | SS | D1(L1,B1),D2(L2,B2) |
| MVZ | Move Zones | D3 | SS | D1(L,B1),D2(B2) |
| N | AND Logical | 54 | RX | R1,D2(X2,B2) |
| NC | AND Logical | D4 | SS | D1(L,B1),D2(B2) |
| NI | AND Logical Immediate | 94 | SI | D1(B1),I2 |
| NR | AND Logical | 14 | RR | R1,R2 |
| O | OR Logical | 56 | RX | R1,D2(X2,B2) |
| OC | OR Logical | D6 | SS | D1(L,B1),D2(B2) |
| OI | OR Logical Immediate | 96 | SI | D1(B1),I2 |
| OR | OR Logical | 16 | RR | R1,R2 |
| PACK | Pack | F2 | SS | D1(L1,B1),D2(L2,B2) |
| RDD | Read Direct | 85 | SI | D1(B1),I2 |

Figure 14. Machine Instruction Mnemonics (Part 2 of 3)

| Mnemonic Code | Instruction | Operation Code | Basic Machine Format | Operand Field Format |
|---|---|---|---|---|
| S | Subtract | 5B | RX | R1,D2(X2,B2) |
| SD | Subtract Normalized, Long | 6B | RX | R1,D2(X2,B2) |
| SDR | Subtract Normalized, Long | 2B | RR | R1,R2 |
| SE | Subtract Normalized, Short | 7B | RX | R1,D2(X2,B2) |
| SER | Subtract Normalized, Short | 3B | RR | R1,R2 |
| SH | Subtract Half-Word | 4B | RX | R1,D2(X2,B2) |
| SIO | Start I/O | 9C | SI | D1(B1) |
| SL | Subtract Logical | 5F | RX | R1,D2(X2,B2) |
| SLA | Shift Left Single Algebraic | 8B | RS | R1,D2(B2) |
| SLDA | Shift Left Double Algebraic | 8F | RS | R1,D2(B2) |
| SLDL | Shift Left Double Logical | 8D | RS | R1,D2(B2) |
| SLL | Shift Left Single Logical | 89 | RS | R1,D2(B2) |
| SLR | Subtract Logical | 1F | RR | R1,R2 |
| SP | Subtract Decimal | FB | SS | D1(L1,B1),D2(L2,B2) |
| SPM | Set Program Mask | 04 | RR | R1 |
| SR | Subtract | 1B | RR | R1,R2 |
| SRA | Shift Right Single Algebraic | 8A | RS | R1,D2(B2) |
| SRDA | Shift Right Double Algebraic | 8E | RS | R1,D2(B2) |
| SRDL | Shift Right Double Logical | 8C | RS | R1,D2(B2) |
| SRL | Shift Right Single Logical | 88 | RS | R1,D2(B2) |
| SSK | Set Storage Key | 08 | RR | R1,R2 |
| SSM | Set System Mask | 80 | SI | D1(B1) |
| ST | Store | 50 | RX | R1,D2(X2,B2) |
| STC | Store Character | 42 | RX | R1,D2(X2,B2) |
| STD | Store Long | 60 | RX | R1,D2(X2,B2) |
| STE | Store Short | 70 | RX | R1,D2(X2,B2) |
| STH | Store Half-Word | 40 | RX | R1,D2(X2,B2) |
| STM | Store Multiple | 90 | RS | R1,R3,D2(B2) |
| SU | Subtract Unnormalized, Short | 7F | RX | R1,D2(X2,B2) |
| SUR | Subtract Unnormalized, Short | 3F | RR | R1,R2 |
| SVC | Supervisor Call | 0A | RR | I |
| SW | Subtract Unnormalized, Long | 6F | RX | R1,D2(X2,B2) |
| SWR | Subtract Unnormalized, Long | 2F | RR | R1,R2 |
| | | | | |
| TCH | Test Channel | 9F | SI | D1(B1) |
| TIO | Test I/O | 9D | SI | D1(B1) |
| TM | Test Under Mask | 91 | SI | D1(B1),I2 |
| TR | Translate | DC | SS | D1(L,B1),D2(B2) |
| TRT | Translate and Test | DD | SS | D1(L,B1),D2(B2) |
| TS | Test and Set | 93 | SI | D1(B1) |
| | | | | |
| UNPK | Unpack | F3 | SS | D1(L1,B1),D2(L2,B2) |
| | | | | |
| WRD | Write Direct | 84 | SI | D1(B1),I2 |
| | | | | |
| X | Exclusive OR | 57 | RX | R1,D2(X2,B2) |
| XC | Exclusive OR | D7 | SS | D1(L,B1),D2(B2) |
| XI | Exclusive OR, Immediate | 97 | SI | D1(B1),I2 |
| XR | Exclusive Logical OR | 17 | RR | R1,R2 |
| | | | | |
| ZAP | Zero and Add Decimal | F8 | SS | D1(L1,B1),D2(L2,B2) |

Figure 14. Machine Instruction Mnemonics (Part 3 of 3)

## Machine Instruction Examples

The following examples are grouped according to machine instruction format. They illustrate the various symbolic operand formats. All symbols employed in the examples must be assumed to be defined elsewhere in the same assembly. All symbols that specify register numbers and lengths must be assumed to be equated elsewhere to absolute values.

Implied addressing (shown in the following examples) requires the use of the USING assembler instruction described later in the publication.

### RR Format

| Name | Operation | Operand |
|---|---|---|
| ALPHA1 | LR | 1,2 |
| ALPHA2 | LR | REG1,REG2 |
| BETA | SPM | 15 |
| GAMMA1 | SVC | 250 |
| GAMMA2 | SVC | TEN |

The operands of ALPHA1, BETA, and GAMMA1 are decimal self-defining values, which are categorized as absolute expressions. The operands of ALPHA2 and GAMMA2 are symbols that are equated elsewhere to absolute values.

### RX Format

| Name | Operation | Operand |
|---|---|---|
| ALPHA1 | L | 1,39(4,10) |
| ALPHA2 | L | REG1,39(4,TEN) |
| BETA1 | L | 2,ZETA(4) |
| BETA2 | L | REG2,ZETA(REG4) |
| GAMMA1 | L | 2,ZETA |
| GAMMA2 | L | REG2,ZETA |

Both ALPHA instructions specify explicit addresses; REG1 and TEN are absolute symbols. Both BETA instructions specify implicit addresses and use index registers. Indexing is omitted from the GAMMA instructions. GAMMA1 and GAMMA2 specify implicit addresses.

### RS Format

| Name | Operation | Operand |
|---|---|---|
| ALPHA1 | BXH | 1,2,20(14) |
| ALPHA2 | BXH | REG1,REG2,20(REGE) |
| ALPHA3 | BXH | REG1,REG2,ZETA |
| BETA1 | SLL | 1,20(9) |
| BETA2 | SLL | REG1,20(9) |
| BETA3 | SLL | REG1,ZETA |

ALPHA1 and ALPHA2 specify explicit addresses, and ALPHA3 specifies an implicit address. Similarly, the BETA instructions illustrate both explicit and implicit addresses.

### SI Format

| Name | Operation | Operand |
|---|---|---|
| ALPHA1 | CLI | 40(9),X'40' |
| ALPHA2 | CLI | 40(REG9),TEN |
| BETA1 | CLI | ZETA,TEN |
| BETA2 | CLI | ZETA,C'A' |
| GAMMA1 | SIO | 40(9) |
| GAMMA2 | SIO | 0(9) |
| GAMMA3 | SIO | 40(0) |
| GAMMA4 | SIO | ZETA |

The ALPHA instructions and GAMMA1 through GAMMA3 specify explicit addresses; the BETA instructions and GAMMA4 specify implicit addresses. GAMMA2 specifies a displacement of zero. GAMMA3 does not specify a base register.

### SS Format

| Name | Operation | Operand |
|---|---|---|
| ALPHA1 | AP | 40(9,8),30(6,7) |
| ALPHA2 | AP | 40(NINE,REG8),30(REG6,7) |
| ALPHA3 | AP | FIELD2,FIELD1 |
| ALPHA4 | AP | FIELD2(9),FIELD1(6) |
| BETA | AP | FIELD2(9),FIELD1 |
| GAMMA1 | MVC | 40(9,8),30(7) |
| GAMMA2 | MVC | 40(NINE,REG8),DEC(7) |
| GAMMA3 | MVC | FIELD2,FIELD1 |
| GAMMA4 | MVC | FIELD2(9),FIELD1 |

ALPHA1, ALPHA2, GAMMA1, and GAMMA2 specify explicit lengths and addresses. ALPHA3 and GAMMA3 specify both implied length and implied addresses. ALPHA4 and GAMMA4 specify explicit length and implied addresses. BETA specifies an explicit length for FIELD2 and an implicit length for FIELD1; both addresses are implied.

## ASSEMBLER INSTRUCTIONS

Just as machine instructions are used to request the machine to perform a sequence of operations, assembler instructions are requests to the assembler to perform certain operations. There are 15 such assembler instructions. Some have been briefly mentioned in the preceding sections. All the assembler instructions are listed below by mnemonic operation code and name and are fully described in the subsequent text. Figure 21 at the end of this section contains a summary description of all assembler instructions.

Assembler Control Instructions

| | |
|---|---|
| ICTL | Input Control |
| START | Start Program |
| ORG | Reset Location Counter |
| CNOP | Conditional No Operation |
| END | End Program |
| EJECT | Start New Page |
| SPACE | Space Listing |

Definition Instructions

| | |
|---|---|
| EQU | Equate Symbol |
| DS | Define Storage |
| CCW | Define Channel Command Word |
| DC | Define Constant |

Base Register Instructions

| | |
|---|---|
| USING | Use Base Address Register |
| DROP | Drop Register |

Program Linking Instructions

| | |
|---|---|
| ENTRY | Identify Entry-Point Symbol |
| EXTRN | Identify External Symbol |

Assembler instruction statements, in contrast to machine instruction statements, do not always cause actual machine instructions to be included in the object program. Some (e.g., DS, DC) generate no instructions but cause storage areas to be set aside for constants and other data. Others (e.g., EQU, SPACE) are effective only at assembly time; they generate nothing in the object program and have no effect on the Location Counter.

### Assembler Control Instructions

The assembler control instructions are used to specify the beginning and end of an assembly, set the Location Counter to a value or word boundary, control the program listing, and indicate the statement format. Except for the CNOP instruction, none of these assembler instructions generate instructions or constants in the object program.

ICTL - Input Control: The ICTL instruction tells the assembler in which card column the statement portion of the source-program cards begin. The mnemonic operation code of the ICTL statement must start in column 26 or higher. The format of the ICTL instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | ICTL | The decimal value 1 or 25 |

If the statements are to begin in column 25, the format is:

```
        ICTL      25
```

If the statements begin in column 1, the format is:

```
        ICTL      1
```

If the ICTL statement is not used, or the operand field does not contain a 1 or 25, column 1 is used for the tape option and column 25 for the card option. When the ICTL statement is used, it must be the first statement in the source program. If it appears anywhere else, it will not be used. If a name is present, the name will not be used.

START - Start Program: The START instruction may be used to indicate the beginning of an assembly, give a name to the program, and set the Location Counter to an initial value. The format of the START instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| A symbol (optional) | START | A self-defining value or blank |

The symbol in the name field becomes the name of the program. The symbol is assigned the address corresponding to the self-defining value in the operand field. This symbol can be specified as an external symbol (using the EXTRN instruction) in other programs, without using the ENTRY instruction to identify it as an entry point in this program. If there is no symbol in the name field, the assembler assigns a name consisting of six blanks.

A self-defining value that specifies the initial setting of the Location Counter is written in the operand field. If the value of the operand is not a multiple of eight, the Location Counter is set at the next double-word boundary. The self-defining value must not exceed the maximum allowable setting of the Location Counter. If the operand field is invalid or blank, the Location Counter is set to zero.

The initial setting of the Location Counter becomes the starting location of the program. This location is the initial loading location if the program is loaded by the absolute loader. It can also be used as the temporary starting location for loading the program while it is being tested. This enables the programmer to match the locations shown in the listing produced by the assembler with the locations in storage print listings. When the program has been checked out, it can then be relocated elsewhere by the relocating loader.

If both the START and ICTL instructions are used, the START instruction must immediately follow the ICTL instruction. If the START instruction appears anywhere else, or if it is not used, the assembler sets the Location Counter initially to zero and gives the program a name of six blanks. Any invalid occurrences of a START instruction will not be used. It should be noted that if the ICTL instruction is not used, the START instruction should be the first in the program.

Either of the START statements below could be used to assign the name PROG2 to the program and set the Location Counter to a value of 7F8:

```
PROG2    START   2040
PROG2    START   X'7F8'
```

<u>ORG - Reset Location Counter:</u> The ORG instruction resets the Location Counter to a relative value. This instruction may be used anywhere in the program, as often as desired. The format of the ORG instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | ORG | A relocatable expression |

The Location Counter is reset to the value of the relocatable expression. An ORG instruction that resets the Location Counter below its initial value as specified in the START instruction will not be used; it will, however, be printed in the listing with an error flag. Any symbol(s) in the expression must be previously defined. If the operand field is blank or invalid, the ORG instruction will not be used. If a name is specified, the name will not be used.

The statement:

```
         ORG     *+500
```

increases the Location Counter by 500 above its current setting. Nothing is assembled for the 500 bytes skipped. That is, these bytes are not cleared by the assembler. (These bytes should, therefore, not be assumed to be set to zero.)

The ORG instruction provides an alternate way of reserving storage areas; the preferred way is with the DS (Define Storage) assembler instruction. However, where a storage area cannot be conveniently defined with the DS instruction, the ORG instruction can be used. For example, to reserve two storage areas of equal size, the following coding might be used:

```
TABLE1   DS      50F
         DS      100H
         .
         .
         .
TABLE2   EQU     *
         ORG     *+TABLE2-TABLE1
```

Note that the EQU assembler instruction permits TABLE2 to be used in the ORG statement as a previously defined symbol.

<u>CNOP - Conditional No Operation:</u> The CNOP instruction allows the programmer to align an instruction at a specific word boundary. If any bytes must be skipped to properly align the instruction, the assembler ensures an unbroken flow by generating a CNOP instruction. This facility is useful in creating calling sequences consisting of a linkage to a subroutine followed by parameters such as Channel Command Words (CCW) which require proper word boundaries.

The CNOP instruction aligns the Location Counter setting to half-word, full-word, or double-word boundary. If the Location Counter is already aligned, the CNOP instruction has no effect. If the specified alignment requires the Location Counter to be incremented, a no-operation instruction (an RR branch-on-condition instruction with a zero R1 and R2 field) is generated for each pair of bytes (half-words) skipped. If an odd number of bytes is skipped, the first byte is set to zero.

The format of the CNOP instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | CNOP | Two decimal values of the form: b, w |

Operand b specifies the byte in a word or double-word at which the Location Counter is to be set; b can be 0, 2, 4, or 6. Operand w specifies whether the byte b is in a word (4) or double-word (8).

The following pairs of b and w values are valid:

| b,w | Explanation |
|---|---|
| 0,4 | Beginning of a word |
| 2,4 | Middle of a word |
| 0,8 | Beginning of a double-word |
| 2,8 | Second half-word of a double-word |
| 4,8 | Middle (third half-word) of a double-word |
| 6,8 | Fourth half-word of a double-word |

Figure 15 shows the position in a double-word that each of these pairs specifies. Note that 0,4 and 2,4 specify two locations in a double-word.

If the operand field is blank or invalid, the CNOP instruction will not be used. A name, if present, will not be used.

Assume that the Location Counter is currently aligned at a double-word boundary. Then the CNOP instruction in this sequence:

```
CNOP    0,8
BALR    2,14
```

has no effect; it is printed in the program listing. This sequence, however:

```
CNOP    6,8
BALR    2,14
```

causes three branch-on-condition instructions (no operations) to be generated, thus aligning the BALR instruction at the last half-word in a double-word:

```
BCR     0,0
BCR     0,0
BCR     0,0
BALR    2,14
```

After the BALR instruction is generated, the Location Counter is at a double-word boundary.

**END - End Program:** The END instruction terminates the assembly of a program. It may also supply a point in the program to which control is transferred after the program is loaded.

The END instruction must always be the last statement in the source program. When the assembler detects this statement, it produces a Load End card in the programmer's object program for use by the load program.

The format of the END instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | END | A relocatable expression or blank |

The expression in the operand field specifies the point to which control is transferred when loading is complete. The

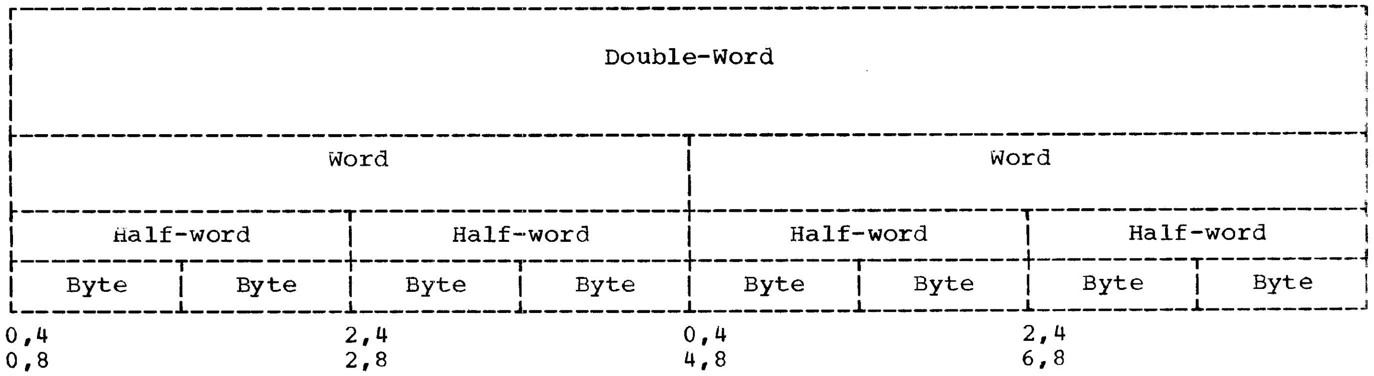


Figure 15. Boundary Alignment with a CNOP Instruction

value of the expression will be punched in the Load End card. If the operand field is blank or invalid, nothing will be punched in the Load End card. In this case, control will be passed to the first storage location (above decimal location 128) occupied by the user's program when the program is loaded. If the operand field is invalid, the statement will be flagged as a possible error. If a name is present, it will not be used.

The point to which control usually is transferred is the first machine instruction in the program, as shown in this sequence:

```
              START    2000
    AREA      DS       50F
    BEGIN     SR       3,3
              .
              .
              .
              END      BEGIN
```

EJECT - Start New Page: The EJECT instruction causes the next line of the listing to appear at the top of a new page. This instruction provides a convenient way to separate routines in the program listing. The format of the EJECT instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | EJECT | Not used |

Normally, the EJECT statement is not included in the program listing; however, anything appearing in the name or operand fields will result in including the statement in the listing. In this case, the EJECT statement is printed prior to skipping to the new page.

SPACE - Space Listing: The SPACE instruction is used to insert one or more blank lines in the listing. The format of the SPACE instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | SPACE | A decimal value |

A decimal value is used to specify the number of blank lines to be inserted in the program listing. If this value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement. A blank operand field will cause one line to be skipped. Normally, the SPACE statement is not included in the program listing. There are, however, some exceptions. Anything in the name field of a SPACE statement results in including the statement in the listing. In this case, the statement is printed prior to spacing. If the operand field is invalid (that is, not a decimal value or one greater than 4095), the statement is flagged and listed. No space operation occurs.

## Definition Instructions

The definition assembler instructions are used to define and enter constant data into a program, specify the contents of Channel Command Words, and reserve areas of core storage. The fields generated by these instructions can be referred to by symbolic names. The EQU instruction is included with the definition instructions because it is used for defining symbols.

EQU - Equate Symbol: The EQU instruction is used to define a symbol by assigning to it the value and length attributes of an expression in the operand field. The format of the EQU instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| A symbol | EQU | An expression |

The symbol in the name field is given the same value attribute as the expression. The length attribute of the symbol will be that of the leftmost term of the expression. If the term is an asterisk (the Location Counter) or a self-defining value, the implied length of the symbol is one. The expression in the operand field can be relocatable or absolute, and the symbol will be similarly defined. Any symbols in the expression must be previously defined and have a positive value. Symbols not conforming to these rules will not be used. The associated EQU statements will be flagged.

If the expression in the operand field or the symbol in the name field, or both, are invalid or not present, the EQU statement will be flagged in the listing and will not be used.

The EQU instruction is the usual way of equating symbols to register numbers, input/output unit numbers, immediate data, actual addresses, and other arbitrary values. The examples below illustrate how this might be done:

```
REG2     EQU     2         General register
IO125    EQU     125       Input/output unit
TEST     EQU     X'3F'     Immediate data
TIMER    EQU     80        Actual address
```

Note: Any time the value 2 is needed in any operand, REG2 may be used. It is not restricted to use in defining register 2.

To reduce programming time, the programmer can equate symbols to frequently used compound expressions and then use the symbols as operands in place of the expressions. Thus, in the statement

```
   FIELD    EQU     ALPHA-BETA+GAMMA
```

FIELD will be defined as ALPHA-BETA+GAMMA and may be used in place of it. Note, however, that ALPHA, BETA, and GAMMA must all be previously defined.

DS - Define Storage: The DS instruction is used to reserve storage areas and to assign names to the areas. This instruction is the preferred way of symbolically defining storage for work areas, input/output areas, etc. The format of the DS instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| A symbol (optional) | DS | An operand describing the area to be reserved, in the form shown below |

The single operand specifies the number, type, and, if desired, the length of the fields to be reserved. The general form of the operand is:

dtLn

Where:

d
: is a decimal number that specifies the number of fields (from 0 to 65,535) to be reserved. It is called the duplication factor. If it is omitted, one field will be reserved.

t
: is the type code specifying the type of field to be reserved and can be one of the following letters:

| Code | Field Type | Implied Length (in Bytes) |
|---|---|---|
| C | Character (byte) | 1 |
| H | Half-word | 2 |
| F | Full-word | 4 |
| D | Double-word | 8 |

Ln
: can be used only if the field code is C. Ln is the length code written as the letter L immediately followed by n, which is the length (in bytes) of each field. n can be a decimal value that is not 0 or greater than 256.

Half-word, full-word, and double-word fields will be aligned to their proper boundaries. With a duplication factor (d) of zero, the DS instruction can be used to cause boundary alignment. Thus, the statement:

```
            DS       0D
```

sets the Location Counter at the next double-word boundary.

If there is a symbol in the name field, it is assigned the current value of the Location Counter after any word alignment. The length attribute of the symbol is the implied length associated with the field code. If a length code (Ln) is specified, the length attribute is the same as the length n.

For example, to define four 10-byte fields and one 100-byte field, the respective DS statements might be:

```
   FIELD    DS       4CL10
   AREA     DS       CL100
```

Then, to move the first 10 bytes at AREA into FIELD, the coding is as follows, assuming implied base registers and displacements:

```
   MVC      FIELD,AREA
```

Note that the length attribute of FIELD, which is 10, is implied. Explicit length specification can be used to move the first 20 bytes at AREA into FIELD. The following instruction illustrates this:

```
   MVC      FIELD(20),AREA
```

Additional examples of DS statements are shown below. The implied length attribute of each symbol appears in parentheses before the symbol:

```
(80) DONE    DS   CL80  One 80-byte field
(1)  DTWO    DS   80C   80 one-byte fields
(4)  DTHREE  DS   6F    Six full-words
(8)  DFOUR   DS   D     One double-word
(2)  DFIVE   DS   4H    Four half-words
```

If the operand is incorrectly specified, the statement is not used, and an error flag appears in the listing.

A DS statement causes the reserved area to be skipped but not cleared. Therefore, the programmer should not assume that the area contains all zeros when the program is loaded. Whenever the assembler processes a DS statement, it terminates the current output card (called a text card) in the object deck and starts the next card at the location following the reserved areas, thus skipping them. To minimize the number of text cards punched, DS statements should be kept together as much as possible. Note however, that text cards are not terminated if no bytes are skipped by DS statements used only for boundary alignment.

<u>CCW - Define Channel Command Word:</u> The CCW instruction provides a convenient way to define and generate an eight-byte Channel Command Word aligned at a double-word boundary. The internal machine format of a Channel Command Word is shown in Figure 16. The format of a CCW instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| A symbol (optional) | CCW | Four operands, separated by commas, specifying the contents of the Channel Command Word in the form described below |

The four operands, from left to right, are:

1. A <u>simple absolute</u> expression specifying the channel command code. The value of this expression is right-justified in byte 1.

2. A <u>relocatable</u> expression specifying the data address. The value of this expression is right-justified in bytes 2-4.

3. A <u>simple absolute</u> expression specifying the flags in bits 32-36 and zeros in bits 37-39. The value of this expression is right-justified in byte 5. Byte 6 is set automatically to all zeros.

4. A <u>simple absolute</u> expression specifying the count. The value of this expression is right-justified in bytes 7-8.

The following is an example of a CCW statement:

```
CCW      X'0F',READIN,X'A8',80
```

Note that the form of the third operand sets bits 37-39 to zero, as required. The bit pattern of this operand is:

```
32      36       40       44
 1010    1000     0000     0000
```

No operand field may be omitted. Operands not used must be written as zeros. An error in the operand field causes eight bytes of zeros, aligned at a double-word boundary, to be assembled.

If there is a symbol in the name field, it is assigned the value of the leftmost byte of the Channel Command Word after any boundary alignment. The length attribute of the symbol is eight. Bytes skipped because of alignment are assembled as zeros.

| Byte | Bits | Usage |
|---|---|---|
| 1 | 0-7 | Command code |
| 2-4 | 8-31 | Data address |
| 5 | 32-36 | Flags |
|  | 37-39 | Must be zero |
| 6 | 40-47 | Assembled automatically as all zeros |
| 7-8 | 48-63 | Count |

Figure 16. Channel Command Word

<u>DC - Define Constant:</u> The DC instruction is used to generate constant data in main storage. Data can be specified as characters, hexadecimal numbers, decimal numbers, and storage addresses. Decimal numbers may be in the form suitable for both fixed-point and floating-point arithmetic operations. The format of the DC instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| A symbol (optional) | DC | A single operand describing the constant, written in the form shown below |

The operand specifies the type of constant and the constant itself. It may also specify an explicit storage length for the constant and indicate how many times the constant is to be duplicated in storage. The format of this operand varies with the constant type. The basic format is either

*dtLn'c'* or AL*n*(*c*)

where:

*d*
: is a decimal number (from 1 to 65,535) that specifies the number of identical constants to be generated. It is called the duplication factor. If it is omitted, one constant is produced. A duplication factor cannot be specified for an expression (type A) constant.

*Note:* A print line is produced for each constant generated. Thus, assembler speed can be increased by keeping duplication factors small and length codes large.

*t*
: is the type code, specifying the type of constant. It can be one of the following letters:

| Code | Constant Type | Machine Format |
|---|---|---|
| C | character | 8-bit BCD code. |
| X | hexadecimal | Fixed-point binary. |
| F | decimal | Full-word fixed-point binary. |
| E | decimal | Short-precision floating-point binary. |
| H | decimal | Half-word fixed-point binary. |
| D | decimal | Long-precision floating-point binary. |
| A | relocatable or absolute expression | Fixed-point binary. |

*Ln*
: is the length code written as the letter L followed by *n*, a decimal value, which is the explicit length (in bytes) of the constant. A length code is not applicable with constant types H, E, and D. If a length code is not given, the implied lengths shown in Figure 17 will be used. An explicit length must not exceed those values shown in Figure 17.

*'c'*
: is the constant itself enclosed in single quotation marks. Note that for constant type A, the expression specifying the constant is enclosed in parentheses (*c*).

If the operand is invalid, the statement is not used but is flagged in the listing.

All constant types except character (C) and hexadecimal (X) are aligned at appropriate boundaries. Constants are not aligned if an explicit length is given. The boundaries for the various constant types are summarized in Figure 17. Any bytes skipped for alignment are set to zero.

A symbol in the name field is given the address value of the first byte assigned after any alignment. The length attribute of the symbol is the implied (or explicit) length of the constant before the duplication factor is applied.

The implied or explicit length of a constant defined by a single DC statement must not exceed 16 bytes *before* the duplication factor is applied. If longer constants are required, successive DC statements should be used. The total storage alloted to a constant defined by one DC statement is the duplication factor times the length of the constant.

The subsequent text, with examples, describes each of the constant types. This material is summarized in Figure 17. Note that the definition of character, hexadecimal, and decimal constants is not limited by the rules pertaining to self-defining values.

*Character Constants (C) :* A character constant may not be more than 16 valid characters. A valid character is a blank or any combination of punches in a card column that translates into the IBM 8-bit EBCDIC. There are 256 such combinations; the table in Appendix A lists the combinations, their eight-bit codes, and, where applicable, their printer graphics.

