# IBM System/3 Basic Assembler Reference Manual

Program Numbers:

5702-AS1 (Models 8 and 10)
5704-AS1 (Model 15)
5704-AS2 (Model 15)
5705-AS1 (Model 12)

SC21-7509-7
File No. S3-21

Program Product

## Preface

This publication is a reference manual for the programmer using the IBM System/3 Basic Assembler language. This language provides facilities for representing machine usable instructions symbolically on a one-for-one basis. The symbolic representations are translated by the IBM System/3 Basic Assembler into the machine usable form necessary for running a program on the System/3.

### System/3 Model 8

The System/3 Model 8 is supported by System/3 Model 10 Disk System control programming and program products. The facilities described in this publication for the Model 10 are also applicable to the Model 8, although the Model 8 is not referenced. It should be noted that not all devices and features which are available on the Model 10 are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of *IBM System/3 Model 8 Introduction,* GC21-5114.

### Related Publications

The *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual,* GA21-9236, contains specifications governing the use of assembler language instructions.


**Eighth Edition (April 1975)**

This is a minor revision of SC21-7509-5 incorporating Technical Newsletters:

SN21-5385 March 17, 1976
SN21-5434 December 31, 1976
SN21-5536 June 24, 1977

This revision makes some changes to various pages and introduces information concerning the IBM System/3 Model 8. Changes to text and small changes to illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by the symbol ● at the left of the figure caption.

This edition applies to version 12, modification 00 of IBM System/3 Model 10 Disk System Basic Assembler (Program Product Number 5702-AS1); version 03, modification 00 of IBM System/3 Model 15 Basic Assembler (Program Product Number 5704-AS1); and to all subsequent versions and modifications unless otherwise indicated in new editions or technical newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/3 Bibliography, GC20-8080, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the branch office serving your locality.

A Reader's Comment Form is at the back of this publication. If the form is gone, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.



©Copyright International Business Machines Corporation 1970, 1971, 1972, 1973, 1975

# Contents

The IBM System/3 Basic Assembler language is a symbolic language. That is, it must be translated into a form usable by the computer before a program can be run. The computer-usable form is called machine language, and the IBM System/3 Basic Assembler language provides a convenient method for representing, on a one-for-one basis, machine language instructions and related data necessary to write a program for IBM System/3. This one-for-one relationship to machine language instructions gives assembler language great programming versatility.

The assembler language is composed of symbols, called mnemonics, which are used to represent the operation codes of two types of instruction statements:

1. *Machine instruction statements* are the symbols that represent machine language instructions on a one-for-one basis. Note that symbolically represented machine instructions are *translated* into machine language by the assembler.

2. *Assembler instruction statements* are instructions which control the functions of the assembler. Each assembler instruction statement causes the assembler to perform a specific operation during the assembly process.

The IBM System/3 Basic Assembler:

- Processes instructions written in assembler language.
- Translates the assembler language instructions into machine language.
- Assigns storage locations.
- Performs other functions necessary to produce an executable machine language program.

In order to call the assembler from its storage location, a specific set of OCL (operation control language) instructions must be used. Following these OCL instructions, the user may elect to include an OPTIONS instruction, a facility which allows him to take advantage of various combinations of output listings and punched decks.

There are certain procedures for storing assembler routines on the Model 10 Disk System, Model 12, and Model 15 R (relocatable) Library and for loading assembler object programs into main storage. These procedures, as well as the other items mentioned briefly above, are discussed more fully in the text.

## MINIMUM SYSTEM REQUIREMENTS

The minimum system configuration and optional device support for the Basic Assembler program is shown in the *IBM System/3 Models 6, 8, 10, and 12 System Generation Reference Manual,* GC21-5126 and in the *IBM System/3 Model 15 System Generation Reference Manual,* GC21-7616.

## MAIN STORAGE REQUIREMENTS

The Model 10 Disk System Basic Assembler (5702-AS1) requires 8,192 bytes of main storage for execution, exclusive of control program requirements.

The Model 12 Basic Assembler (5705-AS1) and the Model 15 Basic Assembler (5704-AS1 or 5704-AS2) require 10,240 bytes of main storage for execution, exclusive of control program requirements.

The IBM System/3 Basic Assembler language is a symbolic language that provides a convenient method for representing, on a one-for-one basis, machine language instructions. The symbolic representations in assembler language coding are translated by the IBM System/3 Basic Assembler into the machine language form usable by the computer. In order to code in assembler language, the user must become familiar with certain terms, coding conventions, instructions, and other features of the language. The remainder of this chapter deals with these items.

## BASIC STATEMENT FORMAT

A statement coded in assembler language can contain up to four entries from left to right: Name, Operation, Operand, and Remark. See *Assembler Coding Conventions* in this manual for an explanation of the contents and functions of each entry.

## TERMS AND EXPRESSIONS

A term is a single symbol, self-defining value, or location counter reference which can be used only in the operand field of an assembler language instruction. The three types of terms are described under *Terms* in this section.

An expression consists of one or more terms. It is used to specify the operand fields of assembler language instructions. Terms and expressions are classed as either absolute or relocatable. A term or expression is absolute if its value is not changed when the assembled program in which it is used is relocated in main storage. A term or expression is relocatable if its value is changed when the program in which it is used is relocated.

Program relocation is the loading of an assembled program (object program) into a different area of main storage from that which was originally assigned by the assembler. The difference (in bytes) between the originally assigned address of the object program and the address of the relocated object program is the amount of relocation. The addresses assigned to all instructions and data in the relocated program are changed by the amount of relocation. In Figure 1, Object Program A is initially loaded at address 2000 in main storage. When Object Program A is loaded a second time, it is placed at address 3000 in main storage. The amount of relocation is 1000 bytes. Therefore, the values of all relocatable terms and expressions used in Object Program A would be increased by 1000 during the second loading.

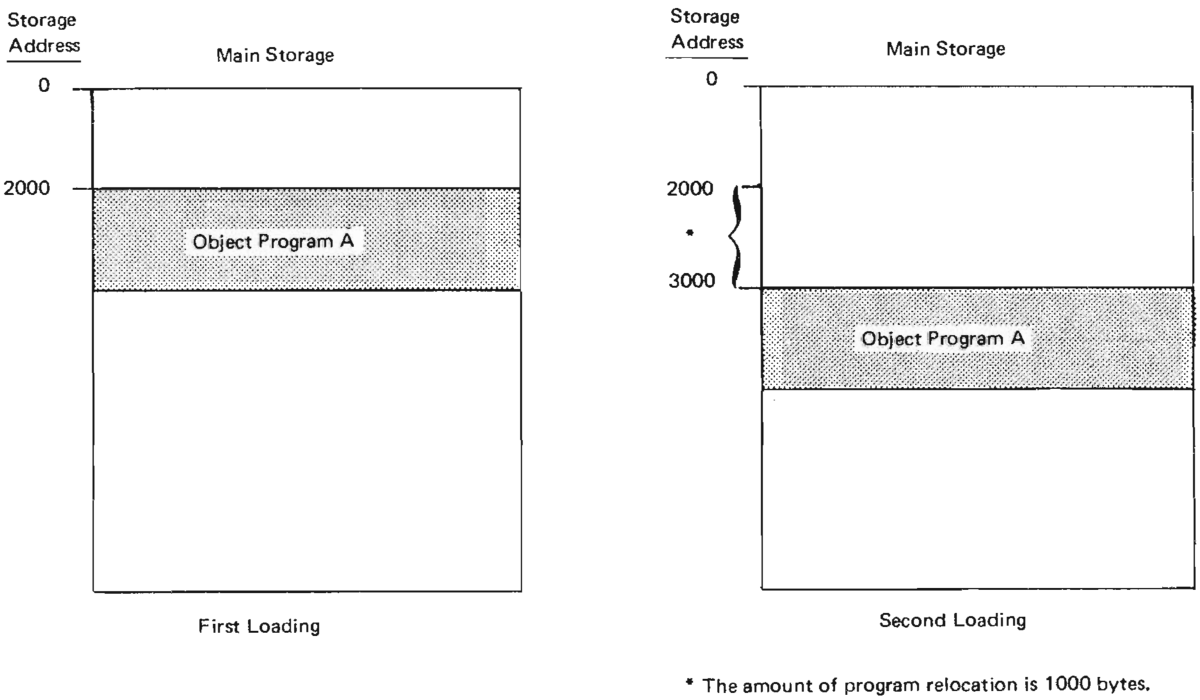



Figure 1. Program Relocation

## TERMS

Three types of terms are used in the IBM System/3 Basic Assembler language.

- Symbol
- Self-defining term
- Location counter reference

### The Symbol

A symbol is a character or combination of characters used to represent storage locations, instructions, input/output units, registers, or arbitrary values. A symbol can be used in either the name field or the operand field of a statement. When used in the name field, the symbol is called a name field entry. When used in the operand field, the symbol is called a symbolic term.

When the assembler finds a symbol in the name field of a statement, it assigns to that symbol an address value attribute. See *Addressing* in this section. The assembler also assigns a length attribute to the symbol, which is the number of bytes in the storage field named by the symbol. There are exceptions. When the assembler encounters EQU, START, or TITLE statements, it does not assign the usual attributes. EQU name field entries derive their values from the operand, START name field entries are assigned a length of 1, and TITLE name field entries are assigned no values at all.

The same symbol cannot be used as a name entry more than once within a program with the exception of the TITLE card. In order for a symbol to be used in the operand field, it must be defined (that is, used as a name) on an instruction other than a TITLE card somewhere in the program. Once it is defined, the symbol may appear in any number of operands. Whether the symbol is used as a name or an operand, these rules must be followed:

1. The symbol can consist of no more than six characters, the first of which must be either alphabetic or $, #, @. The other characters can be any combination of alphabetic, numeric, or $, #,@.
2. Blanks and special characters other than $, #, @ cannot be used in a symbol.

### The Self-Defining Term

The self-defining term is a term which specifies an actual value or bit configuration.

The value expressed by the self-defining term is taken literally by the assembler and is assembled into the instruction. Like all terms, the self-defining term is used only in the operand field.

There are four types of self-defining terms:

- Decimal
- Hexadecimal
- Binary
- Character

#### *Decimal Self-Defining Terms*

A decimal self-defining term is an unsigned decimal number written as a sequence of decimal digits. High order zeros may be used, such as in 0003. If a decimal term is used as an address, its value cannot exceed the number of bytes in main storage. A decimal term consists of no more than five digits and cannot exceed a value of 65,535. This value is equivalent to the binary value that can be contained in two bytes. A decimal self-defining term is assembled as its binary equivalent.

*Examples:* 16 132 00006 43678

In the following example, a decimal self-defining term is used in a Move Immediate (MVI) instruction. The binary equivalent of 25 would be placed in the 1-byte area referenced by the symbol, COST

| NAME | OPERATION | OPERAND |
|---|---|---|
| ALPHA | MVI | COST, 25 |

*Hexadecimal Self-Defining Terms*

Hexadecimal self-defining terms can consist of up to four hexadecimal digits enclosed in apostrophes and preceded by the letter X.

*Examples:* X'C34A' X'04F' X'6' X'DE'

Each digit is assembled into its 4-bit binary equivalent. Therefore, the maximum value would be X'FFFF' (65,535).

The following is an example of the use of a hexadecimal self-defining term. The 1-byte area at SWITCH would contain the hexadecimal value F0 (binary, 11110000) after execution of the instruction.

| NAME | OPERATION | OPERAND |
|---|---|---|
| BETA | MVI | SWITCH, X'F0' |

*Binary Self-Defining Terms*

Binary self-defining terms are written as a sequence of 1's and 0's enclosed in apostrophes and preceded by the letter B; such as B'1011'. This term would appear in storage as 00001011. The high-order (leftmost) bits are padded with 0-bits to make a multiple of eight bits of data (one or two bytes). A maximum of 16 bits of data can be represented in each term. In the following example of a Move Immediate instruction, the binary information will be moved into the 1-byte field at AREA.

| NAME | OPERATION | OPERAND |
|---|---|---|
| GAMMA | MVI | AREA, B'10110011' |

*Character Self-Defining Terms*

Character self-defining terms consist of one or two characters enclosed by apostrophes and preceded by the letter C; such as C'A3'. Any of the valid punch combinations can be used in a character self-defining term.

*Examples:* C'A9' C'EA' C'LB' C'3'

Because certain terms in the assembler language must be enclosed by apostrophes (such as C'EA'), for every apostrophe that is used as a character in a self-defining term, two must be written. For example, the characters A' would be written as C'A'''.

In the following example, a dollar sign ($) would be moved into the byte field at REPORT.

| NAME | OPERATION | OPERAND |
|---|---|---|
| DELTA | MVI | REPORT, C'$' |

**Location Counter Reference**

*Location Counter:* The location counter is an internal counter, maintained by the assembler, which always points to the next available storage location. As each new statement is processed, the location counter is increased by the number of bytes in the assembled statement. The assembler uses the current address in the location counter to assign consecutive storage addresses to program statements.

*Location Counter Reference:* A location counter reference is an asterisk (*) used as a term in the operand of an instruction. When the assembler encounters an asterisk, it substitutes the current value of the location counter (which always points to the next available storage location) for the asterisk.

## EXPRESSIONS

An expression consists of an arithmetic combination of one or more terms. In a multi-term expression, terms must be separated by an arithmetic operator: the arithmetic operators are + for addition, – for subtraction, and * for multiplication.

*Examples:* AREA+X'2D' N–25 R+15 A*8

The rules for coding an expression are:

1. Two terms or two operators must not be used consecutively in an expression.
2. Parentheses cannot be used in an expression.
3. Only absolute terms can be used in a multiply operation.
4. Blanks are not allowed in an expression.
5. a. Using the Model 10 disk system basic assembler, an expression may consist of only one term when that term is a symbol used as the operand of an EXTRN statement.

   b. Using the Model 15 basic assembler, if the expression contains an external symbol, then the expression must be of the form A or A±e. A is a symbol used as the operand of an EXTRN statement and e is an absolute expression.

   *Note:* An A±e expression must not be in a Model 10 subroutine with RPG II.

If there is more than one term in the expression, the terms are reduced to a single value as follows:

1. Each term is evaluated separately.
2. Arithmetic operations are then performed in a left-to-right sequence, except that multiplication is performed before addition or subtraction. An example would be A+B*C, which would be evaluated as A+(B*C), not (A+B)*C. The result would be the value of the expression.
3. The intermediate result of the expression evaluation is a 3-byte, or 24-bit value. Intermediate results must be in the range of $-2^{24}$ through $2^{24}-1$.

Negative values are carried in the two's-complement form. The final value of the expression is the truncated, rightmost 16 bits of the result. The value of the expression before truncation must be in the range of -65536 through +65535. A negative result is considered to be a 2-byte positive value.

*Note:* In address constants the full 24-bit final expression result is truncated on the left to fit the length of the constant.

*Absolute Expressions:* An expression is considered absolute if its value is unaffected by program relocation.

An absolute term may be a non-relocatable symbol, or any of the self-defining terms. All arithmetic operations are permitted between absolute terms.

An absolute expression can contain relocatable terms or a combination of relocatable and absolute terms under the following conditions:

1. The expression must contain an even number of relocatable terms.
2. The relocatable terms must be paired and each pair must consist of terms with opposite signs. The paired terms need not be adjacent.
3. Relocatable terms cannot be used in a multiplication operation.

Pairing relocatable terms with opposite signs cancels the effect of the relocation, because both terms would be relocated by the same value. Therefore, the value represented by the paired terms would, in effect, remain constant regardless of the program relocation. For example, in the absolute expression A–Y+X, A is an absolute term and X and Y are relocatable terms. If A equals 50, Y equals 25, and X equals 10, the value of the expression would be 35. If X and Y are relocated by a factor of 100, their values would become 110 and 125, respectively. However, the expression would still evaluate as 35 (50–125+110=35). Absolute expressions reduce to a single absolute value.

*Relocatable Expressions:* A relocatable expression is one whose value changes by the amount of relocation when the program in which it is used is relocated. All relocatable expressions must reduce to a positive value.

A relocatable expression can be a combination of relocatable and absolute terms under the following conditions:

1. There must be an odd number of relocatable terms.

2. All relocatable terms, except one, must be paired and each pair must consist of terms with opposite signs. The paired terms need not be adjacent.

3. The unpaired term must not be immediately preceded by a minus sign.

4. Relocatable terms cannot enter into a multiplication operation.

All terms in a relocatable expression are reduced to a single value. This single value is the value of the unpaired relocatable term after it has been adjusted (displaced) by the resultant value of the other terms in that expression. For example, in the expression W–X+Y where W, X, and Y are relocatable terms; and W=10, X=3, Y=1; the result would be the relocatable value of 8.

If the program is relocated by 100 bytes, the resultant value of the expression would be increased by the amount of relocation (100), giving the expression a value of 108.

In the following expression, a combination of absolute and relocatable terms are used: A+F*G–D+B. A, D, and B are relocatable terms; F and G are absolute terms. When given the values A=3, B=2, D=5, F=1, and G=4, the result would be a relocatable value of 4. The multiplication occurred first, resulting in 4; then the addition and subtraction of the other terms, including the result of the multiplication, was performed in a left-to-right direction. The result of the arithmetic operations is a relocatable value of 4 for this expression.

Upon relocation, the value of this expression can be determined by adding the amount of relocation to all relocatable terms.

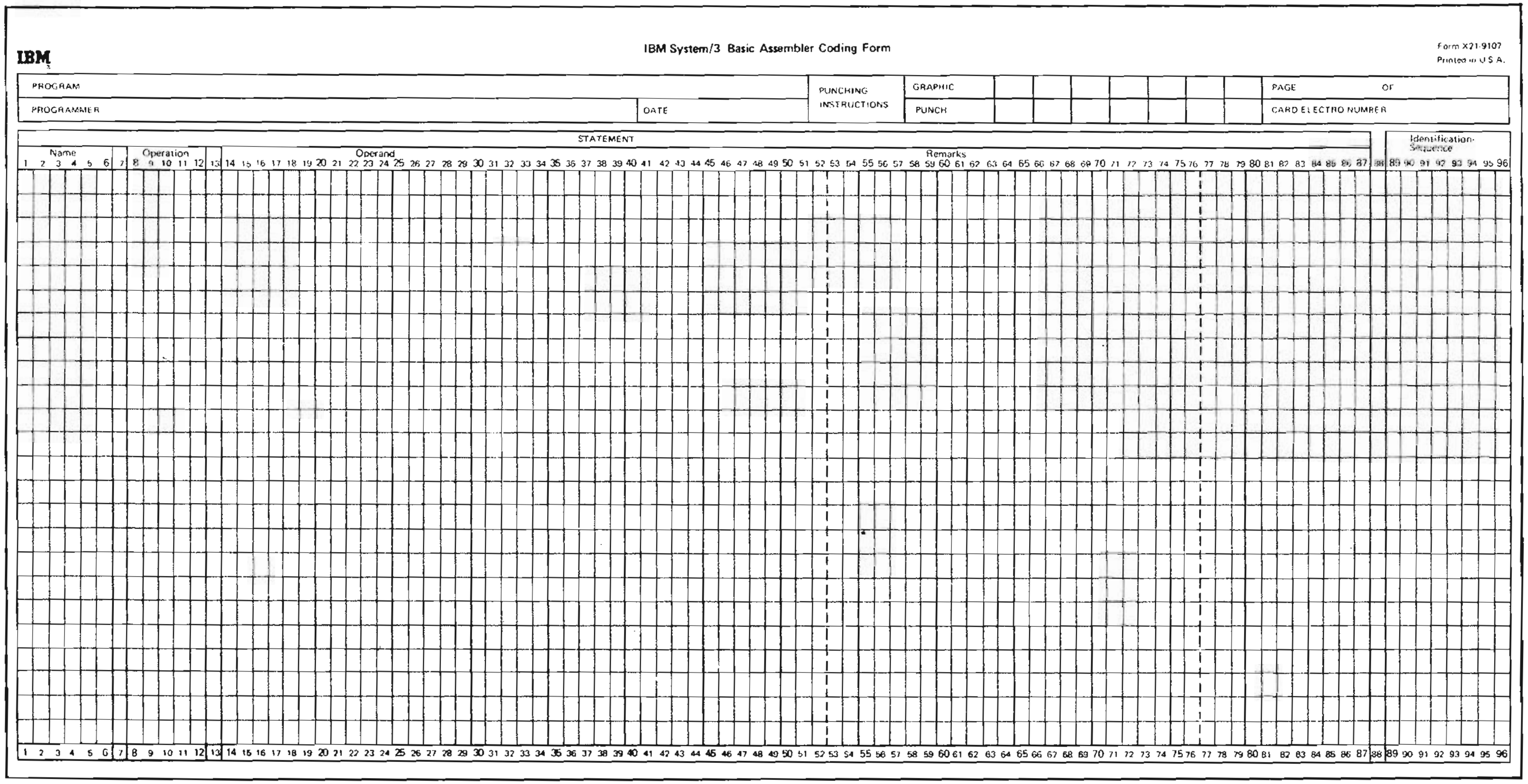
## ASSEMBLER CODING CONVENTIONS

This section explains the general coding conventions associated with the IBM System/3 Basic Assembler language. When coding in assembler language, the programmer uses the IBM System/3 Assembler Coding Form (Figure 2).

### The Statement Format

Each line on the coding form is divided into two segments: Statement (columns 1-87), and Sequence (columns 89-96).

The Statement segment can contain up to four entries, from left to right: Name, Operation, Operand and Remark. The Name field is column dependent. It must start in column 1, unless otherwise specified by the ICTL assembler instruction (see *Assembler Instruction Statements*). All other entries can start in any column, as long as there is at least one blank separating each entry and the entries remain in the stated order. Figure 3 is a diagram of assembler statement entries.




ART: 52908

Figure 2. IBM System/3 Basic Assembler Coding Form

*Name Entry*

- Optional or required depending on the specific instruction.
- Up to six characters can be used in a name.
- First character must be alphabetic (including $, #, @).
- First character must be in column 1 unless otherwise specified by an ICTL assembler instruction.
- No special characters or blanks in a name (except $, #, @).
- At least one blank must follow the Name entry or appear in the first Name entry column (if no name is entered).

*Operation Entry*

- Required entry.
- Contains mnemonic operation code (list of valid machine codes is in *Appendix A. Machine Instructions).*
- Must be followed by a blank.

*Operand Entry*

- Optional or required depending on the specific instruction.
- Contains coding that describes data to be acted upon.
- Operands are separated by a comma.
- No blanks between terms or operands.
- Blanks are allowed within character constants and character self-defining terms only.
- If the entire operand entry is omitted, but a remark entry is desired, absence of the operand must be indicated by a comma in the operand entry, preceded and followed by one or more blanks.
- Must be followed by a blank.

*Remark Entry*

- Optional entry.
- Contains a brief verbal description of the statement's function.
- Cannot extend beyond column 87 or a limit prescribed by ICTL assembler instruction.
- Can contain any combination of valid characters or blanks.
- Must be followed by a blank.

*Identification–Sequence Entry*

- Optional entry.
- Contains statement identification or sequence characters.
- See *ISEQ – Input Sequence Checking* later in this section.

**Comment Statements**

The entire statement field (columns 1-87) can be used for comments by placing an asterisk in column 1 (or the beginning column, as set by the ICTL assembler instruction). Comments can be extended for more than one line by the repeated use of the asterisk in the first column of additional cards. Comment lines may be used anywhere in the source program and are printed on the program listing. Sequence checking is also performed on cards containing comment statements.

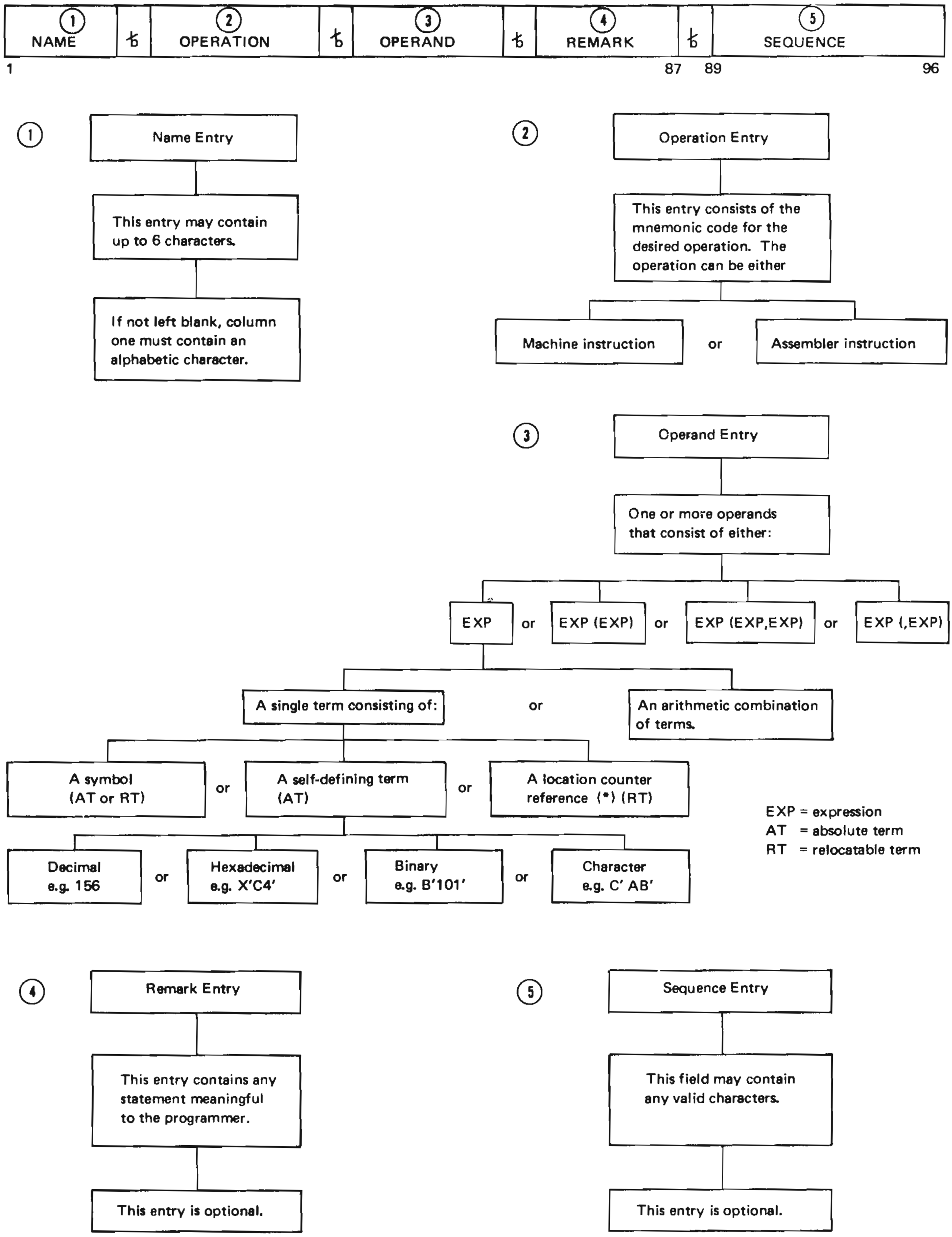



**Figure 3. Assembler Statement Entries**

## ADDRESSING

The programmer must be able to access any part of storage. IBM System/3 provides two methods of addressing: direct and base-register displacement. The relative addressing technique can be used with both methods. For addressing, see the *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual,* GA21-9236.

### Direct Addressing

The direct addressing method allows the programmer to represent a 16-bit instruction address by using an expression as an operand entry. The assembler places the value of the expression in the machine instruction which it generates.

Two bytes are always used in the machine instruction for a direct address. A direct address is indicated by the absence of a register in the operand.

*Example:* MVI A,C'D'

This indicates to the assembler that a direct address is to be generated for location A (see *Machine Instruction Operands).*

### Base-Register Displacement Addressing

Base-register displacement addressing involves setting up a base address from which other addresses can be calculated. This base address must be placed in the base register before the base register is used for addressing.

One byte is always used in the machine instruction for a base-register displacement address and is indicated by the presence of a register in the operand.

*Examples:* MVI A(,2),C'D'
MVI 5(,1),C'D'

This indicates to the assembler that a base-register displacement address is to be generated for location A using base register 2 and for displacement 5 from base register 1.

IBM

PROGRAM

PROGRAMMER

```
Name    Operation  Operand
RX1     EQU        1
        LA         ADBASE,RX1
        USING      ADBASE,RX1
        MVC        A(,RX1),B(2,RX1)
         :
         :
```

Figure 4. Base-Register Displacement Addressing

The base register plus a displacement can reference any higher address within 255 bytes of the specified base address. The displacement portion of the address can be either absolute or relocatable; however, in either case the programmer indicates that a base-displacement address is to be generated by the presence of the register in the operand (see *Machine Instruction Operands*). If relocatable displacements are used, the USING statement (see *Assembler Instruction Statements*) must be used to indicate to the assembler which register contains the base address and what address will be loaded into that register. The USING instruction does not load the register with the specified address; the programmer must use a load instruction to place the indicated address into the register. Figure 4 is an example of base-register displacement addressing.

In Figure 4 two bytes of data will be moved from the location of B to the location of A. The assembler calculates the displacement to the addresses for A and B, if A and B are relocatable and are within a positive 255 bytes of the address in base register XR1. If either A or B is over 255 bytes from the base address, an addressing error occurs and an assembler error statement is generated. If the terms A and B are not relocatable symbols, the assembler uses the absolute values (up to 255) of the terms for the displacement. If absolute displacements are used, the USING assembler statement is not required.

