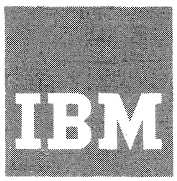
File No. S360-21
Form C28-6555-1



Systems Reference Library

IBM System/360 Basic Programming Support

Basic Assembler

Program Logic Manual

This publication provides detailed information on the internal logic of the IBM System/360 Basic Programming Support Basic Assembler. It is intended for technical personnel who are responsible for analyzing program operation, diagnosing malfunctions, and/or adapting the program to special usage.

PREFACE

This manual is designed to give the reader a thorough understanding of the functioning of the Basic Programming Support Basic Assembler Program.

Effective use of this publication is based on an understanding of the IBM System/360 machine operations and the System/360 Basic Programming Support Basic Assembler Language. This information can be found in the following IBM System/360 publications:

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

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

The first section of this manual presents the overall purpose and description of the Basic Programming Support Basic Assembler program. A breakdown of the Basic Programming Support Basic Assembler into its major components is also provided in this first section. The remainder of the manual is devoted to more detailed descriptions and flowcharts of the major components, down to the function block level. If more detail is needed than that provided by the function block description, the reader is referred to the Basic Programming Support Basic Assembler Program listing.

MAJOR REVISION (June, 1965)

This publication is a major revision of the previous edition, Form C28-6555-0, which is now obsolete. Significant changes have been made throughout this publication. The new edition should be reviewed in its entirety.

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

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

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Program Logic Documentation, Department D89, PO Box 390, Poughkeepsie, N.Y. 12602

ILLUSTRATIONS

FIGURES

PURPOSE OF PROGRAM

The Basic Programming Support Basic Assembler Program is designed to translate a source program written in symbolic language into an object program in machine language. Each phase of the program must be assembled on an IBM System/360 with main storage of greater than 8,192 bytes.

PROGRAM ORGANIZATION

The assembler is a two-phase program, i.e., two-storage-load program. Phase I is the first storage load, and Phase II is the second storage load. Only the phase currently being executed occupies an area in main storage. The program is designed in this manner so that the maximum amount of main storage is available as working area for each phase (main storage not occupied by the assembler, tables, etc.).

The Basic Programming Support Basic Assembler Program has the same functions as any assembly program. It accomplishes the following:

1. Replaces each symbolic operation code with its machine-language equivalent,
2. Replaces each symbolic operand with an actual address,
3. Reserves an area of main storage for each instruction and data area.

Phase I accomplishes the following major functions:

1. Replaces the symbolic operation code of each source statement with its machine language equivalent through the use of the Operation Code Table. (See "Phase I," Section 2, "Tables.") This information is placed in a buffer, the Intermediate Text Buffer, to be translated by Phase II. (See "Phase I," Section 2, "Tables.")
2. Builds a table of all symbolic operands in the source statements. This table is called the Symbol Table. (See "Phase I," Section 2, "Tables.") The location of the symbolic operand within the Symbol Table, along with all self-defining operands, is also placed in the Intermediate Text.
3. Allocates storage to source statements and to data areas with an internal counter called the location counter.

Phase II of the Assembler Program accomplishes the following major functions:

1. Translates the Intermediate Text into machine language. During this translation the symbolic operands are replaced with the actual addresses of symbols from the Symbol Table.
2. Produces the object program.
3. Produces a program listing.

The tables, cards, records, and the translation required to accomplish the functions listed above are described in detail in the discussion of each phase.

OVERALL OPERATION OF PROGRAM

A generalized logic flow of the two-phase assembly program is shown on Charts 01 and 02. A detailed description of each phase is given in the "Phase I" and "Phase II" sections of this publication. In the following text, the flowchart block numbers being described are indicated in parentheses.

Phase I (Chart 01)

At the start of the assembly, the Phase I assembly deck or tape is loaded into lower main storage (02).

If the source program is on cards and is being reassembled, the symbol table that was constructed and punched into cards from a previous assembly is read into main storage (03).

The symbolic language source program, on cards or in the form of card images on tape, follows Phase I of the assembly program and the symbol table (if present). Each source statement from card or tape is read and processed, and the necessary information is punched into the same card, or written onto tape, or punched into a new card before the next source statement is read. If this is a reassembly, the new information is compared to the old information before any action occurs. The type of input depends upon the system configuration.

The first test performed on a source statement is to check for a symbol in the name field (05). If the statement has a symbol in the name field, it is noted (06). The symbol is not placed in the Symbol Table at this time because the correct processing of the symbol will be determined by the operation code of the statement. Symbols in the name field of some Assembler instructions will not be used by the assembler and, therefore, will not be placed in the Symbol Table.

If the name field does not contain a symbol, but the statement begins with an asterisk (07), indicating a comments statement, the only action taken is to indicate this in the Intermediate Text (08).

If the statement is blank (09), it is ignored, and the next source statement is read in (04). If it is not blank, the Operation Code Table that was loaded into main storage as part of the assembler deck is searched for the corresponding machine language operation code (10). The machine language operation code is then stored in the Intermediate Text. If the operation code is not found, or if other errors in any part of the source program statement are found by the assembler, the necessary error flag(s) is placed in the Intermediate Text.

The operand field of the source statement is scanned next (11). Formats are checked, and the expressions in the operand field are translated according to format and placed in the Intermediate Text. (See "Phase I," Section 5, "Operand Field Translation.")

All the information concerning the source statement that can be determined by Phase I is stored in the Intermediate Text, and the Intermediate Text is then punched into a card or written on tape, depending upon the system configuration (13).

The next source statement is then read and processed. Source statements are read and processed until the END statement is encountered (14). When the END statement is encountered, the Symbol Table is punched into blank cards for a card system (15), the value (attribute) section of the Symbol Table is relocated to upper main storage, and the message 1EA is printed. The assembly program is then ready to read Phase II into main storage. However, if the assembler was loaded from tape, the message 1EA is not printed, and Phase II is loaded immediately.

Phase II (Chart 02)

Phase II of the Assembler Program is read into main storage. If the Symbol Table deck is in the card reader, it is read into the area of main storage reserved for the Symbol Table (03).

The first statement containing the Intermediate Text is then read and checked. The output buffer area is checked to determine if enough information has been translated to produce an object program card (07). If enough information has been translated, an object program card is produced (08) before the next statement is read (05).

The Intermediate Text is translated according to the type and format of the input source statement.

An Assembler instruction (09) is translated according to type (14). Each instruction has different requirements and requires a different routine to translate it. (See "Phase II," Section 5, "Intermediate Text Translation.")

A machine instruction (10) must be translated according to format (15) since there are five different formats (RR, RX, RS, SI, SS) for machine instructions. (See "Phase II," Section 5, "Intermediate Text Translation.")

If an error was encountered during Phase I requiring that the instruction be assembled as zeros (11), zeros are stored in the output buffer (16).

A statement that contains comments, or an error that will not allow the instruction to be translated, will cause only the source statement to be printed in the object program listing (12).

The Intermediate Text is read and translated until the END statement is encountered (12.5). When the END statement is encountered, the following conditions occur:

1. The remaining information in the output area is produced in object program form.
2. RLD cards, if any, are produced.
3. If object code is being written on tape, a LDT record is written immediately after the END record and the tape is backspaced. If it is stacked output, only the final LDT record remains, the others being overlaid by the first record of each succeeding job.
4. The message 2EA is printed and the program stops.

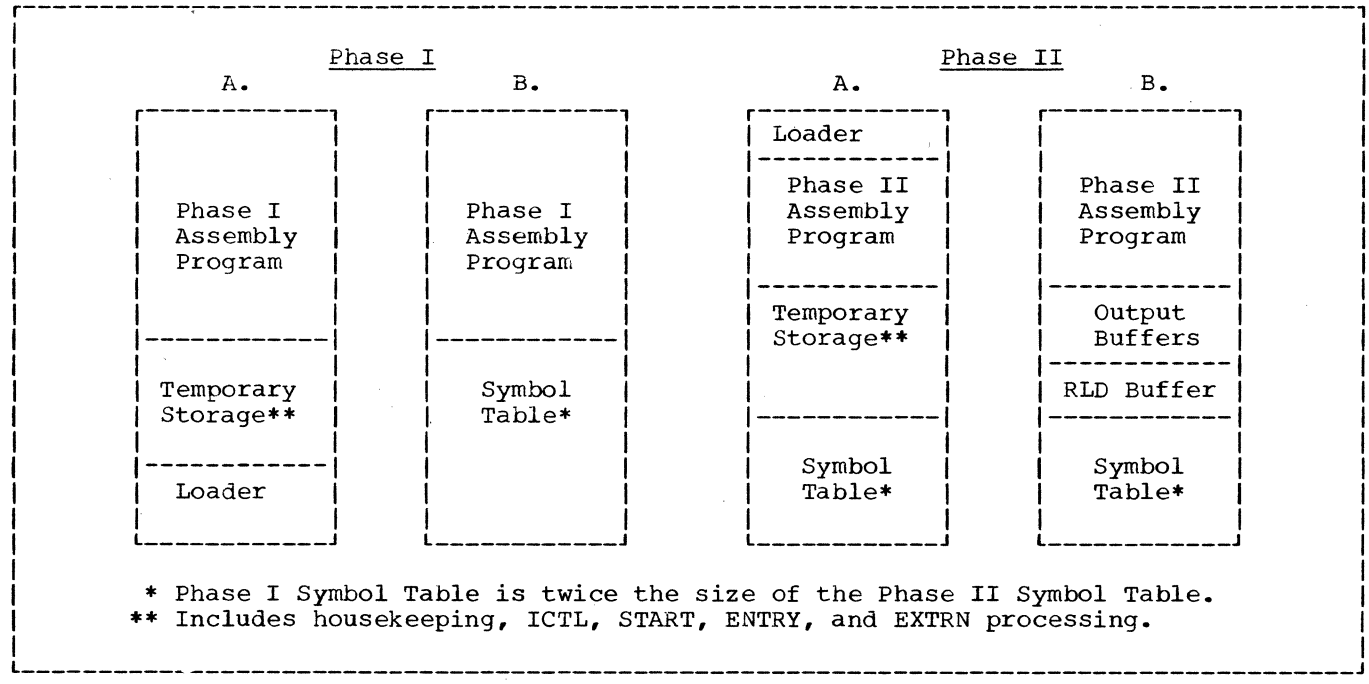


Figure 1. Storage Allocation

STORAGE ALLOCATION

Figure 1 illustrates the relative locations in main storage of Phase I and Phase II of the Assembler Program. Figure 1, part A, shows the allocation at load time, and Figure 1, part B, shows the allocation after the housekeeping and the ICTL, START, ENTRY, and EXTRN processing are completed.

I/O FLOW

Figure 2 illustrates the I/O flow of the Assembler Program.

The Phase I deck is read from the card reader or input tape unit. The Symbol Table deck will then precede the source deck if this is a reassembly procedure as described in the publication, IBM System/360 Basic Programming Support Basic Assembler Language, Form C28-6503.

After the Phase I Assembler deck is read into main storage, and the Symbol Table deck if present, each Statement of the source program is read and partially translated. The partially translated source statement (Intermediate Text) is punched into the source statement card (1442), punched into a new card along with the source statement (2540), or placed on tape (1442 or 2540 with tape).

The Phase II deck is then read from the card reader or tape unit. The Symbol Table deck will then be read into main storage if this is a deferred assembly, as described in the publication, IBM System/360 Basic Programming Support Basic Assembler Language, Form C28-6503.

The Intermediate Text developed by Phase I is then read one card or record at a time and translated, the object program is punched into cards or placed on tape, and the object program listing is produced.

Figure 3 shows the buffer areas for the card-only option and Figure 4 shows the buffer areas for the tape option.

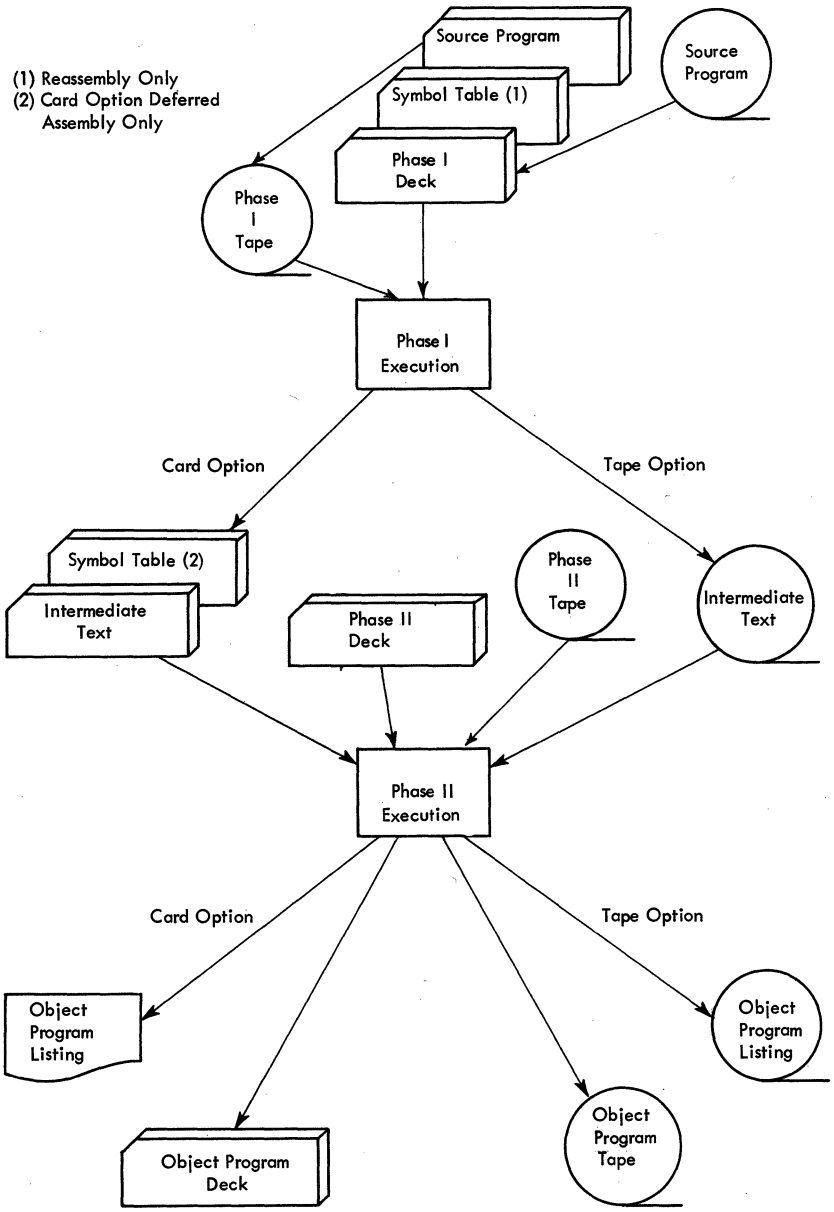


Figure 2. I/O Flow

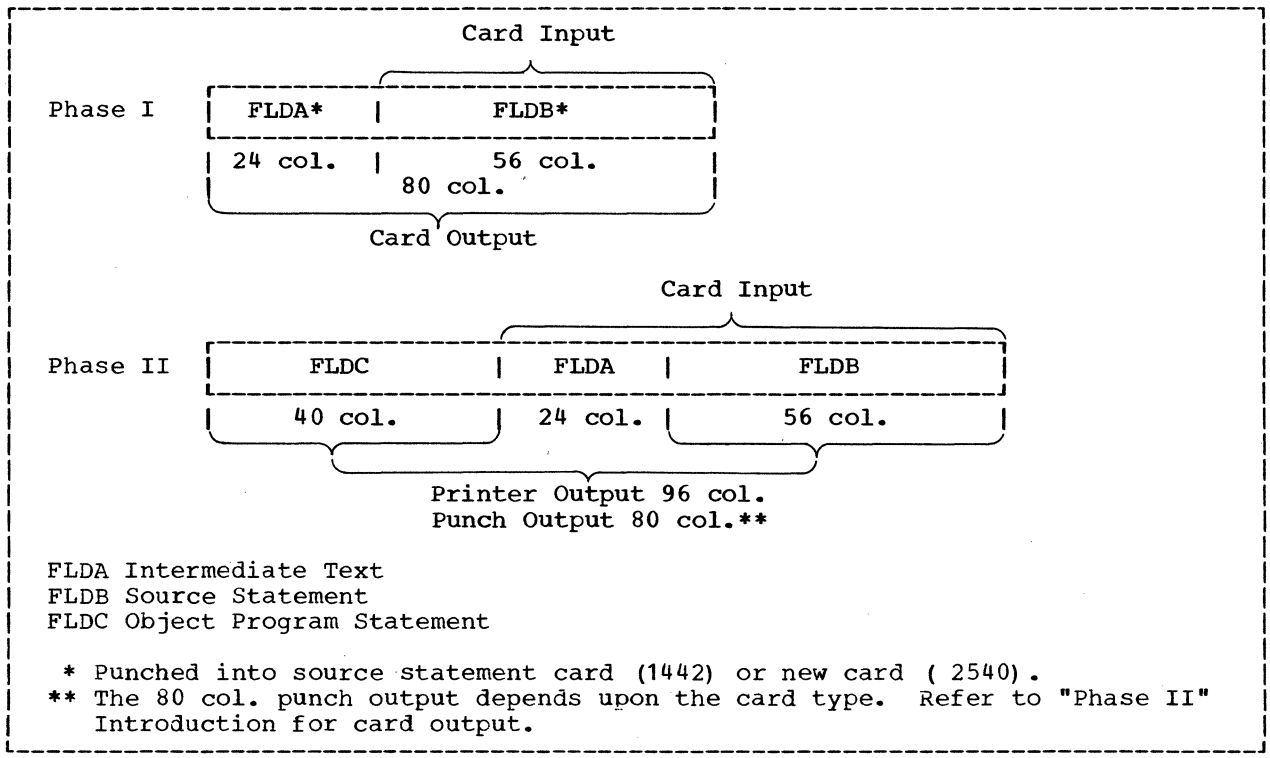


Figure 3. Buffer Areas for Card Option

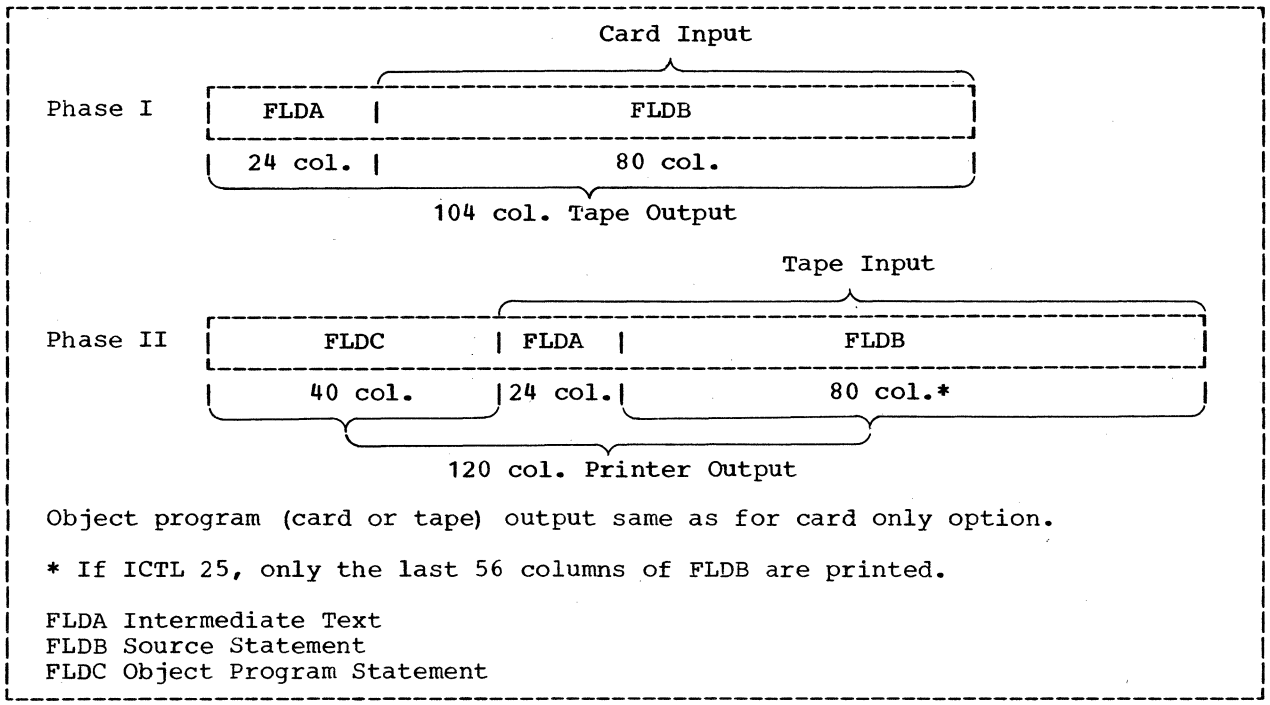


Figure 4. Buffer Areas for Tape Option

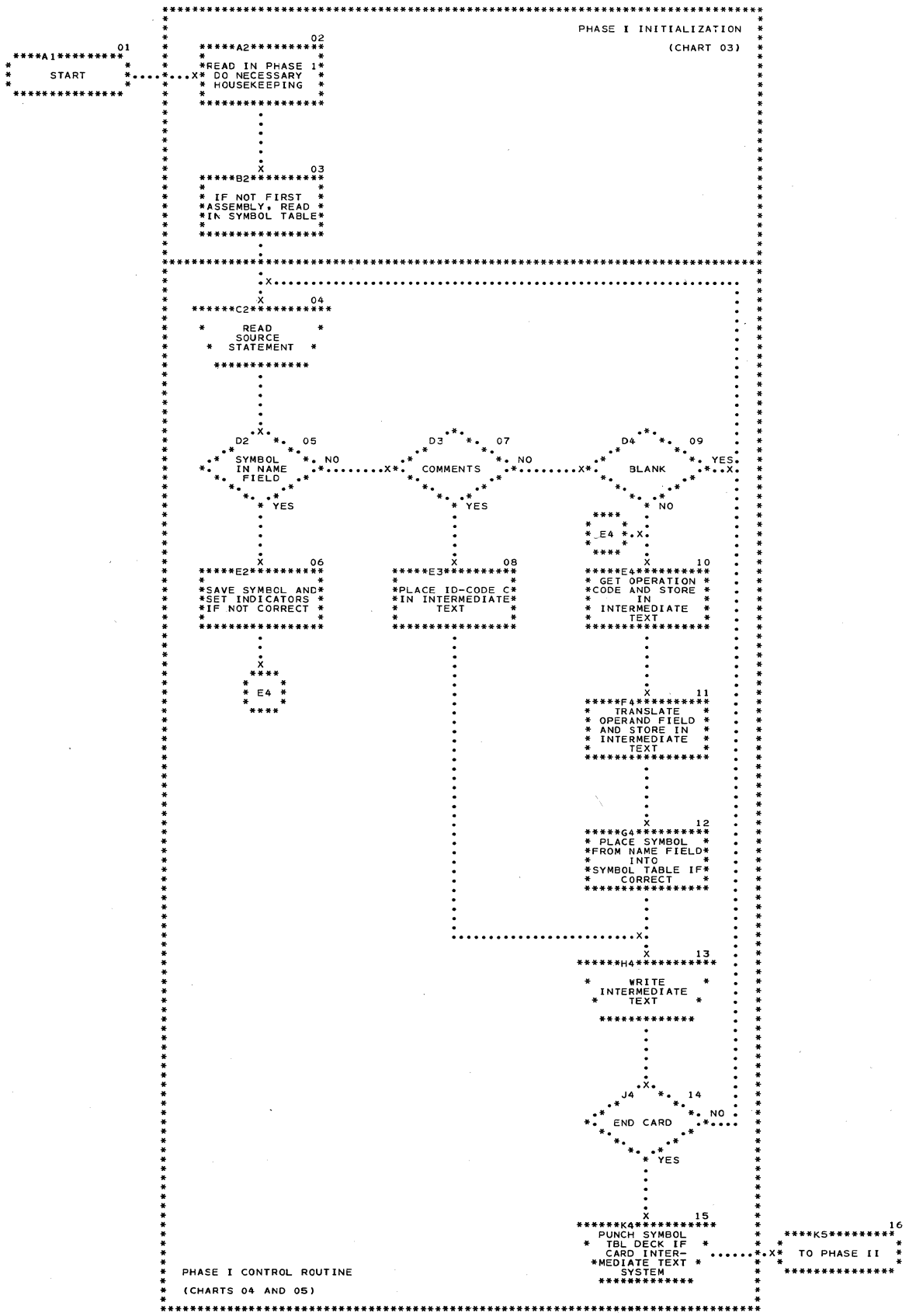
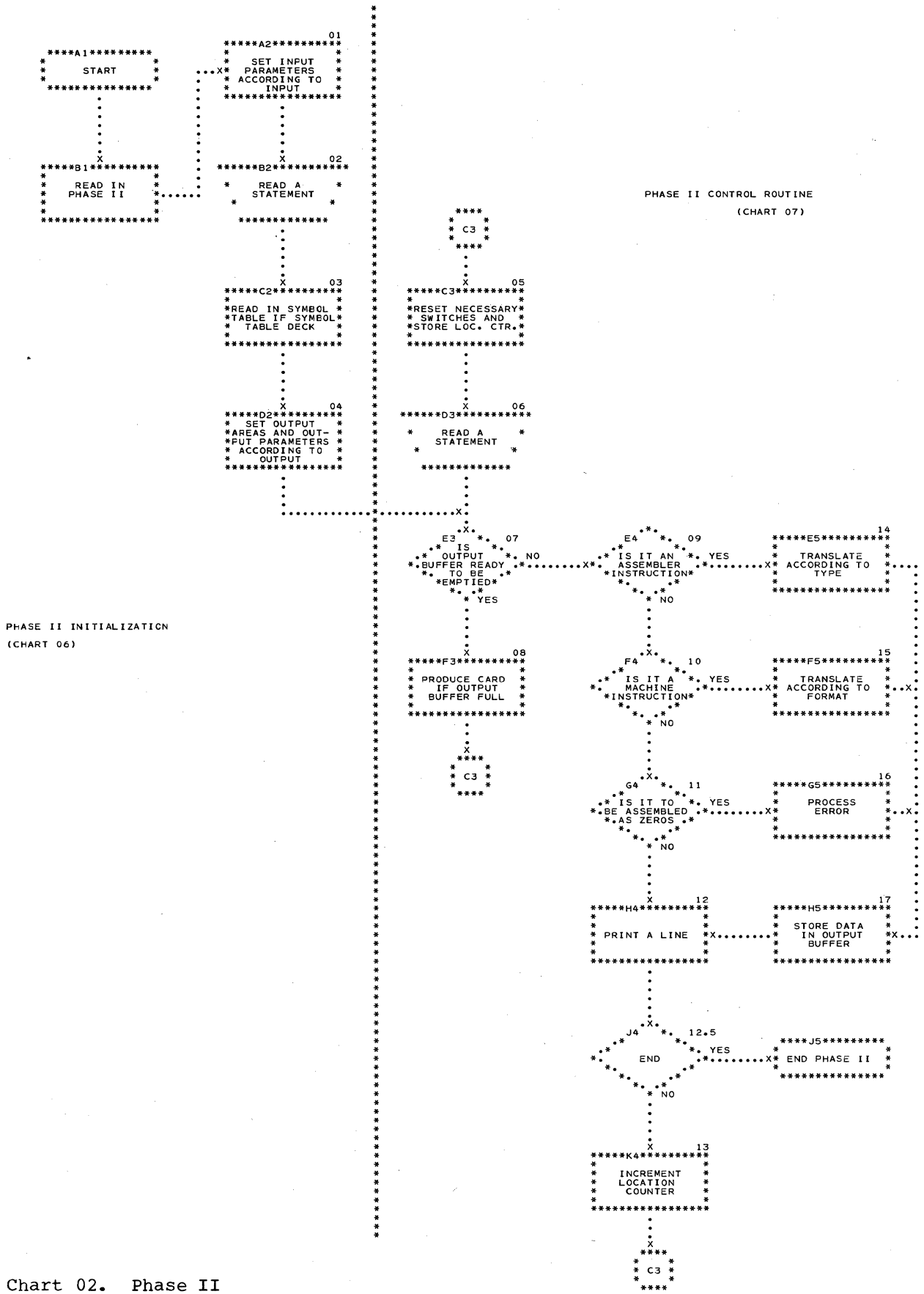


Chart 01. Phase I



PHASE II INITIALIZATION
(CHART 06)

PHASE II CONTROL ROUTINE
(CHART 07)

Chart 02. Phase II

SECTION 1: INTRODUCTION

The major functions of Phase I of the Assembler Program are as follows:

Replace the Symbolic Operation Code with Its Machine Language Equivalent

The symbolic operation code of each assembler language source statement is replaced with its machine language equivalent through the use of a table called the Operation Code Table. This table contains each assembler language symbolic operation code and its equivalent machine language operation code. The Operation Code Table is discussed in detail in Section 2, "Tables."

The Assembler Program reads the symbolic operation code in each source program statement, looks it up in this table, and picks up the corresponding machine operation code. The machine operation code and other pertinent information concerning the source statement are stored in a buffer called an Intermediate Text buffer. The Intermediate Text is discussed in detail in Section 2, "Tables."

Translate Symbolic Operands into Intermediate Text

The symbolic operands of each assembler language source statement are partially translated by Phase I and placed into the Intermediate Text. The symbolic operands of statements refer to the symbolic names of data fields, or symbolic names of other source statements. In order to determine the actual address for a symbolic operand, the actual address of all symbols must first be determined. This is why the final translation of the source statement is accomplished by Phase II.

The actual address of all symbols is determined by building a table during the execution of Phase I. This table, called the Symbol Table, consists of each symbolic name and its associated actual address, and other pertinent data. The Symbol Table is discussed in detail in Section 2, "Tables." The Symbol Table is punched into cards at the completion of Phase I when the card intermediate text system is being used. The Symbol Table remains in main storage when intermediate text is on tape.

If Phase II of the Assembler Program is not executed immediately following Phase I, the Symbol Table cards must be placed in front of the source deck when Phase II is to be executed. When a tape system is being used, Phase II must immediately follow Phase I. If Phase II is executed immediately following Phase I, the Symbol Table remains in main storage, and the Symbol Table cards do not have to be loaded.

During the execution of Phase II, the Symbol Table entry corresponding to the symbolic operand is found, and the value of that symbol is used to generate the object program statement.

Allocate Storage for Intermediate Text

A counter internal to the program is used to allocate storage for the intermediate text compiled from the source statements. This counter, called the location counter, is initialized with the machine address at which the program is to begin by using the START Assembler instruction.

When the length of a translated source statement has been determined, the counter is incremented by this length (after boundary alignment, if necessary), and the next instruction or data area will begin at the new location specified by the location counter.

When a statement is encountered that specifies that an area of storage must be reserved for data, the location counter is incremented by the number of storage bytes specified for that area.

SECTION 2: TABLES

INTERMEDIATE TEXT

The Intermediate Text associated with each statement contains a total of 24 bytes of information. The specific meaning of the 24 bytes of information and the bit patterns associated with the bytes are shown in a figure accompanying the description of each instruction type in Section 5, "Operand Field Translation." A general discussion of the bytes in the Intermediate Text is given in this section.

Figure 5 is a summary of the Intermediate Text associated with each instruction type.

The breakdown of the Intermediate Text is as follows:

Byte 1 - ID-Code

This field is one byte in length and contains an ID-Code which indicates whether the source statement is a machine instruction or an Assembler instruction, whether the instruction contains an error, and whether or not the statement requires further processing by Phase II. The possible ID-Codes and their meanings are as follows:

Code	Meaning
A	Machine instruction, no errors
B	Assembler instruction, no errors
C	Comments*
J	Error in a machine instruction statement
K	Error in an Assembler instruction statement
L	Error, statement not assembled*
M	Error, operand field assembled as zeros

* Error flags printed but no further translation by Phase II

After it is determined whether the operation code is a machine instruction, an Assembler instruction, or a comments statement, an ID-Code of A, B, or C, respectively, is placed into byte 1 of the Intermediate Text. If an error is encountered later during the translation of the source statement, these ID-Codes will be replaced with the appropriate error ID-Code (i.e., J,K,L, or M).

Byte 2 - Operation Code

The machine-language operation code is placed in this field after the search of the Operation Code Table. This field will be blank if the source statement contains an illegal operation code.

Bytes 3-22 - Operand Field

These bytes contain the text compiled from the operand field of the instruction. Since these bytes will contain different information for each instruction type, they are explained in detail in the discussion of each instruction in Section 5, "Operand Field Translation." Bytes 21 and 22 may contain error flags if the error ID-Code is L or M.

Bytes 21-24 - Error Flags

These bytes will contain error flags to indicate the type of error encountered during the translation of the instruction. If no error is encountered, the bytes will be blank. The error flags and their meanings are shown in Phase I, Section 3, "Subroutine Description," under the discussion of subroutine ERR.