Each character in the constant is translated into one byte. Boundary

Reference Summary for DC Statements

| Constant-type Code | Boundary Alignment (If length is implied) | Length (in Bytes) | | Duplication Allowed | Truncation/ Padding Side |
|---|---|---|---|---|---|
| | | Implied | Maximum Explicit | | |
| C | none | variable* | 16 | yes | right |
| X | none | variable* | 16 | yes | left |
| F | word | 4 | 4 | yes | left |
| H | half-word | 2 | invalid | yes | left |
| E | word | 4 | invalid | yes | none |
| D | double-word | 8 | invalid | yes | none |
| A | word | 4 | 4 | no | left |

* But not exceeding 16 bytes

Figure 17. DC Statement Summary

alignment is not performed. The number of bytes required for the constant becomes its implied length unless an explicit length is stated. In the following example, the length attribute of FIELD is 11:

```
FIELD    DC      C'TOTAL IS 11'
```

A single quotation mark used as a character is represented in the constant by two single quotation marks. The same rule applies to ampersands. Thus:

```
         DC      C'DON''T'
         DC      C'A,B&&C'
```

Five bytes are used for each constant.

If the size of the constant exceeds the explicit length, the excess rightmost characters are truncated before applying the duplication factor when either more than 16 characters are specified or a length code is given. The statement is then flagged. For example, the statement:

```
         DC      3CL4'ABCDE'
```

generates:

```
         ABCDABCDABCD
```

If the number of characters is fewer than the explicit length, the constant is padded by adding the necessary right-hand blanks. The statement:

```
            DC      4CL3'NO'
```

generates in storage:

```
     NObNObNObNOb
```

*Hexadecimal Constants (X):* A hexadecimal constant may be up to 32 hexadecimal digits. The valid hexadecimal digits are:

0 1 2 3 4 5 6 7 8 9 A B C D E F

A table for converting hexadecimal to decimal is included in Appendix B. The reader also is referred to the section *Self-Defining Values*. Each hexadecimal digit represents four bits; hence, every pair of digits will be translated into one byte. Boundary alignment will not be performed. If an odd number of hexadecimal digits is present, the four leftmost bits of the leftmost byte are set to zero. Unless an explicit length is specified, the number of bytes required for the constant becomes its implied length.

An eight-digit hexadecimal constant provides a convenient way to set the bit pattern of a full binary word. The constant in the following example would set the first and third bytes of a word to ones. Note that the preceding DS statement is used to align the constant at a full-word boundary:

```
         DS      0F
TEST     DC      X'FF00FF00'
```

If more than 32 hexadecimal digits are present or a length code is specified and the byte size of the constant exceeds the explicit length, the excess leftmost digits will be truncated before the duplication factor is applied. The statement will be flagged in the listing. In the following statement, the A will be truncated and 6F4E will be used as the constant:

```
ALPHA    DC      3XL2'A6F4E'
```

The resulting constant will be generated three times:

```
6F4E6F4E6F4E
```

If the pairs of digits are fewer than the explicit length, the constant will be padded by adding zeros to the left before applying the duplication factor. Thus:

```
DC       2XL3'2DDA'
```

generates two 3-byte constants:

```
002DDA002DDA
```

Full-Word Constants (F): The signed decimal constant in the operand is converted into a binary number. An unsigned number is assumed to be positive. Negative numbers are converted to two's complement notation.

If there is no explicit length, the binary number is placed in a full-word aligned at the proper boundary. An implied length of four is assigned. If a length code is present, alignment does not occur; the binary number is right-justified in the specified number of bytes. An explicit length must not exceed four bytes.

Given the following statement:

```
CONWRD   DC       3F'+658474'
```

three full-word positive constants will be produced. The address value of CONWRD corresponds to the leftmost byte of the first word; the length attribute will be four. Thus, the expression CONWRD+4 can be used to address the second word symbolically.

The maximum permissible value of a full-word constant depends on the length, as follows:

| Length | Highest Value | Lowest Value |
|---|---|---|
| 4 | 2,147,483,647 | -2,147,483,648 |
| 3 | 8,388,607 | -8,388,608 |
| 2 | 32,767 | -32,768 |
| 1 | 127 | -128 |

Note: All lengths can be explicit. A length of 4, however, can also be implied.

If a value exceeds the limits associated with the length, a constant of zero will be generated before applying the duplication factor. The statement will be flagged in the listing. For example, the following statement would generate 12 bytes of zeros:

```
         DC       4FL3'-9500250'
```

Half-Word Constants (H): The signed decimal constant in the operand is converted into a binary number placed in a properly aligned half-word. A length code is not allowed. The implied length of the constant is two bytes.

If the number is unsigned, a positive value is assumed. Negative numbers will be converted to two's complement notation.

The allowable range of numbers is 32,767 through -32,768. If a number exceeds these limits, the constant is set to zero before the duplication factor is applied. The statement is then flagged.

The following statement generates two identical half-word positive constants, right-justified within two bytes:

```
         DC       2H'256'
```

Short-Precision Floating-Point Constants (E): A short-precision floating-point constant is specified as a decimal fraction (mantissa) and an optional decimal exponent. The maximum allowable range for a floating-point constant is from approximately $(5.4)x10^{-79}$ to $(7.2)x10^{75}$. The constant will be aligned at a full-word boundary in the proper machine format for use in floating-point operations. A duplication factor may be applied to the constant. A length code, however, may not be used.

The format of the constant portion of the operand is described in the following text.

Fraction: The fraction is a signed decimal number (up to eight digits) with or without a decimal point. The decimal point can appear before, within, or after the number. If the point is at the rightmost end of the number, it may be omitted. If the sign is omitted, a positive fraction is assumed. A negative fraction is carried in the machine in true form. The fraction, irrespective of its decimal point or sign, must not exceed $2^{24}-1$ (i.e., 16,777,215). The fraction part of a number converted to the short format will differ by no more than one from the exact value rounded to 24 places.

Exponent: The exponent is optional and may be omitted if the decimal point appears in the fraction at the desired position. If the exponent is specified, it must immediately follow the fraction. It consists of the letter E followed by a signed decimal number denoting the exponent to the base ten. A positive exponent is assumed if the sign is omitted.

A negative exponent indicates that the true decimal point is to the left of the point written (or assumed) in the fraction. A positive exponent indicates that the true decimal point is to the right. The value of the exponent determines how many places to the left or right the true decimal point is located.

For example, to convert the number 46.415 to a floating-point format, any of the following statements could be used; they all have the same effect:

```
DC      E'46.415'
DC      E'46415E-3'
DC      E'+46415.E-3'
DC      E'.46415E2'
DC      E'4.6415E+1'
```

If either the fraction or the exponent is outside the permissible range, the full word (or words, if a duplication factor is specified) will be set to zero and a flag will appear in the listing. The statement:

```
DC      4E'3.45E76'
```

would generate four full-words of zeros.

Long-Precision Floating-Point Constants (D): A long-precision floating-point constant is specified as a decimal fraction (mantissa) and an optional decimal exponent. The maximum allowable range for a floating-point constant is from approximately $(5.4)x10^{-79}$ to $(7.2)x10^{75}$. The constant will be aligned at a double-word boundary in the proper machine format for use in floating-point operations. A duplication factor may be applied to the constant. A length code, however, may not be used.

The format of the constant portion of the operand is described in the following text.

Fraction: The fraction is a signed decimal number (up to 17 digits) with or without a decimal point. The decimal point can appear before, within, or after the number. If the point is the rightmost end of the number, it may be omitted. If the sign is omitted, a positive fraction is assumed. A negative fraction is carried in the machine in true form. The fraction, irrespective of its decimal point or sign must not exceed $2^{56}-1$ (that is, 72,057,594,037,927,935). The fraction part of a number converted to the long format will differ by no more than 11 from the exact value rounded to 56 places.

Exponent: The exponent is optional and may be omitted if the decimal point appears in the fraction at the desired position. If the exponent is specified, it must immediately follow the fraction. It consists of the letter E followed by a signed decimal number denoting the exponent to the base ten. A positive exponent is assumed if the sign is omitted.

A negative exponent indicates that the true decimal point is to the left of the point written (or assumed) in the fraction. A positive exponent indicates that the true decimal point is to the right. The value of the exponent determines how many places to the left or right the true point is located.

If either the fraction or exponent is outside the permissible range, the double word (or words, if a duplication factor is specified) will be set to zero. The statement will be flagged.

The following statements illustrate different ways of converting the same number to a long-precision floating-point number:

```
DC      D'-72957'
DC      D'-729.57E+2'
DC      D'-729.57E2'
DC      D'-.72957E5'
DC      D'-7295700.E-2'
```

Expression Constants (A): An expression constant consists of a relocatable or absolute expression enclosed in parentheses instead of single quotation marks. The value of the expression is generated as a 32-bit value constant. Since the expression frequently represents a storage address, the constant generated from it is commonly called an address constant. Hence, the letter A is used as the type code. Note that if the program is relocated, all address constants generated from relocatable expressions will be changed by the relocating program loader.

An explicit length not exceeding four bytes may be specified for expression constants. However, a duplication factor is not allowed.

Unless a length code is present, the 32-bit constant will be aligned at a full-word boundary and given an implied length of four. Thus, in the following statement, the value of AREA+2 will be placed in the next available full word as a 32-bit value. ADCON1 will be given a length attribute of four:

```
ADCON1    DC      A(AREA+2)
```

If a length code is given, the constant will be right-justified in the specified number of bytes; it will not be aligned. Any excess bits to the left will be truncated. For example, in the statement:

```
ADCON2    DC      AL2(FIELD-256)
```

the rightmost 16 bits of the value of FIELD-256 will be right-justified in the next two bytes. The length attribute of ADCON2 will be two. In this case, FIELD must be equivalent to an absolute symbol (see below).

The following considerations govern type A constants:

1. A relocatable expression may be used only if the length is implied (that is, it is four), or if the explicit length is three or four.

2. An expression may have a negative value only if it is an absolute expression. A negative value is stored in two's complement notation.

3. An expression may not begin with an arithmetic operator.

## Base Register Instructions

The USING and DROP base register assembler instructions enable programmers to use expressions representing core storage locations as operands of machine instruction statements, leaving the assignment of base registers and the calculation of displacements to the assembler.

This feature of the assembler simplifies programming and eliminates a likely source of programming errors, thus reducing the time required to check out programs. To take advantage of this feature, the programmer must use the USING and DROP instructions described in this section.

USING - Use Base Address Register: The USING instruction indicates that the general register specified in the operand is available for use as a base register. This instruction also states the base-address value the assembler must assume is in the register at object time. Note that a USING instruction does not load the register specified. It is the programmer's responsibility to see that the specified base-address value is placed into the register. Suggested loading methods are described in Programming with the USING and DROP Instructions. The format of the USING instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | USING | A relocatable expression and a simple absolute expression, separated by a comma |

The relocatable expression specifies a value that the assembler can use as a base address. The second operand is a simple absolute expression specifying the general register that can be assumed to contain the base address represented by the first operand. The value of the second operand must be from 1 to 15. For example, the statement:

```
USING    *,12
```

tells the assembler it may assume that the current value of the Location Counter will be in general register 12 at object time.

If the programmer changes the value in a base register currently being used, the assembler must be told the new value by means of another USING statement. In the following sequence, ALPHA is a relocatable expression:

```
USING    ALPHA,9
         .
         .
         .
USING    ALPHA+1000,9
```

The assembler first assumes the value of ALPHA is in register 9. The second statement causes the assembler to assume ALPHA+1000 as the value in register 9.

If the value of the second operand is zero, implying no base addressing, the first operand should also have a value of zero. If it does not, zero is used instead of the actual value. The implications of using register zero as a base register are discussed later in Base Register Zero.

A USING statement is not used if either of its operands are incorrect. A flag will appear in the listing. Any symbol in the name field will not be used.

DROP - Drop Register: The DROP instruction specifies a previously available register that may no longer be used as a base register. The format of a DROP instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | DROP | A simple absolute expression |

The expression indicates a general register that previously had been named in a USING statement and is now unavailable for base addressing. The following statement, for example, removes register 11 from the list of available registers:

```
DROP      11
```

The DROP statement is ignored if the register it designates had never appeared in a USING statement. If the value of the expression exceeds 15, the statement is not used and is flagged in the listing. Any symbol in the name field may not be used.

It is not necessary to use a DROP statement when the base address in a register changes as a result of a USING statement. DROP statements are not needed at the end of the source program.

A register made unavailable by a DROP instruction can be restored to the list of available registers by a subsequent USING instruction.

Programming with the USING and DROP Instructions: The USING and DROP instructions may be used anywhere in a program, as often as needed. They provide the assembler with the necessary information for construction of a "register table." Entries in the table are added, deleted, and changed by the assembler as each USING and DROP instruction is processed.

Whenever an effective address is specified in a machine instruction statement, the assembler consults this table to determine whether there is an available register containing a suitable base address. If more than one register produces a valid displacement (that is, a displacement not exceeding 4095), the register whose contents produce the smallest displacement is used. If two or more registers produce the same displacement, the highest numbered register is used. If no register produces a valid displacement, the statement is flagged, and the instruction, except the op code, is set to zero.

The sequence of instructions in Figure 18 illustrates the assignment of base registers. Instructions that load the registers are not shown.

## Loading Registers

Several methods exist for loading general registers that will be used for base addressing. However, for a program to be relocated when it is loaded, at least one of the base registers must be loaded with a relocatable address using either of the instructions described below. The exact method of using these instructions can differ from the examples shown.

```
0000    PGMNME   START   0
                 USING   *,11
                 USING   *+4096,12
                 USING   *+8192,13
                 USING   *+4500,14
                   .
                   .
                   .
2000    ALPHA    MR      4,2
                   .
                   .
                   .
5500    BETA     SR      1,2
                   .
                   .
                   .
        B1       BC      15,ALPHA
        B2       BC      15,BETA
        B3       BC      15,GAMMA
                   .
                   .
                   .
9750    GAMMA    AR      1,2
                 DROP    11
```

B1--Although the effective address represented by ALPHA can be wholly contained in the displacement field without a base address, base register 11 is nonetheless assigned since to use base register 0 would make the program nonrelocatable (see Base Register Zero). Because the value in register 11 is zero, the displacement will be 2000.

B2--Either register 12 or 14 would produce valid displacements; register 14 is used, however, because it produces the smaller displacement, which is 1000.

B3--Only register 13 can be used as the base register; the calculated displacement is 1558.

Figure 18. Example of Coding with USING and DROP Instructions

Branch and Link (BALR or BAL) Instruction: In the following sequence, the BALR instruction loads into register 5 the address of the first storage location after the BALR instruction. The USING instruction indicates to the assembler that register 5 contains this location:

```
BALR     5,0
USING    *,5
```

When using this method, the USING instruction must immediately follow the BALR instruction.

Load Full-Word (L) Instruction: In the following coding, the value of RGLOAD is generated as a constant. RGLOAD is a symbol defined elsewhere in the program. This value, which is also specified in the USING instruction, is inserted into register 6 with the Load (L) instruction.

```
CNSTNT   DC       A(RGLOAD)
         .
         .
         .
         L        6,CNSTNT
         .
         .
         USING    RGLOAD,6
```

Note that if the symbol RGLOAD was used in the load instruction, register 6 would contain the full-word located at RGLOAD rather than the value of RGLOAD itself.

The Load instruction should precede the USING instruction to insure that the assumed contents of the register are, in fact, in the register when the program is executed. Otherwise, the assembler would use the specified register as a base register in machine instructions before the load instruction was encountered. This could lead to undesirable results when the program is executed. Observe, however, that the USING instruction need not immediately follow the load instruction, although it is recommended that the two instructions be consecutive.

If one register has been initialized by the Branch-and-Link or Load instruction, other registers may be loaded from it by other instructions. Thus, in the following example, the Load Address (LA) instruction causes 4,080 to be added to the contents of register 4 and the resulting total to be placed in register 3:

```
         BALR     4,0
         USING    HERE,4
HERE     LA       3,4080(0,4)
          .
          .
          .
         USING    HERE+4080,3
```

Note that the LA instruction could have been written alternately as LA 3,4080(4).

Base Register Zero: The specification of general register 0 as a base register indicates that a quantity of zero is to be used as the base address, regardless of the contents of general register 0. Therefore, if general register 0 is made available by a USING instruction for base addressing, the program will not be relocatable when there is no other general register available for referencing locations below location 4096. Figure 19 illustrates a program that would not be relocatable; any reference to AREA1 will require the use of register 0, since register 2 cannot produce a valid displacement. References to AREA2, however, will make use of register 2.

This restriction does not prevent a relocatable program from referring to actual storage locations by means of absolute expressions. For example, to reference a permanently allocated interrupt location at storage address 24, the following statement is perfectly correct:

```
LPSW     24
```

```
0000                 START    0
                     USING    *,0
                     USING    *+2048,2
                      .
                      .
                      .
2000       AREA1     DS       20H
                      .
                      .
                      .
4000       AREA2     DS       10F
```

Figure 19. Example of Coding Using Base Register Zero

## Program Linking Instructions

The program linking assembler instructions allow the programmer to symbolically link independently assembled programs that are loaded and executed together. Symbolic linkages between programs are created by means of symbols defined in one program and used as operands in another program. Such symbols are termed linkage symbols. A linkage symbol is called an "entry-point symbol" in the program in which it is defined; it is an "external symbol" in the program in which it is used as an operand. External and entry-point symbols are also described in the section Symbols.

Every linkage symbol must be properly identified as such in the source program. A linkage symbol used as an external symbol is identified in each using program by the EXTRN instruction. A linkage symbol used as an entry point must be identified in the defining program by the ENTRY instruction.

A program name (defined in the name field of a START statement) is also considered an entry point. A program name, however, does not have to be identified as an entry point by the ENTRY instruction.

ENTRY - Identify Entry-Point Symbol: The ENTRY instruction identifies an entry-point symbol to the program. Each such entry-point symbol (except a program name) must be identified by a separate ENTRY instruction. The format of the ENTRY instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | ENTRY | A relocatable symbol |

The relocatable symbol in the operand field is a symbol which is defined elsewhere in the program and may be used as an entry point by other programs. A symbol that is not defined in the program is flagged in the listing as an undefined symbol. Any symbol in the name field is not used.

An ENTRY statement must be immediately preceded by either the START statement or another ENTRY statement. EXTRN statements should follow ENTRY statements (if any). An ENTRY statement cannot appear in a program unless the START statement has been used.

If an ENTRY statement is incorrectly placed, or if the operand is invalid, the statement is not used. An error flag appears in the listing. Up to 100 ENTRY statements may be used. If over 100 are used, the first 100 encountered are assembled. The remainder are not assembled, but appear in the listing with error flags.

In the following sequence, SQRT is identified as an entry-point symbol. Note that the ENTRY statement appears immediately after the START statement:

```
SUBRO     START    0
          ENTRY    SQRT
          .
          .
          .
SQRT      STM      1,10,SAVE
```

EXTRN - Identify External Symbol: The EXTRN instruction identifies a linkage symbol as an external symbol that is referenced in this program. Each such external symbol must be identified by a separate EXTRN instruction. The format of the EXTRN instruction statement is:

| Name | Operation | Operand |
|---|---|---|
| Not used | EXTRN | A relocatable symbol |

The relocatable symbol in the operand field must be defined in another program and identified in that program as an entry-point symbol by either the START or ENTRY instruction. Any symbol in the name field is not used.

An EXTRN statement must be immediately preceded by either the START statement, an ENTRY statement, or another EXTRN statement. An EXTRN statement cannot appear in a program unless the START statement has been used. Not more than 14 EXTRN statements may appear in a program. If there are more than 14 statements, the symbol in each excess statement is flagged as undefined.

If an EXTRN statement is incorrectly placed, or if the operand is invalid, the statement is not used. An error flag appears in the listing.

As an example, if MTPLY is an entry-point symbol in another program, the using program identifies it as an external symbol, thus:

```
          EXTRN    MTPLY
```

The correct use of an external symbol elsewhere in a program is described below.

## Linking Conventions

The only way an external symbol may be referenced is to:

1. Identify it with the EXTRN instruction.
2. Create an address constant from the external symbol.
3. Load the constant into a general register.
4. Branch to the address via the register or use the register for base addressing.

For example, to link to a program named SINE, the following coding might be used:

```
PROGA     START   1000
          EXTRN   SINE
            .
            .
            .
          L       4,ADSINE
          BALR    15,4
            .
            .
            .
ADSINE    DC      A(SINE)
```

In this example, SINE would be given a value of zero at assembly time; four bytes of zeros would be reserved at the symbolic location ADSINE. When the programs are loaded, the relocating loader adds the effective address assigned to SINE to the four bytes of zeros.

If the programmer wished to link to a location 12 bytes past SINE, the constant could be created as follows:

```
ADSINE    DC      A(SINE+12)
```

The relocating program loader adds 12 to the effective address of SINE and places the sum in the four bytes at ADSINE. The expression in which the external symbol is used must be a relocatable expression.

Another method of linking to SINE+12 is:

```
          START   1000
          EXTRN   SINE
            .
            .
            .
          USING   SINE,4
          L       4,ADSINE
            .
            .
            .
        { BAL     15,SINE+12 }
        { BAL     15,12(0,4) }
        { BAL     15,12(4)   }
            .
            .
ADSINE    DC      A(SINE)
```

In the above sequence, either BAL instruction can be used; if BAL 15,12(0,4) or BAL 15,12(4) is used, the USING statement may be omitted, since implicit base addressing is not involved.

A return branch from the program named SINE may be made via the registers without making any reference to a linkage symbol. Thus, if the branch to SINE was:

```
          BALR    10,4
```

the return branch may be:

```
          BCR     15,10
```

Limitations on Program Linking: The order in which independently assembled programs are loaded generally determines the extent to which they can link to one another. The program(s) containing the entry point(s) must be loaded before the program(s) that will reference these points as external symbols. Note, however, that program names are not affected by this restriction. A program loaded first may refer to programs loaded after it by their names, using an Include Segment (ICS) card and the facilities of the relocating loader. (Refer to Include Segment Card.) Also, the use of relocating loader control cards can remove all restrictions on linking.

In the following situation, two independently assembled programs, Program A and Program B, are to be executed together. Each program contains the coding shown in Figure 20.

Program A

```
PROGA     START     0
          ENTRY     LOOP
          ENTRY     LINK
          EXTRN     SINE
          EXTRN     COSINE
          EXTRN     PROGB
            .
            .
            .
LOOP      ---       ---
            .
            .
            .
LINK      ---       ---
            .
            .
            .
ADSINE    DC        A(SINE)
ADCOSN    DC        A(COSINE)
ADPRGB    DC        A(PROGB)
```

Program B

```
PROGB     START     0
          ENTRY     SINE
          ENTRY     COSINE
          EXTRN     LOOP
          EXTRN     LINK
          EXTRN     PROGA
            .
            .
            .
SINE      ---       ---
            .
            .
            .
COSINE    ---       ---
            .
            .
            .
ADLOOP    DC        A(LOOP)
ADLINK    DC        A(LINK)
ADPRGA    DC        A(PROGA)
```

Figure 20. Example of Program Linking

If Program A is loaded first, it can refer to Program B only by its name, PROGB. Program B however, can refer to Program A by its name, PROGA, and its entry points, LOOP and LINK. If the loading order is reversed, then Program B can refer to Program A only by its name, whereas Program A can refer to Program B by its name and by its entry points, SINE and COSINE.

Thus, if a common data area is to be used by two independently assembled programs, the data area should be assembled separately and then loaded first to enable both programs to refer freely to it.

<u>Program Relocation and Linking:</u> Programs that are linked together at object time must be relocatable. To be relocatable, a program must:

1. Contain all information required by the relocating loader.
2. Not use absolute expressions to refer to any area that can be relocated.
3. Identify all entry-point and external symbols to be used by the ENTRY and EXTRN instructions, respectively.
4. Specify all address constants (type A constants) that represent relocatable expressions with a length of three or four.
5. Not use general register zero as a base register.

## Assembler Instruction Summary

Figure 21 contains all of the assembler instructions and the contents of their name and operand fields. Figure 22 is a Basic Assembler language programming example.

| Reference Summary for Assembler Instructions | | |
|---|---|---|
| Name Field | Mnemonic | Operand Field |
| Not used | ICTL | The decimal value 1 or 25 |
| An optional symbol | START | A self-defining value, a comma, or blank |
| Not used | ENTRY | A relocatable symbol |
| Not used | EXTRN | A relocatable symbol |
| Not used | CNOP | Two decimal values separated by a comma |
| An optional symbol | CCW | Four operands separated by commas |
| An optional symbol | DC | A single operand describing the constant |
| Not used | DROP | A simple absolute expression |
| An optional symbol | DS | A single operand describing the area to be reserved |
| Not used | EJECT | Not used |
| A required symbol | EQU | An expression |
| Not used | ORG | A relocatable expression |
| Not used | SPACE | A decimal value not exceeding 63 |
| Not used | USING | A relocatable expression and a simple absolute expression, separated by a comma |
| Not used | END | A relocatable expression, a comma, or blank |

Figure 21. Assembler Instruction Summary

This test program sorts, in ascending sequence, the 16 hexadecimal characters located at 'IN' and stores them at 'OUT'. (The following example is used to demonstrate instruction mix rather than model coding.)

```
FLAGS      LOC.CTR.      OBJECT    V1L0   SOURCE STATEMENT
                                                 ICTL   25
           000000                         SAMPLE START  0              STARTING ADDR
           000000   05 D0                 GO     BALR   13,0           SET UP BASE REGISTER
                                  000002         USING  *,13
           000002   D2 3F D 09E D 05E            MVC    OUT(64),IN     MOVE DATA TO OUT
           000008   41 60 D 09E                  LA     6,OUT          POINT TO TABLE TOP
           00000C   41 70 0 00F                  LA     7,15           SET FOR 15 PASSES
           000010   41 40 0 038           SET    LA     4,56           SET INDEX REGISTER
           000014   58 20 6 000                  L      2,0(0,6)       LOAD FROM TABLE TOP
           000018   58 34 6 004           LOAD   L      3,4(4,6)       LOAD FROM TABLE
           00001C   15 23                        CLR    2,3            COMPARE VALUES
           00001E   47 C0 D 02A                  BC     12,SUB         TOP = OR LESS BRANCH
           000022   17 23                        XR     2,3            EXCHANGE VALUES
           000024   17 32                        XR     3,2            EXCHANGE VALUES
           000026   17 23                        XR     2,3            EXCHANGE VALUES
           000028   50 34 6 004                  ST     3,4(4,6)       STORE LARGER BACK
           00002C   5B 40 D 05A           SUB    S      4,CON4         REDUCE INDEX
           000030   47 A0 D 016                  BC     10,LOAD        LOOP IF MORE TO SORT
           000034   50 20 6 000                  ST     2,0(0,6)       STORE IN TABLE TOP
           000038   5B 70 D 056                  S      7,CON1         REDUCE PASS COUNTER
           00003C   47 70 D 042                  BC     7,LOOP
           000040   82 00 D 0DE                  LPSW   ENDRUN         END OF RUN
           000044   41 66 0 004           LOOP   LA     6,4(6)
           000048   48 20 D 010                  LH     2,SET+2        MODIFY
           00004C   5B 20 D 05A                  S      2,CON4          INDEX
           000050   40 20 D 010                  STH    2,SET+2          INSTRUCTION
           000054   47 F0 D 00E                  BC     15,SET         RETURN
           000058   00000001              CON1   DC     F'1'           CONSTANT OF 1
           00005C   00000004              CON4   DC     F'4'           CONSTANT OF 4
           000060   00000005              IN     DC     X'00000005'
           000064   0000000A                     DC     X'0000000A'
           000068   00000001                     DC     X'00000001'
           00006C   00000007                     DC     X'00000007'
           000070   00000003                     DC     X'00000003'
           000074   0000000C                     DC     X'0000000C'
           000078   0000000F                     DC     X'0000000F'
           00007C   00000009                     DC     X'00000009'
           000080   0000000B                     DC     X'0000000B'
           000084   00000004                     DC     X'00000004'
           000088   00000000                     DC     X'00000000'
           00008C   0000000E                     DC     X'0000000E'
           000090   00000006                     DC     X'00000006'
           000094   0000000D                     DC     X'0000000D'
           000098   00000002                     DC     X'00000002'
           00009C   00000008                     DC     X'00000008'
           0000A0                         OUT    DS     16F            OUTPUT AND WORK AREA
                                                 CNOP   0,8  ENSURE BOUNDARY ALIGNMENT
           0000E0   0002000000000000      ENDRUN DC     X'0002000000000000'    PSW
           000000                                END    GO
```

Figure 22. Basic Assembler Language Programming Example

## THE BASIC ASSEMBLER PROGRAM

This section describes those operations of the assembler program that have a direct bearing on preparing programs for assembly. Note that the use of the Basic Assembler is described in detail in the Basic Assembler Operating Procedures section of this manual.