*Note:* The programmer must explicitly specify the base register whenever base-register displacement addressing is used.

The programmer terminates the use of a previously defined base register through the use of the DROP instruction (see *Assembler Instruction Statements*). The value of the register is not affected. This register cannot, however, enter into base-register displacement addressing using relocatable displacements until specified again by a USING instruction.

### Relative Addressing

Relative addressing is an addressing technique accomplished by adding bytes to or subtracting bytes from a symbol or location counter reference. The expression *+5, for example, specifies the location 5 bytes beyond the current value of the location counter. Figure 5 is an example of relative addressing.

In Figure 5, the instruction with the operation code ZAZ has a length of 6 bytes, the instruction AZ has a length of 5 bytes and the instruction with MVI has a length of 4 bytes in storage. Using relative addressing, the location of the AZ instruction can be expressed in two ways, AAA+6 or BBB–5.

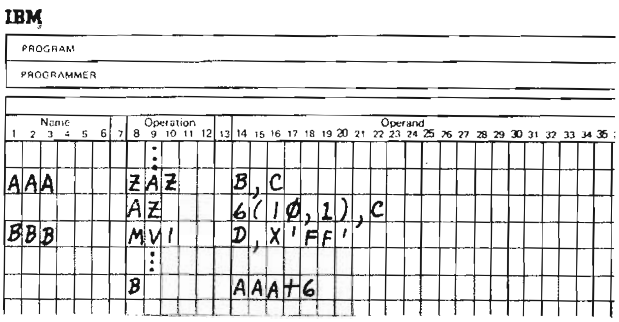



**Figure 5. Relative Addressing**

Figure 6 shows how the AZ instruction can be addressed relative to the nearby symbolic addresses, AAA and BBB.

Relative addressing may also be used with base-register displacement addressing if the displacement is a relocatable term.

*Example:* MVC A+5(,RX1),B(2,RX1)

In the example, A+5 is an example of relative addressing used with base-register displacement addressing.

### Instruction Addressing

A symbol used as a name entry in a machine-instruction statement addresses the *leftmost* byte of storage occupied by that instruction.

### Data Addressing

A symbol used as a name entry in a data definition instruction (see *DC – Define Constant* and *DS – Define Storage*) address the *rightmost* byte of storage occupied by or reserved for that data.

### Control of Location Counter

Addressing in any computer language depends upon the location counter. IBM System/3 allows the programmer to control the location counter by using two assembler instructions: START and ORG. The START assembler instruction can be used to initialize the location counter to a desired value at the beginning of a program. The ORG assembler instruction can be used to change the value of the location counter anywhere in a program.

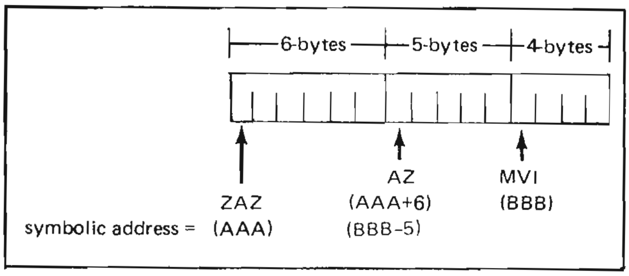



**Figure 6. Schematic of Relative Addressing**

These two instructions are described in detail under *Assembler Instruction Statements.*

## MACHINE INSTRUCTION STATEMENTS

Machine instruction statements are symbols that represent machine language instructions on a one-for-one basis. The assembler translates these symbolic representations into machine language usable by the computer. Machine instruction statements differ from assembler instruction statements in that the machine instruction statements are executable parts of the program's logic (such as MVI, ST, LA, etc), while assembler instruction statements are simply orders to the assembler, each statement directing a specific operation (such as DC, START, SPACE, etc). See *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual,* GA21-9236 for a description of the execution of machine instructions.

The format for a machine instruction statement is closely related to, but not the same as, the machine language instruction format which results from the assembly process (see *Appendix A. Machine Instructions* for machine language instruction formats).

A mnemonic operation code is used in place of the actual machine language operation code and one or more operands provide the information required by the machine instruction. A remark and a sequence entry may be included in the machine-instruction statements, but they will not affect the machine language instruction.

### Name Entry Attributes

Any machine-instruction statement can contain a symbol as a name entry. Other machine-instruction statements can use that symbol as an operand. The assembler assigns value and length attributes (characteristics) to every sumbol used in a program. The value attribute of a symbol which is used as a name entry in a machine-instruction statement is the address of the leftmost byte of storage occupied by the assembled instruction. The length attribute of the symbol is the number of bytes of storage occupied by the assembled instruction. Refer to *Lengths–Explicit and Implied* in this section for a discussion of the length attributes of other types of symbols, terms, and expressions.

### Machine Instruction Mnemonic Codes

The mnemonic operation codes are designed to be easily-remembered codes that remind the programmer of the functions performed by the instructions. The mnemonic codes are translated into machine-language operation codes by the assembler. IBM System/3 Basic Assembler provides mnemonic and extended mnemonic operation codes. The complete set of mnemonic codes is listed in *Appendix A. Machine Instructions.*

### Extended Mnemonic Codes

Extended mnemonic codes are provided for the convenience of the programmer. They are unlike other mnemonic codes in that part of the information usually provided in the operand is in the extended mnemonic code itself. Extended mnemonic codes allow the following:

1. Conditional branches (BC) and jumps (JC) can be specified mnemonically, requiring only a branch address as an operand.

2. Half-byte moves (MVX) can be specified mnemonically, requiring only the use of addresses as operands.

3. The supervisor call form of the command CPU (CCP) machine operation can be specified mnemonically (Model 15 only).

Extended mnemonic codes are not part of the set of machine instructions, but are translated by the assembler into the corresponding operation code and condition combinations.

See *Appendix A. Machine Instructions* for a list of extended mnemonic codes.

### Machine Instruction Operands

This section describes (1) operand fields and subfields, (2) explicit and implied lengths, and (3) operand groups and formats. The operands of machine instruction statements provide the information about addresses, lengths, and immediate data that is required by the assembler to generate executable machine instructions. General rules for coding of operands are covered in *Assembler Coding Conventions.*

#### *Operand Fields and Subfields*

The left operand of a pair is called operand 1, or operand field 1; the right operand is called operand 2, or operand field 2. An operand field may include one or two subfields (length subfield, register subfield) as in the following example of base-register displacement addressing.

*Example:* 40(,2)

The above operand field contains a displacement entry, 40, and a register subfield entry, 2, representing index register 2. The following rules apply to the coding of subfields:

1. Parentheses must enclose a subfield or subfields.

2. Blanks cannot be used within subfield parentheses.

3. A comma must separate two subfields within parentheses (L,R).

4. If the first subfield of a pair is omitted, the comma that separates it from the second subfield must be retained (,R).

5. If the second subfield of a pair is omitted, the comma separating the pair must also be omitted (L).

6. If both subfields are omitted, the separating comma and the parentheses must also be omitted.

Operand subfields can contain immediate data, length, or register information. Only absolute expressions and self-defining terms may be used as subfield entries.

### *Lengths — Explicit and Implied*

A length subfield in an operand may be either explicit or implied. To imply a length, the programmer omits the length subfield from an operand. When a length specification is not included in an operand requiring a length, the assembler includes the implied length of the first operand, such as the length attribute of a name entry (see *Name Entry Attributes* in this section). The length attributes of various terms and expressions are shown in Figure 7.

An explicit length is written by the programmer in the operand as an absolute expression. The explicit length overrides any implied length.

| Term or Expression | Length Attribute |
|---|---|
| 1. Name entry symbol of a machine-instruction | Length, in bytes, of the instruction. |
| 2. Location-counter reference (*) | Length, in bytes, of the instruction in which it appears (except in the EQU assembler statement, where the length attribute assigned is one). |
| 3. Expression | Length attribute of the leftmost term in the expression. |
| 4. Self-Defining Term | Length attribute is one. |
| 5. START name entry | Length attribute is one. |

NOTE: See also *Subfield 3 -- Length* under *Data Defining Instructions.*

Figure 7. Length Attributes of Terms and Expressions

### *Operand Groups*

Machine-instruction statement operands are divided into six groups. The characteristics of each group are as follows:

*Group 1:* Two-operand format in which a length is explicit or implied in both operands.

*Group 2:* Two-operand format in which a length can be explicit in either operand, but not in both. If length is not explicit in either operand, the assembler uses the implied length of operand 1.

*Group 3:* Two-operand format in which a length cannot be specified.

*Group 4:* One-operand format in which only immediate data may be used.

*Group 5:* Two-operand format in which both operands are immediate data.

*Group 6:* Two-operand format in which operand 1 is used by the assembler to calculate a positive displacement and operand 2 is immediate data.

Figure 8 shows the allowable operand formats for each operand group. The instructions using each operand group are also listed. Refer to *Appendix A. Machine Instructions* for the related machine-instruction formats.

For the extended mnemonics of the MVX instruction, the I-field information is inherent in the mnemonic and the I-field is omitted from the operand. For the extended mnemonics of the BC and JC instructions, the second operand (I-field) is not used since the information is inherent in the mnemonic (see *Extended Mnemonic Codes* in this section).

Data movement is from operand 2 to operand 1 in a two-address format instruction (group 1 and group 2). This operand order is equivalent to that of machine instructions.

| GROUP | INSTRUCTIONS | ALLOWABLE OPERAND FORMAT | | | |
|---|---|---|---|---|---|
| 1 | ZAZ,AZ,SZ | A,A<br>A,A(L)<br>A,D(,R)<br>A,D(L,R) | A(L),A<br>A(L),A(L)<br>A(L),D(,R)<br>A(L),D(L,R) | D(,R),A<br>D(,R),A(L)<br>D(,R),D(,R)<br>D(,R),D(L,R) | D(L,R),A<br>D(L,R),A(L)<br>D(L,R),D(,R)<br>D(L,R),D(L,R) |
| 2 | MVC,CLC,ALC<br>SLC,ITC,ED | A,A<br>A,A(L)<br>A,D(,R)<br>A,D(L,R) | A(L),A<br>A(L),D(,R) | D(,R),A<br>D(,R),A(L)<br>D(,R),D(,R)<br>D(,R),D(L,R) | D(L,R),A<br>D(L,R),D(,R) |
| | MVX | A,A(I)<br>A,D(I,R) | A(I),A<br>A(I),D(,R) | D(,R),A(I)<br>D(,R),D(I,R) | D(I,R),A<br>D(I,R),D(,R) |
| 3 | MVI,CLI,SBN<br>SBF,TBN,TBF<br>TIO,SNS,LIO<br>BC | A,I | | D(,R),I | |
| | L,ST,A,LA<br>SCP*,LCP* | A,R | | D(,R),R | |
| 4 | APL,SVC* | I | | | |
| 5 | HPL,SIO,CCP* | I,I | | | |
| 6 | JC | A,I | | | |

*Model 15 only.

The following codes are used to describe the possible operand formats:

| CODE | MEANING | ACCEPTABLE FORM |
|---|---|---|
| A | Address | Relocatable expression, absolute expression, or self-defining value. |
| D | Displacement | Relocatable expression, absolute expression, or self-defining value. |
| L | Length | Absolute expression or self-defining value. |
| R | Register | Absolute expression or self-defining value. |
| I | Immediate Data (bit masks, condition bit masks, or control bits to be used in the instruction) | Absolute expression or self-defining value. |

Figure 8. Operand Format by Group

In groups 3, 5, and 6, the Q-code operand is always on the right. See *Appendix A. Machine Instructions* for an explanation of Q codes.

## ASSEMBLER INSTRUCTION STATEMENTS

When writing a program the programmer uses two types of statements: executable instructions and instruction statements to the assembler. The executable instructions are the machine instruction statements. These are symbolic representations of the programmer's logic, such as branch, move, or compare, which are translated into machine language by the assembler.

Assembler instruction statements, on the other hand, do not generate executable machine codes. They are instructions that control specific assembler functions. These instructions are used to set up areas in storage, to define data, to equate symbols, and to control program listings, location counter, statement formats, and types of addressing. In the remainder of this section, the individual assembler instruction statements are discussed.

### Symbol Definition Instruction

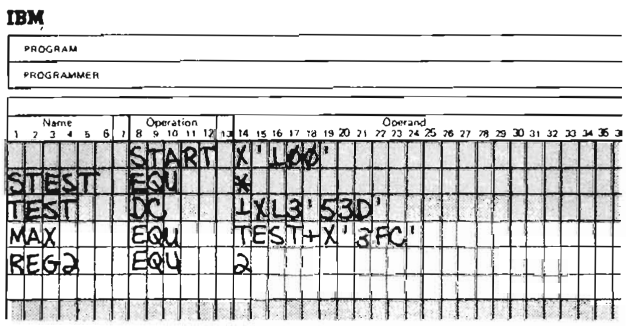
#### *EQU–Equate Symbol*

The EQU instruction is used to equate symbols with register numbers, immediate data, or other arbitrary values. The EQU instruction defines a symbol by assigning to it the length and value of the expression in the operand field of the EQU instruction. The EQU instruction has the following format:

| NAME | OPERATION | OPERAND |
|---|---|---|
| symbol | EQU | an expression |

The expression in the operand field can be either absolute or relocatable. Any symbol appearing in the operand field must have been previously defined. Figure 9 illustrates how this instruction can be used to equate a symbol with the contents of the operand.

In Figure 9, MAX has the value of TEST + X'3FC' (X'102+X'3FC' or X'4FE') any time it is used in the program. The symbol STEST has the value of the first (left most) byte of the data area reserved by the DC instruction. Since the symbol on the DC (TEST) has the value of the rightmost byte, this type of EQU is useful for addressing the leftmost byte. The symbol REG2 in any statement is the same as using the number 2.


IBM

PROGRAM

PROGRAMMER

| Name | Operation | Operand |
|---|---|---|
| | START | X'1ØØ' |
| STEST | EQU | * |
| TEST | DC | 1XL3'53D' |
| MAX | EQU | TEST+X'3FC' |
| REG2 | EQU | 2 |

Figure 9. EQU Assembler Instruction

### Data Defining Instructions

Two data defining instruction statements are available: Define Constant (DC), and Define Storage (DS). These instructions are used to enter data constants and to reserve areas in storage. Each instruction can have a name field entry (symbol) to which other instructions can refer.

*DC—Define Constant*

The DC instruction is used to initialize a storage location with a desired value. The IBM System/3 Basic Assembler Language allows six types of constants: storage address, binary, character, decimal, hexadecimal, and integer. The format of the DC instruction is as follows:

| NAME | OPERATION | OPERAND | | | |
|---|---|---|---|---|---|
| symbol or blank | DC | Duplication Factor (1) | Type (2) | Length (3) | Constant (4) |

Notice that the operand of the DC statement consists of four subfields. The first three describe the constant and the fourth provides the constant. The only blanks permitted within an operand field are blanks embedded in a character constant. The symbol that identifies the DC statement receives the value of the address of the *rightmost* byte of the area defined by the statement.

*Subfield 1–Duplication Factor:* This subfield enables the programmer to repeat the constant in storage. The constant will be generated the number of times indicated by the entry in the first subfield. This entry can be any unsigned, nonzero, decimal value, 1 through 65535. If this subfield is omitted, a duplication factor of 1 is assumed. This duplication factor is applied after the constant is fully assembled. If duplication is specified for an address constant containing a positive location counter reference, the value of the location counter used in each duplication is increased by the length of the constant.

*Subfield 2–Type:* This subfield defines the form of the constant being entered. From the type specification, the assembler determines how it is to interpret the constant and translate it into the appropriate machine format. The type entry is specified by one of the letter codes A, B, C, D, X, or I (see *Subfield 4 – Constant* for related meanings). *The type entry is required.*

*Subfield 3–Length:* The third subfield describes the number of bytes required by the constant. The entry for this subfield may be written two ways:

1. Ln, where n is an unsigned, nonzero, decimal value. The value of n is as follows:

   n = 1-256 for I, B, C, X constants

   n = 1-31 for the D constant

   n = 1-3 for an A constant

2. L (absolute expression), where an absolute expression is enclosed in parentheses. The value limits for the absolute expression are the same as those for n in the previous paragraph. A location counter reference is not allowed in this expression.

The total area allocated for this constant is the result of: Duplication Factor * Length=Total Area. *The length entry is required.*

*Subfield 4–Constant:* This subfield supplies the constant that was described in subfields 1 through 3. In general, the address constant (type A) is enclosed in parentheses, while the data constants (types B, C, D, I, and X) are enclosed in apostrophes. *An entry in the constant subfield of a DC statement is always required.*

*Address Constant (A):* This constant is used to load an address into a storage area.

*Example:* SYMBOL DC AL2 (BETA)

In this example, the address represented by the symbol BETA will be stored in the 2-byte field addressed by SYMBOL. The full 24-bit final expression result is truncated on the left to fit the length of the constant. The maximum length of an address constant is 3.

*Binary Constant (B):* This constant is used to create bit patterns and masks.

*Example:* SYMBOL DC 1BL1'10011'

The byte of storage addressed by SYMBOL will contain 00010011. Truncation or padding with binary zeros occurs on the left if the constant is not the length specified. This constant is enclosed in apostrophes. Each digit within the apostrophes represents a single bit in storage, and each eight bits specified will occupy one byte of storage.

*Character Constant (C):* This constant can be used to place a string of characters in storage.

*Example:* SYMBOL DC 1CL17'PLANT 5 PAYROLL'

The byte of storage addressed by SYMBOL will contain a blank, and the byte of storage addressed by SYMBOL-16 will contain the character P.

*Note:* Two blanks have been padded on the right of the character string.

If the constant is not the specified length, truncation or padding with blanks will occur on the right. Each character (including blanks) within the apostrophes will occupy a byte of storage. If an apostrophe occurs within the string of characters, it must be represented by a double apostrophe.

*Decimal Constant (D):* This constant can be used for arithmetic purposes.

*Example:* SYMBOL DC DL5'125.66'

This constant will appear in zoned-decimal form in a 5-byte storage field, addressed by SYMBOL. The decimal point is used only as a convenience for the programmer, and is *not* assembled into the constant. The value of the constant is calculated without the decimal point. Truncation or padding with decimal zeros occurs at the left of the field, if necessary. Signed decimal constants are permitted, making it possible to have a decimal constant with a negative value. Each decimal digit will occupy one byte of storage.

*Hexadecimal Constant (X):* This constant is used to associate a hexadecimal value with a symbol in a defined area in storage.

*Example:* SYMBOL DC 1XL6'8AC14'

The 6-byte field addressed by SYMBOL will contain the following 12 hexadecimal digits: 00000008AC14.

Truncation or padding with hexadecimal zeros occurs at the left. Each two digits between apostrophes will occupy one byte of storage.

*Integer Constant (I):* This constant is used for fixed-point binary arithmetic.

*Example:* SYMBOL DC 1IL2'-7'

A negative number may be used for an I constant. The negative constant is placed in storage in its two's-complement form. This example would appear in storage in bit form as 1111111111111001. There is always a positive equivalent to a negative constant; in the above example, it is hexadecimal FFF9 or decimal 65,529. The range of I constants must be within $-2^{32}+1$ to $2^{32}-1$. If the number is positive, it is padded on the left with 0-bits. If the number is negative, it is padded on the left with 1-bits.

*DS–Defines Storage*

The DS instruction is much like the DC instruction. It assigns a symbol to an area of storage. Unlike the DC instruction, the DS instruction only reserves the area of storage, it does not insert data. A constant subfield cannot be used with a DS statement. The following illustration shows the DS format.

| NAME | OPERATION | OPERAND | | |
|---|---|---|---|---|
| symbol or blank | DS | duplication factor | type | length |

A duplication factor of zero can be used in a DS statement if the programmer wishes only to assign a length to its corresponding symbol. The symbol will be given the value of the current location counter minus one. The type and length subfields must follow the same rules as for the DC statement.

The duplication factor can be used by the programmer to specify a reserved area larger than 256 bytes.

*Example:* SYMBOL DS 3CL100

This instruction would reserve a 300-byte area, which would be referenced on the right by the name entry SYMBOL.

### Listing Control Instructions

The listing control instructions aid the programmer in documenting his assembler listing. These instructions are TITLE, EJECT, SPACE, and PRINT.

#### *TITLE – Identify Assembly Output*

The TITLE instruction enables the programmer to identify assembled object cards and assembler listings.

| NAME | OPERATION | OPERAND |
|---|---|---|
| label or blank | TITLE | a sequence of characters enclosed in apostrophes |

The name field entry can consist of a maximum of six characters. The first character may be numeric. The contents of the name field in the first TITLE card is punched into the sequence field of all object cards produced by the assembler. This name field entry also appears in all listing header fields.

The name on the TITLE statement is not the object program name, but may be the same as the object program name. See *START – Start Assembly.* The name field entry is used only for identification and may not be referenced by the program.

The operand field contains a sequence of characters enclosed in apostrophes. Any embedded apostrophes must be represented by a double apostrophe. The contents of the name and operand fields are printed at the top of each page of the assembler listing.

A program can contain more than one TITLE statement. When a new TITLE statement is read, the listing is advanced to a new page before the new heading is printed. The name fields of all subsequent TITLE statements are ignored by the assembler. The TITLE instruction is not listed on the assembler listing, but it does increase the statement counter by one. Figure 10 shows an example of the TITLE statement.

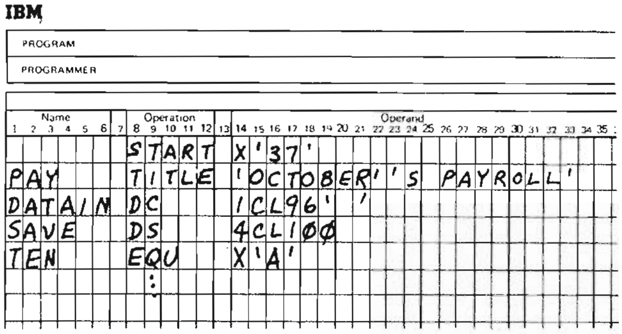



Figure 10. Use of the TITLE Statement

#### *EJECT – Start New Page*

The EJECT instruction causes printing to begin at the top of a new page, under the page heading. Through the use of the EJECT statement, the programmer can separate routines in the assembler listing. The format of the EJECT assembler instructions is as follows:

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | EJECT | Not Used |

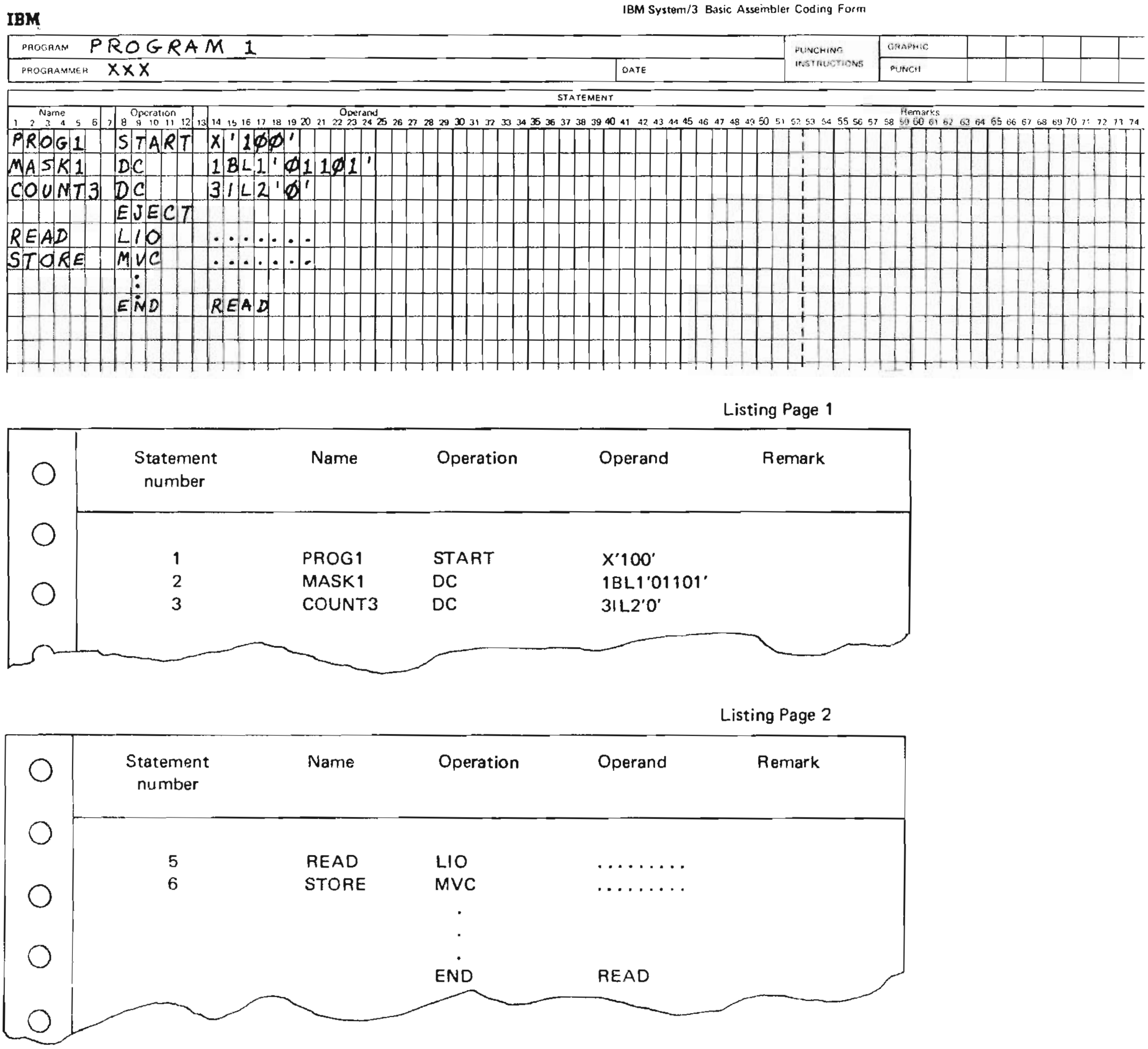
In Figure 11, the EJECT instruction is used to separate executable instructions from the data-defining assembler statements. The EJECT instruction is not listed on the assembler listing, but it does increase the statement counter by one. The coding example in Figure 11 shows the position of EJECT. Note that the corresponding statement number (4) has been omitted in the listing. Statement number 5 appears at the top of the next page, under the heading.

*SPACE – Space Listing*

This instruction is used to insert one or more blank lines between statements in the assembler listing:

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | SPACE | decimal value or a blank |

An unsigned decimal value is used to specify the number of blank lines that are to be inserted. If the operand contains a blank, a zero, or a 1, one blank line will be inserted. If the value of the operand exceeds the number of lines remaining on the current page, the instruction has the same effect on the listing as an EJECT statement. The SPACE instruction, like the EJECT instruction, is not listed on the assembler listing, but does increase the statement counter by one.



IBM System/3 Basic Assembler Coding Form

PROGRAM PROGRAM 1
PROGRAMMER XXX
DATE
PUNCHING INSTRUCTIONS — GRAPHIC / PUNCH

| Name | Operation | Operand |
|---|---|---|
| PROG1 | START | X'1ØØ' |
| MASK1 | DC | 1BL1'Ø11Ø1' |
| COUNT3 | DC | 3IL2'Ø' |
| | EJECT | |
| READ | LIO | ....... |
| STORE | MVC | ....... |
| | : | |
| | END | READ |

Listing Page 1

| Statement number | Name | Operation | Operand | Remark |
|---|---|---|---|---|
| 1 | PROG1 | START | X'100' | |
| 2 | MASK1 | DC | 1BL1'01101' | |
| 3 | COUNT3 | DC | 3IL2'0' | |

Listing Page 2

| Statement number | Name | Operation | Operand | Remark |
|---|---|---|---|---|
| 5 | READ | LIO | ......... | |
| 6 | STORE | MVC | ......... | |
| | | . | | |
| | | . | | |
| | | END | READ | |

Figure 11. EJECT Instruction

*PRINT—Print Optional Data*

The programmer can control the printing of an assembly listing by using the PRINT instruction. A program can have any number of PRINT instructions. Each PRINT instruction controls the listing until the next PRINT instruction is encountered.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | PRINT | operand |

The operand field can include entries from the following groups (one or two operands for the Model 10, one, two, or three operands for the Model 12 and the Model 15):

1. ON—A listing is printed.
   OFF—No listing is printed.

2. DATA—Constants are printed out in full on the assembler listing.
   NODATA—Only the leftmost 8 bytes of the constants are printed on the assembler listing.

3. (Model 12 and Model 15 only)
   GEN—Print statements generated by the macro processor if not overridden by other listing control statements.
   NOGEN—Suppress printing of statements generated by the macro processor.

Operand entries must be separated by a comma.

The ON, GEN and DATA conditions are assumed by the assembler unless otherwise specified by a PRINT instruction. If an operand is omitted, it is assumed to be unchanged and continues according to its last specification. Both of the examples in Figure 12 would cause a listing to be printed with only the leftmost 8 bytes of the constants appearing in the listing.