OPERATION CODE TABLE

The Operation Code Table is divided into five sections, each section containing all the symbolic codes of the same length (e.g., all one-character symbolic codes in one section, all two-character symbolic codes in another section, etc.) plus the equivalent machine-language code. All symbolic codes within each section are also grouped according to the operand field format (e.g., all machine instructions in the SI Format, SS Format, etc.)

Instruction \ Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
RR	A	OP	R2		R1																			
RX	A	OP	X2		R1		D2						B2											
RS	A	OP	R3		R1		D2						B2											
SI	A	OP	I				D1						B1											
SS	A	OP	L2/L		L1		D1						B1				D2					B2		
CNOP	B	E3		ba	d1		d2																	
ICTL	B	ED	d																					
EJECT	B	EB																						
SPACE	B	E7	d																					
ORG	B	E8									rx													
END	B	EF									rx or blank													
EQU	B	E5									ax or rx													
USING	B	E6									rx							sx						
DROP	B	EC																sx						
EXTRN	B	E9				rs					as													
ENTRY	B	EA				rs					as													
START	B	EE		ba		aa					as													
CCW	B	E4	sx								rx							sx				sx		
DS	B	E0	l	ba	d																			
DC (C,D,E,F,H,X)	B	E1	l	ba	m								dc											
DC (A)	B	E2	l	ba	m								ax or rx											
COMMENTS	C																							
Err.Mach.Instr.	J	OP									same as Type A												e.f.	
Err.Assem.Inst.	K	OP									same as Type B												e.f.	
ERR (1)	L																					error flags		
ERR (2)	M	OP	n																			error flags		

Legend

aa = assembled address
as = actual symbol
ax = absolute expression
ba = boundary alignment bits
d = decimal integer
dc = defined constant
e.f. = error flags
l = length of constant
m = duplication factor
n = number of bytes of zeros to be assembled
rs = relocatable symbol
rx = relocatable expression
sx = simple expression

(1) statement not assembled
(2) operand assembled as zeros

Figure 5. Intermediate Text Summary

The organization of this table allows a faster assembly because it is not necessary to search the entire table to find the equivalent machine-language code. Only the section of the table containing the same number of characters as the symbolic code is searched. In addition, the table provides the Assembler Program with the operand field format and the address of the subroutine necessary to translate that format.

The machine-language operation code obtained from the table is placed into the Intermediate Text.

The Operation Code Table is loaded into main storage as part of Phase I.

SYMBOL TABLE

The Symbol Table is divided into two sections. The first section contains the compressed symbols from the source statements; the second section contains the attributes (values) associated with each symbol. Only the attributes are passed on to Phase II because there is no further need of the symbol itself once it has been replaced with the address of the location within the Symbol Table that contains the symbol attributes.

The first two entries in the Symbol Table refer to the location counter. The first entry contains the current setting of the location counter. This entry is necessary because the current setting of the location counter can be referred to within the symbolic program with an asterisk. For example, the expression (*+12) in the operand field of a symbolic statement refers to the current location plus 12 bytes. Therefore, when an asterisk is encountered as a symbol, the Assembler Program replaces the asterisk with the address contained in the first entry of the Symbol Table.

The second entry contains the highest value the location counter reached during Phase I. This location is needed for the producing of an External Symbol Dictionary (ESD) START card, which is used by the relocatable loader. This is a function of Phase II and is discussed in detail in the introduction to Phase II.

The format of the Symbol Table Cards produced by Phase I is shown in Figure 6.

The attributes assigned to a symbol when the symbol is defined in the Symbol Table are as follows:

Location

This is a two-byte field containing the machine address of the symbol.

Length

This is a one-byte field containing the number of storage bytes associated with the symbol. This number will always be one less than the actual number due to programming considerations. For example, if the symbol DATA is the name of the DS instruction that defines a data area of 16 bytes, DATA would have a length of 15 assigned to it in the Symbol Table. When a length is not indicated, the Assembler Program assumes a length of one byte.

If the symbol is defined by a compound expression, the length is the same as the implied length of the leftmost simple expression in the compound expression. If the leftmost expression is a self-defining value, the length attribute is one.

Relocatability

This four-bit (1/2 byte) field contains a 1 - 15 if the symbol is relocatable and a zero if the symbol is not relocatable (i.e., is assigned an absolute address).

If the symbol is a linkage symbol that has been defined in the operand field of an EXTRN instruction, the relocatability field will contain a value from 2 to 15, depending upon which EXTRN instruction in the source program defines the symbol. For example, if the symbol was defined by the first EXTRN instruction, the relocatability value would be 2. If it was defined by the second EXTRN instruction, the relocatability value would be 3, etc. If the symbol is defined in the same program and not in the operand field of an EXTRN instruction, the relocatability attribute will be one.

Defined Bit

Each Symbol Table entry also contains one bit to indicate whether or not the symbol was ever defined.

1 | 2 3 4	5-16	17-28	29-40	41-52	53-64	65-76	77 - 80
12 9 4 | Address of first loc. of Phase II Symbol Table	*	*	*	*	*	*	Blank

* Each of these 12 bytes contains the symbol name, value, and attributes as follows:

Byte	Contents
1-2	Pointer to symbol position in table
3-4	Not used
5-8	Symbol in compressed form
9-10	Symbol value
11	Symbol length
12	Symbol attributes
	bits 0-3 relocatability
	bit 4 defined bit
	1 = defined
	0 = undefined
	bit 5 1 = entry used
	0 = not an entry or entry not defined.

Figure 6. Symbol Table Card Format

SECTION 3: SUBROUTINE DESCRIPTION

This section describes the subroutines of the Assembler Program that are used during Phase I processing.

EVE - Evaluation Routine (Charts AM and AN)

This subroutine is used during Phase I of the Assembler Program to accomplish the following:

1. Scan the Name Field
2. Scan for non-blank character
3. Scan the Operation Code Field
4. Translate the Operand Field

The first three items mentioned above are discussed in detail in Section 4, "Phase I Processing Flow," under "Control Routine." Only the translation of the operand field will be discussed here. However, the parameters necessary for each of the above procedures are shown at the end of this discussion.

This subroutine is shown in detail in Charts AM and AN.

Subroutine EVE will scan the operand field until the first character of the operand field expression is located. Subroutine EVE then translates the first term of the operand field expression and compares the translation to the parameters described below. The translated results, if correct, are placed into the relevant bytes of the Intermediate Text by subroutine STORE.

If the translation is not correct, control exits from this subroutine via the error exit set up by the calling routine.

After the translated term is placed into the Intermediate Text, control exits to the calling routine. The calling routine then determines if the character following the term is correct. If it is correct, and more terms of the operand field need translation, new parameters are set up and subroutine EVE is entered again.

The parameters used by subroutine EVE, which describe the next term of the expression in the operand field to be translated, are as follows:

T is set according to the type of term to be translated as follows:

- 0 = operand should be blank
- 1 = a simple term
- 2 = a compound term
- 3 = an integer (decimal value)

L indicates the number of bytes of intermediate text the term being translated will occupy.

M indicates the maximum value allowed for any self-defining value within the expression being translated.

PC indicates the starting column minus 2 (due to programming considerations) of the Intermediate Text into which the translated term of the expression will be placed.

C indicates the operation code of the statement being translated or the operation to be performed by subroutine EVE, as follows:

- 10 = CNOP
- 20 = EXTRN
- 20 = ENTRY
- 40 = Name Field or Operation Code scan
- 60 = DS or DC (type C,D,E,F,H,X), EJECT, SPACE, or scan until non-blank character
- 80 = END
- 90 = ORG or EQU
- A0 = START
- B0 = DC (type A), USING, DROP, or CCW (2nd term)
- 00 = all other operation codes

The other subroutines used by subroutine EVE are:

DCC - Define Character Constant: This subroutine is used when a character constant appears in the operand field of a source statement. It translates the character constant and checks its validity.

DCF - Define Full-Word Constant: This subroutine is used when a full-word constant appears in the operand field of a source statement. It converts a decimal integer to its binary representation and checks its validity.

DCX - Define Hexadecimal Constant: This subroutine is used when a hexadecimal constant appears in the operand field of a source statement. It translates the hexadecimal constant and checks its validity.

TLUS - Symbol Table Look-up: This subroutine is used when a symbol appears in the operand field of a source statement. It looks up the symbol in the Symbol Table and indicates its value (if known), its location in the Symbol Table, its length and relocatability attributes, and the bit

indicating whether or not the symbol is defined.

STORE - Store into Intermediate Text: This subroutine stores the translated terms of the expression in the operand field into the Intermediate Text.

ERR - Store Error Flags

The ERR subroutine places the necessary error flags into the Intermediate Text. If the error is such that the statement can not be assembled by Phase II, ID-Code L is also placed into the Intermediate Text. This subroutine is entered from the calling routine at location ERR if it is necessary to place the ID-Code into the Intermediate Text along with the error flags. If only the error flags are to be placed into the Intermediate Text, this subroutine is entered at location ERR1. The error flags are dependent upon the error conditions and are set by the calling routine.

The error flags that may be placed into the Intermediate Text by Phase I are shown in Figure 7.

Figure 8 shows what action is taken in Phase I of the Assembler Program when errors are encountered in the source statement.

Flag	Meaning
* A	Expression not simply relocatable
B	START, EXTRN, ENTRY, or ICTL out of order
C	Location counter overflow
E	More than 14 EXTRNs or more than 100 ENTRIES
F	Operand field format error or self-defining value in operand field too large
G	DC D or E range error
J	Symbol Table full
L	Name field error
M	Multiple defined symbol
* N	Statement not used
O	Invalid operation code
* R	Expression not absolute
* S	Specification error
* T	Value too large
V	ORG or EQU symbol not previously defined
* Y	Negative expression
Z	Column 72 not blank

* May also be set by Phase II.

Figure 7. Error Flags

BAR - Boundary Alignment Routine

The BAR subroutine is used to align the location counter to the proper word boundary, if necessary.

This subroutine will interrogate the boundary alignment bits of the Intermediate Text (byte 4) and, if necessary, add to the location counter the number of bytes that will align it to the proper word boundary.

After the location counter is incremented, it is tested to see if it exceeds the maximum allowable value of 65,535. If it does, the location counter is truncated and an error flag is placed into the Intermediate Text.

STORE - Store into Intermediate Text

The STORE subroutine is used to store the translated expressions of the operand field into the Intermediate Text.

This subroutine will store, one byte at a time, the translated terms of the expression in the operand field into the correct bytes of the Intermediate Text. The correct byte is determined by adding the value of the input parameter PC, which is set to the starting byte (minus 2) into which the translated term is to be placed, to the length of the translated term.

Therefore, the translated term is placed into the Intermediate Text beginning with the low-order byte of the field of the Intermediate Text associated with the term.

INSTRUCTION	NAME FIELD ERROR (Error Flag Set For All Instructions)	OPERAND FIELD ERROR
All Machine Instructions	Symbol stored in Symbol Table	Indicate how many bytes of zeros are to be assembled by Phase II for the operand field
CCW	Symbol not stored in Symbol Table	Indicate how many bytes of zeros are to be assembled by Phase II for the operand field
CNOP	Symbol ignored	Statement ignored
DC	Symbol not stored in Symbol Table	Statement ignored Undefined symbol in type A will set operand field to zero
DROP	Symbol ignored	Statement ignored
DS	Symbol not stored in Symbol Table	Statement ignored
EJECT	Symbol ignored	Symbol ignored
END	Symbol ignored	No ENTRY point to this program will be defined
ENTRY	Symbol ignored	Statement ignored
EQU	Statement ignored	Statement ignored
EXTRN	Symbol ignored	Statement ignored
ICTL	Symbol ignored	Operand set to a 1 in a tape system and to 25 in a card system
ORG	Symbol ignored	Statement ignored
SPACE	Symbol ignored	If not a decimal value, statement ignored
START	Symbol not stored in Symbol Table	Location counter set to zero
USING	Symbol ignored	Statement ignored

Figure 8. Error Summary

BUMP - Increment Location Counter

The BUMP subroutine is used to increment the location counter by the required number of bytes and to indicate if the location counter exceeds the maximum allowable value of 65,535.

After the source statement is translated and placed into the Intermediate Text, the location counter is incremented by the number of bytes determined by the operation code of the statement. The location counter is incremented (after boundary alignment) as shown in Figure 9.

This subroutine is entered with the parameter VALUE indicating the amount the location counter is to be incremented by or reset to. It is entered with parameter ADJUST BIT set to a zero if the location counter is to be incremented; parameter ADJUST BIT set to a one if the location counter is to be reset.

After the location counter is incremented or reset, it is tested to see if it exceeds the maximum allowable value. If it does, the location counter is truncated and an error flag is placed into the Intermediate Text.

The BUMP subroutine also places the highest value the location counter reached during Phase I of the assembly into the second entry of the Symbol Table.

Instruction Type	Location Counter Incremented by
RR Format	2 bytes
RX,RS,SI Format	4 bytes
SS Format	6 bytes
ORG	Reset to the value of the translated operand field
START	Translated operand field expression after alignment to a double-word boundary
CCW	8 bytes
CNOP	Depends upon the current setting of the location counter and the translated operand field expression
DS	Translated operand field expression
DC	Length of the constant multiplied by the duplication factor

Figure 9. Increment Values

TLUN/TLUS - Symbol Table Procedures

This subroutine is used during Phase I to build the Symbol Table and enter the attributes of each symbol into the Symbol Table. The subroutine has two entry points as follows:

1. TLUN - used when a symbol is encountered in the name field
2. TLUS - used when a symbol is encountered in the operand field

The symbol entry point in the Symbol Table is determined by dividing the compressed symbol by the length of the Symbol Table. The remainder is added to the starting location of the Symbol Table, and this sum is used as the entry point within the Symbol Table.

Thus, it can be seen that it is possible for more than one symbol to have the same entry point. A different symbol, therefore, may already have been entered into the Symbol Table at this entry point. When this mismatch occurs, the Symbol Table is searched (from the entry point down) until a match occurs or until the next empty location is found.

The symbol name is reduced from 48 to 32 bits as follows:

1. The first two bits of each character (always ones) are disregarded;
2. The next two bits of each character are saved, as the first 12 bits of the 32-bit reduced symbol name;
3. The remaining four bits of each character are converted to decimal and then, as a six-digit decimal number, this group is converted to binary and stored as the remaining 20 bits of the 32-bit reduced symbol name.

TLUN: The symbol entry point within the Symbol Table is checked. If the symbol has already been entered into the Symbol Table at this location, a check is made to determine if the symbol is defined. If the defined bit is a one, the symbol has been previously defined and error flag M is placed in byte 24 of the Intermediate Text.

If the defined bit is zero, the symbol has not been previously defined, and the attributes associated with the symbol are entered into the Symbol Table at this location. The defined bit is then set to a one.

If the symbol is not in the table at this location, and the location is empty, the symbol and its attributes are entered into this location. The defined bit is then set to a one.

If the symbol is not in the table at this location, and the location is not empty, a search of the Symbol Table is made to determine if the symbol was previously entered or to locate the next empty location within the table. If the symbol was previously entered at a different location, and the defined bit is not a one, the attributes associated with the symbol are entered into that location and the defined bit is set to a one. If the defined bit is a one, the symbol has been previously defined, and error flag M is placed into the Intermediate Text. If the symbol has not been previously entered into the Symbol Table, the symbol and its associated attributes are entered into the next empty location in the Symbol Table, and the defined bit is set to a one.

If the Symbol Table is completely scanned before the symbol or an empty location is found, error flag J is placed in byte 24 of the Intermediate Text to indicate that the Symbol Table is full.

This symbol, and any other symbol following that is not already entered into the Symbol Table, will remain undefined.

TLUS: The symbol entry point within the Symbol Table is checked. If the symbol has already been entered into the Symbol Table at this location, a pointer to this location is placed into the relevant bytes of the Intermediate Text.

If the symbol is not in the table at this location, and the location is empty, the symbol is entered into this location. Since the symbol is from the operand field, none of the symbol attributes are known at this time. Therefore, the remainder of the table entry for this symbol remains blank. The attributes for the symbol will be placed into the Symbol Table when the symbol is encountered in the name field of a source statement. A pointer to this location is then placed into the relevant bytes of the Intermediate Text.

If the symbol is not in the table at this location, and the location is not empty, a search of the Symbol Table is made to determine if the symbol was previously entered or to locate the next empty location within the Symbol Table. If the symbol was previously entered at a different location, a pointer to that location is placed into the relevant bytes of the Intermediate Text. If the symbol has not been previously entered into the Symbol Table, the symbol is entered into the next empty location in the Symbol Table, and a pointer to that location is placed into the relevant bytes of the Intermediate Text.

If the Symbol Table is completely scanned before the symbol or an empty location is found, error flag J is placed in byte 24 of the Intermediate Text.

INOUT - Input/Output Subroutine (CHARTS AO and AP)

This subroutine is the same for both Phase I and Phase II. It controls all the I/O operations for the Assembler Program.

The subroutine is entered at location INOUT from the calling routine. The interrupts are disabled and the CAW and device address are stored (02-04).

The I/O device is then tested for the following conditions (05):

1. Device unavailable. This is an error condition and an error halt occurs. The SEREP ID code for the unavailable device is set and the device address is stored in the old IOPSW (bytes 2 and 3).

2. Device busy. The program goes into an internal loop until the device is ready. When the device is ready, a test is made to determine if the CSW is stored.

3. CSW stored. Control is transferred to the interrupt routine.

The condition codes and their meanings are as follows:

Condition code	Meaning
0	Available
1	CSW stored (Immediate operation)
2	Busy
3	Unavailable

When the device is available and ready, an attempt is made to start the I/O operation. Two procedures are possible as follows:

1. The device is started and the data transfer initiated. Interrupts are enabled, and the program enters a wait state until an interrupt occurs (06-07). When an interrupt occurs, either as a result of the current SIO or from a previous SIO, the interrupt procedure is initiated. The interrupt entry switch is set (except when a busy condition was cleared by a TIO operation), and a Retry switch is tested (09-10).

2. The device is started for an immediate operation (rewind, skip), the interrupt procedure is initiated, and the Retry switch is tested.

The Retry switch is tested and if this a tape-positioning operation (backspace, skip) or sensing operation resulting from the error correction procedure, control is transferred to the error correction procedure.

If the Retry switch is not on, the read switch is set if this is a read operation, and the error conditions are checked (11-12). If it is a channel error, the SEREP ID code and the recovery address in the external New PSW are set and the program enters a wait state with AIS in the location counter (13-33-28). If it is a unit error and the unit is not tape, the SEREP ID code for device failure is set, the recovery address is set in the external New PSW, and the program enters a wait state with AIS in the location counter (14-34-28). If there is a unit error, and

the unit is tape, control is transferred to the error correction procedure (AP-A2).

If there are no errors, the busy indicator is checked (15). If the busy indicator is on, control is transferred to the interrupt procedure to check the Retry switch.

If the error switch was set on this pass, it is cleared and the registers restored before EOF and other exceptional conditions are checked.

If there are no exceptional conditions, control is returned to the calling routine. If an exceptional condition is caused by the printer, it is ignored, and control is returned to the calling routine; otherwise the error halt occurs and the recovery address is placed in the external New PSW (32).

Error Recovery Procedure: This procedure is entered for a tape error only. The Retry switch is set and the I/O Old PSW, CSW, and CAW are saved (01). The sense bytes are requested and the original conditions are restored (02-04).

If the sense information is a unit check, a test is made to determine if it is a first read error (05-06). If it is, the error switch is turned on. If it is not a first error, the read switch is checked and 10 read retries are made. If the read attempt is still not successful, an error halt occurs and a standard device failure message is produced.

If it is a write failure, three rewrite and skip operations occur three times before an error halt occurs and a standard device failure message is produced.

Note: The error recovery procedure described above does not conform to IBM standards; i.e., no priority checking or tape cleaning is implemented.

SECTION 4: PHASE I PROCESSING FLOW

This section uses three charts to describe the overall processing flow of Phase I of the Assembler Program. Chart 03 illustrates the initialization procedures, and Charts 04 and 05 illustrate the Control Routine. The numbers in parentheses within the text describing each chart indicate the block on the chart being explained.

INITIALIZATION - CHART 03

Phase I of the Assembler Program is read into main storage (01). The addressability and main storage size, as determined by the Phase I Configuration Card, are set up, and the necessary control parameters to be used by the subroutines of Phase I are generated and stored (02-03).

The Configuration Card is discussed in detail in the publication, IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.

The area reserved for the Symbol Table, determined by the size of main storage, is cleared, and the first two entries are reserved for the uses indicated in Section 2, "Tables" (04).

The input/output addresses are determined and set up for the read and write routines. These addresses are determined by the intermediate text device being used (card or tape). The size of the Intermediate Text to be written or punched as output (24 bytes for a card intermediate text system, 104 bytes for a tape intermediate text system) is stored into the necessary locations of the write routine (05-12). The first source statement is then read in (13).

In a card intermediate text system, if column 1 of the first card indicates it is a Symbol Table card (14), the Symbol Table deck is read into the area of main storage reserved for the Symbol Table (15). The use of the Symbol Table deck is discussed in detail in the publication, IBM System/360 Basic Programming Support Basic Assembler Language, Form C28-6503.

After the last card of the Symbol Table deck is read into main storage, or when the Symbol Table deck is not present, control exits to the Phase I Control Routine (Chart 04).

CONTROL ROUTINE - CHART 04

The first statement of the source program is read in (01). The first column of the card to be scanned by the EVE subroutine (determined by the SYSCON control byte of the Phase I Configuration Card) is set, the control switches are set, and the punch buffer output area is cleared (02).

Column 72 of the source program statement, which must be blank, is tested (03). If it does not contain a blank, error flag Z is placed in byte 22 of the relevant Intermediate Text (04). Any information contained in column 72 is ignored when the source statement is processed, but column 72 is printed as part of the source statement in the program listing.

The first column of the source statement is tested for an asterisk or a blank (05-06). If the source statement begins with a blank, the operation code is then checked (11). If the column contains an asterisk, ID-Code C, indicating the source card is a comments card, is placed in byte 1 of the Intermediate Text. No further translation of the source statement is performed during the assembly process. A comments card is only reproduced in the object program listing.

If the statement begins with neither an asterisk nor a blank, it indicates that the source statement begins with a symbol in the name field. The symbol in the name field is scanned by subroutine EVE to determine if it is valid (08). If the symbol is valid (i.e., begins with an alphabetic character and contains no more than six characters), the valid-symbol indicator is set. However, the symbol is not placed into the Symbol Table at this time. A symbol in the name field of the source statement being translated may not be allowed. Until this can be determined, the symbol is only temporarily stored (09). If the symbol is not valid, error flag L is placed in byte 24 of the Intermediate Text (10). The operation code is then checked (11).

If the operation code field of the source statement is blank (the remainder of the source statement is blank) ID-Code L, indicating the statement is not to be assembled as part of the object program, and error flags NO are placed in bytes 1, 23, and 24, respectively, of the Intermediate Text (17). If the remainder of the source statement is not blank, but the operation code has been omitted, the first characters of the operand field or comments field will be treated as the operation code. Control then exits to produce the Intermediate Text (Chart 05, block F2).

If the operation code field contains an operation code, the Operation Code Table is searched to determine if the operation code is valid (12). If the operation code is not valid, ID-Code L, indicating the statement is not to be assembled as part of the object program, and error flags N and O are placed in bytes 1, 23, and 24, respectively, of the Intermediate Text (17). The next source program statement is then read in (01).

If the operation code contained in the source statement is found during the search of the Operation Code Table, it is checked for type (14). If it is a machine operation code, ID-Code A is placed in byte 1 of the Intermediate Text, and the location counter is aligned to a half-word boundary (18-19). Control then exits to the subroutine necessary to translate the operand field as determined by the operation code.

If the operation code indicates an Assembler Instruction, ID-Code B is placed in byte 1 of the Intermediate Text and control exits to the subroutine necessary to translate the operand field as determined by the operation code. (See Section 5, "Operand Field Translation.")

Subroutine EVE is used to translate the operand field of each statement. The translation of each operand field by subroutine EVE is basically the same regardless of the type of statement. This subroutine is, however, entered with different parameters for each statement type. These parameters are shown in the description of subroutine EVE in Section 3, "Subroutine Description."

Each expression within the operand field is translated separately and placed in the relevant bytes of the Intermediate Text for that statement.

If an error is encountered in the operand field, the remaining expressions within the operand field are not translated. The intermediate text that has been developed up to that point, however, is written as Intermediate Text with the relevant ID-Code and error flags. The ID-Codes used in the Intermediate Text are discussed in the description of the Intermediate Text in Section 2, "Tables." The error flags and their meanings are shown in Section 3, "Subroutine Description," under subroutine ERR. These flags are also shown in Figure 3.

If an error is encountered during the translation of the operand field such that the statement will not be assembled by Phase II (20), control exits to produce the Intermediate Text (Chart 05, block F2). If not, control exits to Chart 05, block B2, to process the name field.

CONTROL ROUTINE - CHART 05

The valid-name indicator (set if the symbol in the name field was valid) and the name-allowed indicator (set by the operand translation subroutine if a symbol in the name field of the source statement being translated is legal) are tested (01).

If neither indicator is set, byte 24 of the Intermediate Text is tested to determine if an invalid symbol (error flag L) was in the name field (05). If byte 24 is blank, control transfers to produce the Intermediate Text (09). If byte 24 contains error flag L, the ID-Code indicating an error is placed in byte 1 of the Intermediate Text , replacing the ID-Code already there (04). Control then transfers to produce the Intermediate Text (09).

If one indicator is set, the valid-name indicator is checked (02). If the valid-name indicator is not set, the procedures described above for neither indicator set are followed (05). If the valid-name indicator is set but not the name-allowed indicator, the ID-Code indicating an error is placed in byte 1 of the Intermediate Text replacing the ID-Code already there (04). It is possible that the symbol in the name field of this source statement is referenced in the operand field of another source statement. If this is the case, the source statement referencing this symbol will contain an error flag, indicating an undefined symbol because this symbol is not placed into the Symbol Table. Flagging both statements will make the error easier to locate. Control then transfers to produce the Intermediate Text (09).

If both indicators are set, indicating there is a valid symbol in the name field and a name is allowed for the source statement being translated, the value of the location counter is placed into the value attribute of the symbol in the Symbol Table (06).

If an error is encountered by subroutine TLUN while placing the symbol into the Symbol Table (e.g., Symbol Table is full), the ID-Code indicating an error is placed in byte 1 of the Intermediate Text along with the error flag placed by subroutine TLUN, replacing the ID-Code already there. Control then transfers to produce the Intermediate Text (09).

If no error is encountered by subroutine TLUN (08), control transfers to produce the Intermediate Text (09).

Depending upon the configuration of the system being used to assemble the program, one of the following three procedures, along with the common procedure, is followed.

Tape Procedure (blocks 09, 12-14): The Intermediate Text is written on tape.

1442 Card Read Punch Procedure (blocks 10, 14, 20-21): If the information in the first 24 columns of the source card is equal to the information in the Intermediate Text (10), this is a reassembly, and the punching of the Intermediate Text is not necessary since the source card already contains the record for that statement. Control then transfers to test for an END card (15).

If the information is not equal (10) and the first 24 columns of the source card are blank (20), the Intermediate Text is punched into these columns (14).

If the information is not equal (10) or the first 24 columns of the source card are not blank (20), the punch buffer is blanked and a '9' is placed into column 1 of the Intermediate Text. If the system has a printer or typewriter, the '9' is punched into the card, and error flag N will be printed in the listing produced by Phase II. If no printer or typewriter is available, the system stops. A manual interrupt will cause processing to be resumed; the '9' is punched into column 1 of the card (14), and processing continues.

2540 Card Read Punch Procedure (blocks 09, 12-14: The Intermediate Text is moved to the output area, if necessary, and punched into the first 24 columns of a blank card. Then the first 48 columns, the Identification-Sequence Field, and column 72 of the source card are punched into the remaining columns of the new card.

Common Procedure (blocks 11, 15-19): After the Intermediate Text is written or punched, a test is made to determine if the statement just translated was an END statement (15). If it was not, the location counter is incremented by the size (number of bytes) of the statement (11), and control exits to read the next source statement (Chart 04, block 01).

If the statement is an END statement (15), and a card intermediate text system is being used (16), the Symbol Table Deck is punched into the blank cards following the source program (17). If a tape intermediate text system is being used, it is not necessary to punch the Symbol Table Deck because Phase II must immediately follow Phase I.

In either case, the attributes of the Symbol Table are relocated to upper main storage (18). (See description of Symbol Table in Section 2, "Tables.") The message 1EA is printed (19), and the program waits for Phase II to be loaded. However, if the assembler was loaded from tape, the message 1EA is not printed, and Phase II is loaded immediately.

SECTION 5: OPERAND FIELD TRANSLATION

This section discusses the translation of the operand fields of each different instruction format. A chart of the logic flow is given for each type. The reader will require a basic knowledge of the subroutines described in Section 3, "Subroutine Description," before using this section.

ASSEMBLER INSTRUCTION STATEMENTS

CCW Translation - Chart AA

The CCW instruction provides a convenient way to define and generate an 8-byte Channel Command Word aligned to a double-word boundary.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Optional	CCW	Four expressions separated by commas

Where the four expressions, from left to right, are:
- A simple absolute expression specifying the command code
- A relocatable expression specifying the data address
- A simple absolute expression specifying the flags and bits 37-39
- A simple absolute expression specifying the count

This subroutine is entered at location CCW (block AA01) from the Phase I Control Routine. It translates the operand into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

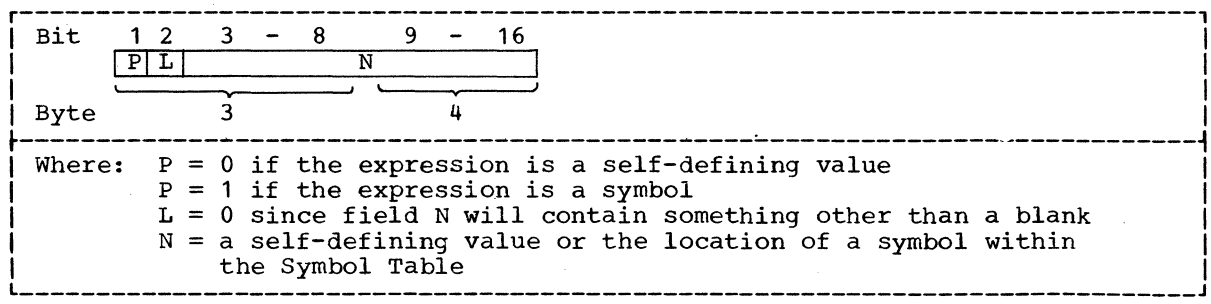
The format of the Intermediate Text is as follows:

BYTE 1	2	* 3-4	5-16	17-20	21-22	23-24
ID-code	Internal code of instruction (E4)	Translated first expression	Translated second expression	Translated fourth expression	Translated third expression	Error flags

* Byte 3 contains the number of bytes of zeros to be assembled by Phase II if an error is encountered.

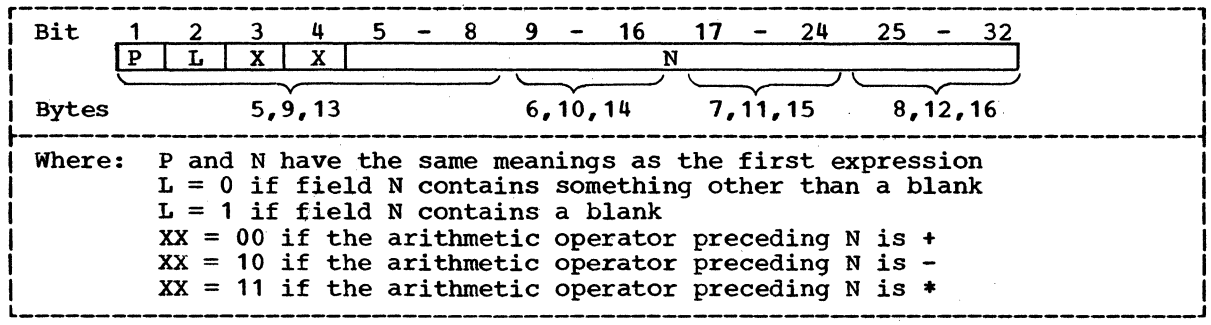
The bit patterns of the translated expressions placed into the Intermediate Text are as follows:

First expression:



Where: P = 0 if the expression is a self-defining value
P = 1 if the expression is a symbol
L = 0 since field N will contain something other than a blank
N = a self-defining value or the location of a symbol within the Symbol Table

Second expression:



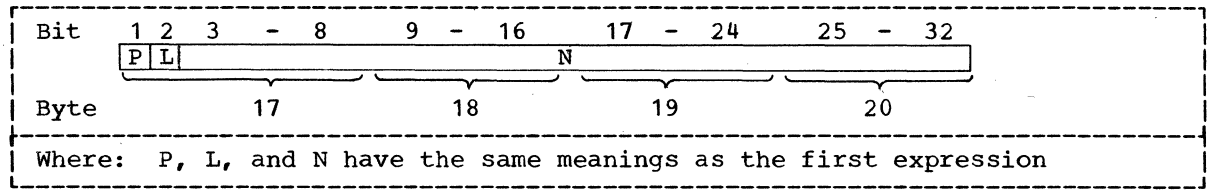
Where: P and N have the same meanings as the first expression
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *

Third expression:

This expression has the same format as the first expression except bytes 21 and 22 are used.