### ASSEMBLER PROCESSING

The assembler is a two-phase program. It is provided as two non-relocatable assembled self-loading decks of cards, one for each phase. It is also available as optional material in symbolic form for both phases. If the programmer plans to use tape for assembler residence, he must first create a tape containing the assembler in card-image form. Because of the many possible configurations, it should be understood that the descriptions in this section require the appropriate input/output devices in all cases.

#### Phase 1

During the first phase, the assembler produces a symbol table (containing symbols contained in the program) and intermediate text for use in the second phase. When tape intermediate text is used, the symbol table remains in storage and therefore is not placed on tape. When the IBM 1442-N1 or 2520-B1 Card Read-Punch is used for card intermediate text, this intermediate text is punched into the first 24 columns of each source program card. The symbol table is punched in blank cards which follow the source cards. Because the intermediate text punched into the source card is still symbolic and pertains to the statement portion of the particular card only, the source program can be reassembled without being repunched. When the IBM 2540 Card Read-Punch, or 2501 Card Reader with a 2520-B2 or B3 Card Punch is used, this intermediate text is punched into the first 24 columns of a new card along with the first 47 columns of the source statement, column 72, and columns 73-80 (the Identification-Sequence Field) (columns 73-80). The symbol table is punched in blank cards. If no errors are detected in Phase 1, a 12 punch will appear in the first column of every card containing intermediate text.

The input to the first phase consists of the Phase 1 deck of the assembler followed by the source program. If card intermediate text is used, blank cards must be available in the punch unit for the symbol table.

One card is punched for every six symbols defined in the program. The maximum number of symbols that can be defined is determined by main storage size, as explained in the section Symbol Table. If the assembler is operating on a machine with 8,192 storage bytes, approximately 50 blank cards will be sufficient to handle the maximum number of symbols allowed; for 16,384 bytes, 230 blank cards; for 24,576 bytes, 380 blank cards; for 32,768 or more bytes, 570 cards.

If tape intermediate text is used, no cards are required.

#### Phase 2

The assembler produces the program listing and object program during the second phase. The format of Phase 2 input varies with the input/output units used.

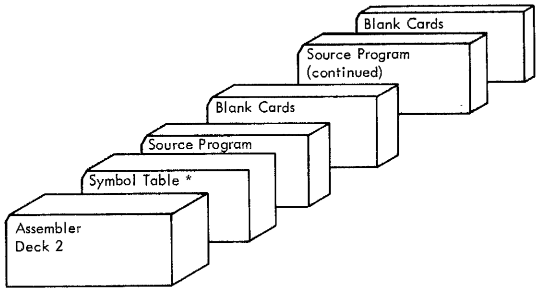
For tape intermediate text, source input is on cards and tape, or on tape entirely if the assembler residence is tape. One input consists of the Phase 2 deck of the assembler from cards or tape. The other input is the intermediate text tape created in Phase 1. If the object program is to be produced on cards, blank cards should be provided at the approximate ratio of 10 blank cards for every 100 original source program cards. If the object program is to be placed on tape, blank cards are not required.

For card intermediate text, the second deck of the assembler is loaded followed by the repunched source program when the IBM 1442-N1 or 2520-B1 Card Read-Punch is used, and by the newly punched intermediate deck when the IBM 2540 Card Read-Punch or the 2501 Card Reader with a 2520-B2 or B3 Card Punch is used. If the second phase does not immediately follow the first phase, the symbol table will not be in storage. Consequently, it is necessary to load the symbol table deck produced by Phase 1. It is placed between the assembler and source program decks. (See Figure 23.)

When the IBM 1442-N1 or 2520-B1 Card Read-Punch is used, the assembler accumulates the assembled object program in storage. When the storage area is full, and the next input card is not blank, the operator is notified to insert blank cards in the 1442-N1 or 2520-B1 Card Read-Punch for punching the object program. As each blank card is punched, it is directed to the stacker reserved for the object deck. If a blank card is encountered when none is needed, the card is directed to the stacker for the input cards. The remaining source cards are then read, and the cycle repeated.

Operator intervention may be avoided, in a 1442-N1 or 2520-B1 card system, by interleaving blank cards with the source program before starting Phase 2 (see Figure 23) at approximately the following ratios:

| Main Storage Size | Approximate Ratio of Blank Cards to Source Program Cards |
|---|---|
| 8,192 | 15 blanks every 150 source cards |
| 16,384 | 80 blanks every 800 source cards |
| 24,576 | 140 blanks every 1400 source cards |
| 32,384 | 200 blanks every 2000 source cards |
| 65,536 | 450 blanks every 4500 source cards |



*Only required when Phase 2 does not immediately follow Phase 1.

Figure 23. Phase 2 Input for Use with IBM 1442-N1 and 2520-B1 Card Read-Punch

If these ratios are observed, it should not be necessary for the operator to intervene; the time required to assemble the program will be reduced.

Blank cards may also be interleaved for Phase 1; their presence affects only the required time to read the blank cards, not this phase of the assembly.

When the IBM 2540 Card Read-Punch or 2501 Card Reader with a 2520-B2 or B3 Card Punch is used, the assembler punches an object program card as soon as one is assembled in storage.

## PROGRAM LISTING

A program listing (if requested) is produced for every assembly, provided an IBM 1443 Model N1 Printer, IBM 1403 Printer, IBM 1052 Printer-Keyboard, or IBM series 2400 tape unit is available on a Model 40 or larger system. Each statement in the source program appears on a separate line of the listing unless the suppress option is used. If the suppress option is used, only those statements containing errors are listed. The programmer obtains the suppress option by indicating to the machine operator that he does not wish a listing. More detailed information on the suppress option is contained in the description of the configuration cards in the Basic Assembler Operating Procedures section.

The program listing consists of five fields, arranged from left to right, as follows.

Flags: This field (print positions 1-10) contains, left-justified, a flag(s) to signal possible errors in the statement. Each flag is represented by a single alphabetic character. See the topic Error Notification.

Location Counter: This field (print positions 12-17) contains the Location Counter value (in hexadecimal) assigned to the statement.

Assembled Output: This field (print positions 20-39) contains the hexadecimal representation of the binary digits generated from the statement.

Source Statement: This field (print positions 40-111) contains a column-for-column reproduction of the contents of the source statement. For the card-option (using a 2540 alone or a 2501 with a 2520-B2 or B3), where statements begin in column 1, only columns 1-47 will be reproduced.

Identification-Sequence Field: This field (print positions 113-120) is a reproduction of columns 73-80 of the source card.

## ERROR NOTIFICATION

The flags produced on the program listing for various source program errors are shown in the following list. Only those errors for which flags are indicated are detected, for example, an instruction which references a storage location on an incorrect boundary is not flagged, such as:

ST 4,A

where A is not on a word boundary. Any error that causes the assembler to either ignore the instruction or assemble zeros in the operand field of the instruction will halt further evaluation of the instruction for other errors. Therefore, when correcting such an error, the programmer is advised to check for any other errors in the instruction.

| Flag | Cause |
|---|---|
| * A | Expression not simply relocatable. |
| * B | START, EXTRN, ENTRY or ICTL out of order. |
| * C | Location counter overflow. |
| * E | More than 14 EXTRNs or more than 100 ENTRYs. |
| * F | Operand field format error or self-defining value in operand field too large. |
| * G | DC, D, or E range error. |
| I | Expression can not be mapped into base and displacement. |
| * J | Symbol table full. |
| K | Relocation list dictionary buffer table full. |
| * L | Name field error. |
| * M | Multiple defined symbol. |
| * N | Statement not used. This flag is normally accompanied by other flags which define the reason the statement was not used. If it appears alone, it indicates that the statement was completely extraneous. If the flag (N) appears by itself when using a 1442-N1 or 2520-B1 card option system, it indicates that the source statement has been modified since a previous assembly but the intermediate text field (columns 1-24) has not been left blank. See section <u>Reassembly Procedure</u>. |
| * O | Invalid OP code. |
| R | Expression not absolute. |
| * S | Specification error. |
| * T | Value too large. |
| U | Undefined symbol. |
| * V | ORG or EQU symbol not previously defined. |
| W | Unused mask bits (37-39) in CCW not zero. |
| X | Duplicate entry statement. |
| * Y | Negative expression. |
| * Z | Column 72 not blank. |

<u>Note:</u> The * indicates those flags which may be punched in the intermediate text cards produced by Phase 1 in card-option systems. For systems unable to produce program listings, these flags provide a limited form of error notification. It should be noted that the intermediate text cards produced by Phase 1 contain an A, B, or C in column 1 if they are error free. Cards in error have a J, K, L, or M in column 1. Error flags are located in columns 23-24 on cards with a J or K in column 1. The error flags appear in columns 21-24 on cards beginning with L or M.

## OBJECT PROGRAM OUTPUT

The object program is generated by the assembler as a deck of cards or card images on tape acceptable as input to the loaders. If the object program is placed on tape, an LDT record follows the last program. It is the programmer's responsibility to inform the operator about the medium (cards or tape) on which the object deck is to be placed. Detailed information on this option can be found in the <u>Basic Assembler Operating Procedures</u> section of this manual. Four types of cards constitute the object program deck. It should be noted that detailed descriptions of each of the four types of cards may be found in the <u>Basic Utility Programs</u> section. General descriptions of each follow.

### External Symbol Dictionary (ESD) Card

An ESD card is generated for each START, ENTRY, and EXTRN statement. The ESD card contains coded information that is used by the relocating loader.

### Text (TXT) Card

The Text cards contain the output assembled from the source program. Up to 56 contiguous bytes of output are punched into each Text card. Each Text card also contains the storage address at which the first byte in the card is to be loaded.

### Relocation List Dictionary (RLD) Card

The purpose of RLD cards is to indicate to the relocating loader those address constants that have to be changed if the program is loaded at a location different from its assembled location. Address constants of this type are defined in the source program by (1) relocatable expressions in type A DC statements and (2) relocatable expressions specifying data addresses in CCW statements; that is, the second operands of CCW statements. Up to 13 address constants are punched into each RLD card.

The maximum number of address constants as described above, that can be defined in a program is determined by the size of main storage thus:

| Main Storage Size (in Bytes) | Maximum Number of Address Constants |
|---|---|
| 8,192 | 30 |
| 16,384 | 60 |
| 24,576 | 90 |
| 32,768 | 120 |
| 65,536 | 240 |

### Load End Card

This card is produced when the assembler encounters the END statement. The Load End card also contains the address to which control is to be transferred when the program has been loaded, if one was specified in the END statement.

## PATCHING OBJECT PROGRAMS

The programmer may modify his object program at execution time through the use of a Replace card. This card is completely described in the Basic Utility Programs section.

## REASSEMBLY PROCEDURE

A special reassembly procedure is provided for assemblies using the IBM 1442-N1 Card Read-Punch without tape. This procedure enables a partially or completely assembled program to be reassembled in less time than a new assembly would require. (Refer to the Special Procedures section of this manual.)

The program that is to use the reassembly procedure may be changed in any manner. New symbols can be added and existing ones redefined, provided that the symbol table is not full. New statements also can be included in the program.

The reassembly procedure is faster than the new assembly procedure because the assembler does not have to repunch the first 24 columns of those source program cards whose statements have not been changed. Hence, the fewer the changes, the faster the assembly.

The input to the first phase of a reassembly consists of the first deck of the assembler, followed in order by the previously punched symbol table decks, the source program with any changes, and the necessary number of blank cards into which a new symbol table is punched. Note that any changed source program cards must be repunched, leaving columns 1-24 blank. This also applies to source program cards that did not have a 12-punch in column 1 as the result of the previous assembly.

The Phase 2 input and output of a reassembly is identical with the second phase of a new assembly (see Phase 2).

## SYMBOL TABLE

For every program assembled, a symbol table composed of the symbols in that program is created. Each entry in the table records the attributes and other pertinent information about a particular symbol.

The maximum size of the symbol table and, hence, the maximum number of symbols that can be defined in a program is determined by the size of main storage, thus:

| Main Storage Size (in Bytes) | Maximum Number of Symbols in Table |
|---|---|
| 8,192 | 200 |
| 16,384 | 1224 |
| 24,576 | 2240 |
| 32,768 | 3272 |
| 65,536 | 4094 |

All symbols defined in a program (including the program name and external symbols) are entered in the symbol table providing the following conditions are met:

1. The symbol table is not full.

2. The symbol conforms to the rules

governing symbol specifications (see the topic Symbols).

3. The symbol does not appear in the name field of an assembler instruction that does not allow the specification of a name. See Figure 21 for a list of these instructions.

4. The symbol is not already contained in the symbol table. For multiple defined symbols, only the first definition of the symbol results in an entry in the symbol table. Additional definitions of the same symbol are simply flagged.

Any reference in the operand field to a symbol not in the symbol table is considered undefined; the statement is flagged. An undefined symbol in a machine instruction statement causes the entire instruction (except the operation code) to be set to zero.

## Symbol Table Overflow

If there are undefined symbols because the symbol table is full, three corrective procedures are available:

1. The assembled object deck produced by the assembler can be corrected with Replace (REP) cards before loading the program. Replace cards, a feature of the loaders, are used to alter an object deck on a byte-for-byte basis.

2. Reduce the number of symbols and then reassemble or run a new assembly.

3. Divide the program into segments and assemble each program segment separately.

Relative addressing may be used to reduce the number of symbols defined in a program. For example, the following sequence:

```
BEGIN    LA     3,10
         LA     1,0
LOOP     L      2,AUGEND(1)
         A      2,ADDEND(1)
         ST     2,SUM(1)
         LA     1,4(1)
         BCT    3,LOOP
         BC     15,OUT
AUGEND   DS     10F
ADDEND   DS     10F
SUM      DS     10F
OUT      LR     3,1
         .
         .
         .
```

could also be written:

```
BEGIN    LA     3,10
         LA     1,0
         L      2,AUGEND(1)
         A      2,AUGEND+40(1)
         ST     2,AUGEND+80(1)
         LA     1,4(1)
         BCT    3,*-16
         BC     15,AUGEND+120
AUGEND   DS     30F
         LR     3,1
         .
         .
         .
```

thus eliminating four symbols. Note that the branch address of the BC instruction is given relative to AUGEND rather than the Location Counter, since any boundary alignment caused by the DS statement would change the number of bytes between the BC and LR instruction.

Note: Using the IBM 1442-N1 or 2520-B1 Card Read-Punch reassembly procedure, the programmer must eliminate all undefined symbols from those cards that refer to such symbols in the operand field. The cards in which the undefined symbols appear in the name field can be left as they are. Since the symbol table is full, no new symbols may be defined for the reassembly.

If, in addition to reducing the number of symbols, the programmer wants to replace defined symbols (that is, symbols in the symbol table) with new symbols, the entire source program deck, with changes, must (for the IBM 1442-N1 or 2520-B1 Card Read-Punch card option) be reproduced with columns 1-24 blank prior to assembling the program. For the tape option or card option (using the IBM 2540 Card Read-Punch alone or the IBM 2501 Card Reader with a 2520-B2 or B3 Card Punch), the source deck with the desired changes can be used as is.

# BASIC UTILITY PROGRAMS

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.

The four utility programs provided are:

- the absolute loader,
- the relocating loader,
- the dump program,
- 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. The absolute loader is available in a non-relocatable assembled low and high self-loading deck. It is available as optional material in symbolic form.

## 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. The relocating loader is available as a non-relocatable assembled low self-loading deck. It is available as optional material in symbolic form.

## 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 edits the listing to fit any of eight basic formats, which are described in Output Formats. The dump program is available in a single-phase relocatable assembled version. It is also available in a two-phase version with a relocatable assembled deck for phase 1 and a non-relocatable assembled self-loading deck for phase 2. The program is available as optional material in symbolic form for the single-phase version and for both phases of the two-phase version.

## Input/Output Support Package

The input/output support package consists of a modular set of subroutines that 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. The input/output support package is available in symbolic form only.

## Loader Generator Program (LDRGEN) - (optional)

The Loader Generator Program (LDRGEN) regenerates loader program decks into a form suitable for direct loading into storage. It is available only as optional material in symbolic form.

## 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. An IBM 2540, 1442-N1 or 2520-B1 Card Read-Punch;
   or a 2501 Card Reader with a 2520-B2 or B3 Card 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 0-143 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 |
| 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 than 8K bytes of storage.

## ABSOLUTE LOADER

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.

| 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. |

Figure 24. Absolute Loader Functions

The absolute loader is available as follows:

1. a self-loading, nonrelocatable deck (assembled in lower storage).
2. a self-loading, nonrelocatable deck (assembled in high storage for an 8K configuration).
3. a symbolic deck that is optional material.

If the user wants to employ the self-loading deck, he should assemble the Absolute Loader source deck and generate a new loader by using the LDRGEN program. Also, 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.

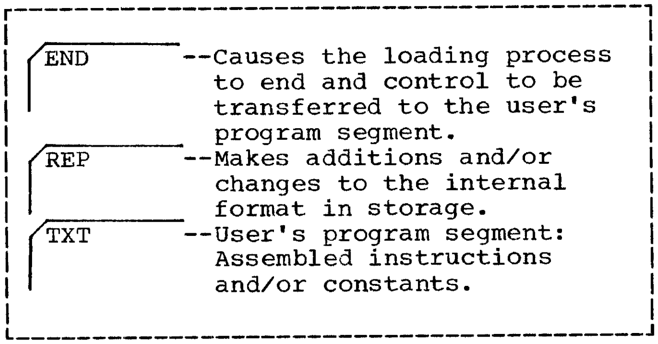
## ABSOLUTE LOADER FUNCTIONS

The functions of the absolute loader and the cards associated with each function are listed in Figure 24.

## 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 25 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.)



--Causes the loading process to end and control to be transferred to the user's program segment.

--Makes additions and/or changes to the internal format in storage.

--User's program segment: Assembled instructions and/or constants.

Figure 25. 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.)

## 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. Figure 26 defines the contents of the Text card fields.

| 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. |

Figure 26. Text Card

## 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.

Figure 27 defines the contents of the Replace card fields.

| 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. |

Figure 27. Replace Card

## 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 143 decimal, or 8F 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. Figure 28 defines the contents of the Load End card fields.

| 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 143 decimal, or 8F 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. |

Figure 28. Load End Card

## 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 "Attachment 1" as listed on the front cover of this manual.

## RELOCATING LOADER

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 Relocation and Linkage.) 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 see Program Waits and Operator Messages. 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 Loader Generator Program section.

The relocating loader is available as follows:

1. a self-loading, nonrelocatable deck (assembled in lower storage) for an 8K configuration
2. a symbolic deck that is optional material

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 Figure 29, 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, 00008F 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 00018F. 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.

In the absence of an initial SLC card, LOCCT is set to the first location available for loading above 143 decimal or 8F hexadecimal.

Figure 30 defines the contents of the Set Location Counter card.

| 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. | |

Figure 29. Unique Relocating Loader Functions

| 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. |

Figure 30. Set Location Counter Card

## 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.

Figure 31 defines the contents of the Include Segment card fields.

| 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. |

Figure 31. Include Segment Card

## 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.

Figure 32 defines the contents of the Type 0 External Symbol Dictionary card fields.

| 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. |

Figure 32. ESD Card Type 0 (Program Name)

<u>ESD Type 1 (Entry Point):</u> 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 <u>Program Waits and Operator</u>

Messages. 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.

Figure 33 defines the contents of the Type 1 External Symbol Dictionary card.

| 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. |

Figure 33. ESD Card Type 1 (Entry Point)

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).

Figure 34 defines the contents of the Type 2 External Symbol Dictionary card fields.

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).

| 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. |

Figure 34. ESD Card Type 2 (External Symbol)

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.

Figure 35 defines the contents of the Text card fields.

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 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:

| 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. |

Figure 35. Text Card

```
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.

Figure 36 defines the control of the Relocation List Dictionary card fields.

| 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:<br>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.<br>Position Header. (Two bytes.) The ESID assigned to this program segment.<br>Flag Byte (bits 0 through 3 are not used). This byte contains three items:<br>1. Size. (Bits 4 and 5.) Two bits which indicate the length (in bytes) of the adjusted address (AA Cell)<br>a. 00 - one-byte cell<br>b. 01 - two-byte cell<br>c. 10 - three-byte cell<br>d. 11 - four-byte cell |

Figure 36. Relocation List Dictionary Card (Part 1 of 2)

| 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. |

Figure 36. Relocation List Dictionary Card (Part 2 of 2)

## 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.

Figure 37 defines the contents of the Replace card fields.

| 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. |

Figure 37. Replace Card

## 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.

Figure 38 defines the contents of the Load End card fields.

| 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. |

Figure 38. Load End Card

### 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 last 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 143 decimal or 8F 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 40 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 OF THE RELOCATING LOADER

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.

Figure 39 defines the contents of the Load Terminate card fields.

| 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 39. Load Terminate Card

### 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.

| | |
|---|---|
| | --If the overlaying load procedure is used, other program segments may be loaded after the preceding program segments are executed. |
| LDT | --Causes loading process to end. If this card specifies an address for transfer of control, this overrules any address saved or specified by an END card. |
| END | --If program segment A's END card does not specify an address to which control is transferred, this card may do so. (LDT also can overrule here.) |
| RLD | --Provides information to loader for evaluating relocatable addresses in Segment B. |
| TXT | --Segment B's instructions and constants. |
| ESD (Type 2) | --Defines external symbol in Segment A to which Segment B refers. |
| ESD (Type 0) | --Defines name and length of Segment B. |
| END | --May list an address within Segment A to which control will be transferred after loading (conditional; LDT card can overrule). |
| RLD | --Provides information to loader for evaluating relocatable addresses in Segment A. |
| REP | --Causes changes or additions to be made to Segment A's internal format. |
| TXT | --Segment A's instructions and constants. |
| ESD (Type 2) | --Defines the name of Segment B as a symbol to which Segment A refers. |
| ESD (Type 1) | --Defines entry point in Segment A to which other segments may refer. |
| ESD (Type 0) | --Defines name and length of Segment A. |
| ICS | --Defines program Segment B as a segment to be loaded and specifies length to be reserved for it. |
| SLC | --Sets location counter at an absolute or symbolic address. |

Figure 40. Two Program Segments Ready for Loading by Relocating Loader (This figure reads from bottom to top.)

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.

Because 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 <u>overlay</u> 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.

4. The programmer continues to overlay 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).

### <u>Loading in Absolute Form</u>

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.

<u>Note:</u> 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 <u>Input/Output Support Package</u>.

## 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:

1. The user cannot use LOAD2 for an overlaying load procedure, since the Reference Table is destroyed whenever LOAD2 is branched to.
2. The program to be loaded by the relocating loader cannot have as entry points the symbols LOAD2 or RESUME. These symbols are entry points in the relocating loader itself.

See Relocation and Linkage and Loader Generator Program (LDRGEN) for more information.

## DUMP PROGRAM

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: locations 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

---

[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.

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 <u>Main Storage Requirements</u> under the primary heading <u>Basic Utility Programs</u> 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. a relocatable assembled deck that may be loaded by either the absolute or relocating loader; this version provides all the facilities listed in <u>Features</u>.
2. a symbolic deck (optional material) 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 <u>Features</u>.

The two-phase version is supplied as follows:

1. Phase 1 of the Two Phase Dump is available as (a) a relocatable assembled deck that may be loaded by either the absolute or relocating loader and (b) a symbolic deck (optional material) that must be assembled.
2. Phase 2 is available as (a) a self-loading, non-relocatable assembled deck and (b) a symbolic deck (optional material) that must be assembled.

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 <u>Two-Phase Dump</u>.

## 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.

Figure 41 defines the fields of a Replace card used for request numbers.

<u>Example</u>

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 place the absolute address assigned to symbolic location RTBL in card columns 7-12 of the Replace card. He will find this location in "Attachment 1" as listed on the front cover of this publication.

| 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. 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. |

Figure 41. Format of Replace Card for Request Numbers

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:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1388<br>↑ | | | | | | | | 1389<br>↑ | | | | | | | |
| Bit | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| Request Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

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. (The addresses required are supplied in "Attachment 1" as listed on the front cover of this manual.)

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 beginning of 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.

Figure 42 shows the format of the call parameter.

| 1 2 | 3 8 | 9 12 | 13 16 | 17 24 | 25 48 |
|---|---|---|---|---|---|
| Length of Parameter | Not Used | Option | Request Number | Count | Address |
| 00<br>or<br>11 | | 0000<br>0001<br>0010<br>0011<br>0100<br>0101<br>0110<br>0111 | $(0-15)_{10}$<br>$(0-F)_{16}$ | $(00-FF)_{16}$ | 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.<br>11 indicates a three half-word parameter and that at least one area of storage is to be dumped. | 00<br>or<br>C0 |
| 3-8 | | Not used. | |
| 9-12 | Option | 0000 indicates no options are exercised.<br>0001 print general registers.<br>0010 print floating-point registers.<br>0011 print floating-point and general registers.<br>0100 print console listing.<br>0101 print console listing and general registers.<br>0110 print console listing and floating-point registers.<br>0111 print all options. | 0<br>1<br>2<br>3<br>4<br>5<br>6<br>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<br>through<br>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<br>through<br>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 |

Figure 42. Call Parameter Format

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.

Figure 43 shows the format of the Control List Entry.

| 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 below). | Optional eight-byte label (2 words) |

| Bit Positions | Field Name | Significance |
|---|---|---|
| 1 | Label flag | 0 indicates that no label is associated with the area.<br>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.<br>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.<br>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. |

Figure 43. Control List Entry Format

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 Figure 44), 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
```

| 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.<br>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. |

Figure 44. Output Formats

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.

Figure 44 shows the output formats of the dump program. See Figure 45 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 addresses required to use Replace cards are supplied in "Attachment 1" as listed on the front cover of this manual.

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 a tape unit with the 7-track or dual density feature[2], the mode set desired may be punched in the first byte. Otherwise, 0000 is punched. For a mode set of 1600 BPI, the user punches C000. For example:

```
OUTDEV  DC      X'C000Addr'
```

If the output device is to be the IBM 2540 Card Read-Punch, or 2520-B2 or -B3 Card Punch, the user punches 0001. For example:

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

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

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

If the output device is to be the IBM 2520-B1 Card-Read Punch, the user punches 0003. For example:

```
OUTDEV  DC      X'0003Addr'
```

---

[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.

[2] For a discussion of the 7-track feature and dual density feature, see IBM 2400 and 2816 Model 1 Component Description, Form A22-6866.

```
    LOC 0        0004000A00000600             LOC 8        0200060060000050              LOC 16        0200046020000050
OLD PSWS         0106004000F12345    000000035001EF84      0000000000000000  0000000000000000      8006000A0000000F    Console Listing
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   Floating-Point
                                                                                                                        Registers

     GPR 0   00000000   11111111   22222222   33333333   44444444   55555555   66666666   77777777                     General
                                                                                                                     Registers
     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   Hexadecimal,
                                                                                                                    one-byte length
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  Hexadecimal,
                                                                                                                    two-byte length
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     Hexadecimal,
                                                                                                                    three-byte
000096   00000D   00000E   00000F   000010   000011   000012   000013   000014   000015   000016   000017   000018     length
         C005

000110   00000001   00000002   00000003   00000004   00000005   00000006   00000007   00000008   00000009   0000000A   Full-word
                                                                                                                    unsigned
000138   0000000B   0000000C   0000000D   0000000E   0000000F   00000010   00000011   00000012   00000013   00000014   hexadecimal
         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   Characters,
                                                                                                                    one-byte length
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  Characters,
                                                                                                                    five-byte
000216   STUVW   XYZAB   CDEFG   HIJKL   MNOPQ   RSTUV   WXYZA   BCDEF   GHIJK   LMNOP   QRSTU   VWXYZ   ABCDE   FGHIJ  length
         C004

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


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   Full-word
                                                                                                                    fixed-point
0002BC   0000000001    0000000002     0000000003     0000000004    0000000005    0000000006    0000000007    0000000008   decimal
         C009

0002DC   32767   32768-   12345   12345   12345   12345   12345   12345   12345   12345   12345   12345   12345   12545- Half-word
                                                                                                                    fixed-point
0002F2   12345   12345-   12345   12345   12345   12345   12345   12345   12345   12345   12345   12345   12345   12345- decimal
         C006

000304   .12345678 E 01    .12345678 E-01    .12345678 E 75    .12345678 E-77    .12345678 E 10    .12345678 E 10        Short-precision
                                                                                                                    floating-point
00031C   .00000000 E 00    .00000000 E 00    .00000000 E 00    .00000000 E 00    .00000000 E 00    .00000000 E 00        decimal
         C007

000330   .12345678901234567 E 75    .12345678901234567 E 00    .12345678901234567 E-77    .12345678901234567 E 00      Long-precision
                                                                                                                    floating-point
000350   .00000000000000000 E 00    .12345678901234567 E 26    .12345678901234567 E 00    .12345678901234567 E 00      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 45. Example of Storage Print Listing

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 <u>another</u> 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 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 device is to be a tape unit with the 7-track feature[2], the mode set used to create the tape must be punched in the first byte. Otherwise, 0000 is punched. For the 7-track feature with a mode set of odd parity, 800 BPI, and data convert on, the user punches 8100. For example:

```
INDEV   DC      X'8100Addr'
```

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:

```
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 <u>another</u> 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

---

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

[2] For a discussion of the 7-track feature and dual density feature, see <u>IBM 2400 and 2816 Model 1 Component Description</u>, Form A22-6866.