IBM

PROGRAM

PROGRAMMER

```
       PRINT ON,NODATA
MAX    DC    5CL3'ABC'
         :
```

IBM

Or

PROGRAM

PROGRAMMER

```
       PRINT NODATA
MAX    DC    5CL3'ABC'
         :
```

Figure 12. The PRINT Statement

**Program Control Instructions**

*ICTL—Input Format Control*

The ICTL statement permits the programmer to change the normal bounds of the source program statements. When included, the ICTL instruction must precede all other source statements. This instruction can be used only once during a program. An invalid or mispositioned ICTL statement causes termination of the assembly.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | ICTL | two decimals in the form of B,E |

The term B specifies the beginning column and the term E specifies the ending column of the source statement. The beginning column must be within columns 1-48. The ending column must be within columns 49-95. The column after the ending column must be blank.

When an ICTL statement is not included in a source program, the beginning column is assumed to be column 1, and column 87 is assumed to be the ending column. Figure 13 is an example of the ICTL instruction. In Figure 13, the name field would start in column 14 and the remark field would end in column 80.

IBM

PROGRAM PROGRAM X3

PROGRAMMER XXX

```
        ICTL  14,80
             PROGX3 START X'100'
             MAX2   EQU   2
             SYMBOL DC    1CL6'SYMBOL'
                      :
                    END
```

Figure 13. The ICTL Statement

*ISEQ—Input Sequence Checking*

The ISEQ instruction is used to check the sequence of source cards. Sequence checking begins with the first card after the ISEQ instruction. The first sequence entry is taken from the sequence identification field of the ISEQ statement. The sequence entry on the next card is then compared to the previous sequence value. The ISEQ assembler statement has the following effect:

1. The sequence entries on source-statement cards are checked for ascending order.

2. Statements that are out of order and statements without sequence entries are flagged in the assembler listing.

3. The total number of flagged statements is noted at the end of the assembler listing.

For example, with the sequence values 13, 27, 31, 6, 8, 45, 47, ƀ and 48, the card numbered 6 and the card without a sequence value would be out of sequence. The assembly does not stop due to a card being out of sequence order. In this example, the card numbered 6 and the card without a sequence entry would be flagged in the error field of the listing. If sequence checking is requested, there is a statement at the end of the listing showing that two cards were out of sequence.

The assembler will not check the sequence unless requested to do so by use of the ISEQ statement.

The following is the ISEQ instruction format:

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | ISEQ | two decimal values in the form L, R; or blank |

The operand entries, L or R, specify the leftmost (L) and rightmost (R) columns of the field to be sequence checked. The value of L must be within the range of 73 through 96 (inclusive). The length of the sequence field may be from 1 to 8. If the programmer wants to discontinue sequencing, an ISEQ instruction card with a blank operand is inserted.

The sequence field must be separated from the last column of the source statement by at least one blank position. The last column of the source statement is column 87 unless otherwise specified by the ICTL assembler statement. The sequence field must not appear before the last column +1 of the source statement. If the sequence field is to start before column 89, the ICTL statement must be used to redefine the beginning and end of the source statement. For example:

| | | |
|---|---|---|
| ICTL | 1, 71 | Source statement is defined within columns 1-71 |
| ISEQ | 73, 80 | Sequence field is in columns 73-80 |

*START–Start Assembly.*

The START instruction may be used to initialize the location counter to a desired value at the beginning of a program. The format of the START instruction is:

| NAME | OPERATION | OPERAND |
|---|---|---|
| symbol | START | a self-defining value or blank |

The assembler uses the single self-defining term in the operand as the initial location-counter value. For example, either of the START instructions in Figure 14 could be used to indicate an initial assembly location of 2040.

If the operand of a START instruction is blank, the location counter is initialized with a value of zero. If neither an ORG nor a START instruction is used to initialize the location counter, the initial value is also zero.

A START instruction must not be preceded by any statement that affects or is dependent upon the setting of the location counter.

The name entry in the name field of a START instruction provides the program with an identifier name called the module name. The module name may be the same as the first TITLE statement.

*Note:* Certain naming restrictions apply when assigning names for your program. For more information on naming restrictions, see *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512. *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130, *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077 (Program Number 5704-AS1), or *IBM System/3 Model 15 System Control Programming Concepts and Reference Manual,* GC21-5162 (Program Number 5704-AS2).

This program name may be used for program linkage. If the START card is not included in the program, or if the name field is blank, a default program name is assigned. See the MODULE NAME MISSING diagnostic in *Appendix C. System/3 Assembler – Source Language Error Codes and Diagnostics.*

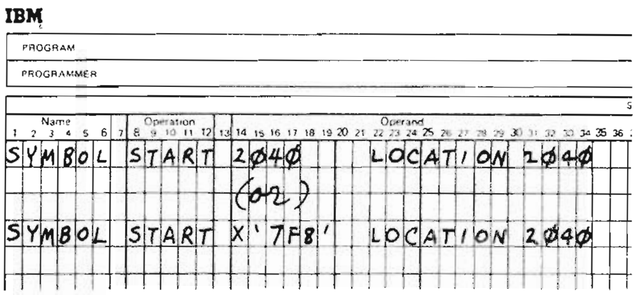



**Figure 14. Using START to Initialize the Location Counter**

*ORG—Set Location Counter*

The ORG statement sets the location-counter value.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | ORG | blank operand or an expression A optionally followed by two absolute expressions in the form A, B, C |

The location counter is set to the smallest value greater than or equal to A which is C more than a multiple of B. In the following example, A can be either a relocatable or absolute expression; B and C must be absolute expressions. The default values for B and C are 1 and 0, respectively. If the second operand (B) is omitted, the third operand (C) must also be omitted.

| *Current Location Counter* | A | B | C | *New Location Counter* |
|---|---|---|---|---|
| 275 | * | 100 | 50 | 350 |
| 340 | * | 10C | 50 | 350 |
| 350 | * | 100 | 50 | 350 |
| 504 | * | 256 | 0 | 512 |
| 750 | 1000 | — | — | 1000 |

All symbols used in the expression A must have been previously defined. The value specified by the ORG statement must be greater than or equal to the starting location-counter value.

If previous ORG statements have reduced the location-counter value for the purpose of redefining the current program, an ORG instruction with a blank operand is used to set the location counter to the previous maximum assigned address plus one (see Figure 15).

| Location Counter | Address | Name | Operation | Operand |
|---|---|---|---|---|
| 0064 | | PROG4 | START | 100 |
| 0064 | 0069 | SYMBOL | DC | 1CL6' ' |
| 006A | *0325 | FILLIN | DS | 7CL100 |
| 00CE | | | ORG | FILLIN-599 |
| 00CE | 01F9 | DATA | DC | 150CL2'AZ' |
| 0326 | | | ORG | |
| | | | : | |
| | | | END | |

* Previous High Address

Figure 15. Using ORG to Control the Location Counter

*USING — Use Register for Base-Displacement Addressing*

The USING statement specifies the register to be used for base-displacement addressing and also specifies the base address that the assembler will assume to be in that register at object time. The USING statement does not load the base address into the register specified. This must be done by the programmer before the register can be used for base-register displacement addressing. See *Addressing* in this section.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | USING | V,R |

In the preceding format, term V represents an expression. Term R represents an absolute expression with a value of 1 or 2. Term R specifies the index register assumed to contain the base address represented by the term V. The programmer has the option of changing the base register or base address at any time by the insertion of another USING statement. Two USING statements enable the programmer to use the two index registers as base registers to two different portions of main storage.

In Figure 16, register 2 is loaded with the address of ADRES1, which will be used as the base address in instructions following the USING statement.

IBM

| Name | Operation | Operand |
|---|---|---|
| PROG1 | START | |
| | : | |
| | LA | ADRES1,2 |
| | USING | ADRES1,2 |
| | : | |

Figure 16. Specifying a Base Register With the USING Statements

*DROP — Drop Base Register*

The DROP instruction specifies a base register that is no longer to be used as a base register. The programmer can reinitiate the base register with another USING instruction.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | DROP | specified register |

The operand must contain an absolute expression of either 1 or 2. This absolute expression represents the register that is no longer to be used as a base register. The contents of the register are unaffected by the DROP instruction. Figure 17 shows an example of the DROP instruction. Another USING statement is used to specify register 1 as the new base register.

```
PROG1  START
         :
       LA     ADRES1,2
       USING  ADRES1,2
         :
       DROP   2
       LA     ADRES2,1
       USING  ADRES2,1
```

**Figure 17. Example of the DROP Statement**

*ENTRY — Identify Entry Point to Program*

This instruction identifies symbols, defined in the current program, which can be used as entry points from other programs.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | ENTRY | any relocatable symbol found in the name field of the current program |

The symbol used in the ENTRY operand can also be referenced by any other program provided that program uses the same symbol in the operand of an EXTRN statement. See the example given in the discussion of EXTRN for additional information on the use of ENTRY.

*EXTRN — Identify External Symbols*

This instruction identifies symbols, used in the current program, which are defined in another program. Each symbol in the operand of an EXTRN statement must be identified by an ENTRY statement or be the module name in some other program.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | EXTRN | one relocatable symbol not found in the name field of the current program, optionally followed by an absolute expression in parentheses |

The external symbol cannot be used in a Name field in the same program that describes that symbol as an EXTRN.

An EXTRN subtype can be specified for the EXTRN symbol by following the symbol with an absolute expression enclosed in parentheses. The value of the absolute expression cannot be less than zero nor more than 255. Any symbol in the expression must have been previously defined. For an explanation of the subtype values and their meanings, see *IBM System/3 Overlay Linkage Editor Reference Manual*, GC21-7561.

Figure 18 shows how ENTRY and EXTRN can be used to make two or more programs act as one main program through sharing data and control. The main program defines symbols A, B, and C and identifies them as entry points. These same symbols are identified as EXTRNs (external symbols) in the subroutine. This allows the subroutine to use these symbols just as it would if the symbols had been defined in the subroutine. SUBR01, on the other hand, is defined and identified as an entry point by the subroutine and as an EXTRN, external symbol, by the main routine. These four symbols – A, B, C, and SUBR01 – can now be used interchangeably by both the main routine and the subroutine.

The main routine has control first. It executes instructions and then branches to SUBR01 which is defined as an entry point in the subroutine. Instructions in the subroutine are executed. Notice that the subroutine uses symbols A, B, and C which were defined in the main routine. Control is then passed back to the main routine.

*Note:* The actual resolution of symbols between programs is not performed by the assembler.

IBM

PROGRAM

PROGRAMMER

```
MAIN     START  0
         ENTRY  A
         ENTRY  B
         ENTRY  C
         EXTRN  SUBR01
ENTRY    EQU    *
           S
         B      SUBR01
            S
A        DC     DL4'1234'
B        DC     DL4'5678'
C        DS     CL5
            S
         END    ENTRY
```

Main Routine

IBM

PROGRAM

PROGRAMMER

```
SUBR     START  0
         ENTRY  SUBR01
         EXTRN  A
         EXTRN  B
         EXTRN  C
SUBR01   ST     RETURN+3,8
         MVC    EDIT(5),MASK
         ZAZ    D(4),A(4)
         AZ     D(4),B(4)
         ED     EDIT(5),D
         MVC    C(5),EDIT
RETURN   B      0
MASK     DC     XL5'2020204B20'
EDIT     DS     DL5
D        DS     DL4
         END
```

Subroutine

Figure 18. Example of ENTRY and EXTRN Statements

*END–End Assembly*

The END instruction terminates assembly of the program. The operand of this instruction can contain an expression (usually a name field entry) which specifies the address to which control is to be transferred after the program is loaded. The END instruction must be the last statement in the program. The relocatable expression in the operand must not contain external symbols. The start-of-control address must be specified for programs loaded with the absolute loader.

| NAME | OPERATION | OPERAND |
|---|---|---|
| blank | END | a relocatable expression or a blank |

Figure 19. shows an END statement. In this example, the program receives control at the address corresponding to BEGIN when it is executed.

IBM

PROGRAM PROGRAM ONE

PROGRAMMER XXX

```
PROG1    START
            :
BEGIN    MVC    OUT,ABC(1)
            :
         END    BEGIN
```

Figure 19. Designating an Entry Point With the END Statement

### ASSEMBLER CONTROL STATEMENTS

Two control statements are used: The HEADERS statement and the OPTIONS statement. Up to 45 of these control statements may be used, in any order. Each statement is limited to six operands. All control statements must appear before any assembler source statements.

#### HEADERS Statement

The HEADERS control statement specifies control information other than output control information to the assembler. The programmer may specify a category level for the object module through the CATG operand, or the length of the control section for any subtype 4 or 5 EXTRNs in the assembler through the COML4 and COML5 operands. For an explanation of category levels and subtype 4 and 5 EXTRNs, see *IBM System/3 Overlay Linkage Editor Reference Manual,* GC21-7561.

The format of the HEADERS statement with the CATG operand is:

```
ƀHEADERSƀCATG-nnnnn
```

Column 2 or greater | At least one blank | Decimal category level

nnnnn

*nnnnn* is a one to five character decimal string whose value must be less than 00256. If more than one CATG operand appears in the assembler control statements, the value of the last valid operand is used for the module category level. The module category level is placed in the module ESL record.

The format of the HEADERS statement with the COML4 and COML5 operands is:

```
ƀHEADERSƀCOML4-nnnnn, COML5-nnnnn
```

Column 2 or greater | At least one blank | Decimal control section length

*nnnnn* is a one to five character decimal string whose value must be less than 65536. If more than one COML4 or COML5 operand is present in the assembler control statements, the length in the last valid operand is used for the appropriate subtype control section length. The lengths specified are placed in the ESL records for the subtype 4 or 5 EXTRNs.

#### OPTIONS Statement

An OPTIONS statement is a control statement for assembler control options. All OPTIONS statements must precede the source deck. The user may specify the following assembler options on OPTIONS statements: DECK, NODECK, LIST, NOLIST, XREF, NOXREF, REL, NOREL, OBJ, OBJ(T), OBJ(P), NOOBJ. XBUF-nnnnn and NOXBUF are also available to users having program 5704-AS2. They may appear on one statement in any order, but must be separated by commas. If the programmer prefers, separate statements may be used for each option. The OPTIONS keyword must start in column 2 or higher (the preceding column must be blank), and there must be one or more blanks between the keyword and the selected options. Blanks are not allowed between the selected options.

The following example shows options appearing on one statement:

```
ƀOPTIONSƀDECK, LIST, NOXREF, REL
```

Column 2 or greater. | At least one blank between keyword and options.

More than one OPTIONS statement may be used. In the following example, three statements are used:

```
ƀOPTIONSƀDECK
ƀOPTIONSƀLIST
ƀOPTIONSƀNOXREF
```

The following list provides a brief description of all the options available:

| *Option* | *Explanation* |
|---|---|
| DECK | The object program is punched. When an object program is punched, it is preceded by a // COPY OCL card and followed by a // CEND OCL card. These cards are provided for placing the object program in the R library with the library maintenance utility program ($MAINT). |
| NODECK | The object program is not punched. |
| LIST | The following sections of the assembler listing are printed (see *Assembler Listing* in this section for a description of the listings):<br>• Options information<br>• External symbol list<br>• Source and object program listing<br>• Diagnostic listing<br>• Error summary statements |
| NOLIST | Only the following listings are printed:<br>• Options information<br>• Any statements in error and the associated diagnostics<br>• Error summary statements<br>The NOLIST option overrides all assembler PRINT statements. |
| XREF | A cross-reference listing is generated. |
| NOXREF | A cross-reference listing is not generated. |
| REL | A relocatable object program is produced. |
| NOREL | An absolute object program is produced.<br>*Note:* Absolute object programs can only be used as stand-alone programs; that is, programs which are not dependent on any other disk management system program.<br>On the Model 10 an absolute loader will precede the absolute deck if DECK is specified and if MFCU2 is specified on the // PUNCH statement. On the Model 12 and Model 15, an absolute loader will precede the absolute deck if DECK is specified and if the SYSPCH device is MFCU, 1442, or MFCM (Model 15 only). The loader punched will program load only on the device type on which it was punched. A blank card is inserted between the absolute loader and the object program. This blank card and the OCL cards included with the object program do not affect the operation of the absolute loader and may be discarded.<br>To prevent cataloging of the absolute object program when NOREL is specified, you should specify NOOBJ. |
| OBJ or OBJ(T) | The object program is placed in the R library with a retain entry of temporary. |
| OBJ(P) | The object program is placed in the R library with a retain entry of permanent. |
| NOOBJ | The object program is not placed in the R library. (See *Placing Assembler Subroutines in R [Routine] Library* in this section.) |

If no OPTIONS statement is used, the assembly is processed as though DECK, LIST, REL, XREF, and OBJ had been specified. NOXBUF is also assumed with program 5704-AS2.

| | |
|---|---|
| XBUF-nnnnn | Specifies the size of the disk external buffers the user has requested. From one to five numeric digits may be used to specify the size of the disk external buffers (program 5704-AS2 only). External buffers should not be specified due to performance considerations if the program size including physical disk buffers does not exceed 56K. However, if external buffers are specified, they should equal the size of the physical disk buffers that normally would be set aside within the program. |
| NOXBUF | Specifies no external buffers are requested for the program (program 5704-AS2 only). |

If DECK or OBJ is entered on the OPTIONS statement and there are errors in the assembly, a halt is issued.

## OCL STATEMENTS FOR ASSEMBLER

The loading and running of a disk-system program, including the assembler, is done under control of a group of programs called *disk system management.* The user tells disk system management to run a program through the use of Operation Control Language (OCL) statements. It is necessary to have a set of OCL statements each time a program is run. This section discusses the OCL statements required for use of the assembler. For a complete discussion of OCL, see *IBM System/3 Model 10 Disk System Control Programming Reference Manual;* GC21-7512, *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130, *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077 (Program Number 5704-AS1), or *IBM System/3 Model 15 System Control Programming Concepts and Reference Manual* (Program Number 5704-AS2), GC21-5162.

The assembler language source program can be obtained from either a system input device, a source library entry, or the macro processor. If the source records are obtained from an 80-column device, they are padded with 16 blanks before being placed in the $SOURCE file. In this case, the user should provide an ICTL statement to prevent the assembler from processing the sequence field of the 80-column record.

### OCL For Loading the Assembler

*Source Program on System Input Device (Cards)*

Figure 20 is a sample set of OCL statements to load the assembler when the source program is on cards. The unit parameter (F1) on the // LOAD statement specifies where the assembler resides. The codes for the disk drive upon which the assembler resides are:

- R1 – drive 1
- F1 – drive 1
- R2 – drive 2
- F2 – drive 2

The first // FILE statement specifies the attributes and location of the file used for source program residence during the assembly process.

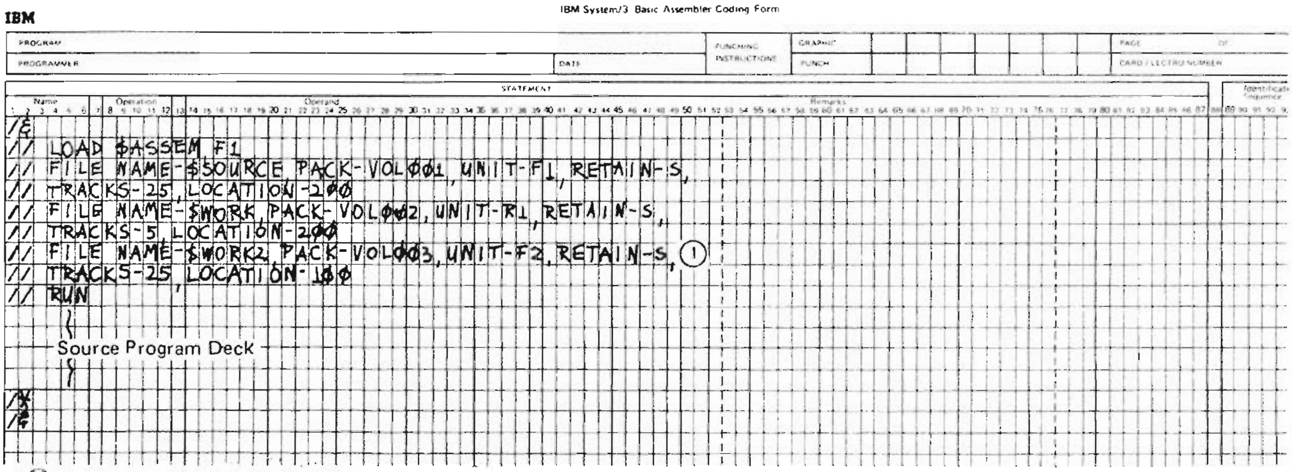
The second // FILE statement specifies attributes and the location of the file used for object output of the assembler. The third // FILE statement specifies attributes and location of the file used for assembler working storage during the assembler process.

The $WORK2 // FILE statement is optional on the Model 10 Disk System. If it is not supplied, the assembler allocates the work space. However, by specifying the proper placement of file locations, as in Figure 20, this file statement improves the performance of the assembler. It should, therefore, be specified.

In all three // FILE statements, the PACK and UNIT parameters indicate the location of the file named in the NAME Parameter. In addition to R1, F1, R2, and F2, the UNIT parameter can specify D1, D2, D3, and D4 for the Model 15. The RETAIN parameter should reflect a scratch file(s). The TRACKS parameter contains the number of tracks required for that file. The user should choose the number of tracks required in accordance with the space requirements charts in the *Assembly Time Data File Requirements* section. See *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130, and *IBM System/3 Model 15 System Control Programming Reference Manual* (Program Number 5704-AS1), GC21-5077, or *IBM System/3 Model 15 System Control Programming Concepts and Reference Manual,* GC21-5162, (Program Number 5704-AS2) for further information.

*Source Program in a Source Library*

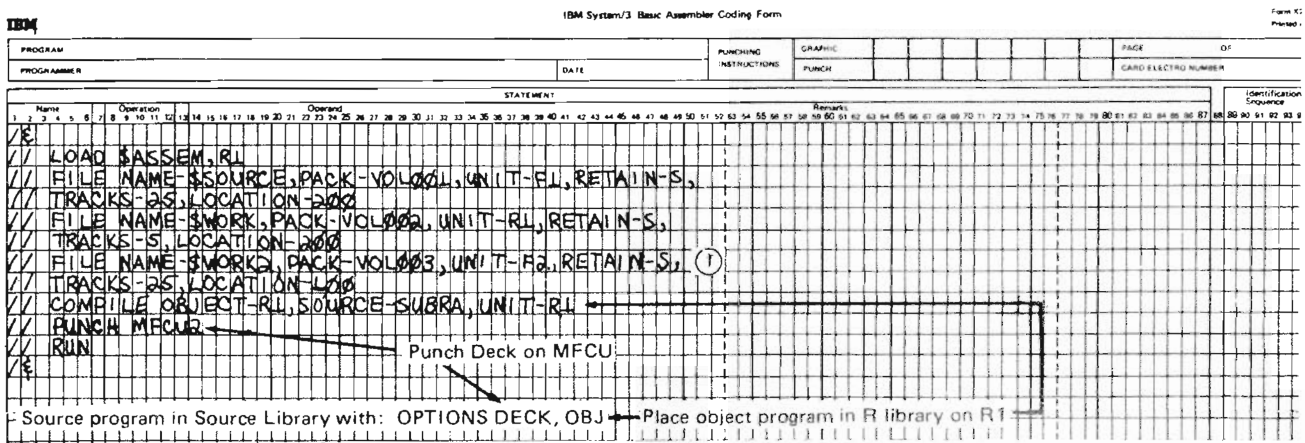
Figure 21 shows a sample set of OCL statements used when the source program is in the source library.



```
/&
// LOAD $ASSEM,F1
// FILE NAME-$SOURCE,PACK-VOLØØ1,UNIT-F1,RETAIN-S,
// TRACKS-25,LOCATION-2ØØ
// FILE NAME-$WORK,PACK-VOLØØ2,UNIT-R1,RETAIN-S,
// TRACKS-5,LOCATION-2ØØ
// FILE NAME-$WORK2,PACK-VOLØØ3,UNIT-F2,RETAIN-S, (1)
// TRACKS-25,LOCATION-1ØØ
// RUN
   Source Program Deck
/*
/&
```

(1) Optional on Model 10 Disk System

**Figure 20. Assembler OCL Statements (Source Program on Cards)**



```
/&
// LOAD $ASSEM,R1
// FILE NAME-$SOURCE,PACK-VOLØØ1,UNIT-F1,RETAIN-S,
// TRACKS-25,LOCATION-2ØØ
// FILE NAME-$WORK,PACK-VOLØØ2,UNIT-R1,RETAIN-S,
// TRACKS-5,LOCATION-2ØØ
// FILE NAME-$WORK2,PACK-VOLØØ3,UNIT-F2,RETAIN-S, (1)
// TRACKS-25,LOCATION-4ØØ
// COMPILE OBJECT-R1,SOURCE-SUBRA,UNIT-R1
// PUNCH MFCU2
// RUN
/&
```



(1) Optional on Model 10 Disk System

**Figure 21. Assembler OCL Statements (Source Program in Source Library)**

Note that the additional OCL statement // COMPILE is required. The following entries in the figure are optional:

PUNCH — This statement specifies where an object deck is punched. For more information on statement, see *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130, *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077 (Program Number 5704-AS1), or *IBM System/3 Model 15 System Control Programming Concepts and Reference Manual,* (Program Number 5704-AS2), GC21-5162.

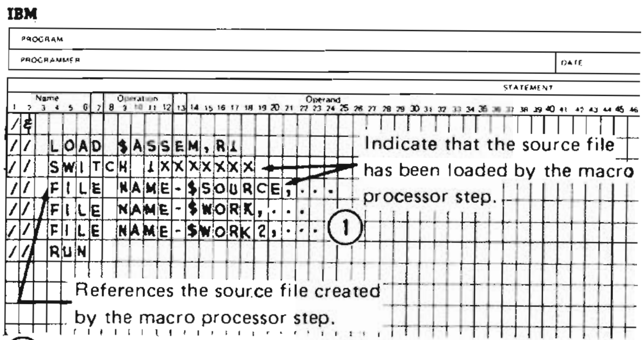
OBJECT operand — This operand is used to indicate to the assembler the library unit used when the OBJ option is used on the OPTIONS statement.

The // LOAD and // FILE statements are as described in the first example. The // COMPILE statement specifies both the location of the source library and the required source program within the library. The // COMPILE statement may appear at any position between // LOAD and // RUN.

*Macro Processor-Produced Source Program*

The macro processor creates a source program on the $SOURCE file. To indicate that the macro processer has already loaded the $SOURCE file, external indicator U1 must be turned on. This is done through a // SWITCH statement. If this indicator is on when the assembler is loaded, the $SOURCE file will not be loaded.

In the following OCL stream, the source program has been created on the $SOURCE file:



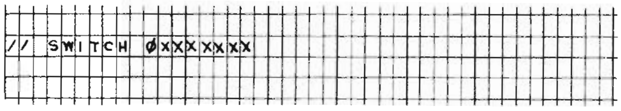

(1) Optional on Model 10 Disk System

*Note:* For more information on the macro processor, see *IBM System/3 Models 10 and 12 System Control Programming Macros Reference Manual,* GC21-7562, or *IBM System/3 Model 15 System Control Programming Macros Reference Manual,* GC21-7608.

*// SWITCH Considerations*

The external indicator U1 indicates that the macro processor has loaded the $SOURCE file and the source program is not in the input stream. If this indicator is on when the assembler is loaded, the $SOURCE file is not loaded.

When the $SOURCE file is to be loaded, external indicator U1 must be off. This can be ensured by entering the following statement after the assembler // LOAD statement:

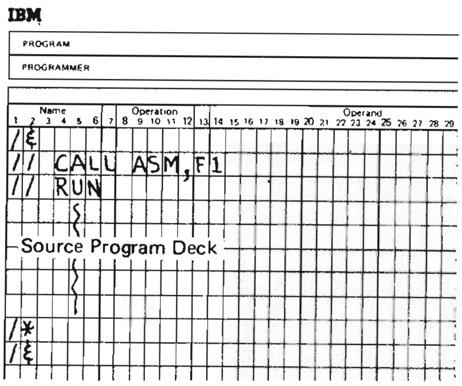



**OCL For Calling the Assembler**

It is possible for the user to store a portion of the OCL statements required for use by the assembler in a procedure library. They may then be called with a // CALL statement, thus reducing the number of written OCL statements required for each assembly. Examples are included for source programs on cards and for source programs in a source library on disk.

*Source Program on Cards*

If the source program is a deck of cards, the OCL cards necessary to assemble the program, and the order in which they must appear, are as follows:




In this example, ASM is the procedure name. F1 refers to the disk pack upon which the assembler OCL procedure is stored. In this case, it would be the fixed disk on drive one.

*Source Program in a Source Library*

If the source program is stored on disk in a source library, the OCL format must be as follows:

IBM

PROGRAM

PROGRAMMER

```
/¢
// CALL ASM,F1
// COMPILE SOURCE-SUBRA,UNIT-R1
// RUN
/¢
```

In this example, ASM is the procedure name and F1 refers to the fixed disk on drive 1. SUBRA is the name of the source program. The user must substitute his own source program name. R1 is the disk pack upon which the source library resides.

**Sample Assembler Procedure Stored in Procedure Library**

A sample assembler procedure is shown in Figure 22. The format is as it would appear in the procedure library. The // LOAD statement and // FILE statements are as described in preceding examples.