Fourth expression:

This expression has a format similar to the first expression except bytes 17 through 20 are used.



Where: P, L, and N have the same meanings as the first expression

DC Translation - Chart AB

The DC instruction is used for generating constant data in main storage.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Optional	DC	dtLn'c' or ALn(c)

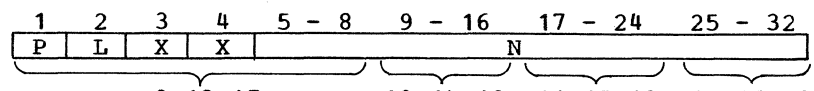
Where: d is the duplication factor
t specifies the type of constant
Ln specifies the length of the constant
c specifies the constant

This subroutine is entered at location DC (block AB01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04). The format of the Intermediate Text is as follows:

BYTE 1	* 2	3	4	** 5-6	*** 7-22	23-24
ID-code	Internal code of instruction	Constant length	Boundary alignment pattern	Duplication factor	Translated constant	Error flags

* E1 except DC type A which is E2.
** One if DC type A or not specified.
*** Bytes 9-20 contain the translated expression for DC type A, and bytes 7, 8, 21, and 22 are blank.

The relocatable expression allowed for code type A may have a maximum of three terms. Therefore, bytes 9-20 are divided into three 4-byte fields. The bit pattern for each translated term is as follows:



Where: P = 0 if the term is a self-defining value
P = 1 if the term is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *
N = a self-defining value or the location of a symbol within the Symbol Table

CNOP Translation - Chart AC (Blocks 01-09)

The CNOP instruction is used to align an instruction to a specific word boundary.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Not used	CNOP	b,w

Where: b is a decimal value of 0, 2, 4, or 6
w is a decimal value of 4 or 8

The following combinations of b and w are valid:

0,4	2,8
2,4	4,8
0,8	6,8

This subroutine is entered at location CNOP (block AC01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3	4	5-6	7-8	9-22	23-24
ID-code	Internal code of instruction (E3)	Blank	Boundary alignment pattern	Translated value of b	Translated value of w	Blank	Error flags

The boundary alignment bit pattern placed into byte 4 is as follows:

Bit	1	2	3	4	5	6	7	8
			M	M	M	N	N	N

Byte 4

Where: NNN = 011 if alignment is to a full-word boundary (w = 4)
NNN = 111 if alignment is to a double-word boundary (w = 8)
MMM = 000 if location counter is to be set to byte 0 of NNN (b = 0)
MMM = 010 if location counter is to be set to byte 2 of NNN (b = 2)
MMM = 100 if location counter is to be set to byte 4 of NNN (b = 4)
MMM = 110 if location counter is to be set to byte 6 of NNN (b = 6)

DS Translation - Chart AD (Blocks 10-20)

The DS instruction is used to reserve main storage areas and to assign symbolic names to the areas that are to be reserved.

The format of the source statement is as follows:

NAME	OPERATION	OPERAND
Optional	DS	dtLn
Where: d is the duplication factor t is the type of field Ln specifies the length (valid for field code C only)		

This subroutine is entered at location DS (block AD10) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3	4	5-6	7-22	23-24
ID-code	Internal code of instruction (E0)	Length of Constant	Boundary alignment pattern	Translated value (d times n)	Blank	Error flags

DROP Translation - Chart AD (Blocks 12-15)

The DROP instruction specifies a previously available general register that is no longer available for base addressing.

The source statement format is as follows:

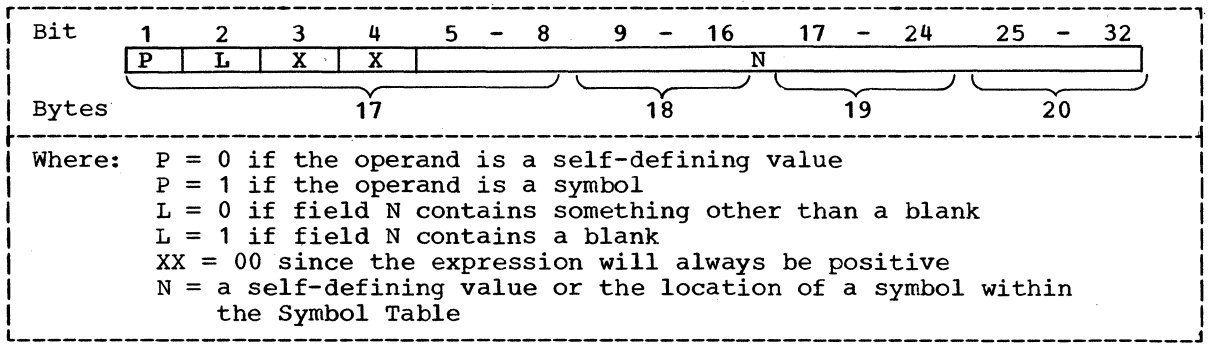
NAME	OPERATION	OPERAND
Not used	DROP	A simple absolute expression

This subroutine is entered at location DROP (block AD12) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-16	17-20	21-22	23-24
ID-code	Internal code of instruction (EC)	Blank	Translated operand	Blank	Error flags

The bit pattern for the translated operand field placed into bytes 17-20 is as follows:



Where: P = 0 if the operand is a self-defining value
P = 1 if the operand is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 since the expression will always be positive
N = a self-defining value or the location of a symbol within the Symbol Table

EJECT Translation - Chart AD (Blocks 01-02)

The EJECT instruction causes the next line of the object program listing to appear at the top of a new page.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Not used	EJECT	Not used

This subroutine is entered at location EJECT (block AD01) from the Phase I Control Routine. The operand field is checked for a blank. If it is not a blank, the necessary ID-Code and error flags are placed in the Intermediate Text. Control then returns to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-22	23-24
ID-code	Internal code of instruction (EB)	Blank	Error flags

END Translation - Chart AD (Blocks 08-11)

The END instruction terminates the assembly of a program. It may also indicate a point in the program to which control may be transferred after the program is loaded.

The source statement format is as follows:

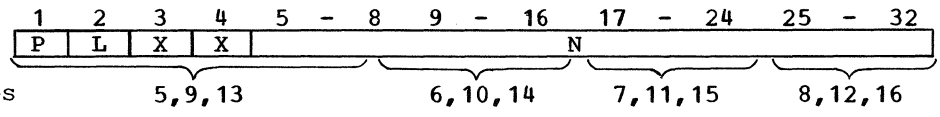
NAME	OPERAND	OPERATION
Not used	END	A relocatable expression or a blank
Where: The relocatable expression specifies the point to which control may be transferred after the object program is loaded.		

This subroutine is entered at location END (block AD08) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	* 5-16	17-22	23-24
ID-code	Internal code of instruction (EF)	Blank	Translated operand	Blank	Error flags
* Blank if the operand field is blank.					

The relocatable expression may have a maximum of three terms. Therefore, bytes 5-16 are divided into three 4-byte fields. The bit pattern for each translated term of the operand field placed into bytes 5-16 is as follows:



Where:
- P = 0 if the term is a self-defining value
- P = 1 if the term is a symbol
- L = 0 if field N contains something other than a blank
- L = 1 if field N contains a blank
- XX = 00 if the arithmetic operator preceding N is +
- XX = 10 if the arithmetic operator preceding N is -
- XX = 11 if the arithmetic operator preceding N is *
- N = a self-defining value or the location of a symbol within the Symbol Table

ENTRY Translation - Chart AD (Blocks 03-07)

The ENTRY instruction identifies a symbol, defined elsewhere in the program, that may be used as an entry point to this program by other programs.

The source statement format is as follows:

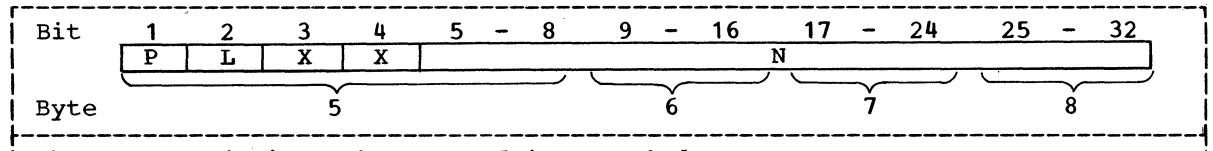
NAME	OPERATION	OPERAND
Not used	ENTRY	A relocatable symbol

This subroutine is entered at location ENTRY (block AD07) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-8	9-14	15-22	23-24
ID-code	Internal code of instruction (EA)	Blank	Translated operand	Actual symbol from operand	Blank	Error flags

The bit pattern of the translated relocatable symbol placed into bytes 5-8 is as follows:



Where: P = 1 since the operand is a symbol
L = 0 since field N will contain something other than a blank
XX = 00 since the location of the symbol is always positive
N = the location of a symbol within the Symbol Table

EQU Translation - Chart AE (Blocks 01-08)

The EQU instruction is used to assign to a symbol in the name field the same length and attributes as an expression in the operand field.

The format of the source statement is as follows:

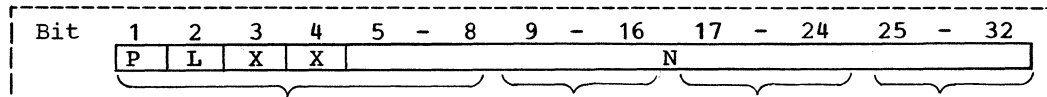
NAME	OPERATION	OPERAND
Required	EQU	An expression (any symbols in the expression must be previously defined)

This subroutine is entered at location EQU (block AE01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-16	17-22	23-24
ID-code	Internal code of instruction (E5)	Blank	Translated operand	Blank	Error flags

The operand field may have a maximum of three terms. Therefore, bytes 5-16 are divided into three 4-byte fields. The bit pattern for each translated term is as follows:



Where: P = 0 if the term is a self-defining value
P = 1 if the term is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *
N = a self-defining value of the location of a symbol within the Symbol Table

EXTRN Translation - Chart AE (Blocks 09-14)

The EXTRN instruction identifies a linkage symbol as an external symbol defined in some other program that will be referred to in this program.

The source statement format is as follows:

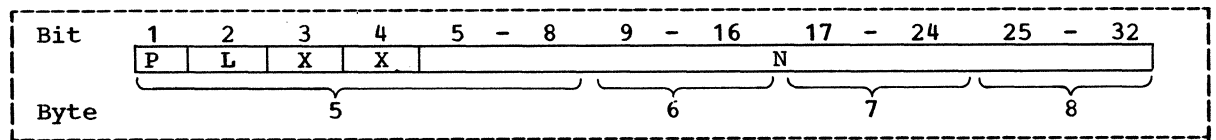
NAME	OPERATION	OPERAND
Not used	EXTRN	A relocatable symbol

This subroutine is entered at location EXTRN (block AE09) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-8	9-14	15-22	23-24
ID-code	Internal code of instruction (E9)	Blank	Translated operand	Actual symbol from operand	Blank	Error flags

The bit pattern of the translated relocatable symbol placed into bytes 5-8 is as follows:



Where: P = 1 since the operand is a symbol
L = 0 since field N contains something other than a blank
XX = 00 since the location of the symbol is always positive
N = the location of a symbol within the Symbol Table

ICTL Translation - Chart AF (Blocks 01-07)

The ICTL instruction tells the Assembler Program in which column the statement portion of the source program statement begins.

The format of the source statement is as follows:

NAME	OPERATION	OPERAND
Not used	ICTL	The decimal value 1 or 25

This subroutine is entered at location ICTL (block AF01) from the Phase I Control Routine. When the ICTL instruction is used, it must be the first statement in the source program. This subroutine translates the operand into intermediate text which will be reproduced in the object program listing. If the operand field is incorrect, Phase I will set the starting column to a 1 for a tape system and to a 25 for a card system. Control returns to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-22	23-24
ID-code	Internal code of instruction (ED)	Translated operand	Blank	Error flags

ORG Translation - Chart AG (Blocks 01-07)

The ORG instruction is used to set the location counter to a specified address.

The format of the source statement is as follows:

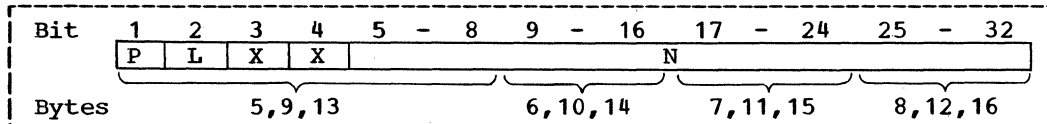
NAME	OPERATION	OPERAND
Not used	ORG	A relocatable expression (any symbols in the expression must be previously defined)

This subroutine is entered at location ORG (block AG01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-16	17-22	23-24
ID-code	Internal code of instruction (E8)	Blank	Translated operand	Blank	Error flags

The relocatable expression may have a maximum of three terms. Therefore, bytes 5-16 are divided into three 4-byte fields. The bit pattern for each translated term is as follows:



Where: P = 0 if the term is a self-defining value
P = 1 if the term is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *
N = a self-defining value or the location of a symbol within the Symbol Table

SPACE Translation - Chart AG (Blocks 08-12)

The SPACE instruction is used to insert one or more blank lines in the object program listing.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Not used	SPACE	A decimal value not greater than 63

This subroutine is entered at location SPACE (block AG08) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program. If the operand field is blank, the Assembler assumes a value of 1. If the operand field is greater than 63, the SPACE instruction is treated the same as the EJECT instruction by Phase II of the Assembler Program. Control returns to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-22	23-24
ID-code	Internal code of instruction (E7)	Translated operand	Blank	Error flags

START Translation - Chart AF (Blocks 08-16)

The START instruction may be used to indicate the beginning of an assembly, to give a name to the object program, and to set the location counter to an initial value.

The format of the source statement is as follows:

NAME	OPERATION	OPERAND
Optional	START	A self-defining value or blank

This subroutine is entered at location START (block AF08) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program. If the operand field contains an error, the location counter will be set to zero. Control returns to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	3	4	5-8	9-14	15-22	23-24
ID-code	Internal code of instruction (EE)	Blank	Boundary alignment pattern	Translated operand	Actual symbol from name field	Blank	Error flags

The translated operand placed into bytes 5-8 will be the assembled 24-bit address of the first byte of the object program.

USING Translation - Chart AH (Blocks 01-07)

The USING instruction indicates that the general register specified is available for use as a register for base addressing. This instruction also states the base-address value that the Assembler may assume will be in the indicated general register at object time.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Not used	USING	A relocatable expression and a simple expression separated by a comma
Where: The relocatable expression specifies a value that the Assembler can use as a base register. The simple absolute expression specifies the general register that can be assumed to contain the base address represented by the relocatable expression.		

This subroutine is entered at location USING (block AH01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

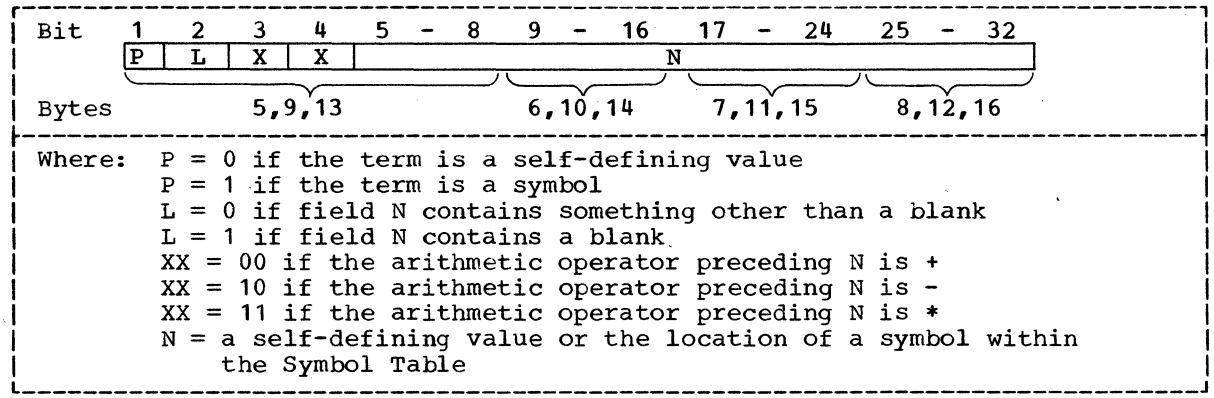
The format of the Intermediate Text is as follows:

BYTE 1	2	3-4	5-16	17-20	21-22	23-24
ID-code	Internal code of instruction (E6)	Blank	Translated relocatable expression	Translated simple expression	Blank	Error flags

The bit patterns for the translated expression of the operand field placed into the intermediate text are as follows:

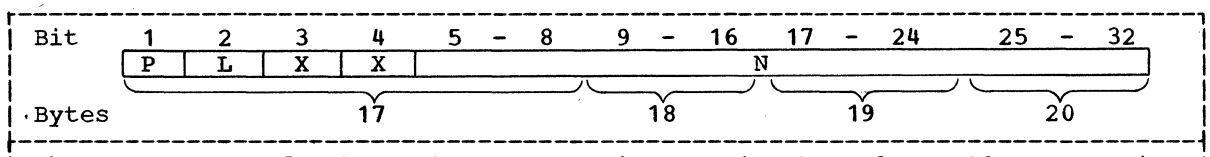
Relocatable Expression:

The relocatable expression may have a maximum of three terms. Therefore, bytes 5-16 are divided into three 4-byte fields. The bit pattern for each translated term is as follows:



Where: P = 0 if the term is a self-defining value
P = 1 if the term is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *
N = a self-defining value or the location of a symbol within the Symbol Table

Simple Expression:



Where: P, L, and N have the same meanings as in the relocatable expression
XX = 00 since the expression will always be positive

MACHINE INSTRUCTION STATEMENTS

RR Format Translation - Chart AH (Blocks 08-17)

Instructions in the RR format are used for register-to-register operations except for SVC (the operand specifies immediate data) and SPM (the operand specifies one general register)

The source statement formats are as follows:

NAME	OPERATION	OPERAND
Optional	All RR instructions except SVC and SPM	R1, R2
Optional	SVC	I
Optional	SPM	R1

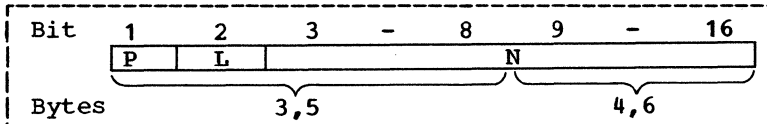
This subroutine is entered at location MACHB (block AH09) from the Phase I Control Routine, except for the SVC instruction which enters at location MACHI (block AH13) and the SPM instruction which enters at location MACHJ (block AH01). This subroutine translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	* 3-4	** 5-6	7-20	21-24
ID-code	Internal code of instruction	Translated R2 or I field	Translated R1 field	Blank	Error flags

* Blank if no R2 field. Byte 3 contains the number of zeros to be assembled by Phase 2 if an error is encountered.
** Blank if SVC instruction.

The bit pattern for the translated R1 field, R2 field, or I field is as follows:



Where: P = 0 if the field is a self-defining value
P = 1 if the field is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
N = a self-defining value or the location of a symbol within the Symbol Table

RS Format Translation - Chart AI

Instructions in the RS Format are used for register-to-storage operations and for shift operations.

The source statement formats are as follows:

Format type 1

NAME	OPERATION	OPERAND
Optional	All instructions except shift instructions	R1, R3, D2(B2)

Format type 2

NAME	OPERATION	OPERAND
Optional	All shift instructions	R1, D2(B2)

This subroutine is entered at location MACHC (block AI01) from the Phase I Control Routine, except for format type 2 which enters at location MACHD (block AI02). This subroutine translates the operand of the instruction into

intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

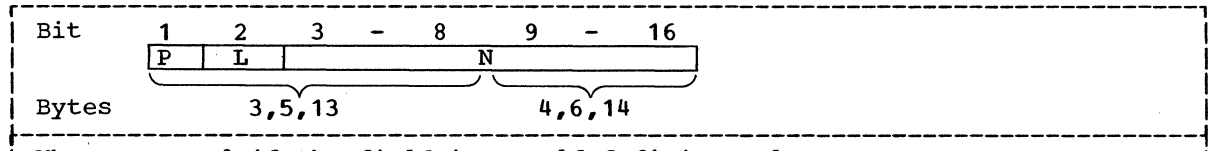
The format of the Intermediate Text is as follows:

BYTE 1	2	* 3-4	5-6	7-12	** 13-14	15-20	21-24
ID-code	Internal code of instruction	Translated R3 field	Translated R1 field	Translated D2 field	Translated B2 field	Blank	Error flags

* Blank if not specified or if format type 2 is being translated. Byte 3 contains the number of bytes to be assembled as zeros if an error is encountered.
** Blank if not specified.

The bit patterns for the translated fields of the operand placed into the Intermediate Text are as follows:

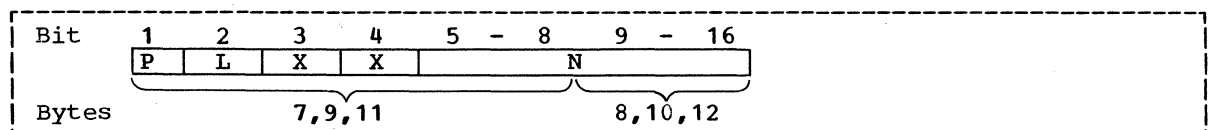
R3 field, R1 field, or B2 field:



Where:
P = 0 if the field is a self-defining value
P = 1 if the field is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
N = a self-defining value or the location of a symbol within the Symbol Table

D2 field:

The D2 field may have a maximum of three terms. Therefore, bytes 7-12 are divided into three 2-byte fields. The bit pattern for each translated term is as follows:



Where:
P, L, and N have the same meanings as indicated for the R3, R1, and B2 fields
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *

RX Format Translation - Chart AJ

Instructions in the RX Format are used for register-to-indexed-storage operations.

The source statement format is as follows:

NAME	OPERATION	OPERAND
Optional	Any RX instruction	R1,D2(B2,X2)

This subroutine is entered at location MACHA (block AJ01) from the Phase I Control Routine. It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The format of the Intermediate Text is as follows:

BYTE 1	2	* 3-4	5-6	7-12	* 13-14	15-20	21-24
ID-code	Internal code of instruction	Translated X2 field	Translated R1 field	Translated D2 field	Translated B2 field	Blank	Error flags

* Blank if not specified. Byte 3 contains the number of bytes to be assembled as zeros if an error is encountered.

The bit patterns for the translated fields of the operand placed into the Intermediate Text are as follows:

X2 field, R1 field, or B2 field:

Bit	1	2	3 - 8	9 - 16
	P	L	N	
Bytes	3,5,13		4,6,14	

Where:
P = 0 if the field is a self-defining value
P = 1 if the field is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
N = a self-defining value or the location of a symbol within the Symbol Table

D2 field:

The D2 field may have a maximum of three terms. Therefore, bytes 7-12 are divided into three 2-byte fields. The bit pattern for each translated term is as follows:

Bit	1	2	3	4	5 - 8	9 - 16
	P	L	X	X	N	
Bytes			7,9,11			8,10,12

Where: P, L, and N have the same meanings as indicated for the X2, R1, and B2 fields
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *

SI Format Translation - Chart AK

Instructions in the SI Format are used for storage and immediate data operations.

The source statement formats are as follows:

Format type 1

NAME	OPERATION	OPERAND
Optional	All SI instructions except those in format type 2	D1(B1),I2

Format type 2

NAME	OPERATION	OPERAND
Optional	LPSW,SSM,HIO, SIO,TIO,TCH,TS	D1(B1)

This subroutine is entered at location MACHE (block AK01) from the Phase I Control Routine, except for format type 2 which enters at location MACHF (block AK02). It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The formats of the Intermediate Text are as follows:

Format type 1

BYTE 1	2	* 3-6	7-12	** 13-14	15-20	21-24
ID-code	Internal code of instruction	Translated I field	Translated D1 field	Translated B1 field	Blank	Error flags

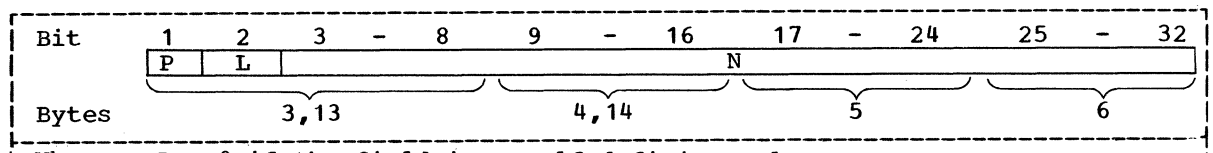
* Byte 3 contains the number of bytes to be assembled as zeros if an error is encountered.
** Blank if not specified.

Format type 2

The format is the same as type 1, except that bytes 3-6 are blank.

The bit patterns for the translated fields of the operand placed into the Intermediate Text are as follows:

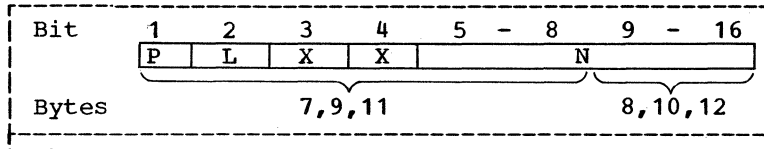
I field or B1 field:



Where: P = 0 if the field is a self-defining value
P = 1 if the field is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
N = a self-defining value or the location of a symbol within the Symbol Table

D1 field:

The D1 field may have a maximum of three terms. Therefore, bytes 7-12 are divided into three 2-byte fields. The bit pattern for each translated term is as follows:



Where: P, L, and N have the same meanings as indicated for the I2 and B1 fields
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *

SS Format Translation - Chart AL

Instructions in the SS Format are used for storage-to-storage operations.

The source statement formats are as follows:

Format type 1

NAME	OPERATION	OPERAND
Optional	PACK, UNPK, MVO, AP, CP, DP,MP,SP,ZAP	D1(L1,B1), D2(L2,B2)

Format type 2

NAME	OPERATION	OPERAND
Optional	All other SS instructions	D1(L,B1), D2(B2)

This subroutine is entered at location MACHG (block AL01) from the Phase I Control Routine, except for format type 2 which enters at location MACHH (block AL02). It translates the operand of the instruction into intermediate text meaningful to Phase II of the Assembler Program and returns control to location C311 in the Phase I Control Routine (Chart 04).

The formats of the Intermediate Text are as follows:

Format type 1

BYTE 1	2	* 3-4	* 5-6	7-12	* 13-14	* 15-20	* 21-22	23-24
ID-code	Internal code of instruction	Translated L2 field	Translated L1 field	Translated D1 field	Translated B1 field	Translated D2 field	Translated B2 field	Error flags

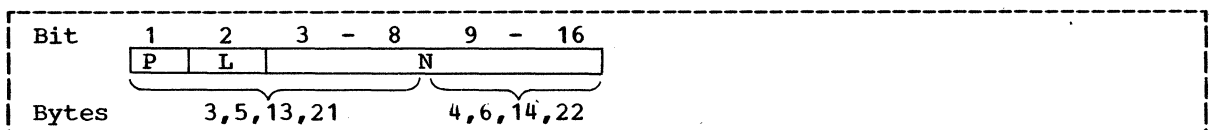
* Blank if not specified. Byte 3 contains the number of bytes to be assembled as zeros if an error is encountered.

Format type 2

The format is the same as type 1, except that bytes 3-4 contain the translated L field and bytes 5-6 are blank.

The bit patterns for the translated fields of the operand placed into the Intermediate Text are as follows:

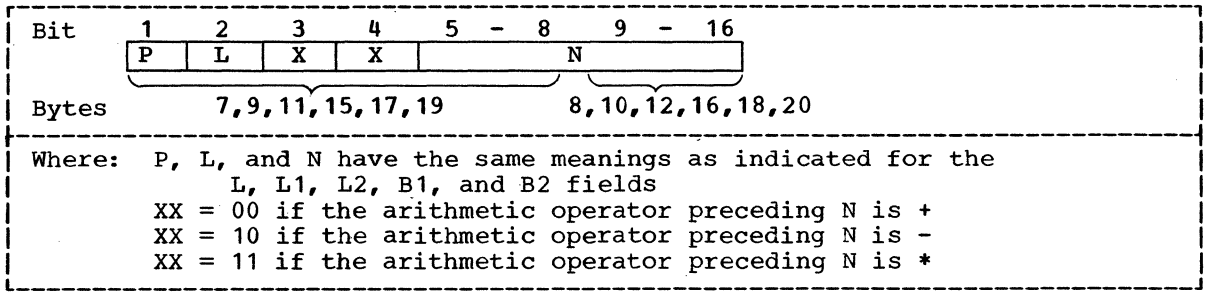
L or L2 field, L1 field, B1 field, or B2 field:



Where:
P = 0 if the field is a self-defining value
P = 1 if the field is a symbol
L = 0 if field N contains something other than a blank
L = 1 if field N contains a blank
N = a self-defining value or the location of a symbol within the Symbol Table

D1 field or D2 field:

The D1 field and the D2 field may contain a maximum of three terms. Therefore, bytes 7-12 and 15-20 are divided into three 2-byte fields. The bit pattern for each translated term is as follows:



Where: P, L, and N have the same meanings as indicated for the L, L1, L2, B1, and B2 fields
XX = 00 if the arithmetic operator preceding N is +
XX = 10 if the arithmetic operator preceding N is -
XX = 11 if the arithmetic operator preceding N is *