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 is to be used. |
| Column 21 | 0 if a printer is to be used, or<br>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<br>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. |
| columns 31-32 | If the input device is a tape unit with the 7-track feature and a mode set was used to create the tape, the same mode set must be punched in columns 31 and 32. Otherwise, leave blank. |

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 under Dump Program, 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-N1 or 2520-B1 Card Read-Punch and into the zero stacker on the punch side if the 2540 Card Read-Punch is being used. These cards must be loaded in the same order that they were produced by Phase 1.

## INPUT/OUTPUT SUPPORT PACKAGE

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.

- A set of flowcharts (Figures 59-64) is provided at the end of the I/O Support Package that illustrates the relationships of all the modules. The discussions point out which flowchart 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 flowcharts and to the listing of the I/O Support Package provided by IBM. (When using the flowcharts, 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 flowcharts 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 a symbolic deck only that contains the entire I/O Support Package. The user may select those modules that suit his particular needs.

### 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 thoroughly 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 Read-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-N1 or 2520-B1 Card Read-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-N1 or 2520-B1 Card Read-Punch). |
| SPUCW | Punch n columns; wait; this entry is only for a device whose punch address differs from the reader address (IBM 2540 Card Read-Punch or 2520-B2 or B3 Card Punch). |
| SWTPW | Write tape; wait. |
| SRWD | Rewind tape; no wait. |
| SWTMW | Write a tapemark; wait. |
| SBSRW | Backspace 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.

| | |
|---|---|
| SBSF | Backspace file; no wait. |
| SFSRW | Forward-space one physical record; wait. |
| 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 should refer to the flowcharts (Figures 59-64) 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 in Figure 59.

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

Figure 46 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.

| 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 |

Figure 46. Names and Listing Group of Required Modules and Their Associated Modules

Preliminary 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.)

Figure 47 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 Figure 47 explains this dependence.

| Primary Entry Modules | Secondary Entry Modules |
|---|---|
| SRDCW | SPCR |
| SWMSW | SSNSW |
| SPRTW | SCTLW |
| SPUC | SPCMW |
| SRTPW | SPCPW |
|  | SKIPW |
|  | SPCRW |
|  | SPUCW |
|  | SWTPW |
|  | SRWD |
|  | SWTMW |
|  | SBSRW |
|  | SBSF |
|  | SFSRW |
|  | SFSF |
|  | SBRTW |

Figure 47. Primary and Secondary Entry Modules

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 Figure 46 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 Figure 46 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 Base 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. Figure 49, 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.

## 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 flowcharts (Figures 59-64) 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 flowcharts (Figures 59-64) at the end of the I/O section; the figure 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 flowcharts 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 (Figure 62)

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. (Figure 59)

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. (Figure 60)

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. (Figure 60)

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. (Figure 61)

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:

1. The function of this routine is to provide exits for user-supplied routines that handle timer and external signal interrupts. (Figure 59)

2. The user may not use the functions of the I.O.S.P. in his external interrupt routine.

## Unit Check Group (Figure 61)

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 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 (Figure 63)

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 flowcharts, (Figures 59-64) 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 (Figure 62) 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 (Figure 62) 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 (Figure 59) 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 (Figure 60) 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 (Figure 60) 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 (Figure 60) 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 (Figure 59) 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 (Figure 61) must be included:

- External Interrupt Base Routine

If the user wishes to provide for tape error retries, the following modules (Figure 61) 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 (Figure 63) 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 I/O ENTRY MODULES

(See Figures 63 and 64.)

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 Program Waits and Operator Messages.)

### 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 48 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 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.

```
              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 48. Coding in User's Program to Test Busy Bit

## 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 or 2520-B1 Card Read-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-N1 or 2520-B1 Card Read-Punch).

Note: The IBM 1442 Card Read-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 use the Command Operation Modifier Routine.

If the IBM 2520-B1 Card Read-Punch is used and a read operation is to be followed by a punch operation, an extra feed cycle is required in order to move the last card read beyond the punch station; otherwise, the last card read will be punched.

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. The device address cannot be a new device because the locate SINTRY Table Block Routine operates in a manner directly opposed to the Multiple Unit Address Adjusting Routine.

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

---

[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.

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-N1 or 2520-B1 Card Read-Punch).

Note: The IBM 1442-N1 Card Read-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.

If the IBM 2520-B1 Card Read-Punch is used and a read operation is to be followed by a punch operation, an extra feed cycle is required in order to move the last card read beyond the punch station; otherwise, the last card read will be punched.

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 Read-Punch.

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.

Figure 49 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.

---

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

## 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.
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 50 and 51 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.

| 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 49. Module Relationships

|  |  |  |
|---|---|---|
| 7. | Constants<br>Housekeeping | --I/O Base Routine; housekeeping and constants area. |
| 6. | SPCPW | --Secondary entry module; requires the presence of numbers 1, 2, 5. |
| 5. | SPRTW | --Primary entry module; requires the presence of number 1. |
| 4. | SRDCW | --Primary entry module; requires the presence of number 1. |
| 3. | Interrupt Analyzer<br>I/O Initiator | --I/O Base Routine; I/O Initiator of Interrupt Analyzer. |
| 2. | Secondary Call Entry Table | --Contains address of secondary entry module (number 6); uses unit address from Primary Call Entry Table (number 1). |
| 1. | Primary Call Entry Table | --Contains address of primary entry module (numbers 4 and 5), device unit address, and space for exceptional condition return. |

Figure 50. Organization of Subroutine Modules without Optional Routines (Read from bottom to top.)

|  |  |  |
|---|---|---|
| 10. | Multiple Unit Addr-Dev. Rt. | --Enables user to use a new unit device address. |
| 9. | Constants<br>Housekeeping | --I/O Base Routine; housekeeping and constants area. |
| 8. | SPCPW | --Secondary entry module; requires the presence of numbers 1, 2, 6. |
| 7. | SPRTW | --Primary entry module; requires the presence of number 1. |
| 6. | SRDCW | --Primary entry module; requires the presence of number 1. |
| 5. | Unit Ex. Cond. Rtrn. Adr. Rt. | --Enables user to take action in his program on an exceptional condition indication. |
| 4. | Unit Ex. Cond. Base Routine | --Enables user to take action in his program on an exceptional condition indication. |
| 3. | Interrupt Analyzer<br>I/O Initiator | --I/O Base Routine, first in deck;<br>I/O Initiator and Interrupt Analyzer. |
| 2. | Secondary Call Entry Table | --Contains the address of the secondary entry (number 8), requires presence of number 1 for unit device address. |
| 1. | Primary Call Entry Table | --Contains the address of primary entry modules (numbers 6 and 7, unit device address for numbers 6, 7, and 8, and space for exceptional condition return address). |

Figure 51. 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 that 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 4095) 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 52 shows the construction of the Primary Call Entry Table and Figure 53 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)         Card reader address
         DC     A(0)          Area for unit exceptional condition return
                              address
*
         DC     A(SWMSW)      Write message and wait
STYPR    DC     A(9)          Typewriter address
         DC     A(0)          Area for unit exceptional condition return
                              address
*
         DC     A(SPRTW)      Print a line and wait
SPRTR    DC     A(11)         Printer address
         DC     A(0)          Area for unit exceptional condition return
                              address
         DC     A(SPUC)       Punch
SPNCH    DC     A(13)         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 52. 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 53. 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 54 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 54. 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 <u>Direct Linkage</u> 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 55 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. <u>Note:</u> 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 56 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(0,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(0,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 54) 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 54) 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 55. 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(0,1)       Load address of Sense Entry
          L        6,PITBAD       Load address of SINTRY
          LA       6,28(0,6)      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 56. 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 57 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:

```
          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
          L        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 57. Control Entry Coding Example

1. Include the Locate SINTRY Table Unit Block and Control 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 because the locate SINTRY Table Block Routine operates in a manner directly opposed to the Multiple Unit Address Adjusting Routine.

CARD-ONLY INSTALLATION

A symbolic 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 <u>Input/Output Support Package</u> 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, 1442-N1 or 2520-B1 Card Read-Punch;
  or a 2501 Card Reader with a 2520-B2 or B3 Card Punch
- One IBM 1052 Printer-Keyboard
- One IBM 1403 or 1443 Printer

CARD-TAPE PACKAGE

Another symbolic 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, 1442-N1 or 2520-B1 Card Read-Punch;
   or a 2501 Card Reader with a 2520-B2 or B3 Card 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 <u>Input/Output Support Package</u> 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.

FLOWCHARTS OF MODULE RELATIONSHIPS

These flowcharts (Figures 59-64) 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 flowcharts 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 flowcharts:

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 flowcharts; 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). Figure 58 defines these codes.

| 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. |

Figure 58. Chart Codes for Basic Utility Programs

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. |

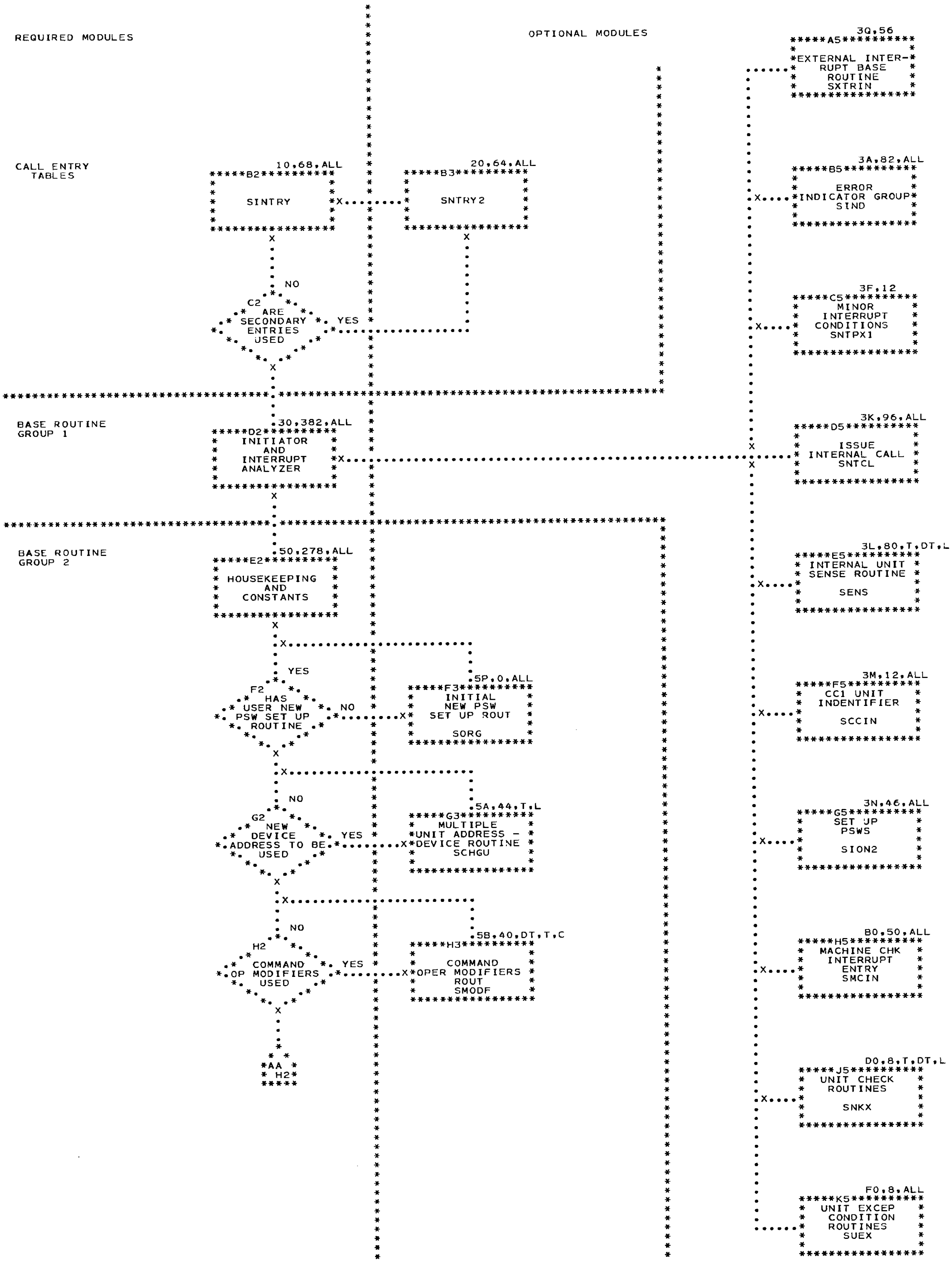


Figure 59. Required Modules and Interrupt Action Modules

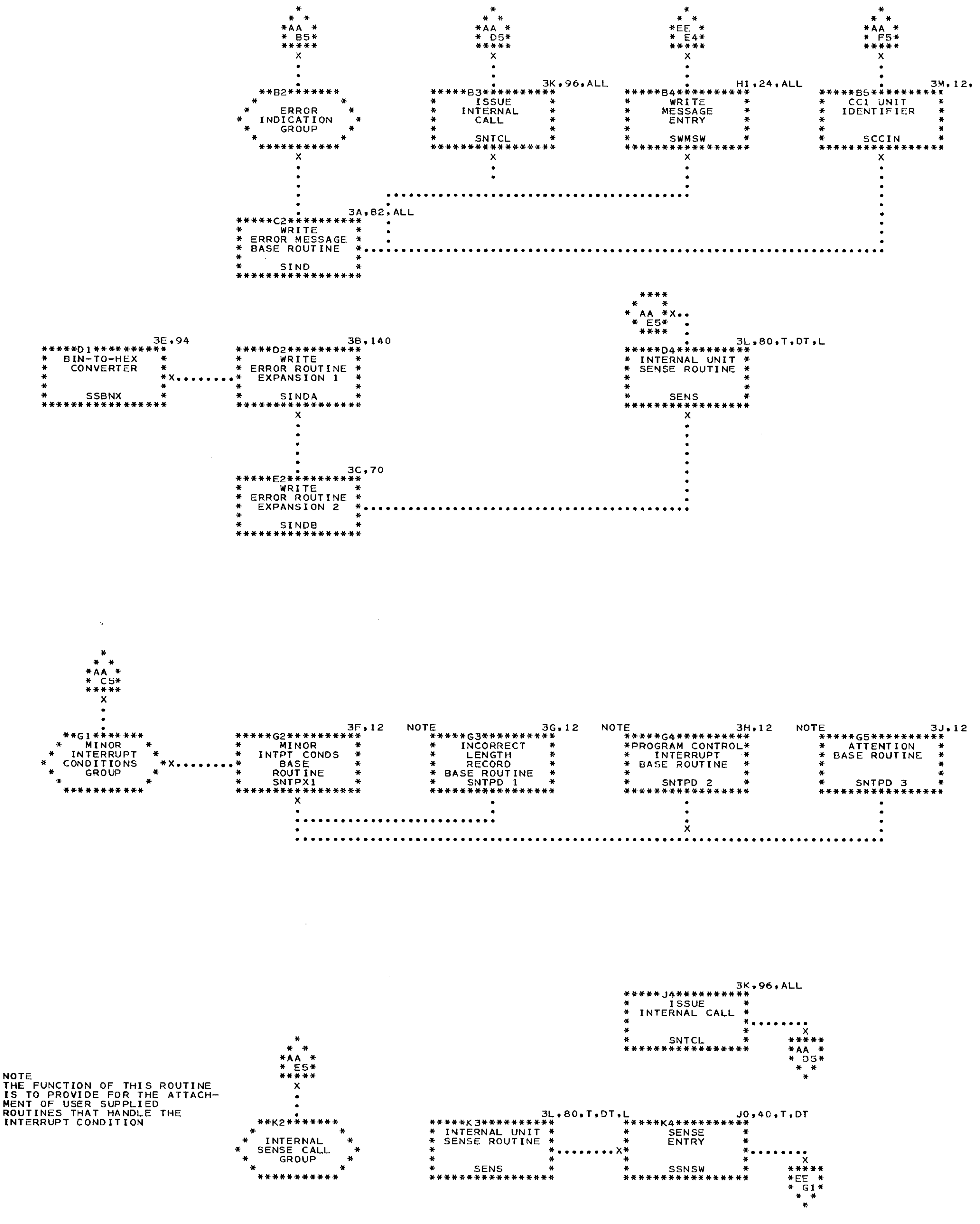


Figure 60. I/O Base Routine - Group 1 Optional Modules

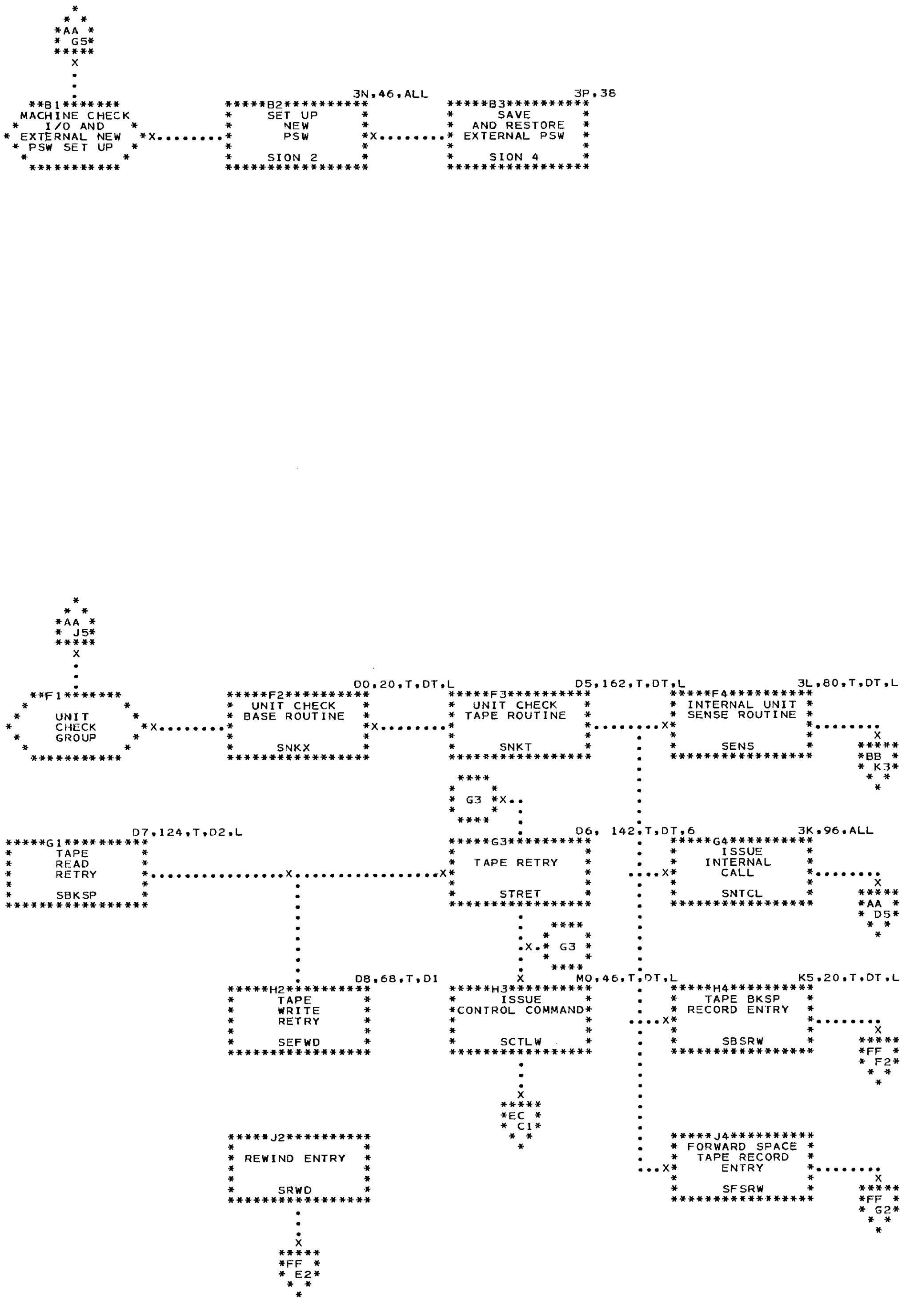


Figure 61. I/O Base Routine-Group 1 PSW Routines, Machine Check Group, Unit Check Group

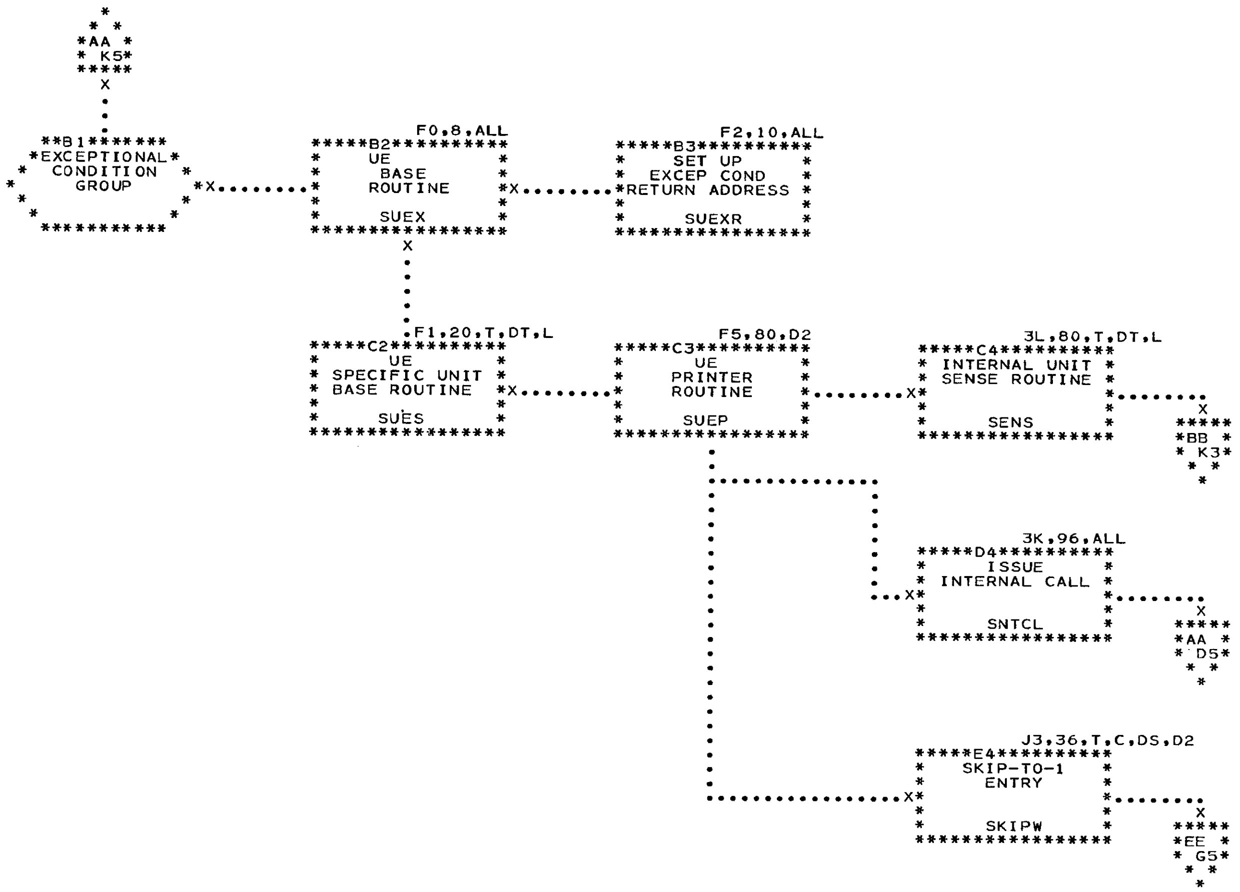
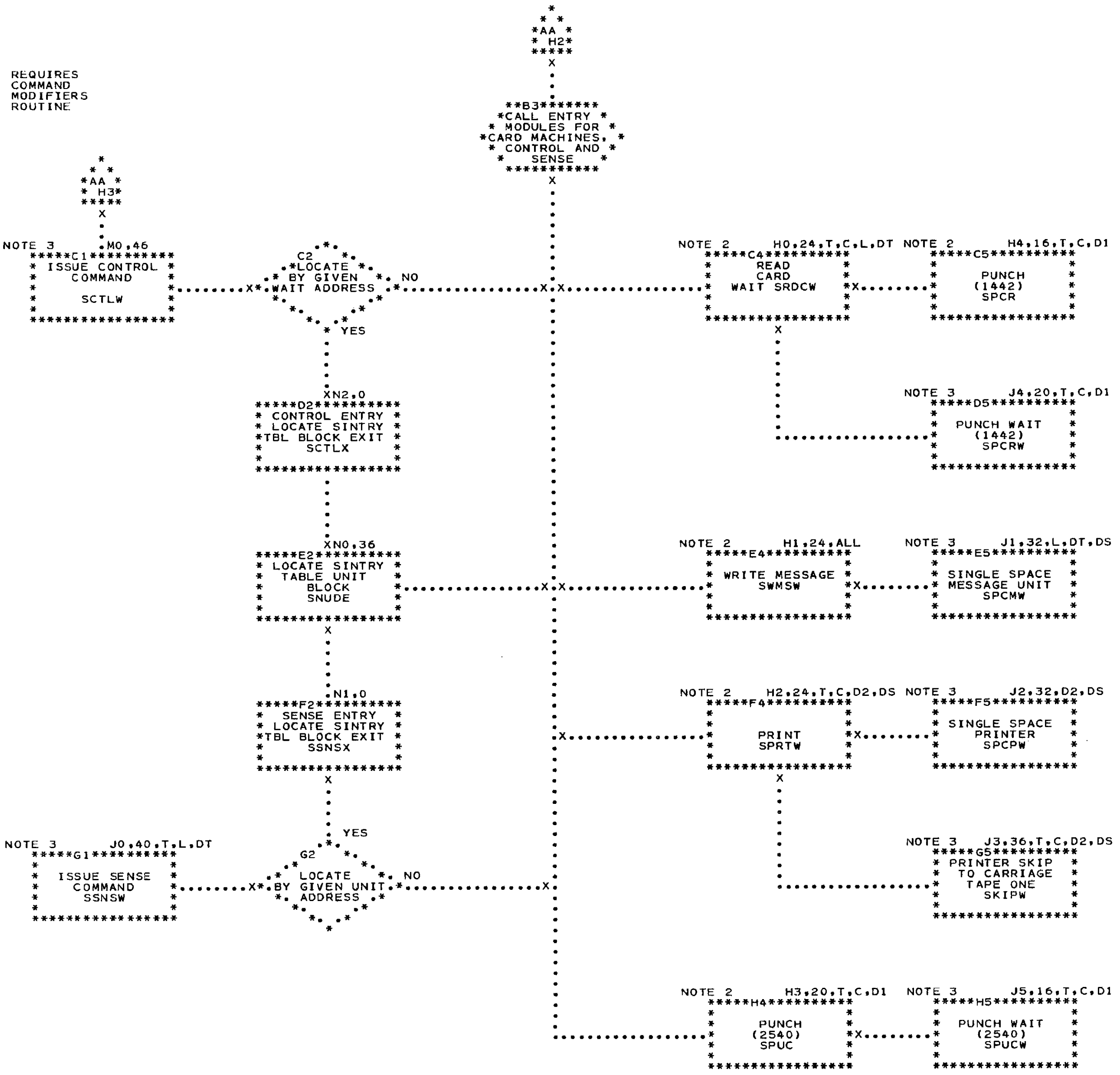


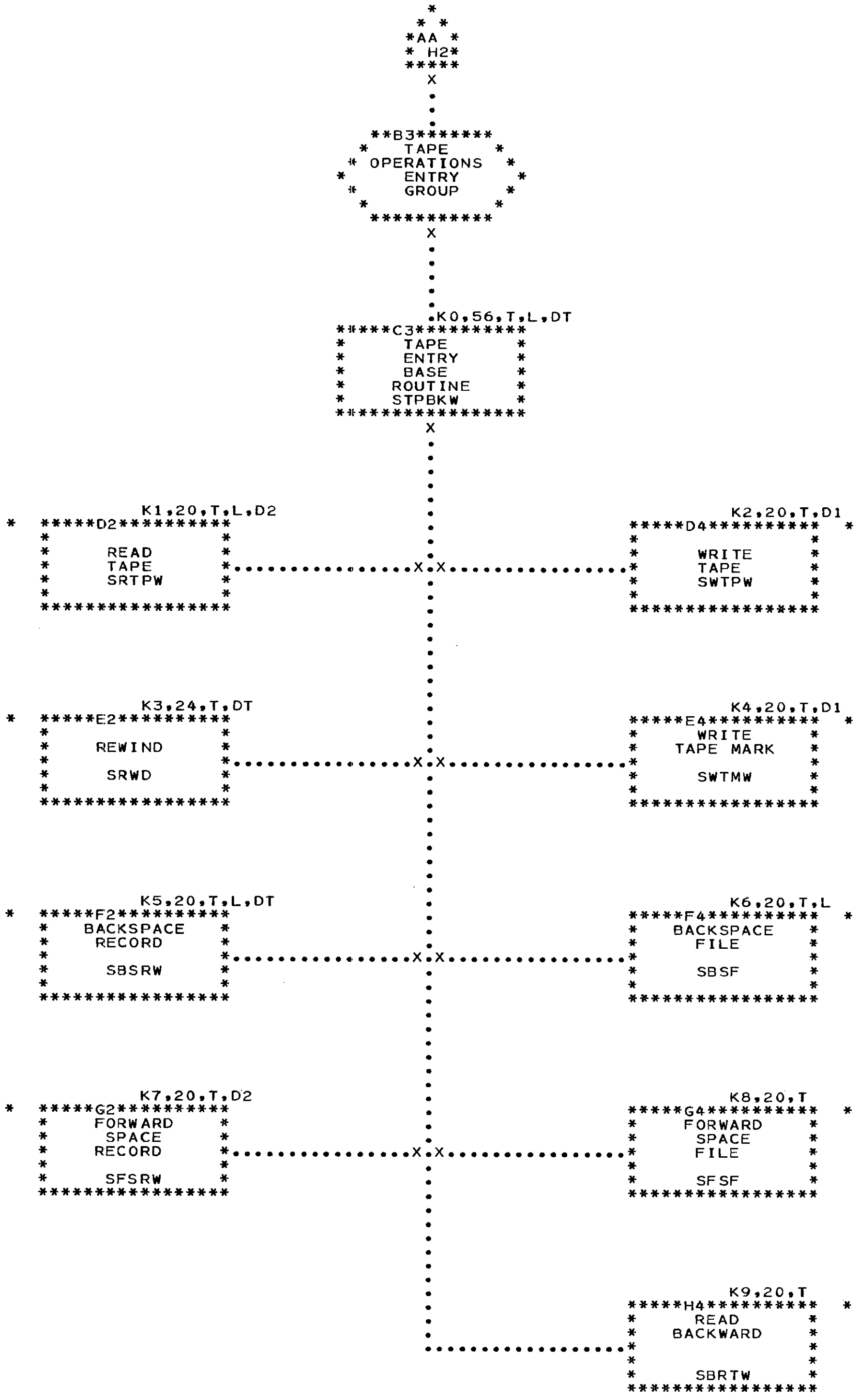
Figure 62. Unit Exceptional Condition Group



NOTE 2 - REQUIRES PRESENCE OF ADDRESS TO ENTRY IN SINTRY TABLE

NOTE 3 - REQUIRES PRESENCE OF ADDRESS TO ENTRY IN SNTRY2 TABLE

Figure 63. I/O Call Entry Group Modules for Non-Tape, Sense and Control Operations



* REQUIRES PRESENCE OF ADDRESS TO ENTRY IN SNTRY 2 TABLE

Figure 64. I/O Call Entry Group Modules for Tape Operations

## RELOCATION AND LINKAGE

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.