## OBJECT PROGRAM DESCRIPTION

The assembler converts the source program into a set of control information, machine language instructions, and data, all of which collectively are called an object program. There is one object program produced per assembly. Each object record is originally produced as a 64-byte field. If the object program is punched on the MFCU, it is translated into a 96-byte punch record (bytes 2 to 64 are translated 4 for 3 for punching; for every three 8-bit bytes, four card code characters are created). See *Object Program After Punch Conversion* in this section. Each object program generated by the assembler contains four types of records:

- HEADER record
- ESL (external symbol list) record
- TEXT-RLD (text-relocation directory) records
- END record

**Record Formats**

The following paragraphs describe the format of each record type.

*HEADER Record*

A HEADER record with record type H is added by the overlay linkage editor when it processes the assembler object program. The HEADER record format is:

| H | Object program information field |
|---|---|
| 1 | 2 … 64 |

- Byte 1 — Record type identifier H.
- Bytes 2-64 — Object program information field.

*ESL Record*

The object program name, that is the module name and all EXTRN and ENTRY symbols are placed in the ESL record. The ESL record format is:

| S | Length -1 | ESL Entries | X'00' |
|---|---|---|---|
| 1 | 2 | 3 … 62 | 63 64 |

- Byte 1 — Record type identifier S.
- Byte 2 — Length -1 of the ESL entries.
- Bytes 3-62 — ESL entries. Up to five MODULE, ENTRY, and/or EXTRN fields.
- Bytes 63-64 — Filled with hexadecimal zeros.

IBM — IBM System/3 Basic Assembler Coding Form

PROGRAM — PROGRAMMER — DATE — PUNCHING INSTRUCTIONS — GRAPHIC — PUNCH — PAGE OF — CARD ELECTRO NUMBER

```
// LOAD $ASSEM,F1
// FILE NAME-$SOURCE,PACK-VOLØØ1,UNIT-R1,RETAIN-S,TRACKS-1Ø
// FILE NAME-$WORK,PACK-VOLØØ2,UNIT-F2,RETAIN-S,TRACKS-1Ø
// FILE NAME-$WORK2,PACK-VOLØØ3,UNIT-R2,RETAIN-S,TRACKS-1Ø ①
// RUN
```

① Optional on Model 10 Disk System

Figure 22. Sample Assembler Procedure in Source Library

*TEXT-RLD Records*

Text records and RLD pointers are combined in this type of input record. The text portion of each record contains the object code for the program, while the RLD pointers indicate where the address constants and relocatable operands of the text are located. If the NOREL option has been selected on the OPTIONS control card, there will be no relocation indicators in the record. The format for the TEXT-RLD record is:

| T | Length-1 | Assembled Address | | Text→ X'00' ←RLD | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 64 |

- Byte 1 — Record type identifier T.
- Byte 2 — Length - 1 (of text only).
- Bytes 3-4 — Assembled address of the low order (rightmost) text byte in the record.
- Bytes 5-64 — Text starts at byte 5 and goes right, RLD starts at byte 64 and goes left. The leftmost end of the RLD section is marked by hexadecimal zeros, which fill the space between the Text and RLD sections. The end of text is always followed by at least one byte of X'00'.

*END Records*

The last record in each object program is an END record. It contains the entry address of the object program. If the user did not include an operand in his source program END statement, the object program END record generated by the assembler will contain the address X'FFFF'. The END record format is:

| E | Entry Address | END card program |
|---|---|---|
| 1 | 2-3 | 4 |

- Byte 1 — Record type identifier E.
- Bytes 2-3 — Entry address of the object program.
- Bytes 4-64 — Program to transfer control to Entry address.

**Object Program After Punch Conversion**

All four types of records (HEADER, ESL, TEXT-RLD, and END) assume the same format when they are punched into cards. The punched record format, using 96-column cards, is as follows:

| Record ID | Data Field | | Self Check Number | | Identification Sequence Field | |
|---|---|---|---|---|---|---|
| 1 | 2 | 85 | 86 | 88 | 89 | 96 |

| | |
|---|---|
| Column 1 | Record type identifier (H, S, T, or E). |
| Columns 2-85 | Data field, transformed 4 for 3. (For every three 8-bit bytes, four card code characters are created for System/3 96-column cards.) |
| Columns 86-88 | A 2-byte self check number transformed 4 for 3, to 3 bytes. |
| Columns 89-96 | Identification/sequence field. |

The punched record format, using 80-column cards, is as follows:

| Record ID | Data Field | | Blank | | Self Check Number | | Identification Sequence Field | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 64 | 65 | 69 | 70 | 72 | 73 | 80 |

| | |
|---|---|
| Column 1 | Record type identifier (H, S, T, or E). |
| Columns 2-64 | Data field, bytes 2 to 64 of the object record. |
| Columns 65-69 | Blank. |
| Columns 70-72 | A 2-byte self check number transformed 4 for 3, to 3 bytes. |
| Columns 73-80 | Identification/sequence field. |

*Note:* When an object module is punched, it is preceded by a // COPY OCL card and followed by a // CEND OCL card. These cards are provided for placing the object module in the R library with the Library Maintenance program ($MAINT).

## ASSEMBLY TIME DATA FILE REQUIREMENTS

There are three data files necessary at assembly time:

1. Source file (NAME-$SOURCE)

2. Object file (NAME-$WORK)

3. Work file (NAME-$WORK2)

Model 10 Disk System: These files must be located on 5444 disk drives. If a // FILE statement is not provided for $WORK2, the assembler allocates its own work space.

Model 12: These files must be located on the simulation area.

Model 15: These files must be located on either 3340, 5444, or 5445 disk drives.

### Source File ($SOURCE)

The source file is used by the assembler for storage of the source program. During the job initialization procedure, a disk system management program places the source program in the source file (if the macro processor has not loaded the file). The source records are obtained from either the system input device or a source library using the // COMPILE statement. (See *OCL statements for Assembly* in this section.) Each source record contains 96 bytes, so that eight records occupy three disk sectors in the source file. (One sector = 256 bytes, and is the smallest addressable unit on a disk.) Figure 23 is a source file space requirements table showing how many tracks are required for the size of the source program indicated.

If the assembler is processing a source file created by the macro processor, the // FILE statement for $SOURCE must correspond to the $SOURCE file produced in the macro processor run.

### Object File ($WORK)

The object file is used by the assembler for intermediate storage of the object program. The object records are stored in four 64-byte entries per sector. (See *Object Program Before Conversion* in this section.) Because each track in the object file can contain 96 records on the 5444, 80 records on the 5445, or 192 records on the 3340, two tracks usually are sufficient for most assemblies.

### Work File ($WORK2)

The work file is a scratch file used by the assembler throughout the assembly process for intermediate data storage. The file contains four types of data:

1. Intermediate text

2. Symbol table entries

3. Cross-reference data

4. Error information

#### *Intermediate Text*

Intermediate text is made up of fixed length (10-byte) records. The number of fixed length records is variable for each source statement, and is dependent on the statement type and the contents of the operand field.

The following rules can be used to determine intermediate text file requirements. (The rules apply only to error-free source statements. A statement that contains errors generally requires less storage space.)

*All Instructions:*

- One record for each machine or assembler instruction, or comment statement.

- One record if there is a name field entry.

*Machine Instructions:* One additional record for each term in the operand field.

| Source Program Size (Statements) | Number of Tracks Required | | |
|---|---|---|---|
| | 5444 * | 5445 | 3340 |
| 100 | 2 | 2 | 1 |
| 200 | 4 | 4 | 2 |
| 300 | 5 | 6 | 3 |
| 400 | 7 | 8 | 4 |
| 500 | 8 | 10 | 4 |
| 600 | 10 | 12 | 5 |
| 700 | 11 | 14 | 6 |
| 800 | 13 | 15 | 7 |
| 900 | 15 | 17 | 8 |
| 1000 | 16 | 19 | 8 |

*Or simulation area

Figure 23. Source File Space Requirements Chart

*Assembler Instructions:*

- END, ENTRY, EQU, EXTRN, ORG, USING – One additional record for each term in the operand field.
- ISEQ, PRINT, SPACE, START – One additional record for each instruction.
- TITLE – Additional records = N/8 (plus one for any non-zero remainder); where N is the number of characters in the TITLE operand field.
- DS/DC
  - One additional record for duplication factor (default or specified value).
  - One additional record for each term in the length specification.
- DC
  - Address constant–One record for each term in the address constant expression.
  - All other constants–Additional records - N/8 (plus one for any nonzero remainder); where N is the number of bytes required to contain the converted constant plus one.

Figure 24 is a sample list of instructions together with the intermediate text space requirements for each.

| | | Text Space |
|---|---|---|
| DECK | START 0 | 3 |
| ENTRY | SLC A(2),A | 5 |
| | MVC A(2),CON1 | 4 |
| | ALC A(2),CON2 | 4 |
| | HPL X'FF',X'FF' | 3 |
| A | DS CL2 | 4 |
| CON1 | DC IL2'500' | 5 |
| CON2 | DC IL2'-320' | 5 |
| | END ENTRY | 2 |

Figure 24. Intermediate Text Space Requirements

*Symbol Table Entries*

Whenever a symbol is used in the name field of an instruction (except a TITLE statement) it becomes a symbol table entry. When the assembler user requests a cross reference, all symbol table entries are added to the work file immediately after the intermediate text. The symbol table entries are also 10-byte, fixed-length records. Assuming an average of one name entry for every four source statements, one sector per 100 source statements is required.

*Cross-Reference Data*

Cross-reference data is written in the same area as the intermediate text and symbol table entries and does not impose any additional space requirements.

*Error Information*

Each statement in error requires a 10-byte error record; therefore, a track will contain at least 600 error records.

*Work File Space Requirements*

Figure 25 is a work file space requirements table showing the number of tracks required for the number of source statements indicated. The requirements for intermediate text and symbol table entries are summed to get the table values. Approximately 40 sectors per 100 source statements are needed to cover most varieties of source statements. If a $WORK2 // FILE statement is not provided on the Model 10 disk system assembler, the source file ($SOURCE) size is used for the work file size.

| Source Program Size (Statements) | Number of Tracks Required | | |
|---|---|---|---|
| | 5444* | 5445 | 3340 |
| 100 | 2 | 2 | 1 |
| 200 | 4 | 4 | 2 |
| 300 | 6 | 6 | 3 |
| 400 | 7 | 8 | 4 |
| 500 | 9 | 10 | 5 |
| 600 | 11 | 12 | 6 |
| 700 | 12 | 14 | 6 |
| 800 | 14 | 16 | 7 |
| 900 | 16 | 18 | 8 |
| 1000 | 18 | 20 | 9 |

*Or simulation area

Figure 25. Work File Space Requirements Chart

## OPERATING PROCEDURES

### Placing Assembler Subroutines in R (Routine) Library

Assembler subroutines can be placed on disk in the R library by two methods.

1. Punching an object deck and using the Library Maintenance program ($MAINT) to place it in the R library.

2. Specifying OBJ in the OPTIONS statement to place the object program directly into the R library. The retain entry can be either temporary or permanent.

For more information on the OCL and utility control statements needed to use $MAINT, see *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130, or *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077.

#### *Placing a Punched Object Program in the R Library*

In the sample procedure shown below, the subroutine SUBRA is being placed in the R library from a punched object deck.

*// LOAD Statement:* In this sample procedure, $MAINT is the routine which interrogates the // COPY statement and calls the proper routine to accomplish the desired results.

F1 is the disk pack upon which the utility program resides.

*// COPY Statement:* The FROM parameter names the device holding the subroutine to be entered. The READER option must be used to copy the assembler punched object program.

IBM System/3 Basic Assembler Coding Form

```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-READER,LIBRARY-R,NAME-SUBRA,TO-R1,RETAIN-P
     {
- Object Deck
     {
// CEND
// END
/&
```

The LIBRARY parameter, R, specifies a relocatable library. The NAME parameter gives the name of the subroutine to be entered. This name must be the same as the program name (that is the name on the START instruction). The following names are restricted and cannot be used in this parameter:

- ALL
- DIR
- SYSTEM

The TO parameter specifies the physical destination of the object program (in this case, R1).

The RETAIN parameter specifies the ultimate disposition of the object program.

*// CEND (Copy End) Statement:* The // CEND statement must follow the object deck.

*// END:* The // END statement must be the end of all library maintenance decks.

#### *Placing an Object Program Directly in the R Library*

When the object program is placed directly in the R library, it has the following characteristics in the library.

- *Name* of the object program is the module name specified in the START instruction or the default module name. See the MODULE NAME MISSING diagnostic in *Appendix C. System/3 Assembler – Source Language Error Codes and Diagnostics.*

- *Retain* entry in the library is temporary if OBJ or OBJ(T) is specified and permanent if OBJ(P) is specified.

- *Library* to receive the object program is the disk specified in the OBJECT operand of the // COMPILE statement. The default disk is the program disk.

#### Using Assembler Object Program with the Program Loader

The user may have the need to load a user-written assembler object program as a stand-alone program. To use an assembler object program in this manner it is necessary to have the program punched into an object deck on the system punch device. The assembler language user obtains an absolute loader by specifying DECK and NOREL on the OPTIONS card (see NOREL option under *OPTIONS Statement*). The 96-column loader contains six cards and the 80-column loader contains one card.

It is the user's responsibility to ensure:

1. That he has not referenced any address greater than the storage capacity of the System/3 on which the program is to be executed.

2. That the address specified on the START instruction statement is greater than X'FF'. (The START assembler statement must specify the address at which the program is to be loaded.)

3. That the END statement indicates the start-of-control address.

*Note:* If absolute object decks for more than one assembly are to be loaded together, then the loader must be removed from the front of the second and all subsequent decks, and the END card must be removed from the back of all decks except the last.

*IBM 5424 MFCU*

The procedure for loading and executing an assembler object program on the IBM 5424 MFCU is as follows:

1. Clear MFCU.

2. Place assembler object deck, including the loader, in primary hopper.

3. Press MFCU START.

4. Ready the printer.

5. Set IPL SELECTOR to MFCU for Model 10 Disk System or ALT for Models 12 and 15.

6. Press console PROGRAM LOAD to load and execute the assembler object program. (L1 or L2 halt is issued for error or not ready conditions on the MFCU.)

*IBM 2560 MFCM (Model 15 only)*

The procedure for loading and executing an assembler object program on the IBM 2560 MFCM is as follows:

1. Clear MFCM.

2. Place assembler object deck, including the loader, in primary hopper.

3. Press MFCM START.

4. Ready the printer.

5. Set IPL SELECTOR to ALT.

6. Press console PROGRAM LOAD to load and execute the assembler object program. (L1 halt is issued for error or not ready conditions on the MFCM.)

*IBM 1442 Card Read Punch (Models 12 and 15)*

The procedure for loading and executing an assembler object program on the IBM 1442 Card Read Punch is as follows:

1. Clear 1442.

2. Place assembler object deck, including the loader, in hopper.

3. Press 1442 START.

4. Ready the printer.

5. Set IPL SELECTOR to ALT.

6. Press console PROGRAM LOAD to load and execute the assembler object program. (L1 halt is issued for error or not ready conditions on the 1442.)

## ASSEMBLER LISTING

An important part of the assembler's output is the assembler listing. The assembler's printed output is on the system printer (under control of the // PRINTER OCL statement for Models 12 and 15).

The listing is a printed reproduction of the source program and the corresponding object code generated for it together with other important information. Figure 26 at the back of this section is a sample listing. Specifically, the listing consists of the following:

### Control Statements

Any OPTIONS or HEADERS statements specified by the user are printed and specification errors are noted. A list of OPTIONS in effect during the assembly is then printed. The page is ejected before the control statement information is listed.

**External Symbol List (ESL)**

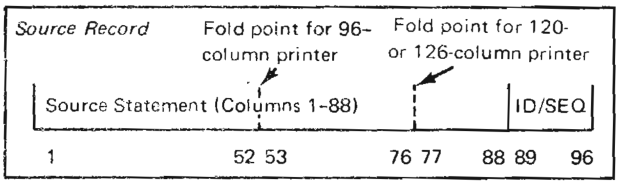
The object program name, EXTRNs, and ENTRYs will appear in the following format:

| *Symbol* | *Type* |
|---|---|
| Program name | MODULE |
| ENTRY symbol | ENTRY |
| EXTRN symbol | EXTRN |

**Source and Object Listing**

The source and object listing consists of the following:

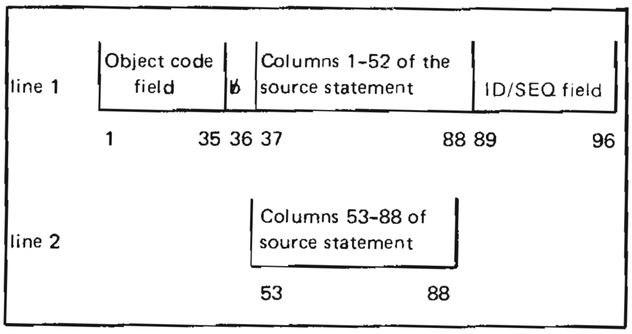
- Error code for improperly coded statements (see *Diagnostics* in this section).
- Location counter value, in hexadecimal, of the high order address of the object code generated by the corresponding source statement.
- The object code, in hexadecimal, generated by the corresponding statement.
- The value, in hexadecimal, of the expression in the operand field of the EQU, USING, DROP, and END statements, the storage address, in hexadecimal, of the low order address of the DC constants, and DS storage areas.
- Statement number, in decimal, for each statement, including comment statements. These numbers are assigned by the assembler. The statement number is a four-digit field which limits the assembly to 9,999 statements.
- The source image, which is formatted according to the size of the printer used:



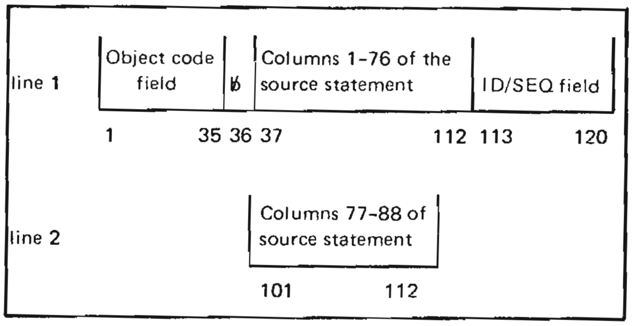

The following examples assume the ID/SEQ field is in columns 89-96 of the source record:

*Note:* The ID/SEQ field may be from one to eight adjacent characters long and may reside anywhere between columns 73-96.

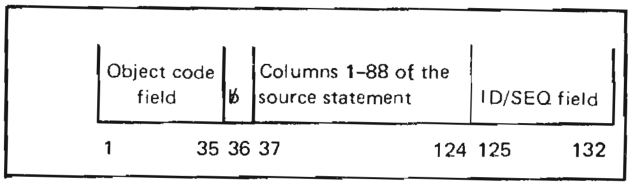
1. On a 96-column printer, the ID/SEQ field is left-justified in columns 89-96 of the print line. If columns 53-88 of the source statement are blank, line 2 will not be printed.




2. On a 120-column or 126-column printer, the ID/SEQ field is left-justified in columns 113-120 of the print line. If columns 77-88 of the source statement are blank, or if the start of the ID/SEQ field on the source record is less than column 77, line 2 will not be printed.




3. With the 132-column printer, the complete source image is printed on one line.




*Note:* Statements generated by the macro processor contain a plus symbol (+) in column 36.

*Diagnostics*

The source and object program listing includes error codes for improperly coded statements. These errors are listed again, with a message, at the end of the source and object program listing under the heading DIAGNOSTICS. This list provides the following information:

- Statement–The statement number, in decimal, (assigned by the assembler) of the statement which is in error.
- Error code–a 3-digit alphameric code. See *Appendix C: System/3 Assembler–Source Language Error Codes and Diagnostics* for a list of error codes and translations.
- Message–A translation of the error code indicating the type of error made.

Also included under DIAGNOSTICS are the following error summary statements:

- A count of the total statements in error in the assembly.
- A count of total sequence errors in the assembly if sequence check is requested.

**Cross-Reference List**

If XREF is specified on the OPTIONS statement this list includes all symbol names referred to in the source program. The following columns are included:

- Symbol–The symbol name.
- Length–The decimal length attribute of the symbol in bytes.
- Values–Value, in hexadecimal, of the symbol.
- Defined–Statement number, in decimal, where the symbol is defined.
- References–Statement numbers, in decimal, where the symbol is referenced. Symbolic references to data areas and machine registers whose contents may be altered by execution of a machine instruction are flagged with an asterisk.

At the end of the cross-reference list, the error summary statements are printed again.

```
SUBRC                                    EXTERNAL SYMBOL LIST

SYMBOL     TYPE                                                          VER 00, MOD 00 01/30/76 PAGE 1

SUBRC      MODULE
```

```
SUBRC   SAMPLE EXIT SUBROUTINE--FIELD AND INDICATOR

ERR LOC   OBJECT CODE      ADDR STMT SOURCE STATEMENT                    VER 00, MOD 00 01/30/76 PAGE 2

                                  2 ********************************************************************
                                  3 *                                                                  *
                                  4 * NAME ............ SUBRC.                                         *
                                  5 *                                                                  *
                                  6 * FUNCTION ........ EXIT SUBROUTINE WITH FIELD AND INDICATOR       *
                                  7 *                   PARAMETERS.                                    *
                                  8 *                                                                  *
                                  9 *                   THE CODE GENERATED BY THE COMPILER IS AS FOLLOWS: *
                                 10 *                                                                  *
                                 11 *                        B      SUBRC                              *
                                 12 *                        DC     IL1'FIELD LENGTH-1'                *
                                 13 *                        DC     AL2'ADDRESS OF RIGHT OF FIELD'     *
                                 14 *                        DC     XL1'00'                            *
                                 15 *                        DC     XL1'INDICATOR MASK'                *
                                 16 *                        DC     XL1'REGISTER 1 DISPLACEMENT'       *
                                 17 *                                                                  *
                                 18 ********************************************************************
    0000                         19 SUBRC  START 0
    0000 34 08 0C13              20        ST    GET+3,ARR             SAVE PARM ADDR
    0004 36 08 0C31              21        A     CON6,ARR              INCREMENT TO RETURN
    0008 34 C8 0C2F              22        ST    RET+3,ARR             SAVE RETURN
    000C 34 02 0C29              23        ST    SAVE+3,2              SAVE XR2
    0010 C2 02 0CCC              24 GET    LA    *-*,2                 GET PARMETER ADDRESS
    0014 2C 01 0C1B 05           25        MVC   TEST+2(2),5(,2)       MOVE IN MASK AND DISPLACEMENT
    0019 78 00 0C                26 TEST   TBN   *-*(,1),*-*           TEST INDICATOR
    001C F2 90 09                27        JF    SAVE                  INDICATOR OFF
    001F B5 02 02                28        L     2(,2),2               GET CONTROL FIELD ADDRESS
    0022 B5 02 05                29        L     5(,2),2               GET LOOK UP ADDRESS
    0025 BC C3 00                30        MVI   0(,2),C'C'            MOVE IN C'C'
    0028 C2 02 0CCC              31 SAVE   LA    *-*,2                 RESTORE
    002C C0 87 0CCC              32 RET    B     *-*                   RETURN
    0030 0006               0031 33 CON6   DC    IL2'6'
                            0008 34 ARR    EQU   8
                            FFFF 35        END

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY =     0
```

```
SUBRC                                  CROSS REFERENCE

SYMBOL   LEN VALUE DEFN   REFERENCES                                    VER 00, MOD 00 01/30/76 PAGE 3

ARR      001  0008 0034   0020  0021* 0022
CON6     002  0031 0033   0021
GET      004  001C 0024   0020*
RET      004  002C 0032   0022*
SAVE     004  0028 0031   0023* 0027
SUBRC    001  000C 0019
TEST     003  0019 0026   0025*

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY =     0
```

Figure 26. Sample Assembler Listing

### External Symbol List (ESL) Table Size

The ESL table is an execution time main storage table containing the module name (START statement name or ASMOBJ) and each EXTRN and ENTRY symbol defined in an assembly. The total of EXTRNs and ENTRYs allowed in a single assembly is limited by the ESL table size.

Using the Model 10 disk system assembler, the limit is 74 EXTRNs and ENTRYs.

Using the Model 12 and Model 15 assembler, the limit varies with the amount of storage available in the execution partition. The limiting sizes and associated storage ranges are:

| *Storage Available* | *Limit of EXTRNs and ENTRYs* |
|---|---|
| 10K | 84 |
| 12K | 124 |
| 14K | 169 |
| 16K | 209 |
| 18K - 48K | 254 |

## MACHINE LANGUAGE INSTRUCTION FORMATS

### Operation Code

The first byte of each instruction, the operation code, specifies the addressing modes to be employed by the instruction in bits 0 through 3, and the operation to be performed in bits 4 through 7.

### Q Code

The second byte of each instruction is the Q code. In 2-address formats, the Q code is always a length count. In other formats, depending upon the operation specified, the Q code can be:

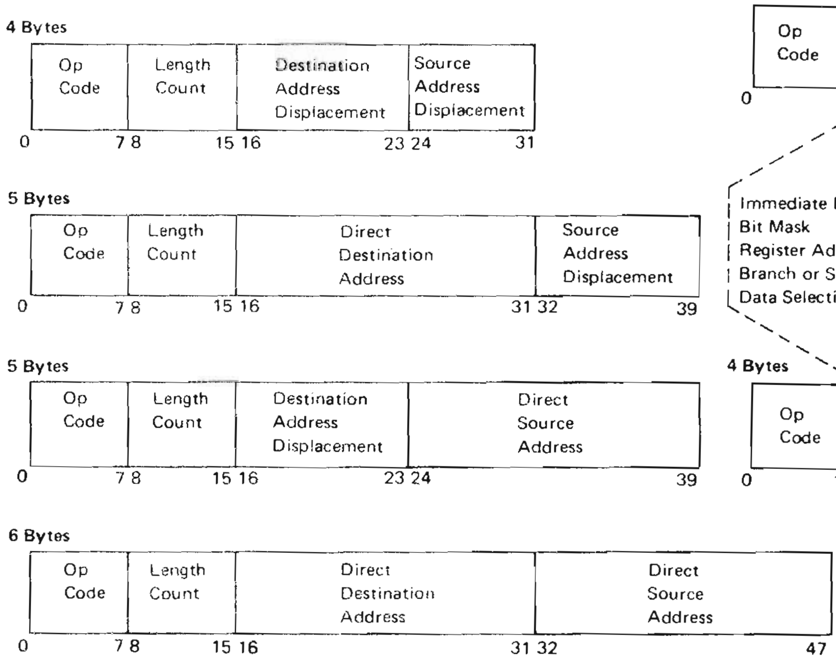
- Length count
- Immediate data
- Bit mask
- Register address
- Data selection
- Branch or skip condition
- Device address and functional specifications

### Control Code

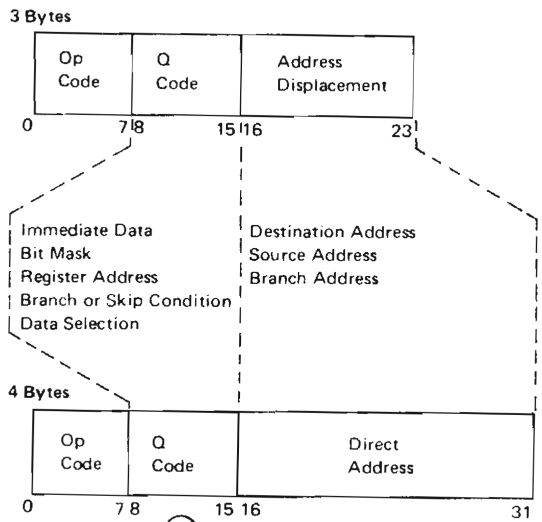
The third byte of an instruction in the Command Format contains additional data pertaining to the command to be executed.

### Storage Addresses

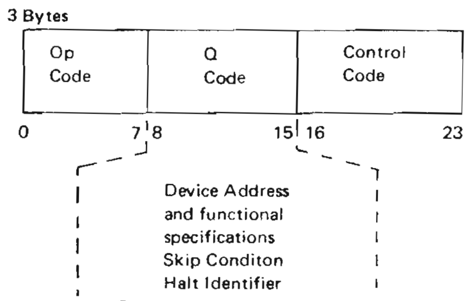
For instructions in the 1-operand and 2-operand formats, the third byte of the instruction and all bytes following are storage address information.