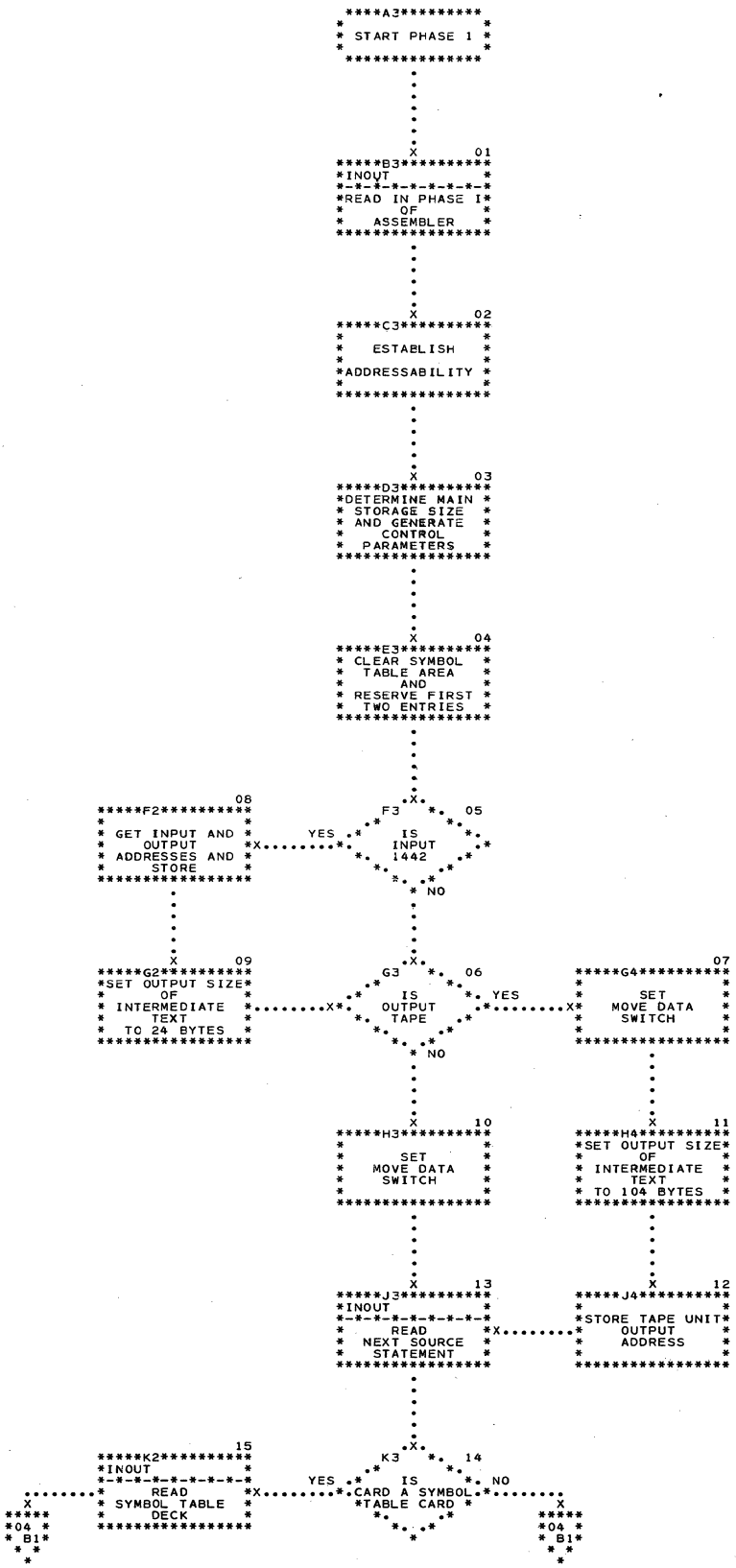


Chart 03. Phase I Initialization

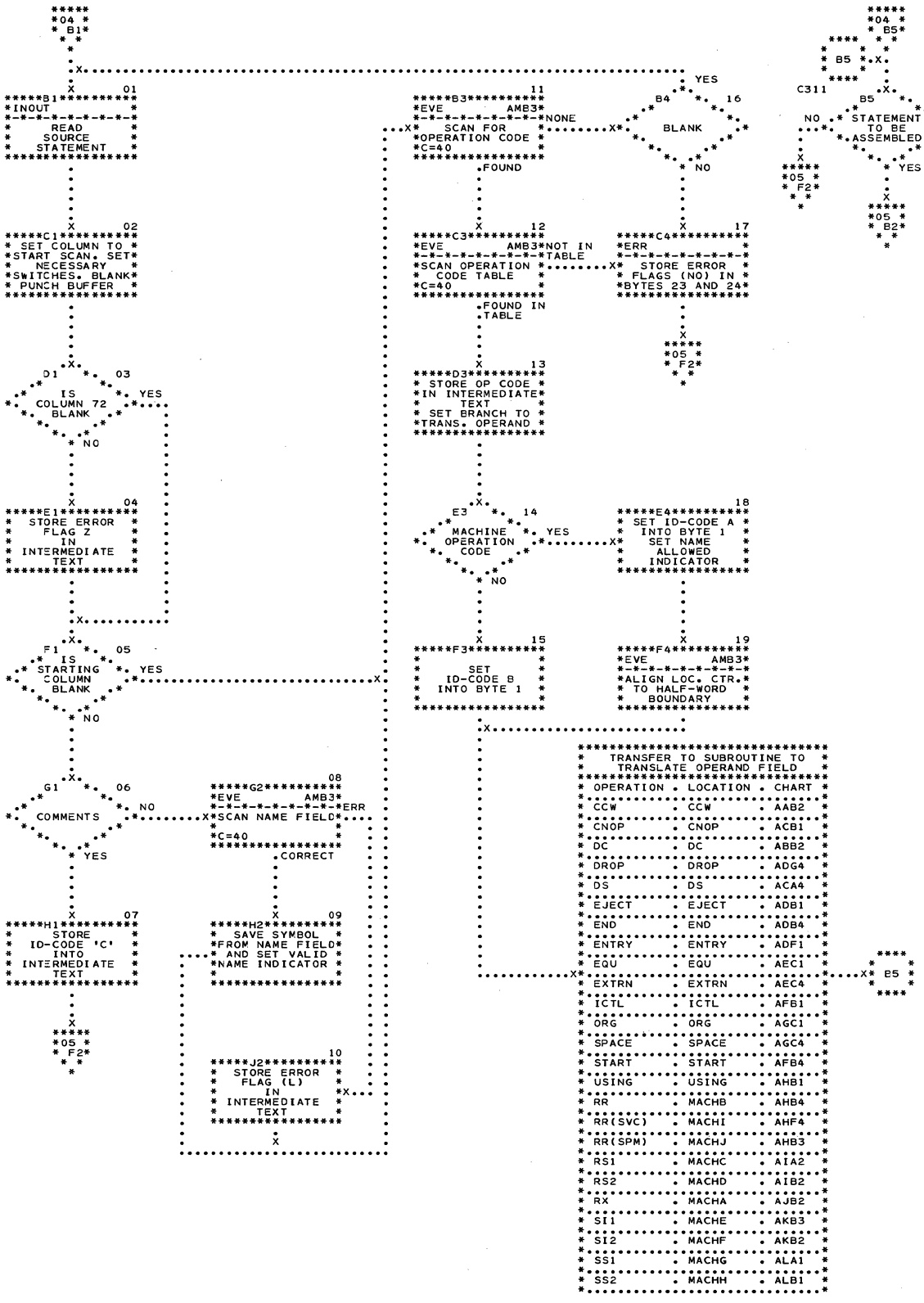


Chart 04. Phase I Control Routine

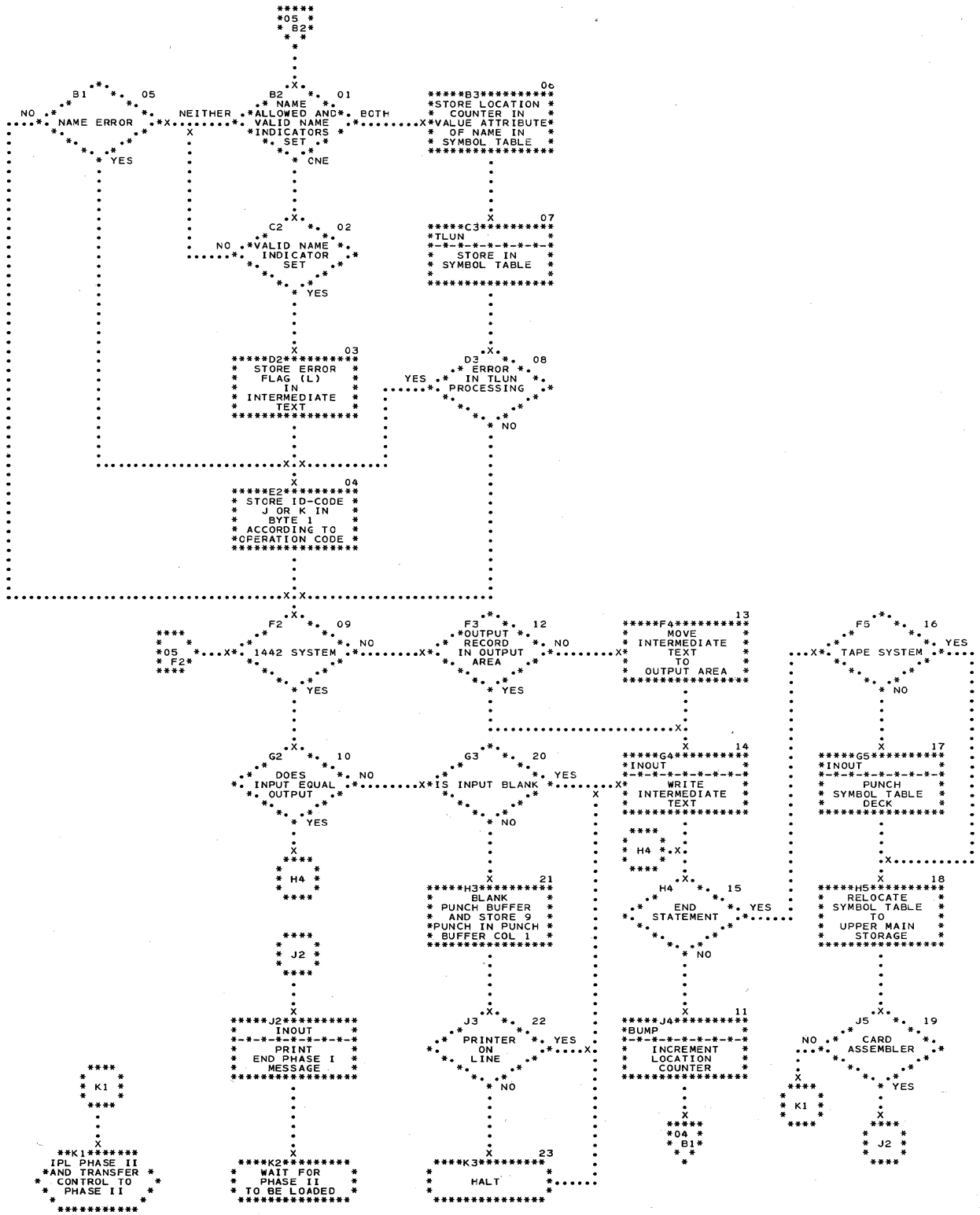


Chart 05. Phase I Control Routine (continued)

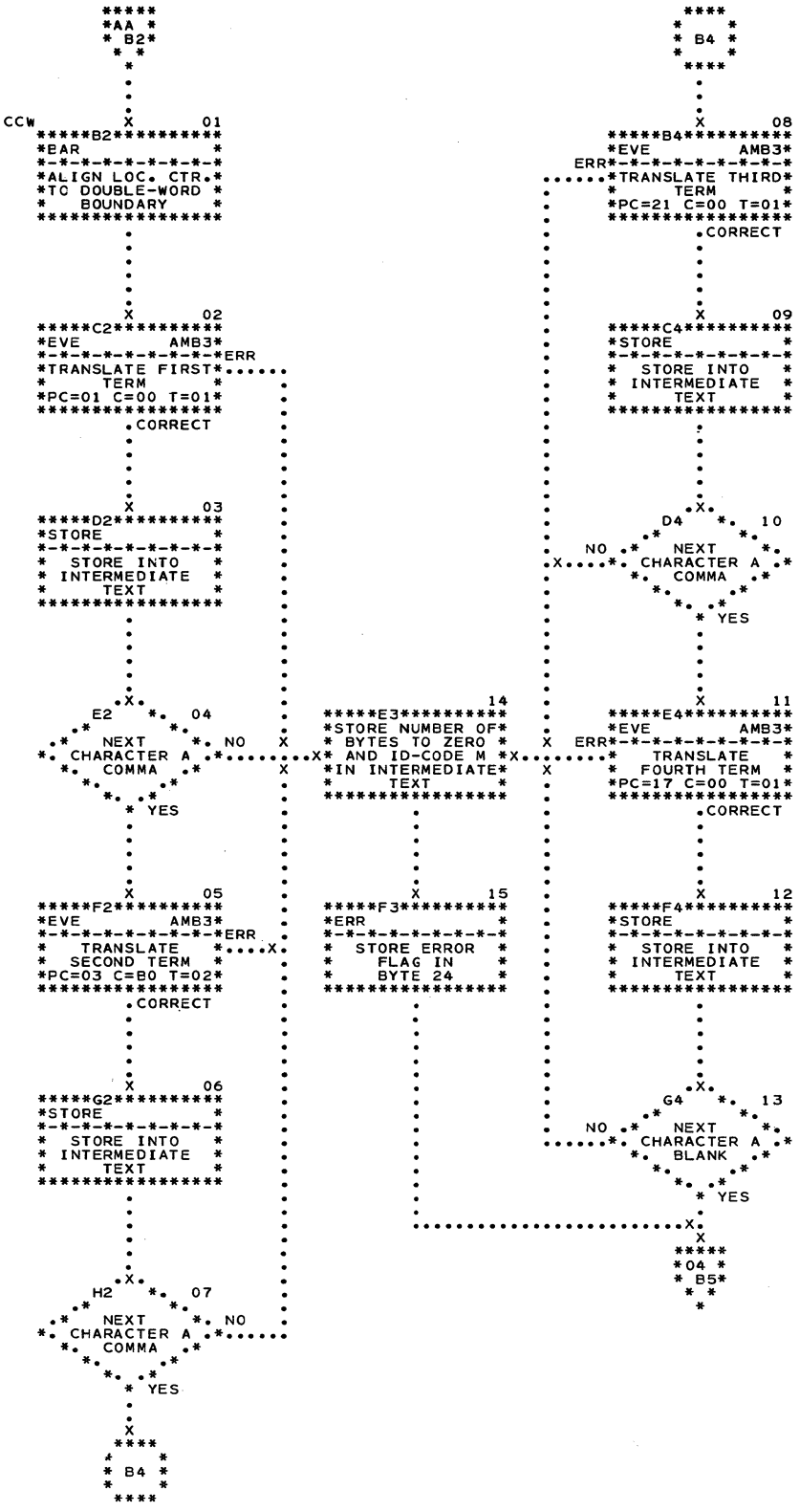


Chart AA. CCW Translation

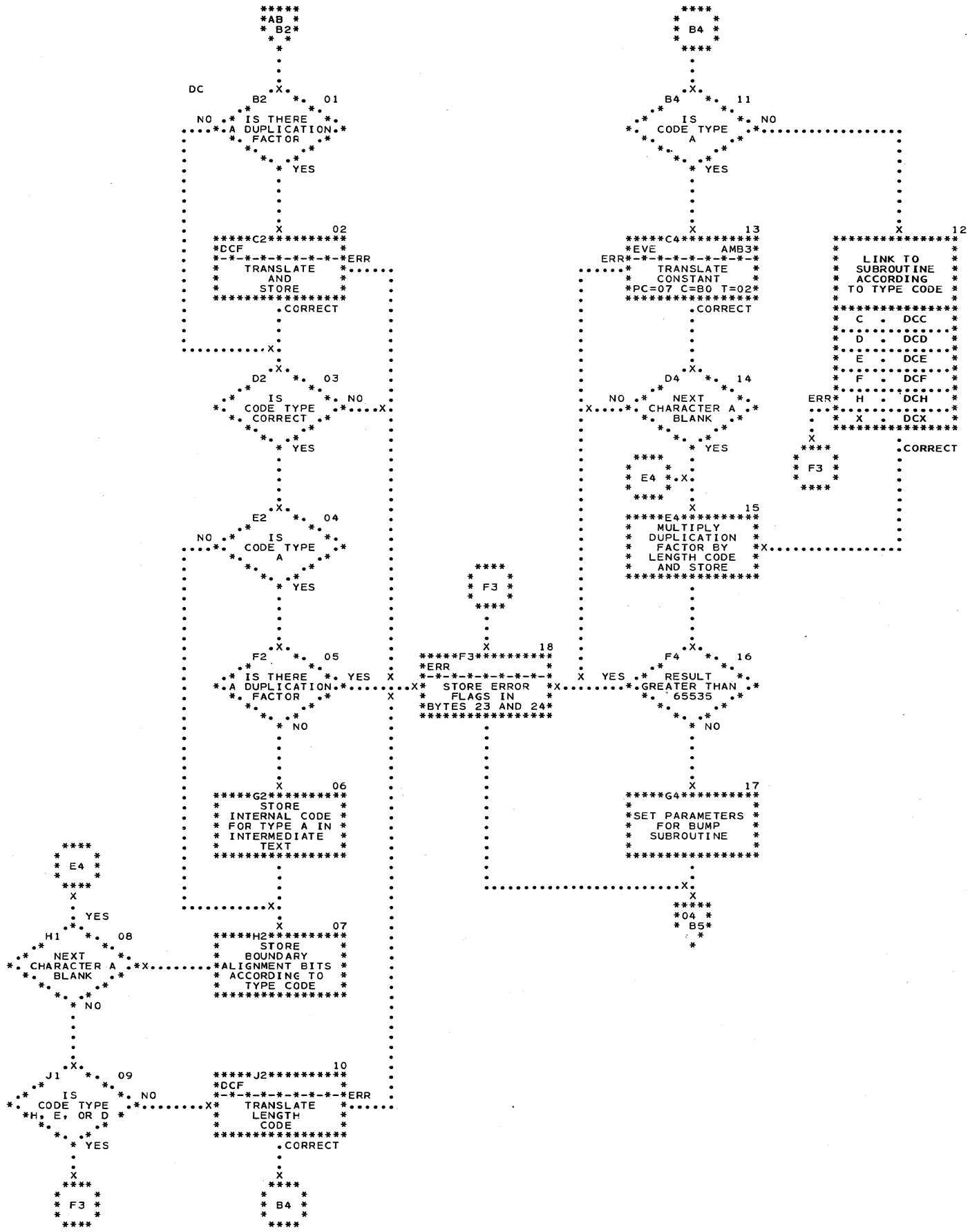


Chart AB. DC Translation

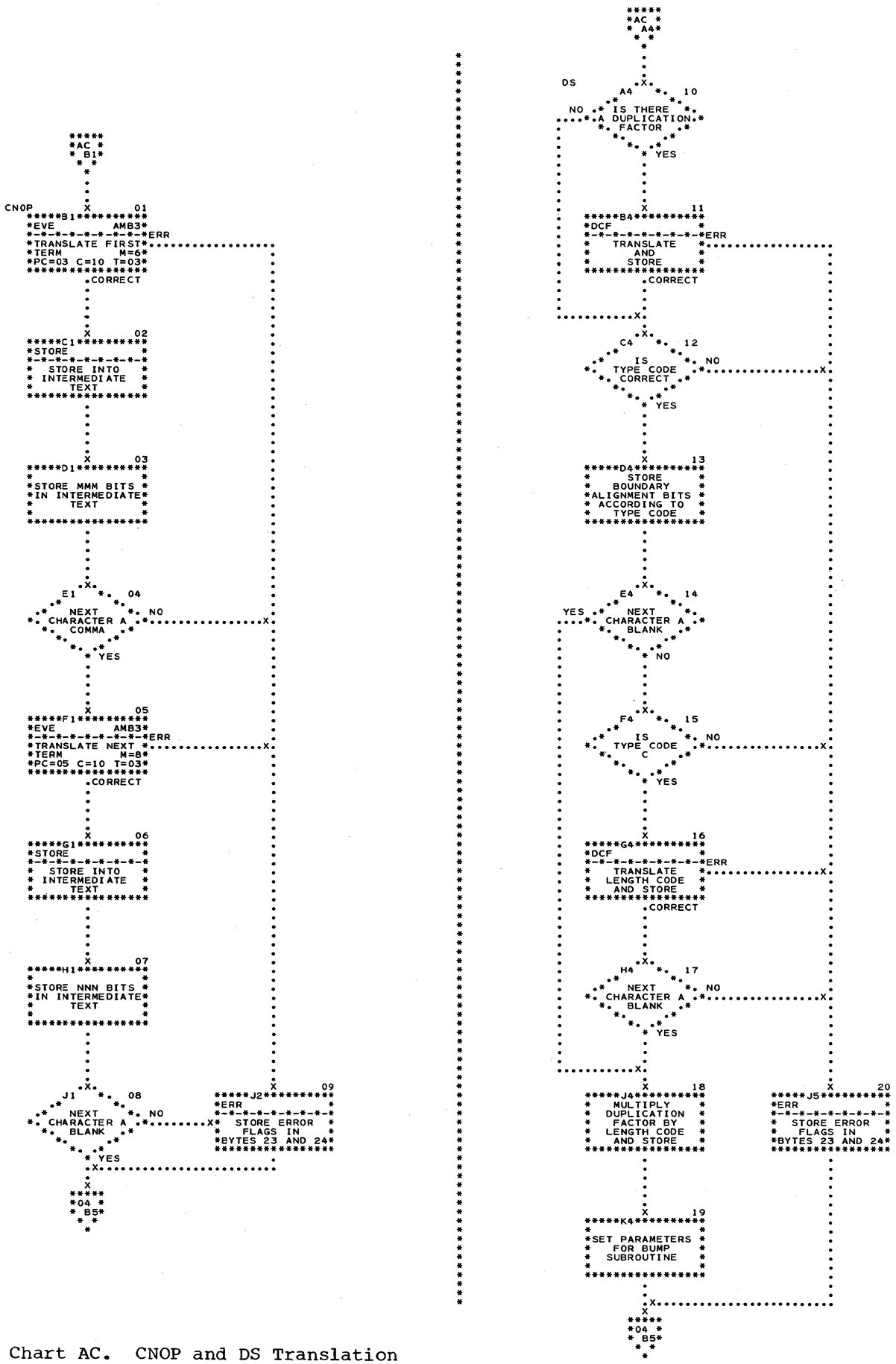


Chart AC. CNOP and DS Translation

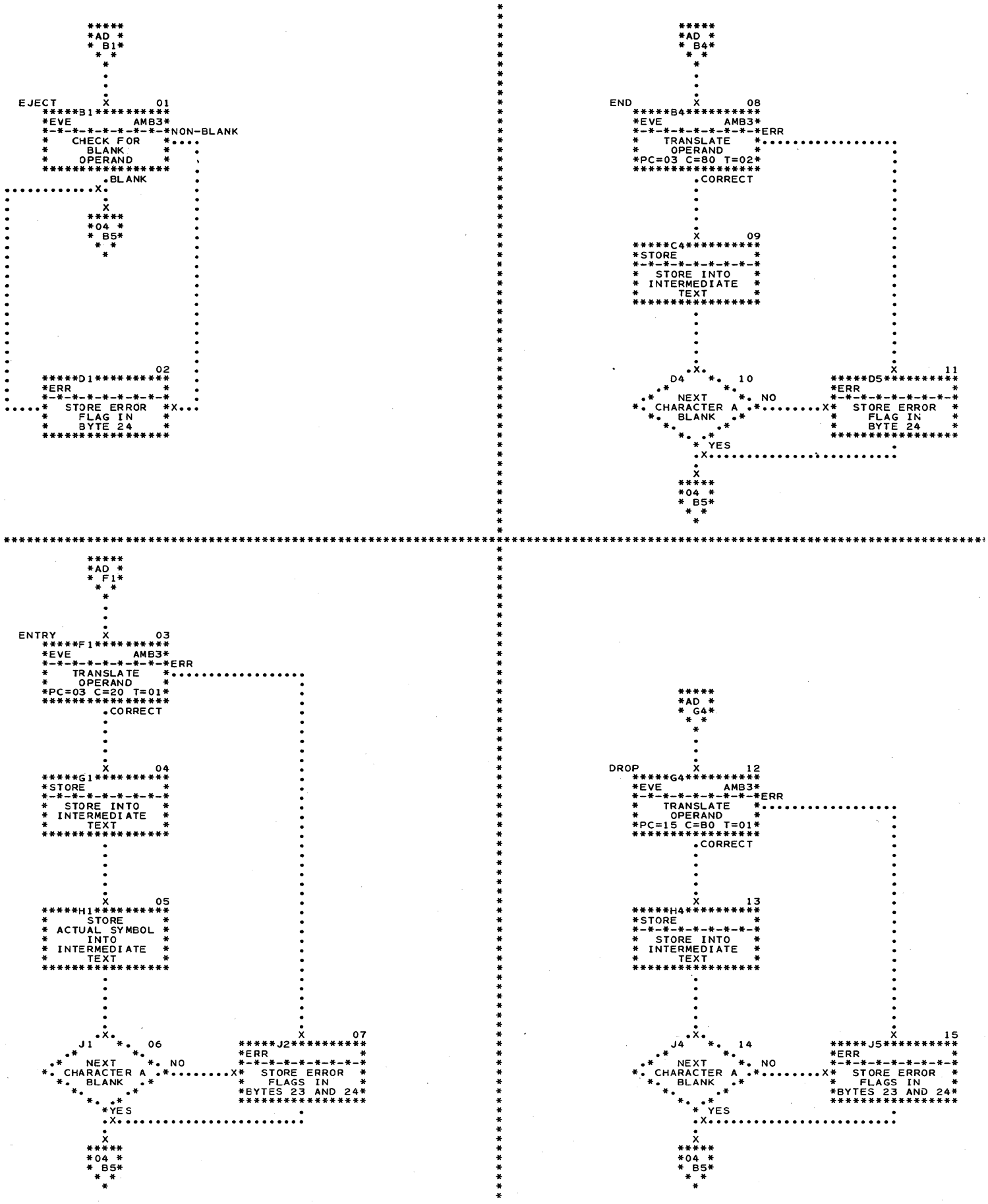


Chart AD. DROP, EJECT, END, and ENTRY Translation

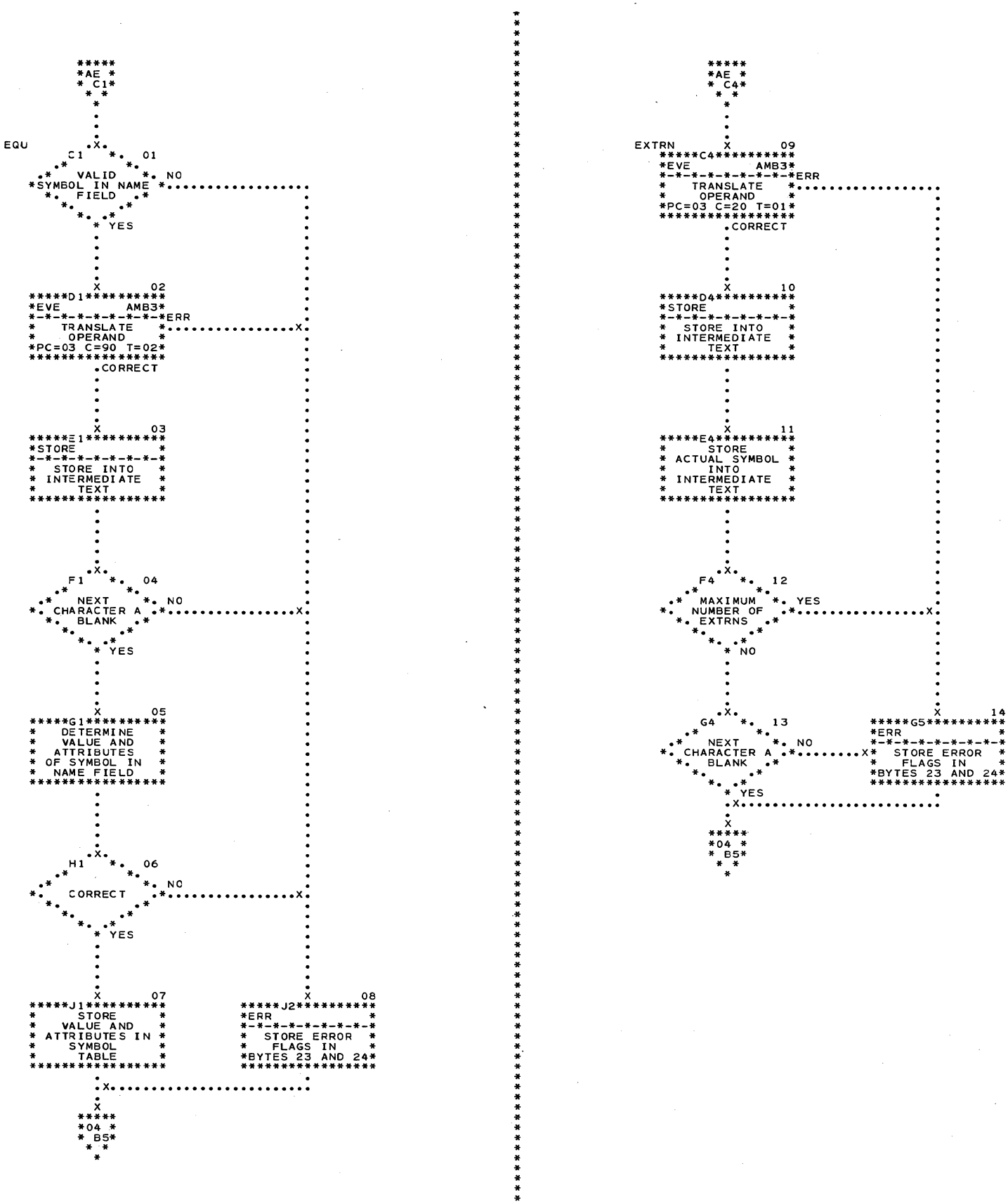


Chart AE. EQU and EXTRN Translation

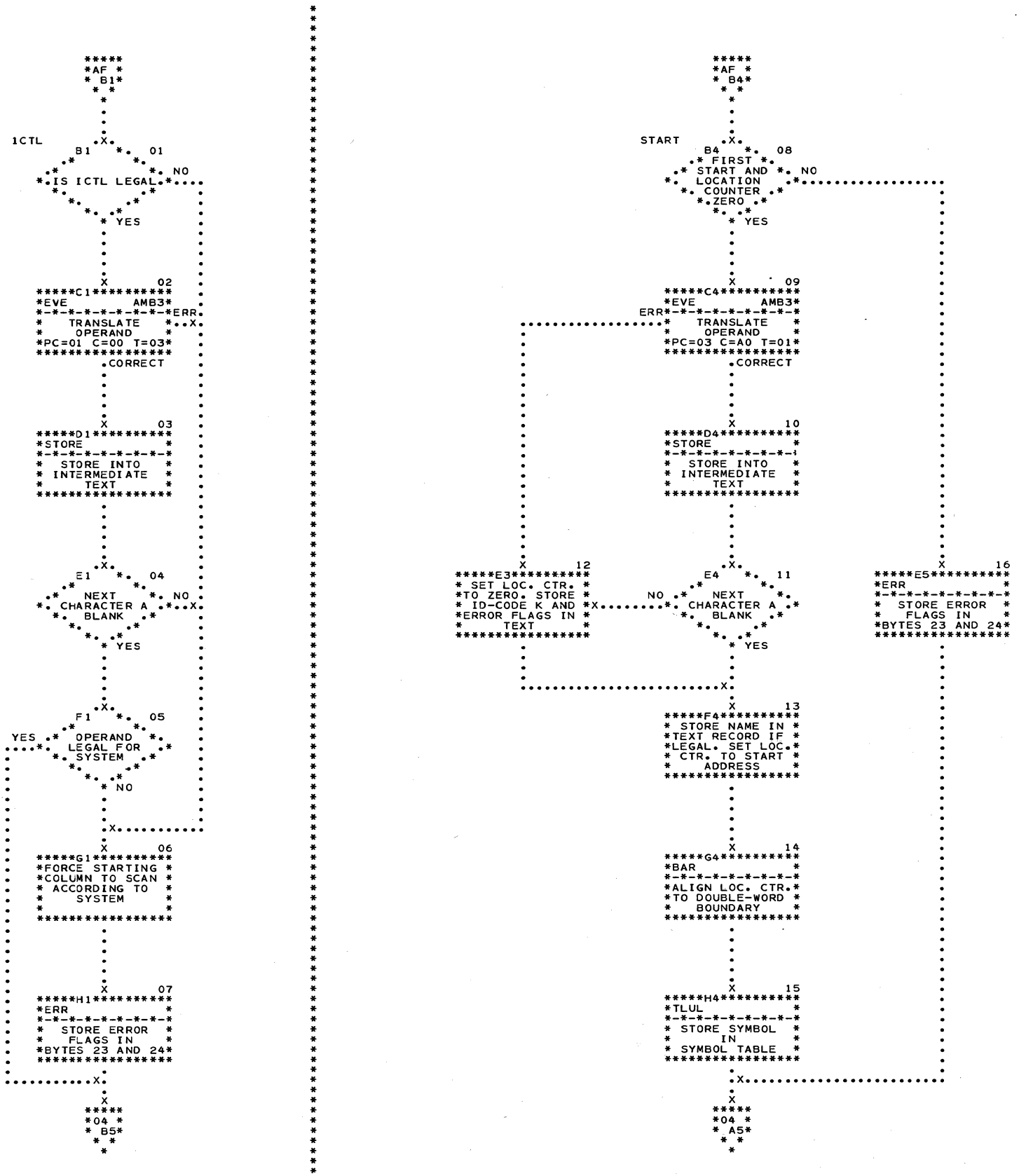


Chart AF. ICTL and START Translation

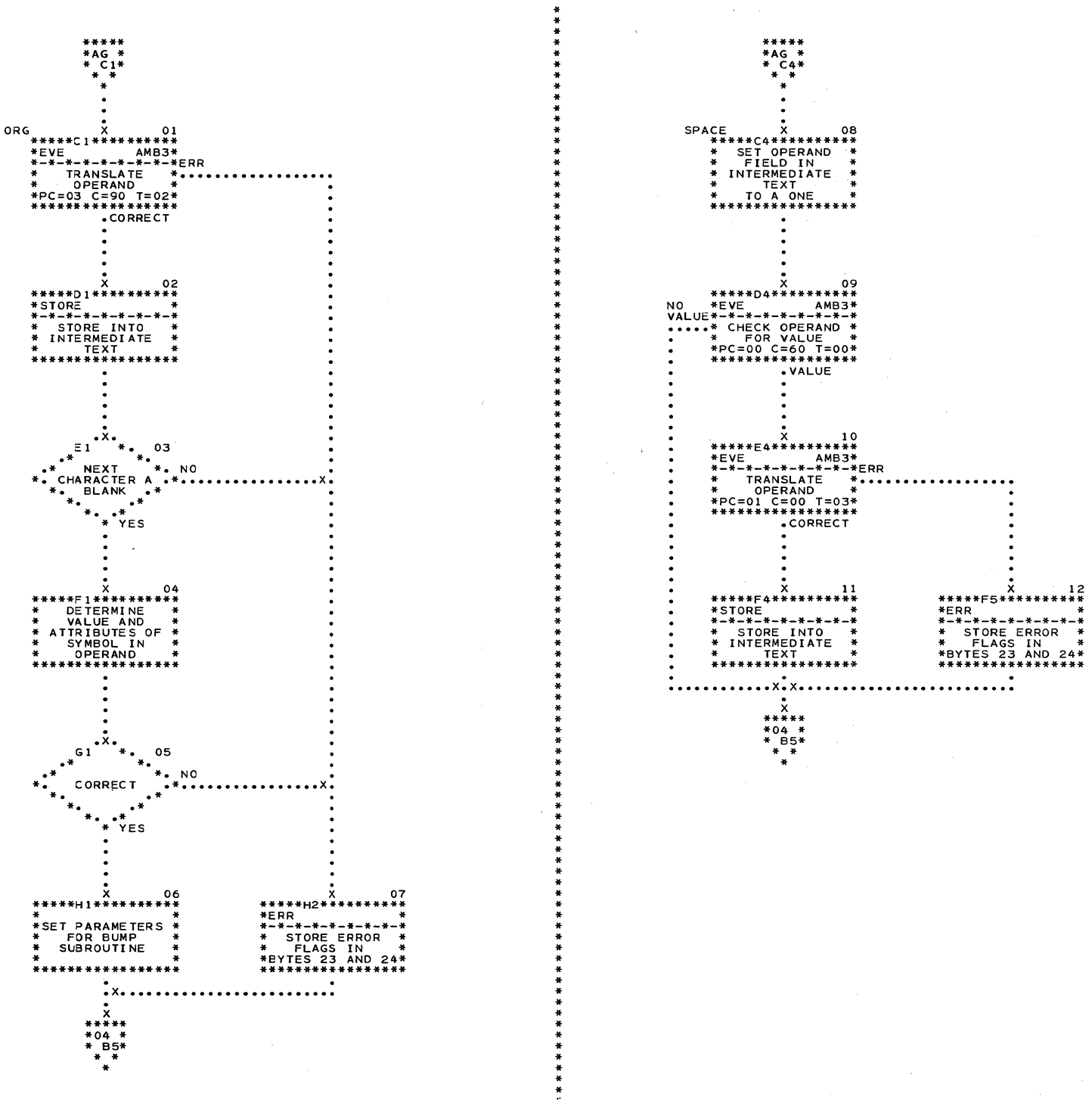


Chart AG. ORG and SPACE Translation

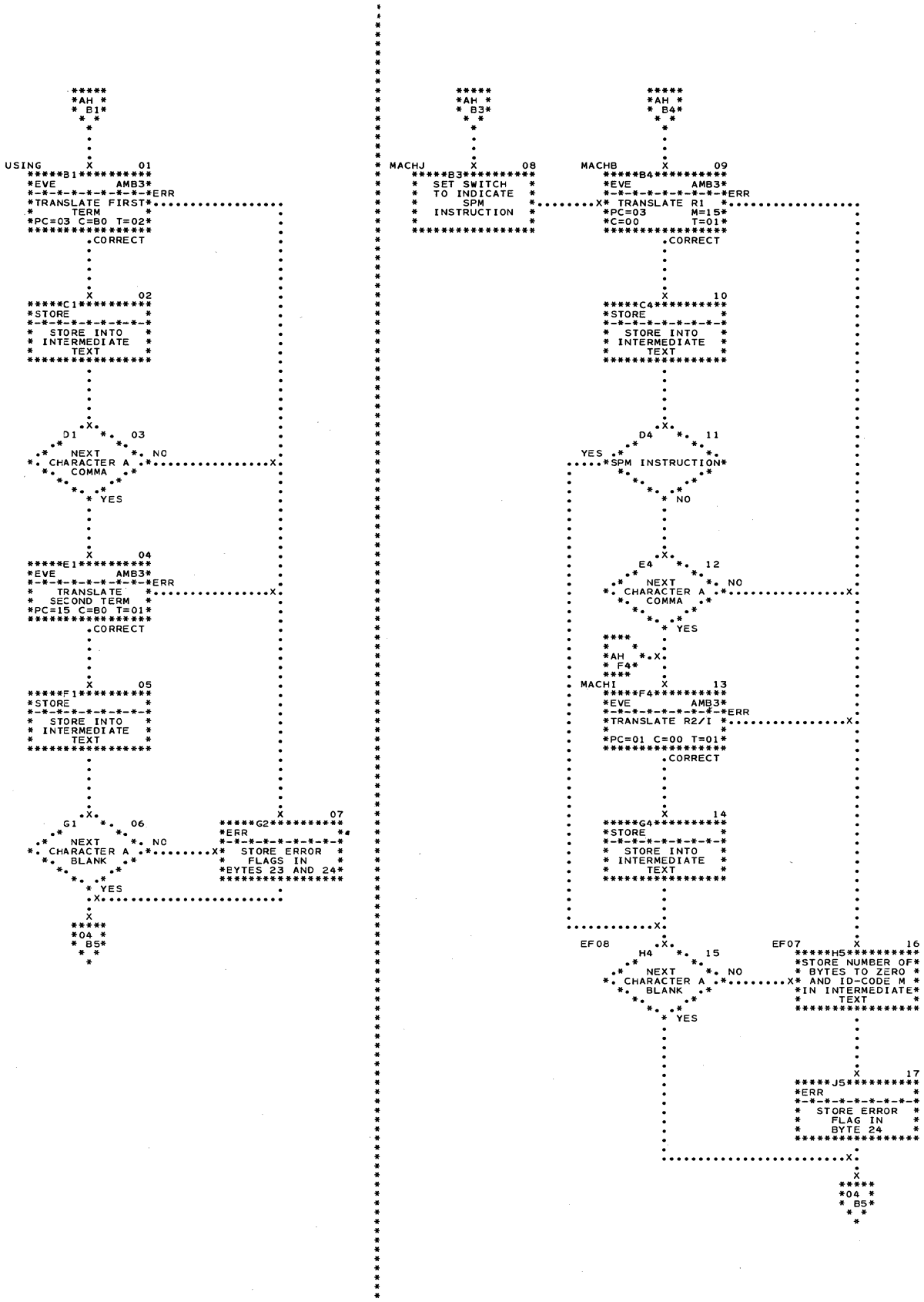


Chart AH. USING and RR Format Translation

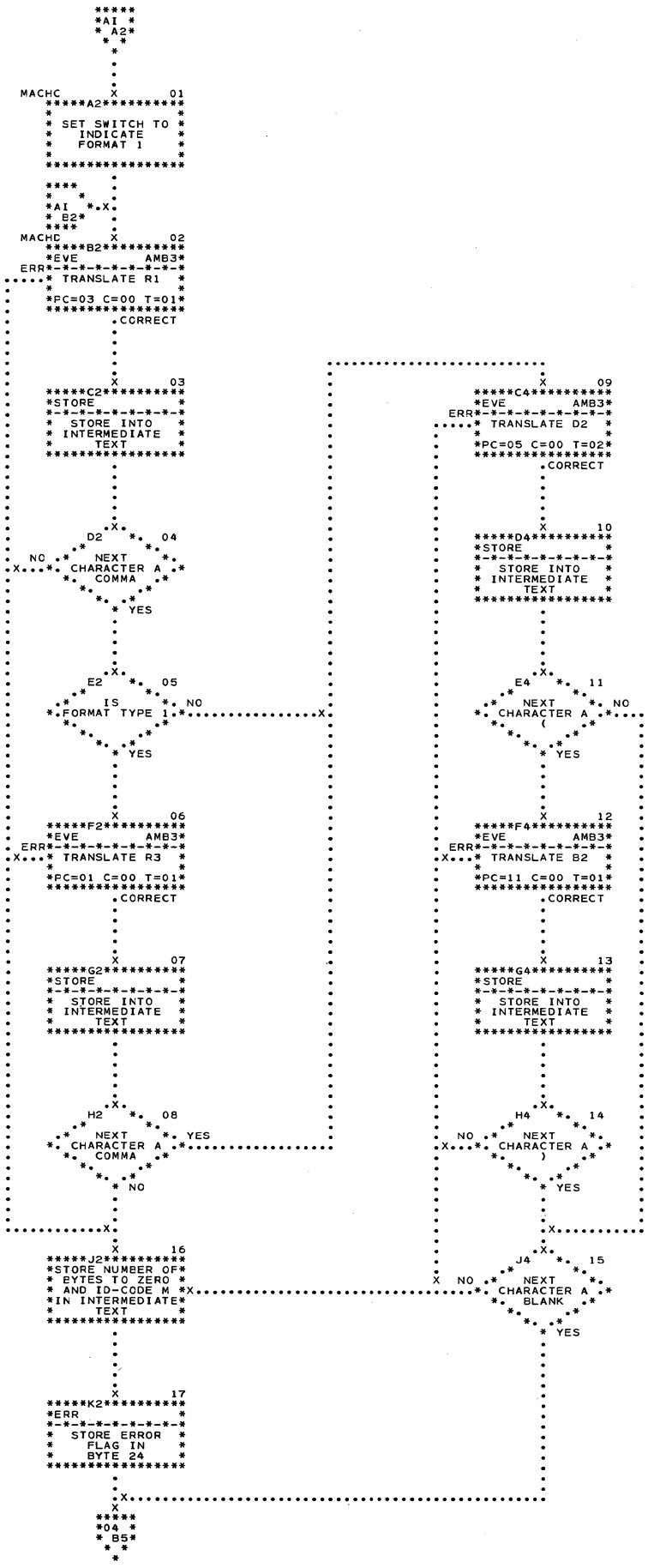


Chart AI. RS Format Translation

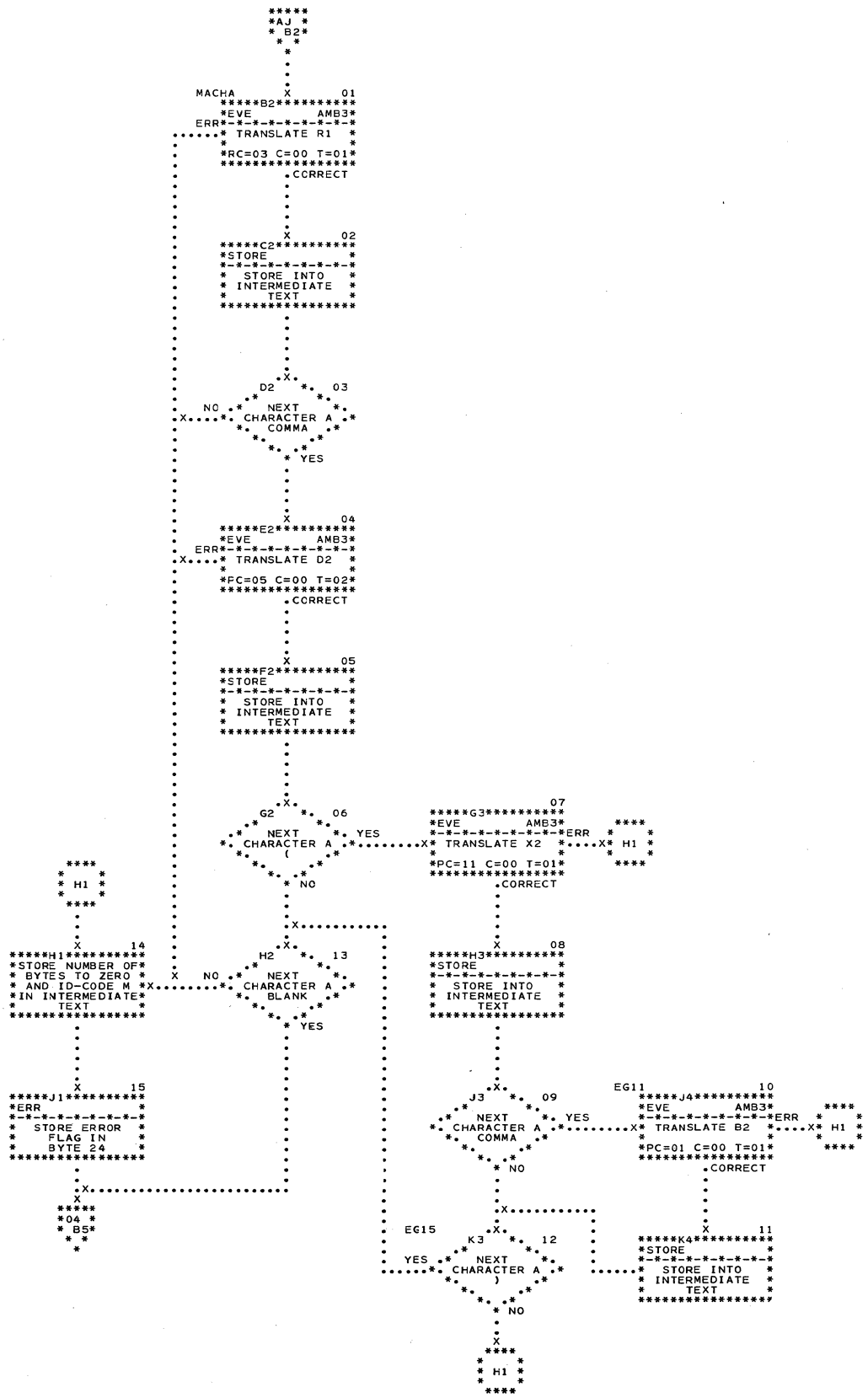


Chart AJ. RX Format Translation

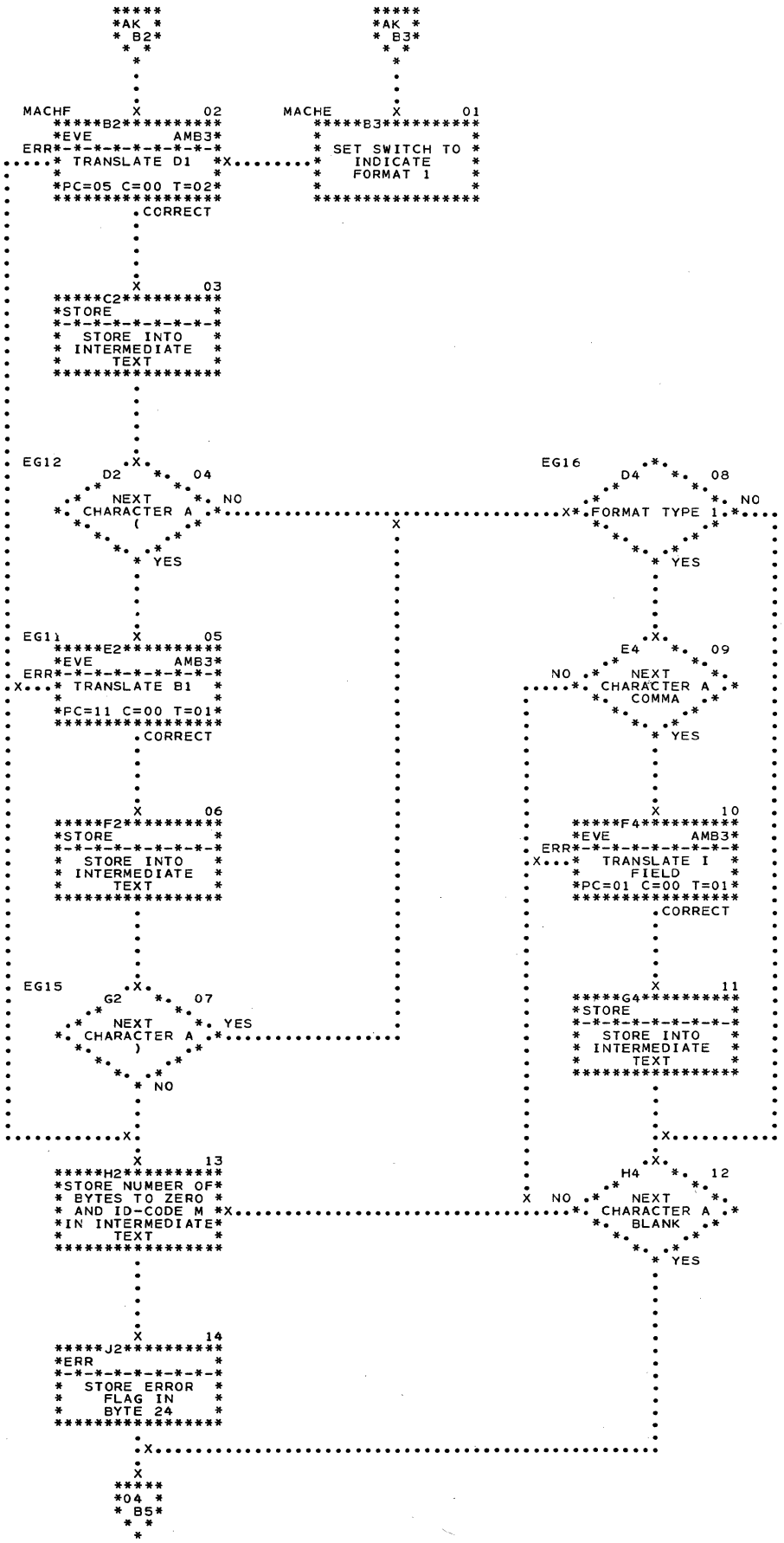


Chart AK. SI Format Translation

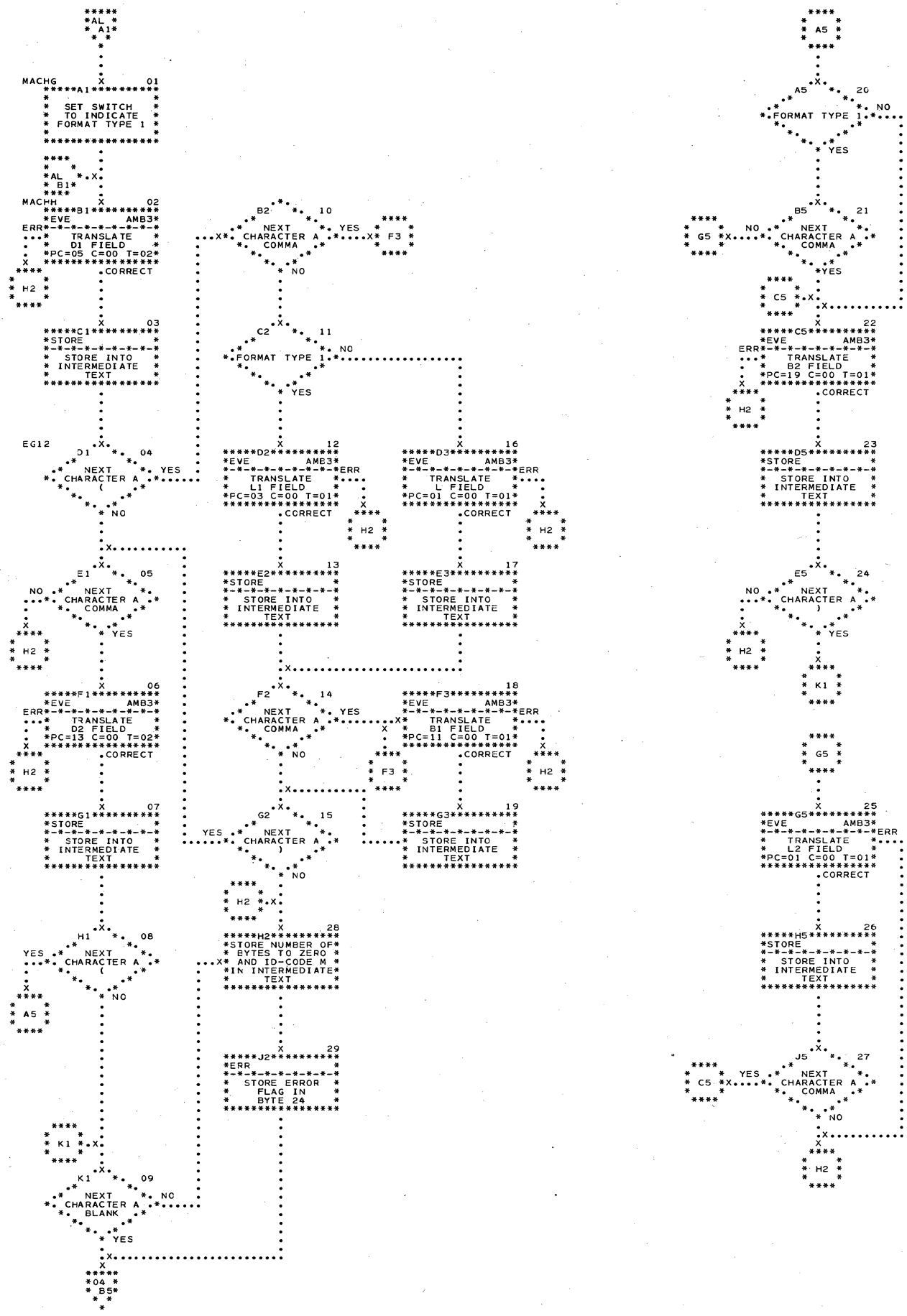


Chart AL. SS Format Translation

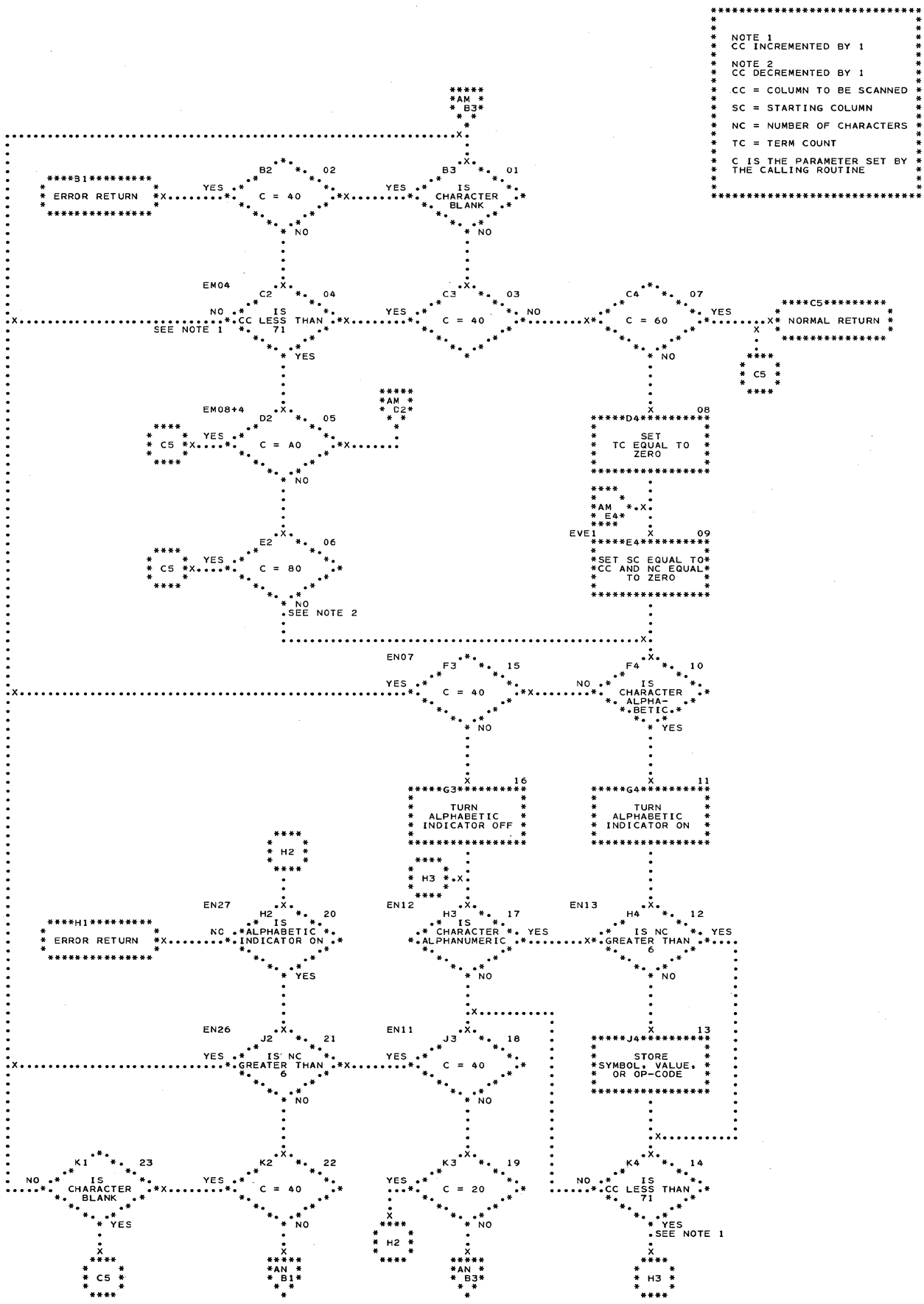


Chart AM. EVE Subroutine

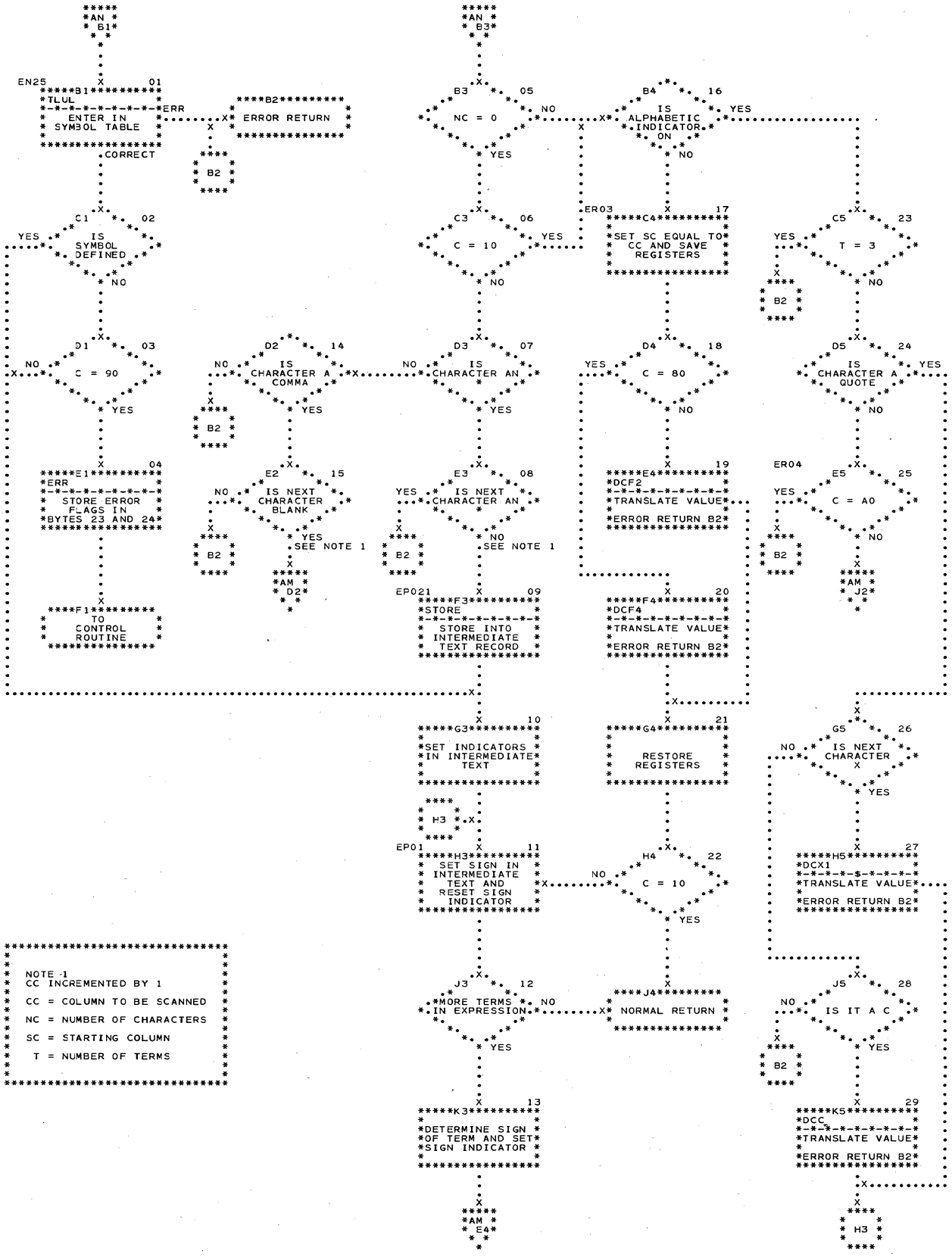


Chart AN. EVE Subroutine (continued)

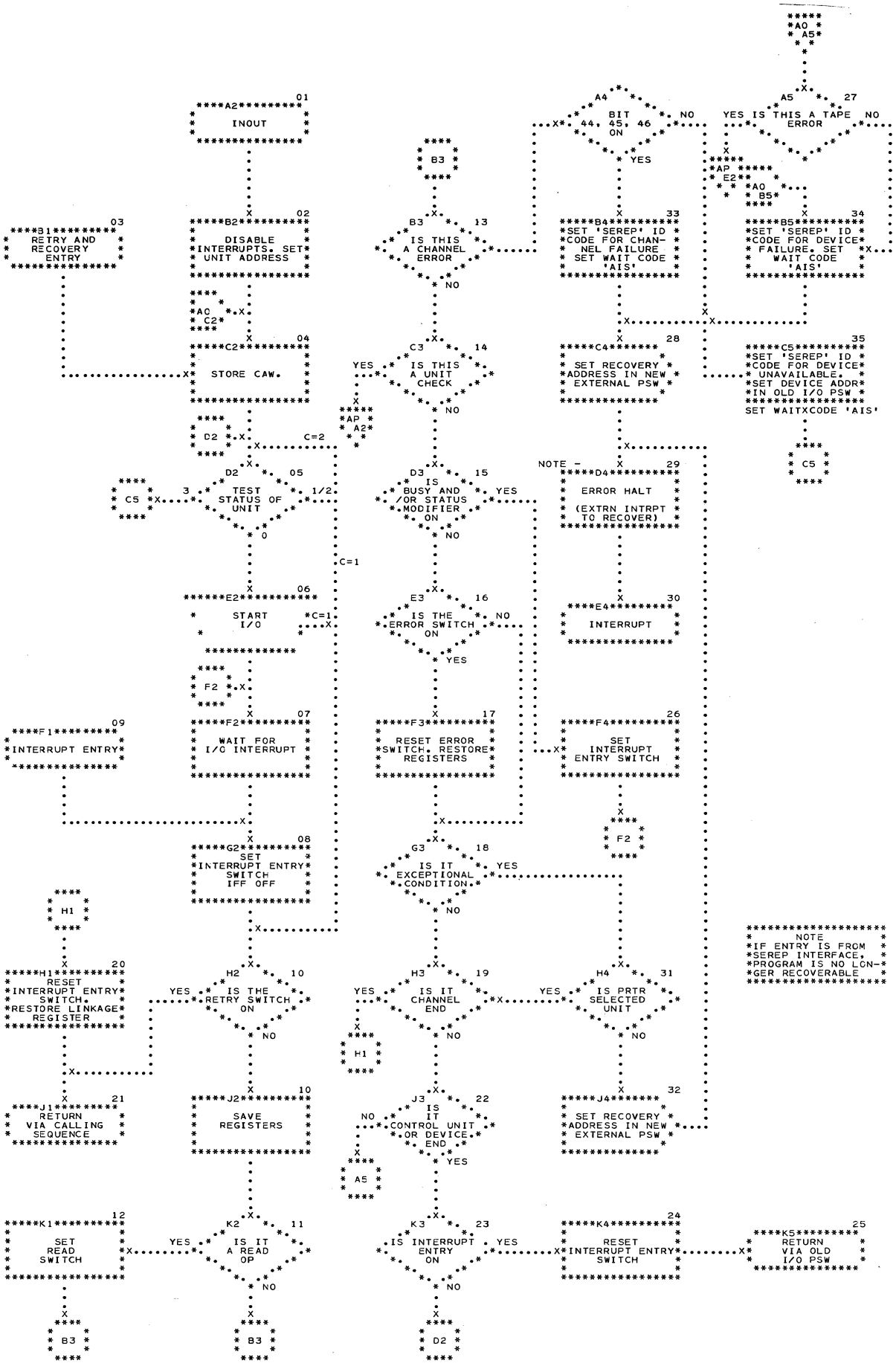


Chart AO. Input/Output Subroutine

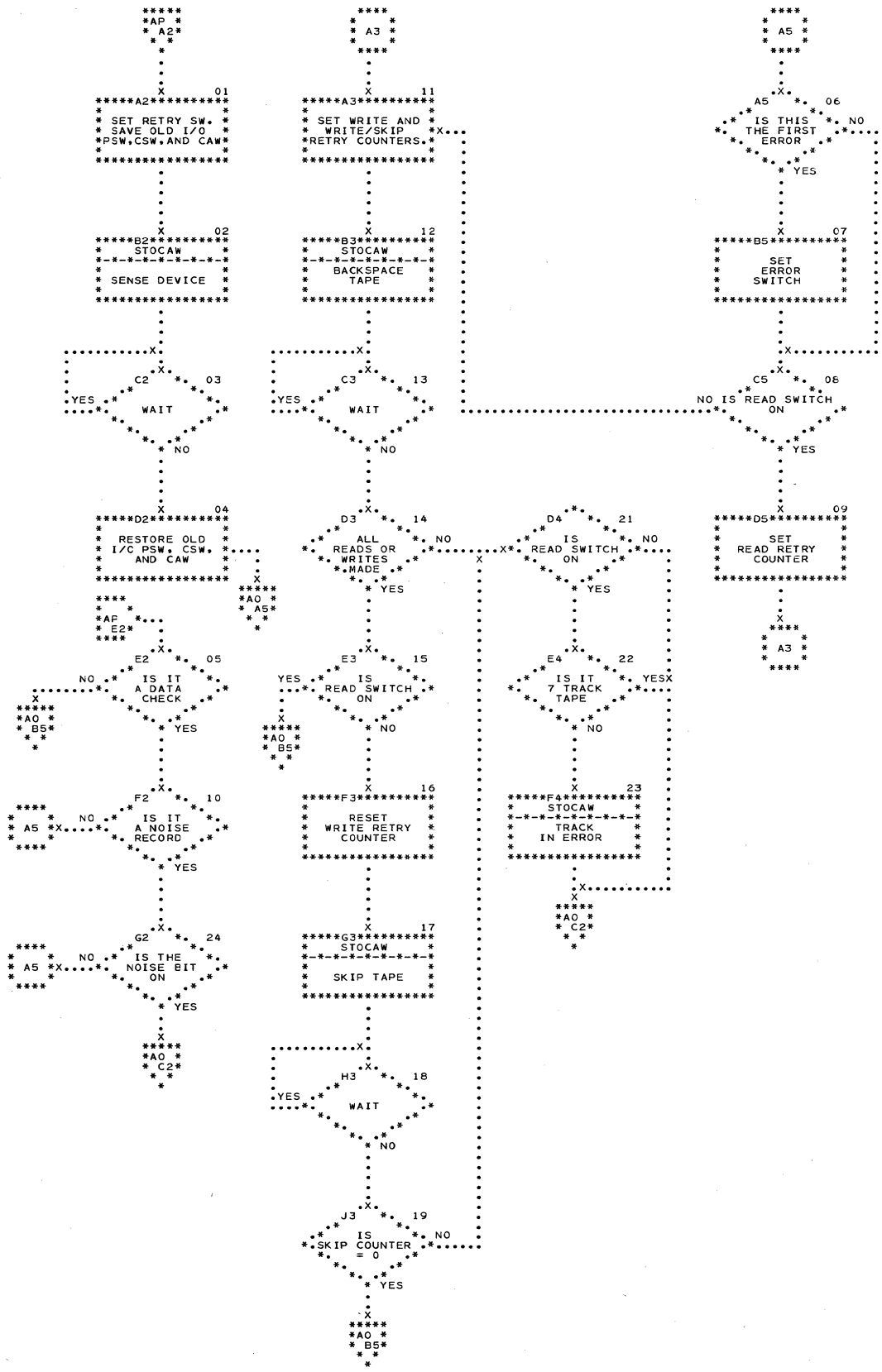


Chart AP. Tape Read/Write Retry Procedures

SECTION 1: INTRODUCTION

The major functions of Phase II of the Assembler Program are as follows:

Translate Intermediate Text into Machine Language

The Intermediate Text, which contains the partially translated source statements developed by Phase I, are translated into machine-language statements which make up the object program.

The symbols contained in the source statements are replaced with the actual machine addresses which can now be obtained from the Symbol Table developed by Phase I.

The type of translation that occurs depends upon the ID-Code in the Intermediate Text and upon the instruction format.

The operand field, part of Intermediate Text, is translated and placed in the output buffer area to be punched into a card or written on tape, and printed as a line in the object program listing.

The translation of each instruction format type is shown in Section 5, "Intermediate Text Translation." The subroutines used by Phase II to translate the Intermediate Text are discussed in Section 3, "Subroutine Description."

Produce Output Cards

The assembler produces three types of cards or card images on tape to be loaded as part of the object program. They are the ESD (External Symbol Dictionary), RLD (Relocation List Dictionary), and TXT (Text) cards. The format for each type of card is shown in Figure 10 at the end of this discussion.

The ESD cards are produced at the beginning of Phase II processing. One ESD card is punched for each START, EXTRN, and ENTRY Assembler instruction.

The TXT cards are produced during Phase II processing. When an output buffer entry becomes full, one text card is punched. Each TXT card may contain the text from several source statements.

The RLD cards are produced at the end of Phase II processing. The RLD cards are punched from the RLD Table (see Section 2, "Tables"). RLD cards are produced for each relocatable address constant and Channel Command Word defined in the source program.

ESD and RLD Cards: The ESD cards are produced to provide complete linkage between separate program segments. The RLD cards are produced to allow the Relocating Loader to reevaluate symbols and expressions within address constants and Channel Command Words in relocated segments. (For a description of the Relocating Loader, see IBM System/360 Basic Programming Support Basic Utilities, Form No. C28-6505.)

These cards become input to the Relocating Loader, which in turn creates an External Symbol Identification Table (ESID) and a Reference Table to provide communication between the ESD's and RLD's of various program segments. Collectively, the RLD's and ESDs are called the Control Dictionary.

When a START statement was encountered by the assembler, Phase I assigned the symbol in the statement the identifying number 1, which is entered into the Reference Table by the loader, along with the program segments relocation factor (the difference between the address where the segment was assembled and the address where it is loaded). Identifying number 1 was also assigned to the symbols appearing in the operands of all ENTRY statements in that assembly.

For EXTRN statements, the assembler assigned each external symbol an identifying number ranging from 2 through 15, indicating the order in which the EXTRN Assembler instructions were found in the source program. This number is placed in one of the Control Dictionary tables by the loader, along with a pointer to the Reference Table which includes the external symbol and its actual address (inserted from START and ENTRY ESD cards in other program segments). This allows the loader to implement cross-referencing.

The identifying number associated with the external symbol also appears in the RLD cards produced by the assembler. This allows address constants with either internal or external symbols to be properly evaluated.

TXT Cards: The object text to be loaded by the loader appears on the text cards. The TXT cards will contain a maximum of 56 bytes of information (columns 17-72). Therefore, each TXT card may contain the text from several source program statements. If less than 56 bytes are specified for one TXT card, the remaining columns of the card are blank.

The text is produced in the order in which it is declared by the source program. TXT cards, therefore, are not necessarily in order by origin (tentative loading address). For example, an ORG statement may cause the program to skip a block of addresses and later in the source program these addresses may be referenced by another ORG statement.

A TXT card is terminated and a new one begun whenever there is a break in the string of specified storage content (boundary alignment excluded), whenever a total of 56 bytes of continuous text has been accumulated, or whenever a DC Assembler instruction with a duplication factor is encountered. An ORG statement, whether it results in a change of location counter value or not, is considered to be a break in the string of specified storage content. Any DS Assembler instruction (except one with zero length and no boundary alignment needed) is also considered to be a break.

Boundary alignment caused by an instruction being coded at an odd byte boundary, a CNOP coded at an odd byte boundary, a DC Assembler instruction without a duplication factor, or a CCW Assembler instruction, will not cause a new TXT card to be started. Instead, zeros will be inserted into the bytes that are skipped to perform the alignment.

A DC instruction with a duplication factor will cause a new TXT card to be produced after the necessary alignment is done. The new TXT card, and those following, will contain a multiple of the bytes that are to be repeated for as long as the bytes fit on the card, or until all desired data is put on cards. The last card, whether full or not, will not be used for any other text. The text developed from the statement following the DC statement will be put on the following card.

For example, the source statement DC 5CL16'A' will produce the following cards:

Card 1

1 16	17-32	33-48	49-64	65-80
Control information	A	A	A	

Card 2

1 16	17-32	33-48	49 - 80
Control information	A	A	

Card 3

1 16	17 - 72	73-80
Control information	Text from the following statements	

Card Formats: The formats of the different types of cards produced by Phase II of the Assembler Program are shown in Figure 10. The ESD and TXT card formats are self-explanatory. The RLD card format is described in detail in the discussion that follows.

The position header, which is two bytes long, is the relocatability attribute (external symbol identification) assigned by the assembler to the program segment in which the constant is being defined.

The relocation header, which is two bytes long, is the relocatability attribute that was assigned to the address symbol when it was defined in the operand field of an EXTRN Assembler instruction. It is the same as the external symbol identification assigned to the symbol in Phase I.

The address is the value of the relative assembled address of the relocatable constant within the program segment indicated by the position header. In the case of a CCW, the address constant is the value of the location counter plus one. This address points to the second field of a CCW.

All RLD's with the same position and relocation headers will be grouped, in the order found, into one card image.

The flags are as follows:

1. Complement Flag - this one-bit flag indicates to the loader whether the relocation constant for the program segment indicated by the relocation header need be added or subtracted from the address constant.
2. Continuation Flag - a 1 indicates that more addresses with this relocation and position header are on this card; a 0 indicates that this is the last address with this relocation and position header (another relocation and position header could follow on this card). This flag occupies one bit.
3. Size - this two-bit flag indicates the size of the symbol (10 - three bytes; 11 - four bytes).

EXTRN[1-2]

1	2 3 4	5 -10	11 12	13 14	15 16	17 - 22	23-24	25	26 - 28	29 - 80
12 2 9	E S D		1 6		ESID no.	Symbol from operand		2	Address of symbol	Blank

ENTRY[1-2]

1	2 3 4	5 -10	11 12	13 14 15 16	17 - 22	23-24	25	26 - 28	29 30	31 32	33 - 80
12 2 9	E S D		1 6		Symbol from operand		1	Address of symbol		ESID no.	Blank

START[1-2]

1	2 3 4	5 -10	11 12	13 14	15 16	17 - 22	23-24	25	26 - 28	29	30 31 32	33 - 80
12 2 9	E S D		1 6		ESID no.	Symbol from operand		0	Address of symbol		No. of bytes in program	Blank

END[1]

1	2 3 4	5	6 - 8	9 -14	15 16	17 - 80
12 2 9	E N D		Blank or address to transfer to		ESID no.	Blank

TEXT[1]

1	2 3 4	5	6-8	9 10	11 12	13 14	15 16	17 - 72	73 - 80
12 2 9	T X T		Start Addr.		Bytes on card		ESID no.	Text (one byte per column)	Blank

RLD[1-3]

1	2 3 4	5 -10	11 12	13 14 15 16	17 - 20	21	22 - 24	25	26 - 28	29 - 72	73 - 80
12 2 9	R L D		Bytes on card		Position and Relocation hdrs.	FLAGS	Address	FLAGS	Address	(see note 3)	Blank

[1] All cards are produced in extended card code. It will be necessary to convert these records from extended card code to get the hexadecimal values.
[2] Columns 11 and 12 on the ESD cards will contain extended card code punches 12-0-1-8-9 and 12-11-1-8-9, respectively (decimal 16).
[3] These bytes are the same as bytes 21-24 and 25-28 until new position and relocation headers are required. Then bytes 17-20 will be repeated with the new values.

Figure 10. Output Formats

Produce Object Program Listing

The object program listing produced by Phase II has the format shown in Figure 11, part B. The only exception will be comments statements, which are reproduced on the listing exactly as the source program statements appear.

For example, if the first statement of a program is as shown in Figure 11, part A, and the program is to be assembled starting at location 80 (specified by the START Assembler instruction), the line on the object program listing would appear as indicated in Figure 11, part C. If the statement contained an error, the error flags would be printed to the left of the location counter.

Although the source program statement will produce one line of listing, several source statements may be reproduced on a TXT card, as indicated in the previous discussion.

Name	Operation	Operand	Comments	ID-Seq.
SETBAS	BALR	14,0	LOAD BASE REG 14	1080

A. Input Statement

ERROR FLAGS	LOCATION COUNTER	ASSEMBLED MACHINE LANGUAGE STATEMENT	SOURCE PROGRAM STATEMENT

B. Listing Format

```
000080 05 E0     SETBAS BALR 14,0  LOAD BASE REG 14  00001080
```

C. Print Line

Figure 11. Object Program Listing Example

REGISTER TABLE

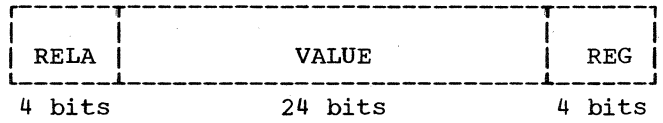
The Register Table contains the general register numbers, their corresponding values, and relocatable attributes. An entry is made into this table each time a USING Assembler instruction is encountered. An entry will be deleted from the table each time a DROP Assembler instruction is encountered or when a USING statement uses a previously entered register.

Note: Using Register zero will not drop Register zero with an absolute attribute. Register zero is placed into the table by the assembler and remains in the table.

The Register Table is referenced whenever it becomes necessary for the assembler to compute addresses (i.e., assign base registers and displacements).

The values assigned to the registers are relocatable.

Each entry in the table occupies a single 32-bit word and has the following configuration:



where:

RELA indicates the relocatability value (0-15)

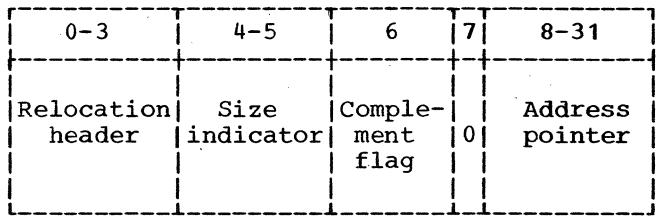
VALUE is the value of the expression in the operand field of the USING statement

REG indicates the general register that is assigned the above value

RELOCATION LIST DICTIONARY (RLD) TABLE

The RLD Table is constructed to hold the data necessary for producing RLD cards or tape records at the conclusion of Phase II processing (see Introduction to Phase II for description of the RLD cards).

Entries are made into the RLD Table for three- or four-byte relocatable address constants and Channel Command Words. Each entry in the RLD Table occupies a single 32-bit word and has the following configuration:



Each of these entries is discussed in the Phase II Introduction under the RLD card.

At the end of Phase II, the RLD Table is scanned, and the entries having the same position and relocation headers are grouped into a single card image for producing an RLD card or tape record. Error flag K is indicated on all address constants and Channel Command Words in excess of the number allowed for the RLD Table.

SECTION 3: SUBROUTINE DESCRIPTION

This section describes the subroutines of the Assembler Program that are used during Phase II processing.

SER - Simple Expression Translation Routine

The SER subroutine is used to translate a field in the Intermediate Text that contains a simple expression (i.e., the R1 field of an RR format machine instruction, the operand field of a DROP Assembler instruction, etc.), and to translate each term of a compound expression.

This subroutine is entered with the following parameters, which are set by the calling routine:

COL indicates the first column of the Intermediate Text to be scanned,

LGH indicates the number of four-bit fields used in the output buffer area,

TXT indicates the rightmost bit in the output buffer area into which the translated expressions are to be placed,

NBT indicates the number of bytes of input to be translated.

After the field is translated, the output parameter RELA will be set to indicate whether the expression is absolute or relocatable. RELA will have one of the following values:

0 the expression is absolute,
1-15 the expression is relocatable.

The field of the Intermediate Text indicated by the above parameters is checked for a self-defining value or a symbol. (Fields 1, 3, and 4 of a CCW instruction are checked for absolute values. If one of the fields is not absolute, error flag R is placed in the print buffer and the entire output area for the CCW is reset to zeros.) Depending upon the value, one of the following procedures plus the Common procedure is followed.

SYMBOL: If the value is a symbol (P bit of the Intermediate Text a 1), the defined bit of the Symbol Table entry is checked. If the symbol is undefined, the procedures described under Undefined are followed. If the symbol is defined, the procedures described under Defined are followed.

Undefined: Error flag U is placed in the print buffer. One of the following procedures is then followed, depending upon the instruction type. For instruction types not indicated, no further action is taken and control is returned to the calling routine.

Machine Instruction: The operand field of the output area is set to zero. The operation code remains unchanged. Control is returned to the Phase II Control Routine.

CCW or DC Assembler Instruction: The entire output area for the instruction is set to zero. Control is returned to the Phase II Control Routine.

DROP, USING, EXTRN, or ENTRY Assembler Instruction: Error flag N is placed in the print buffer along with error flag U. Only the source statement and the error flags will be printed. Control is returned to the Phase II Control Routine.

Defined: The symbol value and the relocatability attribute (RELA) are placed in a temporary output area, and the procedures described under Common are then followed.

SELF-DEFINING VALUE: RELA is set to zero since a self-defining value is absolute, and the absolute value is placed in a temporary output area. The procedures described under Common are then followed.

COMMON: If the translation of the field is not complete (parameter TXT not equal to zero), control is returned to the calling routine. If the translation is complete (TXT equal to zero), the output buffer is checked to see if the value will fit into the buffer.

If the value will fit, it is stored in the output buffer by subroutine STXT, and control is returned to the calling routine.

Note: If the value is the L field of an SS format instruction, the value is reduced by one before it is stored. L with a value of zero is processed the same as an L field with a value of one. The value of zero is not reduced.

If the value will not fit, error flag T is placed in the print buffer, and one of the following procedures is used.

Machine Instruction: If the value is a D1 or D2 field and is absolute, the value is truncated and stored in the output buffer. For all other fields, the operand field of the output buffer for this instruction is set to zero. The operation code, however, remains unchanged. Control is returned to the Phase II Control Routine.

CCW Assembler Instruction: The entire output buffer for this instruction is set to zero and control is returned to the Phase II Control Routine.

USING and DROP Assembler Instructions: The entire statement is ignored and control is returned to the Phase II Control Routine.

Other Assembler Instructions: The translated value is truncated and then stored in the output buffer by subroutine STXT. Control is returned to the calling routine.

When control is returned to the Phase II Control Routine by subroutine SER, the remaining fields of the Intermediate Text for that instruction, if any, are not translated.

EER - Compound Expression Translation Routine

This subroutine is used to translate the fields of the Intermediate Text that may contain a compound expression (i.e., D2 field of a machine instruction, second expression of a CCW Assembler instruction, etc.). The parameters set by the calling routine are the same parameters that are set for the SER subroutine (see preceding discussion).

The compound expression may contain a maximum of three terms. Each term is translated by subroutine SER. After each term has been translated, several tests are made to determine the validity of the terms.

The following conditions must be met:

1. If one relocatable term is used, it cannot be preceded by a minus (-) sign.
2. If two relocatable terms are used, one of the two terms must be minus (-), and the value of RELA for both terms must be equal.
3. If three relocatable terms are used, there may be only one minus (-) sign, and the value of RELA for all three terms must be equal.
4. If the sign of a term is multiply (*), the terms preceding and following the sign must be absolute.

If these conditions are met, the procedures described under Common are followed.

If these conditions are not met, one of the following procedures is used, depending upon the instruction type. Only the instructions that may have a compound expression in the operand field are indicated.

Machine Instruction, CCW or DC Type A Assembler Instruction: Error flag A is placed in the print buffer, the output buffer for the instruction is set to zero (the operation code remains unchanged for a machine instruction), and control is returned to the Phase II Control Routine.

USING Assembler Instruction: Error flags A and N are placed in the print buffer. Only the source statement and the error flags are printed in the object program listing. Control is returned to the Phase II Control Routine.

END Assembler Instruction: Error flag A is placed in the print buffer, and control is returned to the calling routine.

COMMON: The value of RELA is set to the relocatability attribute of the symbols, and the terms are combined according to their signs. The result is stored in a temporary save area.

The only negative answer that is allowed is for a DC type A instruction with a RELA value of zero. If the answer is negative for any other instruction type, or RELA is not zero for a DC type A, error flag Y is placed in the print buffer.

If the answer is positive, or after error flag Y is placed in the print buffer, one of the following procedures is used.

Machine Instructions: If a base register is not indicated, the values of the base register and displacement are obtained from the Register Table by subroutine MARS and stored in the output buffer. Control is then returned to the calling routine.

If a base register is indicated, the value of the displacement should be absolute. If it is not, error flag R is placed in the print buffer before the displacement is tested to see if it is equal to, or less than, 4095, the maximum value allowed.

If the displacement is too large, error flag T is placed in the print buffer, the value is truncated and stored in the output buffer by subroutine STXT, and control is returned to the calling routine.

If the displacement is correct, the value of the combined terms is stored in the output buffer, subroutine SER is entered to translate the base register field, and control is returned to the calling routine.

Assembler Instructions: The value of the combined terms is placed in the output buffer by subroutine STXT, and control is returned to the calling routine.

SWPT Setting	Meaning	When Used
0	Normal - print location counter, assembled data and source statement	For all correctly assembled instructions
1	Duplication - print location counter and assembled data	DC instruction with a duplication factor or a non-zero CNOP
2	Zero fill - print location counter and assembled data (zeros)	Location counter not aligned to a half-word boundary for a machine instruction
4	Comments - print source statement only	Non-blank card with an incorrect ID-Code CNOP zero USING with an absolute value A relocatable value preceded by a minus sign EJECT, ENTRY, EXTRN, SPACE
8	Print location counter and source statement	DS, START
16	Internal repeat switch	Assembler instruction with a size greater than 8
32	Print new location counter and source statement	END, EQU, ORG
64	Print value (USING) or register number (DROP)	USING, DROP
128	Space control (no printing)	SPACE

Figure 12. Print Control Switch (SWPT) Setting

PRTA - Print Subroutine

This subroutine is entered after the translation of each source statement is complete. It is used to print a line in the object program listing.

The data, instruction, and location counter are translated into their hexadecimal equivalent by subroutine TRAN and placed in the assembled machine instruction portion of the print line. A line of the object program listing is then printed by subroutine INOUT.

The number of columns that are printed and the contents of the print line are dependent upon the setting of switch SWPT. This switch is set by the routine that calls the PRTA subroutine. The settings of the switch, their meanings, and when used, are shown in Figure 12.

The PRTA subroutine also counts the number of lines printed per page; when the count exceeds 56, the printer is ejected to the next page.

After a line is printed, control is returned to the calling routine.

PCHA - Store into Punch Buffer Routine

This subroutine is entered after the translation of a machine instruction or a DC, CNOP, or CCW Assembler instruction. The translated text is placed in the punch buffer up to a maximum of 56 bytes.

The actual punching of a card is under the control of the Phase II Control Routine.

Control is returned to the calling routine.

Punch RLD and END Cards Routine

This subroutine is entered after the END statement has been processed.

The RLD Table is searched, and all RLD's with the same position header are grouped into one card image (or until a card is full) and produced. (See Phase II Introduction for RLD card layout and Section 2, "Tables," for a description of the RLD Table.)

After the RLD cards are produced, the END card format is punched, and the program halts.

If the output is tape, an 'LDT' record is written on the tape after the 'END' record. The tape is then backspaced over the LDT record, so that in a stacked assembly all LDT records but that following the last object program are overwritten.

DOUT - Punch TXT Cards Routine

This subroutine is entered from the Phase II Control Routine whenever enough data to fill an output buffer has been accumulated, and, if it is a card system, when a blank card is available.

The necessary control information is placed in the card image along with the translated text. (See Introduction to Phase II for TXT card layout, Figure 10.) The card image is then punched or written on tape, depending upon the system being used.

Control is returned to the Phase II Control Routine except when the output buffer is emptied after the END card is read. Control is then transferred to output RLD and END cards (see preceding discussion).

FERR - Store Error Flags Routine

The FERR subroutine places the necessary error flags in the print buffer. These flags are those placed in the Intermediate Text by Phase I and the flags for the errors encountered during Phase II processing.

The following error flags may be placed in the print buffer by Phase II:

Flag	Meaning
I	Expression cannot be mapped into a base register and displacement
K	RLD Table full
U	Undefined symbol
W	Unused mask bits in CCW instruction not zero
X	Duplicate ENTRY statement
Y	Illegal negative expression in operand field

Note: See "Phase I", Section 3, Subroutine Description, subroutine ERR for Phase I error flags and flags common to both phases.

INOUT - Input/Output Subroutine

This subroutine is the same subroutine used by Phase I. Refer to "Phase I," Section 3, "Subroutine Descriptions."

SECTION 4: PHASE II PROCESSING FLOW

This section uses two charts to describe the overall processing flow of Phase II of the Assembler Program. Chart 06 illustrates the initialization procedures, and Chart 07 illustrates the Control Routine. The numbers in parentheses within the text describing each chart indicate the block on the chart being explained.

INITIALIZATION - CHART 06

Phase II of the Assembler Program is read into main storage. The new PSW's are loaded into main storage, and the print buffer is set to blanks (01). The type of input is then checked (03). The printer is ejected to a new page and a heading is printed.

If the input control indicates that Phase I assembled on tape, the tape unit is selected, the read parameter is set to read 104 bytes (one input record on tape), and the input tape unit is rewound (04-05). A card image is then read from tape (06), and control is transferred to complete the initialization processing (09).

If the input control indicates that Phase I assembled on cards, the first card is read and checked (06-07). If the card is a Symbol Table card, the rest of the Symbol Table deck is read into the area of main storage reserved for the Symbol Table (10). The Symbol Table deck will not be included when Phase I assembled on tape (Phase II must immediately follow Phase I), or when Phase II immediately follows Phase I in a card assembly, because the Symbol Table will still be in storage.

The type of object output is checked, i.e., tape or card (09.1), and the appropriate parameters for the output are modified (13,15) and control is then transferred to location C811 in the Control Routine (Chart 07, block 07) (05-06).

A test is made to determine what type of printing device is on line. One of the following then occurs:

1. If the off-line printing option is used, the parameters for off-line printing are modified (16).
2. If both a printer (1443/1403) and a typewriter (1052) are available, no modifications are made.
3. If the printer is used as a typewriter, all warning messages will be listed on the printer. However, if the programmer uses the no-print option, these warning messages will not be printed but will only be displayed in the I.C. lights on the console.
4. If the typewriter is used as a printer, modifications are made so that an EJECT instruction will be processed as a SPACE 6 instruction.
5. If no device is available, all printing is inhibited (including error flags).

The RLD Table is set up according to the size of main storage, the number of output buffers is determined, and the header for each is cleared (08). The number of buffers is determined by dividing the unused main storage into 64-byte fields. This number is then placed in locations NOPB (number of output buffers) and NBL (number of buffers left) for later use in the processing routine.

The first output buffer is set up to contain a START format with a blank program name and a starting location of zero. This is done to guard against the possibility of a programmer not specifying a START, without which the object program is not relocatable. If the program contains a START statement it will overlay the pseudo-start.

CONTROL ROUTINE - CHART 07

Note: The discussion of the Control Routine starts at location C811 (07) since this is the initial entry point from the initialization routine.

General

The card image, when the 1442 card read-punch unit is used, is checked for a blank. If the card is blank, the output buffer area is checked to determine if enough information has been processed to fill an object program card or card image on tape (8). If not, the blank card is ignored and control is transferred to location C819 (10) for further testing. If an output buffer is full, an object program card is produced by subroutine DOUT (09) before control is transferred to location C819 (10).

If the card is not blank (07), or the object program card has been produced (09), the ID-Code of the Intermediate Text is checked (10-14).

If the ID-Code is B or K (10), indicating an Assembler instruction, control is transferred to location C827 to determine

which Assembler instruction is to be processed (byte 2 of the Intermediate Text is checked), and control is transferred to the routine necessary to translate that instruction (see section 5, "Intermediate Text Translation"). The return to the Control Routine varies with the type of instruction and is shown in Section 5.

If the card was blank (11), as it would have been if an object program card had just been produced, control is transferred to location C804 (01) to set up for processing the next statement.

If the ID-Code is A, J or M (12), indicating a machine instruction or error type M, control is transferred to location C8231 (20) to translate the machine instruction (see Machine Instruction Processing section).

If the ID-Code is L or C (14), indicating the instruction is not to be assembled (L) or it is a comments statement (C), control is transferred to location C825 (16) to list the card in the object program listing (see Common section).

If the card does not contain a correct ID-Code or is not blank, error flag N is placed in the print buffer (15) by subroutine FERR before control is transferred to the print routine (16) (see Common section).

Common

After the above procedures, or after the translation of the instruction is completed correctly, subroutine PRTA is entered to store the data in the print buffer and to list a line of the object program (16). The number of columns to be listed, and the contents of each line, are determined by the procedure followed before entering this subroutine (see Section 3, "Subroutine Descriptions," subroutine PRTA, for detailed explanation).

The location counter is incremented by SIZE (number of bytes) of the instruction just translated (17). Control is transferred to location C804 (01) to set up for processing the next intermediate text statement.

Location C804 (01) will contain a transfer to location FH11 (02) until all START, ENTRY, EXTRN, and ICTL Assembler instructions have been translated.

The statement just processed is checked (02-03), and if it is blank, a START, ENTRY, EXTRN, or ICTL, or contains ID-Code L or C, control transfers to location D805 (05) to prepare for the next intermediate text statement.

If the intermediate text statement is none of the above, control is transferred to location FH12 (04). Location C804 (01) is changed to a branch to location C805, and shared main storage is reset to be used as additional output buffers. NBL and NOPB are incremented accordingly.

Shared main storage is the area of main storage occupied by the initialization routine and the routines necessary to translate the START, ICTL, ENTRY, and EXTRN Assembler instructions.

After shared main storage has been reset, the program switches that affect the processing of each instruction are reset, and the current value of the location counter is stored in the first location of the Symbol Table (05).

The program switches reset after each instruction is processed are:

SIZE contains the size of the instruction, and controls the incrementing of the location counter,
IOCTL contains the trap and operate switches for the INOUT subroutine,
ZBYT is the output byte counter for the PRTA subroutine,
SWPT controls the printing of a line of the object program by subroutine PRTA,
RELA is the relocatability attribute indicator.

After the switches are reset, subroutine INOUT is entered to read the next card or tape record (06). Control is then transferred to location C811 (07) to start the processing of the statement (see General section).

Machine Instruction Processing

If this is a type M CCW, control is transferred to location C908 (chart BD).

If the location counter is not at an even byte (half-word boundary) (18), control is transferred to location C930 (23).

SIZE, which indicates the number of bytes by which to increment the location counter, is set to a 1. Control is transferred to subroutine SUZ which zeros the output buffer and causes a byte of zeros to be printed. Subroutine PCHA is then entered to store one byte of zeros in the text buffer (24), and the location counter

is incremented by SIZE (25). Control is then transferred to translate the machine instruction (19).

If the location counter is on a half-word boundary, or after it is aligned to a half-word boundary, a test is made for a CNOP instruction (19). If this is a CNOP instruction, control is transferred to Chart BA to process the instruction. If this is not a CNOP, the operation code (byte 2 of the Intermediate Text) is placed in the output text buffer (20). A test is then made for an ID-Code M machine instruction (21). If the ID-Code is M, control is transferred to location F730 in subroutine SER to reset the text buffer to zeroes. If the ID-Code is not M, the size of the instruction (determined by the operation code) is stored in SIZE (21).

Control is then transferred to translate the instruction according to format (22) and returns to location C824 (27) to store the translated instruction in the punch buffer (27). (See Section 5, "Intermediate Text Translation," for error returns from the specific instruction translation routines.) After the data has been stored in the punch buffer, control is transferred to location C804 (01) to set up for processing the next intermediate text statement (see General section).

SECTION 5: INTERMEDIATE TEXT TRANSLATION

This section contains the flowcharts showing the translation of the operand fields of the Intermediate Text developed by Phase I. It contains a flowchart for each instruction type and format and a brief description of the flowchart.

The reader will require a basic knowledge of the subroutines described in Section 3, "Subroutine Description," before using this section.

ASSEMBLER INSTRUCTIONS

CNOP Translation - Chart BA (Blocks 01-07)

This routine is entered at location C923 (block 01) from the Phase 2 Control Routine (Chart 07). This routine determines what action is necessary for the PRTA and PCHA subroutines, and processes accordingly.

Control is returned to location C805 in the Phase II Control Routine.

ICTL Translation - Chart BA (Block 08)

The only action required for the ICTL Assembler instruction in Phase II is to set the print routine control switch, SWPT.

This translation routine is part of the CNOP translation routine. The routine is entered at location C926 (block 08); it sets the SWPT switch, and control is returned to location C825 in the Phase II Control Routine (Chart 07).

EJECT Translation - Chart BA (Blocks 09-13)

This routine is entered at location D104 from the Phase II Control Routine (Chart 07).

If the source statement contains an error, the source statement is printed before the printer EJECT instruction is carried out. Otherwise, only the EJECT action occurs.

If tape print option is used, an eject control bit causes the next statement written on tape to have an eject action after printing. If, however, the no-print option is used, the EJECT statement is ignored.

Control is returned to location C825 in the Phase II Control Routine.

SPACE Translation - Chart BA (Blocks 14-20)

This routine is entered at location D107 from the Phase II Control Routine (Chart 07).

If the operand contains a count greater than 63, the statement is processed in the same way as an EJECT statement with an error (see preceding discussion).

If the source statement contains an error other than the one above, the source statement will be printed, and the number of blank lines indicated in the operand (minus one) will be printed. If the source statement does not contain an error, the number of blank lines indicated in the operand will be printed.

If the space count plus the count of the number of lines already printed on the page is equal to or greater than 56, the line counter is reset to zero and the statement is processed as an EJECT instruction. This check is made after each single space; thus it is possible to SPACE to the bottom of the page and EJECT to the top of the next page.

Control is returned to location C825 in the Phase II Control Routine.

START Translation - Chart BB (Blocks 01-03)

This routine is entered at location D002 (block 01) from the Phase II Control Routine (Chart 07). The location counter is set to the translated value of the operand, or to zero if the operand is invalid. An ESD card image is placed in the output buffer, and control is returned to location C825 in the Phase II Control Routine.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

ENTRY and EXTRN Translation - Chart BB (Blocks 04-08)

This routine is entered at location D001 (block 04) from the Phase II Control Routine. The operand is translated and placed into the output buffer area.

If the statement is an EXTRN, an ESD card image is placed into the punch buffer, and control is returned to location C825 in the Phase II Control Routine (Chart 07).

If the statement is an ENTRY, the symbol in the operand can be used in one ENTRY statement only. The ESD card image is placed in the punch buffer if the symbol has not been used previously. Control is returned to location C825 in the Phase II Control Routine.

If the symbol has been previously used, this ENTRY statement is ignored, and an error flag is placed in the output buffer before control is returned to location C825 in the Phase II Control Routine.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

DROP Translation - Chart BB (Blocks 09-13)

This routine is entered at location D140 (block 09) from the Phase II Control Routine (Chart 07). The operand is translated and, if valid, the indicated register is removed from the Register Table.

The source statement is listed, and control is returned to location C804 in the Phase II Control Routine.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control returns to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

USING Translation - Chart BB (Blocks 14-24)

This routine is entered at location D020 (block 14) from the Phase II Control Routine (Chart 07). The first expression is translated and placed in the output buffer area.

If the first expression is valid, the second expression is translated and, if valid, the indicated register is entered into the Register Table. Control is returned to location C825 in the Phase II Control Routine.

If either translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

END, EQU, and ORG Translation - Chart BC

This routine is entered at location D006 (block 01) from the Phase II Control Routine (Chart 07). The operand is translated and, if valid, placed in the output buffer area.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control returns to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

If the instruction is an EQU, control is returned to location C825 in the Phase II Control Routine.

The source statement is printed, and the output is set to a blank buffer, since an ORG or an END instruction is considered to be a break in the string of specified storage content, and therefore a new output card must be started.

If the statement is an ORG instruction, control is returned to location C804 in the Phase II Control Routine.

If the statement is an END instruction, all remaining output buffers are produced; the RLD cards are produced from the RLD Table (see "Introduction"); the END card is produced; the EOJ message is listed and/or displayed; the processing is ended, and the program stops. If object output is on tape, an LDT record is written, and then backspaced over so that it will be overwritten if this is a stacked assembly.

DS Translation - Chart BD (Blocks 09-14)

This routine is entered at location C915 (block 09) from the Phase II Control Routine (Chart 07).

The location counter is incremented by the number of bytes reserved by the DS instruction. The output is set to a blank buffer, since a DS instruction is considered to be a break in the string of specified storage content, and therefore a new output buffer must be started.

The source statement is printed, and control is returned to location C804 in the Phase II Control Routine.

CCW Translation - Chart BD (Blocks 02-08) and Chart BE (Blocks 15-22)

This routine is entered at location C908 (Chart CF, block 02) from the Phase II Control Routine (Chart 07).

The location counter is aligned to a double-word boundary, and the necessary zeros are placed in the output buffer. The location counter is then set to be incremented by eight bytes, the size of the CCW instruction, since the location counter is incremented regardless of errors.

If the ID-Code is M, indicating the instruction is to be assembled as zeros, no further processing occurs, and control is returned to location C824 in the Phase II Control Routine.

The expressions in the operand are translated and placed in the output buffer area. The translation of the second expression, if valid, will cause an entry into the RLD Table (see "Introduction").

If the translation of any expression is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine because of an error, the remaining expressions in the statement are not translated.

DC Translation - Chart BD (Blocks 01, 03-06) and Chart BE (Blocks 01-14)

This routine is entered at location C901 (Chart CF, block 01) from the Phase II Control Routine (Chart 07).

The location counter is set to be aligned to the word boundary determined by the type of DC statement. The number of zeros necessary to accomplish this are placed in the output buffer.

The location counter is set to be incremented according to the L (length) field of the statement, and one of the following procedures is followed.

DC Type A Instruction: The operand is translated and placed into the output buffer area.

If the translation is invalid, error flags are placed into the print buffer and the necessary action, depending upon the severity of the error, is taken.

Control returns to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error.

An entry is made into the RLD Table (see "Introduction") if the translated operand is a three- or four-byte value. Control is returned to location C824 in the Phase II Control Routine.

Other DC Instructions: The constant is placed in the output buffer area and the source statement is listed. The location counter is then incremented by the size of the L field. A line is printed and the location counter is incremented until the duplication factor is equal to zero.

Control is then returned to location C804 in the Phase II Control Routine.

MACHINE INSTRUCTIONS

RR Format Translation - Chart BF (Blocks 01-08)

This routine is a continuation of the machine-instruction translation portion of the Phase II Control Routine starting at location C8201 (Chart 07).

The operand is translated according to the type of RR format (see "Phase I," Section 5, Chart AH and description) and placed in the output buffer area. Control is returned to location C824 in the Phase II Control Routine.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the tran-

slation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine, the remaining terms of the operand are not translated.

RS Format Translation - Chart BF (Blocks 09-15)

This routine is a continuation of the machine-instruction translation portion of the Phase II Control Routine starting at location C8201 (Chart 07).

The expressions in the operand field are translated according to the type of RS format (see "Phase I," Section 5, Chart AI and description) and placed in the output buffer area. Control is returned to location C824 in the Phase II Control Routine.

If the translation is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine, the remaining expressions of the operand are not translated.

RX Format Translation - Chart BG (Blocks 07-13)

This routine is a continuation of the machine-instruction translation portion of the Phase II Control Routine starting at location C8201 (Chart 07).

The expressions in the operand field are translated and placed in the output buffer area. Control is returned to location C824 in the Phase II Control Routine.

If the translation of any expression is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine, the remaining expressions of the operand are not translated.

SI Format Translation - Chart BG (Blocks 01-06)

This routine is a continuation of the machine-instruction translation portion of the Phase II Control Routine starting at location C8201 (Chart 07). The translation of the SI format starts at location FA05 (block 01).

The operand field is translated according to the type of SI format (see "Phase I," Section 5, Chart AK and description) and placed in the output buffer area. Control is returned to location C824 in the Phase II Control Routine.

If the translation of any expression is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine, the remaining expressions in the operand field are not translated.

SS Format Translation - Chart BH

This routine is a continuation of the machine-instruction translation portion of the Phase II Control Routine starting at location C8201 (Chart 07). The translation of the SS format starts at location FB01 (block 01).

The operand field is translated according to the type of SS format (see "Phase I," Section 5, Chart AL and description) and placed in the output buffer area. Control is returned to location C824 in the Phase II Control Routine.

If the translation of any expression is incorrect, error flags are placed in the print buffer, and the necessary action, depending upon the severity of the error, is taken.

Control is returned to location C824 in the Phase II Control Routine, or the translation continues, depending upon the severity of the error. If control is returned to the Phase II Control Routine, the remaining expressions in the operand field are not translated.

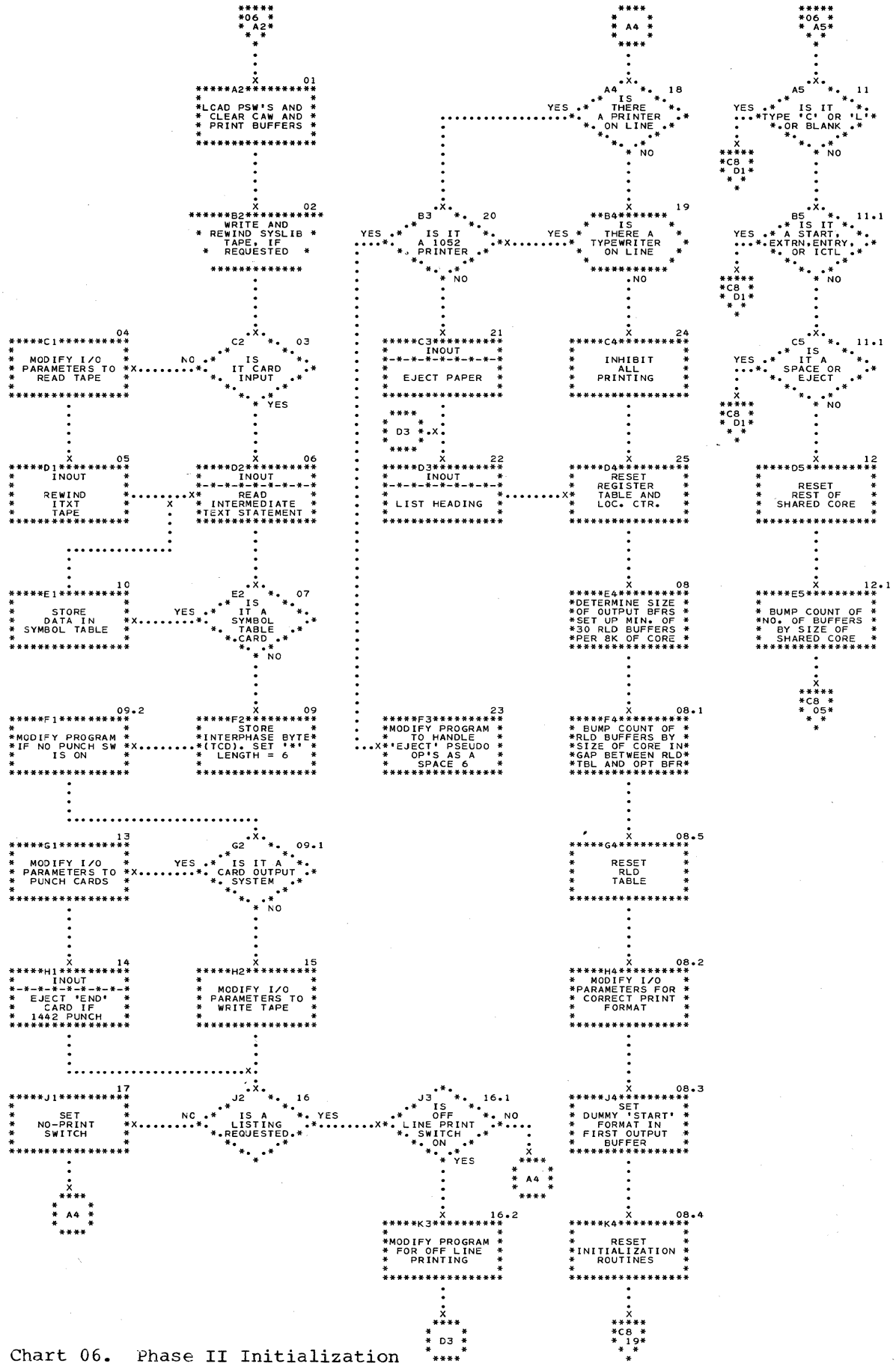


Chart 06. Phase II Initialization

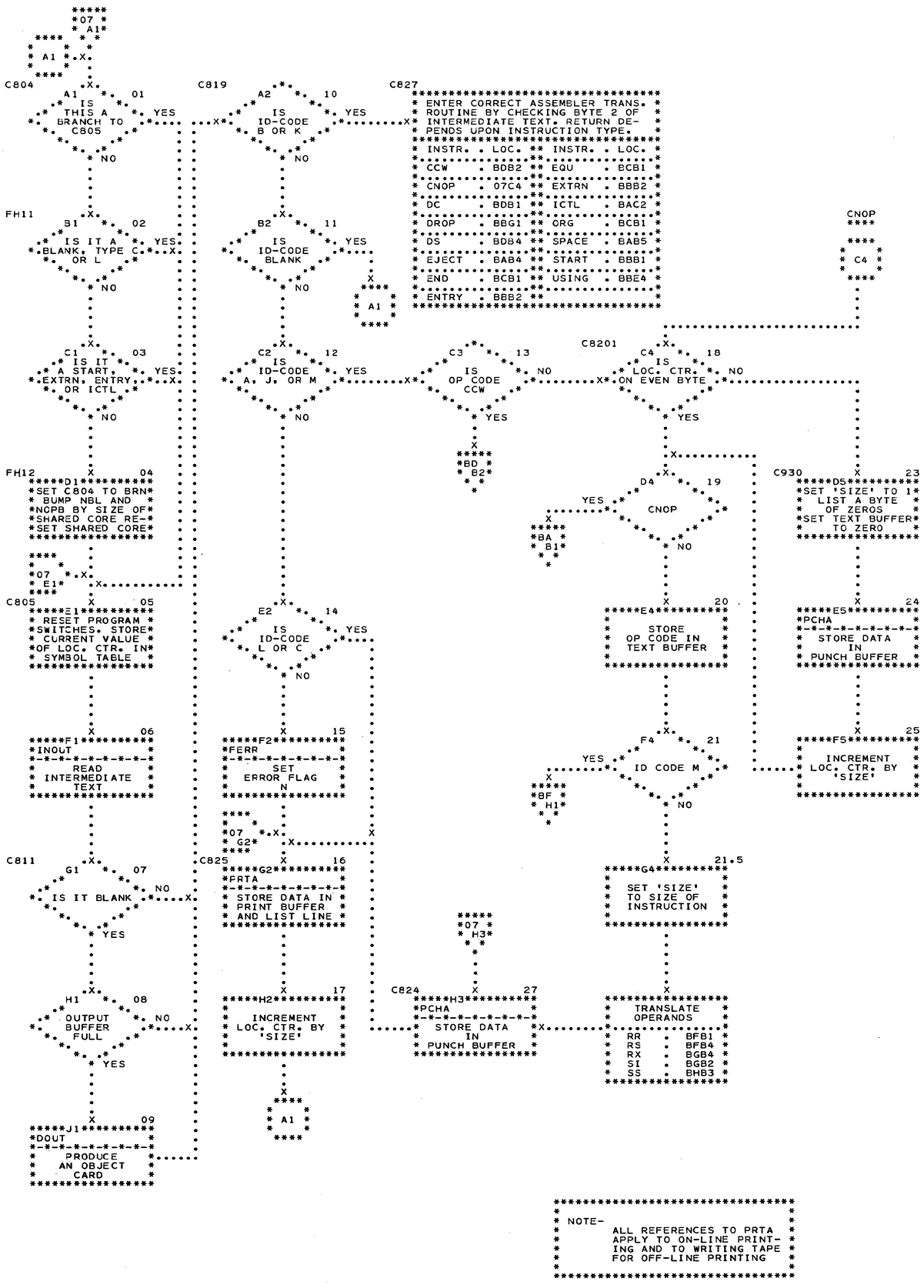


Chart 07. Phase II Control Routine

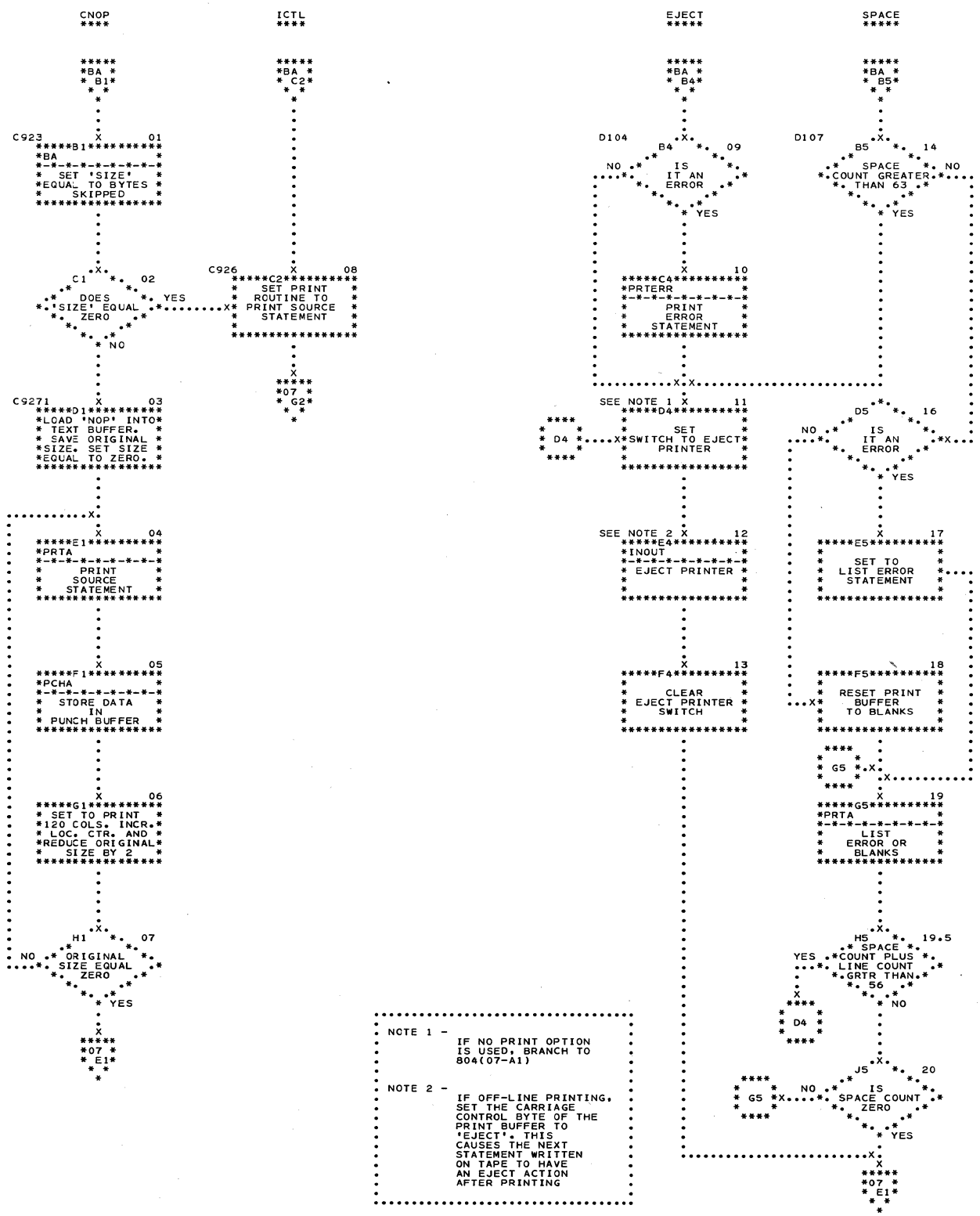


Chart BA. SPACE, EJECT, CNOP, and ICTL Translation

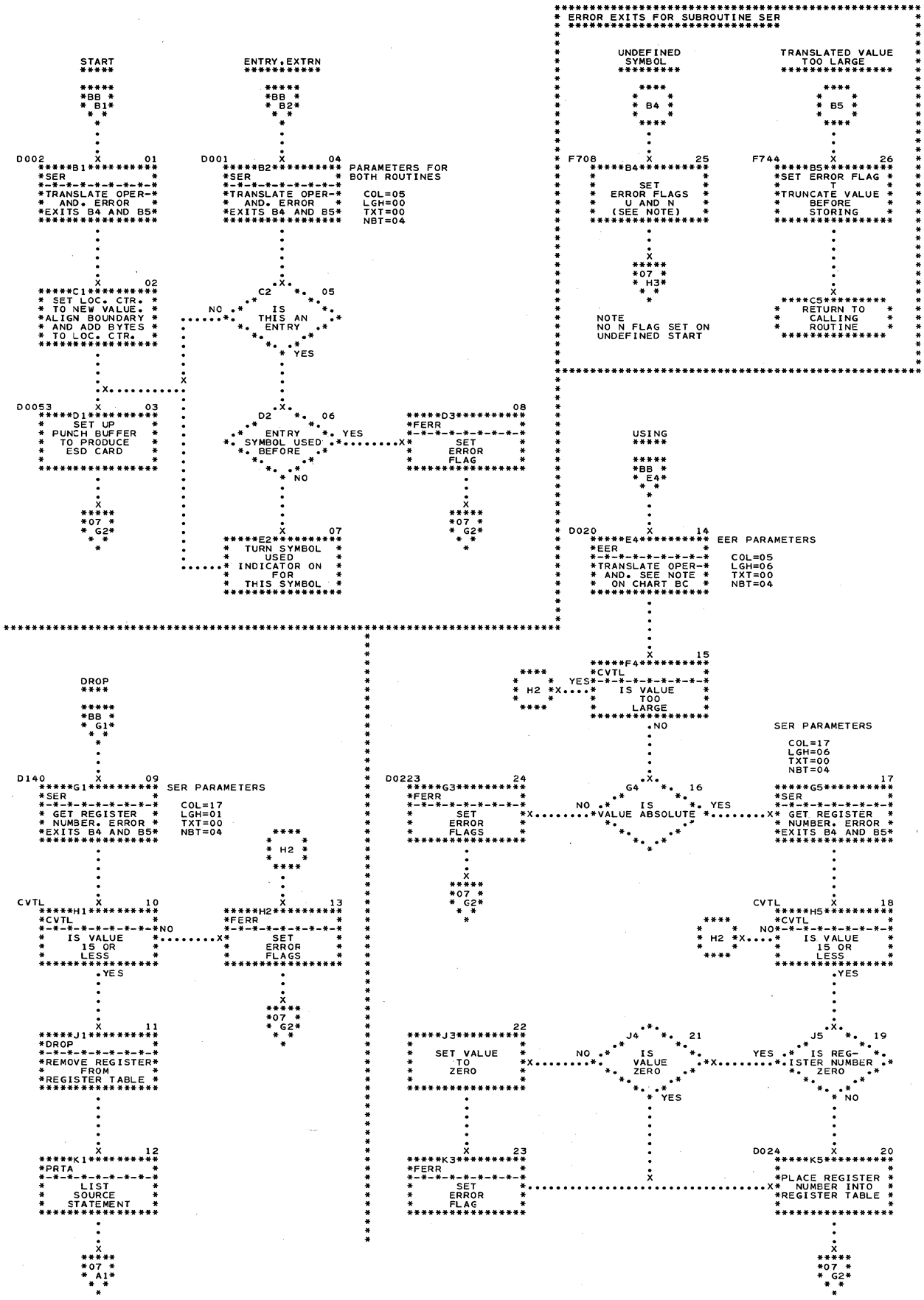


Chart BB. DROP, ENTRY, EXTRN, START, and USING Translation

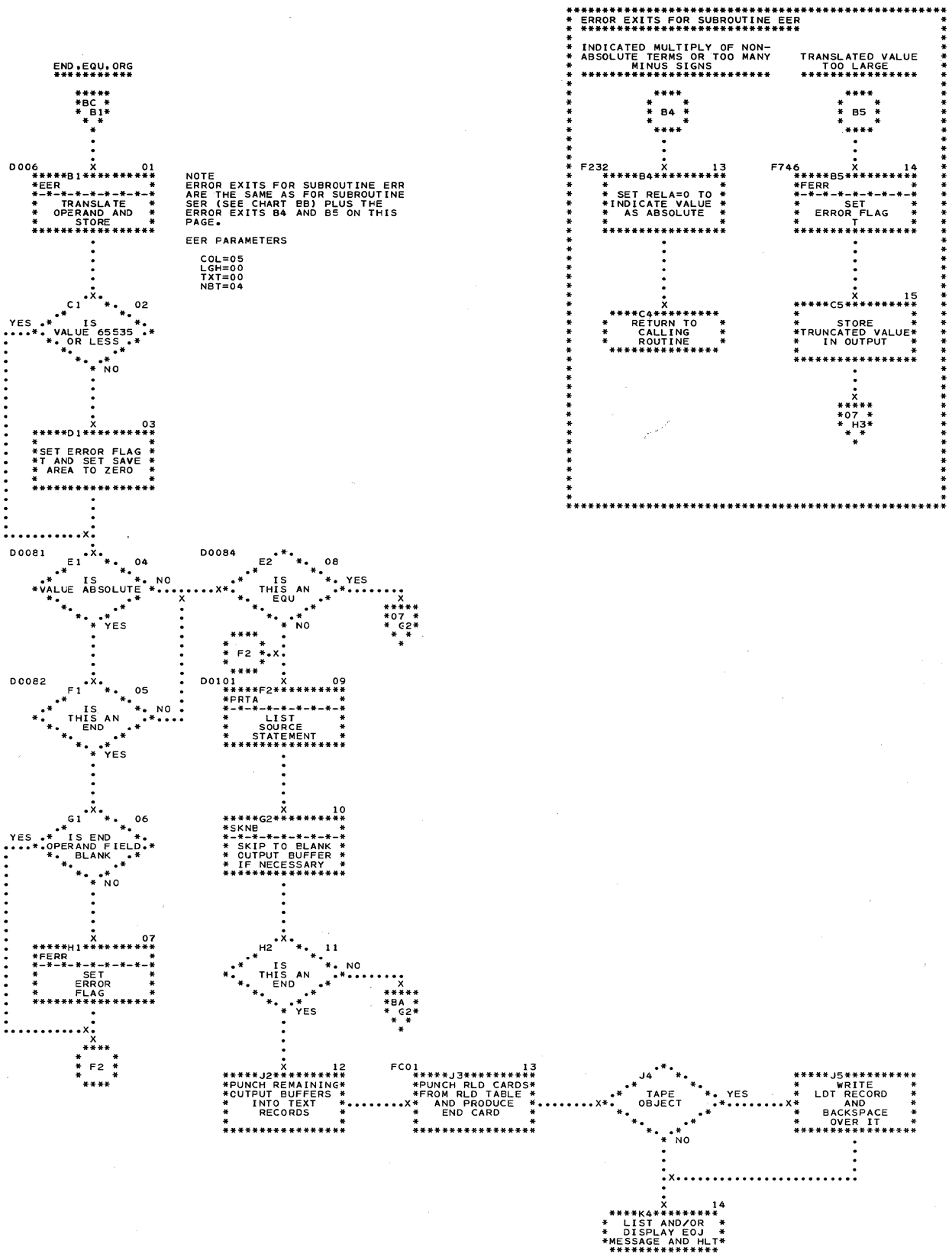


Chart BC. END, EQU, and ORG Translation

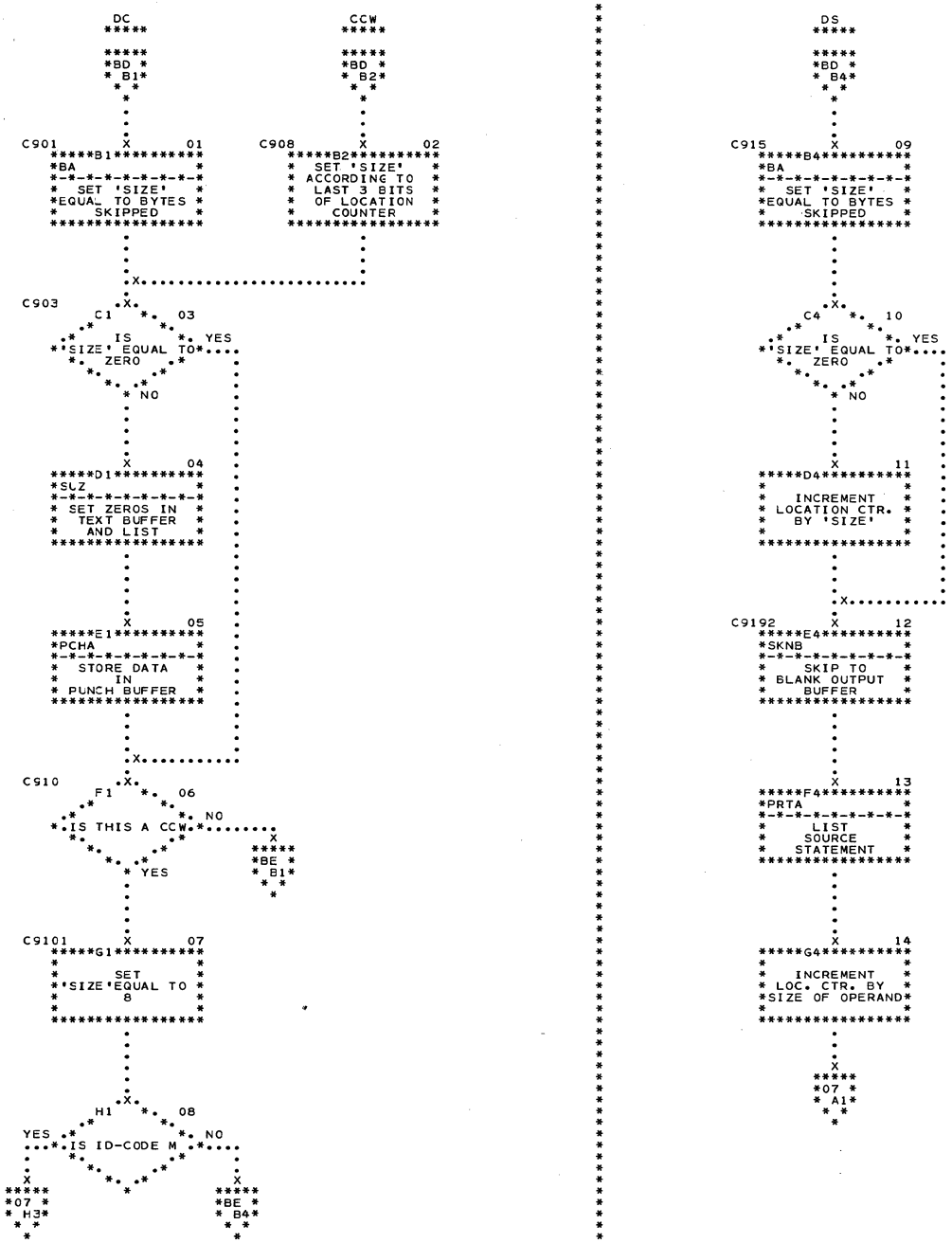


Chart BD. CCW, DC, and DS Translation

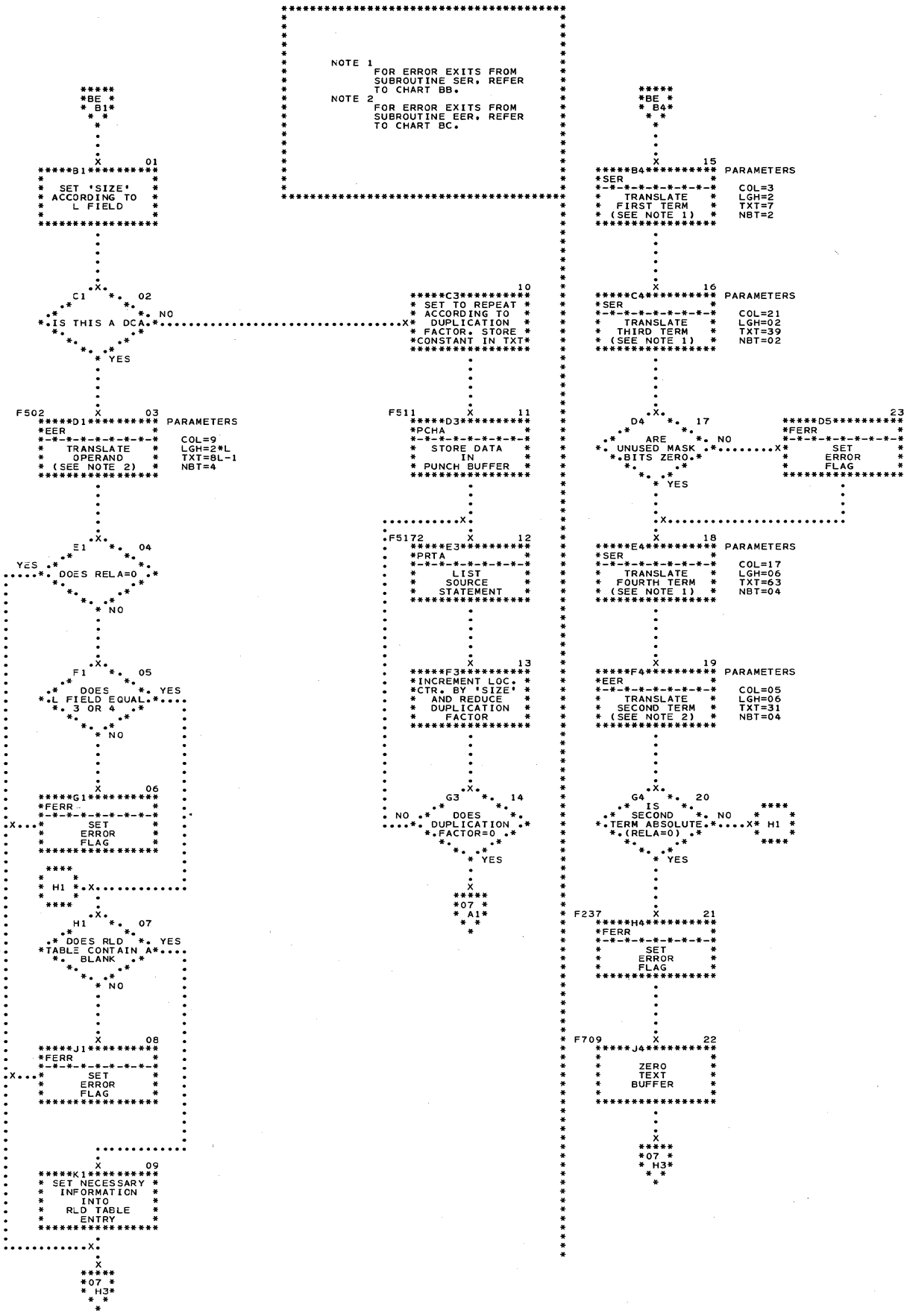


Chart BE. CCW and DC Translation (continued)