Note: The program to be loaded by the relocating loader cannot have as entry points the symbol LOAD2 or RESUME. These symbols are entry points in the relocating loader itself.

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  144
         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 65 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 65 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 65 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 65 are:

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

## LOADER GENERATOR PROGRAM (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 as optional material. It 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.

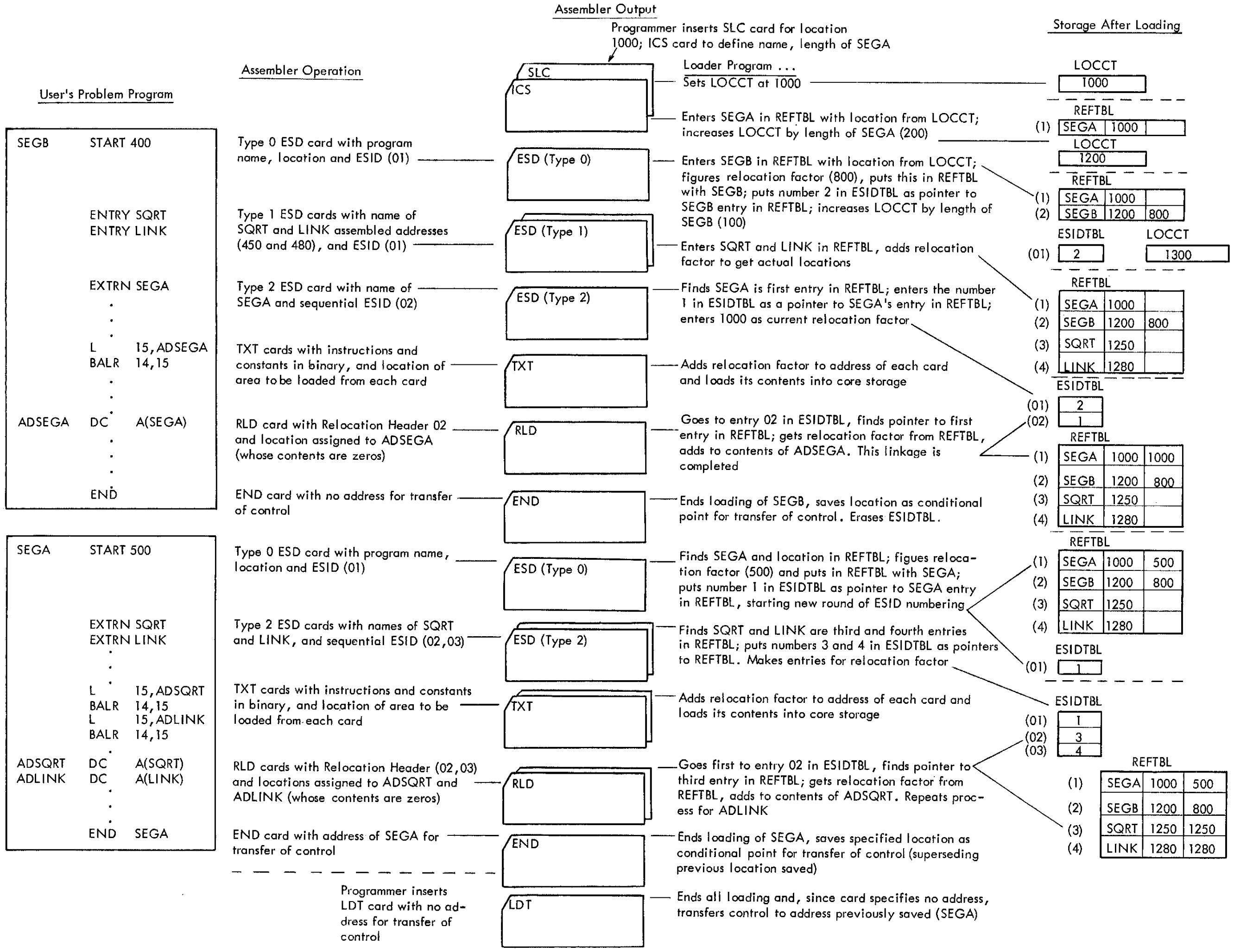


Figure 65. Example of the Loading Process

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 |

Note: The value for OMEGA must be 72 bytes below maximum main storage address.

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 half-word 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.

# BASIC ASSEMBLER OPERATING PROCEDURES

This section provides operating information and techniques for the System/360 Basic Assembler and is concerned only with operating considerations, not with the internal logic of the programs.

The Basic Assembler is essentially a language translator. It translates source programs written in the Basic Assembler language into executable machine-language object programs. The assembler is divided into two parts, Phase 1 and Phase 2.

Input to Phase 1 consists of source program statements punched into cards or written on magnetic tape. Phase 1 partially translates the source program statements into machine-language object code. The partially translated statements are passed to Phase 2 (see Figure 66) where the translation process is completed. The output produced by Phase 1 (that is, the partially translated source statements) must be passed to Phase 2 via punched cards or magnetic tape.

Note: Certain character constants (C' ') that do not fall into the normal BCD configuration, when entered into System/360 by means of another computer, may lose bits during the card-to-tape phase.

The assembler is available as two non-relocatable, assembled self-loading decks, one for each phase. It is also available as optional program material in symbolic form for both phases.

## Program Listings

The assembler provides a program or error listing for each assembly if a printer or printer-keyboard is attached to the system, and the assembler has been instructed to provide listings or error listings. This is described in detail in the section Phase 1 Configuration Card.

## Assembled Object Program Output

Assembled object programs produced by the assembler may be punched in cards or written on tape. The specification of the object program storage medium is described in detail in the section Phase 1 Configuration Card.

## Machine Configuration

The IBM System/360 Basic Programming Support Basic Assembler program requires the following minimum machine configuration:

- An IBM System/360 with 8,192 bytes of storage
- An IBM 2540, 1442-N1 or 2520-B1 Card Read-Punch;
  or an IBM 2501 Card Reader with a 2520-B2 or B3 Card Punch
- The Standard Instruction Set

Note: In reference to the card assembler, the IBM 2501 Card Reader with the

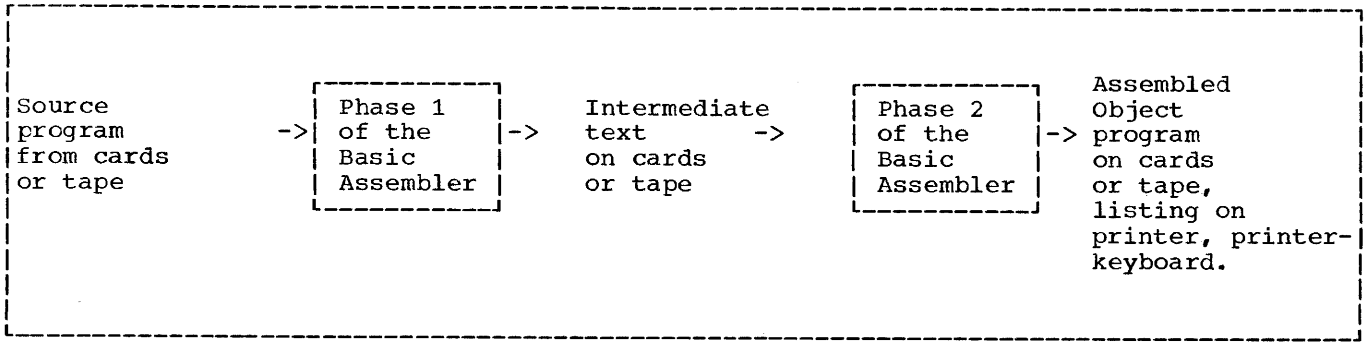


Figure 66. Basic Programming Support Basic Assembler

IBM 2520-B2 or B3 Card Punch is equivalent to the IBM 2540 Card Read-Punch.

If additional input/output devices are attached to the system, the assembler's operational capabilities are increased. The various input/output devices and their uses are listed below.

IBM 2400-Series Magnetic Tape Unit:

From one to five magnetic tape units can be used for the storage of any of the following:

1. Source program
2. Basic Assembler object decks
3. Intermediate text
4. Program listing (Model 40 or larger system only. See note in the section Assembling with Card and Tape Configuration.)
5. Object program

Note: 7-track tape units must have the Data Conversion special feature.

One IBM 1403 or 1443-2 Printer: Used by the assembler to provide program listings, complete with operator and error messages, for each assembly.

One IBM 1052 Printer-Keyboard: Used by the assembler to provide program listings, complete with operator and error messages, for each assembly.

One IBM 1403 or 1443-2 Printer and One IBM 1052 Printer-Keyboard: The assembler uses the IBM 1403 or 1443-2 Printer to print program listings. The IBM 1052 Printer-Keyboard is used for operator and error messages.

## ASSEMBLER INITIALIZATION

Since all installations do not have the same machine configuration, the Basic Assembler program must be tailored for operation at each installation. This tailoring consists of defining to the assembler:

1. The main-storage size of the system.
2. The input/output devices attached to the system and their addresses.
3. What use is to be made of cards and magnetic tape.

In addition to the initialization associated with the machine configuration, other initialization may instruct the assembler to print or suppress program listings or to print only error listings.

The Basic Assembler is initialized through the use of configuration cards. There are two configuration cards, one for each phase of the assembly program. The cards are called the Phase 1 and Phase 2 Configuration Cards.

### PHASE 1 CONFIGURATION CARD

The Phase 1 Configuration Card is a Replace card which describes to Phase 1 of the assembler the machine configuration upon which it is to operate. The card is inserted in the Phase 1 deck just before the END card. The Phase 1 Configuration Card has the format shown in Figure 67.

| Cols. | Fld. | Description |
|---|---|---|
| 1- 4 | 1 | Contains a 12-2-9 punch followed by the characters REP. This identifies the card as a configuration card. |
| 5- 6 | | Blank. |
| 7-12 | 2 | Contains 000090. This is the hexadecimal starting address, in storage, where the data in columns 17-55 is to be placed. |
| 13-14 | | Blank. |
| 15-16 | 3 | Contains 01. This is a constant. |
| 17-20 | 4 | Contains a 0, if the source input device is a card reader, followed by the three-digit hexadecimal address of the reader. If the source input device is a tape unit, the field contains a 1, followed by the three-digit hexadecimal address of the source device. |
| 21 | 5 | Comma. |
| 22-25 | 6 | If the intermediate text data is to be stored on cards, this field contains a 0, followed by the three-digit hexadecimal address of the card punch. If the intermediate text is to be stored on tape, the field contains a 1, followed by the three-digit hexadecimal address of the tape unit. |
| 26 | 7 | Comma. |
| 27-30 | 8 | If the assembler is to be copied on tape, this field contains a 1, followed by the three-digit hexadecimal address of the tape unit which will be used. If the assembler is not to be copied, the field contains 0's. |
| 31 | 9 | Comma. |
| 32-35 | 10 | Contains a 0, followed by the three-digit hexadecimal address of the system message device -- a printer or a printer-keyboard -- or 0's, if neither of these devices is attached to the system. |
| 36 | 11 | Comma. |

Figure 67. Format of Phase 1 Configuration (Part 1 of 2)

| Cols. | Fld. | Description |
|---|---|---|
| 37-40 | 12 | If the program listing is to be printed on the printer or printer-keyboard, this field contains a 0, followed by the three-digit hexadecimal address of one of these. If the listing is to be written on tape, the field contains a 1, followed by the address of the tape unit. If none of these devices is used, the field contains 0's. |
| 41 | 13 | Comma. |
| 42-45 | 14 | If the final object program is to be punched on cards, this field contains a 0, followed by the three-digit hexadecimal address of the card punch. If the object program is to be written on tape, the field contains a 1, followed by the address of the tape unit. |
| 46 | 15 | Comma. |
| 47-50 | 16 | Contains a four-digit hexadecimal number. The first two digits indicate the mode set code (see Figure 68) for the source input tape. The second two digits indicate the mode set code for the object program output. |
| 51 | 17 | Comma. |
| 52-53 | 18 | The first two digits of this field indicate the mode set code (Figure 68) for the device on which a listing is to be written. |
| 54 | | 0 for 8K<br>1 for 16K<br>2 for 32K<br>3 for 64K<br>5 for 24K |
| 55 | | 0 for 2540 or 2501 with a 2520-B2 or B3<br>2 for 2520-B1<br>4 for 1442-N1 |
| 56 | | Must be blank |
| 57-80 | | May include anything that programmer wishes. |

Figure 67. Format of Phase 1 Configuration (Part 2 of 2)

The code 03 must be used for unit record devices, and 7-track tapes that will be used only on System/360. For 9-track tape, code CB must be used for 800 BPI and code C3 for 1600 BPI.

The following mode set codes are used for 7-track tapes that will be used on System/360 and some other machine:

| Code | Density | Parity |
|---|---|---|
| 2B | 200 BPI | Even |
| 3B | 200 BPI | Odd |
| 6B | 556 BPI | Even |
| 7B | 556 BPI | Odd |
| AB | 800 BPI | Even |
| BB | 800 BPI | Odd |

Figure 68. Mode Set Codes

## PHASE 2 CONFIGURATION CARD

The Phase 2 Configuration Card is a Replace card that must be inserted in the middle of the Phase 2 deck just before an existing dummy END card (not before the actual END card of the Phase 2 deck). This dummy END card contains no transfer address. The Phase 2 Configuration Card is identical to the Phase 1 Configuration Card, except that columns 22-25 and 54 are punched in the following manner:

Columns 22-25
Must reflect the input device for the intermediate text. If the intermediate text data has been stored in cards, this field contains a 0, followed by the three-digit hexadecimal address of the card reader. If the intermediate text data has been stored on tape, the field contains a 1, followed by the three-digit hexadecimal address of the tape unit.

Column 54
- 0 - Error Listing on Printer
- 1 - Error Listing on Printer-Keyboard
- 4 - Program Listing on Printer
- 5 - Program Listing on Printer-Keyboard

The results of mispunching configuration cards are unpredictable.

## RUNNING AN ASSEMBLY JOB

### A. ASSEMBLING ON A CARD SYSTEM USING THE 2540 OR 2501 CARD READER WITH A 2540-B2 OR B3

Assembler: cards
Source deck: cards
Intermediate text: cards
Object program: cards
Listing: printer or printer-keyboard

1. Make the printer and printer-keyboard ready for use.
2. Clear the card reader of cards.
3. Insert the proper configuration cards immediately before the proper END card in both decks of the assembler. See Phase 2 Configuration Card section. Detailed information concerning these cards is presented in the section Assembler Initialization.
4. Place the Phase 1 deck of the assembler in the reader hopper; then place the source program deck in the hopper.
5. Place blank cards in the punch hopper. The number of blank cards must be equal to or greater than the number of cards contained in the source program deck.
6. Place additional blank cards for the symbol table in the punch hopper at the ratio of approximately one blank card for every twenty source program cards.
7. Initialize card reader-punch for use.
8. Press end-of-file key on the card reader-punch.
9. Select the card reader with the load-unit switches on the system control panel and press load key.
10. A program wait with the location counter containing 1EI occurs at the completion of Phase 1. If the system has provisions for typing messages, a message "1EI" is typed on the printer or printer-keyboard. The contents of the stackers at this point are shown in Figure 69.
11. Make printer or printer-keyboard ready for use if necessary.

12. Clear the card reader of cards.

13. Place the Phase 2 deck of the assembler in the card read hopper.

14. Place intermediate text deck on the assembler deck.

15. Place blank cards in the punch hopper at the ratio of one blank card for every ten source program cards.

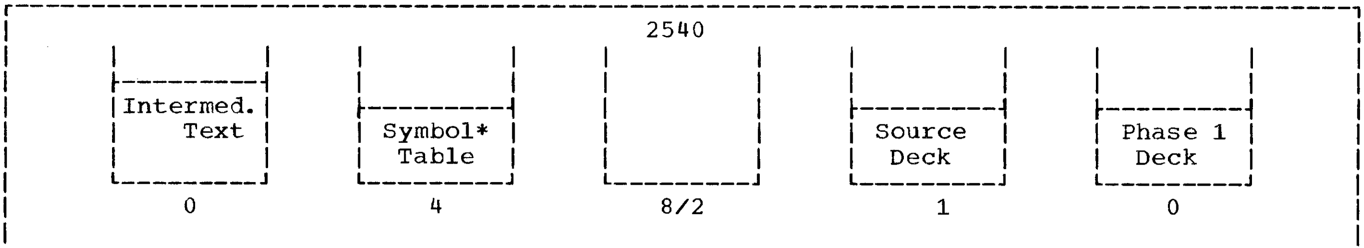
16. Press end-of-file key on card reader-punch.

17. Select the card reader with the load-unit switches on the system control panel and press the load key.

18. A "2EI" message or a program wait with the location counter containing 2EI signals the end of the second phase of assembling. The contents of the stackers at the completion of the assembly job are shown in Figure 70.

19. When a punching error is detected on a card, the card containing the error and the card <u>immediately following</u> it will fall into the reject hopper.

20. For program execution information refer to the <u>Using the Loaders</u> section.



The intermediate text deck consists of data produced by Phase 1 for use as input to Phase 2.
*The symbol table deck is produced for use in a special application described in the section <u>Special Procedures</u>.

Figure 69. Stacker Contents for the IBM 2540 Card Read-Punch at End of Phase 1

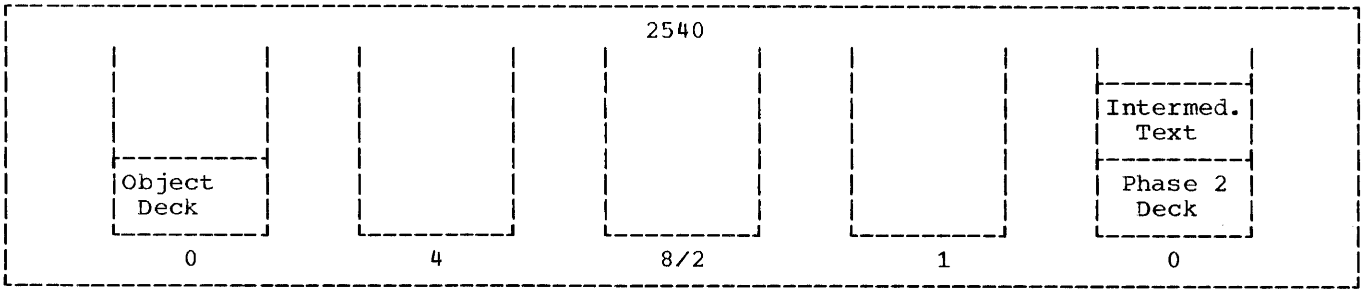


Figure 70. Stacker Contents for the IBM 2540 Card Read-Punch at End of Assembly

## B. ASSEMBLING ON A CARD SYSTEM USING THE 1442-N1 OR 2520-B1

Assembler: cards
Source deck: cards
Intermediate text: cards
Object program: cards
Listing: printer or printer-keyboard

1. Make the printer and printer-keyboard ready for use.

2. Clear the card reader of cards.

3. Insert the proper configuration cards immediately before the proper END card in both decks of the assembler. See <u>Phase 2 Configuration Card</u> section. Detailed information concerning

configuration cards is presented in the section Assembler Initialization.

4. Place the Phase 1 deck of the assembler in the reader hopper. Then place the source program deck in the hopper. Columns 1-24 of the source program must be blank.

5. Place blank cards for the symbol table in the hopper at the ratio of approximately one blank card for every twenty source program cards.

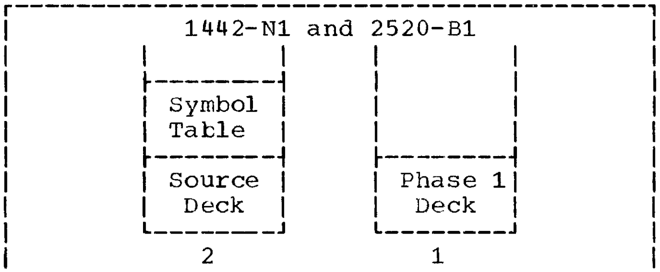
6. Initialize card reader-punch for use. (Do not press end-of-file key.)

7. Select the card reader with the load-unit switches on the system control panel and press the load key.

8. A program wait with the location counter containing 1EI occurs at the completion of Phase 1. If the system has provisions for typing messages, a message "1EI" is typed on the printer or printer-keyboard. The contents of the stackers at this point are shown in Figure 71.

9. Make the printer and printer-keyboard ready for use if necessary.

10. Clear the card reader of cards.



The intermediate text deck consists of data produced by Phase 1 for use as input to Phase 2. The symbol table deck is produced for use in a special application described in the section Special Procedures. Note that the intermediate text is contained in columns 1-24 of the source deck on the 1442-N1 and 2520-B1.

Figure 71. Stacker Contents for IBM 1442-N1 and 2520-B1 Card Read-Punch at End of Phase 1

11. Place Phase 2 deck of the assembler in the card read hopper.

12. Place the source program deck (containing the intermediate text) in the card hopper.

13. Place blank cards in the hopper behind the source program deck (containing the intermediate text).

14. Select the card reader with the load-unit switches on the system control panel and press the load key.

15. A program wait with the location counter containing 2HA or a message "2HA" indicates that blank cards must be placed in the 1442-N1 or 2520-B1 card hopper. Remove any cards in the hopper, insert blanks, and replace the cards just removed. The number of blank cards is governed by the machine's storage size: 15 blanks for an 8K machine, 80 blanks for a 16K machine, 140 blanks for a 24K machine, 200 blanks for a 32K machine, and 450 blanks for a 64K machine. After inserting the blanks, press the interrupt key.

16. A "2EI" message or a program wait with the location counter containing 2EI signals the end of the second phase of assembling. completion of assembling are shown in Figure 72.

18. For program execution information refer to Using the Loaders section.

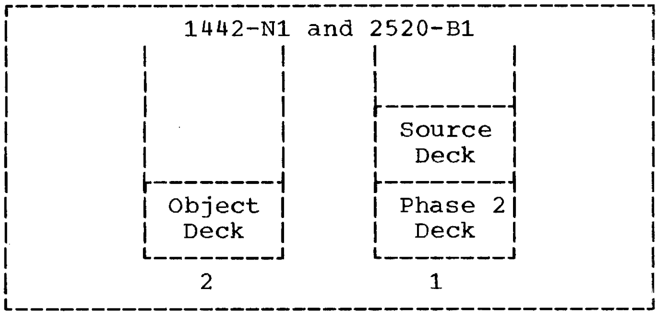


Figure 72. Stacker Contents for the IBM 1442-N1 and 2520-B1 Card Read-Punch at End of Assembly

## C. COPYING THE ASSEMBLER ON TAPE

1. Punch the correct configuration cards (described in detail in Assembler Initialization) and place them before the END cards in Phases 1 and 2.

2. Ready the card reader.

3. Place Phases 1 and 2 of the assembler in the read hopper.

4. Load tape on tape unit whose address was specified in field 8 of configuration cards.

5. Ready tape unit.

6. Press end-of-file on card reader.

7. Select the card reader with the load-unit switches on the system control panel and press load key.

8. When the "1EI" message appears in the location counter, press load key a second time to write Phase 2 on tape.

9. When the "2EI" message is printed or appears in the location counter, the assembler has been written entirely on the selected tape. Rewind before using. (Note that the assembler may be written on one tape unit and run on a different tape unit, provided they both have the same number of tracks.)

## D. ASSEMBLING WITH CARD AND TAPE CONFIGURATION

Assembler: cards or tape (If tape used, intermediate text must be on tape.)
Source deck: cards or tape
Intermediate text: cards or tape
Object program: cards or tape
Listing: printer, printer-keyboard, or tape

Note: If tape is used for listing, it must be 800 BPI or less. Also, tape may be used for listing only with a Model 40 or larger system because the speed of these systems is sufficient to handle "chain data".

1. Ready assembler input device.

2. Ready source program input device.

3. Ready intermediate text device.

4. Ready object program device.

5. Ready listing device.

6. Select assembler input device with load-unit switches on the system control panel and press load key.

7. If intermediate text medium is punched cards, see steps A 10-20 or B 8-18 in the preceding sets of instructions.

8. If intermediate text is on tape and the assembler is on cards, a "1EI" message or a program wait with the location counter containing 1EI signals the end of Phase 1. Load Phase 2 of the assembler. If the assembler is on tape, the message will not appear and control will automatically pass to Phase 2. Note that when the intermediate text is on tape, Phase 2 must always follow immediately after Phase 1, since no symbol table is punched.

9. A "2EI" message or a program wait with the location counter containing 2EI signals the end of the second phase of assembling.

10. For program execution information refer to Using the Loaders section.

## SPECIAL PROCEDURES

There are three special procedures available for use with card systems. They are:

1. A procedure for saving time when reassembling a previously assembled program on a 1442-N1 or 2520-B1 card system.

2. A procedure for running an assembly job on a card system when Phase 2 is not executed immediately after Phase 1.

3. A procedure for saving time during Phase 2 when using a 1442-N1 or 2520-B1 card system that punches the assembled object programs into cards.

### 1442-N1 or 2520-B1 Card System Reassembly Procedure

A special reassembly procedure is provided for card systems using the IBM 1442-N1 or 2520-B1 Card Read-Punch. This procedure enables a previously assembled program to be reassembled in less time than that required for a new assembly.