(A) Two-Address Formats




(B) One-Address Formats




(C) Command Format

| Op | Mnemonic | Type |
|---|---|---|
| 04 | ZAZ | |
| 06 | AZ | 2 ADDRESS |
| 07 | SZ | |
| 08 | MVX | Direct |
| 0A | ED | Op \| Q \| Operand One \| Operand Two |
| 0B | ITC | |
| 0C | MVC | 6 bytes |
| 0D | CLC | |
| 0E | ALC | |
| 0F | SLC | |
| 14 | ZAZ | |
| 16 | AZ | 2 ADDRESS |
| 17 | SZ | |
| 18 | MVX | Direct Indexed |
| 1A | ED | Op \| Q \| Operand One \| D2 |
| 1B | ITC | |
| 1C | MVC | 5 bytes |
| 1D | CLC | |
| 1E | ALC | R1 |
| 1F | SLC | |
| 24 | ZAZ | |
| 26 | AZ | 2 ADDRESS |
| 27 | SZ | |
| 28 | MVX | Direct Indexed |
| 2A | ED | Op \| Q \| Operand One \| D2 |
| 2B | ITC | |
| 2C | MVC | 5 bytes |
| 2D | CLC | |
| 2E | ALC | R2 |
| 2F | SLC | |
| 30 | SNS | |
| 31 | LIO | 1 ADDRESS |
| 34 | ST | |
| 35 | L | Direct |
| 36 | A | Op \| Q \| Operand One |
| 38 | TBN | |
| 39 | TBF | 4 bytes |
| 3A | SBN | |
| 3B | SBF | |
| 3C | MVI | |
| 3D | CLI | |
| 3E | SCP* | |
| 3F | LCP* | |
| 44 | ZAZ | |
| 46 | AZ | 2 ADDRESS |
| 47 | SZ | |
| 48 | MVX | |
| 4A | ED | Op \| Q \| D1 \| Operand Two |
| 4B | ITC | |
| 4C | MVC | 5 bytes |
| 4D | CLC | |
| 4E | ALC | R1 |
| 4F | SLC | |
| 54 | ZAZ | |
| 56 | AZ | 2 ADDRESS |
| 57 | SZ | |
| 58 | MVX | Indexed |
| 5A | ED | Op \| Q \| D1 \| D2 |
| 5B | ITC | |
| 5C | MVC | 4 bytes |
| 5D | CLC | |
| 5E | ALC | R1 R1 |
| 5F | SLC | |

| Op | Mnemonic | Type |
|---|---|---|
| 64 | ZAZ | |
| 66 | AZ | 2 ADDRESS |
| 67 | SZ | |
| 68 | MVX | |
| 6A | ED | Op \| Q \| D1 \| D2 |
| 6B | ITC | |
| 6C | MVC | 4 bytes |
| 6D | CLC | |
| 6E | ALC | R1 R2 |
| 6F | SLC | |
| 70 | SNS | |
| 71 | LIO | 1 ADDRESS |
| 74 | ST | |
| 75 | L | |
| 76 | A | Op \| Q \| D1 |
| 78 | TBN | |
| 79 | TBF | 3 bytes |
| 7A | SBN | |
| 7B | SBF | |
| 7C | MVI | R1 |
| 7D | CLI | |
| 7E | SCP* | |
| 7F | LCP* | |
| 84 | ZAZ | |
| 86 | AZ | 2 ADDRESS |
| 87 | SZ | |
| 88 | MVX | Indexed Direct |
| 8A | ED | Op \| Q \| D1 \| Operand Two |
| 8B | ITC | |
| 8C | MVC | 5 bytes |
| 8D | CLC | |
| 8E | ALC | R2 |
| 8F | SLC | |
| 94 | ZAZ | |
| 96 | AZ | 2 ADDRESS |
| 97 | SZ | |
| 98 | MVX | Indexed |
| 9A | ED | Op \| Q \| D1 \| D2 |
| 9B | ITC | |
| 9C | MVC | 4 bytes |
| 9D | CLC | |
| 9E | ALC | R2 R1 |
| 9F | SLC | |
| A4 | ZAZ | |
| A6 | AZ | 2 ADDRESS |
| A7 | SZ | |
| A8 | MVX | Indexed |
| AA | ED | Op \| Q \| D1 \| D2 |
| AB | ITC | |
| AC | MVC | 4 bytes |
| AD | CLC | |
| AE | ALC | R2 R2 |
| AF | SLC | |

* Model 15 only.

*Legend:*

D1 – Displacement, operand 1
D2 – Displacement, operand 2
R1 – Register 1
R2 – Register 2

| Op | Mnemonic | Type | |
|---|---|---|---|
| B0 | SNS | | |
| B1 | LIO | 1 ADDRESS | |
| B4 | ST | | |
| B5 | L | Indexed | |
| B6 | A | Op Q D1 | |
| B8 | TBN | | |
| B9 | TDF | ← 3 bytes → | |
| BA | SBN | | |
| BB | SBF | | |
| BC | MVI | XR2 | |
| BD | CLI | | |
| BE | SCP* | | |
| BF | LCP* | | |
| C0 | BC | Direct | |
| C1 | TIO | Op Q Address | |
| C2 | LA | ← 4 bytes → | |
| D0 | BC | | |
| D1 | TIO | Op Q D2 | +XR1 |
| D2 | LA | ← 3 bytes → | |
| E0 | BC | | |
| E1 | TIO | Op Q D2 | +XR2 |
| E2 | LA | ← 3 bytes → | |
| F0 | HPL | | |
| F1 | APL | | |
| F2 | JC | Op Q R | |
| F3 | SIO | ← 3 bytes → | |
| F4 | CCP* | | |

*Model 15 only.

| Bits 0-3 | Op Code (one byte), Bits 4-7: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | Q Code One Byte | Operands: First | Operands: Second | Total Instr Length | Type | Summary: Op | Q | Operand | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 2 Bytes Direct | 2 Bytes Direct | 6 | X | | | | | |
| 1 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 2 Bytes Direct | 1 Byte Disp Index-By R1 | 5 | X | | | | | D1 |
| 2 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 2 Bytes Direct | 1 Byte Disp Index-By R2 | 5 | X | | | | | D2 |
| 3 | SNS | LIO | | | ST | L | A | | TBN | TBF | SBN | SBF | MVI | CLI | SCP* | LCP* | | 2 Bytes Direct | | 4 | Y | | | | | |
| 4 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R1 | 2 Bytes Direct | 5 | X | | | D1 | | |
| 5 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R1 | 1 Byte Disp Index-By R1 | 4 | X | | | D1 | D1 | |
| 6 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R1 | 1 Byte Disp Index-By R2 | 4 | X | | | D1 | D2 | |
| 7 | SNS | LIO | | | ST | L | A | | TBN | TBF | SBN | SBF | MVI | CLI | SCP* | LCP* | | 1 Byte Displacement Indexed By R1 | | 3 | Y | | | D1 | | |
| 8 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R2 | 2 Bytes Direct | 5 | X | | | D2 | | |
| 9 | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R2 | 1 Byte Disp Index-By R1 | 4 | X | | | D2 | D1 | |
| A | | | | | ZAZ | | AZ | SZ | MVX | | ED | ITC | MVC | CLC | ALC | SLC | | 1 Byte Displacement Indexed By R2 | 1 Byte Disp Index-By R2 | 4 | X | | | D2 | D2 | |
| B | SNS | LIO | | | ST | L | A | | TBN | TBF | SBN | SBF | MVI | CLI | SCP* | LCP* | | 1 Byte Displacement Indexed By R2 | | 3 | Y | | | D2 | | |
| C | BC | TIO | LA | | | | | | | | | | | | | | | | 2 Bytes Direct | 4 | Z | | | | | |
| D | BC | TIO | LA | | | | | | | | | | | | | | | | 1 Byte Disp Index-By R1 | 3 | Z | | | D1 | | |
| E | BC | TIO | LA | | | | | | | | | | | | | | | | 1 Byte Disp Index-By R2 | 3 | Z | | | D2 | | |
| F | HPL | APL | JC | SIO | CCP* | | | | | | | | | | | | | | | 3 | F | | | | | |

*Model 15 only.

## MNEMONIC OPERATION CODES (MACHINE)

| *Instruction** | *Mnemonic Operation Code* | |
|---|---|---|
| Zero and Add Zoned Decimal | ZAZ | Two-address Format** |
| Add Zoned Decimal | AZ | |
| Subtract Zoned Decimal | SZ | |
| Move Hex Character | MVX | |
| Move Characters | MVC | |
| Compare Logical Characters | CLC | |
| Add Logical Characters | ALC | |
| Subtract Logical Characters | SLC | |
| Insert and Test Characters | ITC | |
| Edit | ED | |
| Move Logical Immediate | MVI | One-address Format** |
| Compare Logical Immediate | CLI | |
| Set Bits On Masked | SBN | |
| Set Bits Off Masked | SBF | |
| Test Bits On Masked | TBN | |
| Test Bits Off Masked | TBF | |
| Store Register | ST | |
| Load Register | L | |
| Add to Register | A | |
| Branch On Condition | BC | |
| Test I/O and Branch | TIO | |
| Sense I/O | SNS | |
| Load I/O | LIO | |
| Load Address | LA | |
| Load CPU*** | LCP | |
| Store CPU*** | SCP | |
| Advance Program Level | APL | Command Format** |
| Halt Program Level | HPL | |
| Start I/O | SIO | |
| Command CPU*** | CCP | |
| Jump On Condition | JC | |

* For information concerning specifications for the use of these instructions with the Model 10, see the *IBM System/3 Model 10 Components Reference Manual,* GA21-9103, or with the Model 15, see the *IBM System/3 Model 15 Components Reference Manual,* GA21-9193.

** See *Machine Language Instruction Formats* in this appendix.

*** These instructions are for the Model 15 but they can also be generated on the Model 12 through the macros $LCP, $SCP, and $CCP. For more information concerning the use of the Model 12 macros, see *IBM System/3 Models 10 and 12 System Control Programming Macros Reference Manual*, GC21-7562.

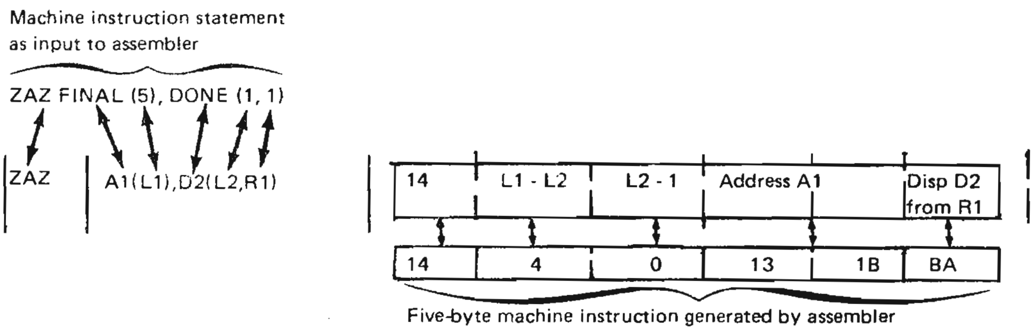
## EXTENDED MNEMONIC CODES

| *Instruction* | *Mnemonic Operation Code* | *Q Code* |
|---|---|---|
| Move Hex Character (MVX) | | |
| Move to Zone from Zone | MZZ | X'00' |
| Move to Numeric from Zone | MNZ | X'02' |
| Move to Zone from Numeric | MZN | X'01' |
| Move to Numeric from Numeric | MNN | X'03' |
| Branch On Condition (BC) | | |
| Branch | B | X'87' |
| Branch High | BH | X'84' |
| Branch Low | BL | X'82' |
| Branch Equal | BE | X'81' |
| Branch Not High | BNH | X'04' |
| Branch Not Low | BNL | X'02' |
| Branch Not Equal | BNE | X'01' |
| Branch Overflow Zoned | BOZ | X'88' |
| Branch Overflow Logical | BOL | X'A0' |
| Branch No Overflow Zoned | BNOZ | X'08' |
| Branch No Overflow Logical | BNOL | X'20' |
| Branch True | BT | X'10' |
| Branch False | BF | X'90' |
| Branch Plus | BP | X'84' |
| Branch Minus | BM | X'82' |
| Branch Zero | BZ | X'81' |
| Branch Not Plus | BNP | X'04' |
| Branch Not Minus | BNM | X'02' |
| Branch Not Zero | BNZ | X'01' |
| Jump On Condition (JC) | | |
| Jump | J | X'87' |
| Jump High | JH | X'84' |
| Jump Low | JL | X'82' |
| Jump Equal | JE | X'81' |
| Jump Not High | JNH | X'04' |
| Jump Not Low | JNL | X'02' |
| Jump Not Equal | JNE | X'01' |
| Jump Overflow Zoned | JOZ | X'88' |
| Jump Overflow Logical | JOL | X'A0' |
| Jump No Overflow Zoned | JNOZ | X'08' |
| Jump No Overflow Logical | JNOL | X'20' |
| Jump True | JT | X'10' |
| Jump False | JF | X'90' |
| Jump Plus | JP | X'84' |
| Jump Minus | JM | X'82' |
| Jump Zero | JZ | X'81' |
| Jump Not Plus | JNP | X'04' |
| Jump Not Minus | JNM | X'02' |
| Jump Not Zero | JNZ | X'01' |
| Command CPU (CCP–Model 15 only) | | |
| Supervisor Call | SVC | X'10' |

*Assembler Language to Machine Language Relationships*

The following charts show the relationship between a machine instruction statement as coded by the System/3 Basic Assembler Language programmer and the machine language as generated by the assembler.

For example, the instruction coded by the programmer is ZAZ FINAL(5),DONE(1,1). From the second line of the first of the charts we can develop the relationship between the instruction and the machine code as follows (assume FINAL is a relocatable symbol with value X'131B' and DONE is an absolute symbol with value X'BA'):




Used in this manner, the following charts show what machine code results from a particular assembler language statement, and vice versa, what assembler language format obtains a particular machine code format.

The abbreviations used on the following pages mean:

| | |
|---|---|
| A1 | Direct address, operand 1 |
| A2 | Direct address, operand 2 |
| D1 | Displacement, operand 1 |
| D2 | Displacement, operand 2 |
| L1 | Length of operand 1 |
| L2 | Length of operand 2 |
| R1 | Register 1 |
| R2 | Register 2 |
| RX | Local storage register |
| I | Immediate data |

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| ZAZ | A1(L1),A2(L2) | 04 | L1-L2 L2-1 | Address A1 | | Address A2 | |
| ZAZ | A1(L1),D2(L2,R1) | 14 | L1-L2 L2-1 | Address A1 | | Disp D2 from R1 | |
| ZAZ | A1(L1),D2(L2,R2) | 24 | L1-L2 L2-1 | Address A1 | | Disp D2 from R2 | |
| ZAZ | D1(L1,R1),A2(L2) | 44 | L1-L2 L2-1 | Disp D1 from R1 | Address A2 | | |
| ZAZ | D1(L1,R1),D2(L2,R1) | 54 | L1-L2 L2-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| ZAZ | D1(L1,R1),D2(L2,R2) | 64 | L1-L2 L2-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| ZAZ | D1(L1,R2),A2(L2) | 84 | L1-L2 L2-1 | Disp D1 from R2 | Address A2 | | |
| ZAZ | D1(L1,R2),D2(L2,R1) | 94 | L1-L2 L2-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| ZAZ | D1(L1,R2),D2(L2,R2) | A4 | L1-L2 L2-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

If L1 or L2 is not specified, the implied length is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| AZ | A1(L1),A2(L2) | 06 | L1-L2 \| L2-1 | Address A1 | | Address A2 | |
| AZ | A1(L1),D2(L2,R1) | 16 | L1-L2 \| L2-1 | Address A1 | | Disp D2 from R1 | |
| AZ | A1(L1),D2(L2,R2) | 26 | L1-L2 \| L2-1 | Address A1 | | Disp D2 from R2 | |
| AZ | D1(L1,R1),A2(L2) | 46 | L1-L2 \| L2-1 | Disp D1 from R1 | Address A2 | | |
| AZ | D1(L1,R1),D2(L2,R1) | 56 | L1-L2 \| L2-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| AZ | D1(L1,R1),D2(L2,R2) | 66 | L1-L2 \| L2-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| AZ | D1(L1,R2),A2(L2) | 86 | L1-L2 \| L2-1 | Disp D1 from R2 | Address A2 | | |
| AZ | D1(L1, R2), D2(L2, R1) | 96 | L1-L2 \| L2-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| AZ | D1(L1,R2),D2(L2,R2) | A6 | L1-L2 \| L2-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

If L1 or L2 is not specified, the implied length is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| SZ | A1(L1),A2(L2) | 07 | L1-L2 L2-1 | Address A1 | | Address A2 | |
| SZ | A1(L1),D2(L2,R1) | 17 | L1-L2 L2-1 | Address A1 | | Disp D2 from R1 | |
| SZ | A1(L1), D2(L2,R2) | 27 | L1-L2 L2-1 | Address A1 | | Disp D2 from R2 | |
| SZ | D1(L1,R1),A2(L2) | 47 | L1-L2 L2-1 | Disp D1 from R1 | Address A2 | | |
| SZ | D1(L1,R1),D2(L2,R1) | 57 | L1-L2 L2-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| SZ | D1(L1,R1),D2(L2,R2) | 67 | L1-L2 L2-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| SZ | D1(L1,R2),A2(L2) | 87 | L1-L2 L2-1 | Disp D1 from R2 | Address A2 | | |
| SZ | D1(L1,R2),D2(L2,R1) | 97 | L1-L2 L2-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| SZ | D1(L1,R2),D2(L2,R2) | A7 | L1-L2 L2-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

If L1 or L2 is not specified, the implied length is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| MVX | A1(I),A2 | 08 | I | Address A1 | | Address A2 | |
| MVX | A1(I),D2(,R1) | 18 | I | Address A1 | | Disp D2 from R1 | |
| MVX | A1(I),D2(,R2) | 28 | I | Address A1 | | Disp D2 from R2 | |
| MVX | D1(I,R1),A2 | 48 | I | Disp D1 from R1 | Address A2 | | |
| MVX | D1(I,R1),D2(,R1) | 58 | I | Disp D1 from R1 | Disp D2 from R1 | | |
| MVX | D1(I,R1),D2(,R2) | 68 | I | Disp D1 from R1 | Disp D2 from R2 | | |
| MVX | D1(I,R2),A2 | 88 | I | Disp D1 from R2 | Address A2 | | |
| MVX | D1(I,R2),D2(,R1) | 98 | I | Disp D1 from R2 | Disp D2 from R1 | | |
| MVX | D1(I,R2),D2(,R2) | A8 | I | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

I may be specified on either operand, and must have the value X'00',X'01',X'02', or X'03'.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

For the extended mnemonics of the MVX instruction, I-field information is inherent in the mnemonic and the I-field is omitted from the operand field. See *Extended Mnemonic Codes* for the extended MVX and the associated Q-codes.

| Assembler Instruction Format | | Machine Instruction Format | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| MVC | A1(L1),A2 | 0C | L1-1 | Address A1 | | Address A2 | |
| MVC | A1(L1),D2(,R1) | 1C | L1-1 | Address A1 | | Disp D2 from R1 | |
| MVC | A1(L1),D2(,R2) | 2C | L1-1 | Address A1 | | Disp D2 from R2 | |
| MVC | D1(L1,R1),A2 | 4C | L1-1 | Disp D1 from R1 | Address A2 | | |
| MVC | D1(L1,R1),D2(,R1) | 5C | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| MVC | D1(L1,R1),D2(,R2) | 6C | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| MVC | D1(L1,R2),A2 | 8C | L1-1 | Disp D1 from R2 | Address A2 | | |
| MVC | D1(L1,R2),D2(,R1) | 9C | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| MVC | D1(L1,R2),D2(,R2) | AC | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| CLC | A1(L1),A2 | 0D | L1-1 | Address A1 | | Address A2 | |
| CLC | A1(L1),D2(,R1) | 1D | L1-1 | Address A1 | | Disp D2 from R1 | |
| CLC | A1(L1),D2(,R2) | 2D | L1-1 | Address A1 | | Disp D2 from R2 | |
| CLC | D1(L1,R1),A2 | 4D | L1-1 | Disp D1 from R1 | Address A2 | | |
| CLC | D1(L1,R1),D2(,R1) | 5D | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| CLC | D1(L1,R1),D2(,R2) | 6D | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| CLC | D1(L1,R2),A2 | 8D | L1-1 | Disp D1 from R2 | Address A2 | | |
| CLC | D1(L1,R2),D2(,R1) | 9D | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| CLC | D1(L1,R2),D2(,R2) | AD | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| ALC | A1(L1),A2 | 0E | L1-1 | Address A1 | | Address A2 | |
| ALC | A1(L1),D2(,R1) | 1E | L1-1 | Address A1 | | Disp D2 from R1 | |
| ALC | A1(L1),D2(,R2) | 2E | L1-1 | Address A1 | | Disp D2 from R2 | |
| ALC | D1(L1,R1),A2 | 4E | L1-1 | Disp D1 from R1 | Address A2 | | |
| ALC | D1(L1,R1),D2(,R1) | 5E | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| ALC | D1(L1,R1),D2(,R2) | 6E | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| ALC | D1(L1,R2),A2 | 8E | L1-1 | Disp D1 from R2 | Address A2 | | |
| ALC | D1(L1,R2),D2(,R1) | 9E | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| ALC | D1(L1,R2),D2(,R2) | AE | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SLC | A1(L1),A2 | 0F | L1-1 | Address A1 | | Address A2 | |
| SLC | A1(L1),D2(,R1) | 1F | L1-1 | Address A1 | | Disp D2 from R1 | |
| SLC | A1(L1),D2(,R2) | 2F | L1-1 | Address A1 | | Disp D2 from R2 | |
| SLC | D1(L1,R1),A2 | 4F | L1-1 | Disp D1 from R1 | Address A2 | | |
| SLC | D1(L1,R1),D2(,R1) | 5F | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| SLC | D1(L1,R1),D2(,R2) | 6F | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| SLC | D1(L1,R2),A2 | 8F | L1-1 | Disp D1 from R2 | Address A2 | | |
| SLC | D1(L1,R2),D2(,R1) | 9F | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| SLC | D1(L1,R2),D2(,R2) | AF | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| ITC | A1(L1),A2 | 0B | L1-1 | Address A1 | | Address A2 | |
| ITC | A1(L1),D2(,R1) | 1B | L1-1 | Address A1 | | Disp D2 from R1 | |
| ITC | A1(L1),D2(,R2) | 2B | L1-1 | Address A1 | | Disp D2 from R2 | |
| ITC | D1(L1,R1),A2 | 4B | L1-1 | Disp D1 from R1 | Address A2 | | |
| ITC | D1(L1,R1),D2(,R1) | 5B | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| ITC | D1(L1,R1),D2(,R2) | 6B | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| ITC | D1(L1,R2),A2 | 8B | L1-1 | Disp D1 from R2 | Address A2 | | |
| ITC | D1(L1,R2),D2(,R1) | 9B | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| ITC | D1(L1,R2),D2(,R2) | AB | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

Operand one must address the data field at the leftmost byte.

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| ED | A1(L1),A2 | 0A | L1-1 | Address A1 | | Address A2 | |
| ED | A1(L1),D2(,R1) | 1A | L1-1 | Address A1 | | Disp D2 from R1 | |
| ED | A1(L1),D2(,R2) | 2A | L1-1 | Address A1 | | Disp D2 from R2 | |
| ED | D1(L1,R1),A2 | 4A | L1-1 | Disp D1 from R1 | Address A2 | | |
| ED | D1(L1,R1),D2(,R1) | 5A | L1-1 | Disp D1 from R1 | Disp D2 from R1 | | |
| ED | D1(L1,R1),D2(,R2) | 6A | L1-1 | Disp D1 from R1 | Disp D2 from R2 | | |
| ED | D1(L1,R2),A2 | 8A | L1-1 | Disp D1 from R2 | Address A2 | | |
| ED | D1(L1,R2),D2(,R1) | 9A | L1-1 | Disp D1 from R2 | Disp D2 from R1 | | |
| ED | D1(L1,R2),D2(,R2) | AA | L1-1 | Disp D1 from R2 | Disp D2 from R2 | | |

NOTES:

L1 may be specified on either operand; if L1 is not specified, the implied length of operand one is used.

If D1 or D2 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| MVI | A1,I | 3C | I | Address A1 | | | |
| MVI | D1(,R1),I | 7C | I | Disp D1 from R1 | | | |
| MVI | D1(,R2),I | BC | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| CLI | A1,I | 3D | I | Address A1 | | | |
| CLI | D1(,R1),I | 7D | I | Disp D1 from R1 | | | |
| CLI | D1(,R2),I | BD | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SBN | A1,I | 3A | I | Address A1 | | | |
| SBN | D1(,R1),I | 7A | I | Disp D1 from R1 | | | |
| SBN | D1(,R2),I | BA | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SBF | A1,I | 38 | I | Address A1 | | | |
| SBF | D1(,R1),I | 78 | I | Disp D1 from R1 | | | |
| SBF | D1(,R2),I | B8 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| TBN | A1,I | 38 | I | Address A1 | | | |
| TBN | D1(,R1),I | 78 | I | Disp D1 from R1 | | | |
| TBN | D1(,R2),I | B8 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| TBF | A1,I | 39 | I | Address A1 | | | |
| TBF | D1(,R1),I | 79 | I | Disp D1 from R1 | | | |
| TBF | D1(,R2),I | B9 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| ST | A1,RX | 34 | RX | Address A1 | | | |
| ST | D1(,R1),RX | 74 | RX | Disp D1 from R1 | | | |
| ST | D1(,R2),RX | B4 | RX | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| L | A1,RX | 35 | RX | Address A1 | | | |
| L | D1(,R1),RX | 75 | RX | Disp D1 from R1 | | | |
| L | D1(,R2),RX | B5 | RX | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| A | A1,RX | 36 | RX | Address A1 | | | |
| A | D1(,R1),RX | 76 | RX | Disp D1 from R1 | | | |
| A | D1(,R2),RX | B6 | RX | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| BC | A1,I | C0 | I | Address A1 | | | |
| BC | D1(,R1),I | D0 | I | Disp D1 from R1 | | | |
| BC | D1(,R2),I | E0 | I | Disp D1 from R2 | | | |

NOTES:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

For the extended mnemonics of the BC, the second operand (I-field) is not used since the information is inherent in the mnemonic. See *Extended Mnemonic Codes* for the extended branches and their associated Q-codes.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| TIO | A1,I | C1 | I | Address A1 | | | |
| TIO | D1(,R1),I | D1 | I | Disp D1 from R1 | | | |
| TIO | D1(,R2),I | E1 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SNS | A1,I | 30 | I | Address A1 | | | |
| SNS | D1(,R1),I | 70 | I | Disp D1 from R1 | | | |
| SNS | D1(,R2),I | B0 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| LIO | A1,I | 31 | I | Address A1 | | | |
| LIO | D1(,R1),I | 71 | I | Disp D1 from R1 | | | |
| LIO | D1(,R2),I | B1 | I | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| LA | A1,RX | C2 | RX | Address A1 | | | |
| LA | D1(,R1),RX | D2 | RX | Disp D1 from R1 | | | |
| LA | D1(,R2),RX | E2 | RX | Disp D1 from R2 | | | |

NOTE:

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| LCP | A1,RX | 3F | RX | Address A1 | | | |
| LCP | D1(,R1),RX | 7F | RX | Disp D1 from R1 | | | |
| LCP | D1(,R2),RX | BF | RX | Disp D1 from R2 | | | |

NOTES:

The Model 15 LCP instruction can also be generated on the Model 12 through the $LCP macro instruction; see *IBM System/3 Models 10 and 12 System Control Programming Macros Reference Manual*, GC21-7562.

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SCP | A1,RX | 3E | RX | Address A1 | | | |
| SCP | D1(,R1),RX | 7E | RX | Disp D1 from R1 | | | |
| SCP | D1(,R2),RX | BE | RX | Disp D1 from R2 | | | |

NOTES:

The Model 15 SCP instruction can also be generated on the Model 12 through the $SCP macro instruction; see *IBM System/3 Models 10 and 12 System Control Programming Macros Reference Manual*, GC21-7562.

If D1 is relocatable, the assembler computes the displacement based on the USING instruction.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| APL | I | F1 | I | 00 | | | |

NOTE:

The APL is a NO-OP instruction on the Model 15.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| HPL | I1,I2 | F0 | I2 | I1 | | | |

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| Operation | Operands | Op-Code | Q-Code | Operands | | | |
| | | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 |
| SIO | I1,I2 | F3 | I2 | I1 | | | |

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| CCP | I1,RX | F4 | RX | I1 | | | |

NOTES:

The Model 15 CCP instruction can also be generated on the Model 12 through the $CCP macro instruction; see *IBM System/3 Models 10 and 12 System Control Programming Macros Reference Manual*, GC21-7562.

For the SVC form of the CCP instruction, the Q-code is inherent in the mnemonic and the RX field is omitted from the operand field. See *Extended Mnemonic Codes* for the associated Q-code.

| Assembler Instruction Format | | Machine Instruction Format | | | | | |
|---|---|---|---|---|---|---|---|
| **Operation** | **Operands** | **Op-Code** | **Q-Code** | **Operands** | | | |
| | | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** |
| JC | A1,I | F2 | I | * | | | |

*If the first operand is absolute, this value is placed in byte 3.
If the first operand is relocatable, the displacement from the next sequential instruction to address A1 is placed in byte 3.

NOTE:

For the extended mnemonics of the JC, the second operand (I-field) is not used since the information is inherent in the mnemonic. See *Extended Mnemonic Codes* for the extended jumps and their associated Q-codes.

## Appendix B: Assembler Instruction Reference Table

| Operation Entry | Name Entry | Operand Entry |
|---|---|---|
| DC | Any Symbol or Blank | One operand entry containing: Duplication Factor, Type, Length, Constant. |
| DROP | Blank | Specified register (1 or 2). |
| DS | Any Symbol or Blank | One operand entry containing: Duplication Factor, Type, Length. |
| EJECT | Blank | Blank. |
| END | Blank | A relocatable expression or blank. |
| ENTRY | Blank | Any relocatable name entry found in the current program. |
| EQU | Any Symbol | An expression. |
| EXTRN | Blank | One relocatable symbol not found in the current program which may be followed by an absolute expression enclosed in parentheses. |
| ICTL | Blank | Two decimals in the form of B,E. |
| ISEQ | Blank | Blank or two decimal values in the form L, R. |
| ORG | Blank | Blank operand or an expression (A) optionally followed by two absolute expressions in the form A,B,C. |
| PRINT | Blank | Model 10 Disk System: One or two entries from DATA, NODATA, ON, OFF.<br>Model 12 and Model 15: One to three entries from DATA, NODATA, GEN, NOGEN, ON, OFF. |
| SPACE | Blank | Blank or a decimal value. |
| START | Name or Blank | A self-defining value or blank. |
| TITLE | Name or Blank | A sequence of characters enclosed in apostrophes. |
| USING | Blank | A relocatable expression (V) and an index register (R) in the form V,R. |

## Appendix C: System/3 Assembler — Source Language Error Codes and Diagnostics

| Code | Diagnostic | Explanation |
|---|---|---|
| N01 | INVALID NAME LENGTH | Name field entry greater than six characters |
| N02 | INVALID CHARACTER IN NAME | Name starts with non-alphabetic or contains an invalid character |
| N03 | NAME NOT ALLOWED ON THIS INSTRUCTION | Name field entry not allowed on this instruction |
| N04 | REFERENCE TO UNDEFINED SYMBOL | The referenced symbol is not defined in this program |
| N05 | NAME MISSING FROM INSTRUCTION REQUIRING ONE | Name field entry missing from EQU instruction |
| N06 | PREVIOUSLY DEFINED SYMBOL | Symbol has been previously defined in this program |
| N07 | MODULE NAME MISSING | START instruction missing, or START instruction present but name field entry (module name) missing. Assembler assigns the default module name ASMOBJ. |
| O01 | INVALID OPERATION CODE | Undefined operation field entry |
| O02 | INVALID ORIGIN | Attempt to ORG to a value less than the initial value of the location counter |
| O03 | INVALID OR ILLEGAL ICTL | Operand error on ICTL, or ICTL not the first statement in the program. (ICTL treated as last source statement in program) |
| O04 | INVALID START INSTRUCTION | START instruction encountered after location counter is initialized |
| O05 | LOCATION COUNTER ERROR | Location counter overflow (greater than 65536) or attempt to reference the location counter at 65536 |
| O06 | MISSING END STATEMENT | END statement missing from the program |
| P01 | INVALID OPERAND DELIMITER | An operand field syntactical delimiter is either misplaced or missing |
| P02 | INVALID OPERAND FORMAT | The operand field is not of the proper format for this instruction |
| P03 | MISSING OPERAND | Operand field entry missing from instruction requiring one |
| P04 | INVALID SYNTAX IN EXPRESSION | Violation of one or more expression syntax rules |
| P05 | EXPRESSION VALUE TOO LARGE | Final expression value not in range $-2^{16}$ to $2^{16}-1$ |
| P06 | INVALID OPERAND | One or more operand entries do not meet specifications for this instruction |
| P07 | ARITHMETIC OVERFLOW | Intermediate expression value not in the range $-2^{24}$ to $2^{24}-1$ |
| P08 | ADDRESSABILITY ERROR | Relocatable displacement outside the range of USING instruction |
| P09 | REGISTER SPECIFICATION ERROR | Index register specification not 1 or 2 |
| P10 | INVALID CONSTANT | Error in constant specification on DC instruction |
| P11 | INVALID CONSTANT TYPE | Data type specified on DC/DS is not valid |
| P12 | INVALID DUPLICATION FACTOR | Error in duplication factor specification on DC/DS |
| P13 | INVALID LENGTH SPECIFICATION | Error in length specification |
| P14 | INVALID STATEMENT DELIMITER | The column following the statement field is not blank |
| P15 | RELOCATABLE MULTIPLICATION | A relocatable term used in multiply operation |
| P16 | RELOCATABILITY ERROR | A relocatable expression is used where an absolute expression is required, or an absolute expression is used where a relocatable expression is required |
| P17 | INVALID SYMBOL | Invalid character in or invalid length of a symbol in the operand field |
| P18 | INVALID SELF-DEFINING TERM | Error in the format of a self-defining term |
| P19 | SELF-DEFINING VALUE TOO LARGE | Value of self-defining term is outside of range $-2^{16}$ to $2^{16-1}$ |
| P20 | INVALID IMMEDIATE FIELD | Immediate field not in range X'00' to X'FF' |
| P21 | INVALID DISPLACEMENT | Absolute displacement not in range 0 to 255 |

| Code | Diagnostic | Explanation |
|---|---|---|
| P22 | INVALID EXTRN | Symbol is invalid or already defined in the program or subfield is invalid. |
| P23 | TOO MANY ESL RECORDS | More than allowed number of EXTRN and ENTRY statements were found in the program. This count includes multiple EXTRNs and ENTRYs, ENTRYs with valid symbols which are not defined, and EXTRNs with valid symbols which are defined in the program. See *ESL Table Size* in *Part II. Programmer's Guide.* |

# Appendix D: Assembler Language Subroutine To RPG II Linkage

Assembler subroutines can be linked to an RPG II program. The RPG II program passes parameters as it branches to the assembler subroutine. To write a subroutine that will be linked to an RPG II program the following rules must be used:

1. The name of the assembler subroutine must be SUBRxx. xx can be any valid alphabetic characters for user-written subroutines. (Numeric characters are reserved for IBM-supplied subroutines.) The name used must be the same as the name used in the RPG II program.

2. Upon entry to the assembler language subroutine, the address recall register (ARR) contains a pointer to the parameters which represent the fields to be referenced by the assembler subroutine. The return point to the RPG II program is the first byte after the parameters.

3. If the subroutine makes use of registers 1 and 2, the contents of these registers must be stored upon entry to, and restored before exit from, the subroutine.

## USING FIELDS IN THE RPG II PROGRAM

When linkage is effected from RPG II to an assembler subroutine, three possible areas in the RPG II program can be referenced by the subroutine. They are: field, table or array, and indicator.

### Referencing a Field in an RPG II Program

The following parameters (symbolic form of code generated by the compiler) are passed by RPG II when a field is to be referenced:

```
B    SUBRxx

DC   IL1'Field length -1'

DC   AL2(rightmost address of field)
```

### Referencing a Table or Array in an RPG II Program

The following parameters (symbolic form of code generated by the compiler) are passed by RPG II when a table or array is to be referenced:

```
B    SUBRxx

DC   IL1'Entry length-1'

DC   AL2(leftmost address of table control field)
```

The subroutine can refer to the table or array defined in the RPG II program by utilizing the control field created for that table or array. This control field, one of which is created for each table or array built by the RPG II program, is in the following format:

| *Bytes* | *Meaning* |
|---|---|
| 1-2 | Rightmost address of the first entry. |
| 3-4 | Rightmost address of the last entry. |
| 5-6 | Initialized to rightmost address of first entry; used at object time for rightmost address of the last looked-up entry of a table. |
| 7-8 | Length of an entry. |

The subroutine can obtain the data retrieved from the last RPG II table LOKUP by using the address in bytes 5-6. To access the table or array itself, the address in bytes 1-2 must be used.

Data used by the subroutine must be left unpacked for the RPG II program.

#### Referencing an Indicator in an RPG II Program

The following parameters (symbolic form of code generated by the compiler) are passed by RPG II when an indicator is to be referenced:

```
B    SUBRxx
DC   XL1'00'
DC   XL1'Mask for the indicator'
DC   XL1'Displacement to the indicator from XR1'
```

*Note:* The parameters passed to the assembler subroutine are determined by the coding done in the RPG II program. For a description of this coding, see the *IBM System/3 RPG II Reference Manual,* SC21-7504, *IBM System/3 Model 6 RPG II Reference Manual,* SC21-7517, or *IBM System/3 Card System RPG II Reference Manual,* SC21-7500.

### RPG II LINKAGE SAMPLE PROGRAM 1

In this sample program, the RPG II program links to the assembler language subroutine SUBRA (Figure 27). When control is returned to the RPG II program, the character 'A' will have been moved into the field in the RPG II program.

### RPG II LINKAGE SAMPLE PROGRAM 2

In this sample program, the RPG II program links to the assembler subroutine SUBRB (Figure 28). The first parameters passed reference a table. The second parameters reference an indicator. The subroutine refers to both sets of parameters. The subroutine first tests the indicator in the RPG II program. If the indicator is off, control is returned to the RPG II program. If the indicator is on, a character 'C' is moved into the last looked up entry in the table. When control is returned to the RPG II program, it checks for a 'C' in the table.

### I/O SUBROUTINES

Subroutines that support input or output devices can also be linked to an RPG II program. These subroutines are commonly referred to as RPG II SPECIAL subroutines.

#### Linkage for I/O Subroutines

The following linkage is generated by RPG II to communicate with the user-supplied I/O subroutine.

1. DTF (define-the-file) format:

| *Bytes* | *Description* |
|---|---|
| 0 | Device code (X'00') |
| 1 | UPSI mask |
| 2-3 | Attributes |
| 4-5 | Reserved for data management |
| 6-7 | Address of next DTF |
| 8-B | Reserved for data management |
| C-D | Logical record address |
| E | Completion code<br>X'42' = End-of-file<br>X'41' = Controlled cancel (not recognized by Model 10 card system)<br>X'40' = Normal completion (not recognized by Model 10 card system) |
| F | Operation<br>X'C0' = Get and put (model 10 card system only)<br>X'80' = Get<br>X'40' = Put<br>X'20' = Update<br>X'10' = Close |
| 10-11 | Input I/O address |
| 12-13 | Output I/O address |
| 14-15 | Block length |
| 16-17 | Record length |
| 18-19 | Address of array DTT if array linkage is used |

The address of byte 0 of the DTF will be passed to the I/O subroutine in index register 2. Bytes 0-3, 6-7, C-D, and 10-17 are filled in by RPG II at compile time. Byte E, completion code, is inserted by the I/O subroutine when control is returned to RPG II. Byte F, the operation byte, is inserted at object time. The information in bytes 0 and 4-B must be available, unchanged at close time, for data management.

The DTT (define-the-table) is used for array linkage. DTT format:

| *Bytes* | *Description* |
|---|---|
| 0-1 | Address of rightmost byte of the first element of the array. |
| 2-3 | Address of rightmost byte of the last element of the array. |
| 4-5 | RPG last LOKUP element. |
| 6-7 | Length of array element. |

2. The I/O subroutine must save and restore the registers altered in the routine. Control should be returned to the address in the address recall register (ARR).

*Note:* The combined get and put operation code, X'C0', is utilized by the System/3 Model 10 Card System only. The System/3 Model 10 Disk System, System/3 Model 12, and System/3 Model 15 use alternate get and put operations to accommodate combined files. When coding an I/O subroutine to be used on either system, be certain to consider this fact.

When an input operation is done, the I/O subroutine must move the address of the physical buffer currently being used to the logical buffer address location in the DTF (bytes C-D). In the Model 10 Card System, address bytes 10-11 will be the same as bytes C-D (one physical buffer).

When an output operation is requested, the I/O subroutine must move the data from the logical buffer (address in bytes C-D of the DTF) to the physical buffer (address in bytes 12-13 of the DTF). The two addresses are the same in the Model 10 Card System. Bytes 0-B are unused in the Model 10 Card System.

The I/O subroutine must do its own open when the first call to it is issued. It must also do its own close to the file on a close call.

If a dual I/O is requested, the second area will be immediately behind the first (Model 10 Disk System, Model 12, and Model 15 only).

The I/O subroutine cannot be overlaid in the Model 10 Disk System, Model 12, and Model 15.

Sequential processing only is supported.

When an I/O subroutine issues a halt, three halts should be displayed as follows:

1. The first halt issued should be the FF halt reserved by RPG II for SPECIAL I/O subroutine usage.
2. The second halt should be the last two digits of the subroutine name.
3. The third halt may be any valid halt that can be displayed.

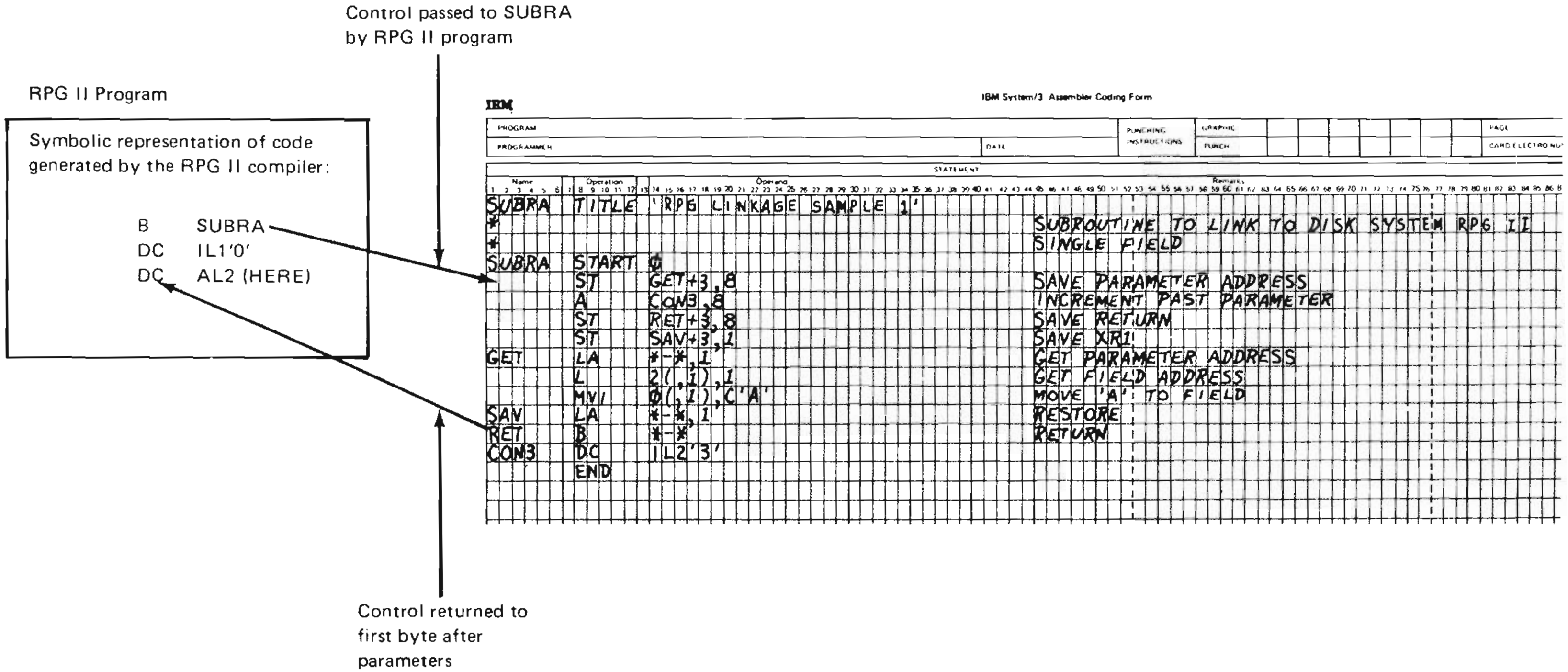



Figure 27. Assembler Language Subroutine (SUBRA) for Sample Program 1

Control passed to SUBRB
by RPG II program

RPG II Program

Symbolic representation of code
generated by the RPG II compiler:

```
     B    SUBRB                                          (3)
(1) {DC   IL1'0'
    {DC   AL2 (leftmost address of table control field)
    {DC   XL1'00'
(2) {DC   XL1'80'
    {DC   XL1'31'
```

IBM System/3 Assembler Coding Form

```
SUBRB  TITLE  'RPG LINKAGE SAMPLE 2'
*                                  SUBROUTINE TO LINK TO DISK SYSTEM RPG II.
*                                  TABLE,INDICATOR
SUBRB  START  0
       ST     GET+3,ARR            SAVE PARAMETER ADDRESS
       A      CON6,ARR             INCREMENT TO RETURN
       ST     RET+3,ARR            SAVE RETURN
       ST     SAVE+3,2             SAVE XR2
GET    LA     *-*,2                GET PARAMETER ADDRESS
       MVC    TEST+2(2),5(,2)      MOVE IN MASK AND DISPLACEMENT
TEST   TBN    *-*(,1),*-*          TEST INDICATOR
       JF     SAVE                 INDICATOR OFF
       L      2(,2),2              GET CONTROL FIELD ADDRESS
       L      5(,2),2              GET LOOK UP ADDRESS
       MVI    0(,2),C'C'           MOVE IN CHARACTER 'C'
SAVE   LA     *-*,2                RESTORE
RET    B      *-*                  RETURN
CON6   DC     IL2'6'
ARR    EQU    8
       END
```

(1) Parameters passed for a table

(2) Parameters passed for an indicator

(3) Control field for table

```
DC   AL2 (address of first entry)
DC   AL2 (address of last entry)
DC   AL2 (address of last looked-up entry [TABB])
DC   IL2'1' (length of entry)
```

Control returned to
first byte after
parameters

Figure 28. Assembler Language Subroutine (SUBRB) for Sample Program, 2

## LIBRARY DECK GENERATOR PROGRAM (MODEL 10 ONLY)

The System/3 Model 10 Card System user can write assembler language subroutines to be used as SPECIAL or EXIT routines in an RPG II program. These assembler routines, however, cannot be inserted directly into the RPG II compiler. The assembler language subroutine must first be assembled by the System/3 Model 10 Disk System Basic Assembler and then translated by the Library Deck Generator (LDG) program before it can be placed in the RPG II compiler.

The entire operation, from writing an assembler subroutine to selection of that subroutine by the IBM System/3 Model 10 Card System RPG II compiler is outlined as follows:

1. The assembler subroutine is written by the programmer. If standard control cards supplied by the LDG program are not being used, the programmer must also code control cards for the subroutine.

2. The assembler subroutine is assembled on the System/3 Model 10 Disk System by the Basic Assembler.

3. The LDG program is read into System/3 Model 10 Disk System storage. The *** parameter card, assembler subroutine object deck, and blank cards are placed in the MFCU.

4. The LDG program produces a deck of cards, containing the subroutine, which can be placed in the RPG II compiler. The deck produced by the LDG program contains the following:

   Header card
   Control cards
   Text
   Q-card
   End card

5. The deck produced by the LDG program may now be placed in the RPG II compiler deck. When an RPG II program is compiled, this subroutine will be selected, when required, just as any other compiler subroutine.

The following material describes the information needed to use an assembler language subroutine in an RPG II program. This material is divided into four major sections:

Writing the assembler language program
Running the LDG program
Output of the LDG program
Example of a SPECIAL subroutine

### Writing the Assembler Language Program

The following information must be considered when the assembler language program is written.

#### *Title Instruction*

The name field of the TITLE instruction must contain 00GEB in columns 1-5.

#### *Control Cards*

Control cards are needed for every assembler language subroutine. Control cards contain code, executed during compile time, which determines whether the subroutine should be included as part of the program being compiled. Library routines are selected only when the execution of a control card determines they are needed. In addition, control cards are needed to ensure that the entry point for the subroutine is placed in the proper location in core for the RPG II compiler to find and use it.

There are two ways to get the control cards you need. In some cases, you will need to code them yourself; in others standard control cards are supplied by the LDG program. If your subroutine is to be used as a normal SPECIAL or EXIT routine, the LDG program will supply three control cards. See Figure 29 for samples of these. When these control cards are provided, a SPECIAL routine is selected if bytes 12-13 of the file description compression matches the identification characters of the routine, and if the SPECIAL device code B'0xxx1010' is present in byte 16 of the same file description compression. EXIT routines are selected if the identifier in the library routine is the same as an entry in the symbol table (bytes 3-4) and if byte 2 of the same entry contains bit configuration 11100000. When these decks are selected, the address of the entry point of associated object code is placed in the symbol table entry, bytes 3-4 for an EXIT reference and/or bytes 8-9 of the file description compression for a SPECIAL reference.

You must code control cards for your subroutine when:

- The subroutine is not a SPECIAL or EXIT routine.
- The subroutine needs a function not provided by the standard control cards.

The following paragraphs describe several compiler resident routines which can be used by programmer coded control cards.

*Coding Control Cards*

There are three types of control cards each identified by a special character in column 1. Each type performs a different function:

- Cards with a J in column 1 (J-cards) are usually used to control the selection of a routine for an object program. They also place the routine entry address in compile time storage for use by the RPG II compiler.
- Cards with a K in column 1 (K-cards) are used only when one routine from a set of related routines is to be used in any job. A J card will determine if any of these routines are needed and if so will start the scan for K cards which in turn control selection of the proper routine.
- Cards with an L in column 1 (L-cards) are used to pass information from RPG II compile time core to a subroutine or vice versa. They are executed only if the deck in which they appear has been selected for use with the current program.

Control card identification characters must be defined for assembly at X'0000' and are placed in column 1 of control cards. The only allowable characters are J, K, L, and blank. There should be one non-blank control card identifier character for each block of code for a control card. The blank is used as a delimiter between control card strings.

For example, DCb̸b̸b̸b̸ CLI0'JKLLb̸Lb̸Lb̸L' shows identifiers for seven control cards and four control card strings. The first is a 4-card string with identifiers 'JKLL' used. The others are single card strings, each of which has an 'L' identification.

LDG identifies the control cards and assigns one control card identification character to each one. The control card strings are merged with the text cards for the routine functional code in the following manner. The first control card string is merged in front of the text, and one additional control card string is merged into the text cards where there is a break in the text caused by a DS or an ORG which changes the location counter.

Each control card must contain executable code. Control cards are coded in the order needed for the purposes described above. Each must begin at X'0017'; therefore, an ORG to 23 or X'0017' must precede the code for each card.

Your control cards must contain instructions for calculating the address at which your subroutine will be loaded. To calculate the true entry address, use the current relocation factor described here.

| *Label* | *Address* | *Function* |
|---|---|---|
| RELOCF | X'030C' to X'030D' | Contains the current relocation factor. Is modified when the end card of the selected deck is encountered or J1EAA1 is entered. |

See Figure 29, Part 1, found at the end of this section, for an example of the use of the current relocation factor.

The following paragraphs describe several compiler resident routines which can be used by programmer coded control cards.

*J-Card Scan Routine* reads the library deck until a J-card is encountered. The routine has three entry points.

| *Label* | *Address* | *Function* |
|---|---|---|
| J3EAA1 | X'031A' | Scans for J-card. When one is found, control is passed to that card. All other cards are ignored. |
| J2EAA1 | X'3014' | Clears X'00E0' to X'00FF' and X'007C' to X'007F' to hex zeroes then scans for J-card as J3EAA1. |
| J1EAA1 | X'030E' | Resets the relocation factor to the next object address and performs as J2EAA1. |

*K-Card Scan Routine* has one entry point.

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| K1EAB1 | X'0320' | Scans for K-card. When one is found, control is passed to that card. All other cards except J-cards are ignored. If a J-card is found, a halt '40' is executed. |

*Relocate Deck Routine* has one entry point.

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| R1EAC1 | X'032C' | Initiates or continues relocation of the current deck. Will recognize and execute L-cards and reorganize and print Q-cards. Exits to J1EAA1 when end card is encountered. |

*Scan File Description Compressions Routine* has two entry points. This routine steps through the file description compressions. It returns a pointer to the next compression in register 2. If the condition code is high, the pointer is valid. Any other condition indicates the pointer is invalid.

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| F1EAE1 | X'0338' | Initializes pointer to first file description compression and sets condition code. |
| F2EAE1 | X'033E' | Points register 2 to the next compression and sets the condition code. (Register 2 need not be pointing to the last compression.) |

*Scan Extension Compressions Routine* has two entry points and steps through the extension compressions and returns a pointer to the next compression in register 2. A high condition code indicates a valid pointer. Any other condition code indicates an invalid (undefined) pointer.

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| E1EAF1 | X'0344' | Initializes pointer to first extension compression and sets condition code. |
| E2EAF1 | X'034A' | Points register 2 to the next compression and sets condition code. (Register 2 need not point to last compression.) |

*Text Handling Routine* builds up full text card in storage and, when a card is full, punches that card. The area from X'0080' to X'00DF' is the location of the punch buffer and this must be considered when using this area of core.

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| BKEAH1 | X'0350' | Forces any partial text card to be punched. |
| STXLA1 | X'035C' | Accepts a string of text to be added to the current text immediately following the last text passed. Requires a 1-byte parameter following the branch. Parameter contains a displacement relative to register 1 to the length byte of the text being passed. The text string should be preceded by this length byte which contains the length of text. |

*Wait On Punch Busy Routine:*

| *Label* | *Entry Point* | *Function* |
|---|---|---|
| WTPUN1 | X'0362' | Returns when the previous punch operation has been successfully completed and the buffer is not busy. |

*Title of Subroutine*

The title of the routine must be a defined constant to be loaded starting at X'0000'. It must be equal to or less than 80 characters in length. This title is printed on the RPG II compiler listing with the address of the entry point of the routine if it is selected at compile time.

*Routine Functional Code*

This code must be assembled starting at X'0000'. The code must contain a break in continuity (a DS or an ORG which changes the location counter value) where control cards are to be inserted.

**Assembling the Subroutine**

The assembler subroutine is assembled by the Model 10 disk system basic assembler. The OCL considerations for assembly are discussed in *Section II: Programmer's Guide* under the headings *OPTIONS Statement* and *OCL Statements For Assembler.*

An OPTIONS card must be used to successfully assemble the subroutine.

**Running the LDG Program**

The following paragraphs describe a special parameter card that must be used with the assembler deck, the OCL required to load the LDG program, and error conditions that may result.

*Library Deck Generator Parameter Card (***)*

A parameter card must precede the assembler generated object deck to provide the LDG program with information regarding output. Entries for the parameter card are as follows:

| *Columns* | *Entry* | *Explanation* |
|---|---|---|
| 1-3 | *** | Three asterisks identify a parameter card. |
| 4-9 | SUBRxx | These characters identify the subroutine. Substitute any two characters for xx — the second may be blank, but the first must not. Note that the LDG program will not diagnose an error in these columns. |
| 10 | , (comma) | Required. |
| 11 | S | Standard control cards will be provided by the LDG program for the subroutine identified by the characters found in columns 8-9 of this parameter card. The title, also extracted from this parameter card, will be assigned to the subroutine. The entry point of the routine must be the first byte of the routine. GEB will be forced as module identifier. |
| | N | Non-standard control cards will be supplied by the user as will identification characters and title. (The format of this material may be found in Figure 29.) If N is specified, the title specified in this parameter card is ignored. Thus, if N is used, columns 21-96 may be left blank. |
| 12 | , (comma) | Required. |
| 13 | D | Default values for component version, modification level, and indication of complete or partial deck replacement for header card are provided by the LDG program. |
| | G | Default values are not assumed. The user must provide them in columns 15-19. |
| 14 | , (comma) | Required if column 11 contains an S or column 13 a G. |
| 15-16 | VV | Two numbers indicating the component version. |
| 17-18 | MM | Two numbers indicating modification level. |
| 19 | 0 (zero) | Partial deck replacement for header card. |
| | 1 | Complete deck replacement for header card. |
| 20 | , (comma) | Required only if column 13 contains a G and column 11 an S. |
| 21-96 | Subroutine title | If column 11 contains an N, the title is not required. If column 13 contains a D, the title of the subroutine must begin in column 15. |

*Examples:*

IBM

```
***SUBRA ,N,D
```

User will supply all control cards, identifying characters, and title for subroutine 'Aƀ'.

IBM

```
***SUBRBB,S,G,Ø2ØØ1,SPECIAL ROUTINE BB
```

Library Deck Generator will supply standard control cards which will be used for selection of subroutine BB. The title will be printed on the 4th tier of the cards and on the compiler listing. The values given in columns 15-19 will be used on the header card. The component version (02) will go in columns 59-60 of the header card, the modification level (00) will go in columns 31-32, and deck replacement indicator (1) will be placed in column 85.

*Loading the LDG Program*

IBM

```
// LOG PRINTER
// LOAD $ASLDG,R1
// RUN
***SUBR  (PARAMETER CARD)
      }
  ASSEMBLER OBJECT PROGRAM
      }
/*
```

*Error Conditions*

Several errors are considered to be terminal. If terminal errors occur, the card image is printed, the error message is printed, the deck is run through to the '/*' card, and a C halt is displayed. When this halt is reset, processing is discontinued by the end-of-job routine.

If the error is not terminal, the card image is printed, an error message is printed, and a C halt is displayed. The program is restartable, however, and processing will continue.

Following is a list of error messages generated by this phase. An asterisk (*) preceding the number indicates which are warning errors.

1. Number of control cards generated incorrect.
2. Length of control card text, too great for one card.
3. Card sequence incorrect.
4. Title too long or the first text is contiguous.

*5. First control card character may not be blank.

6. Not enough breaks for control strings.

*7. More breaks than control strings.

*8. Last text not at highest address expected.

9. Improper card in deck.
10. End card out of sequence.
11. Invalid control card identification.
12. First object card must be an ESL card.
13. Insufficient core for control card storage.
14. Invalid entries on *** control card.

*15. /* card or *** card out of sequence.

*16. GEB not used as module identifier.

17. *** card required before object deck.
18. Too many control card identifiers specified or invalid sequence.