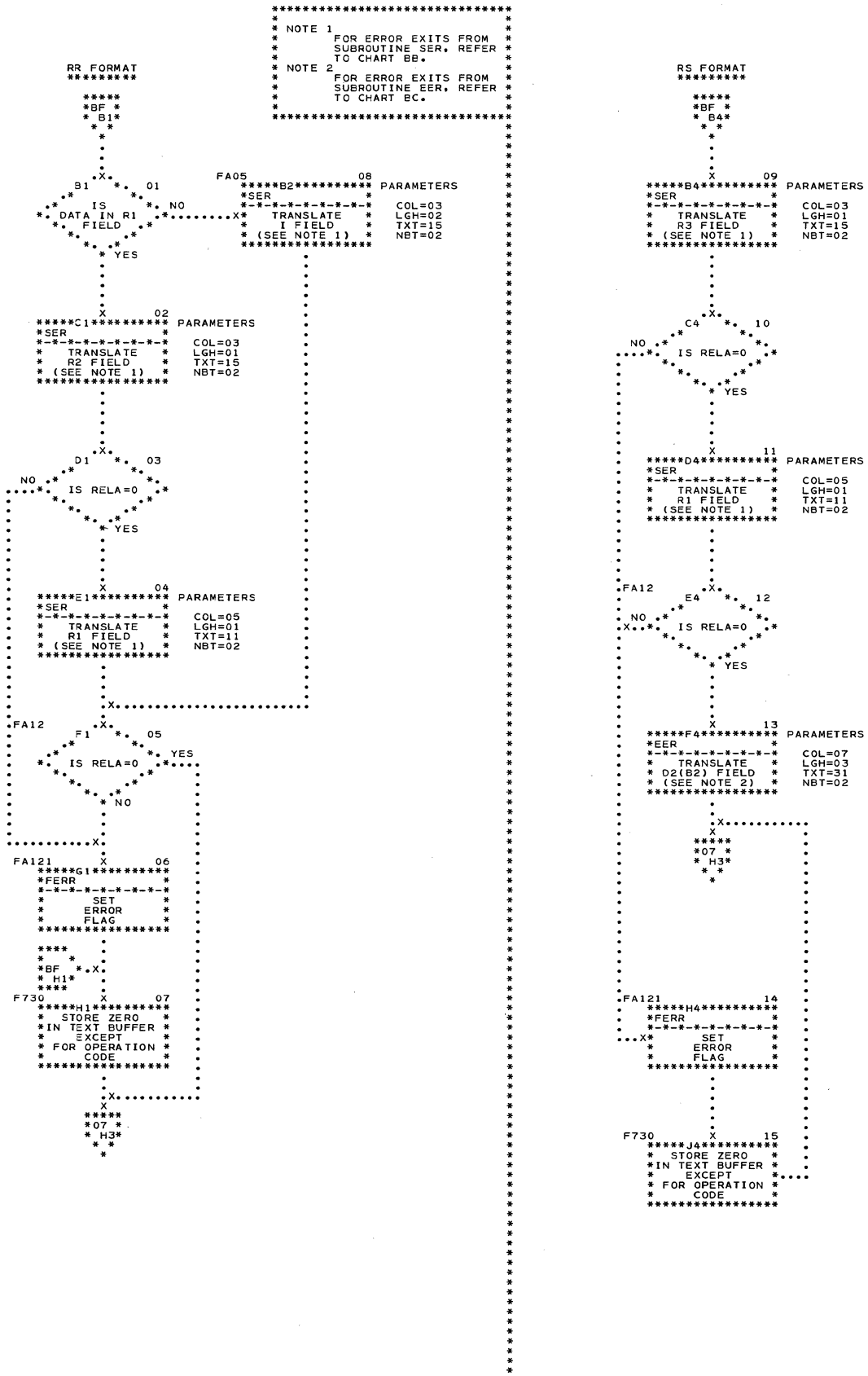


Chart BF. RR and RS Format Translation

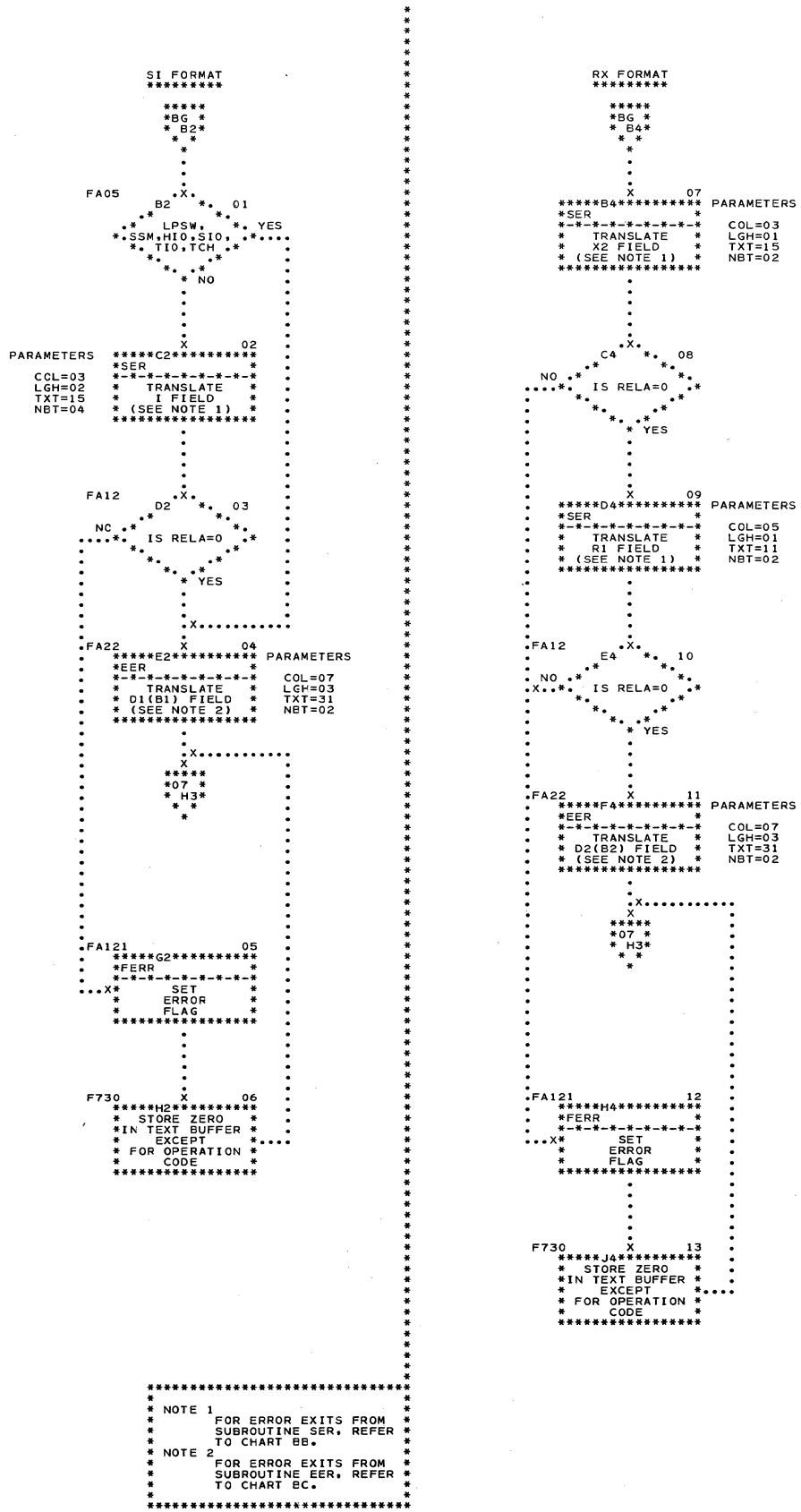


Chart BG. RX and SI Format Translation

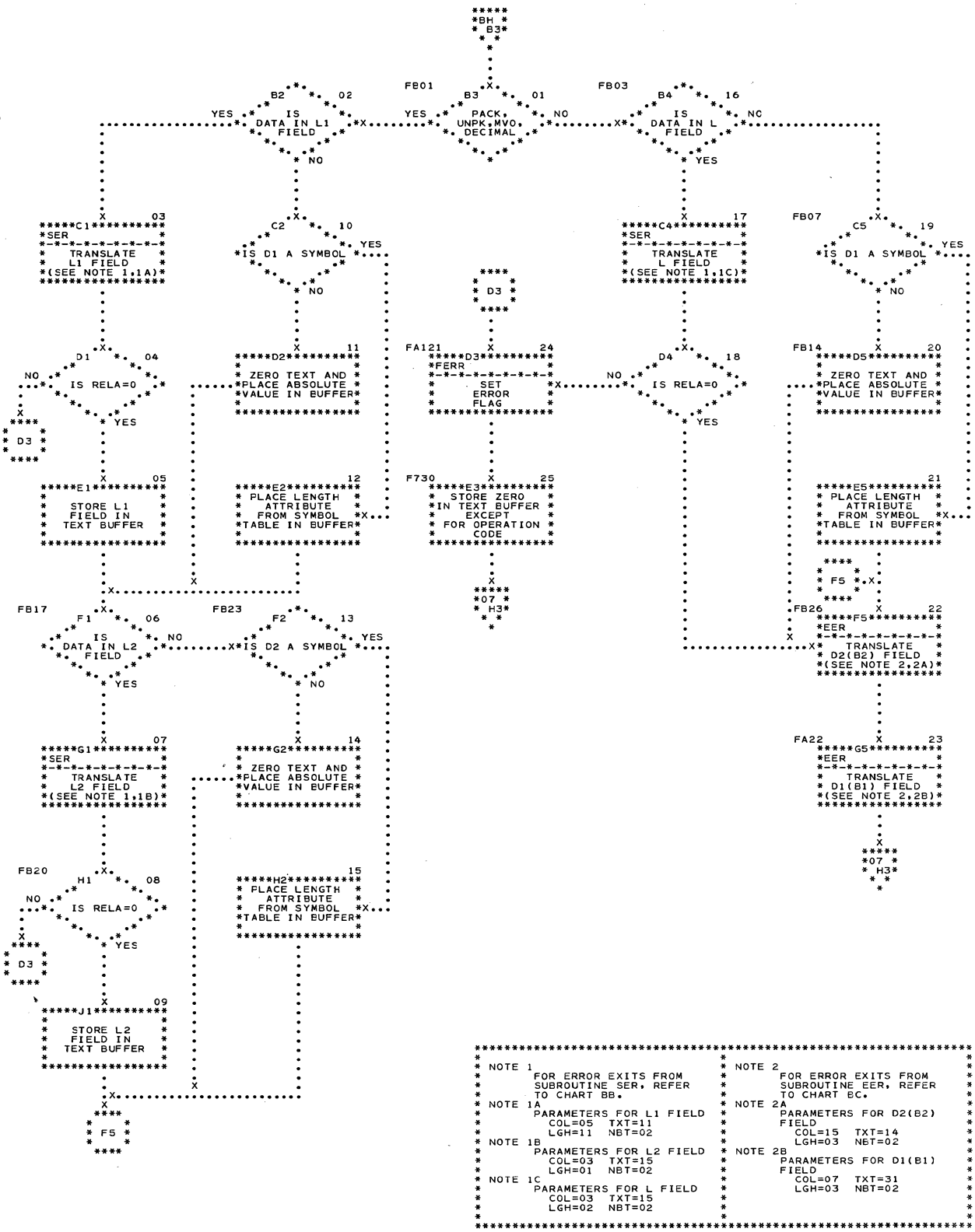


Chart BH. SS Format Translation

This appendix contains three cross-reference tables which may be used with the Autochart diagrams within this manual. Use of the tables will enable a user to locate any label appearing on any chart, all references to a particular entry connector, and all blocks where a subroutine is used.

TABLE I: LABELS

This table lists all labels that appear on charts. To the right of each label is the grid location and the block number associated with the block that bears that label.

BLOCK LABEL	BLOCK LOC	BLOCK NUMBER
CCW	AAB2	AA01
CNOP	ACB1	AC01
CVTL	BBH1 BBH5	BB10 BB18
C311	04B5	0420
C804	07A1	0701
C805	07E1	0705
C811	07G1	0707
C819	07A2	0710
C8201	07C4	0718
C824	07H3	0727
C825	07G2	0716
C901	BDB1	BD01
C903	BDC1	BD03
C908	BDB2	BD02
C910	BDF1	BD06
C9101	BDG1	BD07
C915	BDB4	BD09
C9192	BDE4	BD12
C923	BAB1	BA01
C926	BAC2	BA08
C9271	BAD1	BA03
C930	07D5	0723
DC	ABB2	AB01
DROP	ADG4	AD12
DS	ACA4	AC10
D001	BBB2	BB04
D002	BBB1	BB01
D0053	BBD1	BB03
D006	BCB1	BC01
D0081	BCE1	BC04
D0082	BCF1	BC05
D0084	BCE2	BC08
D0101	BCF2	BC09
D020	BBE4	BB14
D0223	BBG3	BB24
D024	BBK5	BB20
D104	BAB4	BA09
D107	BAB5	BA14
D140	BBG1	BB09
EF07	AHH5	AH16
EF08	AHH4	AH15
EG11	AJJ4 AKE2	AJ10 AK05
EG12	AKD2 ALD1	AK04 AL04
EG15	AJK3 AKG2	AJ12 AK07
EG16	AKD4	AK08

BLOCK LABEL	BLOCK LOC	BLOCK NUMBER
EJECT	ADB1	AD01
EM04	AMC2	AM04
EM08+4	AMD2	AM05
END	ADB4	AD08
ENTRY	ADF1	AD03
EN07	AMF3	AM15
EN11	AMJ3	AM18
EN12	AMH3	AM17
EN13	AMH4	AM12
EN25	ANB1	AN01
EN26	AMJ2	AM21
EN27	AMH2	AM20
EP01	ANH3	AN11
EP021	ANF3	AN09
EQU	AEC1	AE01
ER03	ANC4	AN17
ER04	ANE5	AN25
EVE1	AME4	AM09
EXTRN	AEC4	AE09
FA05	BFB2 BGB2	BF08 BG01
FA12	BFE4 BFF1 BGD2 BGE4	BF12 BF05 BG03 BG10
FA121	BFG1 BFH4 BGG2 BGH4 BHD3	BF06 BF14 BG05 BG12 BH24
FA22	BGE2 BGF4 BHG5	BG04 BG11 BH23
FB01	BHB3	BH01
FB03	BHB4	BH16
FB07	BHC5	BH19
FB14	BHD5	BH20
FB17	BHF1	BH06
FB20	BHH1	BH08
FB23	BHF2	BH13
FB26	BHF5	BH22
FC01	BCJ3	BC13
FH11	07B1	0702
FH12	07D1	0704
F232	BCB4	BC13
F237	BEH4	BE21
F502	BED1	BE03
F511	BED3	BE11
F5172	BEE3	BE12
F708	BBB4	BB25
F709	BEJ4	BE22
F730	BFH1 BFJ4 BGH2 BGJ4 BHE3	BF07 BF15 BG06 BG13 BH25
F744	BBB5	BB26
F746	BCB5	BC14
MACHA	AJB2	AJ01
MACHB	AHB4	AH09
MACHC	AIA2	AI01
MACHD	AIB2	AI02
MACHE	AKB3	AK01
MACHF	AKB2	AK02
MACHG	ALA1	AL01
MACHH	ALB1	AL02
MACHI	AHF4	AH13
MACHJ	AHB3	AH08
NOTE -	AOD4	AO29
ORG	AGC1	AG01
SEE NOTE 1	BAD4	BA11
SEE NOTE 2	BAE4	BA12
SPACE	AGC4	AG08
START	AFB4	AF08
USING	AHB1	AH01
1CTL	AFB1	AF01

TABLE II: ENTRY CONNECTOR REFERENCES

This table lists all off-page entry connectors that are referenced by any off-page exit connector on another chart. The entry connectors are arranged in alphameric order with the grid location and block number of all blocks that reference this connector to the right and on as many additional lines as necessary.

ENTRY CONN	EXIT CONN	BLOCK NUMBER
AMD2	ANE2	AN15
AME4	ANK3	AN13
AMJ2	ANE5	AN25
ANB1	AMK2	AM22
ANB3	AMK3	AM19
AOA5	APD2	AP04
AOB5	APE2	AP05
	APE3	AP15
	APJ3	AP19
AOC2	APF4	AP23
	APG2	AP24
APA2	AOC3	AO14
APE2	AOA5	AO27
BAB1	07D4	0719
BAG2	BCH2	BC11
BDB2	07C3	0713
BEB1	BDF1	BD06
BEB4	BDH1	BD08
BFH1	07F4	0721
C8D1	06A5	0611
	06B5	0611
	06C5	0611
04A5	AFH4	AF15
04B1	03K2	0315
	03K3	0314
	05J4	0511
04B5	AAG4	AA13
	ABG4	AB17
	ACJ1	AC08
	ACK4	AC19
	ADB1	AD01
	ADD4	AD10
	ADJ1	AD06
	ADJ4	AD14
	AEG4	AE13
	AEJ1	AE07
	AFH1	AF07
	AGF4	AG11
	AGH1	AG06
	AHG1	AH06
	AHJ5	AH17
	AIK2	AI17
	AJJ1	AJ15
	AKJ2	AK14
	ALK1	AL09
05B2	04B5	0420
05F2	04B5	0420
	04C4	0417
	04H1	0407
07A1	BBK1	BB12
	BDG4	BD14
	BEG3	BE14
07E1	BAH1	BA07
	BAJ5	BA20
07G2	BAC2	BA08
	BBD1	BB03
	BBD3	BB08
	BBG3	BB24
	BBH2	BB13
	BBK5	BB20
	BCE2	BC08
07H3	BBB4	BB25
	BCC5	BC15
	BDH1	BD08
	BEJ4	BE22
	BEK1	BE09
	BFF4	BF13
	BFH1	BF07
	BGE2	BG04
	BGF4	BG11
	BHE3	BH25
	BHG5	BH23

TABLE III SUBROUTINE USAGE

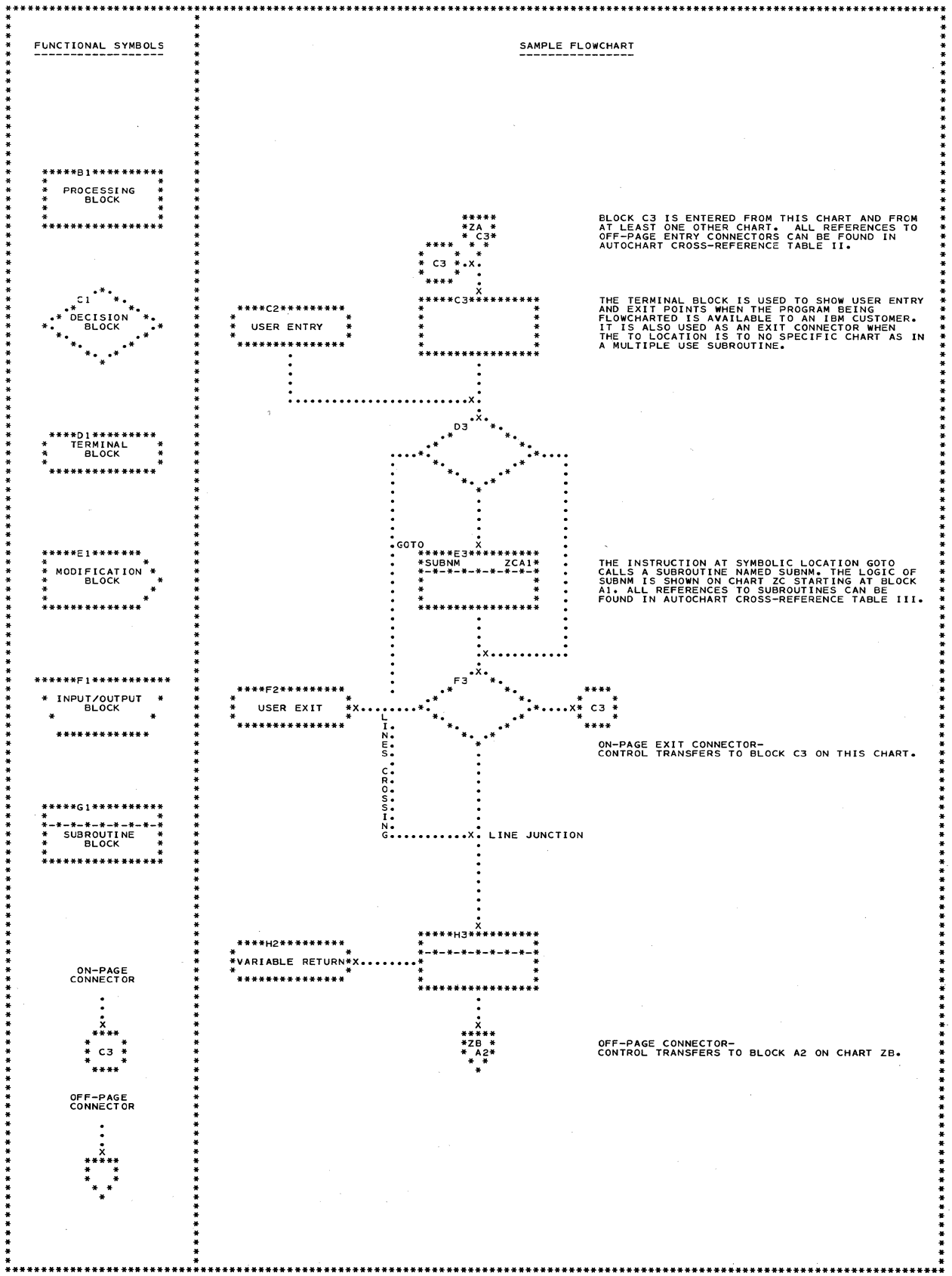
This table lists the subroutine names as they appear in subroutine blocks on the individual charts. The first appearance of a subroutine block has the name of that block as it appears in the first line of the block. To the right of the name is found the entry point into the subroutine, if it is present, and the grid location and block number of each block where this subroutine reference appears.

SUBROUTINE NAME	ENTRY CONN	BLOCK LOC	BLOCK NUMBER
BA		BAB1	BA01
		BDB1	BD01
		BDB4	BD09
BAR		AAB2	AA01
		AFG4	AF14
BUMP		05J4	0511
CVTL		BBF4	BB15
		BBH1	BB10
		BBH5	BB18
DCC		ANK5	AN29
DCF		ABC2	AB02
		ABJ2	AB10
		ACB4	AC11
		ACG4	AC16
DCF2		ANE4	AN19
DCF4		ANF4	AN20
DCX1		ANH5	AN27
DOUT		07J1	0709
DROP		BBJ1	BB11
EER		BBE4	BB14
		BCB1	BC01
		BED1	BE03
		BEF4	BE19
		BFF4	BF13
		BGE2	BG04
		BGF4	BG11
		BHF5	BH22
		BHG5	BH23
ERR		AAF3	AA15
		ABF3	AB18
		ACJ2	AC09
		ACJ5	AC20
		ADD1	AD02
		ADD5	AD11
		ADJ2	AD07
		ADJ5	AD15
		AEG5	AE14
		AEJ2	AE08
		AFE5	AF16
		AFH1	AF07
		AGF5	AG12
		AGH2	AG07
		AHG2	AH07
		AHJ5	AH17
		AIK2	AI17
		AJJ1	AJ15
		AKJ2	AK14
		ALJ2	AL29
		ANE1	AN04
		04C4	0417
EVE	AMB3	AAB4	AA08

SUBROUTINE NAME	ENTRY CONN	BLOCK LOC	BLOCK NUMBER
EVE	AMB3	AAC2	AA02
		AAE4	AA11
		AAF2	AA05
		ABC4	AB13
		ACB1	AC01
		ACF1	AC05
		ADB1	AD01
		ADB4	AD08
		ADF1	AD03
		ADG4	AD12
		AEC4	AE09
		AED1	AE02
		AFC1	AF02
		AFC4	AF09
		AGC1	AG01
		AGD4	AG09
		AGE4	AG10
		AHB1	AH01
		AHB4	AH09
		AHE1	AH04
		AHF4	AH13
		AIB2	AI02
		AIC4	AI09
		AIF2	AI06
		AIF4	AI12
		AJB2	AJ01
		AJE2	AJ04
		AJG3	AJ07
		AJJ4	AJ10
		AKB2	AK02
		AKE2	AK05
		AKF4	AK10
		ALB1	AL02
		ALC5	AL22
		ALD2	AL12
		ALD3	AL16
		ALF1	AL06
		ALF3	AL18
		ALG5	AL25
		04B3	0411
		04C3	0412
		04F4	0419
		04G2	0408
FERR		BBD3	BB08
		BBG3	BB24
		BBH2	BB13
		BBK3	BB23
		BCB5	BC14
		BCH1	BC07
		BED5	BE23
		BEG1	BE06
		BEH4	BE21
		BEJ1	BE08
		BFG1	BF06
		BFH4	BF14
		BGG2	BG05
		BGH4	BG12
		BHD3	BH24
		07F2	0715
INOUT		BAE4	BA12
		03B3	0301
		03J3	0313
		03K2	0315
		04B1	0401
		05G4	0514
		05G5	0517
		05J2	
		06C3	0621
		06D2	0606
		06D3	0622
		06H1	0614
		07F1	0706

SUBROUTINE NAME	ENTRY CONN	BLOCK LOC	BLOCK NUMBER
PCHA		BAF1	BA05
		BDE1	BD05
		BED3	BE11
		07E5	0724
		07H3	0727
PRTA		BAE1	BA04
		BAG5	BA19
		BBK1	BB12
		BCF2	BC09
		BDF4	BD13
		BEE3	BE12
		07G2	0716
PRTERR		BAC4	BA10
SER		BBB1	BB01
		BBB2	BB04
		BBG1	BB09
		BBG5	BB17
		BEB4	BE15
		BEC4	BE16
		BEE4	BE18
		BFB2	BF08
		BFB4	BF09
		BFC1	BF02
		BFD4	BF11
		BFE1	BF04
		BGB4	BG07
		BGC2	BG02
		BGD4	BG09
		BHC1	BH03
		BHC4	BH17
		BHG1	BH07
SKNB		BCG2	BC10
		BDE4	BD12
STOCAW		APB2	AP02
		APB3	AP12
		APF4	AP23
		APG3	AP17

SUBROUTINE NAME	ENTRY CONN	BLOCK LOC	BLOCK NUMBER
STORE		AAC4	AA09
		AAD2	AA03
		AAF4	AA12
		AAG2	AA06
		ACC1	AC02
		ACG1	AC06
		ADC4	AD09
		ADG1	AD04
		ADH4	AD13
		AED4	AE10
		AEE1	AE03
		AFD1	AF03
		AFD4	AF10
		AGD1	AG02
STORE		AGF4	AG11
		AHC1	AH02
		AHC4	AH10
		AHF1	AH05
		AHG4	AH14
		AIC2	AI03
		AID4	AI10
		AIG2	AI07
		AIG4	AI13
		AJC2	AJ02
		AJF2	AJ05
		AJH3	AJ08
		AJK4	AJ11
		AKC2	AK03
		AKF2	AK06
		AKG4	AK11
		ALC1	AL03
		ALD5	AL23
		ALE2	AL13
		ALE3	AL17
		ALG1	AL07
		ALG3	AL19
		ALH5	AL26
		ANF3	AN09
TLUL		AFH4	AF15
		ANB1	AN01
TLUN		05C3	0507


FUNCTIONAL SYMBOLS
PROCESSING BLOCK
C1
DECISION BLOCK
D1
TERMINAL BLOCK
E1
MODIFICATION BLOCK
F1
INPUT/OUTPUT BLOCK
G1
SUBROUTINE BLOCK
ON-PAGE CONNECTOR
C3
OFF-PAGE CONNECTOR
SAMPLE FLOWCHART
ZA C3
C3
C2
USER ENTRY
C3
BLOCK C3 IS ENTERED FROM THIS CHART AND FROM AT LEAST ONE OTHER CHART. ALL REFERENCES TO OFF-PAGE ENTRY CONNECTORS CAN BE FOUND IN AUTOCHART CROSS-REFERENCE TABLE II.
THE TERMINAL BLOCK IS USED TO SHOW USER ENTRY AND EXIT POINTS WHEN THE PROGRAM BEING FLOWCHARTED IS AVAILABLE TO AN IBM CUSTOMER. IT IS ALSO USED AS AN EXIT CONNECTOR WHEN THE TO LOCATION IS TO NO SPECIFIC CHART AS IN A MULTIPLE USE SUBROUTINE.
D3
GOTO
E3
SUBNM ZCA1
THE INSTRUCTION AT SYMBOLIC LOCATION GOTO CALLS A SUBROUTINE NAMED SUBNM. THE LOGIC OF SUBNM IS SHOWN ON CHART ZC STARTING AT BLOCK A1. ALL REFERENCES TO SUBROUTINES CAN BE FOUND IN AUTOCHART CROSS-REFERENCE TABLE III.
F2
USER EXIT
F3
C3
ON-PAGE EXIT CONNECTOR- CONTROL TRANSFERS TO BLOCK C3 ON THIS CHART.
LINES CROSSING
LINE JUNCTION
H2
VARIABLE RETURN
H3
ZB A2
OFF-PAGE CONNECTOR- CONTROL TRANSFERS TO BLOCK A2 ON CHART ZB.

address: The unique identification of a register, storage location, or other data source or destination.

Assembler instruction statement: An assembler language statement that directs the operation of the assembler or affects the format, content, location assignments, or register usage of its output, and usually does not cause any machine language instructions to be produced.

compound expression: A combination of two or, at most, three simple expressions, connected to each other by arithmetic operators.

ESD cards: These cards or card images on tape are part of the object program deck, and are produced for each START, ENTRY, and EXTRN Assembler instruction to provide complete linkage between different object programs.

Intermediate Text: The partially translated source statement produced by Phase I and used as input to Phase II.

location counter: The internal counter used to assign consecutive storage addresses to program statements.

machine instruction: An Assembler language statement specifying an operation that corresponds to a machine operation or operations, and that causes the Assembler to produce a corresponding machine-language instruction or instructions.

Operation Code Table: The table that is part of Phase I and contains each symbolic operation code, its machine equivalent, and the location of the routine necessary to translate the operand field.

Register Table: The table constructed by Phase II that contains general register numbers, their corresponding values and relocatable attributes and is used to assign base registers and displacements when the assembler must compute addresses.

relocation: Modification of address constants to compensate for the difference between the origin assumed in an object program and its assumed origin relative to other object programs with which it is combined.

RLD cards: These cards or card images on tape are part of the object program, and are produced for each relocatable address constant and CCW defined in the program to allow the Relocatable Loader to reevaluate symbols and expressions within address constants and CCW's.

RLD Table: The table constructed by Phase II that contains the data necessary to produce the RLD cards.

simple expression: A single symbol or a single self-defining value used as an operand.

symbol: Any collection of characters by which something is known and can be referred to; a name.

Symbol Table: The table constructed by Phase I that contains all the symbols, and their attributes, contained in the source program.

Symbol Table Deck: The deck (produced in a card system only) that contains the attributes of all the symbols in the object program and is used as input to Phase II.

TXT cards: These cards or card images on tape, are part of the object program deck, and contain the machine language statements constructed from the symbolic statements in the source program.

READER'S COMMENTS

itle: IBM System/360 Basic Programming Support
Basic Assembler
Program Logic Manual

Form: C28-6555-1

s the material:

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

ow did you use this publication?

___ As an introduction to the subject ___ For additional knowledge
Other _____

fold

lease check the items that describe your position:

___ Customer personnel ___ Operator ___ Sales Representative
___ IBM personnel ___ Programmer ___ Systems Engineer
___ Manager ___ Customer Engineer ___ Trainee
___ Systems Analyst ___ Instructor Other_____

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

___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

xplanation:

fold

Name _____

Address _____

FOLD ON TWO LINES,STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

staple staple

fold fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

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

POSTAGE WILL BE PAID BY

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

ATTN: PROGRAM LOGIC DOCUMENTATION
DEPARTMENT D89

fold fold

staple staple

Printed in U.S.A. C28-6555-1

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