To use this procedure, one must have the symbol table deck produced by Phase 1 of the previous assembly (see Figure 69).

Input to Phase 1 during a reassembly consists of the Phase 1 assembler deck followed, in order, by the previously punched symbol table, the source program, and blank cards (into which the new symbol table will be punched). The number of blank cards should be approximately equal to the number of cards in the previously punched symbol table. Note that the only difference between the Phase 1 input required for a new assembly and the Phase 1 input required for a reassembly is the inclusion of the symbol table deck in the latter case. Other than preparing the Phase 1 input, the actions required to run a reassembly job are exactly the same as those required for a new assembly.

## Interrupted Assemblies on Card Systems

If a card system assembly job is interrupted after the completion of Phase 1, but before the conclusion of Phase 2, a special procedure is provided to complete the assembly job without re-executing Phase 1. Tape assembly jobs may not be interrupted.

When this procedure is used, it is only necessary to run Phase 2 of the assembler (Phase 1 was run before the interruption). Input to Phase 2, in this case, is the same as that required for a new assembly, with one exception. That exception consists of placing the symbol table deck (produced by Phase 1 before the interruption) on top of the Phase 2 assembler deck in the input card hopper. The rest of the Phase 2 input is then placed on top of the symbol table deck. Other than setting up the Phase 2 input, the actions required to run Phase 2 are the same as those required to run Phase 2 of a new assembly.

## 1442-N1 or 2520-B1 Card Systems with Card Output

Occasionally, when running an assembly job on a 1442-N1 or 2520-B1 card system with card output, a program wait occurs during Phase 2 with the location counter containing 2HA, indicating the need for more blank cards. If the system has provisions for typing messages, a message "2HA" is typed out. In either case, blank cards must be placed in the 1442-N1 or 2520-B1 card hopper and the interrupt key must be pressed (see Figure 70).

This intervention may be avoided by interleaving blank cards with the source program deck before starting Phase 2 of the assembler. The size of the system's storage governs the manner in which the blank cards are interleaved with the source program, as shown in the following:

| Main-Storage Size | Action |
|---|---|
| 8K | Insert approximately 15 blank cards after each 150 source program cards. |
| 16K | Insert approximately 80 blank cards after each 800 source program cards. |
| 24K | Insert approximately 140 blank cards after each 1400 source program cards. |
| 32K | Insert approximately 200 blank cards after each 2,000 source program cards. |
| 64K | Insert approximately 450 blank cards after each 4,500 source program cards. |

# BASIC UTILITY PROGRAMS OPERATING PROCEDURES

This section provides operating information and techniques for the basic utility programs. The basic utility programs enable the user to print out the contents of registers and/or storage, load assembled programs, and program the use of input/output devices. The material is concerned only with operating considerations, not with the internal logic of the programs.

## THE SINGLE-PHASE DUMP PROGRAM

The single-phase dump program is designed to produce listings of the contents of registers and/or storage.

When used, the single-phase dump program resides in storage along with the user's program. Figure 73 shows the relationship between the single-phase dump program and the user's program.

### INITIALIZATION OF THE SINGLE-PHASE DUMP PROGRAM

The single-phase dump program is available from IBM as an assembled relocatable object deck. It is also available as optional material in symbolic form. Before the program can be used, it will require modification for operation on the installations's machine. This modification consists of altering three constants near the end of the IBM-supplied program. These constants are shown in Figure 74. They are identified in the figure by the number 1 in column one on the left-hand side. One must ensure that these locations properly describe the installation's machine configuration before using the single-phase dump program. Note that a card's position in the deck should not be altered during the modification process.

### USING THE SINGLE-PHASE DUMP PROGRAM

The single-phase dump program is essentially a subprogram designed for use by the programmer; its use, therefore, is primarily his concern. He must define, in his program, the registers and/or storage areas whose contents are to be listed. In addition, he must define the format of the listing. Finally, he must transfer control to the single-phase dump program in order to have the listing produced.

In order to execute a program that uses the single-phase dump program, both programs must reside in storage, and the proper linkage must exist between them. These requirements can be fulfilled by either of two methods.

One method consists of assembling the single-phase dump and user's programs together. The resulting assembled object program contains both the single-phase dump and problem programs with the appropriate linkages. It can be loaded into storage for execution by the Absolute or Relocating Loaders. (The Relocating Loader cannot be used to accomplish this on an 8K configuration.)

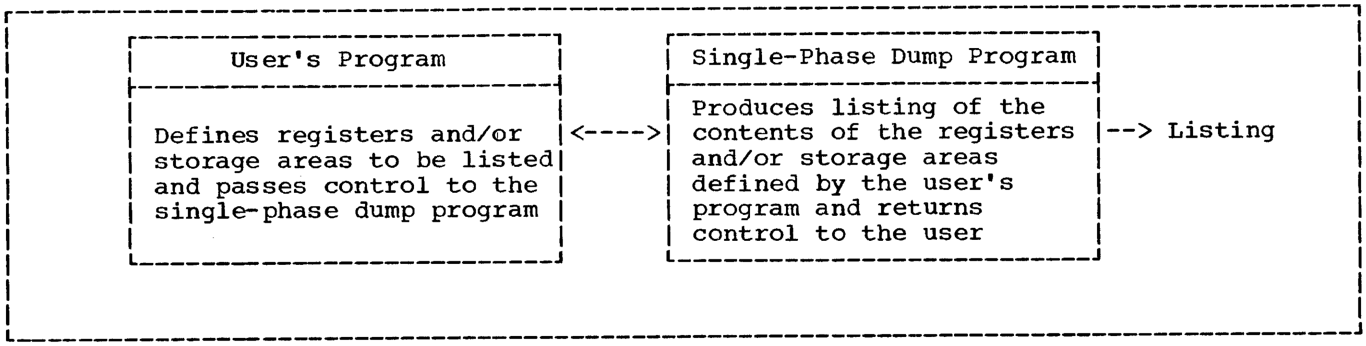


Figure 73. The Single-Phase Dump Program

1. Single-Phase Dump Program
2. Phase 1 of the Two-Phase Dump Program
3. Phase 2 of the Two-Phase Dump Program

| 1 | 2 | 3 | Name | Operation | Operand | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | DSTOPL | DC | AL3 (zzzzz) | zzzzz represents the machine's storage size in bytes. Valid operands may be:<br>AL3(8192) AL3(16384) AL3(32768) AL3(65536) |
| | | 3 | INDEV | DC | X'zzzzzzzz' | zzzzzzzz represents eight hexadecimal digits generated in the following way:<br><br>_Digits_ _Description_<br>1-2 If the input device is a tape unit with the 7-track feature, these digits specify the mode set used to create the tape.[1] Otherwise, these digits are 00.<br>3 Always 0.<br>4 Specifies the Phase 2 input device:<br>0 = IBM 2400 Series Magnetic Tape Unit<br>1 = IBM 2540, 1442-N1, or 2520-B1 Card Read-Punch or 2501 Card Reader<br>5 Always 0.<br>6-8 The three-digit hexadecimal address of the Phase 2 input device. |
| 1 | 2 | 3 | OUTDEV | DC | X'zzzzzzzz' | zzzzzzzz represents eight hexadecimal digits generated in the following way:<br><br>_Digits_ _Description_<br>1-2 Always 0 for Single-Phase Dump and Phase 2 of Two-Phase Dump. For Phase 1 of Two-Phase Dump, if the output device is a tape unit with the 7-track or dual density feature, the mode set desired may be used.[1] Otherwise, these digits are 00.<br>3 Always 0.<br>4 For the Single-Phase Dump Program and Phase 2 of the Two-Phase Dump Program, this digit specifies the type of output device used to produce listings.<br>0 = IBM 1403 or 1443-2 Printer<br>1 = IBM 1052 Printer-Keyboard<br>For Phase 1 of the Two-Phase Dump Program, this digit specifies the type of output device used by Phase 1.<br>0 = IBM 2400 Series Magnetic Tape Unit<br>1 = IBM 2540 Card Read-Punch or 2520-B2 or B3 Card Punch<br>2 = IBM 1442-N1 Card Read-Punch<br>3 = IBM 2520-B1 Card Read-Punch<br>5 Always 0.<br>6-8 The three-digit hexadecimal address of the output device. |

[1] For a discussion of the 7-track feature and dual density feature, see _IBM 2400 and 2816 Model 1 Component Description_, Form A22-6866.

Figure 74. Dump Program Initialization Cards (Part 1 of 2)

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | TYPWTR | DC | X'zzzz' | zzzz represents a 0 followed by the three-digit hexadecimal address of the printer or printer-keyboard used to produce operator messages. If neither device is available, the operand must be specified as X'FFFF'. Placing hexadecimal F's in TYPWTR only informs the dump program that no IBM 1052 Printer-Keyboard is available and does not disable the Input/Output Routine from trying to print error messages. There are two ways to disable printing by the I/O routines.<br>1. Prior to assembly time, remove the Write Error Message Base Routine module from the I/O routines.<br>2. At object time, insert a Replace card to put the LPSW instruction at SAGINW+4 (in the I/O Base Routine - Group 1, Interrogate I/O Interrupt or Condition Code 1 module) back in the same form as on the assembly listing after it has been overlaid by the branch instruction in the Write Error Message Base Routine module. |

Figure 74. Dump Program Initialization Cards (Part 2 of 2)

The other method consists of assembling the single-phase dump and user programs separately. In this case, the respective assembled object programs must be loaded into storage for execution by the Relocating Loader. Note that during the load process, the dump program should precede the user's program into storage so that the loader can establish the proper linkages.

## THE TWO-PHASE DUMP PROGRAM

The Two-Phase Dump Program produces listings of the contents of registers and/or main storage. The program consists of two phases, Phase 1 and Phase 2.

Phase 1 is designed to produce card image records (on punched cards or magnetic tape) of the contents of registers and/or storage. When used, it resides in storage along with the user's program. Figure 75 shows the relationship between Phase 1 and the user's program.

## INITIALIZATION OF THE TWO-PHASE DUMP PROGRAM

The Two-Phase Dump Program is available from IBM as an assembled relocatable object deck for Phase 1 and as an assembled nonrelocatable deck (self-loading deck) for Phase 2. It is also available as optional material in symbolic form for both phases. Before the program can be used, each phase may require modification for operation on the installation's machine. This modification consists of altering three constants near the end of each phase in the IBM-supplied programs or punching information in the END card in the case of Phase 2 assembled nonrelocatable deck.

The constants in question are shown in Figure 74. Constants to be modified in the Phase 1 program are identified by the number 2 in column two on the left-hand side of the figure. Constants to be modified in the Phase 2 program are identified by the number 3 in column three. One must ensure that these constants properly describe the installation's machine configuration before assembling the Phase 1 or Phase 2 source decks or by altering the assembled relocatable decks with Replace cards at object time or by punching information in the END card of the Phase 2 self-loading deck. Note that a card's position in the source deck should not be altered during the modification process.

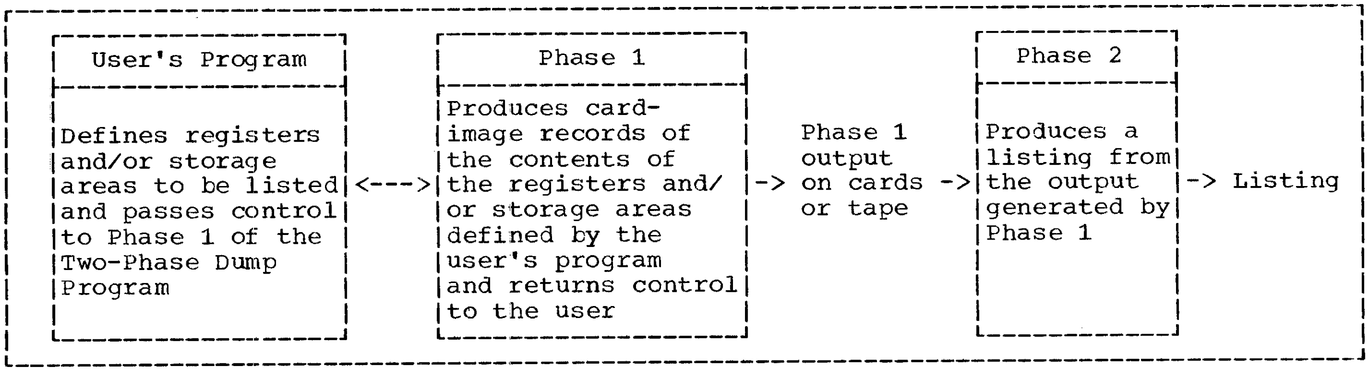


Figure 75. The Two-Phase Dump Program

USING THE TWO-PHASE DUMP PROGRAM

Each phase of the Two-Phase Dump Program has its own set of operating procedures. These procedures are described in the following text.

Phase 1

Phase 1 is used in essentially the same way as the Single-Phase Dump Program (see the topic Using the Single-Phase Dump Program). The two differ only with respect to their output. Phase 1 produces card or tape output for subsequent use by Phase 2. The Single-Phase Dump Program produces listings. Note that if tape is used for output, this tape is only rewound at end-of-reel by Phase 1. This enables the user to place the dump output of more than one program on a single reel for later processing by Phase 2.

Phase 2

Phase 2 produces listings of the contents of registers and/or storage from the output generated by Phase 1.

Phase 2, as supplied by IBM, is a self-loading version. To use the self-loading deck, the following 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:

| Cols. | Description |
|---|---|
| 17-20 | The address of the output device -- a printer or an IBM 1052 Printer-Keyboard. |
| 21 | 0 if a printer is to be used.<br>1 if an IBM 1052 Printer-Keyboard is to be used. |
| 22-25 | Address of the input device to be used. |
| 26 | 0 if the input device is a tape unit.<br>1 if the input device is a card reader. |
| 27-30 | The address of the IBM 1052 Printer-Keyboard if one is available.<br>FFFF if no IBM 1052 Printer-Keyboard is available. |
| 31-32 | If the input device is a tape unit with the 7-track feature and a mode set was used to create the tape, the same mode set must be punched in columns 31 and 32. Otherwise, leave blank. |

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

To use Phase 2 of the Two-Phase Dump Program in its self-loading version, the following steps must be performed:

1. Run cards out of the card reader.
2. Place the properly initialized

self-loading deck of Phase 2 in the card-read hopper.

3. Place the Phase 1 output on the appropriate unit. This unit address was defined to Phase 2 in the END card. If tape is used, the tape will be rewound at the beginning of execution. If card reader was used for Phase 1 output, this output should be followed by at least one blank card to ensure that the last listing will print.

4. Set the load unit switches on the system control panel to address the card reader and press the load key.

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. The assembled deck can then be loaded by either the absolute or relocating loader. In order to use Phase 2 in this form (an assembled relocatable version), the following are required:

1. A properly prepared and assembled Phase 2 program.
2. The output generated by Phase 1. The output can be on cards or tape.
3. A self-loading loader on cards.

To execute Phase 2 in its assembled relocatable version, perform the following steps:

1. Run cards out of the card reader.
2. Prepare the self-loading Absolute or Relocating Loader to read from the device containing Phase 2 of the two-phase dump program. The method of initialization is described in the section <u>The Absolute and Relocating Loaders</u>.
3. Place the self-loading loader in the card read hopper.
4. Place Phase 2 of the two-phase dump program on the appropriate device.
5. Place the Phase 1 output on the appropriate unit. This unit address was defined to Phase 2 during the Phase 2 initialization process (see the topic <u>Initializing the Two-Phase Dump Program</u>). If tape, the tape will be rewound at the beginning of Phase 2. If card, Phase 1 output should be followed by at least one blank card to ensure that the last listing will print.
6. Set the load-unit switches on the system control panel to address the card reader, and press the load key.

## THE ABSOLUTE AND RELOCATING LOADERS

Two load programs are available from IBM: the Absolute Loader and the Relocating Loader. Both are designed to load assembled programs (from cards or tape) into storage for execution. The Absolute Loader is available in two versions: one is assembled to occupy lower storage and the other to occupy higher storage on an 8K configuration. Both versions of the Absolute Loader are available in non-relocatable assembled self-loading decks. The Relocating Loader is available in a non-relocatable assembled self-loading deck to occupy lower storage. Certain installations may want loaders that reside elsewhere in storage and/or disable the printing of I/O error messages. For these reasons, both loaders are available from IBM in symbolic form as optional material. See the description of the Loader Generator program for information on generating self-loading loaders. (Use of the Relocating Loader is recommended for users with greater than 8K main storage.)

### PREPARING THE LOADERS FOR USE

The Absolute and Relocating Loaders are available from IBM in self-loading form on punched cards. Before either program can be used, it may require modification for operation on the installation's machine. This modification consists of altering the program's END card.

The END card is the last card in the deck. It must include the following:

| Cols. | Description |
|---|---|
| 17-20 | Blank if the program to be loaded is on the same device as the loader.<br>0 followed by the three-digit hexadecimal address of the unit from which the program is to be loaded if it is on a different unit from the loader. |
| 21-24 | 0 followed by the three-digit hexadecimal address of the installation's printer or printer-keyboard (used to produce operator messages).<br>Blank if neither device is available. |

## LOADER OPTIONS AND MODIFICATIONS

The loader source programs available from IBM are designed for residence in lower storage, beginning at location 144. The programs can be broken into the following general groups:

Introduction
I/O Routines
Loader Routines
Initial Entry Routine (IER)

They are organized as such so that the user can overlay the Initial Entry Routine with the beginning or end of his program, if he wishes. After loading, he can overlay the loader routines during execution and still use the loader's I/O routines if he is exercising that option.

If the loaders are to be modified for residence in higher storage, it is recommended that the groups be reorganized to make optimum use of available storage, as described below.

To modify the Absolute Loader for residence in upper storage, the following alterations to the source deck are necessary:

1. Remove the constant IOTA EQU * from the end of the deck. Insert the constant IOTA EQU *-160 in the beginning of the deck, in place of the comment card *IOTA EQU *-160.
2. Move the Initial Entry Routine from the end of the deck to the position specified by the comments following the new constant IOTA.
3. Move the Loader Routines (Hex-Bin Conversion Routine through the end of Constants Area) to precede the I/O Routines. The constants THE END and OMEGA should now precede the END card.
4. Alter the START card to the desired starting address of the new loader.
5. Assemble the modified deck and generate a self-loading deck using the LDRGEN program.

To modify the Relocating Loader for residence in upper storage, the following alterations to the source deck are necessary:

1. Remove the constant IOTA EQU * from the end of the deck. Insert the constant IOTA EQU *-160 in the beginning of the deck, in place of the comment card *IOTA EQU *-160.
2. Move the Initial Entry Routine from the end of the deck to the position specified by the comments following the new constant IOTA.
3. Move the Loader Routine (Hex-Bin Conversion Routine through end of constants Area) to precede the I/O Routines. The constant OMEGA should still precede the END card.
4. In the section of the Loader Routines, <u>Routine to Search Reference Table...</u> (found in the program listing), repunch the card:

```
*ST   12,BELOW
```

to delete the asterisk. Replace the new card in the source deck.

5. Change the existing constants TOP, BELOW, and CTRSET to the following:

```
TOP         EQU    MON
BELOW       DC     A(LOAD2)
CTRSET      DC     XL4'80'
```

6. Alter the START card to the desired starting address of the new loader.
7. Assemble the modified deck and generate a self-loading deck using the LDRGEN program.

## LOADING CAPACITY

The Relocating Loader available from IBM is set for a maximum storage size of 8K. To modify the Relocating Loader designed for residence in lower storage for a larger storage size than 8K, it is necessary to alter the constant TOP; this constant may be altered as described in the listing (the description of this alteration occurs just before the actual constant), or it may be altered to 131071 for 128K storage. The source deck should then be assembled and a new loader generated using the LDRGEN program.

## USING THE LOADERS

To load an assembled program into storage for execution, the following two items are required:

1. An Absolute or Relocating Loader in self-loading form on punched cards.
2. The assembled program to be loaded. The program may exist on punched cards or magnetic tape.

To run a job, perform the following steps:

1. Run cards out of the card reader.
2. Place the Absolute or Relocating Loader in the reader hopper. The loader must be initialized as described under <u>Preparing the Loaders for Use</u>.
3. Place the assembled program on the input unit from which it is to be loaded.
4. Set the load-unit switches on the system control panel to address the card reader, and press the load key.

## LOADER GENERATOR PROGRAM

The self-loading Absolute and Relocating Loaders available from IBM reside in lower storage during execution (higher storage in an 8K configuration). They are not in a form suitable for relocation. Since installations may want self-loading loaders that reside elsewhere in storage, IBM supplies a means to create them. This involves the use of the IBM-supplied Loader Generator (LDRGEN) program.

IBM provides both the Absolute and Relocating Loaders in symbolic form on punched cards. To create a self-loading loader, one must assemble the associated symbolic deck. The assembled loader is then loaded into storage with the LDRGEN program.

The LDRGEN program is designed to regenerate assembled loaders into a form suitable for direct loading into storage -- that is, to make them self-loading. Figure 76 shows the sequence of operations required to produce a self-loading loader.

## PREPARING THE LDRGEN DECK FOR ASSEMBLY

The LDRGEN program as supplied by IBM is in symbolic form as optional material on punched cards. Before the LDRGEN source deck can be assembled for use, the address of the card reader-punch upon which the self-loading loaders are to be written must be defined. This is accomplished by inserting an Equate card in the LDRGEN source deck just before the END card. The format of the Equate card is:

| Name | Operation | Operand |
|---|---|---|
| OUTPUT | EQU | A decimal or hexadecimal self-defining value equivalent to the address of the output unit. |

Once the Equate card has been inserted in the deck, the LDRGEN program can be assembled.

## RUNNING A JOB

In order to produce a self-loading loader, both the assembled loader (to be regenerated in self-loading form) and the assembled LDRGEN program must be loaded into storage. Since neither is self-loading, a separate load program must be used. Neither of these programs can overlay the self-loading loader used to load them. Two such programs are available in self-loading form: the Absolute Loader and the Relocating Loader. The use of each is described in the following text.

```
┌──────────────────────────────────────────────────────────────────────────────────┐
│                                                                                  │
│            ┌────────────┐                  ┌────────────┐                        │
│  Loader    │ Assembler  │      Assembled   │  LDRGEN    │      Self-Loading      │
│  Source -->│  Program   │-->    Loader  -->│  Program   │-->      Loader         │
│  Program   │            │      on Cards    │            │        on Cards        │
│            └────────────┘                  └────────────┘                        │
│                                                                                  │
└──────────────────────────────────────────────────────────────────────────────────┘
```

Figure 76. The LDRGEN Program

Using the Absolute Loader

Since the Absolute Loader loads programs into the storage locations assigned to them by the assembler, care must be taken to ensure that none of the programs to be loaded overlays another.

To use the Absolute Loader, one must have:

1. The Absolute Loader in self-loading form.
2. An assembled LDRGEN program.
3. The assembled loader to be regenerated in self-loading form. Note that the storage locations at which the loader was assembled are the ones assigned to the self-loading loader produced by the LDRGEN program.
4. Several Replace cards. These cards replace data in the LDRGEN program. They define addresses in the assembled loader and, if applicable, specify the number of duplicate self-loading loader decks to be produced by the LDRGEN program. The addresses in the assembled loader that they specify and the places in the LDRGEN program at which these addresses are inserted are shown in the following lists.

| Address of Symbol in Assembled Loader | Inserted at Symbolic Location in LDRGEN Program |
|---|---|
| ALPHA | ALPHAA |
| ALPHA | ALPHAB |
| BETA | BETAA |
| IOTA | IOTAA |
| OMEGA | OMEGAA |

If duplicate self-loading decks are desired, a Replace card is used to insert the number of duplicates desired in a half-word area in the LDRGEN program named CON.

To run a job, the self-loading Absolute Loader is placed in the card read hopper. The assembled loader is placed behind it. The Replace cards are inserted in the assembled LDRGEN deck immediately after the Text cards, and the entire deck is placed behind the assembled loader in the card reader hopper. The card reader-punch upon which the self-loading loader is to be written is prepared for use. Then the load-unit switches on the system control panel are set to address the card reader, and the load key is pressed.

Using the Relocating Loader

Since the Relocating Loader loads programs into storage at the locations specified by Set Location Counter (SLC) cards, care must be taken when specifying these cards so as to ensure that the programs to be loaded do not overlay one another. SLC cards are described in the Basic Utility Programs section.

To use the Relocating Loader, one must have:

1. The Relocating Loader in self-loading form.
2. An assembled LDRGEN program.
3. The assembled loader to be regenerated in self-loading form. Note that the storage locations into which this program is loaded by the Relocating Loader are the ones assigned to the self-loading loader produced by the LDRGEN program.
4. A single Replace card, if duplicate self-loading decks are to be produced by the LDRGEN program. The Replace card inserts the number of duplicates in a half-word area in the LDRGEN program called CON. Note that the Replace cards that define addresses when the Absolute Loader is used are not required in this case. The Relocating Loader performs this function automatically.

To run a job, the self-loading Relocating Loader is placed in the card read hopper. The assembled loader is placed behind it. If applicable, the

Replace card (for duplicate decks) is inserted in the assembled LDRGEN deck immediately after the Text cards, and the entire deck is placed behind the assembled loader in the card read hopper. A Load Terminate card, with LDRGEN in columns 17-22, is then placed in the card read hopper. The card reader-punch upon which LDRGEN will write the self-loading program(s) is prepared for use. The load-unit switches on the system control panel are set to address the card reader and the load key is pressed.

## INPUT/OUTPUT SUPPORT PACKAGE

IBM supplies a group of routines designed to provide the programmer with the coding required to use input/output devices. This group of routines is called the Input/Output Support Package.

The routines are available in symbolic form. The use of the IBM-supplied decks is exclusively the task of the programmer and, therefore, will not be described in this publication. For detailed information on the Input/Output Support Package, refer to the Basic Utility Programs section of this manual.

A program wait occurs whenever the Basic Assembler or Basic Utility programs find it necessary to communicate with the operator. A program wait is indicated by the wait light on the system control panel.

When a program wait occurs, the three low-order bytes of the current PSW contain a three-character code, each character consisting of eight bits. This code identifies the reason for the program wait. This code can be displayed on the system control panel through use of the storage-select switch and the address switches. The storage-select switch is set to display the current PSW. The address switches are set to display the three low-order bytes in the PSW. Smaller System/360 models may display only the last byte or the last two bytes.

The first character of the code identifies the program being executed when the program wait occurred. The characters and the programs with which they are associated are shown in the following:

| Character | Program |
|---|---|
| A | Assembler (both phases) |
| 1 | Assembler (Phase 1 only) |
| 2 | Assembler (Phase 2 only) |
| D | Dump Programs |
| G | Loader Generator |
| I | I/O Support Package |
| L | Loaders |

The third character of the code can be one of the following:

A Operator action is necessary. No decision on the part of the operator is required.

D Operator action is necessary. The operator must, however, make a decision on the course of action to be taken.

S A program wait has occurred because of a machine error. The job cannot continue. SEREP interface has been set up, and the SEREP program should be loaded and executed. Save the SEREP[1] printout for Field Engineering analysis. If attention is required, Field Engineering should be notified. Once SEREP has completed its processing, the operator must re-initialize the system to rerun the error-interrupted job or to proceed with the next job.

W A program wait has occurred because of a program check. The job cannot continue.

I Operator information only

If the installation has provisions to print operator messages, the three-character code is printed on the output device. In some cases, the code is followed by a descriptive message. In others, it is followed by a string of hexadecimal characters which define the conditions that exist as the result of an erroneous I/O operation.

The following is a list (in alphabetical order) of all possible message codes and their hexadecimal equivalents. It is provided to enable easy translation of the display on the system control panel into the proper message code.

| Message Code | Hexadecimal Equivalent | Message Code | Hexadecimal Equivalent |
|---|---|---|---|
| AIA | C1C9C1 | I0D | C9F0C4 |
| AID | C1C9C4 | I0S | C9F0E2 |
| AIS | C1C9E2 | I1D | C9F1C4 |
| AMS | C1D4E2 | I1S | C9F1E2 |
| APW | C1D7E6 | I3S | C9F3E2 |
| DEA | C4C5C1 | LAA | D3C1C1 |
| DRA | C4D9C1 | LDA | D3C4C1 |
| DTA | C4E3C1 | LED | D3C5C4 |
| GCS | C7C3E2 | LKA | D3D2C1 |
| GDD | C7C4C4 | LOA | D3D6C1 |
| GDS | C7C4E2 | LPA | D3D7C1 |
| GEA | C7C5C1 | LUA | D3E4C1 |
| GIA | C7C9C1 | 1EI | F1C5C9 |
| GMS | C7D4E2 | 2EI | F2C5C9 |
| GNS | C7D5E2 | 2HA | F2C8C1 |
| IMS | C9D4E2 | 2SA | F2E2C1 |

The program waits and messages presented in the following paragraphs are grouped according to the programs with which they are associated. Note that the I/O support package is built into each of the utility programs. Therefore, program waits listed under the I/O support package

---

[1] SEREP (System Environment Recording, Editing, and Printing) provides Field Engineering with detailed, accurate information about the system's environment at the time of a machine failure.

can occur during the execution of any of the utility programs. Where "Not typed" appears in parentheses after the three-character code, the code is displayed in the PSW but not typed on the output device.