### Output of the LDG Program

The header card in stacker 2 should be placed in front of the remainder of the output deck in stacker 3. Insert the subroutine deck in the RPG II Compiler deck using the Program Maintenance Program. The subroutine deck must have GEB in columns 91-93.

### Example

Figure 29 is an example of a SPECIAL subroutine. This sample program can be used as a base for any SPECIAL or EXIT subroutine. The only changes required are modifying the subroutine identification characters, entry point, label, and routine title. Areas of change are outlined in the sample listing. Control cards are created for you.

```
00GEB  ANY TITLE DESIRED MAY BE USED

ERR LOC  OBJECT CODE      ADDR STMT SOURCE STATEMENT
                                2 * ****************************************************************** *   00020000
                                3 *                                                                    *   00030000
                                4 *    THIS IS A SAMPLE CODING FOR THE CONTROL CARDS FOR A 'SPECIAL'   *   00040000
                                5 *                                                                    *   00050000
                                6 *    DEVICE REFERENCED IN AN RPG PROGRAM. ALL LABELS WHICH WILL      *   00060000
                                7 *                                                                    *   00070000
                                8 *    NEED TO BE MODIFIED FOR A PARTICULAR PROGRAM HAVE LABELS        *   00080000
                                9 *                                                                    *   00090000
                               10 *    STARTING WITH THE CHARACTER '#'. THIS DECK IS IN THE FORMAT     *   00100000
                               11 *                                                                    *   00110000
                               12 *    REQUIRED BY THE LIBRARY DECK GENERATOR.                         *   00120000
                               13 *                                                                    *   00130000
                               14 *    THESE CONTROL CARDS MAY BE USED FOR ANY SPECIAL OR EXIT         *   00140000
                               15 *                                                                    *   00150000
                               16 *    SUBROUTINE.                                                     *   00160000
                               17 *                                                                    *   00170000
                               18 * ****************************************************************** *   00180000

                               20 * ****************************************************************** *   00200000
                               21 *                                                                    *   00210000
                               22 *      STANDARD LABELS AND LABELS USED TO LINK TO THE LIBRARY        *   00220000
                               23 *                                                                    *   00230000
                               24 *         SELECT ROUTINE AND RPG COMPILER COMMUNICATIONS AREA        *   00240000
                               25 *                                                                    *   00250000
                               26 * ****************************************************************** *   00260000

    0000                       28 START   START 0                 PROGRAM SHOULD BE STARTED AT 0           00280000
                          0001 29 XR1     EQU   1                 STANDARD LABEL FOR INDEX REGISTER 1      00290000
                          0002 30 XR2     EQU   2                 STANDARD LABEL FOR INDEX REGISTER 2      00300000
                          0008 31 ARR     EQU   8                                     ADDRESS RECALL REG   00310000

                          030D 33 RELOCF EQU    START+X'030D'     RELOCATION FACTOR FOR CURRENT DECK       00330000
                          030E 34 J1EAA1 EQU    START+X'030E'     ENTRY POINT TO RESET RELOCATION          00340000
                               35 *                                 FACTOR AND SCAN TO NEXT 'J' CARD       00350000
                          031A 36 J3EAA1 EQU    START+X'031A'     ENTRY TO SCAN TO NEXT 'J' CARD WITH-     00360000
                               37 *                                 OUT RESETTING RELOCATION FACTOR        00370000
                          032C 38 R1EAC1 EQU    START+X'032C'     ENTRY POINT TO INITIATE OR CONTINUE      00380000
                               39 *                                 RELOCATION OF THIS DECK                00390000
                          0338 40 F1EAE1 EQU    START+X'0338'     ENTRY POINT TO INITIATE THE SCAN OF      00400000
                               41 *                                 THE FILE DESCRIPTION COMPRESSIONS      00410000
                          033E 42 F2EAE1 EQU    START+X'033E'     ENTRY POINT TO CONTINUE FILE DISC.       00420000
                               43 *                                 COMP. SCAN                             00430000
                               44 *                                     BOTH OF THE PREVIOUS ENTRIES       00440000
                               45 *                                     RETURN A POINTER IN XR2 AND A      00450000
                               46 *                                     CONDITION CODE 'HIGH' IF THAT      00460000
                               47 *                                     POINTER IS VALID                   00470000

                          028C 49 COMMON EQU    START+X'028C'     START OF THE RPG COMPILER                00490000
                               50 *                                 COMMUNICATIONS AREA                    00500000
                          02E6 51 ENDCOR EQU    COMMON+90         HOLDS LAST ADDRESS IN MEMORY -FIRST      00510000
                               52 *                                 BYTE USED FOR SYMBOL TABLE -           00520000
                          02EA 53 ENDST  EQU    COMMON+94         HOLDS LAST ADDRESS USED FOR SYMBOL       00530000
                               54 *                                 TABLE.                                 00540000
```

Figure 29 (Part 1 of 4). Sample Coding for SPECIAL Device

```
ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT

                                56 * ************************************************************** *      00560000
                                57 *                                                                *      00570000
                                58 *      THE FOLLOWING IS A SKELETON FOR A FILE DESCRIPTION        *      00580000
                                59 *                                                                *      00590000
                                60 *        COMPRESSION                                             *      00600000
                                61 *                                                                *      00610000
                                62 * ************************************************************** *      00620000

    0000                  0000  64 FCFG    DS    CL1               FLAG BYTE FOR COMP. ALWAYS X'FF'           00640000
    0001                  0002  65         DS    CL2               OUTPUT BUFFER @                            00650000
    0003                  0004  66         DS    CL2               INPUT  BUFFER ADDRESS                      00660000
    0005                  0006  67         DS    CL2               PRINT  BUFFER ADDRESS                      00670000
    0007                  0008  68 FCENT@  DS    CL2               IOCS ENTRY POINT ADDRESS                   00680000
    0009                  0009  69         DS    CL1               FLAG BYTE                                  00690000
    000A                  000A  70         DS    CL1               FLAG BYTE                                  00700000
    000B                  000C  71 FCIDNT  DS    CL2               HOLDS IDENT FOR SPECIAL ROUTINE            00710000
    000D                  000E  72         DS    CL2               EXTERNAL INDICATOR ASSIGNMENT              00720000
    000F                  000F  73 FCDVA   DS    CL1               DEVICE CODE B'0XXX1010' FOR SPECIAL        00730000
    0010                  0010  74         DS    CL1               BLOCKING FACTOR                            00740000
    0011                  0011  75         DS    CL1               RECORD LENGTH                              00750000


                                77 * ************************************************************** *      00770000
                                78 *                                                                *      00780000
                                79 *      THE FOLLOWING IS A SKELETON FOR A SYMBOL TABLE ENTRY      *      00790000
                                80 *                                                                *      00800000
                                81 * ************************************************************** *      00810000

    0012                  0012  83 STLEN   DS    CL1               LENGTH FOR FIELD ENTRY                     00830000
    0013                  0013  84 STFLAG  DS    CL1               FLAG BYTE SPECIAL NEEDS B'        '        00840000
    0014                  0015  85 STIDNT  DS    CL2               IDENT FOR SPECIAL C'##' HOLDS ENTRY        00850000
                                86 *                                 POINT AFTER SELECTION                    00860000


                                88 * ************************************************************** *      00880000
                                89 *                                                                *      00890000
                                90 *      THE FOLLOWING DC CONTAINS THE ID'S FOR THE CONTROL CARDS  *      00900000
                                91 *                                                                *      00910000
                                92 * ************************************************************** *      00920000


    0000                        94         ORG   0                                                          00940000
    0000 D1D1D1           0002  95         DC    CL3'JJJ'          THREE CONTROL CARDS ALL WITH IDENT         00950000
                                96 *                               'J' AND INSERTED IN FRONT OF THE           00960000
                                97 *                                DECK                                      00970000
                                99 * ************************************************************** *      00990000
                               100 *                                                                *      01000000
                               101 *      THIS CONTROL CARD SCANS THE 'F' COMPRESSIONS FOR REFERENCE TO *  01010000
                               102 *                                                                *      01020000
                               103 *       '##'  IF FOUND IT SETS THE FLAG BYTE AT X'0078' TO X'FF'. *     01030000
                               104 *                                                                *      01040000
                               105 *       IF EITHER FOUND OR NOT FOUND IT STARTS THE SCAN FOR THE NEXT *  01050000
                               106 *                                                                *      01060000
                               107 *       CONTROL CARD.                                            *      01070000
                               108 *                                                                *      01080000
                               109 * ************************************************************** *      01090000


    0017                       111         ORG   X'0017'                 REQUIRED FOR EACH CONTROL CARD       01110000
                          0078 112 FLG     EQU   START+X'78'             AREA FROM X'78' TO X'FF' IS          01120000
                               113 *                                       USABLE FOR WORKING STORAGE         01130000
                               114 *                                       THIS BYTE USED TO FLAG IF          01140000
                               115 *                                       ROUTINE IS REFERENCED ON 'F'       01150000
                               116 *                                       SPECIFICATIONS                     01160000
                          0000 117         USING START,XR1               VALID AT ENTRY TO ANY CTL. CARD      01170000
    0017 7C 00 78              118         MVI   FLG(,XR1),X'00'         INITIALIZE FLAG FOR NOT USED         01180000
                               119 *                                    ON FILE DESCRIPTION SPECS.          01190000
    001A 4E 01 43 030D         120         ALC   #ENTRY(2,XR1),RELOCF    CALCULATE TRUE ENTRY ADDRESS         01200000
    001F C0 87 0338            121         B     F1EAE1                  INITIATE SCAN OF 'F' COMPS.          01210000
                          0000 122         USING FCFG,XR2                VALID UPON RETURN FROM F1EAE1        01220000
    0023 6D 01 45 0C           123 SPCA1   CLC   #IDENT(2,XR1),FCIDNT(,XR2) IS THE IDENT THE RIGHT CHAR       01230000
    0027 B8 0A 0F              124         TBN   FCDVA(,XR2),B'00001010'   AND IS DEVICE CODE THAT FOR        01240000
    002A B9 85 0F              125         TBF   FCDVA(,XR2),B'10000101'   'SPECIAL'                          01250000
    002D F2 96 07              126         JC    SPCA2,X'96'             IF THIS IS NOT THE RIGHT COMP, JUMP  01260000

    0030 7C FF 78              128         MVI   FLG(,XR1),X'FF'         SET FLAG TO INDICATE USED ON         01280000
                               129 *                                    FILE DESCRIPTION SPECS.             01290000
    0033 9C 01 08 43           130         MVC   FCENT@(2,XR2),#ENTRY(,XR1) MOVE ENTRY ADDRESS TO THE         01300000
                               131 *                                      FILE DESCRIPTION COMP.            01310000
    0037 C0 87 033E            132 SPCA2   B     F2EAE1                  ELSE SCAN TO NEXT COMP               01320000
    003B D0 84 23              133         BH    SPCA1(,XR1)             IF POINTER STILL OK LOOP             01330000
    003E C0 87 031A            134         B     J3EAA1                  GET NEXT 'J' CARD                    01340000
                               135 *                                    THIS ENTRY WILL NOT CLEAR THE       01350000
                               136 *                                    BYTE AT FLG.                        01360000


    0042 0000             0043 138 #ENTRY  DC    AL2(SUBR##)             ENTRY POINT ADDR. TO BE RELOCAT      01380000
    0044 7B7B             0045 139 #IDENT  DC    CL2'##'                 TWO CHARACTER IDENT FOR ROUTINE      01390000

                          0002 141         DROP  XR2                                                        01410000
```

Identify your subroutines by replacing these # signs with identifying characters.

Figure 29 (Part 2 of 4). Sample Coding for SPECIAL Device

```
ERR LOC  OBJECT CODE          ADDR STMT SOURCE STATEMENT

                                    143 * *******************************************************************  *      01430000
                                    144 *                                                                      *      01440000
                                    145 *        THIS CONTROL CARD DETERMINES THE END ADDRESS TO BE USED       *      01450000
                                    146 *                                                                      *      01460000
                                    147 *        IN THE SEARCH OF THE SYMBOL TABLE DONE BY THE NEXT CONTROL    *      01470000
                                    148 *                                                                      *      01480000
                                    149 *        CARD.                                                         *      01490000
                                    150 *                                                                      *      01500000
                                    151 * *******************************************************************  *      01510000


                               007D 153 END@     EQU    START+X'7D'                 THIS TWO BYTE AREA WILL HOLD        01530000
                                    154 *                                           THE ADDRESS TO CONTROL THE          01540000
                                    155 *                                           SYMBOL TABLE SCAN. IT WILL BE       01550000
                                    156 *                                           THE ADDRESS OF THE END OF THE       01560000
                                    157 *                                           SYMBOL TABLE OR THE FIRST           01570000
                                    158 *                                           TABLE ADDRESS TABLE POINTER         01580000
                                    159 *                                           WHICH EVER IS HIGHEST               01590000

    0017                            161          ORG    X'0017'                                                         01610000
    0017 4C 01 7D 02EA              162          MVC    END@(2,XR1),ENDST           INITIALIZE END ADDRESS TO END       01620000
                                    163 *                                             OF SYMBOL TABLE                   01630000
    001C C2 02 FFFC                 164          LA     X'FFFC',XR2                 INITIALIZE XR2 TO NEGATIVE 4        01640000
    0020 36 02 02E6                 165          A      ENDCOR,XR2                  POINT XR2 TO FIRST ENTRY IN         01650000
                                    166 *                                             SYMBOL TABLE                      01660000
                               0011 167          USING  STLEN-1,XR2                                                     01670000
    0024 B9 18 02                   168          TBF    STFLAG(,XR2),X'18'          TEST IF ENTRY FOR TABLE OR          01680000
                                    169 *                                             ARRAY                             01690000
    0027 F2 10 04                   170          JT     SPCB0                       IF NEITHER --> JUMP                 01700000
    002A 6C 01 7D 04                171          MVC    END@(2,XR1),STIDNT(,XR2)    ELSE RESET THE END ADDRESS          01710000
    002E C0 87 031A                 172 SPCB0    B      J3EAA1                      GO GET NEXT CARD                    01720000
                               0002 173          DROP   XR2                                                             01730000
                                    175 * *******************************************************************  *      01750000
                                    176 *        THIS CONTROL CARD CHECKS THE SYMBOL TABLE FOR REFERENCES FROM  *     01760000
                                    177 *                                                                      *      01770000
                                    178 *        CALCULATIONS. IF REFERENCED THERE OR ON 'F' SPECS RELOCATION   *     01780000
                                    179 *                                                                      *      01790000
                                    180 *        OF THE DECK IS INITIATED                                      *      01800000
                                    181 *                                                                      *      01810000
                                    182 * *******************************************************************  *      01820000


    0017                            184          ORG    X'0017'                     START OF CONTROL CARD TEXT          01840000
    0017 4E 01 51 030D              185          ALC    #ENT(2,XR1),RELOCF          CALCULATE ENTRY ADDRESS             01850000
    001C 4C 01 30 02E6              186          MVC    SPCB2+3(2,XR1),ENDCOR       INITIALZE LA BELOW                  01860000
    0021 5E 01 30 55                187 SPCB1    ALC    SPCB2+3(2,XR1),STSTEP(,XR1) STEP BACK TO NEXT ENTRY             01870000
    0025 4D 01 30 02EA              188          CLC    SPCB2+3(2,XR1),ENDST        CHECK FOR END OF SYMBOL TABLE       01880000
    002A F2 82 18                   189          JL     SPCB3                       IF BEYOND END --> JUMP              01890000
    002D C2 02 0000                 190 SPCB2    LA     *-*,XR2                     POINT TO ENTRY                      01900000
                               0011 191          USING  STLEN-1,XR2                                                     01910000
    0031 9D 01 04 53                192          CLC    STIDNT(2,XR2),#IDN(,XR1)    IS THE IDENT CORRECT AND            01920000
    0035 B8 E0 02                   193          TBN    STFLAG(,XR2),B'11100000'    THE ENTRY FOR AN EXIT LABEL         01930000
    0038 D0 96 21                   194          BC     SPCB1(,XR1),X'96'           IF NOT CORRECT ENTRY --> LOOP       01940000
    003B 9C 01 04 51                195          MVC    STIDNT(2,XR2),#ENT(,XR1)    ELSE MOVE IN ENTRY POINT            01950000
    003F BA 01 02                   196          SBN    STFLAG(,XR2),B'00000001'    SET FLAG FOR ROUTINE FOUND          01960000
    0042 F2 87 07                   197          J      SPCB4                       START RELOCATION OF ROUTINE         01970000
    0045 7D FF 7B                   198 SPCB3    CLI    FLG(,XR1),X'FF'             WAS ROUTINE REFERENCED FROM         01980000
                                    199 *                                             FILE DESCRIPTION  SPECS. ?        01990000
    004B C0 01 030E                 200          BNE    J1EAA1                      NO - UNUSED SCAN TO NEXT DECK       02000000
    004C C0 87 032C                 201 SPCB4    B      R1EAC1                      YES - USED AS SPECIAL RELOCATE      02010000


    005D 0000                  0051 203 #ENT     DC     AL2(SUBR##)                 ENTRY POINT FOR RELOCATING          02030000
    0052 7B7B                  0053 204 #IDN     DC     CL2'##'                     IDENTIFICATION                      02040000


    0054 FFFC                  0055 206 STSTEP   DC     1L2'-4'                     NEGATIVE LENGTH OF SYMBOL           02060000
                                    207 *                                             TABLE ENTRY                       02070000
```

Replace these # signs with the characters identifying your subroutine.

Figure 29 (Part 3 of 4). Sample Coding for SPECIAL Device

```
ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT
                                209 * ******************************************************************* *     02090000
                                210 *                                                                     *     02100000
                                211 *       THE FOLLOWING DC CONTAINS THE PROGRAM TITLE TO BE PRINTED     *     02110000
                                212 *                                                                     *     02120000
                                213 *         ON THE RPG LISTING AND SHOULD BE CHANGED TO REFLECT THE     *     02130000
                                214 *                                                                     *     02140000
                                215 *         NAME OF THE SUBROUTINE.                                     *     02150000
                                216 *                                                                     *     02160000
                                217 * ******************************************************************* *     02170000

    0000                        219        ORG    0                          SIGNALS START OF TITLE             02190000


    0000 E2D7C5C3C9C1D340 001D  221        DC     CL30'SPECIAL I/O ROUTINE ##'                                  02210000
    0008 C961D640D9D6E4E3       221
    0010 C9D5C540787B4040       221
    0018 404040404040           221
```

Replace these # signs with the characters identifying your subroutine.

```
                                223 * ******************************************************************* *     02230000
                                224 *                                                                     *     02240000
                                225 *       THE FOLLOWING CODE REPRESENTS THE FUNCTIONAL CODE FOR THE     *     02250000
                                226 *                                                                     *     02260000
                                227 *       USER ROUTINE. THE ABOVE CONTROL CARDS ASSUME THE ENTRY POINT  *     02270000
                                228 *                                                                     *     02280000
                                229 *       IS AT SUBR##.  THE ENTRY POINT IS UNIQUE TO EACH SUBROUTINE.  *     02290000
                                230 *                                                                     *     02300000
                                231 *       THE ENTRY POINT IS THE LABEL ON THE ROUTINE CODE, NOT THAT    *     02310000
                                232 *                                                                     *     02320000
                                233 *       ON THE START CARD.                                            *     02330000
                                234 *                                                                     *     02340000
                                235 * ******************************************************************* *     02350000


                                237 * ******************************************************************* *     02370000
                                238 *                                                                     *     02380000
                                239 *           THE ROUTINE MUST MEET THE FOLLOWING REQUIREMENTS          *     02390000
                                240 *                                                                     *     02400000
                                241 *        1. WHEN ENTERED FOR INPUT OR OUTPUT (NOT EXIT) IT MUST       *     02410000
                                242 *           ACCEPT THE STANDARD SPECIAL I/O LINKAGE PARAMETERS.       *     02420000
                                243 *                                                                     *     02430000
                                244 *        2. WHEN ENTERED VIA AN EXIT FROM CALCULATIONS IT MUST        *     02440000
                                245 *           ACCEPT THE STANDARD EXIT LINKAGE AND PARAMETERS.          *     02450000
                                246 *                                                                     *     02460000
                                247 *        3. IT MUST INDICATE END OF FILE BY PROVIDING THE CORRECT     *     02470000
                                248 *           COMPLETION CODE IN THE DTF.                               *     02480000
                                249 *                                                                     *     02490000
                                250 *        4. IF A DIFFERENT AREA IS USED FOR THE ACTUAL INPUT OR       *     02500000
                                251 *           OUTPUT BUFFER THE DATA MUST BE MOVED TO OR FROM THE ADDRESS*     02510000
                                252 *           SUPPLIED IN THE DTF.                                      *     02520000
                                253 *                                                                     *     02530000
                                254 * ******************************************************************* *     02540000


    0000                        256        ORG    0                     SIGNALS START OF ROUTINE TEXT           02560000

                           0000 258 SUBR## EQU    *                     THIS IS THE ENTRY POINT TO THE ROUT.    02580000

                                260 *******      ROUTINE CODE IS PLACED HERE       *********                    02600000

                           0000 262        END    SUBR##                THIS INSURES PROPER LISTING FROM RPG    02620000
```

Replace these # signs with the characters identifying your subroutine.

Figure 29 (Part 4 of 4). Sample Coding for SPECIAL Device

## Appendix E: Assembler Language Subroutine To COBOL or FORTRAN Linkage

This section describes standard linkage conventions for use between modules produced by the System/3 language translators: COBOL, FORTRAN, and Basic Assembler. Programmers using standard linkage conventions are able to code routines in the language most appropriate to the function being performed, with the assurance that effective and permanent communication has been established. Figure 30 illustrates the standard described on the following pages.

```
*
*       SAMPLE SYSTEM/3 LINKAGE -- MODULE A CALLS MODULE B
*
              EXTRN MODB
@XR1    EQU   X'01'
@XR2    EQU   X'02'
*
MODA    START X'0000'
*
*       INITIALIZE XR1 AND XR2 TO TEST SAVING
*
              L     XR1,@XR1
              L     XR2,@XR2
              B     MODB            CALL MODULE B
              DC    AL2(PLIST)
              HPL   X'6F',X'6F'     HALT 00 AFTER RETURN
*
*     PARAMETER LIST
*
PLIST       EQU   *
            DC    AL2 (SAVA)        ADDRESS OF SAVE AREA
            DC    AL2 (PARM1)       ADDRESS OF FIRST PARAMETER
            DC    AL2 (PARM2)       ADDRESS OF SECOND PARAMETER
            DC    XL1'00"
*
*     PARAMETERS
*
PARM1       EQU   EQU    *
            DC           CL5'FIRST'
PARM2       EQU          *
            DC           CL6'SECOND'
*
*      SAVE AREA
*
SAVA          DC    XL1'B0'         INDICATOR BYTE -- ASSEMBLER MAIN
              DC    CL6'MODE'       MODULE NAME
*
XR1           DC    CL2'R1'
XR2           DC    CL2'R2'
              END   MODA
```

Figure 30 (Part 1 of 2). Illustration of Standard Linkages

```
*
*       SAMPLE SYSTEM/3 LINKAGE -- MODULE A CALLS MODULE B
*
@XR1       EQU   X'01'
@XR2       EQU   X'02'
@ARR       EQU   X'08'
@IAR       EQU   X'10'
*
           ENTRY MODB
*
MODB       START X'0000'
*
           ST    SAVAR1,@XR1              SAVE CONTENTS OF XR1
           LA    SAVA,@XR1                XR1 WILL BE BASE FOR SAVE AREA
           USING SAVA,@XR1
           ST    SAVAR2(,@XR1),@XR2       SAVE CONTENTS OF XR2
           ST    SAVART(,@XR1),@ARR       SAVE CONTENTS OF ARR
           L     SAVART(,@XR1),@XR2       XR2 POINTS TO ADDRESS OF PARM
                                            LIST
           L     1(,@XR2),@XR2            XR2 POINTS TO PARAMETER LIST
           ALC   SAVART(,@XR1),TWO(,@XR1) SET RETURN POINT 2 PAST ARR.
*
*     BODY OF ROUTINE
*
           L     SAVAR2(,@XR1),@XR2       RESTORE XR2
           L     SAVAR1(,@XR1),@XR1       RESTORE XR1
           L     SAVART,@IAR              RETURN
*
*     SAVE AREA
*
SAVA       DC    XL1'30'                  INDICATOR BYTE -- ASSEMBLER LANG
           DC    CL6'MODB'                MODULE NAME
SAVAR1     DC    XL2'00'                  CONTENTS OF XR1 ON ENTRY TO THIS
*                                           MODULE
SAVAR2     DC    XL2'00'                  CONTENTS OF XR2 ON ENTRY TO THIS
*                                           MODULE
SAVART     DC    AL2(00)                  RETURN POINT
*
TWO        DC    IL2'2'
*
           END
```

Figure 30 (Part 2 of 2). Illustration of Standard Linkages

## STANDARDS

In order to be standard, linkage must be accomplished as follows:

1. Each module must have a *save area* (Figure 31).

| Byte | Bit | Description | Program |
|---|---|---|---|
| 0 | 0 | 0=Not a main program<br>1=Main program | Subroutine<br>Main program |
| | 1-3 | 000=FORTRAN<br>001=COBOL<br>011=Basic Assembler | Subroutine<br>Main program |
| | 4-7 | Reserved | |
| 1-6 | | EBCDIC name, left justified | Subroutine<br>Main program |
| 7-8 | | Value of index register 1 (XR1) at entry | Subroutine |
| 9-A | | Value of index register 2 XR2) at entry | Subroutine |
| B-C | | Return point in calling program | Subroutine |

*Note:* Main program refers to the program with the highest level of control.

Figure 31. Save Area

2. Each module that calls another module must have one or more *parameter lists* (Figure 32).

| Byte | Description |
|---|---|
| 0-1 | Address of save area in this program |
| 2-3 | Address of first parameter |
| (2N)-(2N+1) | Address of Nth parameter |
| (2N+2) | XL1'00' to indicate end of parameter list |

*Note:* The first two bytes as well as the end-of-parameter-list indicator (XL1'00') must be present in all parameter lists. If no parameters are to be passed, the parameter list will be only three bytes in length. In this case, byte 3 will be 0 and the called program will indicate a parameter list length of 2.

*Note:* Addresses in parameter lists refer to the first byte (byte with the lowest address) of the item.

Figure 32. Parameter List

3. When control reaches a program entry point, the address recall register (ARR) must point to a 2-byte field containing the address of the first byte of the parameter list.

   The Basic Assembler language code to call a COBOL or FORTRAN subroutine would normally be as follows:

```
        EXTRN   SUBR
        B       SUBR
        DC      AL2(PARAMS)
RETNPT  EQU     *
```

   Note that the pointer to the parameter list points to the left byte of the save area address.

4. Normal return is accomplished by placing in the instruction address register (IAR) a value that is two larger than the contents of the ARR when the program was entered.

5. Index registers 1 and 2 (XR1 and XR2) must be saved upon entry in the called program's save area, and restored at exit.

6. The address recall register need not be restored, but the return address must be determined and placed in the called program's save area.

Along with the Basic Assembler, you will receive a sample program. By executing the sample program you can verify that the Basic Assembler is operational.

## MODEL 10 AND MODEL 12 SAMPLE PROGRAM

This section describes the sample program and explains the operating procedures necessary for executing it. General operating procedures for the Basic Assembler are found in the *IBM System/3 Model 10 Disk System Operator's Guide,* GC21-7508, *IBM System/3 Model 12 Operator's Guide,* GC21-5144, and in Part II of this manual.

### Program Description

The sample program is called Prime Number Test Program. The program reads a number from the console display data switches, tests to see if it is a prime number, and indicates the results of the test on the message display unit. If the number zero is tested, the program is terminated.

Three halt codes are used in this program to request input and indicate whether the number is prime. They are:

| *Halt Code* | *Meaning* |
|---|---|
| EN | Enter a number to be tested. |
| IP | The number tested is prime. |
| NP | The number tested is not prime. |

Figure 33 shows the OCL that assembles, link edits, and executes the sample program. Figure 34 shows the sample program statements.

IBM

IBM System/3 Basic Assembler Coding Form

```
// NOHALT
// LOAD $ASSEM,F1
// FILE NAME-$SOURCE,RETAIN-S,UNIT-R2,PACK-R2R2R2,TRACKS-5
// FILE NAME-$WORK,RETAIN-S,UNIT-F2,PACK-F2F2F2,TRACKS-2
// FILE NAME-$WORK2,RETAIN-S,UNIT-F1,PACK-F1F1F1,TRACKS-5
// COMPILE SOURCE-$ASSPL,UNIT-R1,OBJECT-R1
// RUN
// LOAD $OLINK,F1
// FILE NAME-$SOURCE,RETAIN-S,UNIT-R2,PACK-R2R2R2,TRACKS-10
// FILE NAME-$WORK,RETAIN-S,UNIT-F2,PACK-F2F2F2,TRACKS-10
// RUN
```

IBM

IBM System/3 Basic Assembler Coding Form

```
// PHASE NAME-$ASSPO
// OPTIONS MAP-XREF
// INCLUDE NAME-$ASSPR,UNIT-R1
// END
// HALT
// LOAD $ASSPO,F1
// RUN
```

NOTES:

1. Specifies the location of the assembler program.
2. Name of assembler sample program in the source library.
3. Specifies the source library with the sample program.
4. Library in which the output assembler object (R) module is stored.
5. Name given to the output assembler object (O) program.
6. Module name and object program name (R).
7. Specifies the object (O) program, stored on the Overlay Linkage Editor program pack by default.

If the system configuration does not include drive 2, references in the OCL to F2 and R2 must be changed to specify devices available on the system.

Figure 33. Model 10 and Model 12 Sample Program OCL

```
 OPTIONS NODECK                                                                     0001
THE LIST OF OPTIONS USED DURING THIS ASSEMBLY IS--    NODECK,LIST,XREF,REL,OBJ
```

```
$ASSPR                          EXTERNAL SYMBOL LIST
SYMBOL    TYPE                                          VER 13.  MOD 00  01/30/76  PAGE  1
$ASSPR    MODULE
```

```
$ASSPR   PRIME NUMBER TEST PROGRAM
ERR LOC   OBJECT CODE       ADDR STMT SOURCE STATEMENT          VER 13.  MOD 00  01/30/76  PAGE  2

                                  2 *                                                                        0003
                                  3 *   THIS PROGRAM READS A NUMBER FROM THE CONSOLE DISPLAY DATA SWITCHES, TESTS IT FOR     0004
                                  4 * PRIMENESS, AND INDICATES THE RESULTS ON THE MESSAGE DISPLAY UNIT.                      0005
                                  5 *                                                                        0006
                                  6 *   THERE ARE THREE HALT CODES USED IN THIS PROGRAM:                     0007
                                  7 *             HALT CODE    MEANING                                       0008
                                  8 *                EN        ENTER A NUMBER TO BE TESTED.  IF NUMBER ENTERED IS ZERO THE  0009
                                  9 *                          PROGRAM TERMINATES.                           0010
                                 10 *                IP        NUMBER IS PRIME.                              0011
                                 11 *                NP        NUMBER IS NOT PRIME.                          0012
                                 12 *                                                                        0013
    0000                         13 $ASSPR START 0                                                           0014
                          0000   14        USING *,XR1                          ESTABLISH BASE REGISTER     0015
    0000 C2 01 0000              15        LA    *,XR1                          LOAD BASE REGISTER          0016
    0004 F0 7C 2F                16 BEGIN  HPL   X'2F',X'7C'                    'EN' HALT                   0017
    0007 70 00 78                17        SNS   SENSE(,XR1),0                  SENSE THE DATA SWITCHES     0018
    000A 50 01 78 70             18        CLC   SENSE(2,XR1),ZERO(,XR1)        TEST INDICATION TO QUIT     0019
    000E F2 01 05                19        JNE   PREPAR                         NUMBER TO TEST              0020
    0011 C0 87 0004              20        B     4                              GO TO END OF JOB            0021
    0015 84               0015   21        DC    XL1'84'                                                    0022
                                 22 *                                                                        0023
                                 23 *             PREPARE THE INPUT NUMBER                                  0024
    0016 50 01 78 76             24 PREPAR CLC   SENSE(2,XR1),THREE(,XR1)       TEST FOR ONE,TWO AND THREE  0025
    001A F2 04 4C                25        JNH   PRIME#                         CALL ONE, TWO AND THREE PRIME  0026
    001D 78 01 78                26        TBN   SENSE(,XR1),X'01'              TEST FOR EVEN               0027
    0020 F2 90 40                27        JF    NPRIME                         EVEN, NOT PRIME             0028
    0023 5C 01 7F 74             28        MVC   TEST#(2,XR1),TWO(,XR1)                                     0029
    0027 5C 01 7B 78             29        MVC   END#+1(2,XR1),SENSE(,XR1)      DIVIDE INPUT BY TWO         0030
    002B 7C 00 79                30        MVI   END#-1(,XR1),0                  TO USE FOR END TESTING     0031
    002E 5E 02 7B 7B             31        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0032
    0032 5E 02 7B 7B             32        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0033
    0036 5E 02 7B 7B             33        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0034
    003A 5E 02 7B 7B             34        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0035
    003E 5E 02 7B 7B             35        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0036
    0042 5E 02 7B 7B             36        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0037
    0046 5E 02 7B 7B             37        ALC   END#+1(3,XR1),END#+1(,XR1)                                 0038
                                 38 *                                                                        0039
                                 39 *             MAIN TEST LOOP                                            0040
    004A 5E 01 7F 72             40 LOOPST ALC   TEST#(2,XR1),ONE(,XR1)         INCREMENT TEST              0041
    004E 50 01 7F 7A             41        CLC   TEST#(2,XR1),END#(,XR1)        TEST FOR COMPLETE           0042
    0052 F2 84 14                42        JH    PRIME#                         COMPLETE, CALL IT PRIME     0043
    0055 5C 01 7D 78             43        MVC   TEMPAR(2,XR1),SENSE(,XR1)      MAKE COPY AND               0044
    0059 5F 01 7D 7F             44 SUBTR  SLC   TEMPAR(2,XR1),TEST#(,XR1)       FIND REMAINDER             0045
    005D D0 84 59                45        BP    SUBTR(,XR1)                     BY SUBTRACTING             0046
    0060 D0 01 4A                46        BNZ   LOOPST(,XR1)                   REMAINDER NOT ZERO          0047
                                 47 *                                                                        0048
                                 48 *             NUMBER NOT PRIME                                          0049
    0063 F0 2F 3F                49 NPRIME HPL   X'3F',X'2F'                    NOT PRIME (NP) HALT         0050
    0066 D0 87 04                50        B     BEGIN(,XR1)                    GO BACK TO BEGINING         0051
                                 51 *                                                                        0052
                                 52 *             NUMBER IS PRIME                                           0053
    0069 F0 03 3F                53 PRIME# HPL   X'3E',X'03'                    IS PRIME (IP) HALT          0054
    006C D0 87 04                54        B     BEGIN(,XR1)                    GO BACK TO BEGINING         0055
```

Figure 34 (Part 1 of 2). Listing of Statements in Model 10 and Model 12 Basic Assembler Sample Program

```
$ASSPR  PRIME NUMBER TEST PROGRAM

ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT          VER 13.  MOD 00   01/30/76   PAGE  3

                                 56 *                                                          0C57
                                 57 *          DATA AREA                                       0058
    006F 0000              C070  58 ZERO   DC    IL2'0'                      BINARY ZERO       0059
    0071 0001              0072  59 ONE    DC    XL2'0001'                          CNE        0C6C
    0073 0002              0074  60 TWC    DC    BL2'00000010'                      TWO        0061
    0075 0003              0076  61 THREE  DC    AL2(3)                             THREF      CC62
    0077                   0078  62 SENSE  DS    CL2                                           0063
    0079                   007A  63 END#   DS    CL2                                           0064
    007B                   007B  64        DS    CL1                                           0065
    007C                   007D  65 TEMPAR DS    CL2                                           0066
    007E                   007F  66 TEST#  DS    CL2                                           0067
                           0001  67 XR1    EQU   1                           BASE REGISTER     0068
                           0000  68        END   $ASSPR                                        0C69

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY =     0
```

```
$ASSPR                          CROSS REFERENCE

SYMBOL  LEN VALUE DEFN   REFERENCES                         VER 13.  MOD 00   01/30/76   PAGE  4

$ASSPR  001  0000 0013   0068
BEGIN   003  0004 0016   0050  0054
END#    002  007A 0063   CC29* 0030* 0031  0031* 0032  0032* 0033  0033* 0034  0034* 0035  0035*
                         0036  0036* C037  CC37* 0041
LOOPST  004  004A 0040   0046
NPRIME  003  0063 C049   0027
ONE     002  0072 0059   0040
PREPAR  0C4  0016 0024   0019
PRIME#  003  0069 CC53   0025  0042
SENSE   002  0078 0062   0017* 0018  C024  0026  0029  0043
SUBTR   004  0059 0C44   0045
TEMPAR  002  0070 0065   C043* 0044*
TEST#   002  007F 0066   0028* 0040* 0041  0044
THREE   002  0076 0061   C024
TWO     002  0074 0060   0028
XR1     0C1  0001 0067   0014  0015* 0017  0018  0018  0024  0024  0026  0028  0028  0C29  0029
                         0030  0031  0031  CC32  0032  0033  0033  0034  0034  0035  0035  0036
                         0036  0037  CC37  CC40  0C4C  CC41  CC41  0043  0043  0044  0044  0045
                         0046  0050  0054
ZERO    002  0070 CC58   0018

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY =     0

OL105 I  THE CODE LENGTH OF $ASSPR IS   128 DECIMAL.
OL103 I  TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS    2
         NAME-$ASSPR,PACK-R1R1R1,UNIT-R1,RETAIN-T,LIBRARY-R,CATEGORY-000
```

Figure 34 (Part 2 of 2). Listing of Statements in Model 10 and Model 12 Basic Assembler Sample Program

## MODEL 15 SAMPLE PROGRAM

This section describes the sample program and explains the operating procedures necessary for executing it. General operating procedures for the Basic Assembler are found in the *IBM System/3 Model 15 Operator's Guide*, GC21-5075 and in Part II of this manual.

### Program Description

The sample program is called System Input Device List Program. The program reads records from the system input device and lists them on the system printer. Statements are read and listed until one of the delimiters (/*,/&, or /.) is encountered. If the delimiter is /*, another file can be listed under operator control.

There are three messages displayed by this program:

| *Message* | *Meaning* |
|---|---|
| EOF ON SYSIN | End of file encountered on the system input device. More files can be printed if the EOF condition is caused by /*. The operator replies P to print another file or C to cancel. |
| PRINTER ERROR | A permanent printer error has occurred. The program issues the message and then goes to end of job. (The message is displayed and then removed when end of job is reached. However, the message is in the system history area and may be displayed from there.) |
| SYSIN ERROR | A permanent system input device error has occurred. The program issues the message and then goes to end of job. (The message is displayed and then removed when end of job is reached. However, the message is in the system history area and may be displayed from there.) |

The sample program uses Model 15 macros and therefore the assembly step must be preceded by a macro processor step.

Figure 35 shows the OCL that assembles, link edits, and executes the sample program. Figure 36 shows the sample program statements.

IBM

IBM System/3 Basic Assembler Coding Form

```
// ASAMP JOB
// LOAD $MPXDV,F1
// FILE NAME-$SOURCE,RETAIN-T,UNIT-R2,PACK-R2R2R2,TRACKS-6
// COMPILE SOURCE-$ASSPL,UNIT-F1
// RUN



// LOAD $ASSEM,F1
// SWITCH 1XXXXXXX
// FILE NAME-$WORK,RETAIN-S,UNIT-R2,PACK-R2R2R2,TRACKS-2
// FILE NAME-$WORK2,RETAIN-S,UNIT-D1,PACK-D1D1D1,TRACKS-10
// FILE NAME-$SOURCE,RETAIN-S,UNIT-R2,PACK-R2R2R2
// COMPILE OBJECT-R1
```

IBM

IBM System/3 Basic Assembler Coding Form

```
// RUN
// LOAD $OLINK,F1
// FILE NAME-$WORK,RETAIN-S,UNIT-D1,PACK-D1D1D1,TRACKS-10
// FILE NAME-$SOURCE,RETAIN-S,UNIT-R2,PACK-R2R2R2,TRACKS-10
// RUN
// PHASE NAME-$ASSPO
// OPTIONS MAP-XREF
// INCLUDE NAME-$ASSPR,UNIT-R1
// INCLUDE NAME-$$LPRT,UNIT-F1
// END
// LOAD $ASSPO,F1
// RUN
```

*Notes:*

1. Specifies the program pack.
2. Name of the assembler sample program in the source library.
3. Library in which the output assembler object (R) module is stored.
4. Name given to the output assembler object (O) program.
5. Module name and object program name (R).
6. Specifies the system pack.

If the system configuration does not include the 5444 drive 2 or the 5445 drive 1, references in the OCL to R2 and D1 must be changed to specify devices available on the system.

**Figure 35. Model 15 Sample Program OCL**

```
OPTIONS NODECK                           OBJECT TO LIBRARY ONLY           00010000
THE LIST OF OPTIONS USED DURING THIS ASSEMBLY IS--   NODECK,LIST,XREF,REL,OBJ


$ASSPR                                 EXTERNAL SYMBOL LIST
SYMBOL   TYPE                                                       VER 01, MOD 00  11-09-73  PAGE   1
$ASSPR   MODULE
$$LPRT   EXTRN


$ASSPR
ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT               VER 01, MOD 00  11-09-73  PAGE   2
                                 1        ICTL  1,71                                                       00020000
                                 2        ISEQ  73,80                                                      00030000
                                 3        PRINT NOGEN,NODATA                                               00040000


$ASSPR SYSTEM INPUT DEVICE (SYSIN) LIST PROGRAM
ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT               VER 01, MOD 00  11-09-73  PAGE   3
                                 5 *    THIS PROGRAM READS A FILE FROM THE SYSTEM INPUT DEVICE AND LISTS   00060000
                                 6 *     IT ON THE PRINTER.                                                00070000
                                 7 *                                                                       00080000
                                 8 *    THERE ARE THREE MESSAGES ISSUED BY THIS PROGRAM:                   00090000
                                 9 *     MESSAGE           TYPE   MEANING                                  00100000
                                10 *     'EOF ON SYSIN'    WTOR  END OF FILE ENCOUNTERED ON SYSIN.          00110000
                                11 *                             MORE FILES MAY BE PRINTED IF THE          00120000
                                12 *                             EOF CONDITION IS CAUSED BY A '/*'.        00130000
                                13 *                             THE OPERATOR REPLYS TO THIS MESSAGE       00140000
                                14 *                             ARE 'P' TO PRINT ANOTHER FILE AND         00150000
                                15 *                             'C' TO CANCEL AND GO TO EOJ.              00160000
                                16 *     'PRINTER ERROR'   WTO   THERE HAS BEEN A PERMANENT PRINTER         00170000
                                17 *                             ERROR.  THE PROGRAM ISSUES THE            00180000
                                18 *                             MESSAGE AND GOES TO END OF JOB.           00190000
                                19 *     'SYSIN ERROR'     WTO   THERE HAS BEEN A PERMANENT SYSIN           00200000
                                20 *                             ERROR.  THE PROGRAM ISSUES THE            00210000
                                21 *                             MESSAGE AND GOES TO END OF JOB.           00220000

    4000                        23 $ASSPR  START X'4000'                                                   00240000
                           0001 24         EXTRN $$LPRT                  PRINTER DATA MANAGEMENT           00250000
                           408C 25         USING BASE,BRG                ESTABLISH A BASE REGISTER         00260000
    4000 C2 01 408C             26         LA    BASE,BRG                 FOR THE DATA AREAS               00270000

                                28 * PREPARE THE PRINTER FILE FOR USE                                      00290000
    4004 D2 02 07               29         LA    PRNDTF(,BRG),$DTF                                         00300000
                                30 *       $ALOC                         ALLOCATE PRINTER FILE             00310000
                                33 *       $OPEN                         OPEN PRINTER FILE                 00320000
    400F BC 01 13               36         MVI   $DFSPA(,$DTF),1         SET FOR SINGLE SPACE              00330000
    4012 BC 40 0F               37         MVI   $DFOPC(,$DTF),$OCPRT    SET OP-CODE TO PRINT              00340000
    4015 7C 01 00               38         MVI   SYSINL+$SRFCT(,BRG),$SRRDF SET SYSIN OP-CODE FOR 1ST BUFF 00350000

                                40 * PREPARE TO PRINT A NEW FILE                                           00370000
    4018 7C 01 17               41 FILES   MVI   PRNDTF+$DFSKB(,BRG),1   SET TO SKIP BEFORE FIRST PRINT    00380000

                                43 * READ FROM SYSIN AND PRINT UNTIL END OF FILE                           00400000
    401B D2 02 00               44 FILEL   LA    SYSINL(,BRG),SYS                                          00410000
                                45 *       $READ OPC-N                   READ FROM SYSIN                   00420000
    4022 BD 50 00               49         CLI   $SRFCT(,SYS),$SREOF     TEST FOR EOF    ('/*','/&','/.')  00430000
    4025 F2 81 30               50         JE    EOF                                                       00440000
    4028 BD 80 00               51         CLI   $SRFCT(,SYS),$SREOJ     TEST FOR EOJ    ('/&','/.')       00450000
    402B F2 81 53               52         JE    EOJ                                                       00460000
    402E BD 60 00               53         CLI   $SRFCT(,SYS),$SRERR     TEST FOR SYSIN ERROR              00470000
    4031 F2 81 3C               54         JE    SYSER                                                     00480000
    4034 BC 00 00               55         MVI   $SRFCT(,SYS),$SRRDD     SET FOR NEXT SYSIN READ           00490000
    4037 6C 01 14 04            56         MVC   PRNDTF+$DFLRA(2,BRG),$SRBFZ(,SYS) POINT TO CURRENT RECORD 00500000
    403B D2 02 07               57         LA    PRNDTF(,BRG),$DTF                                         00510000
                                58 *       $PUTP DEV-1403                PRINT THE CURRENT RECORD          00520000
    4042 BD 41 0E               60         CLI   $DFCMP(,$DTF),$CPPER    TEST FOR PRINTER ERROR            00530000
    4045 F2 81 32               61         JE    PRNERR                                                    00540000
    4048 BC 00 10               62         MVI   $DFSKB(,$DTF),0         SET FOR NO SKIP BEFORE            00550000
    404B BD 48 0E               63         CLI   $DFCMP(,$DTF),$CPOVF    TEST FOR PAGE OVERFLOW            00560000
    404E F2 01 03               64         JNE   NOSKIP                                                    00570000
    4051 BC 01 10               65         MVI   $DFSKB(,$DTF),1         SET FOR SKIP TO LINE ONE          00580000
    4054 C0 87 401B             66 NOSKIP  B     FILEL                                                     00590000
```

Figure 36 (Part 1 of 4). Listing of Statements in Model 15 Basic Assembler Sample Program.

```
$ASSPR SYSTEM INPUT DEVICE (SYSIN) LIST PROGRAM

ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT           VER 01, MOD 00  11-09-73  PAGE   4

                                  68 * END OF FILE ON SYSIN                                             00610000
    4058 D2 02 28                 69 EOF     LA    EOFMSG(,BRG),LOG                                     00620000
                                  70 *       $LOG                             WTOR EOF MESSAGE          00630000
    405F 7D C3 37                 74         CLI   REPLY(,BRG),C'C'           OPERATOR SAY CANCEL       00640000
    4062 F2 81 1C                 75         JE    EOJ                                                  00650000
    4065 7D D7 37                 76         CLI   REPLY(,BRG),C'P'           OPERATOR SAY PRINT ANOTHER 00660000
    4068 C0 81 4018               77         BE    FILES                                                00670000
    406C C0 87 4058               78         B     EOF                        INVALID REPLY, TRY AGAIN  00680000

                                  80 * ERROR ON SYSIN                                                   00700000
    4070 D2 02 38                 81 SYSER   LA    SERMSG(,BRG),LOG                                     00710000
                                  82 *       $LOG                             WTO SYSIN ERROR MESSAGE   00720000
    4077 F2 87 07                 86         J     EOJ                        GO TO EOJ                 00730000

                                  88 * ERROR ON PRINTER                                                 00750000
    407A D2 02 44                 89 PRNERR  LA    PERMSG(,BRG),LOG                                     00760000
                                  90 *       $LOG                             WTO PRINTER ERROR MESSAGE 00770000

                                  95 * END OF JOB ROUTINE                                               00790000
                             4081 96 EOJ     EQU   *                                                    00800000
    4081 D2 02 07                 97         LA    PRNDTF(,BRG),$DTF                                    00810000
                                  98 *       $CLOS                            CLOSE PRINTER FILE        00820000
                                 101 *       $EOJ                             GO TO EOJ                 00830000
```

```
$ASSPR SYSTEM INPUT DEVICE (SYSIN) LIST PROGRAM

ERR LOC  OBJECT CODE       ADDR STMT SOURCE STATEMENT           VER 01, MOD 00  11-09-73  PAGE   5

                                 105 * CONSTANTS AND DATA AREAS                                         00850000
                             408C 106 BASE    EQU   *                          BASE REGISTER ADDRESS    00860000

                                 108 * SYSIN TABLES                                                     00880000
                                 109 *YSINL  $RLST BUF1-BUFFR1,BUF2-BUFFR2,  SYSIN PARAMETER LIST      X00890000
                                 110 *             WORK-WORKAR                                          00900000
                                 116 *       $RLSD                            SYSIN EQUATES             00910000

                                 133 * PRINT FILE TABLES                                                00930000
                                 134 *RNDTF  $DTFP DEV-1403,READ-0,IOBA-PRNIOB, PRINT FILE DTF         X00940000
                                 135 *             IOAA-PRNBUF,RECL-96,                                X00950000
                                 136 *             OVFL-60,PAGE-66                                      00960000
                                 160 *       $DTFO D1403-Y                    PRINTER DTF DISPLACEMENTS 00970000

                                 223 * SYSTEM LOG TABLES                                                00990000
                                 224 *OFMSG  $LWTO COMP-AS,HALT-AM,SUBH-PG,TLEN-12, SYSIN EOF WTOR     X01000000
                                 225 *             TADR-EOFMGC,REPLY-Y,RLEN-1,RADR-REPLY                01010000
    40C3 E7                  40C3 238 REPLY   DC    CL1'X'                     WTOR REPLY               01020000
                                 239 *ERMSG  $LWTO COMP-AS,HALT-AM,SUBH-PG,TLEN-11, SYSIN ERROR WTO    X01030000
                                 240 *             TADR-SERMGC                                          01040000
                                 251 *ERMSG  $LWTO COMP-AS,HALT-AM,SUBH-PG,TLEN-13, PRINTER ERROR WTO  X01050000
                                 252 *             TADR-PERMGC                                          01060000
                             40DC 263 EOFMGC  EQU   *                                                   01070000
    40DC C5D6C640D6D540E2 40E7 264         DC    CL12'EOF ON SYSIN'                                     01080000
                             40E8 265 SERMGC  EQU   *                                                   01090000
    40E8 E2E8E2C9D540C5D9 40F2 266         DC    CL11'SYSIN ERROR'                                      01100000
                             40F3 267 PERMGC  EQU   *                                                   01110000
    40F3 D7D9C9D5E3C5D940 40FF 268         DC    CL13'PRINTER ERROR'                                    01120000

                                 270 * SYSIN BUFFER AND WORK AREAS                                      01140000
    4100                          271         ORG   *,128                      ORG TO REQUIRED BOUNDARY 01150000
                             4100 272 BUFFR1  EQU   *                          BUFFER ONE               01160000
    4100 0000000000000000 417F 273         DC    XL128'0'                                               01170000
                             4180 274 BUFFR2  EQU   *                          BUFFER TWO               01180000
    4180 0000000000000000 41FF 275         DC    XL128'0'                                               01190000
                             4200 276 WORKAR  EQU   *                          WORK AREA                01200000
    4200 0000000000000000 422E 277         DC    XL47'0'                                                01210000

                                 279 * PRINTER BUFFER AND WORK AREAS                                    01230000
    427C                          280         ORG   *,256,X'7C'                ORG TO REQUIRED BOUNDARY 01240000
                             427C 281 PRNBUF  EQU   *                          PRINTER BUFFER           01250000
    427C 4040404040404040 4305 282         DC    CL138' '                                               01260000
                             4306 283 PRNIOB  EQU   *                          PRINTER IOB              01270000
    4306 0000000000000000 4337 284         DC    XL50'0'                                                01280000

                                 286 * REGISTER LABELS                                                  01300000
                             0001 287 BRG     EQU   1                          BASE REGISTER            01310000
                             0002 288 SYS     EQU   2                          SYSIN PARAMETER LIST POINTER  01320000
                             0002 289 LOG     EQU   2                          SYSLOG PARAMETER LIST POINTER 01330000
                             4000 290         END   $ASSPR                                              01340000

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY--     0

TOTAL SEQUENCE ERRORS IN THIS ASSEMBLY--         0
```

Figure 36 (Part 2 of 4). Listing of Statements in Model 15 Basic Assembler Sample Program.

```
$ASSPR                          CROSS REFERENCE

SYMBOL  LEN VALUE DEFN    REFERENCES                                   VER 01, MOD 00  11-09-73  PAGE   5

$$LPRT  001  0001 0024    0059
$ASSPR  001  4000 0023    0290
$A1COI  001  0010 0193
$A1DAT  001  0001 0198
$A1H56  001  0002 0196
$A1INT  001  0004 0195
$A1MFM  001  0008 0194
$A1PCH  001  0020 0192
$A1PRT  001  0040 0191
$A1PR2  001  0001 0197
$A1RD   001  0080 0190
$A2ALL  001  0040 0203
$A2AMP  001  0004 0208
$A2EOF  001  0008 0206
$A2HUC  001  0002 0207
$A2IND  001  0080 0202
$A2MBF  001  0010 0205
$A2OPN  001  0001 0209
$A2SIN  001  0020 0204
$CPCND  001  0010 0214
$CPEOF  001  0042 0217
$CPOVF  001  0048 0213    0063
$CPPER  001  0041 0216    0060
$CPSUC  001  0040 0215
$DFARR  001  0009 0168
$DFAT1  001  0002 0164
$DFAT2  001  0003 0165
$DFCHA  001  0005 0166
$DFCHB  001  0007 0167
$DFCMP  001  000E 0171    0060  0063
$DFDEV  001  0000 0162
$DFLP   001  0010 0183
$DFLRA  001  000D 0170    0056*
$DFMSK  001  001F 0185
$DFOPC  001  000F 0172    0037*
$DFOVF  001  001C 0182
$DFPGS  001  0020 0186
$DFPIB  001  0017 0179
$DFPIO  001  0019 0180
$DFPOS  001  001E 0184
$DFPQ   001  0014 0177
$DFPR   001  0015 0178
$DFPRL  001  0018 0181
$DFSKA  001  0012 0175
$DFSKB  001  0010 0173    0041* 0062* 0065*
$DFSPA  001  0013 0176    0036*
$DFSPB  001  0011 0174
$DFUPS  001  0001 0163
$DFXRS  001  0008 0169
$DTF    001  0002 0161    0029* 0036  0037  0057* 0060  0062  0063  0065  0097*
$OCPRT  001  0040 0221    0037
$SRBF1  001  0002 0118
$SRBF2  001  0004 0119    0056
$SREOF  001  0050 0129    0049
$SREOJ  001  0080 0131    0051
$SRERR  001  0060 0130    0053
```

Figure 36 (Part 3 of 4). Listing of Statements in Model 15 Basic Assembler Sample Program.

```
$ASSPR                               CROSS REFERENCE

SYMBOL   LEN VALUE DEFN   REFERENCES                                   VER 01, MOD 00  11-09-73  PAGE    7

$SRFCT   001  0000 0117   0038* 0049  0051  0053  0055*
$SRNOM   001  0040 0128
$SRRD    001  0009 0126
$SRRDD   001  0000 0123   0055
$SRRDF   001  0001 0124   0038
$SRRDL   001  0002 0125
$SRWRK   001  0006 0120
BASE     001  408C 0106   0025  0026
BRG      001  0001 0287   0025  0026* 0029  0038  0041  0044  0056  0057  0067  0074  0076  0081
                          0089  0097
BUFFR1   001  4100 0272   0113
BUFFR2   001  4180 0274   0114
EOF      003  4058 0069   0050  0078
EOFMGC   001  400C 0263   0235
EOFMSG   001  4084 0227   0069
EOJ      001  4081 0096   0052  0075  0086
FILEL    003  4018 0044   0066
FILES    003  4018 0041   0077
LOG      001  0002 0289   0069* 0081* 0089*
NOSKIP   004  4054 0066   0064
PERMGC   001  40F3 0267   0262
PERMSG   001  4000 0254   0089
PRNBUF   001  427C 0281   0153
PRNDTF   001  4093 0137   0029  0041* 0056* 0057  0097
PRNERR   003  407A 0089   0061
PRNIOB   001  4306 0283   0152
REPLY    001  40C3 0238   0074  0076  0237
SERMGC   001  40E8 0265   0250
SERMSG   001  40C4 0242   0081
SYS      001  0002 0288   0044* 0049  0051  0053  0055  0056
SYSER    003  4070 0081   0054
SYSINL   001  408C 0111   0038* 0044
WORKAR   001  4200 0276   0115

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY--        0

TOTAL SEQUENCE ERRORS IN THIS ASSEMBLY--            0


OL105 I   THE CODE LENGTH OF $ASSPR IS    824 DECIMAL.
OL103 I   TOTAL NUMBER OF LIBRARY SECTORS REQUIRED IS     5
          NAME-$ASSPR,PACK-R1R1R1,UNIT-R1,RETAIN-T,LIBRARY-R,CATEGORY-000
```

Figure 36 (Part 4 of 4). Listing of Statements in Model 15 Basic Assembler Sample Program.

# Appendix G: IBM 1255 Magnetic Character Reader Support (Models 12 and 15 Only)

Support is provided by the following IBM-supplied subroutines:

- SUBR07 – 1255 (Model 15 only)
- SUBR08 – 1255 (Model 12 and Model 15)
- SUBR09 – 1419 (Model 12 and Model 15)

For detailed information concerning this support, see the *IBM System/3 Models 12 and 15 1255 and 1419 Magnetic Character Reader Reference and Program Logic Manual*, GC21-5132.

# Index

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/3 Basic Assembler Reference (File No. S3-21) Printed in U.S.A. SC21-7509-7

SC21-7509-7