SPECTRA 70

RADIO CORPORATION OF AMERICA · ELECTRONIC DATA PROCESSING

SYSTEM 70/15

ASSEMBLY SYSTEM REFERENCE MANUAL

RADIO CORPORATION OF AMERICA

70 - 15 - 602

October, 1965

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: April, 1965
Revised: October, 1965

FOREWORD

♦ This manual provides the user with all the information necessary to write symbolic programs in the 70/15 Assembly Language. The instructions for assembling these programs, in preparation for subsequent execution on the RCA 70/15 Processor, are included. In addition, a section on the Input-Output Control System (IOCS) is provided. This I/O system reduces and simplifies the coding required for control of input/output operations. The I/O section has been incorporated into this manual so that it may serve as a single source for all symbolic programming references.

It is assumed that the reader is familiar with the RCA 70/15 Processor. A complete description can be found in the RCA 70/15 Processor Reference Manual (No. 70-15-601).

CONTENTS

CONTENTS (Cont'd)

CONTENTS (Cont'd)

INTRODUCTION

♦ The 70/15 Assembly System translates a symbolic machine-oriented source language program into a computer-recognizable object program. The Assembler assists the programmer by reducing the amount of complex details required to code in machine language and, as a result, increases the efficiency of the programmer. Source language input and object program output may be punched on cards or stored on magnetic tape. The Assembler produces an output listing of both the source and object codes.

FEATURES

♦ Several of the Assembler features are summarized below.

Operation Code Mnemonics

♦ Each machine instruction is assigned a unique mnemonic operation code as specified in the assembly language. The programmer uses these mnemonics instead of the less readable binary-bit configurations to specify the desired instructions.

Symbolic Addressing

♦ Every memory location is available for assignment as a symbolic name for program reference. This symbolic name allows reference to branch points, tables, constants, and storage areas without requiring knowledge of absolute memory addresses which will subsequently be assigned by the Assembler.

Expressions

♦ The Assembler provides the ability to combine symbols and numeric values to form desired addresses. For example, relative addressing is a feature of the assembler where items in the program are addressed relative to a defined symbol; i.e., an incremental or a decremental value relative to a symbolic name.

External References

♦ The Assembler enables a program or a subprogram to reference data and/or control information outside its boundaries. This feature is provided via a unique program symbolic name that represents the desired reference. Thus, separately assembled subprograms may be linked together at execution time.

In addition to these features, the Assembler also has provisions for permitting subsequent object program relocation and appropriate error checking of source programs with associated warning flags.

SYSTEM REQUIREMENTS

EQUIPMENT REQUIREMENTS

♦ *Minimum Equipment*

Processor	1	70/15A*, 70/15B**
Card Reader	1	70/237 or 70/251 with any punched card read feature
Card Punch	1	70/234, 70/236
Printer	1	70/242, 70/243

Optional Equipment

Processor*	1	70/15B
Magnetic Tapes**		70/432, 70/442, 70/445

RELATED PROGRAMMING SYSTEMS

♦ The following loaders are provided for loading the 70/15 Assembly System into memory:

Card System - 70/15 Loader AA (Absolute)

Tape Library System - 70/15 Loader AT (Absolute/PLT)

The 70/15 Tape Assembly System is designed around the utilization of device interchangeability. Even though the system is card oriented, magnetic tapes may be substituted for any of the following:

Assembler Program	Object Program Listing
Source Input	Object Program Deck
Second Pass Input	

The I/O commands within the 70/15 Assembler reference the different peripheral devices as logical devices. The logical devices are linked with the actual devices at load time by the loaders. I/O Define cards (see page 53 for format description) must be included with the Loader and the Assembler at load time in order to accomplish this linkage. The I/O Define cards specify, among other things, where a device is to be found (trunk and unit number) and what type of device is to be found there (card reader, punch, etc.). For each of the following logical device numbers, an I/O Define card must be supplied at load time:

00 - Input Device for Pass 1	08 - Object Program Output Device
01 - Input Device for Pass 2	09 - Assembly Listing Device

Through this feature, it is readily seen that magnetic tape(s) may be designated for any or all of the above functions.

* Card Assembly System
** Tape Assembly System

ASSEMBLY STATEMENT FORMATS

♦ The 70/15 Assembly System translates symbolic machine-oriented source language programs into machine language programs. The source language program must conform to certain conventions, one of which is the assembly statement format. The assembly statements are written on RCA SPECTRA 70 coding forms (Figure 1) and occupy columns 1-71. Column 72 is unavailable to the programmer and columns 73-80 are available for both the program identification and statement sequencing.

A statement consists of from one to four fields. These are: the Name field, the Operation field, the Operand field, and the Comments field. (The Identification-Sequence field is explained on page 12.) The statement fields are written on the RCA SPECTRA 70 coding form left-justified within the following columns:

Columns	*Field*
1 - 4	Name field
5 - 9	Blank
10 - 14	Operation field
15	Blank
16 - 71	Operand and Comments fields
72	Blank
73 - 80	ID-Sequence field

GENERAL CONSIDERATIONS

♦ 1. Every statement must have a machine or Assembler instruction specified in the Operation field except when the entire statement line is used for comments. The other fields may or may not be used, depending on the particular instruction and the programmer's wishes.

2. Embedded blanks must not be used within the Name or Operation fields. Blanks can only occur as a character constant or a self-defining value within the Operand field.

3. Only one statement can be written on a coding line. A statement can use a maximum of 71 columns and cannot be continued onto another line of coding.

4. Column 72 cannot be used by the programmer and must be left blank.

5. When the word *Blank* is used in this manual in reference to the contents of a column or field, it indicates that no punch is to be made in that column or field.

CHARGE NO. _____

DATE REQ'D _____

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE _____

DATE _____ PAGE _____ OF _____

PROGRAM _____

PROGRAMMER _____

NAME		OPERATION		OPERAND		COMMENTS		IDENTIFICATION
1 2 3 4 5 6 7 8	9	10 11 12 13 14	15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71	72	73 74 75 76 77 78 79 80		

28-00-119

Figure 1. RCA Spectra 70 Assembly Program Form

RULES FOR CHARACTERS AND SYMBOLS

♦ The following rules govern the use of characters and symbols when writing Assembler Statements:

Character Set

♦ The character set for 70/15 Assembly Statements consists of the following graphics:

Alphabetic	A through Z
Numeric	0 through 9
Special Characters	$ * + - , ( ) ' BLANK

Symbols

♦ A symbol is a set of one to four characters used to name a statement line. The set of characters that may be used is limited to any alphabetic character (A through Z) or numeric character (0 through 9). In addition, the first character of the symbol must be alphabetic. The $ precedes special symbols that reference standard memory locations and may be used only in the Operand field of an instruction. (See page 11.)

Examples

♦ *Valid:* MASK

A12

B

Invalid: INPUT (too many characters)

123 (first character not alphabetic)

A.2 (decimal point is not a valid character for symbol)

Defining Symbols

♦ Symbols provide the means for referencing instructions, constants, or storage areas and are initially defined in the Name field of a statement. Normally, any symbol which is referenced is defined within the same program. The 70/15 Assembly System, however, permits independently assembled programs to reference symbols in other independently assembled programs. This is accomplished via the EXTRN and ENTRY Assembler Program Linking instructions.

External Symbols

♦ As was stated above, the 70/15 Assembly System allows a programmer to refer to a symbol which is defined in another assembled program. This type of symbol is called an external symbol. Any symbol that is referred to by a statement in another assembled program is called an entry point. An entry point can, of course, also be referenced by statements within its own program. The programmer writes an ENTRY statement for every entry point, and an EXTRN statement for every external symbol. The use of this feature is meaningful only when the programs are to be loaded and executed together. The ENTRY and EXTRN statements are described on pages 24 and 25 respectively.

Additional Restrictions on Symbols

♦ 1. A symbol can be defined only once per assembly. Duplicate names are flagged and the first definition is used in all cases.

2. A symbol cannot be used as a name and an external in the same assembled program.

3. The size of the processor's main storage (4K or 8K) determines the maximum number of symbols that can be used in one program. In the Card Assembly System, the maximum number of symbols for a 4K processor is 90; maximum for an 8K processor is 700. The maximum number of symbols for the Tape Assembly System is 550.

NAME Field (Columns 1-4)

♦ In the Name field a symbol is used to identify a particular statement so that it can be referred to by another statement. A name does not have to be given to every statement. It should be used when the statement is to be referred to by another statement.

Whenever a name is given, it must start in column 1 and extend no further than column 4. If column 1 is a blank, the assembler assumes that the statement has not been named. If column 1 contains an asterisk (*), the entire line is considered a comment (See page 12 for the description of the Comments field).

OPERATION Field (Columns 10-14)

♦ A programming or Assembler mnemonic instruction is written in the Operation field. This field starts in column 10. Valid mnemonics vary from one to five characters in length. The valid programming and Assembler instructions and their mnemonics are described under "Programming Instructions" (page 13) and "Assembler Instructions" (page 16). A blank in column 10 will indicate an invalid instruction.

The Assembler generates a Halt and Branch instruction and four zero bytes whenever an invalid mnemonic is specified. This allows the programmer to patch any size instruction in the area without resorting to a separate patch area. The statement is also flagged as an error.

OPERAND Field (Columns 16-71)

♦ The Operand field defines the locations, data, or device which are used by the Operation field. The Operand field must begin in column 16. In machine instructions, it is used to specify the second through sixth bytes of the instruction. The format of the Operand field for an Assembler instruction varies according to the requirements of the machine instruction.

The Operand field is divided into subfields. Each subfield is itself called an operand. The number of these operands depends on the instruction.

Commas are used to separate operands within a statement. It should be remembered that blanks cannot be written within a particular operand or between operands. When a blank is detected, it immediately terminates the Operand field and the remaining information is treated as a comment. If the Operand field is incomplete, the statement is flagged as an error on the assembly listing.

OPERAND Field (Cont'd)

The most common Operand field format uses two operands. Each of these operands represents the locations in memory that are to be used in the execution of this instruction. For additional information concerning the examples given in the following sections, refer to the programming instruction formats (page 13).

Defining Operands

♦ The Operand field defines locations, data and lengths of data, or devices which are used by the Operation field. The following sections describe the methods by which operands may be defined in the assembly language. However, a description of the Location Counter and its function is useful at this point.

The Location Counter

♦ The Assembler assigns locations to those statements which use or reserve memory areas. The Assembler keeps a counter, called the Location Counter, to facilitate the handling of the location assignment. The Assembler normally assigns consecutive locations to statements. (The START and ORG Assembler instructions (See description under Programming Instructions, page 16) allow the programmer to set and reset the location counter respectively.) The Location Counter is incremented by the number of bytes required by a statement after the analysis of the statement. The Location Counter is then pointed at the leftmost available memory byte. Statements that are given symbolic names have their names associated with the Location Counter setting of the leftmost byte of the field required by the statement. This is then stored in a table and, when the statement is subsequently referenced by its name in the Operand field of another statement, the associated memory location is placed in the Operand field of the generated-machine instruction.

Location Counter Overflow. The Assembler normally compiles programs for a 4K or 8K memory. The Location Counter can maintain addresses up to 65K. However, the Assembler flags any statement which sets the Location Counter above the limit of the available memory of the processor. The statement, nevertheless, will be processed with the excess location Counter value.

Program References. The programmer can specify the current value of the Location Counter as an address by writing an asterisk (*) at the appropriate point in the Operand field. This is an alternative to naming the statement and then using this name in the Operand field. The programmer can specify relative addressing after the asterisk. The programmer should note that the location referred to is the leftmost byte of the statement.

Chart 1. Example of Program Referencing Using the Location Counter

NAME	OPERATION	OPERAND	COMMENTS
	B	*+6	JUMP TO LOCATION AFTER DC
	DC	A(*)	GENERATE AN ADDRESS CONSTANT OF DC LOCATION

Relative Addressing

◆ *Using Location Counter.* The programmer may assign an address value to an operand that will be relative to the current value of the location counter. This is done by writing an asterisk (*) at the appropriate point in an Operand field. The programmer can then specify an address which is equal to, greater, or less than the current Location Counter value. The address assigned will be based on the leftmost byte location of the current statement. The plus sign (+) is used for addition and the minus sign (-) is used for subtraction. (See Chart 2.)

Using Symbols. Any symbol (statement name) can have an amount added or subtracted from it. As in relative addressing using the current Location Counter value, the plus sign (+) is used for addition and the minus sign (-) for subtraction. The amount of relative addressing is expressed by a self-defined value. The value of the relative address always represents the number of bytes. (See Chart 3.)

Chart 2. Example of Relative Addressing Using Location Counter

NAME	OPERATION	OPERAND	COMMENTS
	BC	X'F',*+4	BRANCH TO NEXT INSTRUCTION

Chart 3. Example of Relative Addressing Using Symbols

NAME	OPERATION	OPERAND	COMMENTS
	MVC	ABLE(X'E'),ABLE+X'E'	MOVE A 14-BYTE FIELD FROM THE AREA
*			ADJACENT TO ABLE TO ABLE
	MVC	ABLE(14),ABLE+14	SAME AS ABOVE BUT WITH DECIMAL SELF
*			-DEFINING VALUES

Self-Defining Values

♦ Self-defining values define: relative addresses, operand length, I/O information, and masks to the Assembler. They also provide an alternative method of defining storage addresses. They are converted by the Assembler to their binary equivalents.

The Assembler accepts three types of self-defining values: decimal, hexadecimal, and character. The programmer may choose any type he desires.

Decimal

♦ An unsigned decimal number can be used to further define an operand. The value is one to four decimal-digits. The Assembler converts this value to its binary equivalent. The length of this binary value depends on the format of the generated-machine instructions. The number must be represented by the set of digits zero (0) through nine (9). A value of more than four digits will be truncated and flagged in the Assembler listing. The excess high-order or leftmost characters will be dropped. (See Chart 4.)

Parentheses used in Chart 4 are required by the instruction format. See description under "Programming Instructions" (page 13).

Hexadecimal

♦ A hexadecimal value is defined by writing the letter X followed by the hexadecimal digits enclosed in single quotation marks. The value can consist of from one to four hexadecimal digits. A self-defined hexadecimal value of more than four digits is truncated at four digits and flagged in the Assembler listing. The low-order or rightmost characters are dropped. The decimal digits zero (0) through nine (9) and the alphabetic characters A through F are the valid hexadecimal digits. Any invalid character results in an error flag. The equivalent four-bit binary value for each of the valid hexadecimal digits is shown on page 10.

Chart 4. Examples of Decimal Self-Defining Values

NAME	OPERATION	OPERAND	COMMENTS
	RDF	1(2),FRST,LAST	READ A RECORD FROM DEVICE 2, TRUNK 1 INTO
*			THE AREA DEFINED BY FRST AND LAST
	MVC	TO(225),FROM	MOVE 225 BYTES
	AP	SUM(12),ADD(3)	ADD THREE BYTES OF PACKED DECIMAL TO A 12
*			-BYTE FIELD
	MVC	ABLE(1),90	MOVE ONE BYTE FROM LOCATION (5A)16 TO
*			LOCATION ABLE

Hexadecimal (Cont'd)

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

Chart 5 shows examples of hexadecimal self-defining values.

Note that the maximum of four digits refers to self-defined values and not to a constant defined by a DC (Define Constant) See page 20.

Character

♦ A character value is defined by writing the character C followed by a single character enclosed in single quotation marks. (See Chart 6.) The character can be any of the 256 combinations available in an eight-bit format. The Assembler assumes that the character is punched in the Extended Binary-Coded Decimal Interchange Code (EBCDIC). The self-defined character value ' (single quotation mark) is written as two consecutive quotation marks within the beginning and ending single quotation marks (e.g., C'''').

Chart 5. Examples of Hexadecimal Self-Defining Values

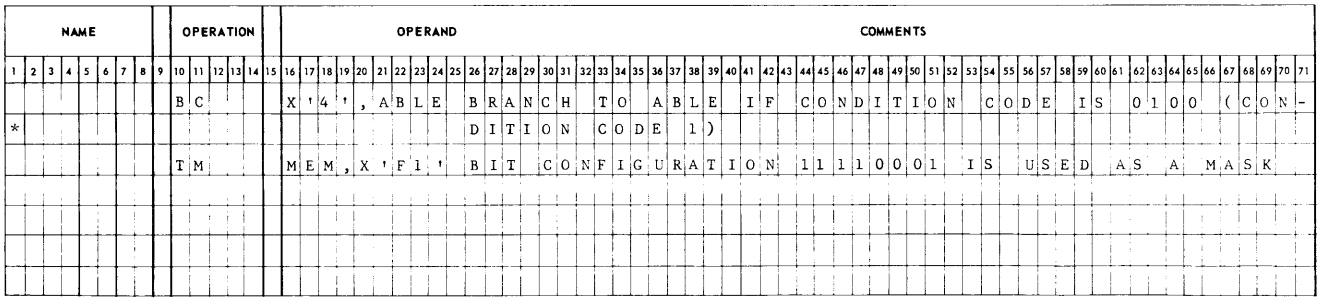
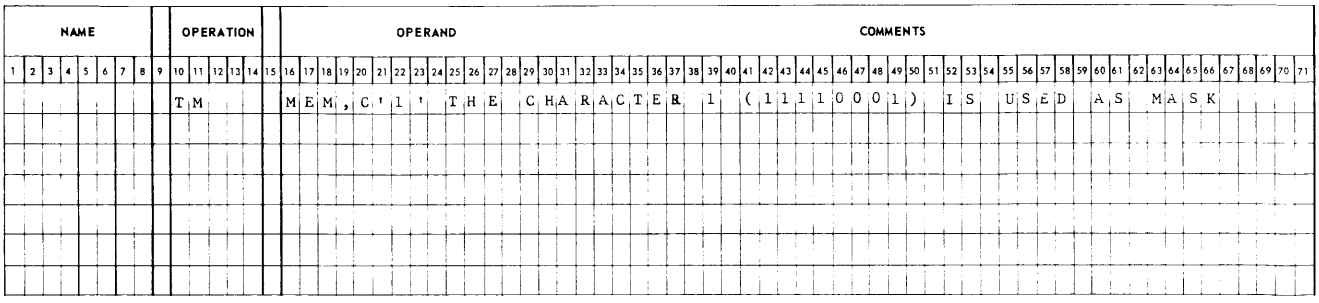


Chart 6. Example of Character Self-Defining Value



Relocatable Operands

♦ The 70/15 Assembly System, along with its normal function of generating machine code from source code, provides all the necessary control information which would permit a program to be relocated. Provided that a relocatable loader is used, all operands may be relocated with the exception of the following:

1. Self-defining values used to define absolute machine addresses.
2. Hexadecimal or character constants (see page 20, Defining Constants).
3. $P, $L, $LS

Standard Memory Locations

♦ The following special symbols may be used to reference certain standard memory locations. They must be used in the operand field of an instruction and must be immediately preceded by a dollar sign ($).

Special Symbol	*Location*	*Significance*
$P	130	MSB* of subroutine parameter
$L	200	MSB of Branch to Loader
$LS	196	MSB of Branch to instruction following the loader read logic.

*MSB - Most Significant Byte

1. $P is the MSB of the standard memory locations which are utilized for communication between a program and a subroutine. The length of this area is 16 bytes and may be used by any program which desires to pass on parameter information to a subroutine. See "Input/Output Control System," for an example of its use.
2. $L is the MSB of an instruction which branches to the Loader. Production runs may be stacked and executed one at a time. This may be accomplished by requiring that each program execute a Halt and Branch to $L as its last instruction. The halt gives the operator time to perform such functions as dismounting and mounting tapes, etc. To execute the next program, all that the operator need do is depress the START button.
3. $LS is the MSB of an instruction which branches to the instruction following the loader read logic. This feature permits a program to call in another program without having the Loader read from the input parameter device. The initiating program accomplishes this by moving a card image of a Loader call card into the Loader's input read area and then branching to $LS. Therefore, a program may, based on some condition, decide to call in one program instead of another. The programs to be called, are, of course, assumed to be on a program library tape.

COMMENTS Field

♦ The programmer can write descriptive comments within the program statement. They are allowed strictly for documentation purposes and they do not have any effect on the object program.

A Comments field can be specified in one of two ways. The primary procedure is to write a comment to the right of the operand field. At least one column to the right of the last operand must be blank. In addition, the entire statement line can be used for comments if an asterisk (*) is written in the first column (Column 1).

Comments must end in column 71. A second card with an asterisk in column 1 may be used to continue the comments.

IDENTIFICATION - Sequence Field (Column 73-80)

♦ The Identification - Sequence field is available for both program identification and statement sequencing. Columns 73-80 are reserved for these functions. The Assembler does not perform a sequence check on these columns. The first four characters (columns 73-76) are used by the Assembler. When written into the Start card, they are used as the name assigned to the Identification field produced for the object program. The last four characters (columns 77-80) are used as the start of the sequence counter. If they are not numeric, the Assembler ignores them and sets its sequence counter to all zeros. Every assembled instruction card will have its sequence number derived from the sequence counter set by the START card.

PROGRAMMING INSTRUCTIONS

♦ The 70/15 Assembly Language provides the means for symbolically representing all machine instructions. The symbolic format of these instructions varies according to their machine formats.

Machine instructions are aligned by the Assembler on even-byte boundaries. Any byte that is skipped will be set to zero.

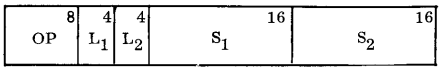
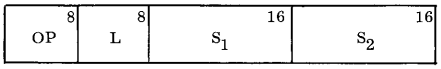
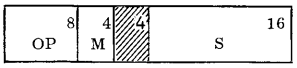
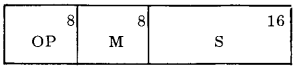
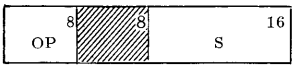
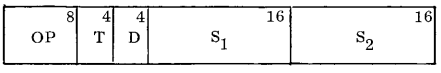
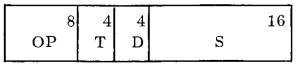
A name or symbol can be given to any programming instruction and any other statement may reference this symbol. The address value given to the symbol is the address of the leftmost byte of the assembled machine code. The length or number of bytes of each instruction can be determined from the machine format.

The length subfields (L, L_1, L_2), expressed within the symbolic instructions, are always one more than the count subfield of the machine instruction, as defined in the 70/15 Processor Reference Manual. The symbolic representation will be the conventional decimal count. For example, a three-character Move will have a symbolic count of three.

ASSEMBLY/ MACHINE CODE FORMATS

♦ Table 1 illustrates the basic machine formats, the instructions associated with these formats, and the format of the applicable Operand field.

Table 1. Assembly/Machine Code Formats

Assembler Operand Field Format	Basic Machine Format	Applicable Instructions
$S_1(L_1)$, $S_2(L_2)$		AB, AP, CP, PACK, SB, SP, UNPK
$S_1(L)$, S_2		CLC, ED, XC, MVC, NC, OC
M, S		BC
S, M		TM, HB
S		STP2
T(D), S_1, S_2		RDA, RDF, RDR, WR, WRA, WRC, WRE, IOS
T(D), S		PS

LIST OF INSTRUCTIONS

♦ Table 2 is a listing of the instructions in order by instruction name. The column headings of the table and their definition are described below.

Instruction Name:	This column gives the name of the instruction.
Mnemonic Op:	This column gives the mnemonic operation code for this instruction.
Machine Code:	This column gives the hexadecimal representation of the actual machine operation code.
Length in Bytes:	This column gives the actual length in bytes of this instruction.
Operand Field Format:	This column gives the symbolic format of the operand field for this instruction.
Example:	This column gives an example of the operand field format in Assembler language.

Table 2. 70/15 Instructions

Instruction Name	Mnemonic Op	Machine Code	Length In Bytes	Operand Field Format	Example
Add Binary	AB	F6	6	$S_1(L_1)$, $S_2(L_2)$	SUM(2), ADD(2)
Add Decimal	AP	FA	6	$S_1(L_1)$, $S_2(L_2)$	SUM(2), ADD(1)
Logical And	NC	D4	6	$S_1(L)$, S_2	BCIN+1(1), NOPC
Branch on Condition	BC	47	4	M, S	X'F', GOTO
Compare Decimal	CP	F9	6	$S_1(L_1)$, $S_2(L_2)$	ABLE(3), BAKE(3)
Compare Logical	CLC	D5	6	$S_1(L)$, S_2	ABLE(1), BAKE
Edit	ED	DE	6	$S_1(L)$, S_2	MASK(2), DATA
Exclusive Or	XC	D7	6	$S_1(L)$, S_2	NUM(2), ONES
Halt and Branch	HB	81	4	S, M	$L, X'01'
I/O Sense	IOS	E1	6	T(D), S_1, S_2	1(1), FRST, LAST
Move	MVC	D2	6	$S_1(L)$, S_2	TO(3), FROM
Logical Or	OC	D6	6	$S_1(L)$, S_2	INTO(2), MASK
Pack	PACK	F2	6	$S_1(L_1)$, $S_2(L_2)$	FOUR(2), EGHT(4)
Post Status	PS	66	4	T(D), S	1(1), *
Read Auxiliary	RDA	C5	6	T(D), S_1, S_2	1(1), FRST, LAST
Read Forward	RDF	E5	6	T(D), S_1, S_2	1(1), FRST, LAST
Read Reverse	RDR	E2	6	T(D), S_1, S_2	1(1), FRST, LAST
Set P2	STP2	82	4	S	RTRN
Subtract Binary	SB	F7	6	$S_1(L_1)$, $S_2(L_2)$	DIFF(2), SUB(1)
Subtract Decimal	SP	FB	6	$S_1(L_1)$, $S_2(L_2)$	DIFF(1), SUB(1)
Test Under Mask	TM	91	4	S, M	MEM, X'F1'
Unpack	UNPK	F3	6	$S_1(L_1)$, $S_2(L_2)$	EGHT(3), FOUR(1)
Write	WR	E3	6	T(D), S_1, S_2	2(2), FRST, LAST
Write Auxiliary	WRA	C3	6	T(D), S_1, S_2	2(2), FRST, LAST
Write Control	WRC	E7	6	T(D), S_1, S_2	2(2), FRST, LAST
Write Erase	WRE	E4	6	T(D), S_1, S_2	2(2), FRST, LAST

LEGEND

L = Length (1-256) in bytes
L_1 = Length of First Field (1-16) in bytes
L_2 = Length of Second Field (1-16) in bytes
S = Storage Address
S_1 = Storage Address of First Field
S_2 = Storage Address of Second Field
M = Mask (Self-defining value)
T = Trunk (Self-defining value)
D = Device (Self-defining value)

ASSEMBLER INSTRUCTIONS

♦ The Assembler instructions are used to supplement the machine instructions. They instruct the Assembler in the same sense that the machine instructions instruct the machine. The Assembler instructions are listed below and then described in detail.

Assembler Control Instructions

START - Start Program

ORG - Set Location Counter

END - End Program

Definition Instructions

DS - Define Storage

DC - Define Constant

Program Linking Instructions

ENTRY - Identify Entry Point

EXTRN - Identify External Symbol

Extended Mnemonic Instruction

B - Branch Unconditional

The Branch Unconditional is the only Assembler instruction to generate an actual machine instruction. The Start Program, Set Location Counter, Define Storage, and Define Constant are the only other Assembler instructions to affect the Location Counter.

ASSEMBLER CONTROL INSTRUCTIONS

START - Start Program

♦ The START statement must be written as the first statement in a program and it must always be present. The Name field is used to name the program and the Operand field is used to set the Location Counter to its initial value.

The format is as follows:

NAME	OPERATION	OPERAND
Symbol	START	A self-defining value

START - Start Program (Cont'd)

♦ The symbol written in the Name field is considered to be an entry point. The name on the START card is entered in the Entry-Extrn table, thus allowing another program to refer to this program via an EXTRN statement. A name consisting of four blanks is generated if a symbol is not defined in the Name field.

A character, hexadecimal, or decimal self-defining value can be written in the Operand field. This value is used to set the Location Counter to its initial value. (See Chart 7.) Thus the START Assembler instruction serves as an ENTRY and ORG verb as well as signalling the first line of an assembly. The value assigned must be an even-numbered byte. If it is not, the Assembler will assign an even-numbered address one byte higher than the address designated by the START instruction.

If a START directive is not written as the first statement or the Operand field is blank, the Location Counter is set to zero, and the first statement is flagged. A START Assembler instruction at any other point in the program is disregarded from a generation viewpoint but is flagged as an error.

Chart 7. Example of START Instruction

NAME	OPERATION	OPERAND	COMMENTS
1 2 3 4 5 6 7 8	9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71	
MAIN	START	X'100'	START AT LOCATION (256)10

ORG - Set Location Counter

♦ The ORG Assembler instruction allows the programmer to change the Location Counter. It can be written at any point in the program and as many times as desired.

The format is as follows:

NAME	OPERATION	OPERAND
Not Used	ORG	S

The new Location Counter value, S, is the same as in the machine instructions. It can be a symbol, an asterisk (see Chart 8), or a self-defining value. When S is a symbol, it must have been previously defined. The Location Counter is set to the value of the operand. Any value less than the original value of the Location Counter (from the START instruction) is considered an error. It is used but it is flagged in the listing. A blank or invalid operand is disregarded and flagged in the listing. The program continues to compile using the previous Location Counter setting if the operand field is invalid. The Name field is not used by the Assembler.

Chart 8. Example of ORG Instruction

NAME	OPERATION	OPERAND	COMMENTS
	ORG	*+100	INCREASE THE LOCATION COUNTER BY (100)10. THE NEXT
*		100	LOCATIONS ARE ENTIRELY DISREGARDED

END - End of Program

♦ The END assembly directive is used to indicate that all the statements have been processed. It therefore must be the last statement in the user's program. The END instruction is also used to indicate to the Loader the starting location after the program has been loaded. The format of an END instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	END	S

The program starting location, S, is the same as in the machine instructions. It can be a symbol, an asterisk, a self-defining value, or a blank.

An invalid operand is treated as a blank operand; both conditions result in the generation of a blank address field on the load End card. The listing contains a flag if the operand is invalid.

The Assembler generates a special output card also known as an END card for use by the Loader.

DEFINITION INSTRUCTIONS

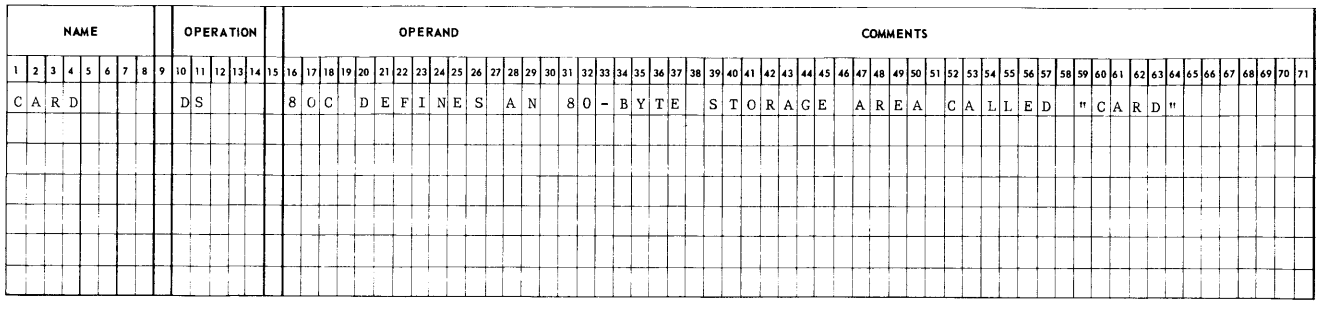
DS - Define Storage

♦ The Define Storage Assembler instruction is used to identify and reserve storage locations. It is the normal means of giving a name to work and input/output areas. The format is as follows:

NAME	OPERATION	OPERAND
Symbol	DS	A decimal count of the number of bytes to be reserved followed by the Symbol C.

The length of a DS operand field is only limited by the difference between the core size and the present value of the Assembler's Location Counter. A DS statement does not clear the storage it reserves. The Assembler does not generate any output cards for these areas. Therefore, the user should not expect the area to be cleared to zeros or blanks. The Assembler terminates the object program text card it is currently generating and starts a new one for the first statement following the Define Storage statement. The programmer reduces the size of the output deck by defining his DS statement with as few broken sequences as possible.

Chart 9. Example of DS Instruction



DC - Define Constant

♦ This instruction is used for generating data constants within a program. The constants specified may consist of any valid EBCDIC character, hexadecimal digit, or memory address. The DC instruction's statement format is as follows:

NAME	OPERATION	OPERAND
Symbol or Blank	DC	A single descriptor specifying the constant in one of the formats defined below.

There are three types of descriptors for constants. Each descriptor specifies both the type of constant, and the constant itself. The formats of the constant descriptors are as follows:

C'z'
X'z'
A(z)

Table 3 defines the format of the three types of constants.

Table 3. Constant Formats

Description	Constant Type	Machine Format	Implied Length in Bytes
C	Character	Eight-bit EBCDIC	Number of characters within 'z' not including quotes.
X	Hexadecimal	Fixed-Point Binary	Half the number of characters within 'z' not including quotes.
A	A symbol, asterisk or self-defining value	Fixed-Point Binary	2

Z represents the constant, itself, and appears within quotes or parenthesis.

As shown in Table 3, the length of a constant is implied by the type of descriptor used. The maximum length of any C or X type constant is 16 bytes. The A type constant is always assigned two bytes. If constants of greater length are desired, they may be specified by the use of successive DC statements.

DC - Define Constant (Cont'd)

Constants and self-defining values use the same format C'z' and X'z' (character and hexadecimal values). When these values are used as operands associated with a DC operation code, their maximum number, as stated above, may be 16 EBCDIC characters or 32 hexadecimal numbers. When they are self-defined values used to define relative addresses, masks, operand lengths, etc., their maximum is four decimal characters or four hexadecimal characters.

Character Constants - C

♦ A character constant may consist of a maximum of 16 valid characters. Valid characters consist of punch combinations which translate into an eight-bit EBCDIC code character. The 70/15 Processor Reference Manual contains a list of the 256 punch combinations along with their internal eight-bit codes and their associated printer symbols.

Each character of the constant is converted into a single eight-bit byte. In the example below, (Chart 10, Statement 1) the constant name HER has a length of six bytes.

Quotation marks are the legal characters in the C-Type constant. A single quotation mark is specified by two successive single quotation marks. (See Chart 10, Statement 2. This constant will be assigned five bytes of memory.)

Chart 10. Examples of Character Constant - C

NAME	OPERATION	OPERAND	COMMENTS
HER	DC	C'FEMALE'	STATEMENT 1
	DC	C'IS N''T'	STATEMENT 2

Hexadecimal Constants - X

♦ Hexadecimal constants may consist of, and only of, the 16 hexadecimal digits:

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

The number of digits specified in the constant should be an even number, up to and including a maximum of 32 hexadecimal digits. Each hexadecimal digit requires four binary bits to represent it within memory. For this reason, hexadecimal digits are packed two per eight-bit byte in internal storage. An example of a hexadecimal constant definition is shown in Chart 11.

If the hexadecimal description specifies an odd number of digits, a zero digit is appended to the leftmost end of the constant, and a flag is placed on the statement in the listing.

If the hexadecimal descriptor specifies more than 32 digits, the leftmost 32 digits are retained and the remaining rightmost characters are ignored. This condition causes the statement to be flagged in the listing.

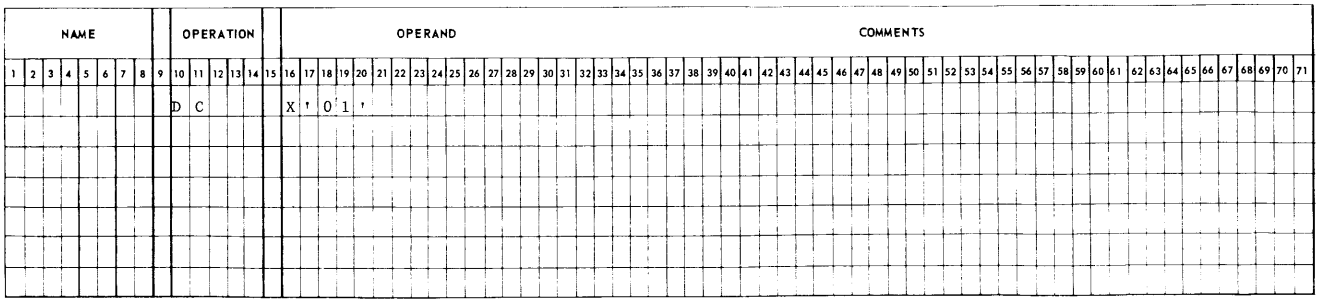
Expression Constants - A

♦ An expression constant is normally used to define an address. The programmer can also use it when it is desired to define a binary constant in a decimal format. Two bytes are always generated regardless of the value specified. The constant is right-justified, and the leftmost byte is located in the next available even position. A zero byte is inserted if a byte must be skipped. The constant is defined within parenthesis. Since an expression constant is frequently used to represent an address, it is often called an address constant. The format is as follows:

NAME	OPERATION	OPERAND
ADDR	DC	A(S)

S can be a symbol, an asterisk, or a self-defining value (see Chart 12).

Chart 11. Example of Hexadecimal Constant - X



Expression Constants - A (Cont'd)

The expression constant allows a self-defining value to have one additional feature. A minus sign (-) can precede the value. The Assembler generates the two's complement of the value within the 16 bits. Any self-defining value that exceeds 16 bits is truncated at the high-order end and causes a flag indication on the listing (see Chart 13).

Chart 12. Example of Expression Constant - A

NAME	OPERATION	OPERAND	COMMENTS
ACON	DC	A(CARD+80)	

Chart 13. Example of Expression Constant - A

NAME	OPERATION	OPERAND	COMMENTS
TBLE	DC	A(-1)	THIS WILL GENERATE A BINARY CONSTANT EQUAL TO (111
*		1111111111111111)	

PROGRAM LINKING INSTRUCTIONS

♦ The ENTRY and EXTRN Assembler verbs provide the facility to load and execute, as a unit, programs which are independently assembled. The linkage between the program is provided by defining a symbol in one program and referring to the symbol as an operand in the second program. The location associated with the symbol in the defining program is called an entry point. The references to the entry point in the other assembled programs are called external references. Entry points are specified by ENTRY or START instructions. External references are defined by EXTRN instructions.

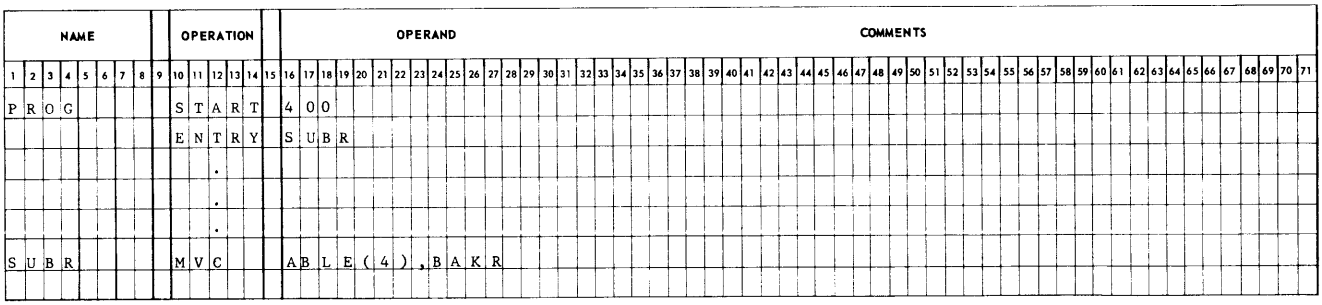
ENTRY - Identify Entry Point

♦ An entry point is defined by an ENTRY instruction. Entry points cannot be specified by relative addressing, therefore, each entry point must be defined by a separate ENTRY verb. The one exception is the START instruction which also defines an entry point. The format is as follows:

NAME	OPERATION	OPERAND
Not Used	ENTRY	Symbol

The symbol in the Operand field must be defined in a name field somewhere within this assembly program deck. The symbol can be referred to by other operands within this program. All ENTRY statements must follow the START instruction. An invalid ENTRY statement is flagged in the listing. An invalid statement could be caused by an undefined symbol, or invalid operand, or an incorrectly placed ENTRY statement. (See Chart 14.)

Chart 14. Example of ENTRY Instruction



EXTRN - Identify External Symbol

♦ A reference to an external symbol is defined by an EXTRN instruction. A separate EXTRN statement must be written for every external location referred to by a program. The format of an EXTRN statement is as follows:

NAME	OPERATION	OPERAND
Not Used	EXTRN	Symbol

All EXTRN statements must follow ENTRY statements. If no ENTRY statements appear, the EXTRNs must follow the START instruction. The Name field is disregarded. An invalid EXTRN instruction is flagged in the assembly listing. An invalid EXTRN could be caused by an invalid operand or the improper placement of the EXTRN statement.

Order of Statements

♦ The order of statements for an assembly source program that will be linked to other programs at object program running time must be:

```
START

ENTRY 1
ENTRY 2
      o
ENTRY n

EXTRN 1
EXTRN 2
EXTRN o
      o
      o
EXTRN n

OPERATION STATEMENTS (Including DS, DC, ORG Instructions).

      o
      o
      o
      o

END
```

EXTENDED MNEMONIC INSTRUCTION

B - Unconditional Branch

♦ The Assembler recognizes a B instruction as an unconditional transfer of control to the location specified by the Operand field. Its purpose is to simplify the writing of Unconditional Branch instructions. The format of an Unconditional Branch instruction is as follows:

NAME	OPERATION	OPERAND
Symbol	B	S

where: S can be a symbol, asterisk, or a self-defining value. The above statement could also be expressed as shown in Chart 15.

Chart 15. Example of Unconditional Branch Instruction

NAME	OPERATION	OPERAND	COMMENTS
NAME	BC	X'F',S	

INPUT/OUTPUT CONTROL SYSTEM (IOCS)

INTRODUCTION

♦ The RCA 70/15 Input/Output Control System (IOCS) is a set of related routines that aids the programmer in programming input/output operations for the input/output devices listed in Table 4. By including a set of coded parameter requests (in 70/15 Assembly language) in his program, the programmer can reduce and simplify the amount of coding necessary for input/output operations.

The programmer supplies only the parameters in his program. The IOCS, which at execution time is a subroutine of the user program, uses the parameters to perform the desired input/output operations. Then IOCS returns control to the user program.

IOCS performs error detection, but does not, however, provide the programmer with device interchangeability. Devices must be selected by the programmer in his program. In order that a program may operate with a variety of input/output devices, the programmer must make provisions for modifying or bypassing control functions, recognizing flag records or processor functions, and interpreting abnormal and alarm returns from the IOCS.

Input/output operations can be executed in a Simultaneous or a Nonsimultaneous mode. Operating in the Simultaneous mode means that processing can occur at the same time that one or more input/output operations are occurring. IOCS does this by using the Read Auxiliary instruction and buffered devices. The programmer specifies to the IOCS whether or not simultaneous or nonsimultaneous processing is to take place. (See device parameter area page 32.)

IOCS Output

♦ Output of the IOCS is the initiation or completion of the I/O function requested. There are three results of an I/O request which are as follows:

1. Control can return to the user program's normal return address under the following conditions:

   (a) If nonsimultaneous processing is specified, control is returned upon completion of the input/output operation.

   (b) If simultaneous processing is specified, control is returned upon initiation of the input/output operation. The operation is not initiated if an error is detected from the previous input/output operation.

2. Control is given to the user's alarm or abnormal return address when an alarm or abnormal condition is detected.

IOCS Output (Cont'd)

3. An error halt occurs in the IOCS if an inoperable condition code is detected upon designating the I/O device. When the error occurs, 8F is displayed in the M register on the Console. The trunk and unit of the device designated when the error occurred is stored in standard location $P+5.

IOCS Versions

♦ Two versions of IOCS are provided: one for systems that have magnetic tape units, the other for systems that do not. These versions are called the magnetic tape version, which requires approximately 950 bytes, and the nonmagnetic tape version, which requires approximately 450 bytes. Both versions are supplied as a deck of source language cards.

EQUIPMENT CONTROL CAPABILITIES

♦ The IOCS is capable of controlling the operation of the peripheral devices given in Table 4.

Table 4. Devices Controlled by IOCS

Device	Model
Magnetic Tape Unit	70/432-1,-2 70/442-1,-2
Magnetic Tape Station (7- or 9- channel)	70/445-1,-2
Card Reader	70/237-10,-20,-30
Videoscan Document Reader (Demand feed only.)	70/251-10,-21,-22,-30
Paper Tape Reader/Punch	70/221-10,-11,-20,-21
Input/Output Typewriter	70/216
Card Punch	70/234-10,-11 70/236-10,-11
Bill Feed Printer (continuous forms only)	70/248-10,-11
Printer	70/242-10,-20 70/243-10,-20

Devices not controlled by IOCS are:

1. Videoscan Document Reader, when used in any way other than Demand Feed.

2. Bill Feed Printer, when used for card reading and card printing.

3. Reader-Punch. This is the Model 70/236 Card Punch with the Reader-Punch option.

EQUIPMENT CONTROL CAPABILITIES (Cont'd)

4. Data Exchange Control, Model 70/627.

5. Communication Control, Model 70/652.

Notes:

1. The Reader-Punch can have its operations controlled by IOCS, because it is controlled by ordinary Read, Write, and Write Control instructions. However, the timing of this device may cause it to be controlled improperly by the IOCS.

2. The Bill Feed Printer requires special purpose routines for its input/output.

3. The Data Exchange Control and the Communication Control both use the interrupt feature, and the IOCS does not provide for interrupts. (The Input/Output Typewriter, which uses the interrupt feature, can be controlled by IOCS. However, the programmer must provice his own code for all interrupt analysis and processing.)

4. For the IOCS to control the Videoscan Document Reader, the following conditions must prevail:

   (a) Only demand feed operations can be attempted.

   (b) The WCM (Write Control Mandatory) key on the document reader must be depressed for documents to be selected into the same stacker. This causes a manual override of the hardware stacker selection. All documents are routed to the reject stacker.

COMBINING IOCS WITH THE USER PROGRAM

♦ To use the 70/15 Input/Output Control System, the programmer must include certain coding in his program. This coding consists of two classes: (1) the instructions needed to define the device parameter area, and (2) the IOCS calling sequences.

A Device Parameter area is a portion of memory that must be set aside for storing information that concerns an input/output device. One parameter area must be reserved for each device used by the program. The area is reserved by including a set of Define Constant instructions, in a specified order in the user program, which supplies information needed by the IOCS.

An IOCS calling sequence is a set of instructions that serves the twofold purpose of supplying information to the IOCS and establishing a reentry address that enables the IOCS to return to the user program.

The IOCS is a closed subroutine with seven ENTRY points. These points are IN, OUT, CHK, CTRL, RWD, RWDA, and TMRK. Each entry is made from a programmer-specified calling sequence. Based on the entry point, the calling sequence, and the programmer-supplied device parameters, the IOCS executes the function desired by the programmer and returns control to a return address specified by the programmer.

COMBINING IOCS WITH THE USER PROGRAM (Cont'd)

There are three return addresses associated with each calling sequence. These are the normal return address specified in the calling sequence, and the alarm address, and the abnormal return address specified in the Device Parameter Area.

The IOCS is incorporated into a program by any of the three following methods:

1. Assembling the IOCS with the user program.
2. Linking the user program with the IOCS by means of a binding run.
3. Loading the IOCS with a user program that has designated the entrances to the IOCS within the user's program as EXTRN's.

Linking Source IOCS and Source User Program at Assembly Time

♦ To include the source IOCS routine in the user source program at assembly time, the IOCS in source (assembly language) form can be placed with the user's source program as one program deck (or one source program on magnetic tape or paper tape) and assembled. It must be remembered that this is now one source program and must contain only one START card and one END card. Also, all ENTRY and EXTRN cards that refer to points in the IOCS must be removed before assembly. This composite deck is the input to the Assembler. The output is an executable, object program including IOCS. The output can be on cards or 80-character card images on magnetic tape or paper tape.

Linking IOCS and User Program In Binding Run

♦ To link the object IOCS and object user program in a binding run the IOCS source deck is supplied with the following ENTRY statements:

NAME	OPERATION	OPERAND(1)
	ENTRY	IN
	ENTRY	OUT
	ENTRY	RWD
	ENTRY	RWDA
	ENTRY	CTRL
	ENTRY	TMRK
	ENTRY	CHK

where IN, OUT, etc., are tags in the NAME column of the IOCS source program entry points.

(1) All references to RWD, RWDA, and TMRK apply only to the magnetic tape version of the IOCS.

Linking IOCS and User Program In Binding Run (Cont'd)

The user program must contain the following EXTRN statements:

NAME	OPERATION	OPERAND(1)
	EXTRN	IN
	EXTRN	OUT
	EXTRN	RWD
	EXTRN	RWDA
	EXTRN	CTRL
	EXTRN	TMRK
	EXTRN	CHK

The IOCS (with the ENTRY's) must be assembled, and the user program (with the EXTRN's) must be assembled. These two object programs, with appropriate parameter cards (made up by the programmer), are input to the Binder routine supplied by RCA. Output is one object program that combines IOCS and the user object program.

Linking IOCS and User Program at Load Time

♦ To link the object IOCS and object user program at load time, the IOCS source deck is supplied with the following ENTRY statements:

NAME	OPERATOR	OPERAND(1)
	ENTRY	IN
	ENTRY	OUT
	ENTRY	RWD
	ENTRY	RWDA
	ENTRY	CTRL
	ENTRY	TMRK
	ENTRY	CHK

where IN, OUT, etc., are tags in the NAME column somewhere in the IOCS source program.

(1) All references to RWD, RWDA, and TMRK apply only to the magnetic tape version of the IOCS.

Linking IOCS and User Program at Load Time (Cont'd)

The user program must contain the following EXTRN statements:

NAME	OPERATOR	OPERAND(1)
	EXTRN	IN
	EXTRN	OUT
	EXTRN	RWD
	EXTRN	RWDA
	EXTRN	CTRL
	EXTRN	TMRK
	EXTRN	CHK

The IOCS (with the ENTRY's) must be assembled, and the user program (with the EXTRN's) must be assembled. These two object programs, with appropriate parameter cards (made up by the user), are input to the Relocatable Loader (a routine supplied by RCA). Output is one object program in memory that is ready for execution. There is no physical output, i.e., punched cards, magnetic tape, etc.

WRITING CALLING SEQUENCES AND DEVICE PARAMETER AREA

Device Parameter Area

♦ The Device Parameter area required by the Input/Output System contains information pertinent to the device. A Device Parameter area must be defined for each device through the use of seven Assembler statements. (See Chart 16.) If simultaneous processing is specified, only one Device Parameter area per device can be defined. If nonsimultaneous processing is specified, more than one Device Parameter area per device can be defined.

Simultaneity Indicator

♦ The entry DC A(S) , (second line of coding, Chart 16.), in the Device Parameter area where

S = 0 or 00 means nonsimultaneous and

S = 1 or 01 means simultaneous,

causes 2 bytes to be reserved. Only one byte (the right-most) is used for the simultaneity indicator. The left-most byte is used by IOCS as a device servicing indicator (additional to the Trunk Status table). This device pending indicator may have one of the following two values:

01_{16} = post status and/or error checking to be accomplished.

00_{16} = device has been serviced and nothing is pending.

---

(1) All references to RWD, RWDA, and TMRK apply only to the magnetic tape version of the IOCS.

Simultaneity Indicator (Cont'd)

Normally the device pending indicator is set by the IOCS. This is done so that a check (CHK), issued before the first input/output operation to a device, is effectively treated as a no-op. If the programmer desires to execute a CHK function on a device before any input/output operation on that device has been performed, he can set the device pending indicator on by defining S = 11 or S = 10. Then, an initial CHK function is executed, rather than ignored.

IOCS Calling Sequences

♦ To execute an I/O function, the IOCS is entered by one of the following function names from a calling sequence coded by the programmer:

IN - The IN function transfers data from input devices such as magnetic tape, card reader, etc., into memory.

OUT - The OUT function transfers data from memory to output devices.

CHK - The CHK function is used to sense and to store status information pertaining to a particular device.

CTRL - CTRL is used to perform such control functions as selecting punch stackers, printer carriage control functions, magnetic tape movement, etc.

RWD - The RWD function is used only to rewind magnetic tapes to BT marker.

RWDA - The RWDA function is used only to rewind and disconnect magnetic tape.

TMRK - The TMRK function is used to write a tape mark for either 7-channel or 9-channel magnetic tape.

To execute an IN, an OUT, RWD, RWDA, TMRK, or a CHK function, the user program must move the appropriate address of the Device Parameter area and the normal return address to the standard area $P. It must then execute an Unconditional Branch to the appropriate entry point in the I/O Control System.

To execute a CTRL function, the user program must move the address of the Device Parameter area and normal return address to $P as above. However, the program must also include the hexadecimal representation for the desired control function in the calling sequence, and then execute an Unconditional Branch to the entry point. The calling sequences are a standard means of performing these operations.

Chart 16. Device Parameter Area

NAME	OPERATION	OPERAND	COMMENTS
NAME	DC	A(LD)	LOGICAL DEVICE NO. (0-9) STATEMENT 1
	DC	A(S)	0-NON-SIMULTANEOUS 1-SIMULTANEOUS STATEMENT 2
	DC	A(D1)	I/O AREA STARTING ADDRESS STATEMENT 3
	DC	A(D2)	I/O AREA END ADDRESS STATEMENT 4
	DC	A(ABNL)	ABNORMAL RETURN ADDRESS STATEMENT 5
	DC	A(ALRM)	ALARM RETURN ADDRESS STATEMENT 6
	DS	12C	STORAGE FOR STATUS INFO (4C-CARD) STATEMENT 7

Device Parameter Area (See Chart 16)

♦ Statement 1 This statement contains the name that is to be used in the calling sequences that refer to this device. The constant, defined within the parentheses, specifies the logical device number. The allowable values are 00-09. An actual device is assigned to the logical device by the I/O Define card at load time.

Note: The allowable values of 00-09 are imposed by the Loader. If the entry is inserted by the user by some means other than by an I/O Define card, the allowable values are 00-FF.

Statement 2 This statement contains the constant, defined within the parentheses, that specifies whether simultaneous or nonsimultaneous processing is desired for this device. Zero (0) indicates nonsimultaneous processing; one (1) indicates simultaneous processing.

Note: The first byte of this two-byte constant is used as a service-pending indicator by the IOCS. The programmer is not to refer to this byte because its contents are variable.

Statement 3 This statement contains the value, defined within the parentheses, which must be the symbolic name assigned to the starting address of the I/O area. This may be the LHE (left-hand end) for forward operations or the RHE (right-hand end) for reverse operations. The user program can modify this value during program execution if desired.

Statement 4 This statement contains the value, defined within the parentheses, which must be the symbolic name assigned to the ending address of the I/O area. This may be the RHE for forward operations or the LHE for reverse operations. The user program can modify this value during program execution if desired.

Statement 5 This statement contains the value, defined within the parentheses, which must be the name assigned to the first instruction in the user program's abnormal return subroutine. Control is transferred to this address under certain conditions, called abnormal conditions.

Statement 6 This statement contains the value, defined within the parentheses, which must be the symbolic name assigned to the first instruction of the user program's alarm return subroutine. Control is transferred to this address under certain conditions, called alarm conditions.

Note: If the programmer desires to change the alarm or abnormal addresses in statements 5 and 6, he should ensure that no instruction is pending before making the change. When changing the addresses in state-

Device Parameter Area (Cont'd)

Note: (Cont'd) ments 5 and 6, it must be remembered that it is possible for the IOCS to return to either of these returns at two different times.

Statement 7 This statement allocates a 12-byte working storage area in which status information, the final A-address, and other data are stored during program execution. If the nontape version of the I/O Control System is used, only four bytes of working storage must be provided. If the magnetic tape version is used, statement 7 of all Device Parameter areas, regardless of device type, must allow for a 12-byte area.

The device parameters will subsequently appear in memory as follows:

Logical Device #	Device Pending Indicator	Simultaneity Indicator	Starting Address	Ending Address	Abnormal Return Address	Alarm Return Address
+0 +1	+2	+3	+4 +5	+6 +7	+8 +9	+10 +11

A-Final Address	Standard Device Byte	I/O Sense Byte	Working Storage for I/O Control			
			Cyclic Parity Character (9 Level Tape)	OP Code	Trunk and Device	Dl Address
+12 +13	+14	+15	+16	+17	+18	+19 +20

(Cont.)	
D2 Address	Write Control Character
+21 +22	+23

The first 12 bytes are the parameters defined by the programmer. The second 12 bytes are reserved by the programmer and filled in by the IOCS. If the nontape version is being used, only four bytes are reserved by the program and used by the IOCS.

After termination of every operation, or when control is returned to either the abnormal or alarm addresses, the A-final address, the standard device byte, and the I/O sense byte are posted in the Device Parameter area for that device. In magnetic tape operations, the entire I/O instruction is stored in bytes +17 -- +22 for error recovery (rollback) procedures.

The IN Calling Sequence

♦ The IN calling sequence, required to execute an input function, is shown in Chart 17.

Statement 1 This statement contains a Move instruction that transfers the return address (specified by Statement 3) and the address of the Device Parameter area (specified by Statement 4) to the standard area $P.

The IN Calling Sequence (Cont'd)

Statement 2 — This statement contains an Unconditional Branch to the IN entry point of the IOCS.

Statement 3 — This statement contains the address to which control is returned under normal circumstances.

Statement 4 — This statement contains the name assigned to the Device Parameter area describing the device to which this command is issued.

The OUT Calling Sequence

♦ The OUT calling sequence, required to execute an Output function, is shown in Chart 18.

Statement 1 — This statement contains a Move instruction that transfers the return address (specified by Statement 3) and the address of the Device Parameter area (specified by Statement 4) to the standard area $P.

Statement 2 — This statement contains an Unconditional Branch to the OUT entry point of the IOCS.

Statement 3 — This statement contains the address to which control is returned under normal circumstances.

Statement 4 — This statement contains the name assigned to the Device Parameter area describing the device to which this command is issued.

Chart 17. IN Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P STATEMENT 1
	B	IN	BRANCH TO IN STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4

Chart 18. OUT Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P STATEMENT 1
	B	OUT	BRANCH TO OUT STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4

The RWD Calling Sequence

♦ The RWD calling sequence, required to execute a magnetic tape rewind to BT markers is shown in Chart 19.

Statement 1 — This statement contains a Move instruction that moves the normal return address (specified by Statement 3), and the address of the Device Parameter area (specified by Statement 4), to the standard location in lower memory called $P.

Statement 2 — This statement contains an Unconditional Branch to the RWD entry point of the IOCS.

Statement 3 — This statement contains the address to which control is returned after the RWD operation is completed.

Statement 4 — This statement contains the name assigned to the Device Parameter area that defines the device to which this command is issued.

RWDA Calling Sequence

♦ The RWDA calling sequence, required to execute a magnetic tape rewind and disconnect, is shown in Chart 20.

Statement 1 — This statement contains a Move instruction which moves the normal return address (specified by Statement 3) and the address of the Device Parameter area (specified by Statement 4) to a standard location in lower memory called $P.

Statement 2 — This statement contains an Unconditional Branch to the RWDA entry point of the IOCS.

Chart 19. RWD Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P STATEMENT 1
	B	RWD	BRANCH TO RWD STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4

Chart 20. RWDA Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P STATEMENT 1
	B	RWDA	BRANCH TO RWDA STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4

RWDA Calling Sequence (Cont'd)

Statement 3 This statement contains the address to which control is returned after the RWDA operation is completed.

Statement 4 This statement contains the name assigned to the Device Parameter area that defines the device to which this command is issued.

The TMRK Calling Sequence

♦ The TMRK calling sequence, required to write a tape mark (either 7- or 9-channel) on magnetic tape, is shown in Chart 21.

Statement 1 This statement contains a Move instruction that moves the normal return address (specified by Statement 3) and the address of the Device Parameter area (specified by Statement 4) to a standard location in lower memory called $P.

Statement 2 This statement contains an Unconditional Branch to the TMRK entry point of the IOCS.

Statement 3 This statement contains the address to which control is returned after the TMRK operation is completed.

Statement 4 This statement contains the name assigned to the Device Parameter area that defines the device to which this command is issued.

The CTRL Calling Sequence

♦ The CTRL calling sequence, required to execute a write control function, is shown in Chart 22.

Statement 1 This statement contains a Move instruction that moves the normal return address (specified by Statement 3), the address of the Device Parameter area (specified by Statement

Chart 21. TMRK Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P STATEMENT 1
	B	TMRK	BRANCH TO TMRK STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4

Chart 22. CTRL Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(5),*+10	MOVE 5 BYTES TO $P STATEMENT 1
	B	CTRL	BRANCH TO CTRL STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA STATEMENT 4
	DC	X'02'	CONTROL OPERATION STATEMENT 5

The CTRL Calling Sequence (Cont'd)

4), and the control information (specified by Statement 5) to a standard location in lower memory called $P.

Statement 2 — This statement contains an Unconditional Branch to the CTRL entry point of the IOCS.

Statement 3 — This statement contains the address to which control is returned after the CTRL operation has been completed.

Statement 4 — This statement contains the name assigned to the Device Parameter area that defines the device to which this command is to be issued.

Statement 5 — This statement contains the control information that is sent to the device. This data is written in the form of two hexadecimal characters that generate the proper binary configuration required by the device. The formats of the various control bytes, by device, are defined on page . The control byte is sent to the device by means of a Write Control instruction by the IOCS.

Table 5 contains the two control characters that are issued to various devices. It should be noted that any control to the printer that specifies paper advance following the next write is voided by the next OUT issued to the printer.

Table 5. Control Characters

Model, Device	Hexadecimal Control Characters	Meaning
70/216 I/O Typewriter		Write control is not possible on the 70/216 I/O Typewriter
70/221 Paper Tape Reader/Punch		Write control is not possible on 70/221 Paper Tape Reader/Punch
70/234 Card Punch	01	Set translate mode.
	02	Set column binary mode.
70/236 Card Punch	01	Set translate mode.
	02	Set column binary mode.
Additional Control Bytes for 70/236 Card Punch (Reader/Punch)(2)	04	Override hardware stacker selection made at wait station No. 2 (after card has been read).
	08	Override hardware stacker selection at postpunch station (after card has been punched).

(2) The Videoscan Document Reader, the Bill Feed Printer, and the Reader/Punch require special consideration. See the section on equipment control capabilities.

The CTRL Calling Sequence (Cont'd)

Table 5. Control Characters (Cont'd)

Model, Device	Hexadecimal Control Characters	Meaning
70/236 (cont'd)	10	Override stacker selection at post-punch read (after hole count check has been made). (There is a timing consideration in the use of this control byte. The instruction must be executed within 3.5 milliseconds after punching has been completed.)
	20	Move the transport one station. (This does not read a card.)
70/237 Card Reader	01	Select accept stacker.
	02	Select reject stacker.
	04	Set translate mode
	08	Set column binary mode.
70/432, 70/442, 70/445 Magnetic Tapes	01	Unwind to tape mark.
	02	Rewind to tape mark.
	08	Unwind one gap.
	10	Rewind one gap.
	20	Rewind to load point and disconnect.
	80	Rewind to BT marker.
70/242, 70/243 Printer or 70/249 Bill Feed Printer (used to print continuous forms only).(2)	4X	Paper advance immediately per line count specified by X.
	0X	Paper advance following next write, per line count specified by X.
	CX	Paper advance immediately to channel selected by X.
	8X	Paper advance following next write to channel specified by X.
70/251 Document Reader (Demand Feed only)(2)	01	Select accept stacker.
	02	Select reject stacker.
	04	Set translate mode.
	08	Set column binary mode.
	10	Demand feed.

(2) The Videoscan Document Reader, the Bill Feed Printer, and the Reader/Punch require special consideration. See the section on equipment control capabilities.

The CHK Calling Sequence

♦ The CHK calling sequence, required to execute a check function, is shown in Chart 23.

Statement 1 — This statement contains a Move instruction which transfers the return address (specified by Statement 3) and the address of the Device Parameter area (specified by Statement 4) to the standard area $P.

Statement 2 — This statement contains an Unconditional Branch to the CHK entry point of the IOCS.

Statement 3 — This statement contains the address to which control is returned under normal circumstances.

Statement 4 — This statement contains the name assigned to the Device Parameter area describing the device to which this command is to be issued.

Chart 23. CHK Calling Sequence

NAME	OPERATION	OPERAND	COMMENTS
	MVC	$P(4),*+10	MOVE 4 BYTES TO $P — STATEMENT 1
	B	CHK	BRANCH TO CHK — STATEMENT 2
	DC	A(RTRN)	NORMAL RETURN ADDRESS — STATEMENT 3
	DC	A(NAME)	ADDRESS OF DEVICE PARAMETER AREA — STATEMENT 4

FUNCTIONS OF THE IOCS

♦ The following functions are performed by the IOCS each time the routine is entered at any of its seven entry points.

1. The Logical Device number, specified in the Device Parameter area, is translated into the actual device number assigned by the I/O Define Card at load time.

2. When simultaneous processing is specified, a check (CHK) is performed before each I/O operation (except the first) to test the termination of the previous operation. When nonsimultaneous processing is specified, a check is performed after the I/O operation.

3. The A-final address, the standard device byte, and the I/O sense byte are posted in the Device Parameter area as part of the activity of the check (CHK) function.

4. The device pending indicator in the Device Parameter area is then checked to see if the last reference to this device was serviced. If necessary, the standard device byte and the I/O sense byte are checked. If other than a normal condition is detected, steps are taken to perform Rollback or return to the alarm or abnormal return addresses.

5. If neither the device pending indicator nor an exceptional condition is detected, the IOCS checks to see if this was a CHK ENTRY. If it was a CHK, control is returned to the user's normal address. If it was not a CHK ENTRY, control is turned over to the I/O issue portion of the IOCS.

6. The I/O issue portion of the IOCS acts depending upon the ENTRY point, the simultaneity indicator in the device parameters, the device type in the Device Correspondence table, and the relation of the starting and ending addresses in the Device Parameter area.

Exceptional Conditions

♦ Exceptional conditions are classified into the following two groups:

1. Abnormal

2. Alarm

When an exceptional condition arises during the execution of an input/output operation, it results in a transfer of control. This transfer of control is to the address specified by the programmer in the Device Parameter area for processing that type of condition rather than to the normal return address as specified in the calling sequence of the I/O request.

The IOCS usually returns control to the user's normal return address. When an abnormal or alarm condition occurs, IOCS returns control to the programmer-specified Alarm routine address or to the programmer-specified Abnormal routine address. The programmer specifies these return addresses in the Device Parameter area.

Exceptional Conditions (Cont'd)

There are five abnormal conditions, all concerned with end-of-file or end-of-forms. The many alarm conditions are usually concerned with equipment malfunctions, invalid characters, and illegal operations.

Read parity error rollback and read-after-write rollback are automatically performed by the IOCS for magnetic tapes.

Table 6 specifies the I/O sense bytes of the various device types, showing the bit that is set as a result of the described exceptional condition, and the return address to which the IOCS will transfer control.

IOCS determines the type of error and performs error recovery if appropriate. Should the IOCS return to the alarm address after an I/O attempt to read or write a magnetic tape, the tape is positioned before the bad record on a read reverse, after the bad record on a read forward, or before the next available segment of tape on a write.

Nonsimultaneous Mode IN Function

Card Readers

♦ 1. A Read Forward instruction is issued.

2. Data is transferred from the card reader to memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when the lesser of (D2-D1+1) or 80 bytes have been transferred.

3. A CHK is performed, and the appropriate routine (abnormal or alarm) in the user program is entered if an error exists.

4. The A-final address, standard device byte, and I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the normal return address if no errors occur.

Magnetic Tape

♦ 1. The packing density, etc., is set if necessary.

2. If $D_1 < D_2$, A Read Forward instruction is issued. If $D_1 > D_2$, a Read Reverse instruction is issued.

3. Data is transferred to memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are transferred.

4. A check (CHK) is performed and the appropriate routine (abnormal or alarm) in the user program is entered if an error exists.

5. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

Table 6. I/O Sense Bytes

Device \ Return Address / Bit Positive Set	Alarm	Alarm	Alarm	Alarm	Alarm	Alarm	Abnormal	Alarm
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
70/236 Card Punch 70/234	Punch Comparison Error (PCE)	Punch Memory Parity Error	Not Used	Transmission Parity Error	Intervention Required Stacker full Hopper empty Chip box full	Hold	Not Used	Illegal Operation
70/237 70/232 Card Reader	Read Error	Service Request Not Honored	Stacker Selection Too Late	Invalid Punch Code	Not Used	Hold	Not Used	Illegal Operation
70/251 Document Reader	Read Error	Service Request Not Honored	Stacker Selection Too Late	Unreadable Data	Feed Error	Not Used	Read Late	Illegal Operation
70/221 Paper Tape Reader	Parity Error	Service Request Not Honored	Not Used	Not Used	Intervention Required	Hold	Not Used	Illegal Operation
Punch	Parity Error	Not Used	Not Used	Not Used	Intervention Required	Hold	Low Tape	Illegal Operation
70/242 70/243 Printer	Print Error	Not Used	Not Used	Channel Parity Error	Invalid Code	Hold	Low Paper	Illegal Operation
70/214 Input/Output Typewriter	Write Error	Time Lapse	Not Used	Transmission Parity Error	Data Error	Not Used	Not Used	Illegal Operation
70/432 Magnetic Tape Unit 70/442 Magnetic Tape Unit 70/445 Magnetic Tape Station	*Read Error or Read After Write Error	Service Request Not Honored	Data block Greater Than Count	Transmission Error	Magnetic Tape Alarm	BT/ET	Tape Mark	Illegal Operation
70/249 Bill Feed Printer	Channel 12 Sensed	Channel 9 Sensed	Print Check	Transmission Parity Error	Code Error	Manual Service in Progress (Hold)	End-of-Forms	Illegal Operation

*The return to the user program at the alarm address occurs only after rollback procedures in the IOCS have failed.

Magnetic Tape (Cont'd)

6. Control is returned to the normal return address if no errors occur.

Note 1: The nontape version of IOCS should not be used for configurations that include magnetic tapes.

Note 2: If data termination is caused by a (D2-D+1) byte transfer and a gap is not encountered, a data-greater-than-count condition exists. This error condition effects a return to the alarm return address in the user program.

Paper Tape Reader

♦ 1. If $D_1 < D_2$, a Read Forward instruction is issued. If $D_1 > D_2$, a Read Reverse instruction is issued.

2. Data is transferred to memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are read.

3. A check (CHK) is performed and the appropriate routine (abnormal or alarm) in the user program is entered if an error exists.

4. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the normal return address if no errors occur.

Input/Output Typewriter

♦ An I/O Device Request Interrupt occurs in the processor when an operator writes a message into memory by means of the Input/Output Typewriter. When the processor is interrupted, the cause of this interrupt is not immediately evident and must be determined by a user-programmed routine. After it is determined that the interrupt is a request to transmit from the Input/Output Typewriter, the user program then enters the IOCS through the IN entry point in the normal way, that is, by use of an IN calling sequence.

1. A Read Forward instruction is issued to the typewriter.

2. Data is transferred to memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are transferred, or the END key is pressed, or the ERROR key is pressed, or a 15-second time-out occurs.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

4. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the normal return address if no errors occur.

Nonsimultaneous Mode OUT Function

Card Punches

♦ 1. A Write instruction is issued.

Card Punches (Cont'd)

2. Data is transferred from memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) or 80 bytes are transferred.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

4. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the user's normal return address if no errors occur.

Magnetic Tape

♦ 1. A Write Control instruction is issued to set the packing density, if necessary.

Note: If data termination is caused by a (D2-D1+1) byte transfer and a gap is not encountered, a data-greater-than-count exists. This error condition effects a return to the alarm return address in the user program.

2. A Write instruction is issued.

3. Data is transferred from memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are written.

4. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

5. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

6. Control is returned to the normal return address if no errors occur.

Paper Tape Punch

♦ 1. A Write instruction is issued.

2. Data is transferred from memory beginning at the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are written.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

4. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the normal return address if no errors occur.

Input/Output Typewriter

♦ 1. A Write instruction is issued.

2. Data is transferred from memory beginning with the D_1 address, which is specified in the Device Parameter area, and terminating when (D2-D1+1) bytes are written.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

4. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

5. Control is returned to the normal return address if no errors occur.

Note: Error checking occurs after the execution of the Write instruction. However, error checking does not occur following the execution of a Write instruction that effects form control.

Printer

♦ 1. A Write Control instruction is issued to the printer to effect form control. The form control is specified by the programmer. The first byte of the line that is to be printed is interpreted as the control byte. This character is not printed. This control character has the following format:

2^0, 2^1, 2^2, 2^3 = a binary count of the number of lines to advance (0-15) or a binary number that selects one of 11 loop channels (1-11), depending on the setting of bit 2^7.

2^4, 2^5 = ignored.

$2^6 = 0$ - paper advance after print.

$2^6 = 1$ - paper advance before print.

$2^7 = 0$ - paper advance per count in 2^0 - 2^3.

$2^7 = 1$ - paper advance under control of paper tape loop channel selected by bits 2^0 - 2^3.

2. A Write instruction is issued.

3. Data is transferred from memory beginning at the D_1 address plus one byte and is terminated when (D2-D1+1) bytes are transferred, or the printer's buffer is full (132 or 160 bytes).

4. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists.

5. The A-final address, the standard device byte, and the I/O sense byte are stored in the Device Parameter area.

Printer (Cont'd)

6. Control is returned to the normal return address if no errors occur.

7. When specifying forms control and no printing is desired, at least part of a blank line should be generated after the forms control character. The starting and ending addresses of the message given in the Device Parameter area should not specify only a single character. This causes the IOCS to print a full line.

   Note: Error checking occurs after the execution of the Write instruction. However, error checking does not occur following the execution of a Write instruction that effects form control.

Simultaneous Mode IN Function

Card Readers

♦ 1. A Read Auxiliary instruction is issued.

2. After the instruction is successfully initiated, control is transferred to the normal return address in the user program.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists. This is a check for the previous I/O operation. On the first input/output operation to a device, a check is not performed.

4. Data is transferred into memory beginning at the D_1 address, which is specified in the Device Parameter area, and terminating when 80 bytes are transferred or the end of memory is reached. (Transfer of (D2-D1+1) bytes does not terminate the instruction.)

5. No check (CHK) is made until the user program refers to this device again.

Magnetic Tape

♦ 1. The packing density, etc. is set, if necessary.

2. A Read Auxiliary instruction is issued if $D_1 < D_2$. If $D_1 > D_2$, a Read Reverse instruction is issued.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists. This is a CHK for the previous I/O operation. On the first input/output operation to a device, a CHK is not performed.

4. After the instruction is successfully initiated, control is transferred to the normal return address in the user program.

5. Data is transferred into memory beginning at the D_1 address, which is specified in the Device Parameter area, and terminating when a gap is encountered or when the end of memory is reached.

Magnetic Tape (Cont'd)

6. No check (CHK) is made until the user program refers to this device again.

   Note: The nontape version of IOCS should not be used for configurations that include magnetic tape.

Paper Tape Reader

♦ 1. A Read Auxiliary instruction is issued if $D_1 < D_2$. If $D_1 > D_2$, a Read Reverse instruction is issued.

2. After the instruction is successfully initiated, control is transferred to the normal return address in the user program.

3. A check (CHK) is performed and the appropriate routine in the user program is entered if an error exists. This is a CHK for the previous I/O operation. On the first I/O operation to a device a CHK is not performed.

4. Data is transferred into memory beginning at the D_1 address, which is specified in the Device Parameter area, and terminating when a gap on tape is encountered or the end of memory is reached.

5. No check (CHK) is performed until the user program refers to this device again.

Input/Output Typewriter

♦ An IN function in the Simultaneous mode should not be issued to the Input/Output Typewriter.

Simultaneous Mode OUT Function

♦ The OUT functions, when performed in the Simultaneous mode, are identical to those in the Nonsimultaneous mode with the following exception: Control is transferred to the normal return address immediately after the Write Auxiliary instruction is issued. The CHK for an I/O operation is not given until the next calling sequence that refers to this device. The primary use of the OUT function in the Simultaneous mode is to use the buffered output devices.

Check Function (CHK)

♦ Entering the IOCS by means of the CHK entry causes a check to be made of the previous termination for the device referred to. The IOCS executes Rollback, if necessary, posts status conditions to the Device Parameter area, and, if no errors have occurred, transfers control to the normal return address. If a condition exists that is not normal, control is passed to the appropriate address (either alarm or abnormal) in the user program for processing.

The CHK function executes a Post Status instruction and an I/O Sense instruction. If the secondary indicator is set (bit 2^2 of the standard device byte), or if the device inoperable bit is set (bit 2^1 of the standard device byte), the IOCS branches to the user's abnormal or alarm address according to the conditions listed by device in Table 6 on page 44.

(CHK) (Cont'd)

The standard device byte is stored in the reserved memory location for that trunk and unit. The sense byte is stored internally in the IOCS program. Both are stored in the Device Parameter area.

Control Function (CTRL)

♦ There are three device control functions that are not automatically performed by the IOCS. These are:

1. Setting the Binary mode on card equipment.
2. Selection of output stackers on card equipment.
3. Control functions for the Videoscan Document Reader.

The control (CTRL) function permits the programmer to perform these functions. If the programmer esires his program to work with a variety of devices (device interchangeability), he should modify or bypass these CTRL functions accordingly.

Control operations that are not covered by the RWD, RWDA, or TMRK calling sequences must be done by using the CTRL calling sequence. Upon completion of the operation, a check (CHK) is performed.

Rewind and Disconnect Function (RWDA)

♦ The function of the RWDA calling sequence is to rewind and disconnect magnetic tapes only. When the operation is completed, control is transferred to the normal return address. If a RWDA is issued to a device other than to a magnetic tape, no I/O instruction is issued and control is immediately transferred to the normal return address.

Write a Tape Mark Function (TMRK)

♦ The function of the TMRK calling sequence is to write a tape mark on either 7- or 9-channel magnetic tape. When the operation is complete, control is transferred to the normal return address. If a TMRK is issued to a device other than to a magnetic tape, no I/O instruction is issued and control is immediately transferred to the normal return address.

Rewind Function (RWD)

♦ The function of the RWD is to rewind magnetic tape to the BT marker only. When the rewind has been initiated, control is transferred to the normal return address. If a RWD is issued to a device other than to a magnetic tape, no I/O instruction is issued and control is immediately transferred to the normal return address.

Programming Considerations

♦ 1. If simultaneous processing is specified, users of the IOCS must define only one Device Parameter area for each device. If the user wishes to execute an IN or OUT function in an area other than the area specified in the Device Parameter area, he must modify the starting and ending addresses indicated in the parameters.

2. If nonsimultaneous processing is specified or a check (CHK) function is executed before issuing the next I/O function, more than one Device Parameter area can be specified.

Programming Considerations (Cont'd)

3. The Read Reverse I/O command is available for use only by the magnetic tape version of the IOCS.

4. If simultaneous processing is specified, users of the IOCS must perform a check (CHK) on the final read (end-of-run) to determine the validity of the information.

5. The 70/15 IOCS performs physical input/output operations, but does not perform logical input/output operations. If a tape file contains batched records, it is the programmer's responsibility to process the individual records within the block. IOCS treats the block of records as a single input-output record.

   Note: The 70/221 Paper Tape Punch does not generate a gap automatically at the termination of a Write instruction. The programmer is responsible for generating gaps if they are desired. This is done by including one or more bytes containing all zeros (0000 0000) at the end of each block written.

DEVICE ASSIGNMENT

Introduction

♦ The logical device code, which appears on the first line of coding of the Device Parameter area, must be the same code for a given device as the code on the I/O Define card for that device. For example, if the Card Reader is called logical device number 6 on the I/O Define cards to be introduced with this program to the Loader, then the Card Reader must have logical device code 6 on the first line of coding of the Device Parameter area as follows: NAME DC A(6).

Device Correspondence Table

♦ At load time, the Device Correspondence table (DCT) is created by the Loader using information contained in the I/O Define cards. The DCT is the table in which actual devices are related to their device numbers. The DCT contains three bytes of information for each device. The Loader sets up places for 10 entries and fills them in as it reads I/O Define cards. The user may store additional entries provided that they are in the correct format. The starting address of the DCT is 152_{10}.

The magnetic tape version of the IOCS can handle as many devices as there are entries in the DCT. The nontape IOCS assumes that there is only one device per trunk (hence, six devices). Multiple device trunks can be used if the programmer specifies nonsimultaneous processing, or does not refer to a second device until the previous device is terminated. When the programmer ensures device servicing on multiple device trunks, the nontape IOCS can handle as many devices as there are designated in the DCT.

The format of the DCT is as follows:

Logical Device	00			01			09		
Memory Locations	152	153	154	155	156	157	179	180	181
Contents	A	B	C	A	B	C	A	B	C

Device Correspondence Table (Cont'd)

A, B, and C are bytes defined as follows:

Byte A		Byte B		Byte C
4 bits	4 bits	4 bits	4 bits	8 bits
TK. No.	DEV. No.	ALT. TK. No.	DEV. TYPE	CONTROL BYTE

Where: TK. No. is trunk number (range 0-5)

DEV. No. is logical device number (range 0-9). (This can have the range 0-F if the programmer inserts additional entries in the DCT).

ALT. TK. No. is the number of the alternate trunk by which the device can be accessed. This is not used by IOCS.

DEV. TYPE code has the following values and meanings:

0 - DXC Data exchange control (not supported by IOCS).

1 - Magnetic Tape.

2 - Card Reader or Videoscan Document Reader (Card Reader only).

3 - Card Punch.

4 - Paper Tape Reader.

5 - Paper Tape Punch.

6 - On-line Printer or Bill Feed Printer (continuous forms only).

7 - Input/Output Typewriter.

8 - Card Reader/Punch.

9 - Single channel communications device (not supported by IOCS).

Note: This device group will be expanded later to include the Videoscan Document Reader and the Bill Feed Printer.

CONTROL BYTE, byte C (see Table 7), is the information needed to prime a 7-channel magnetic tape device for the correct packing density and level. The hexadecimal codes for the control are defined in the section on I/O Define cards. Byte C is set to zeros by the IOCS for all devices other than 7-channel magnetic tapes.

Device Correspondence Table (Cont'd)

Table 7. Legitimate 7-Channel Control Byte Configuration*

Bits	Hexa-decimal	Density	Parity	Pack/ Unpack	Translator
11110000	F0	800	odd	on	off
10110000	B0	556	odd	on	off
01110000	70	200	odd	on	off
11101000	E8	800	odd	off	on
10101000	A8	556	odd	off	on
01101000	68	200	odd	off	on
11100000	E0	800	odd	off	off
10100000	A0	556	odd	off	off
01100000	60	200	odd	off	off
11001000	C8	800	even	off	on
10001000	88	556	even	off	on
01001000	48	200	even	off	on
11000000	C0	800	even	off	off
10000000	80	556	even	off	off
01000000	40	200	even	off	off

* This byte is not checked by the IOCS. It is the responsibility of the programmer to ensure that the setting of this byte is correct.

I/O Define Cards

♦ At program load time, a set of load cards must be supplied to the Loader program. One type of load card is the I/O Define card, which supplies DCT information to the loader. Each I/O Define card causes the Loader to set up one three-byte entry in the DCT. The information contained in the I/O Define cards is given in the following format:

Column	1	2	3	4 5	6	7 8	9	10	11 12	13 - 80
Content	V	0	L	x x	A	t u	a	d	h h	Ignored

Column — Content

1 - Must contain V. Identifies this card as a load card.

2 - Must contain 0. This identifies the load card as an I/O Define card.

3 - Must contain L, for logical device xx.

4-5 - The designation xx is the logical device number to which the actual device tu is assigned. The range of xx is 00 to 09.

6 - Must contain A, for actual device tu.

7-8 - The designation tu is actual device number. The range of trunk number, t, is 0 to 6, and its unit, u, is from 0 to F.

I/O Define Cards (Cont'd)

Column (Cont'd) — Content (Cont'd)

9 - The designation a is the alternate trunk (0-6) from which the device may also be accessed.

10 - The designation d is the device type being defined. Its range and meaning are as follows:

0 DXC (not supported by IOCS).

1 Magnetic Tape

2 Card Reader

3 Card Punch

4 Paper Tape Reader

5 Paper Tape Punch

6 Printer

7 Input/Output Typewriter

8 Card Reader/Punch

9 Single channel communications (not supported by IOCS).

11-12 - The designation hh is the hexadecimal representation of the control byte to be issued to a 7-channel Magnetic Tape Station to set its mode of processing. If 00 (or blanks) is specified, a 9-channel tape is assumed.

Designating I/O Devices

♦ Note that there are three numeric codes associated with each device. These are: xx, logical device number,
d, device type, and
tu, trunk and unit of referenced device, sometimes called actual device numbers.

The device type, d, is a code which stands for the description in words of the device. For example, 2 means Card Reader.

In the I/O Define card format, the logical device number, xx, is a programmer-supplied code name for the device referred to. In effect, it is the name of the device for the programs in which it is used.

The trunk and unit, tu, is the specific hardware designation of the device. In effect, it is a hardware address of the device. For an installation, the device type (d) connected to a particular trunk and unit (tu) is the same for all programs using that configuration. Only the logical device type (xx) may change, or, in other words, the programmer can call a device type on a trunk and unit by different names in different programs.

Designating I/O Devices (Cont'd)

Example 1: Same devices, different names. The programmer supplies a set of I/O Define cards to specify:

Logical Device	Trunk, Unit	Device Type
00	0,0	1 (Magnetic Tape)
01	0,1	1 (Magnetic Tape)
02	1,0	2 (Card Reader)
03	2,0	3 (Card Punch)
04	3,0	6 (Printer)

Then the programmer uses 02 to refer to the Card Reader in his program, 04 to refer to the Printer, etc.
If the I/O Define cards specified:

Logical Device	Trunk, Unit	Device Type
01	0,0	1 (Magnetic Tape)
00	0,1	1 (Magnetic Tape)
09	1,0	2 (Card Reader)
07	2,0	3 (Card Punch)
08	3,0	6 (Printer)

Then the programmer must use 09 to refer to the Card Reader, 08 to refer to the Printer, etc.

Example 2: Interchanging devices. If a program specifies the Card Punch as the output device and calls it logical device 05, then an I/O Define card must specify:

Logical Device	Trunk, Unit	Device Type
05	2,0	3 (Card Punch)

where the trunk and unit are taken to be as before. Then, if it is desired to have the output on magnetic tape instead of on cards, the program refers only to device number 05. All that is required to make this change is to replace the above described I/O Define card with an I/O Define card specifying the following:

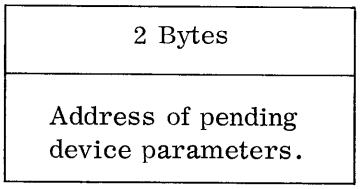
Logical Device	Trunk, Unit	Device Type
05	0,1	1 (Magnetic Tape)

where one of the Magnetic Tape Units described is now used as the output device.

The IOCS, acting only on the reference in the program to logical device number 05, goes to the Device Correspondence table (DCT) for the fifth entry, and using the information stored in the three-byte area for that device, performs the required I/O function.

Trunk Status Table

♦ A Trunk Status table is used to monitor all I/O functions that require servicing. The Trunk Status table is located within the boundaries of the IOCS and consists of one two-byte entry for each trunk. The format of this table is as follows:



This table is created and used by the IOCS along with the left-most byte of the simultaneity indicator in the Device Parameter area. The programmer need not be concerned with it.

GENERAL CONSIDERATIONS

Programming System Requirements

♦ Any of the standard 70/15 tape or card loaders can be used to load the IOCS and establish the Device Correspondence table. The RCA 70/15 Binder routine can be used to consolidate (bind) the IOCS with the user program. The RCA 70/15 Assembler assembles the user program. The IOCS can be included with the programmer's source program at assembly time, or it can be assembled separately.

Compatibility

♦ Programs that use the 70/15 IOCS can be run on a 70/25 Processor provided that the operating configurations are the same. However, the instructions in the 70/15 IOCS required for 70/25 compatibility use approximately 65 bytes of memory. If the programmer wants to save this area, he removes the set of cards containing these instructions from the supplied IOCS source deck. These cards are clearly identified for easy removal. All 70/15 IOCS comments start in the same column of the cards. The compatibility cards are identified in this comments field.

Maintenance

♦ The IOCS is to be maintained by the programmer in his program library using to 70/15 File Update routine. For updating purposes, the name of the IOCS is "IOCS".

All tags in the IOCS are four-character symbols beginning with the letters IO except for the five entry (ENTRY) points. The programmer should avoid using such tags to prevent duplicate tags at assembly time if he includes the source IOCS in his source deck.

Accuracy Control

♦ Whenever an inoperable condition code setting is detected following the attempted execution of any I/O instruction (Read Forward, Read Reverse, Read Auxiliary, Write, Write Control, Write Auxiliary) the IOCS:

1. Moves the actual trunk and unit number to the fixed location \$P+5 = 0087_{16} where it can be displayed by the operator.

Accuracy Control (Cont'd)

2. Halts. When displayed, Halt $8F_{16}$ is indicated.

3. Re-executes the operation when the operator presses the START button.

An invalid I/O operation can be generated if the programmer specifies a D_1 address greater than the D_2 address. In this event, the IOCS issues a Read Reverse instruction.

Detection of parity errors in the reading of magnetic tape causes the IOCS to re-read the block. If these re-reads are unsuccessful 10 times, the condition is considered to be a nonrecoverable read parity error and control is transferred to the alarm address.

When a read-after-write error is detected, the IOCS backspaces the tape one block, erases 2400 bytes (three inches of tape) and then rewrites the message. The procedure is repeated and if the error persists after 10 attempts, the error is considered to be a nonrecoverable read-after-write error, and control is transferred to the user's alarm address.

When the IOCS transfers control to the user's alarm address after 10 consecutive read or write attempts on magnetic tape, the programmer is given the option of attempting the read or write again.

Any time the IOCS transfers control to either the alarm address or the abnormal address, the user's normal return address and the address of the device parameters causing the transfer are undisturbed in $P.

PROBLEM AREA PROGRAM SUGGESTIONS

Introduction

To aid the user programmer in the use of IOCS in the Simultaneous mode, three examples of suggested solutions to input/output problems are given. These are:

1. Simultaneous Input Functions. An explanation of an efficient method of performing the IN (input) function in the Simultaneous mode.

2. Simultaneous Output Functions. An explanation of an efficient method of performing the OUT (output) function in the Simultaneous mode.

3. Error Recovery on the 70/236 Card Punch. An explanation of the required method for error recovery on this device to maintain a continuous flow of processing.

Introduction (Cont'd)

The first two examples include a flow chart, a narrative, and the coding of the problem in 70/15 Assembly language.

Note: The coding is not part of the IOCS, but is to be a part of the user program. Only the coding necessary to solve the general problem is presented. Places are indicated where programmer's coding to solve the programmer's specific problem can be inserted.

In the examples that follow, the identifying letters and numbers in the narratives refer to corresponding notation on the flow charts.

Simultaneous Input Functions

♦ The following is a recommended method and not a mandatory procedure. Because the 70/15 does not have base address registers or index registers, it is much easier to read data into one input area and move the data to a record area for processing rather than to use alternating input areas. However, specialized routines can make alternate areas more feasible.

Narrative Description (See Figure 2 and Chart 24)

Block A - Represents the entrance from regular program housekeeping to the steps necessary to initialize the input.

Block 10 - This function initiates the first read and any re-reads caused by errors. Use of this method eliminates changing the simultaneity indicator for first reads and re-reads.

Block 20 - This check (CHK) function terminates the first read and succeeding reads. This is the only point to which the IOCS returns other than the normal return.

Block 30 - This end-of-input test is necessary at this point to prevent initiating another read to a device that may be inoperable because of an empty hopper, etc. This function can be bypassed on magnetic tape by use of the tape mark which would be an abnormal return from the check function.

Block 40 - This move to a work area frees the input area to be used to read the next record.

Block 50 - This IOCS function initiates the next read into the input area. This reading is overlapped by the processing below.

Block 60 - This represents the programmer's coding required to process the data in the work area. This can include computing, output functions, or another nonsimultaneous input function. The 70/15 Processor will not accept another simultaneous input function until the current one is processed.

When the processing is finished, the programs should return to the CHK function at B.

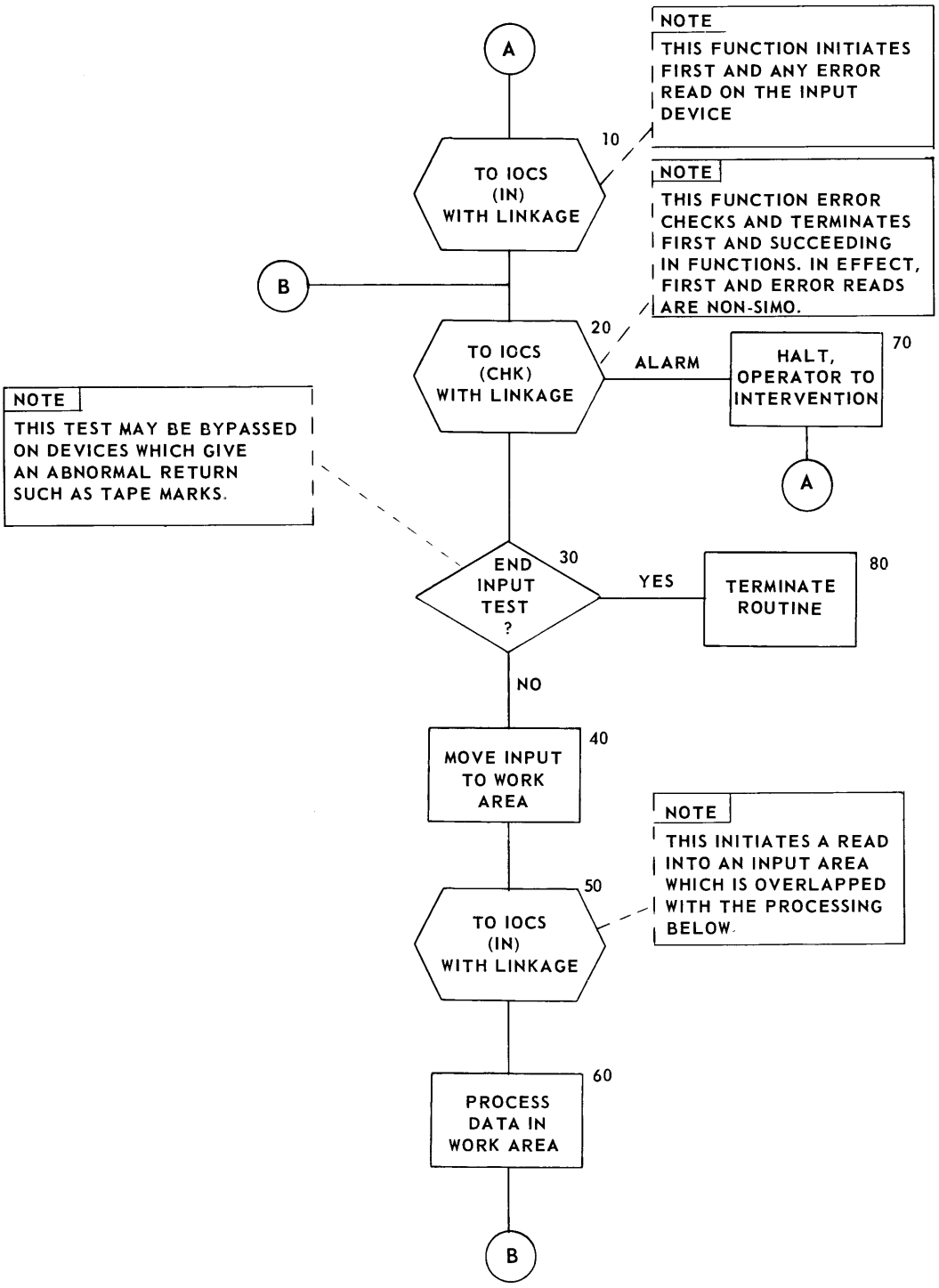
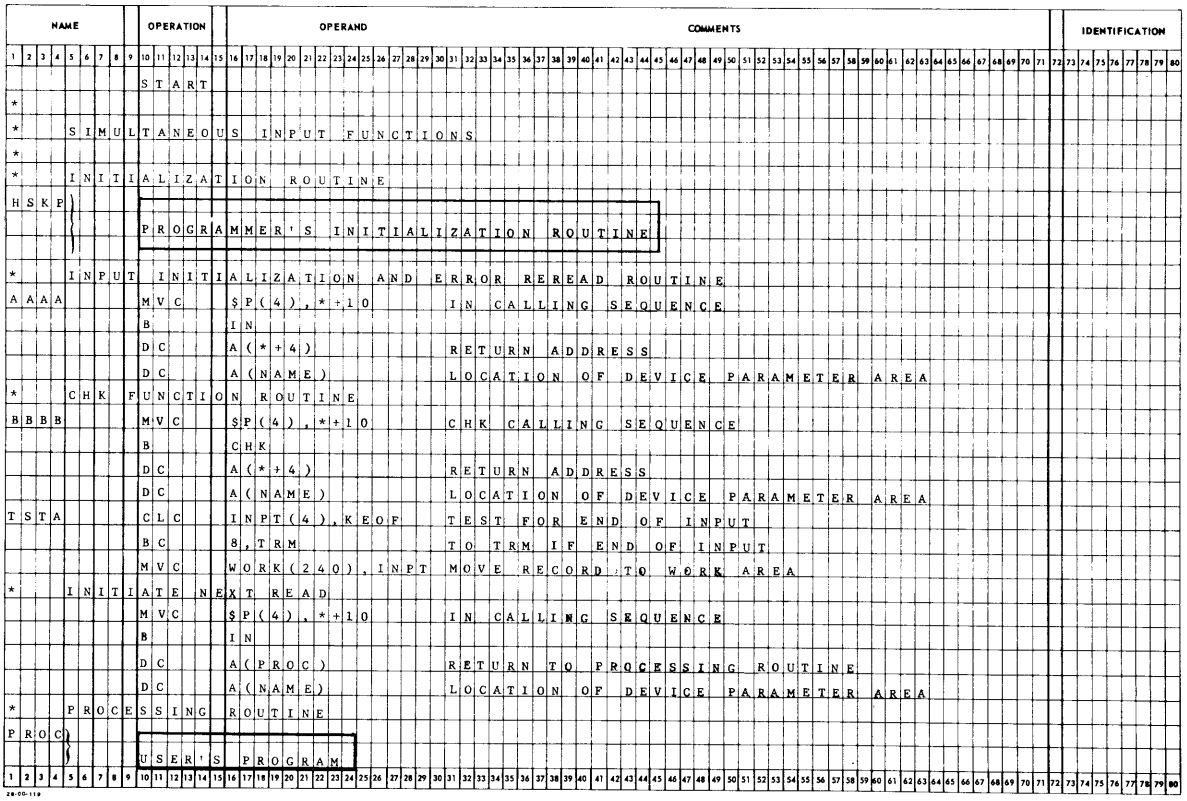


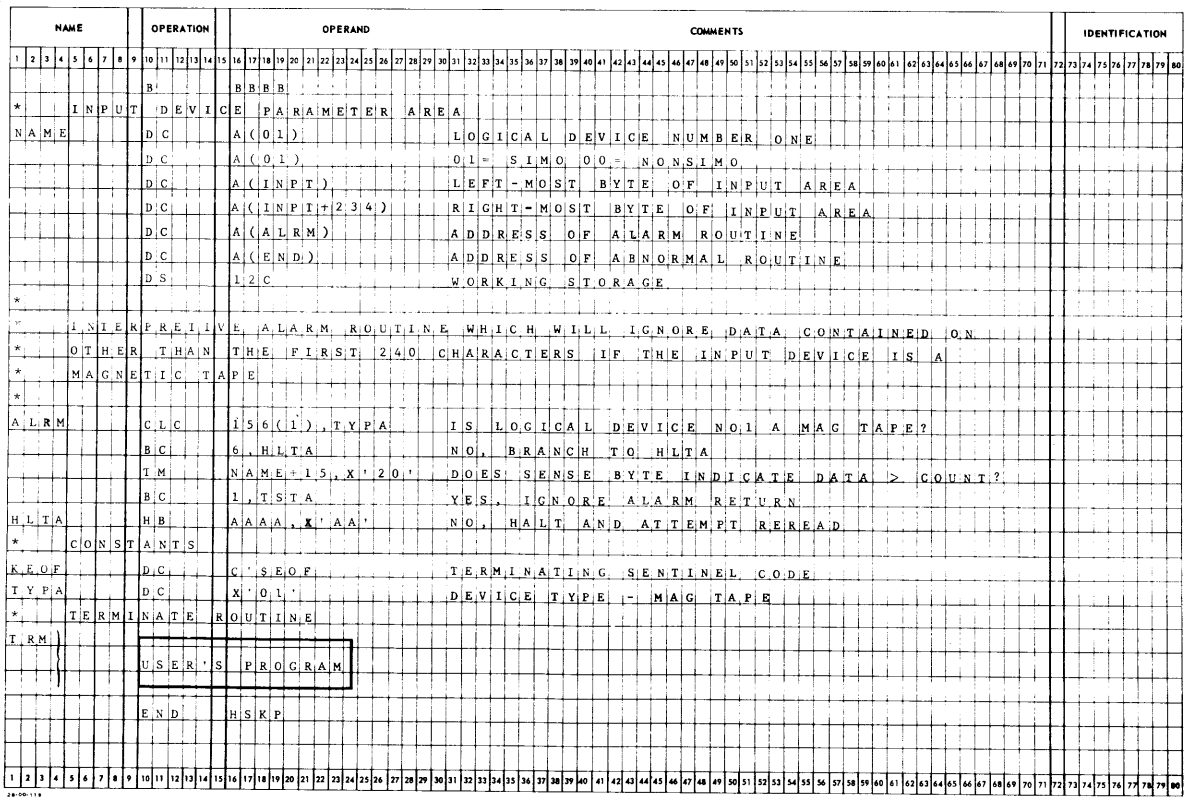
Figure 2. Simultaneous Input Functions, Flow Chart

Chart 24. Simultaneous Input Functions, Sheet 1 of 2



```
NAME      OPERATION  OPERAND                  COMMENTS                                          IDENTIFICATION
          START
*
*         SIMULTANEOUS INPUT FUNCTIONS
*
*         INITIALIZATION ROUTINE
HSKP
          PROGRAMMER'S INITIALIZATION ROUTINE
*
*         INPUT INITIALIZATION AND ERROR REREAD ROUTINE
AAAA      MVC       $P(4),*+10             IN CALLING SEQUENCE
          B         IN
          DC        A(*+4)                 RETURN ADDRESS
          DC        A(NAME)                LOCATION OF DEVICE PARAMETER AREA
*         CHK FUNCTION ROUTINE
BBBB      MVC       $P(4),*+10             CHK CALLING SEQUENCE
          B         CHK
          DC        A(*+4)                 RETURN ADDRESS
          DC        A(NAME)                LOCATION OF DEVICE PARAMETER AREA
TSTA      CLC       INPT(4),KEOF           TEST FOR END OF INPUT
          BC        8,TRM                  TO TRM IF END OF INPUT
          MVC       WORK(240),INPT         MOVE RECORD TO WORK AREA
*         INITIATE NEXT READ
          MVC       $P(4),*+10             IN CALLING SEQUENCE
          B         IN
          DC        A(PROC)                RETURN TO PROCESSING ROUTINE
          DC        A(NAME)                LOCATION OF DEVICE PARAMETER AREA
*         PROCESSING ROUTINE
PROC
          USER'S PROGRAM
```

Chart 24. Simultaneous Input Functions, Sheet 2 of 2



```
NAME      OPERATION  OPERAND                  COMMENTS                                          IDENTIFICATION
          B         BBBB
*         INPUT DEVICE PARAMETER AREA
NAME      DC        A(01)                  LOGICAL DEVICE NUMBER ONE
          DC        A(01)                  01= SIMO 00= NONSIMO
          DC        A(INPT)                LEFT-MOST BYTE OF INPUT AREA
          DC        A(INPT+234)            RIGHT-MOST BYTE OF INPUT AREA
          DC        A(ALRM)                ADDRESS OF ALARM ROUTINE
          DC        A(END)                 ADDRESS OF ABNORMAL ROUTINE
          DS        12C                    WORKING STORAGE
*
*         INEFFECTIVE ALARM ROUTINE WHICH WILL IGNORE DATA CONTAINED ON
*         OTHER THAN THE FIRST 240 CHARACTERS IF THE INPUT DEVICE IS A
*         MAGNETIC TAPE
*
ALRM      CLC       156(1),TYPA            IS LOGICAL DEVICE NO1 A MAG TAPE?
          BC        6,HLTA                 NO, BRANCH TO HLTA
          TM        NAME+15,X'20'          DOES SENSE BYTE INDICATE DATA > COUNT?
          BC        1,TSTA                 YES, IGNORE ALARM RETURN
HLTA      HB        AAAA,X'AA'             NO, HALT AND ATTEMPT REREAD
*         CONSTANTS
KEOF      DC        C'SEOF'                TERMINATING SENTINEL CODE
TYPA      DC        X'01'                  DEVICE TYPE - MAG TAPE
*         TERMINATE ROUTINE
TRM
          USER'S PROGRAM
          END       HSKP
```

Simultaneous Output Functions

♦ The following is the recommended method of using the IOCS to control output to a buffered device. The use of simultaneity to other than a buffered device does not have any advantages, because the IOCS does not use the Write Auxiliary instructions.

Narrative Description (See Figure 3 and Chart 25)

Block 10 - Represents processing. This includes receiving input data and processing it before preparing it in a proper format for output.

Block 20 - Error checks and terminates the output function. On the first entry, it is a dummy, but it does not cause erroneous functions to be performed. It is at this point that the IOCS advises the using program of such items as low paper, errors, and other processor signals.

This function can be bypassed if the programmer desires the next output function to terminate the previous output function. The programmer should be aware of several problems. One is that on an alarm or abnormal return, the IOCS has not been issued the output function and it is necessary to re-enter the routine to do this. The second problem is that if the programmer wishes to substitute a magnetic tape for a buffered device, the IOCS performs Rollback on the new data while destroying the old.

Block 30 - Represents the making of a new output record.

Block 40 - Initiates all output functions to the device. If step 20 is eliminated, step 40 also error checks the previous function to this device. This I/O function is overlapped by processing.

Block 50 - Represents abnormal returns from the IOCS for items such as low paper or sensing of a 9- or 12-punch in the paper tape loop.

Block 60 - Represents alarm conditions such as PCE and may be an interpretive routine which may, for example, permit the printing of nonprintable characters and be a true error condition on others.

Block 70 - Represents a step from normal processing to a terminate routine. This leads to the final output functions.

Block 80 - Represents the final output processing and any output functions.

Block 90 - Represents the final check to the IOCS as this was not done in the above processing.

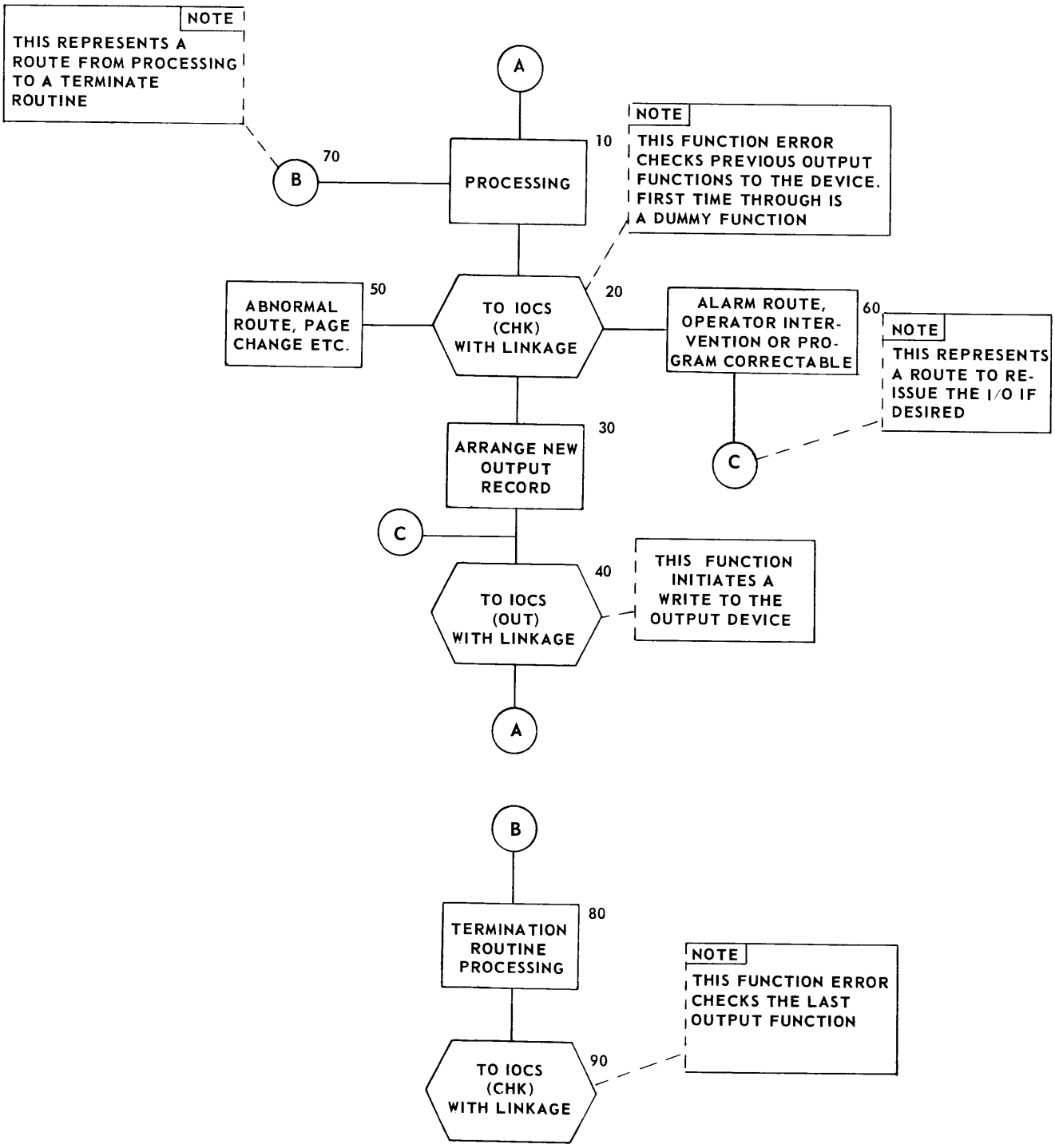
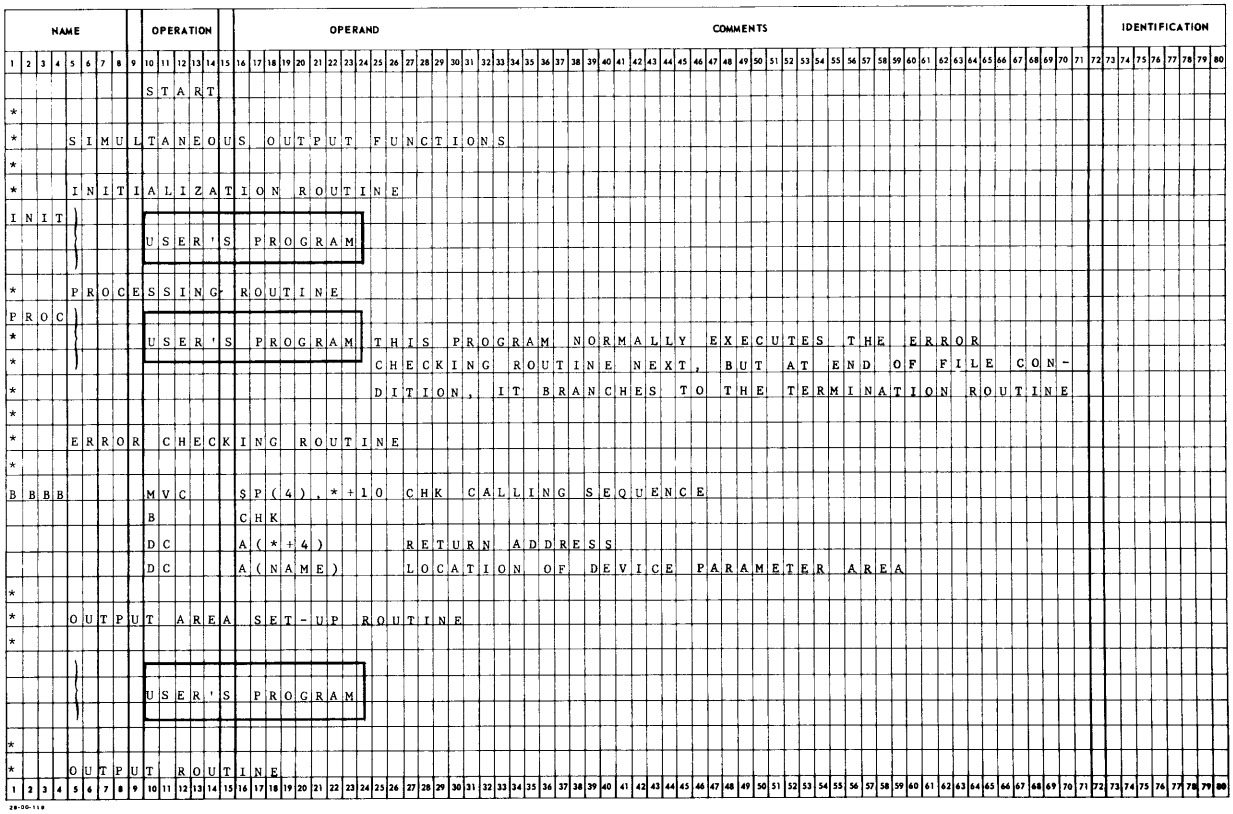


Figure 3. Simultaneous Output Functions, Flow Chart

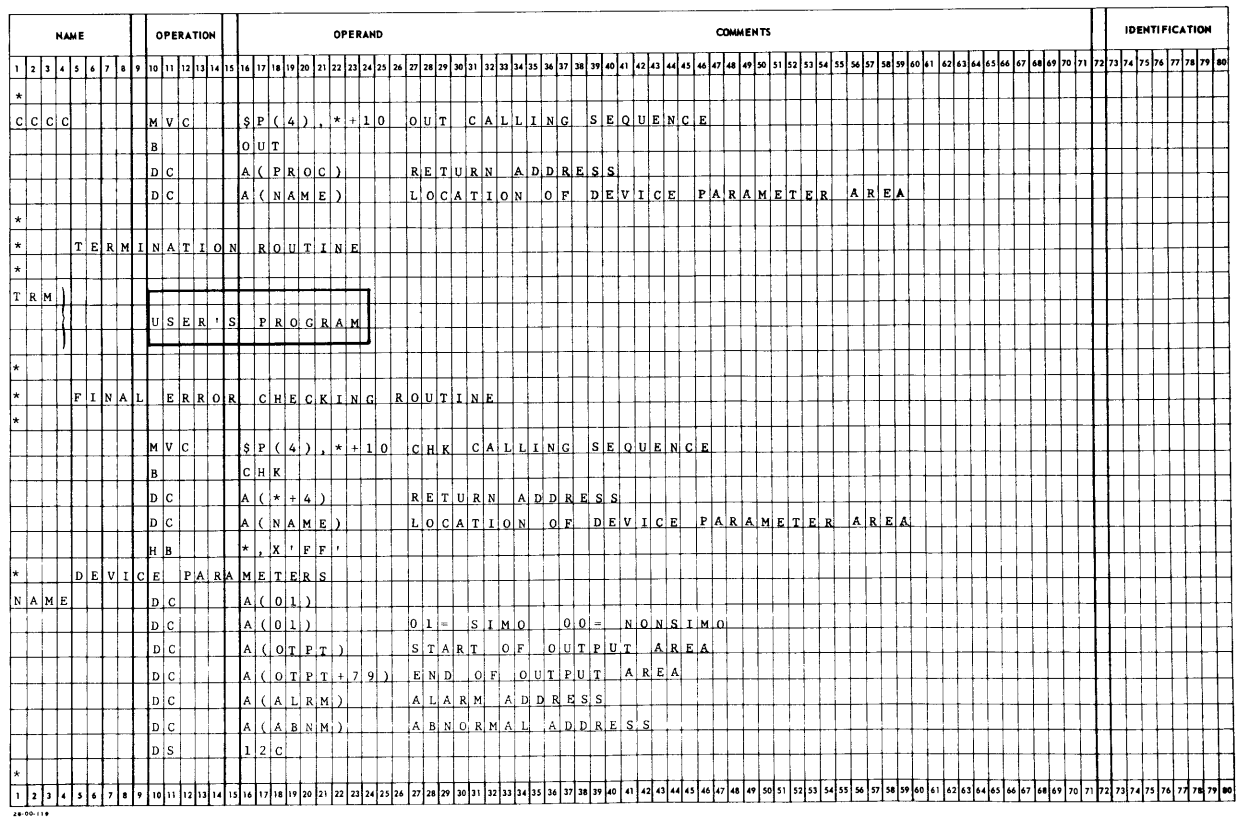
Chart 25. Simultaneous Output Functions, Sheet 1 of 3



```
NAME      OPERATION  OPERAND                COMMENTS                                                    IDENTIFICATION
          START
*
*         SIMULTANEOUS OUTPUT FUNCTIONS
*
*         INITIALIZATION ROUTINE
INIT      
          USER'S PROGRAM
*
*         PROCESSING ROUTINE
PROC      
*         USER'S PROGRAM     THIS PROGRAM NORMALLY EXECUTES THE ERROR
*                            CHECKING ROUTINE NEXT, BUT AT END OF FILE CON-
*                            DITION, IT BRANCHES TO THE TERMINATION ROUTINE
*
*         ERROR CHECKING ROUTINE
*
BBBB      MVC        SP(4),*+10  CHK CALLING SEQUENCE
          B          CHK
          DC         A(*+4)      RETURN ADDRESS
          DC         A(NAME)     LOCATION OF DEVICE PARAMETER AREA
*
*         OUTPUT AREA SET-UP ROUTINE
*
          USER'S PROGRAM
*
*         OUTPUT ROUTINE
```

28-00-11 0

Chart 25. Simultaneous Output Functions, Sheet , 2 of 3



```
NAME      OPERATION  OPERAND                COMMENTS                                                    IDENTIFICATION
*
CCCC      MVC        SP(4),*+10  OUT CALLING SEQUENCE
          B          OUT
          DC         A(PROC)     RETURN ADDRESS
          DC         A(NAME)     LOCATION OF DEVICE PARAMETER AREA
*
*         TERMINATION ROUTINE
*
TRM       
          USER'S PROGRAM
*
*         FINAL ERROR CHECKING ROUTINE
*
          MVC        SP(4),*+10  CHK CALLING SEQUENCE
          B          CHK
          DC         A(*+4)      RETURN ADDRESS
          DC         A(NAME)     LOCATION OF DEVICE PARAMETER AREA
          HB         *,X'FF'
*         DEVICE PARAMETERS
NAME      DC         A(01)
          DC         A(01)       01= SIMO  00= NONSIMO
          DC         A(OTPT)     START OF OUTPUT AREA
          DC         A(OTPT+79)  END OF OUTPUT AREA
          DC         A(ALRM)     ALARM ADDRESS
          DC         A(ABNM)     ABNORMAL ADDRESS
          DS         12C
*
```

28-00-11 0

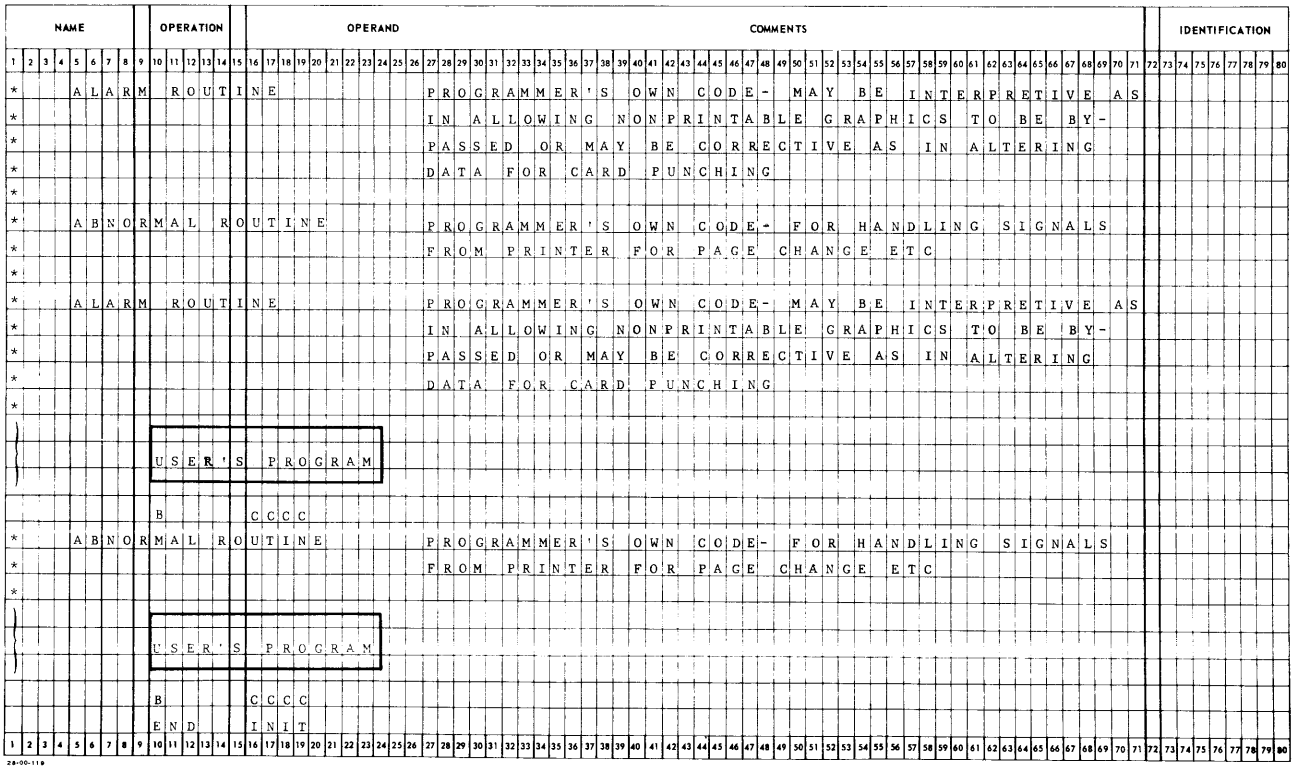
Error Recovery on the 70/236 Card Punch

Figure 4 reveals how error recovery must be attempted on the 70/236 Card Punch to maintain continuous flow processing. No sample coding or narrative description matching the flow chart is given for this example.

Triple buffering allows error recovery from every type of punch error. Error recovery from a punch card error requires repunching the card in error plus the one just punched, because both are routed to the reject stacker. This hardware scheme requires the retaining of the card image after it has been punched. Recovery from parity errors generated through transmission, or in the punch buffer, are recovered from by repunching the area just punched. If there are parity errors in the buffer, the card in error is routed to the reject stacker. If there is a parity error in transmission, the punch transport does not advance.

This routine provides for a recovery of almost any error without operator intervention.

Chart 25. Simultaneous Output Functions, Sheet 3 of 3



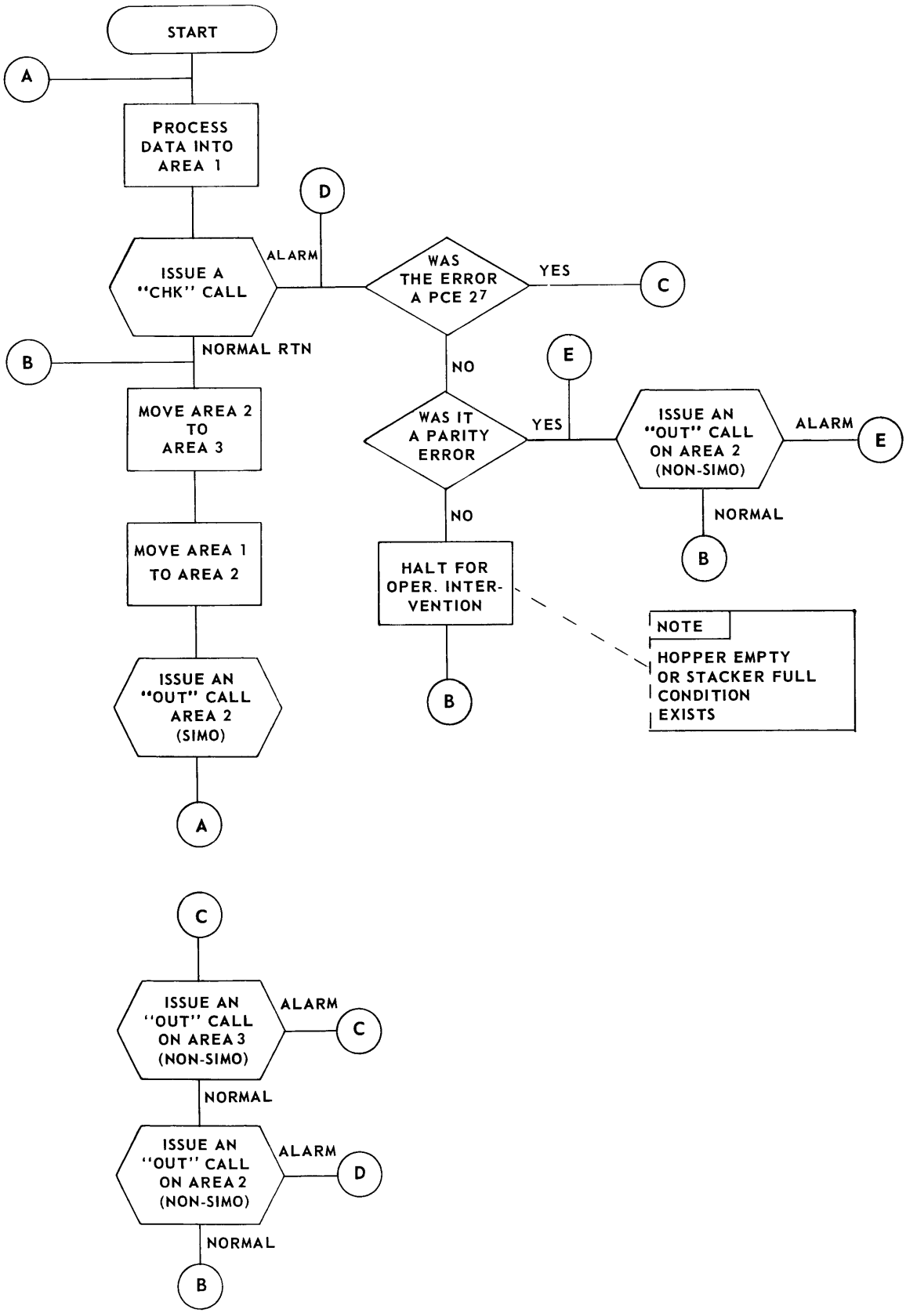


Figure 4. 70/236 Card Punch Error Recovery Procedure, Flow Chart

ASSEMBLER PROGRAM

ASSEMBLER PROCESSING

♦ The 70/15 Assembly System is in the form of a classical two-pass Assembler, but there is no intermediate punched output. The Assembler consists of two segments. Each segment processes one pass.

First Pass

♦ In the first pass, a table of name-address assignments is created. If a magnetic tape is selected as the input device for the second pass and the card reader is used as the first pass input, the cards read will be written to the magnetic tape, which will then be read in the second pass without operation or intervention.

If a card reader is selected as the input device for the second-pass, the operator must restack the reader hopper with the source program deck. A Halt and Branch at the termination of pass one will give the operator time to compose the second-pass input.

Second Pass

♦ During the second pass the operands and operation codes are defined, the object program deck is generated, and the assembly listing is printed.

DEVICE ASSIGNMENT/ INTERCHANGEABILITY

♦ The input/output commands within the 70/15 Assembly System do not assume the use of a standard set of peripheral device trunk and unit numbers. Instead, they are coded with logical device numbers and are linked to actual devices only at execution time. A standard function of the Loader performs these linkage assignments.

A set of I/O Define cards (see "Formats", page 53), one for each logical device required, when passed through the Loader sets up a Device Correspondence table which serves the I/O needs of the Assembler. These device assignments must be made before the Loader transfers control to the Assembler. Corresponding to these logical numbers the following devices must be defined:

00 - Input Device for Pass 1

01 - Input Device for Pass 2

07 - Assembly System Device*

08 - Object Program Output Device

09 - Assembly Listing Device

Since the logical device numbers do not require that a particular actual device (e.g., card reader) be specified, magnetic tape may be designated as the actual device for any of the preceding I/O Define cards.

*Tape Assembly.

GENERATED OBJECT PROGRAM

♦ The object program generated by the 70/15 Assembly System consists of five types of cards. Their functional descriptions are listed below. (See "Formats", page 53.)

Program Card

♦ This card is the first card of every program. It contains the name of the first source statement. Its format is similar to that of a file name and, therefore, can serve the program as a file label on tape libraries. It also contains the address to which the program is assembled. This card corresponds to the source START statement.

Text Card

♦ This card contains the generated object code. The text card holds approximately 10 machine instructions along with other loader control information such as relocation factors, memory assignment, and text card identification. The text card may contain partial instructions, and the instructions will be continued from one text card to the next.

Entry Card

♦ This card corresponds to the source ENTRY statement. It contains the name and address of a program entry point. One object program Entry card is generated for each source ENTRY statement.

Extrn Card

♦ This card corresponds to the source EXTRN statement. It contains the symbolic name and address of the last program reference to the external name. One object program Extrn card is generated for each external name.

End Card

♦ This card corresponds to the source END statement. It contains the address of the first logical instruction in the program that is to be executed.

ASSEMBLY LISTING

♦ The assembly listing contains on each line the image of the source language statement, the machine code that resulted from it, and the locations assigned to each statement. In addition, the listing displays page headers and any necessary error flags. The source language statement composed almost exclusively of EBCDIC graphics is printed without editing. Any EBCDIC nongraphic in the statement card will be represented as a blank. However, since each byte of machine code is probably not an EBCDIC graphic, the resultant machine code related to each source statement is represented on the listing in its hexadecimal form (two characters 0-F to a byte). The address assigned to each statement, for the same reason, is also represented in hexadecimal form.

The Identification field (columns 73-80) of each object program card is also displayed on the listing. Since one machine code Text card can be generated from many sequential source statements, the Text Card Identification will be printed only with that statement that contributes the least significant byte of text, and indeed this byte will correspond to the address assigned to the statement. Because there is a one-to-one correspondence in a source statement and the generated object program card for the Start (Program), Entry (Entry), Extrn (Extrn) or End (End) card, they will always be printed with an object program card identification. DS's generate no output, and, therefore, never display an Object Program Identification.

ASSEMBLY LISTING (Cont'd)

A diagram of a page of assembly listing is shown below.

Program Name	70/15 Assembly Listing	Date	Page #
1-4	7-10	13-27	33-112	117-120
ef	loc	code	statement	id

Program Name: The four-character name from the program's Start card.

Date: The contents of the loader date area with slashes inserted; i.e., mm/dd/yy.

Page # Page count.

ef: Print area reserved for possible error flags.

Up to four simultaneous error indicators can be displayed, one character for each error condition in the statement. The list of error flag characters and their meaning follows:

Error Flag	*Meaning*
N	Invalid Name
O	Invalid Operation
L	Invalid mask, length, trunk or unit.
I	Invalid Operand
P	Punch Error During Assembly
U	Undefined Symbol
M	Multiple Defined Symbol
H	Location Counter Overflow
S	Statement Out of Order
T	Name Table Overflow
D	Invalid Decimal Field
X	Invalid Hexadecimal Constant
R	Read Error During Assembly

loc: Hexadecimal representation of the address assigned to the first byte of the statement.

code: Hexadecimal representation of the coding generated by the statement. If more than six bytes are generated by a statement, a new print line is used for each additional six bytes or remainder.

statement: Image of the 80-column source statement.

id: Four-character Identification field on the object program card associated with the statement.

RESTRICTIONS

- The 70/15 Assembly Program will not accept source language programs on magnetic tape for processing that have been blocked. If the input source language program is on magnetic tape, it must be in 80-byte record blocks.

COMPATIBILITY

- The 70/15 Assembly System can be executed on the 70/25 Processor without any modification. A standard 70/15 loader with the compatibility card added must be used to load the program into the 70/25.

OPERATING PROCEDURES

♦ In order to assemble a program using the 70/15 Assembly System, the operating procedures listed below must be followed.

CARD SYSTEMS

♦ 1. Compose the deck illustrated in Figure 5. If the proper loader is already in memory, start the deck with the four I/O Define cards.

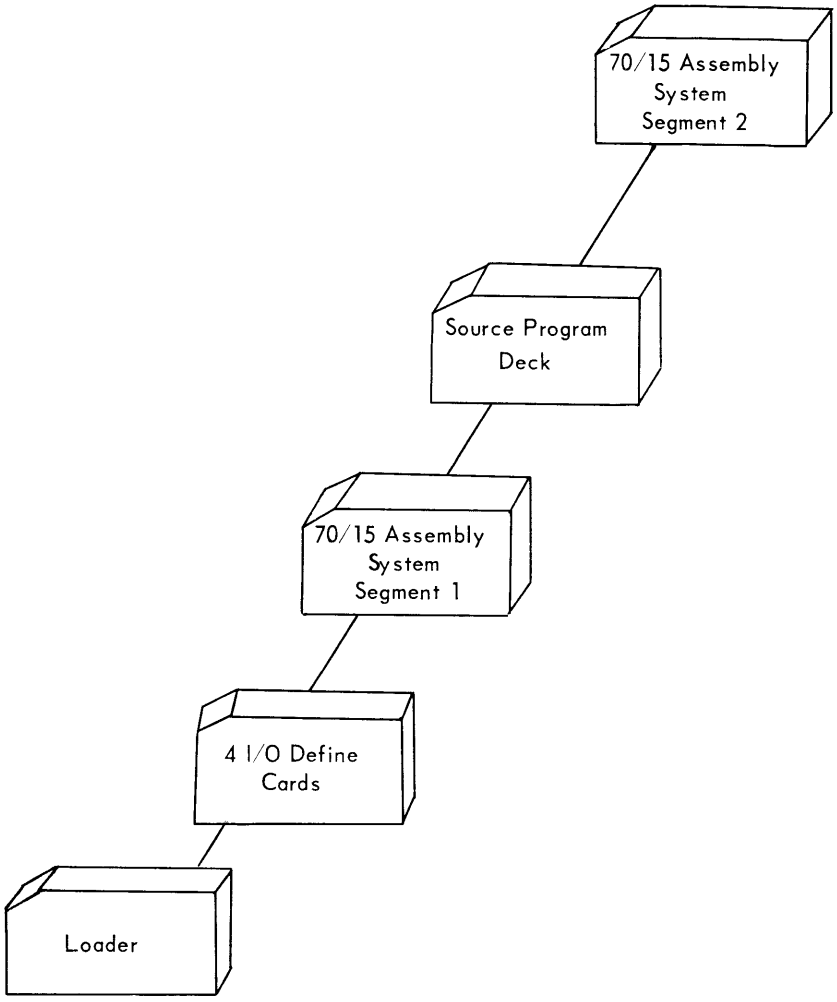


Figure 5. Composed Deck for Card Systems

2. Place the composed deck in the card reader.

3. If the loader is already in memory, this step is omitted. Insert the trunk and unit number of the card reader into the M register and depress the Load button to bring the loader into memory. The processor will halt with an $(FF)_{16}$ in the M register.

4. Depress the START button and the assembler will be loaded and the first pass will be executed. When the first pass is completed, the processor will halt with an $(F1)_{16}$ in the M register.

CARD SYSTEMS (Cont'd)

5. The card reader input stacker now has segment 2 of the assembler ready for loading. Replace the source deck behind this segment and depress the START button.

6. When pass two is completed, the processor will halt with an $(FF)_{16}$ in the M register.

TAPE LIBRARY SYSTEMS

♦ 1. If the source program is on cards, compose the deck that is illustrated in Figure 6. If the source program is on magnetic tape, the four I/O Define cards and the assembler Call card are the only required cards.

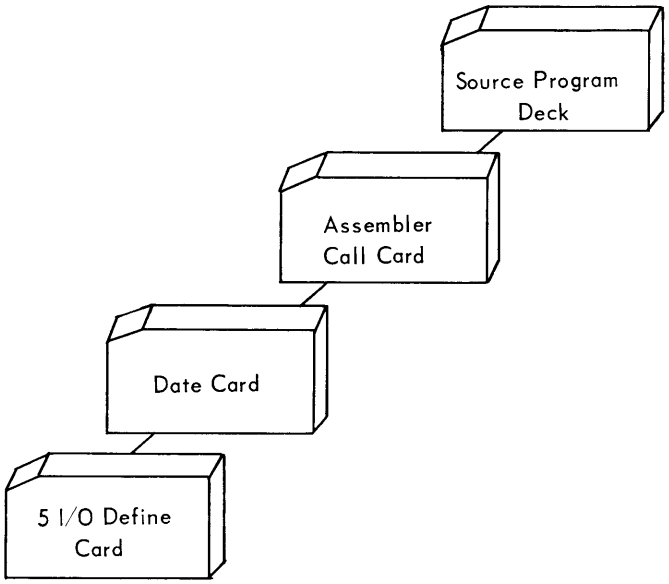


Figure 6. Composed Deck for Tape Library Systems (Card Output).

2. Place the composed deck in the card reader.

3. If the Loader is already in memory, step 4 is omitted.

4. If the Loader is not in memory, set the trunk and unit number of the Program Library Tape (PLT) into the M register and depress the Load button. When the Loader has been read in, the processor will halt with an $(FF)_{16}$ in the M Register.

5. Depress the START button. The four I/O define cards will be loaded and the Assembler call card will cause the Loader to scan the PLT for the Assembly Program and to subsequently load and execute the first pass.

6. If the source program is on cards and the Card Reader has been designated as the input device for the second pass, the Processor will halt $(F1)_{16}$ at the completion of the first pass. Restack the source program in the Card Reader and depress the START button to execute the second pass.

TAPE LIBRARY SYSTEMS (Cont'd)

7. If the source program is on magnetic tape, or if magnetic tape has been designated as the second pass input device, there is no halt between the first and second pass.

8. If the object program is to be stored on tape in PLT format, the PLT Absolute Loader should be inserted between the Assembler call card and the source program deck. The Assembler will write the loader onto the front of the object program tape.

9. Final halt $(FF)_{16}$, when assembly has been completed.

STACKED ASSEMBLIES

♦ Assemblies may be stacked only if the source program is on magnetic tape or the second pass input device is magnetic tape. In order to permit stacked assemblies in the PLT system, the output tapes will not be rewound at the end of assembly. Rewinding can be effected by Execute cards placed after the last source deck.

PROGRAM HALTS

♦ A program halt takes place whenever a nonrecoverable error condition occurs. A list of halt display characters (contents of M register on console), their meaning and suggested operator action will be provided at a later date. It should be noted that source language errors will not cause error halts. All assemblies must be rerun from start if any nonrecoverable errors occur during assembly processing. Nonrecoverable I/O errors will cause the processor to halt with an $(8F)_{16}$ displayed in the M register. The trunk and device number which caused the error are stored in the standard location $P (Byte 130).

FORMATS

I/O DEFINE CARD (Input)

♦ I/O Define cards must be passed through the Loader prior to execution of the Assembler in order to define the peripheral devices required by the Assembler. One card is required for each logical device referenced by the Assembler. The format of the I/O Define card is as follows:

Card Column	1	2	3	4	5	6	7	8	9	10	11	12	13-80
Legend	V	0	L	x	x	A	t	u	d	d	h	h	Ignored

Note: All upper case letters must appear on the I/O Define card in their respective columns.

where: V0 = identification of I/O Define Card.
L = indicates that logical device number follows.
xx = the logical device number to which the actual device is assigned. The range of xx is 00 to 09.
A = indicates that the actual trunk and device numbers follow.
t = actual devices' trunk and may range from 0 to 6.
u = actual device unit and may range from 0 to F.
dd = the device type being defined; i.e.,

01 = Magnetic Tape
02 = Card Reader
03 = Card Punch
04 = Paper Tape Reader
05 = Paper Tape Punch
06 = On-Line Printer
07 = Interrogating Typewriter
08 = Card Reader-Punch
09 = Single Channel Communications

hh = the hexadecimal representation of the control byte to be issued to a 7-channel magnetic tape station to set its mode of processing. Table 8 shows the possible hexadecimal representations and meanings of the control bytes.

I/O DEFINE CARD (Cont'd)

Table 8. Control Bytes

Binary	Meaning	Hexadecimal
11110000	800, odd, pack/unpack on, translator off	FØ
10110000	556, odd, pack/unpack on, translator off	BØ
01110000	200, odd, pack/unpack on, translator off	7Ø
11101000	800, odd, pack/unpack off, translator on	E8
10101000	556, odd, pack/unpack off, translator on	A8
01101000	200, odd, pack/unpack off, translator on	68
11100000	800, odd, pack/unpack off, translator off	EØ
10100000	556, odd, pack/unpack off, translator off	AØ
01100000	200, odd, pack/unpack off, translator off	6Ø
11001000	800, even, pack/unpack off, translator on	C8
10001000	556, even, pack/unpack off, translator on	88
01001000	200, even, pack/unpack off, translator on	48
11000000	800, even, pack/unpack off, translator off	CØ
10000000	556, even, pack/unpack off, translator off	8Ø
01000000	200, even, pack/unpack off, translator off	4Ø

1. If a control byte contains Ø's in bit positions 2^7 and 2^6, the density setting will not be changed from its present setting.

2. If a control byte contains 1's in bit positions 2^4 and 2^3, the Tape Controller will terminate the command. The 2^2 position of the standard device byte will be set to 1, and the 2^0 position of the first secondary status byte will be set to 1. This is an illegal operation. The Translator and Pack/Unpack modes may never be on at the same time.

3. If a control byte contains a 1 in position 2^4 and a Ø in position 2^5, the Tape Controller will force odd parity. If a Read command is attempted, and the tape was written in even parity, parity error will be indicated. Odd parity must be observed for pack/unpack.

If 00 (or blanks) are specified, a 9-channel tape station is assumed.

OBJECT PROGRAM CARDS (Output)

♦ The object program is composed of machine language coding and Loader parameter cards. Five different object program formats are processed by the loader. Columns 1 and 2 of all five cards indicate card type. They are described on the following page.

Program Card

Card Column	1	2	3	4	5	6	7	8	9	10	11-21	22-80
Legend	V	3	n	a	m	e	a	a	b	b	date	Ignored

where: n a m e = the program name.
aa = the EBCDIC address to which the program is assembled.
bb = the EBCDIC address of the highest byte +1 of this program.
date = the contents of the date area at assembly time.

Entry Card

Card Column	1	2	3	4	5	6	7	8	9-80
Legend	V	4	n	a	m	e	a	a	Ignored

where: n a m e = the name of the Entry point being defined.
aa = the EBCDIC address where it is assembled in the program.

Text Card

Card Column	1	2	3	4	5	6	7	8	9	10	11-72	73-80
Legend	V	5	f	f	f	f	n	a	a		Text	Ignored

where: ffff = four EBCDIC characters corresponding to 32-float bit factors to be applied to the text. Each two bytes of text correspond to a float bit. If the bit is one, the float factor is added to the two bytes before transferring them to memory. The most significant bit corresponds to the most significant pair of bytes to be assembled to an even memory location. If the first byte of text is assembled to an odd location, it is not floated.
n = the EBCDIC number of bytes of text to be transferred (1-62).
aa = the address to where the first byte of text is assembled.
Text = Generated EBCDIC Object code.

External Card

Card Column	1	2	3	4	5	6	7	8	9-80
Legend	V	7	n	a	m	e	a	a	Ignored

where: n a m e = the name of the external reference.
aa = the EBCDIC address of the assembled location of the last reference to external name in the program.

End Card

Card Column	1	2	3	4	5	6	7	8	9-80
Legend	V	9	n	a	m	e	a	a	Ignored

where: n a m e = the name of entry point to which the Loader is to branch.

aa = the EBCDIC address of the assembled location to which the Loader is to branch. A blank in this field directs the Loader to issue a read of the parameter source.

The following card is used to call an object program from a library tape:

LOADER CALL CARD

Card Column	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15-80
Legend	V	1	n	a	m	e		h	h	h	h	i		b	Ignored

where: n a m e = is the name of the program to be loaded. The Assembler call name is AST1.

hhhh = is the hexadecimal address to where the program is to be loaded (031C).

i = when nonblank signifies that the loader should not erase the Entry-Extrn table but continue to make use of it. A blank implies erasure and generation of a new table.

b = if nonblank it denotes PLT batched five cards/block.

System 70/15 Processor **Date** December 22, 1965 **No.** 1

70/15 ASSEMBLY REFERENCE MANUAL #70-15-602
First Printing April, 1965
Revised October, 1965

General

This bulletin contains revised information to the 70/15 Assembly System Reference Manual. These changes and additions advise the reader of the latest information available concerning the 70/15 Assembly System Reference Manual. This information will be included in the next printing of the manual. Specifically, the revised information is itemized below.

Page	Para.	Line	
28			Table 4. First line. Add "(7- or 9-channel)" after the words "Magnetic Tape Unit".
30	3	6, 7	Delete the sentence, "Also, all ENTRY and EXTRN... before assembly.".
40			Table 5. Model number of Bill Feed Printer should be 70/248.
44			Table 6.

Card Reader: delete model number 70/232.

Printer: change condition represented by bit position 2^7 from "Print Error" to "Channel 12 sensed."

Change condition represented by bit position 2^6 from "Not Used" to "Channel 9 Sensed."

Conditions represented by bit positions 2^7 and 2^6 are Abnormal conditions.

Input/Output Typewriter: Model number should be 70/216.

Magnetic Tape Devices:

Condition represented by bit position 2^2 should read "BT/ET

Page	Para.	Line	
44			(Abnormal if read operation)."
			Bill Feed Printer:
			Model number is 70/248.
			Conditions represented by bit positions 2^7 and 2^6 are Abnormal conditions.
45	9	3	Change the word "END" to "EOT".
47	6	7, 8	Change latter part of sentence to read, "... or a binary number that selects one of 10 loop channels (1-11, but not 9), depending on the setting of bit 2^7."
48	2	1	Change the first part of the sentence to read, "When specifying forms control through use of the OUT calling sequence and no pointing is desired,..."
48	5	1	Change the first part of the sentence to read, "A check (CHK) is performed before the read and the appropriate,..."
48	10	1	Change first part of sentence to read, "A check (CHK) is performed before the read and the appropriate,..."
49	1	1	Change first part of sentence to read, "No check (CHK) for this read is made,..."
49	6	1	Change first part of sentence to read, "No check (CHK) for this read is performed,..."
49	8	4	Delete the word "Auxiliary".
50	3	2	Correct the typographical error "esires"to read "desires".
52	2		Delete paragraph under "DEV. NO." Replace with "DEV. No. is actual device number (range 0-F)".
56	5	2	Correct typographical error:
			Change "... to 70/15..." to"... the 70/15...".
56	6	2	Change "... <u>five</u> entry (ENTRY) points..." to "... seven seven entry (ENTRY) points...".
56	7	3	Eliminate ", Write Auxiliary" from the list of I/O instructions.

Page	Para.	Line	
59			Figure 2. Block 70. Change wording in block to read, "Halt, operator intervention".
60			Chart 24. Ninth line from bottom of chart — label field is "KEOF": insert a single quote (') in column 22, so that the operand field is "C'$EOF'".
61	10	1	Change explanation of Block 80 to read "Represents the final output processing and any final output functions."
62			Figure 3. Block 50. Indicate that the exit from block 50 is to entry point C by drawing a flow line from block 50 to a circle containing "C".
64			Chart 25. Delete top 7 comment lines.
72			The following description replaces the PROGRAM HALTS description:

Listing of Program Halts ♦ A program halt takes place whenever an error condition occurs. Following is a list of halt display characters (contents of M register on console), their meaning and suggested operator action.

M Register	Definition	Operator Action
00	I/O error from logical device 00.	Check device for inoperable condition; press START to retry.
01	I/O error from logical device 01.	Check device for inoperable condition; press START to retry.
08	I/O error from logical device 08	Check device for inoperable condition; press START to retry.
09	I/O error from logical device 09.	Check device for inoperable condition; press START to retry.
8F	Nonrecoverable I/O error.	Restart assembly. The device number of the inoperable unit is stored in the standard location $P (Byte 130_{10}).

System 70/15 **Date** March 25, 1966 **No.** 2

70-15-602

70/15 ASSEMBLY SYSTEM REFERENCE MANUAL
First Printing: April, 1965
Second Printing: October, 1965

General

This bulletin contains revised information to the 70/15 Assembly System Reference Manual. These changes and additions advise the reader of the latest information available concerning the 70/15 Assembly System Reference Manual. This information will be included in the next printing of the manual.

Page

2 Under "Related Programming Systems" insert "07-Assembly System Device" in fifth sentence, second paragraph.

6 Under "Additional Restrictions on Symbols" change paragraph 3, second sentence to read" ... for a 4K processor is 80." Third sentence to read " ..... for the Tape Assembly System is 495.

10 Under "Character" change second sentence to read "The character can be any of the printable characters".

16 Under "Assembler Instructions" insert after last paragraph:

"Service Routine Linkage Instructions".

MP - Multiply
DP - Divide

"The Assembler will generate an actual machine instruction in the format of the Add Decimal (AP) instruction. The operation codes will be the 70/25 operation codes for MP and DP mnemonics. (See 70/15 Utility Routines Manual 70-15-301)".

17 Under "Start-Start Program (Cont'd)" second sentence of last paragraph delete "but is flagged as an error".

21 Under "Character Constants -C" add sentence to first paragraph. "A character constant must be a printable character".

Page

22 Under "Hexadecimal Constants-X" change second paragraph to read "If the hexadecimal description specifies an odd number of digits, a zero digit is appended to the rightmost end of the constant".

26 Insert after "Chart 15".

Service Routine Linkage Instructions

MP-Multiply
DP-Divide

The Assembler recognizes the MP and DP mnemonics as valid 70/15 instructions and generates machine coding in the Add Decimal (AP) format. The computer will enter the P2 state when the generated machine codes for the MP or DP are encountered. Servicing in the P2 state will be performed by the Multiply/Divide Service Routine (see the 70/15 Utility Routines Manual for a complete description of the Multiply/Divide Service Routine). The purpose of the mnemonics is to simplify the coding of the parameter lines for the Multiply/Divide Service Routine. The format is as follows:

Name	Operation	Operand
Symbol	MP	$S_1(L_1)$, $S_2(L_2)$
Symbol	DP	$S_1(L_1)$, $S_2(L_2)$

Chart 15A Example of Multiply/Divide Service Routine Parameter Line

Name	Operation	Operand	Operation
	MP	FOUR(2),FIVE(2)	Multiply FOUR by FIVE
	DP	SIX(5),TWO(2)	Divide SIX by TWO

66 Under "First Pass" delete all but first sentence of first paragraph.

Under "Device Assignment/Interchangeability" add sentence to third paragraph". "The card assembler does not provide input/output control for magnetic tape".

Page

70 Under "Card Systems" change procedure 1 to read "..... with the five I/O Define cards. In Figure 5 change "4 I/O Define Cards" to read "5 I/O Define Cards".

71 Under "Tape Library Systems" change paragraph 1 to read "---five I/O Define Cards ----". Change paragraph 5 to read "The five I/O Define Cards will be ----".

72 Under "Stacked Assemblies" delete paragraph and insert "In the 70/15 Tape Assembly, provision is made to automatically process sequentially stacked programs. The output tapes will not be rewound until the assembly is terminated by the appearance of a $EOF card. An end file tape mark will be written on each output tape before rewinding."

Under "Program Halts" delete paragraph and insert the following:

"A program halt takes place whenever a non-recoverable error condition occurs. A list of halt display characters (Contents of M register on console), their meaning, and suggested operator action are listed below. It should be noted that source language errors will not cause error halts unless a non-printable character is encountered. Non-recoverable I/O errors will cause the processor to halt with an $(8F)_{16}$ displayed in the M register. The trunk and device number which caused the error are stored in the standard location $P (Byte 130)".

Card Assembly Program Halts

Display Chars.	Meaning	Branch	Action
01	READ ERROR	Continue	Restack card reader with error card followed by remainder of card input and depress start.
02	PRINT ERROR	Continue	Note and depress start.
03	PUNCH ERROR	Continue	Note and depress start.
8F	I/O INSTRUCTION NOT ACCEPTED.	Continue	Check inoperable condition of trunk and unit stored at $P+5. Depress start to re-execute the operation.
F1	END OF PASS I	Continue	Place Pass II and source program in card reader and depress start.
FF	END OF ASSEMBLY.	None	None

Tape/Card Assembly Program Halts.

PASS I

Display Chars.	Meaning	Branch	Action
00	READ ERROR	Continue	If card reader, restack card reader with error card followed by remainder of card input, and depress start. If tape, depress start to retry.
01	WRITE ERROR INTERMEDIATE TAPE	Continue	Depress start to retry write.
08	WRITE(PUNCH) ERROR OBJECT PROGRAM	Continue	Depress start to retry write (PUNCH).
09	WRITE ERROR ASSEMBLY LISTING DEVICE	Continue	If tape, depress start to retry write. If printer, check for nonprintable character in source program card or invalid data in standard data area, make correction, and restart Assembly.
8F	I/O INSTRUCTION NOT ACCEPTED	Continue	Check inoperable condition of trunk and unit stored at $P+5. Depress start to re-execute the operation.
90	TAPE LABEL PURGE DATA HAS NOT BEEN ACCEPTED.	Continue	$P contains the device number. Depress start to override label check.
B0	$EOF STATEMENT IN MIDDLE OF SYM-BOLIC DECK.	Continue	Depress start to bypass $EOF statement.
F1	END OF PASS I	Continue	Place Pass II and source program in card reader and depress start.
FE	INPUT DEVICES ARE IMPROPERLY DEFINED	Continue	Correct I/O define cards, reload card reader, and depress start.

Tape/Card Assembly Program Halts (Cont'd)

PASS II

Display Chars.	Meaning	Branch	Action
01	READ ERROR SOURCE INPUT DEVICE	Continue	If card reader, restack card reader with error card followed by remainder of card input and depress start. If tape, depress start to retry.
08	WRITE(PUNCH) ERROR OBJECT PROGRAM DEVICE	Continue	Depress start to retry write (PUNCH)
09	WRITE ERROR ASSEMBLY LISTING DEVICE	Continue	If tape, depress start to retry write. If printer, check for nonprintable character in source program card, make correction, and restart Assembly.
8F	I/O INSTRUCTION NOT ACCEPTED	Continue	Check inoperable condition of trunk and unit stored at $P+5. Depress start to re-execute the operation.

Page

72 Replace item 7 under "Tape Library Systems" with the following paragraph.

7. If the source program is on magnetic tape, there is no halt between the first and second pass. If the source program is on cards and magnetic tape has been designated as the second pass input device, the cards read will be written to magnetic tape eliminating a halt between the first and second pass.

76 Under "Loader Call Card" change "hhhh" definition to read "... is program to be loaded (0328)".

System 70/15 **Date** September 28, 1966 **No.** 3

ASSEMBLY SYSTEM REFERENCE MANUAL #70-15-602
(FIRST PRINTING: APRIL, 1965)
(REVISED: OCTOBER, 1965)

General: This bulletin contains revised information to the 70/15 Assembly System Reference Manual. These changes and additions advise the reader of the latest information available on the 70/15 Assembly System Reference Manual. This information will be included in the next printing of the manual. When required, subsequent bulletins may be issued to maintain the accuracy of the manual. The revised information is itemized below.

Page 6 — Under Additional Restrictions on Symbols, change the third paragraph to read:

3. The size of the source processor's main storage (4K, 8K, or 16K) determines the maximum number of symbols that can be used in one program. In the Card Assembly System, the maximum number of symbols for a 4K processor is 80; maximum for an 8K processor is 700; maximum for a 16K processor is 2,000. In the Tape Assembly System, the maximum number of Symbols for an 8K processor is 495; the maximum for a 16K processor is 1,860.

7 — Change the second paragraph under The Location Counter to read:

The Assembler normally compiles programs for a 4K, 8K, or 16K memory. The location counter maintains addresses up to 16K. However, the Assembler flags any statement which sets the Location Counter above the limit stored at $148\text{-}149_{(10)}$. The statement, nevertheless, will be processed with the excess Location Counter value.

17 — Change the third paragraph to read:

If a START directive is not written as the first statement of the card assembly, the Location Counter is set to zero and the first statement is flagged.

Page

17 (Cont'd) If a start directive is not written as the first statement of the Tape Assembly all cards will be transcribed to logical 08 until a START directive is encountered. The Location Counter is set to zero if the START directive operand is blank.