## TWO PHASE DUMP PROGRAM

### DEA END OF DUMP-PHASE 2

Phase 2 of the two-phase dump program has been completed.

Action: Proceed with the next job.

### DRA MT NEXT INPUT REEL

Phase 2 of the two-phase dump program has encountered an end-of-reel condition on its input tape. The reel has been rewound and unloaded.

Action: Mount next reel on the input unit and make the device ready. Then, press the interrupt key on the system control panel to proceed with the job.

### DTA MT NEW OUTPUT REEL

Phase 1 of the two-phase dump program has encountered an end-of-reel condition on its output tape. The reel has been rewound and unloaded.

Action: Mount a new work tape on the output unit. Then, press the interrupt key on the system control panel to proceed with the job.

## SELF-LOADING DUMP PROGRAM

### DEA END OF DUMP

The self-loading dump program has been completed.

Action: Proceed with the next job.

## THE BASIC ASSEMBLER

### AIA (not typed)

The assembler has detected an I/O error which can be retried.

Action: Continue the program by depressing the Interrupt key. If after five retries the error still exists, load and execute SEREP.

### AID (not typed)

The assembler is unable to properly perform an I/O operation. The address of the associated I/O unit is contained in the low-order 11 bits of general register 2.

Action: The action taken varies with the type of operation in error.

- Tape Operation - Core location 44 hexadecimal (CSW unit status) should be interrogated.
    1. If the unit exception bit (bit 7) is set, an end-of-file condition on input or an end-of-reel condition on output has occurred. The address of the device causing the unit exception will be located in the lower half of register 2.

The operator should change that tape and press the interrupt key to continue the job.

2. If the unit exception bit is not set, the operator should press the interrupt key to retry the operation. If after five retries the condition still exists, the operator should dump all of storage and discontinue the job.

- Read - If a reader check light is on, the cards in the reader should be run out and reloaded. The operator should then press the interrupt key to retry the operation. If after one retry the condition still exists, the operator should mark the card in error and discontinue the job.
- Punch - Rerun the job.
- Write Line - Press interrupt key to repeat operation.
- Space or Eject - Press interrupt key to repeat operation.

AIS (not typed)

The assembler has detected an equipment failure while trying to execute an I/O operation. SEREP interface has been set up.

Action: Load and execute SEREP.

AMS (not typed)

A machine check has occurred.

Action: Load and execute SEREP.

APW (not typed)

A program check has occurred. The assembler program has been altered in some way.

Action: Dump all of storage and compare against listing to find the area altered. Correct if possible and rerun the job.

1EI

Phase 1 of the assembler has been completed.

Action: Proceed with Phase 2.

2EI

Phase 2 of the assembler has been completed.

Action: Proceed with next job.

2HA

Phase 2 of the assembler requires that blank cards be placed in the 1442-N1 or 2520-B1 card hopper.

Action: Remove any cards in the 1442 card hopper, insert blank cards, and replace the cards just removed.

2SA

Phase 2 of the assembler requires a blank card at the punch station or blank cards in the 1442-N1 or 2520-B1 card hopper.

Action: Remove cards from the hopper, run cards out of the 1442-N1 or 2520-B1, and place them at the bottom of the cards just removed from the hopper. Place blank cards in the hopper and place the cards removed from the 1442-N1 or 2520-B1 on top of the blanks. Make the unit ready and press the interrupt key on the system control panel to continue.

## THE ABSOLUTE AND RELOCATING LOADERS

Several of the program waits associated with the load programs concern load control cards. References to these cards are made in abbreviated form in the descriptions that follow. The abbreviated titles and their equivalent names are given in the following list:

| | |
|---|---|
| ESD | External Symbol Dictionary card |
| ICS | Include Segment card |

| | |
|---|---|
| LDT | Load Terminate card |
| REP | Replace card |
| RLD | Relocation List Dictionary card |
| SLC | Set Location Counter card |

Note: The preceding cards are described in the Basic Utility Programs section.

### LAA WAIT

The relocating loader has encountered an invalid RLD or ESD card in the program being loaded.

Action: Mark card and discontinue job.

### LDA WAIT

The relocating loader has encountered duplicate entry points in the program being loaded.

Action: Discontinue job.

### LED WAIT

One of the following situations has occurred:

1. The relocating loader has encountered an end-of-file condition without having read an LDT card.
2. The absolute loader has encountered an end-of-file condition without having read an END card.

Action: Discontinue job if the program is being loaded from tape. If the program is being loaded from cards, make the reader not ready. A card with a 12-2-9 punch in column one and the characters END or LDT (whichever is appropriate) in columns two through four is then placed in the reader hopper. The device is made ready and the interrupt key on the system control panel is pressed.

Note: The programmer should have included the proper LDT or END card in his source program. The operator action described in the preceding does not guarantee proper execution of the user's program.

### LKA (not typed)

The absolute or relocating loader has encountered an invalid SLC, ICS, or REP card in the program being loaded. This message is displayed but not typed for an invalid hexadecimal character.

Action: Mark card and discontinue job.

### LOA WAIT

An attempt has been made to load a program into main storage locations reserved for use by the absolute or relocating loader.

Action: Discontinue job.

### LPA (not typed)

A program check has occurred. Note that this wait can occur during the execution of any program loaded into storage by either the Absolute or Relocating Program Loader.

Action: Discontinue job.

### LUA WAIT

The relocating loader has encountered an undefined symbol in an SLC, ESD type 2, or LDT card in the program being loaded.

Action: Mark card and discontinue job.

## INPUT/OUTPUT SUPPORT PACKAGE

The Input/Output Support Package is used by the IBM-supplied utility programs and by the programmer. In the case of the utility programs, the I/O package is built in prior to their distribution.

When the input/output support routines are unable to properly execute an I/O operation, a program wait occurs to notify the operator of the unusual condition, and SEREP Interface is set up. An operator message accompanies the program wait if the installation has provisions for printing messages.

The Input/Output Support Package has three levels of messages. They are:

1. CCC
2. CCC IOOPSW CSW
3. CCC IOOPSW CSW SBYTES

where:

CCC
is the three-character code which identifies the reason for the message.

IOOPSW
is the contents of the old input/output program status word in hexadecimal notation. The channel and unit number of the I/O device in error is contained in bits 21-31 of this word.

CSW
is the contents of the channel status word associated with the operation in error. It is in hexadecimal notation.

SBYTES
is the contents of the six sense bytes in hexadecimal notation.

All three levels will only appear when the full complement of error message expansions is included. The Basic Utility Programs, other than the Basic I/O Support Package, contain only the first level.

IMS (not typed)

A machine check has occurred.

Action: Load and execute SEREP.

I0D IOOPSW CSW SBYTES

The input/output support package is unable to properly perform an I/O operation.

Action: The action taken varies with the operation in error:

- Tape - If unit is not ready, make ready and press console Interrupt to retry operation. If retry is unsuccessful, discontinue job.
- Punch Card - If the punch is not ready or out of cards, make it ready and press console Interrupt to retry the punch operation. If retry is unsuccessful, discontinue job.
- Read Card - If the reader is not ready or out of cards, make it ready and press console Interrupt to retry the read operation. If retry is unsuccessful, discontinue job.
- Write a Line - Press interrupt key to repeat the operation. If retry is unsuccessful, discontinue job.
- Skip or Space - Press interrupt key to repeat the operation. If retry is unsuccessful, discontinue job.

I0S IOOPSW CSW SBYTES

The input/output support package is unable to properly execute an operation. The standard retries have been attempted and the error persists.

Action: Load and execute SEREP.

I1D IOOPSW CSW SBYTES

One of the following has occurred:

1. A request to start an I/O operation has been rejected because of a programming error. In this case, the busy bit (bit 35) in the channel status word is off.
2. An overlapped I/O operation has been completed unsuccessfully while an attempt was being made to start a new operation. In this case, the busy bit in the channel status word is on.

Action: If the busy bit in the channel status word is off, press the interrupt key on the system control panel to repeat the request for an I/O operation. If the operation is again rejected, discontinue the job and call the customer engineer. If the busy bit in the channel status word is on, rerun the job.

I1S IOOPSW CSW SBYTES

One of the following has occurred:

1. A request to start an input/output operation has been rejected because of a machine error.

2. An overlapped I/O operation has been completed unsuccessfully while an attempt was being made to start a new operation.

Action: Load and execute SEREP.

I3S IOOPSW CSW SBYTES

The Input/Output Support Package attempted to use an I/O device which was not operational or not available.

Action: Load and execute SEREP.

## LOADER GENERATOR PROGRAM

GCS (not typed)

A channel error has occurred.

Action: Load and execute SEREP.

GDD (not typed)

The LDRGEN program has attempted to punch a card but the operation resulted in an error.

Action: Mark the erroneously punched card and press the interrupt key to repeat the operation.

GDS (not typed)

A device failure has occurred.

Action: Load and execute SEREP.

GEA (not typed)

The LDRGEN program has been executed. This is a normal end-of-job situation.

Action: Proceed with next job.

GIA (not typed)

The punch unit has run out of blank cards.

Action: Place blank cards in the punch hopper and press interrupt key to continue job.

GMS (not typed)

Machine check has occurred.

Action: Load and execute SEREP.

GNS (not typed)

The device specified as the output unit in the LDRGEN program is not available.

Action: Load and execute SEREP.

A sample Card Assembler and Utilities program is provided to test the Basic Assembler and Basic Utility Programs (Card) supplied to the user. The sample problem sorts, in ascending order, 16 full word constants located at address "IN" and stores them at address "OUT". The 16 sorted numbers are also printed on the output device, and a message is typed on the IBM 1052 Printer-Keyboard at the end of the run.

## Identifying the Card Deck

The sample program deck (Figure 77) consists of 72 source cards. The first card of the sample program is identified by:

| CARD COLUMNS | | | |
|---|---|---|---|
| | 32-35 | 38-39 | 73-80 |
| | ICTL | 25 | SMPL1030 |

Each source card contains SMPL, the program identifier, in columns 73-76, followed by the sequence number in columns 77-80. The last source card is identified by:

| CARD COLUMNS | | | |
|---|---|---|---|
| | 32-34 | 38-39 | 73-80 |
| | END | GO | SMPL1740 |

Sixteen data cards are included in the source deck as DC's. The first data card is identified by:

| CARD COLUMNS | | | |
|---|---|---|---|
| 25-26 | 32-33 | 38-48 | 73-80 |
| IN | DC | X'00000005' | SMPL1440 |

The last data card is identified by:

| CARD COLUMNS | | | |
|---|---|---|---|
| | 32-33 | 38-48 | 73-80 |
| | DC | X'00000008' | SMPL1590 |

## Running the Sample Problem

1. The sample problem is supplied with 16 full word hexadecimal constants starting at address "IN." If left in place and run as described here, these numbers will sort from 0 through 15. If the user wishes to sort 16 other numbers, he may replace the original numbers with his own. The first full word constant must be given the name "IN."

2. Assemble the sample program. Prepare Phase 1 and Phase 2 Configuration Cards as described in the Assembler Initialization section. Insert Phase 1 Configuration Card in the Phase 1 deck of the Basic Assembler and Phase 2 Configuration Card in the Phase 2 deck of the Basic Assembler.

3. If the system has a storage capacity of greater than 8K and the user desires to assemble the Dump Program source deck:

   a. Add to the Single-Phase Dump Program an ENTRY SINTRY statement, and

   b. Supply to the Dump Program

      - the address of the available output device (OUTDEV)

      - the address of the available IBM 1052 Printer-Keyboard (TYPWTR), and

      - the storage capacity of the computer (DSTOPL)

   See description in Basic Utility Programs.

or

If the system has a storage capacity of 8K, or if the user desires to use the Single-Phase Dump Program object deck supplied by IBM to avoid having to assemble:

a. Remove the Load End card (the last card) from the assembled Dump Program deck as supplied by IBM.

b. Using Replace cards, alter the constants OUTDEV, TYPWTR, and DSTOPL as described in the Basic Utility Programs section.

c. If the High Absolute Loader is used, remove the two symbolic address constants (DUMP and SINTRY + 12) from the IBM supplied Sample Problem. These two cards are identified by an * in column 71. Replace the address constants with the following two cards:

```
column 25
        ADDUMP DC A(X'90')
        ADSIN DC A(X'C9C')
```

d. If the High Absolute Loader is used, assemble the Sample Problem at starting address 1240 (hexadecimal).

4. Assemble the Dump Program, if the source deck is used.

5. Place in the card reader these cards in the following order:

   Loader Assembled Deck (see Note 1)

   Assembled Dump Program

   Assembled Sample Problem

   Load Terminate Card (see Note 2)

   Note 1: On an 8K system, the High Absolute Loader must be used to load the sample problem and Dump Program. However, on a system with greater storage capacity, the Relocating Loader may, if desired, be used instead.

   Note 2: SAMPLE must be in column 17-22. This card must be prepared by the user. See the Basic Utility Programs section.

6. Load and execute program.

7. At the end of the run, the Wait state will be entered and FF will be in the instruction address portion of the current PSW.

8. The output will be as follows:

   a. On the output device will be printed: the console listing and general registers, followed by the name "SORTDUMP", followed by the 16 numbers sorted in ascending order.

   b. On the IBM 1052 Printer-Keyboard the following message will be typed: "End of Sample Problem Demonstration".

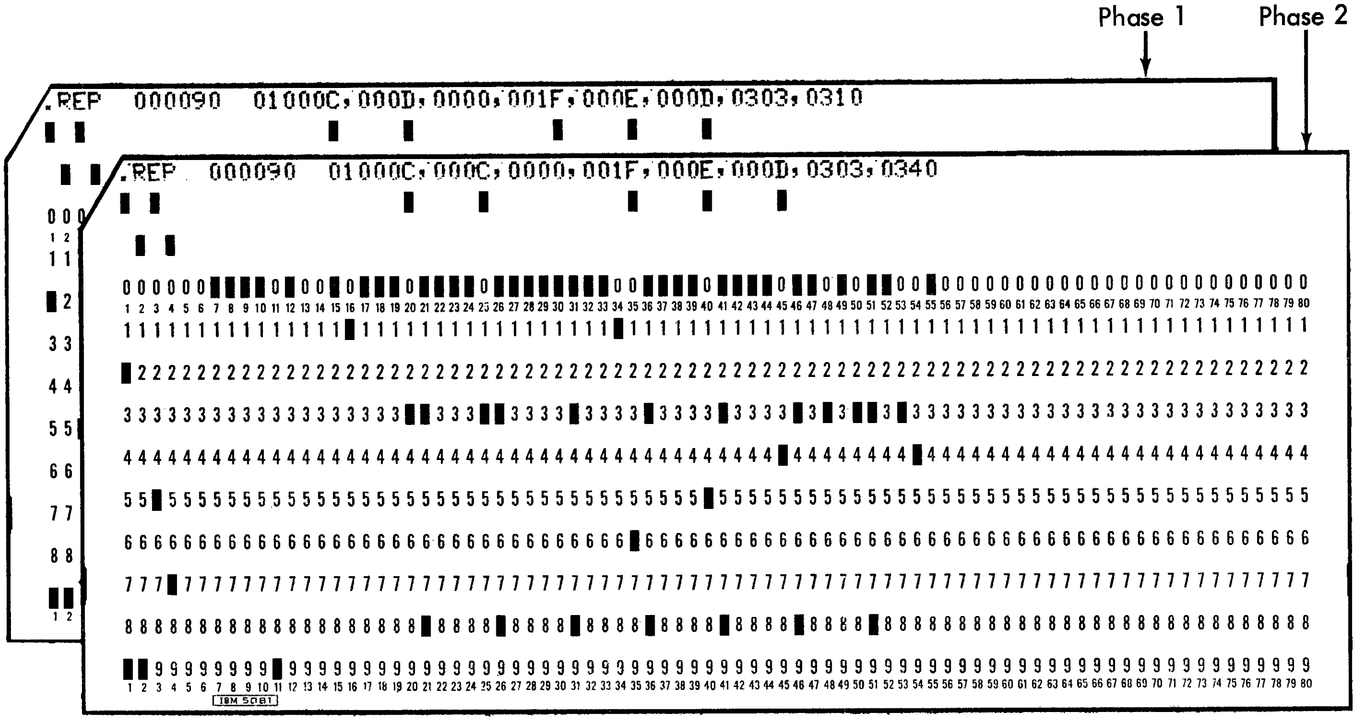
Figure 78 shows (for illustrative purposes only) the Configuration Cards used to assemble the Sample program. The user must prepare his own Configuration Cards in order to tailor the Basic Assembler program for operation at his own installation and to print or suppress program listings, or to print only error listings.

Figure 79 shows the Sample Problem output as produced using the Single-Phase Dump Program object deck and the High Absolute Loader.

```
         ICTL  25                                      SMPL1030
SAMPLE   START 0      STARTING ADDR                    SMPL1040
         EXTRN DUMP  DEFINE DUMP                       SMPL1050
         EXTRN SINTRY        DEFINE SINTRY             SMPL1060
GO       BALR  13,0  SET UP BASE REGISTER              SMPL1070
         USING *,13                                    SMPL1080
         MVC   OUT(64),IN      MOVE DATA TO OUT        SMPL1090
         LA    6,OUT         POINT TO TABLE TOP        SMPL1100
         LA    7,15          SET FOR 15 PASSES         SMPL1110
SET      LA    4,56          SET INDEX REGISTER        SMPL1120
         L     2,0(0,6)      LOAD FROM TABLE TOP       SMPL1130
LOAD     L     3,4(4,6)      LOAD FROM TABLE           SMPL1140
         CLR   2,3           COMPARE VALUES            SMPL1150
         BC    12,SUB        TOP = OR LESS BRANCH      SMPL1160
         XR    2,3           EXCHANGE VALUES           SMPL1170
         XR    3,2           EXCHANGE VALUES           SMPL1180
         XR    2,3           EXCHANGE VALUES           SMPL1190
         ST    3,4(4,6)      STORE LARGER BACK         SMPL1200
SUB      S     4,CON4        REDUCE INDEX              SMPL1210
         BC    10,LOAD       LOOP IF MORE TO SORT      SMPL1220
         ST    2,0(0,6)      STORE IN TABLE TOP        SMPL1230
         S     7,CON1        REDUCE PASS COUNTER       SMPL1240
         BC    7,LOOP                                  SMPL1250
         L     15,ADDUMP     CALL DUMP PROGRAM         SMPL1260
         BALR  14,15                                   SMPL1270
         DC    X'C05001'     DUMP CALL PARAMETERS      SMPL1280
         DC    AL3(LIST)                               SMPL1290
         L     1,ADSIN       ADDR OF TYPWTR            SMPL1300
         L     1,0(1)                                  SMPL1310
         LA    2,ENDMSG      ADDR OF MSG               SMPL1320
         LA    3,35          SIZE OF MSG               SMPL1330
         LA    4,UNEX        UNIT EXCEPTION ADDR       SMPL1340
         BALR  0,1                                     SMPL1350
UNEX     LPSW  ENDJOB        END OF JOB                SMPL1360
LOOP     LA    6,4(6)                                  SMPL1370
         LH    2,SET+2       MODIFY                    SMPL1380
         S     2,CON4         INDEX                    SMPL1390
         STH   2,SET+2          INSTRUCTION            SMPL1400
         BC    15,SET        RETURN                    SMPL1410
CON1     DC    F'1'          CONSTANT OF 1             SMPL1420
CON4     DC    F'4'          CONSTANT OF 4             SMPL1430
IN       DC    X'00000005'                             SMPL1440
         DC    X'0000000A'                             SMPL1450
         DC    X'00000001'                             SMPL1460
         DC    X'00000007'                             SMPL1470
         DC    X'00000003'                             SMPL1480
         DC    X'0000000C'                             SMPL1490
         DC    X'0000000F'                             SMPL1500
         DC    X'00000009'                             SMPL1510
         DC    X'0000000B'                             SMPL1520  Data Cards
         DC    X'00000004'                             SMPL1530
         DC    X'00000000'                             SMPL1540
         DC    X'0000000E'                             SMPL1550
         DC    X'00000006'                             SMPL1560
         DC    X'0000000D'                             SMPL1570
         DC    X'00000002'                             SMPL1580
         DC    X'00000008'                             SMPL1590
OUT      DS    16F           OUTPUT AND WORK AREA      SMPL1600
         DS    0D            BOUNDARY ALIGNMENT        SMPL1610
ENDJOB   DC    X'00020000'   PSW WITH WAIT BIT         SMPL1620
         DC    X'000000FF'                             SMPL1630
ENDMSG   DC    C'END OF SAMPLE PR'  TYPWTR MSG         SMPL1640
         DC    C'OBLEM DEMONSTRAT'                     SMPL1650
```

```
         DC    C'ION'                                  SMPL1660
ADSIN    DC    A(SINTRY+12)  TYPWTR ADDR             * SMPL1670
ADDUMP   DC    A(DUMP)       DUMP PROG ADDR          * SMPL1680
LIST     DC    X'CA'         DUMP CONTROL LIST         SMPL1690
         DC    AL3(OUT)                                SMPL1700
         DC    X'00'                                   SMPL1710
         DC    AL3(OUT+64)                             SMPL1720
         DC    C'SORTDUMP'                             SMPL1730
         END   GO                                      SMPL1740
```

Figure 77. Program Deck for Sample Problem



```
.REP  000090  01000C,000D,0000,001F,000E,000D,0303,0310
.REP  000090  01000C,000C,0000,001F,000E,000D,0303,0340
```

Figure 78. Phase 1 and Phase 2 Configuration Cards for Sample Problem

```
                                   ICTL  25                                  SMPL1030
001240                      SAMPLE START  X'1240'                            SMPL1040
                                   EXTRN DUMP  DEFINE DUMP                   SMPL1050
                                   EXTRN SINTRY      DEFINE SINTRY           SMPL1060
001240  05 D0               GO     BALR  13,0  SET UP BASE REGISTER          SMPL1070
                    001242         USING *,13                                SMPL1080
001242  D2 3F D 0C2 D 082          MVC   OUT(64),IN     MOVE DATA TO OUT     SMPL1090
001248  41 60 D 0C2                LA    6,OUT        POINT TO TABLE TOP     SMPL1100
00124C  41 70 0 00F                LA    7,15         SET FOR 15 PASSES      SMPL1110
001250  41 40 0 038         SET    LA    4,56         SET INDEX REGISTER     SMPL1120
001254  58 20 6 000                L     2,0(0,6)     LOAD FROM TABLE TOP    SMPL1130
001258  58 34 6 004         LOAD   L     3,4(4,6)     LOAD FROM TABLE        SMPL1140
00125C  15 23                      CLR   2,3          COMPARE VALUES         SMPL1150
00125E  47 C0 D 02A                BC    12,SUB       TOP = OR LESS BRANCH   SMPL1160
001262  17 23                      XR    2,3          EXCHANGE VALUES        SMPL1170
001264  17 32                      XR    3,2          EXCHANGE VALUES        SMPL1180
001266  17 23                      XR    2,3          EXCHANGE VALUES        SMPL1190
001268  50 34 6 004                ST    3,4(4,6)     STORE LARGER BACK      SMPL1200
00126C  5B 40 D 07E         SUB    S     4,CON4       REDUCE INDEX           SMPL1210
001270  47 A0 D 016                BC    10,LOAD      LOOP IF MORE TO SORT   SMPL1220
001274  50 20 6 000                ST    2,0(0,6)     STORE IN TABLE TOP     SMPL1230
001278  5B 7C D 07A                S     7,CON1       REDUCE PASS COUNTER    SMPL1240
00127C  47 70 D 064                BC    7,LOOP                              SMPL1250
001280  58 F0 D 136                L     15,ADDUMP    CALL DUMP PROGRAM      SMPL1260
001284  05 EF                      BALR  14,15                               SMPL1270
001286  C05001                     DC    X'C05001'    DUMP CALL PARAMETERS   SMPL1280
001289  00137C                     DC    AL3(LIST)                           SMPL1290
00128C  58 10 D 132                L     1,ADSIN      ADDR OF TYPWTR         SMPL1300
001290  58 11 0 000                L     1,0(1)                              SMPL1310
001294  41 20 D 10E                LA    2,ENDMSG     ADDR OF MSG            SMPL1320
001298  41 30 0 023                LA    3,35         SIZE OF MSG            SMPL1330
00129C  41 40 D 060                LA    4,UNEX       UNIT EXCEPTION ADDR    SMPL1340
0012A0  05 01                      BALR  0,1                                 SMPL1350
0012A2  82 00 D 106         UNEX   LPSW  ENDJOB       END OF JOB             SMPL1360
0012A6  41 66 0 004         LOOP   LA    6,4(6)                              SMPL1370
0012AA  48 20 D 010                LH    2,SET+2      MODIFY                 SMPL1380
0012AE  5B 2C D 07E                S     2,CON4        INDEX                 SMPL1390
0012B2  40 20 D 010                STH   2,SET+2        INSTRUCTION          SMPL1400
0012B6  47 F0 D 00E                BC    15,SET       RETURN                 SMPL1410
0012BA  0000
0012BC  00000001            CON1   DC    F'1'         CONSTANT OF 1          SMPL1420
0012C0  00000004            CON4   DC    F'4'         CONSTANT OF 4          SMPL1430
0012C4  00000005            IN     DC    X'00000005'                         SMPL1440
0012C8  0000000A                   DC    X'0000000A'                         SMPL1450
0012CC  00000001                   DC    X'00000001'                         SMPL1460
0012D0  00000007                   DC    X'00000007'                         SMPL1470
0012D4  00000003                   DC    X'00000003'                         SMPL1480
0012D8  0000000C                   DC    X'0000000C'                         SMPL1490
0012DC  0000000F                   DC    X'0000000F'                         SMPL1500
0012E0  00000009                   DC    X'00000009'                         SMPL1510
0012E4  0000000B                   DC    X'0000000B'                         SMPL1520
0012E8  00000004                   DC    X'00000004'                         SMPL1530
0012EC  00000000                   DC    X'00000000'                         SMPL1540
0012F0  0000000E                   DC    X'0000000E'                         SMPL1550
0012F4  00000006                   DC    X'00000006'                         SMPL1560
0012F8  0000000D                   DC    X'0000000D'                         SMPL1570
0012FC  00000002                   DC    X'00000002'                         SMPL1580
001300  00000008                   DC    X'00000008'                         SMPL1590
001304                      OUT    DS    16F          OUTPUT AND WORK AREA   SMPL1600
001348                             DS    0D           BOUNDARY ALIGNMENT     SMPL1610
001348  00020000            ENDJOB DC    X'00020000'  PSW WITH WAIT BIT      SMPL1620
00134C  000000FF                   DC    X'000000FF'                         SMPL1630
```

```
001350  C5D5C440D6C640E2    ENDMSG DC    C'END OF SAMPLE PR'  TYPWTR MSG     SMPL1640
001358  C1D4D7D3C540D7D9
001360  D6C2D3C5D440C4C5           DC    C'OBLEM DEMONSTRAT'                 SMPL1650
001368  D4D6D5E2E3D9C1E3
001370  C9D6D5                     DC    C'ION'                              SMPL1660
001373  00
001374  C0000C9C            ADSIN  DC    A(X'C9C')
001378  00000090            ADDUMP DC    A(X'90')
00137C  CA                  LIST   DC    X'CA'        DUMP CONTROL LIST      SMPL1690
00137D  001304                     DC    AL3(OUT)                            SMPL1700
001380  00                         DC    X'00'                               SMPL1710
001381  001344                     DC    AL3(OUT+64)                         SMPL1720
001384  E2D6D9E3C4E4D4D7           DC    C'SORTDUMP'                         SMPL1730
0C1240                             END   GO                                  SMPL1740
```

Figure 79. Assembly Listing for Sample Problem (Part 1 of 3)

```
    LOC 0        00004CD480000CA0              LOC 8       0000043000004AF4              LOC 16      020013E000001790
OLD PSWS         000000000000C9C1    0004000250000072A   040000054C8C4042    45901FE8D201104A    FF04000060001604
CSW   0000000004000000               CAW    00001ED0     00004CD4           TIMER  16139200      E5F2D4F2
NEW PSWS         00000000000018B8    0006000000E2E5C3    0106000000D3D7C1    0002000000C9D4E2    0000000000001814

GREGS  0 - 7        60001604    40001240    0000000E    0000000F    FFFFFFFC    00001719    0000133C    00000000
GREGS  8 - 15       00000080    00001700    02C5D5C4    0000000B    00001380    40001242    40001286    40000092

        SORTDUMP

001304   0000000000    0000000001    0000000002    0000000003    0000000004    0000000005    0000000006    0000000007

001324   0000000008    0000000009    0000000010    0000000011    0000000012    0000000013    0000000014    0000000015
```

Figure 79. Output Device Listing for Sample Problem (Part 2 of 3)

```
1EI




2EI






END OF SAMPLE PROBLEM DEMONSTRATION
```

Figure 79. 1052 Printer-Keyboard Message for Sample Problem (Part 3 of 3)

| 8-Bit BCD Code | Character Set Punch Combination | Decimal | Hexa-Decimal | Printer Graphics |
|---|---|---|---|---|
| 00000000 | 12,0,9,8,1 | 0 | 00 | |
| 00000001 | 12,9,1 | 1 | 01 | |
| 00000010 | 12,9,2 | 2 | 02 | |
| 00000011 | 12,9,3 | 3 | 03 | |
| 00000100 | 12,9,4 | 4 | 04 | |
| 00000101 | 12,9,5 | 5 | 05 | |
| 00000110 | 12,9,6 | 6 | 06 | |
| 00000111 | 12,9,7 | 7 | 07 | |
| 00001000 | 12,9,8 | 8 | 08 | |
| 00001001 | 12,9,8,1 | 9 | 09 | |
| 00001010 | 12,9,8,2 | 10 | 0A | |
| 00001011 | 12,9,8,3 | 11 | 0B | |
| 00001100 | 12,9,8,4 | 12 | 0C | |
| 00001101 | 12,9,8,5 | 13 | 0D | |
| 00001110 | 12,9,8,6 | 14 | 0E | |
| 00001111 | 12,9,8,7 | 15 | 0F | |
| 00010000 | 12,11,9,8,1 | 16 | 10 | |
| 00010001 | 11,9,1 | 17 | 11 | |
| 00010010 | 11,9,2 | 18 | 12 | |
| 00010011 | 11,9,3 | 19 | 13 | |
| 00010100 | 11,9,4 | 20 | 14 | |
| 00010101 | 11,9,5 | 21 | 15 | |
| 00010110 | 11,9,6 | 22 | 16 | |
| 00010111 | 11,9,7 | 23 | 17 | |
| 00011000 | 11,9,8 | 24 | 18 | |
| 00011001 | 11,9,8,1 | 25 | 19 | |
| 00011010 | 11,9,8,2 | 26 | 1A | |
| 00011011 | 11,9,8,3 | 27 | 1B | |
| 00011100 | 11,9,8,4 | 28 | 1C | |
| 00011101 | 11,9,8,5 | 29 | 1D | |
| 00011110 | 11,9,8,6 | 30 | 1E | |
| 00011111 | 11,9,8,7 | 31 | 1F | |
| 00100000 | 11,0,9,8,1 | 32 | 20 | |
| 00100001 | 0,9,1 | 33 | 21 | |
| 00100010 | 0,9,2 | 34 | 22 | |
| 00100011 | 0,9,3 | 35 | 23 | |
| 00100100 | 0,9,4 | 36 | 24 | |
| 00100101 | 0,9,5 | 37 | 25 | |
| 00100110 | 0,9,6 | 38 | 26 | |
| 00100111 | 0,9,7 | 39 | 27 | |
| 00101000 | 0,9,8 | 40 | 28 | |
| 00101001 | 0,9,8,1 | 41 | 29 | |
| 00101010 | 0,9,8,2 | 42 | 2A | |
| 00101011 | 0,9,8,3 | 43 | 2B | |
| 00101100 | 0,9,8,4 | 44 | 2C | |
| 00101101 | 0,9,8,5 | 45 | 2D | |
| 00101110 | 0,9,8,6 | 46 | 2E | |
| 00101111 | 0,9,8,7 | 47 | 2F | |
| 00110000 | 12,11,0,9,8,1 | 48 | 30 | |
| 00110001 | 9,1 | 49 | 31 | |
| 00110010 | 9,2 | 50 | 32 | |

| 8-Bit BCD Code | Character Set Punch Combination | Decimal | Hexa-Decimal | Printer Graphics |
|---|---|---|---|---|
| 00110011 | 9,3 | 51 | 33 | |
| 00110100 | 9,4 | 52 | 34 | |
| 00110101 | 9,5 | 53 | 35 | |
| 00110110 | 9,6 | 54 | 36 | |
| 00110111 | 9,7 | 55 | 37 | |
| 00111000 | 9,8 | 56 | 38 | |
| 00111001 | 9,8,1 | 57 | 39 | |
| 00111010 | 9,8,2 | 58 | 3A | |
| 00111011 | 9,8,3 | 59 | 3B | |
| 00111100 | 9,8,4 | 60 | 3C | |
| 00111101 | 9,8,5 | 61 | 3D | |
| 00111110 | 9,8,6 | 62 | 3E | |
| 00111111 | 9,8,7 | 63 | 3F | |
| 01000000 | | 64 | 40 | blank |
| 01000001 | 12,0,9,1 | 65 | 41 | |
| 01000010 | 12,0,9,2 | 66 | 42 | |
| 01000011 | 12,0,9,3 | 67 | 43 | |
| 01000100 | 12,0,9,4 | 68 | 44 | |
| 01000101 | 12,0,9,5 | 69 | 45 | |
| 01000110 | 12,0,9,6 | 70 | 46 | |
| 01000111 | 12,0,9,7 | 71 | 47 | |
| 01001000 | 12,0,9,8 | 72 | 48 | |
| 01001001 | 12,8,1 | 73 | 49 | |
| 01001010 | 12,8,2 | 74 | 4A | |
| 01001011 | 12,8,3 | 75 | 4B | . (period) |
| 01001100 | 12,8,4 | 76 | 4C | < |
| 01001101 | 12,8,5 | 77 | 4D | ( |
| 01001110 | 12,8,6 | 78 | 4E | + |
| 01001111 | 12,8,7 | 79 | 4F | |
| 01010000 | 12 | 80 | 50 | & |
| 01010001 | 12,11,9,1 | 81 | 51 | |
| 01010010 | 12,11,9,2 | 82 | 52 | |
| 01010011 | 12,11,9,3 | 83 | 53 | |
| 01010100 | 12,11,9,4 | 84 | 54 | |
| 01010101 | 12,11,9,5 | 85 | 55 | |
| 01010110 | 12,11,9,6 | 86 | 56 | |
| 01010111 | 12,11,9,7 | 87 | 57 | |
| 01011000 | 12,11,9,8 | 88 | 58 | |
| 01011001 | 11,8,1 | 89 | 59 | |
| 01011010 | 11,8,2 | 90 | 5A | |
| 01011011 | 11,8,3 | 91 | 5B | $ |
| 01011100 | 11,8,4 | 92 | 5C | * |
| 01011101 | 11,8,5 | 93 | 5D | ) |
| 01011110 | 11,8,6 | 94 | 5E | |
| 01011111 | 11,8,7 | 95 | 5F | |
| 01100000 | 11 | 96 | 60 | - |
| 01100001 | 0,1 | 97 | 61 | / |
| 01100010 | 11,0,9,2 | 98 | 62 | |
| 01100011 | 11,0,9,3 | 99 | 63 | |
| 01100100 | 11,0,9,4 | 100 | 64 | |
| 01100101 | 11,0,9,5 | 101 | 65 | |
| 01100110 | 11,0,9,6 | 102 | 66 | |
| 01100111 | 11,0,9,7 | 103 | 67 | |
| 01101000 | 11,0,9,8 | 104 | 68 | |
| 01101001 | 0,8,1 | 105 | 69 | |
| 01101010 | 12,11 | 106 | 6A | |
| 01101011 | 0,8,3 | 107 | 6B | , (comma) |

| 8-Bit BCD Code | Character Set Punch Combination | Decimal | Hexa-Decimal | Printer Graphics |
|---|---|---|---|---|
| 01101100 | 0,8,4 | 108 | 6C | % |
| 01101101 | 0,8,5 | 109 | 6D | |
| 01101110 | 0,8,6 | 110 | 6E | |
| 01101111 | 0,8,7 | 111 | 6F | |
| 01110000 | 12,11,0 | 112 | 70 | |
| 01110001 | 12,11,0,9,1 | 113 | 71 | |
| 01110010 | 12,11,0,9,2 | 114 | 72 | |
| 01110011 | 12,11,0,9,3 | 115 | 73 | |
| 01110100 | 12,11,0,9,4 | 116 | 74 | |
| 01110101 | 12,11,0,9,5 | 117 | 75 | |
| 01110110 | 12,11,0,9,6 | 118 | 76 | |
| 01110111 | 12,11,0,9,7 | 119 | 77 | |
| 01111000 | 12,11,0,9,8 | 120 | 78 | |
| 01111001 | 8,1 | 121 | 79 | |
| 01111010 | 8,2 | 122 | 7A | |
| 01111011 | 8,3 | 123 | 7B | # |
| 01111100 | 8,4 | 124 | 7C | @ |
| 01111101 | 8,5 | 125 | 7D | ' (quote) |
| 01111110 | 8,6 | 126 | 7E | = |
| 01111111 | 8,7 | 127 | 7F | |
| 10000000 | 12,0,8,1 | 128 | 80 | |
| 10000001 | 12,0,1 | 129 | 81 | |
| 10000010 | 12,0,2 | 130 | 82 | |
| 10000011 | 12,0,3 | 131 | 83 | |
| 10000100 | 12,0,4 | 132 | 84 | |
| 10000101 | 12,0,5 | 133 | 85 | |
| 10000110 | 12,0,6 | 134 | 86 | |
| 10000111 | 12,0,7 | 135 | 87 | |
| 10001000 | 12,0,8 | 136 | 88 | |
| 10001001 | 12,0,9 | 137 | 89 | |
| 10001010 | 12,0,8,2 | 138 | 8A | |
| 10001011 | 12,0,8,3 | 139 | 8B | |
| 10001100 | 12,0,8,4 | 140 | 8C | |
| 10001101 | 12,0,8,5 | 141 | 8D | |
| 10001110 | 12,0,8,6 | 142 | 8E | |
| 10001111 | 12,0,8,7 | 143 | 8F | |
| 10010000 | 12,11,8,1 | 144 | 90 | |
| 10010001 | 12,11,1 | 145 | 91 | |
| 10010010 | 12,11,2 | 146 | 92 | |
| 10010011 | 12,11,3 | 147 | 93 | |
| 10010100 | 12,11,4 | 148 | 94 | |
| 10010101 | 12,11,5 | 149 | 95 | |
| 10010110 | 12,11,6 | 150 | 96 | |
| 10010111 | 12,11,7 | 151 | 97 | |
| 10011000 | 12,11,8 | 152 | 98 | |
| 10011001 | 12,11,9 | 153 | 99 | |
| 10011010 | 12,11,8,2 | 154 | 9A | |
| 10011011 | 12,11,8,3 | 155 | 9B | |
| 10011100 | 12,11,8,4 | 156 | 9C | |
| 10011101 | 12,11,8,5 | 157 | 9D | |
| 10011110 | 12,11,8,6 | 158 | 9E | |
| 10011111 | 12,11,8,7 | 159 | 9F | |
| 10100000 | 11,0,8,1 | 160 | A0 | |
| 10100001 | 11,0,1 | 161 | A1 | |
| 10100010 | 11,0,2 | 162 | A2 | |
| 10100011 | 11,0,3 | 163 | A3 | |
| 10100100 | 11,0,4 | 164 | A4 | |

| 8-Bit BCD Code | Character Set Punch Combination | Decimal | Hexa-Decimal | Printer Graphics |
|---|---|---|---|---|
| 10100101 | 11,0,5 | 165 | A5 | |
| 10100110 | 11,0,6 | 166 | A6 | |
| 10100111 | 11,0,7 | 167 | A7 | |
| 10101000 | 11,0,8 | 168 | A8 | |
| 10101001 | 11,0,9 | 169 | A9 | |
| 10101010 | 11,0,8,2 | 170 | AA | |
| 10101011 | 11,0,8,3 | 171 | AB | |
| 10101100 | 11,0,8,4 | 172 | AC | |
| 10101101 | 11,0,8,5 | 173 | AD | |
| 10101110 | 11,0,8,6 | 174 | AE | |
| 10101111 | 11,0,8,7 | 175 | AF | |
| 10110000 | 12,11,0,8,1 | 176 | B0 | |
| 10110001 | 12,11,0,1 | 177 | B1 | |
| 10110010 | 12,11,0,2 | 178 | B2 | |
| 10110011 | 12,11,0,3 | 179 | B3 | |
| 10110100 | 12,11,0,4 | 180 | B4 | |
| 10110101 | 12,11,0,5 | 181 | B5 | |
| 10110110 | 12,11,0,6 | 182 | B6 | |
| 10110111 | 12,11,0,7 | 183 | B7 | |
| 10111000 | 12,11,0,8 | 184 | B8 | |
| 10111001 | 12,11,0,9 | 185 | B9 | |
| 10111010 | 12,11,0,8,2 | 186 | BA | |
| 10111011 | 12,11,0,8,3 | 187 | BB | |
| 10111100 | 12,11,0,8,4 | 188 | BC | |
| 10111101 | 12,11,0,8,5 | 189 | BD | |
| 10111110 | 12,11,0,8,6 | 190 | BE | |
| 10111111 | 12,11,0,8,7 | 191 | BF | |
| 11000000 | 12,0 | 192 | C0 | |
| 11000001 | 12,1 | 193 | C1 | A |
| 11000010 | 12,2 | 194 | C2 | B |
| 11000011 | 12,3 | 195 | C3 | C |
| 11000100 | 12,4 | 196 | C4 | D |
| 11000101 | 12,5 | 197 | C5 | E |
| 11000110 | 12,6 | 198 | C6 | F |
| 11000111 | 12,7 | 199 | C7 | G |
| 11001000 | 12,8 | 200 | C8 | H |
| 11001001 | 12,9 | 201 | C9 | I |
| 11001010 | 12,0,9,8,2 | 202 | CA | |
| 11001011 | 12,0,9,8,3 | 203 | CB | |
| 11001100 | 12,0,9,8,4 | 204 | CC | |
| 11001101 | 12,0,9,8,5 | 205 | CD | |
| 11001110 | 12,0,9,8,6 | 206 | CE | |
| 11001111 | 12,0,9,8,7 | 207 | CF | |
| 11010000 | 11,0 | 208 | D0 | |
| 11010001 | 11,1 | 209 | D1 | J |
| 11010010 | 11,2 | 210 | D2 | K |
| 11010011 | 11,3 | 211 | D3 | L |
| 11010100 | 11,4 | 212 | D4 | M |
| 11010101 | 11,5 | 213 | D5 | N |
| 11010110 | 11,6 | 214 | D6 | O |
| 11010111 | 11,7 | 215 | D7 | P |
| 11011000 | 11,8 | 216 | D8 | Q |
| 11011001 | 11,9 | 217 | D9 | R |
| 11011010 | 12,11,9,8,2 | 218 | DA | |
| 11011011 | 12,11,9,8,3 | 219 | DB | |
| 11011100 | 12,11,9,8,4 | 220 | DC | |
| 11011101 | 12,11,9,8,5 | 221 | DD | |

| 8-Bit BCD Code | Character Set Punch Combination | Decimal | Hexa-Decimal | Printer Graphics |
|---|---|---|---|---|
| 11011110 | 12,11,9,8,6 | 222 | DE | |
| 11011111 | 12,11,9,8,7 | 223 | DF | |
| 11100000 | 0,8,2 | 224 | E0 | |
| 11100001 | 11,0,9,1 | 225 | E1 | |
| 11100010 | 0,2 | 226 | E2 | S |
| 11100011 | 0,3 | 227 | E3 | T |
| 11100100 | 0,4 | 228 | E4 | U |
| 11100101 | 0,5 | 229 | E5 | V |
| 11100110 | 0,6 | 230 | E6 | W |
| 11100111 | 0,7 | 231 | E7 | X |
| 11101000 | 0,8 | 232 | E8 | Y |
| 11101001 | 0,9 | 233 | E9 | Z |
| 11101010 | 11,0,9,8,2 | 234 | EA | |
| 11101011 | 11,0,9,8,3 | 235 | EB | |
| 11101100 | 11,0,9,8,4 | 236 | EC | |
| 11101101 | 11,0,9,8,5 | 237 | ED | |
| 11101110 | 11,0,9,8,6 | 238 | EE | |
| 11101111 | 11,0,9,8,7 | 239 | EF | |
| 11110000 | 0 | 240 | F0 | 0 |
| 11110001 | 1 | 241 | F1 | 1 |
| 11110010 | 2 | 242 | F2 | 2 |
| 11110011 | 3 | 243 | F3 | 3 |
| 11110100 | 4 | 244 | F4 | 4 |
| 11110101 | 5 | 245 | F5 | 5 |
| 11110110 | 6 | 246 | F6 | 6 |
| 11110111 | 7 | 247 | F7 | 7 |
| 11111000 | 8 | 248 | F8 | 8 |
| 11111001 | 9 | 249 | F9 | 9 |
| 11111010 | 12,11,0,9,8,2 | 250 | FA | |
| 11111011 | 12,11,0,9,8,3 | 251 | FB | |
| 11111100 | 12,11,0,9,8,4 | 252 | FC | |
| 11111101 | 12,11,0,9,8,5 | 253 | FD | |
| 11111110 | 12,11,0,9,8,6 | 254 | FE | |
| 11111111 | 12,11,0,9,8,7 | 255 | FF | |

# APPENDIX B. HEXADECIMAL-TO-DECIMAL CONVERSION

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

| Hexadecimal | Decimal |
|---|---|
| 000 to FFF | 0000 to 4095 |

For numbers outside the range of the table, add the following values to the table figures:

| Hexadecimal | Decimal |
|---|---|
| 1000 | 4096 |
| 2000 | 8192 |
| 3000 | 12288 |
| 4000 | 16384 |
| 5000 | 20480 |
| 6000 | 24576 |
| 7000 | 28672 |
| 8000 | 32768 |
| 9000 | 36864 |
| A000 | 40960 |
| B000 | 45056 |
| C000 | 49152 |
| D000 | 53248 |
| E000 | 57344 |
| F000 | 61440 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 010 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 020 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 030 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 040 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 050 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 060 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 070 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 080 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 090 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A0 | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B0 | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C0 | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D0 | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E0 | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F0 | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |
| 100 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 110 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 120 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 130 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 140 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 150 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 160 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 170 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 180 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 190 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A0 | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B0 | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C0 | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D0 | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E0 | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F0 | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 200 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 210 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 220 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 230 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 240 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 250 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 260 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 270 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 280 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 290 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A0 | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B0 | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C0 | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D0 | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E0 | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F0 | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |
| 300 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 310 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 320 | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 330 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 340 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 350 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 360 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 370 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 380 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 390 | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A0 | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B0 | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C0 | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D0 | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E0 | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F0 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 400 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 410 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 420 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 430 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 440 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 450 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 460 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 470 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 480 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 490 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A0 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B0 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C0 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D0 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E0 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F0 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |
| 500 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 510 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 520 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 530 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 540 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 550 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 560 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 570 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 580 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 590 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A0 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 5B0 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C0 | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D0 | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E0 | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F0 | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 600 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 610 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 620 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 630 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 640 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 650 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 660 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 670 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 680 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 690 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A0 | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B0 | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 6C0 | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D0 | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E0 | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F0 | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |
| 700 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 710 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 720 | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 730 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 740 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 750 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 760 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 770 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 780 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 790 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A0 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B0 | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C0 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D0 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E0 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F0 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 800 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 810 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 820 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 830 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 840 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 850 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 860 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 870 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 880 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 890 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A0 | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B0 | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C0 | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D0 | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E0 | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F0 | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |
| 900 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 910 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 920 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 930 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 940 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 950 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 960 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 970 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 980 | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 990 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A0 | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B0 | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 9C0 | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D0 | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E0 | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F0 | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A00 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A10 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A20 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A30 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A40 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A50 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A60 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A70 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A80 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A90 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA0 | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| AB0 | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC0 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD0 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE0 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF0 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |
| B00 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B10 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B20 | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B30 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B40 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B50 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B60 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B70 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B80 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B90 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA0 | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| BB0 | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC0 | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BD0 | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE0 | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BF0 | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C00 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C10 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C20 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C30 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C40 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C50 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C60 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C70 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C80 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C90 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA0 | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB0 | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC0 | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD0 | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE0 | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF0 | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |
| D00 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D10 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D20 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D30 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D40 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D50 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D60 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D70 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D80 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D90 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA0 | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB0 | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC0 | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| DD0 | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE0 | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF0 | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E00 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E10 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E20 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E30 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E40 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E50 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E60 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E70 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E80 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E90 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA0 | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB0 | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| EC0 | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED0 | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE0 | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF0 | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |
| F00 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F10 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F20 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F30 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F40 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F50 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F60 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F70 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F80 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F90 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA0 | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB0 | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC0 | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD0 | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE0 | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF0 | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |

# APPENDIX C. SYSTEM/360 ASSEMBLERS-LANGUAGE FEATURES COMPARISON CHART

Features not shown below are common to all assemblers. In the chart:

Dash = Not allowed.
X = as defined in the manual IBM System/360 Operating System Assembler Language, Form C28-6514.

| Feature | Basic Programming Support/360: Basic Assembler | 7090/7094 Support Package Assembler | Other System/360 Assemblers | OS/360 Assembler |
|---|---|---|---|---|
| No. of Continuation Cards/Statement (exclusive of macro-instructions) | 0 | 0 | 1 | 2 |
| Input Character Code | EBCDIC | BCD or EBCDIC | EBCDIC | EBCDIC |
| ELEMENTS: | | | | |
| Maximum Characters per symbol | 6 | 6 | 8 | 8 |
| Character self-defining terms | 1 Char. only | X | X | X |
| Binary self-defining terms | -- | -- | X | X |
| Length attribute reference | -- | -- | X | X |
| Literals | -- | -- | X | X |
| Extended mnemonics | -- | X | X | X |
| Maximum Location Counter value | $2^{16}-1$ | $2^{24}-1$ | $2^{24}-1$ | $2^{24}-1$ |
| Multiple Control Sections per assembly | -- | -- | X | X |
| EXPRESSIONS: | | | | |
| Operators | +-* | +-*/ | +-*/ | +-*/ |
| Number of terms | 3 | 16 | 3 | 16 |
| Number of parentheses | -- | -- | 1 Level | 5 Levels |
| Complex relocatability | -- | -- | X | X |
| ASSEMBLER INSTRUCTIONS: | | | | |
| DC and DS | | | | |
| Expressions allowed as modifiers | -- | -- | -- | X |
| Multiple operands | -- | -- | -- | X |
| Multiple constants in an operand | -- | -- | Except Address Consts. | X |

(Continued)

Appendix C: Assembler Languages--Features Comparison Chart (Continued)

| Feature | Basic Programming Support/360: Basic Assembler | 7090/7094 Support Package Assembler | Other System/360 Assemblers | OS/360 Assembler |
|---|---|---|---|---|
| Bit length specifications | -- | -- | -- | X |
| Scale modifier | -- | -- | X | X |
| Exponent Modifier | -- | -- | X | X |
| DC types | Except B, P, Z, V, Y, S | Except B, Y, V | X | X |
| DC duplication factor | Except A, S | Except A, S | Except S | X |
| DC duplication factor of zero | -- | -- | Except S | X |
| DC length modifier | Except H, E, D, S | Except S | X | X |
| DS types | Only C, H, F, D | Only C, H, F, D | X | X |
| DS length modifier | Only C | Only C | X | X |
| DS maximum length modifier | 256 | 256 | 256 | 65,535 |
| DS constant subfield permitted | -- | -- | X | X |
| COPY | -- | -- | -- | X |
| CSECT | -- | -- | X | X |
| DSECT | -- | -- | X | X |
| ISEQ | -- | -- | X | X |
| LTORG | -- | -- | X | X |
| PRINT | -- | -- | X | X |
| TITLE | -- | X | X | X |
| COM | -- | -- | -- | X |
| ICTL | 1 oprnd 1 or 25 only | 1 oprnd | X | X |
| USING | 2 oprnds oprnd 1 reloc only | 2 oprnds oprnd 1 reloc only | 6 oprnds | X |
| DROP | 1 oprnd only | 1 oprnd only | 5 oprnds | X |

(Continued)

Appendix C: Assembler Languages--Features Comparison Chart (Continued)

| Feature | Basic Programming Support/360: Basic Assembler | 7090/7094 Support Package Assembler | Other System/360 Assemblers | OS/360 Assembler |
|---|---|---|---|---|
| CCW | oprnd 2 reloc only | oprnd 2 reloc only | X | X |
| ORG | no blank oprnd | no blank oprnd | X | X |
| ENTRY | 1 oprnd only | 1 oprnd only | 1 oprnd only | X |
| EXTRN | max 14<br>1 oprnd only | 1 oprnd only | 1 oprnd only | X |
| CNOP | 2 dec digits | 2 dec digits | 2 dec digits | X |
| PUNCH | -- | -- | -- | X |
| REPRO | -- | -- | X | X |
| Macro Instructions | -- | -- | X | X |

## APPENDIX D. HEXADECIMAL TO MNEMONIC OPERATION CODE TABLE

The table in this appendix provides for easy conversion from the hexadecimal equivalent of the actual machine operation codes to their associated assembler mnemonic operation codes.

Second Hexadecimal Digit

First Hexadecimal Digit

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | SPM | BALR | BCTR | BCR | SSK | ISK | SVC | | | | | | RR |
| 1 | LPR | LNR | LTR | LCR | NR | CLR | OR | XR | LR | CR | AR | SR | MR | DR | ALR | SLR | |
| 2 | LPDR | LNDR | LTDR | LCDR | HDR | | | | LDR | CDR | ADR | SDR | MDR | DDR | AWR | SWR | |
| 3 | LPER | LNER | LTER | LCER | HER | | | | LER | CER | AER | SER | MER | DER | AUR | SUR | |
| 4 | STH | LA | STC | IC | EX | BAL | BCT | BC | LH | CH | AH | SH | MH | | CVD | CVB | RX |
| 5 | ST | | | | N | CL | O | X | L | C | A | S | M | D | AL | SL | |
| 6 | STD | | | | | | | | LD | CD | AD | SD | MD | DD | AW | SW | |
| 7 | STE | | | | | | | | LE | CE | AE | SE | ME | DE | AU | SU | |
| 8 | SSM | | LPSW | | WRD | RDD | BXH | BXLE | SRL | SLL | SRA | SLA | SRDL | SLDL | SRDA | SLDA | RS or SI |
| 9 | STM | TM | MVI | TS | NI | CLI | OI | XI | LM | | | | SIO | TIO | HIO | TCH | |
| A | | | | | | | | | | | | | | | | | |
| B | | | | | | | | | | | | | | | | | |
| C | | | | | | | | | | | | | | | | | SS |
| D | | MVN | MVC | MVZ | NC | CLC | OC | XC | | | | | TR | TRT | ED | EDMK | |
| E | | | | | | | | | | | | | | | | | |
| F | | MVO | PACK | UNPK | | | | | ZAP | CP | AP | SP | MP | DP | | | |

Where more than one page number follows an Index entry, the most important reference is listed first.

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

# READER'S COMMENT FORM

IBM System/360 C28-6503-6
Basic Programming Support
Basic Assembler and Basic Utility Programs (Card)
Specifications and Operating Guide

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. All comments will be handled on a non-confidential basis. Copies of this and other IBM publications can be obtained through IBM Branch Offices.

| | Yes | No |
|---|---|---|
| ● Does this publication meet your needs? | ☐ | ☐ |
| ● Did you find the material: | | |
| Easy to read and understand? | ☐ | ☐ |
| Organized for convenient use? | ☐ | ☐ |
| Complete? | ☐ | ☐ |
| Well illustrated? | ☐ | ☐ |
| Written for your technical level? | ☐ | ☐ |

- What is your occupation? ____________________
- How do you use this publication?
  - As an introduction to the subject? ☐
  - For advanced knowledge of the subject? ☐
  - For information about operating procedures? ☐
  - As an instructor in a class? ☐
  - As a student in a class? ☐
  - As a reference manual? ☐

  Other ____________________
- Please give specific page and line references with your comments when appropriate.

**COMMENTS:**

- Thank you for your cooperation. No postage necessary if mailed in the U. S. A.

Staple

C28-6503-6

## YOUR COMMENTS, PLEASE . . .

This publication is one of a series that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, help us produce better publications for your use. Each reply is carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in using your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 170
ENDICOTT, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

**IBM Corporation**
**P. O. Box 6**
**Endicott, N. Y. 13760**

Attention: Programming Publications, Dept. 157

Fold Fold

Cut Along Line

**International Business Machines Corporation**
**Data Processing Division**
**112 East Post Road, White Plains, N.Y. 10601**
**[USA Only]**

**IBM World Trade Corporation**
**821 United Nations Plaza, New York, New York 10017**
**[International]**

Additional Comments: