GE-425/435 Macro Assembly Program Reference Manual

ADVANCE INFORMATION



GE-425/435 MACRO ASSEMBLY PROGRAM REFERENCE MANUAL

ADVANCE INFORMATION

DECEMBER 1963

The Material contained herein is advance information relating to programming and computer applications and is supplied to interested persons without representation or warranty as to its content, accuracy, or freedom from defects or errors. The General Electric Company therefore assumes no responsibility, and shall not be liable for damages arising from the supply or use of this material.



COMPUTER DEPARTMENT PHOENIX, ARIZONA

FOREWORD

This manual is a reprint of the Advance Information draft published in December 1963. To make the manual less bulky and easier to use we have reduced the draft photographically and placed two pages side-by-side on a sheet. Each reduced page is carefully numbered to make it easy for you to read the pages in the right sequence.

The following manuals used in conjunction with the GE-425/435 Macro Assembly Program Reference Manual, are identified by their respective CPB numbers:

GE-425/435 Basic Input/Output System Reference Manual	CPB-352
GE-425/435 Extended Input/Output System Reference Manual	CPB-353
GE-425/435 Loader Reference Manual	CPB- 354
GE-425/435 Program Monitor Reference Manual	CPB-355
GE-425/435 Sort Generator Reference Manual	CPB-356
GE-425/435 Merge Generator Reference Manual	CPB-357
GE-425/435 Report Program Generator Reference Manual	CPB-358

Copyright (c) 1963

by

GENERAL ELECTRIC COMPANY

The Macro Assembly Program has been designed to provide a convenient, flexible programming environment. The program is capable of translating two levels of language into the instruction formats of the GE-425 and GE-435 computers.

The macro assembly language provided is a logical and powerful extension to the essentially machine oriented, basic assembly language, which is also available in the Macro Assembly Program. This extension takes the form of macro-instructions that can be used to perform operations upon data described in a field-oriented manner.

Macro-instructions have been provided to accomplish the following functions:

- Input/output operations
- Arithmetic operations
- Data movement
- Procedure control

The input/output macro-instructions communicate with the basic input/output system and the extended input/output system. This in turn provides for standardization and ease of programming for all peripheral equipment available for the GE-425/435. For complete documentation of these two software packages see the Basic Input/Output System and the Extended Input/Output System Manuals.

SCOPE OF THIS MANUAL

This manual is intended as a reference for the Macro Assembly Program. The manual consists of four sections, each of which is described briefly below:

Basic Assembly Language -- The reader is first introduced to the basic, "one-to-one" language of the assembler. The format for this symbolic language is discussed. The GE-425/435 computer instruction repertoire is listed, followed by a description of the complementary instruction words which must accompany certain of the instructions. Pseudo-operations, which instruct and control the assembly process are described individually.

Macro Assembly Language -- The second section of this manual describes the macro assembly language, with which one instruction may generate several computer operations. A general discussion of the source language is presented first, followed by four chapters which describe the four divisions of the macro assembly language. These divisions are:

IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION

PROCEDURE DIVISION

The third section of this manual describes the assembly program functions of the Macro Assembly Program.

The program operating instructions are contained in the fourth section of the manual.

Each of the first two sections is preceded by a detailed Table of Contents.

Before proceeding into the detailed use of the languages, an introduction to the computer configuration, coding form, and program operation is in order.

MINIMUM SYSTEM CONFIGURATION

The Macro Assembly Program requires the following minimum configuration:

- GE-425/435 Central Processor with an 8K memory
- Magnetic tape control unit
- Four tape handlers
- Printer
- Card reader
- Card punch.

H Material so marked is not available at the present time.

Additional memory modules are used to advantage by the program, and additional magnetic tapes may be substituted for slower peripherals.

PROGRAMMING FORM

The programming form used for the Macro Assembly Program (see Figure 1) provides for the use of one of the languages, or for a mix of the basic and macro assembly languages. All source language is written onto this one form, representing the columns of the 80-column card into which each line entry is to be punched. Because of the dual headings on the form, it may be used for different divisions of the macro assembly language.

An overall view of the use of each of the columns is given in Table 1, and a reference is listed to a further explanation of the information contained therein. The first group of field entries represent the top level of the dual headings, and includes entries used by both the Procedure Division and the basic assembly language, as well as the one-level headings shared with the Data Division. The entries which are listed below the double lines are those pertaining to the lower level of headings used only by the Data Division.

GE-425/435 PROGRAMMING FORM

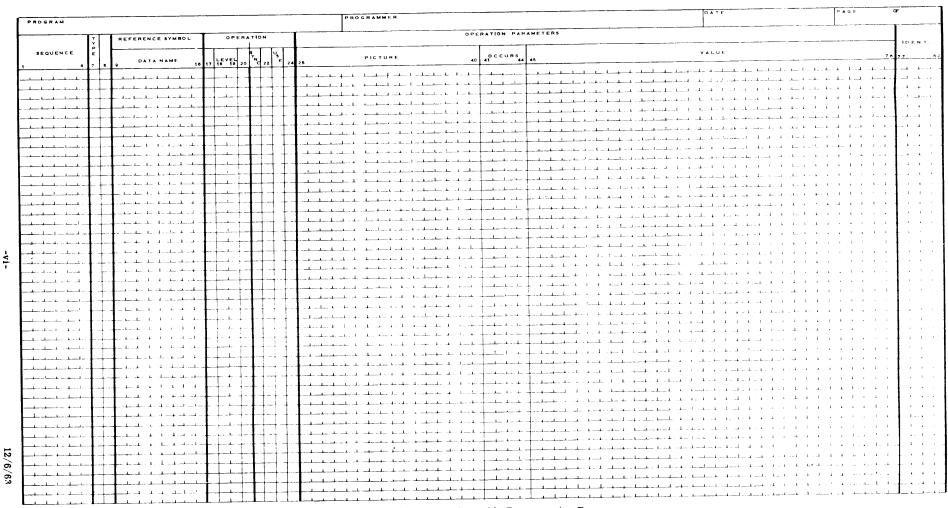


Figure 1. Macro Assembly Programming Form

Table 1. Programming Field Entries

Heading	Col. No.	Macro Division or Basic Coding	Entry	Reference Chapter
Sequence No.	1-6	Data Procedure Basic	Sequence numbers checked using GE-425/435 collating sequence	2-7,21 2-58 1-3
Туре	7	Data Procedure Basic	*,C,R *,C *	2-7,32 2-9,58 1-3
Reference Symbol	9-16	Procedure Basic	An eight character(or less) program reference symbol.	2-9,59 1-3,8
Operation	17-24	Data Procedure Basic	File Parameters A macro instruction, program mnemonic instruction or operation. Mnemonic code.	2-41 2-9,60 1-3,41,71
Operation Parameters	25-76	Data Procedure Basic	File parameter options The parameter needed to complete the operation function.	2-41 2-10,60 1-4,13
Iden	77-80	Data Procedure Basic	Any four character identification	2-9 2-10,61 1-4
Data Name	9-16	Data	Symbolic Data Name	2-22
Level	18-19	Data	FD,01,02,03,04	2-23
Sync	21	Data	L,R	2-30
Use	23	Data	U,I,S	2-35
Picture	25-40	Data	9 , X , A	2-24
Occurs	41-44	Data	1, 2, 3,	2-38
Value	45-76	Data	Any Working Storage Value	2-29

 $\begin{tabular}{ll} \underline{Note}\colon & The \ entries \ for \ the \ Identification \ and \ Environment \ Division \ are \ discussed \\ & in \ Chapters \ IV \ and \ V. \end{tabular}$

PROGRAM OPERATION

Source program 80-column punched cards are prepared from the programming form and can be entered into the Macro Assembly Program either from the card reader or from magnetic tape as card format records.

The Macro Assembly Program is composed of three phases:

- 1. Translator phase
- 2. Selector phase
- 3. Assembler phase

The translator phase of the Macro Assembly Program analyzes the source input deck and determines if the macro-assembly language has been used. If the source program does not contain an Environment or Data Division, the assembler phase is loaded into the computer bypassing the phases not essential to the assembly.

A complete explanation of the phases and their functions is developed in Sections One, Two, and Three. (Refer to flow diagram in Assembly Program Organization, Section Three.)

CONTENTS

I.	INTRODUCTION	1- 1
	Scope of this Section of the Manual	1- 2
II.	PROGRAMMING FORM FIELDS	1- 3
III.	LANGUAGE FORMAT	1- 7
	Machine Instruction Format	1- 7 1- 8 1-10 1-12 1-13
IV.	RELOCATABLE SEGMENTS	1-31
	Local and Global Symbols	1-31 1-34 1-35
v.	COMPUTER OPERATION CODES	1-41
	Listing of Computer Operation Codes Complementary Instruction Words	1-41 1-55
	Branch Counter BCTR. Move Counter MCTR. List Pointer Word LPW. Data Control Words DCWC-DCWW. Address Modification Sequence Words Second Address Sequence Words. Indirect Address Words IAW.	1-55 1-57 1-58 1-60 1-61 1-63 1-64
	Input/Output Mnemonic Operation Formats	1-66
	Class 1 Input/Output Mnemonics	1-66 1-68

Л.	PSEUDO-OPERATIONS	1-71
	System Oriented · Pseudo-Operations	1-73
	Segment Name SGMT	1-73
	Define Internal Global Symbols DIG	1-75
	Define External Global Symbols DXG	1-77
	Define Global Reference Remotely DGRR	1-79
	Define Global Reference Ends DGRE	1-82
	Define Global Reference DGR	1-83
		1-85
	Equals Global Symbol EQUG	1-87
	Call Segment at Load Time CALL	
	Include Library Segment INCL	1-89
	Include Symbolic Segment INCS	1-90
	Memory Allocation Pseudo-Operations	1-92
	Block Started by Symbol BSS	1-92
	Block Started by Symbol in the Loader Area	
	BSSL	1-94
	Block Preceded by Symbol BPS	1-96
	Block Preceded by Symbol in the Loader Area	100
	BPSL	1-98
		1-100
	Last Symbol in Block LSB	
	Last Symbol in Block in the Loader Area LSBL	1-102
	Accumulator Reference Point ARP	1-104
	Accumulator Reference Point in the Loader Area	
	ARPL	1-106
	Accumulator	1-108
	Fill Word FILL	1-111
	Constant Producing Pseudo-Operations	1-113
	Decimal Constants DECS-DECD-DECT-DECQ	1-114
	Alphanumeric Constant AN	1-115
	Last Symbol Alphanumeric LSAN	1-118
	•	
	Octal Constants OCTS-OCTD-OCTT-OCTQ	1-120
	Assembly Output Control Pseudo-Operations	1-122
	Title TTL	1-122
	Eject Page EJT	1-123
	Identify Binary Output IDEN	1-124
	Full Binary Cards FULL	1-125
	Symbolic Analyzer SYAL	1-126
	Dynamic mary Lot Dimi	1 14

12/6/63

Miscell	aneous Pseudo-Operations	1-127
Ori Equ Equ Pro Con Tra	igin ORG	1-127 1-129 1-130 1-132 1-133 1-134 1-136
TABLE	S	1-139
Α.	Alphabetic Listing Computer Instruction Repertoire	1-139
В.	Alphabetic Listing Pseudo-Operations	1-151
SUPPL	EMENT TO CHAPTER IV	1-153

1-iii 12/6/63

I. INTRODUCTION

As the word "basic" suggests a basic assembly language is the nearest to the machine language. The GE-425/435 basic assembly language consists of a series of symbolic codes called mnemonic operation codes; one code exists for each operation which the computer can execute.

Designed as a subset of the more powerful and versatile Macro Assembly Program, the basic assembly language operates in two environments. When using the Macro Assembly Program the basic language may be combined with macro instructions as described in the Macro Assembly Language section of this manual. In other applications a programmer may elect to code entirely within the basic language. As explained in the General Introduction to this manual, the Macro Assembly Program has one assembler which assembles both the basic assembly language source instructions and the translated macro assembly language instructions.

All large programs can be thought of as a combination of logically distinct parts, or segments. The assembler permits the user to divide his coding into segments and to assemble each segment separately. During checkout only those segments with errors need to be reassembled.

Two modes of assembling are possible with this assembler: absolute or relocatable. In many cases, a relocatable, segmented environment is advantageous as it provides an opportunity for easy use of the more powerful tools provided within the GE-425/435 software system. Use of the relocatable format simplifies the debugging stage of program development. Changes to parts of a production program may be made easier in a relocatable environment. In addition, when a program is written in a relocatable mode, the programmer is able to call in subroutines from the library at load time, and he is assured that the latest version of the subroutine is being used.

The basic assembly language was designed to provide --

- a symbolic representation of the entire GE-425/435 instruction repertoire.
- a set of pseudo-operations for the reservation of memory areas and for the handling of decimal and octal constants.
- the optional capability of referencing a program that exists in a subroutine library.

- compatibility with the macro assembly language.
- a symbol analyzer at the termination of the assembly.
- a convenient coding form identical to that used for the macro assembly language.

SCOPE OF THIS SECTION OF THE MANUAL

The purpose of this section of the Macro Assembly Program Manual is to explain the basic assembly language and its related assembly process. The format of the basic language and the rules governing its entry into the coding form are first explained. Following this, a general discussion on the relocatable mode of programming is given to clarify this concept before the basic assembly language instructions are presented. The GE-425/435 instruction repertoire, although listed in this section, is not described in detail. A complete explanation of these instructions is found in the GE-425/435 Reference Manual.

In reading through the discussion of the basic language format, the reader will encounter several examples showing pseudo-operations which are not defined until Chapter VI - Pseudo-Operations. Returence to Table B which lists the functions of all of these operations, may aid the reader until he has become familiar with these operations in Chapter VI.

II. PROGRAMMING FORM FIELDS

The GE-425/435 Programming Form is used for both the macro and the basic assembly language. Figire II-1 illustrates this form. The entries on the form are shown merely to indicate their placement in the basic assembly language. An explanation of the various entries will be given in Chapter III of this section. In the paragraphs below the headings used by the basic assembly language entries are described.

Sequence (col. 1-6)

To insure proper sequence within the source input, the programmer may place sequence numbers in columns 1-6 of his programming form. Columns 1, 2, and 3 may be related to a coding form page number and columns four and five to a line number within the page. Column 6 is then available for insertions which may be necessary later. The sequence numbers assigned by the programmer are checked according to the GE-425/435 collating sequence. If a sequence error is detected during assembly, it is flagged as an error on the listing and the assembly continues.

Type (col. 7)

An asterisk (*) in column 7 indicates that the entire source input contains comments. These comments are listed on the assembly output; they have no effect on the assembly process.

Reference Symbol (col. 9-16)

The Reference Symbol enables the programmer to assign symbols to instructions, constants, etc. The Reference Symbol may be written anywhere within the eight-column field.

Operation (col. 17-24)

The Operation portion of the coding form is used for the entry of the mnemonic codes representing the GE-425/435 machine operation or the pseudo-operation which controls and instructs the assembler. These codes may be written anywhere within the operation field.

On occasion, the programmer may wish to specify a value for the operation field directly, and avoid the use of a mnemonic operation which may be meaningless. For example he may wish to indicate the type of a parameter

in a subroutine call. This is done by placing the desired octal value in the two most significant columns of the Operation field.

Operation Parameters (col. 25-76)

The parameters needed to complete the function indicated by the Operation are entered left-justified in the Operations Parameters portion of the coding form. A parameter may be a symbolic address, an absolute address, an expression, a literal, or a value, depending on the operation with which it is associated. (Further discussion on each of these parameters is found in the chapter on Language Format.) In addition comments may be placed in the Operations Parameters following the first blank column.

Identification (col. 77-80)

The Identification provides four columns for entering the identification of the symbolic assembly input of the program. The four characters are assigned by the user; they are not checked by the assembler, but are printed on the assembly listing.

1-3 12/6/63 1-4 12/6/63



GE-425/435 Programming Form

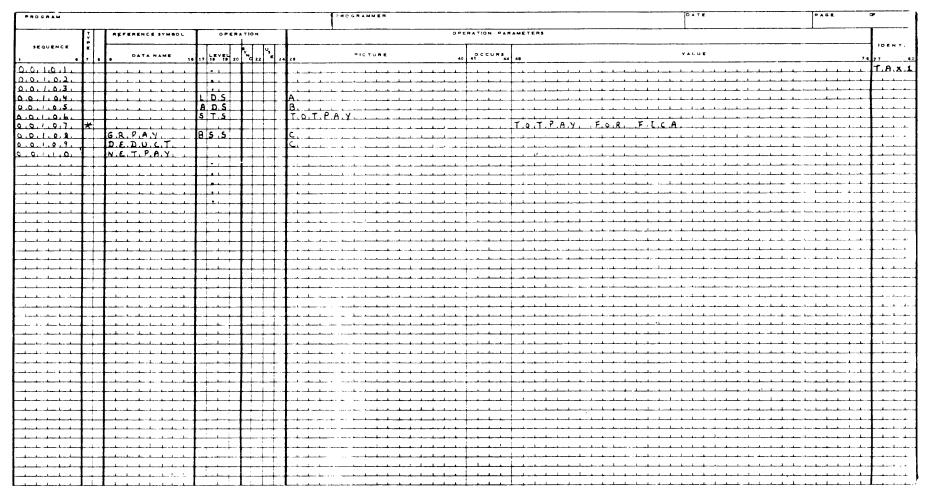


Figure 1. Sample Coding Form

III. LANGUAGE FORMAT

The basic assembly language is composed of two types of instructions: the GE-425/435 instruction repertoire and the pseudo-operations which control and instruct the assembler. These instructions are entered in the coding form using the entries defined in the previous chapter. This chapter will describe the format of the various language entries. A brief discussion of the GE-425/435 instruction format precedes the language format description.

MACHINE INSTRUCTION FORMAT

The GE-425/435 instruction repertoire includes two types of instructions: the single address instruction and the two-address instruction. A detailed discussion of these two types of instructions is found in the GE-425/435 Reference Manual. For convenience to the reader a brief definition of each type is given below.

Single Address Instructions

The single address instructions consist of three parts: the Operation Code, the Address Field, and the Address Control Field. The three parts are located in the 24-bit GE-425/435 word as shown below:

23	18 17 16 15	14 0	
Operation	Address		
Code	Control	Address Field	
	Field		

The Operation Code determines the operation to be performed; the Address Field supplies the address for the operation; the Address Control Field indicates whether the address is to be modified by one of six fixed index registers changed by an address modification sequence, or used without modification.

Two Address Instructions

The format of the first word of the Two Address instruction is identical with that of the Single Address instruction; however, the Address Control Field has a somewhat different significance. The Address Control Field

indicates whether the first address should be modified before entering a Second Address Sequence, or whether the location of one of the fixed index registers should be used as the second address.

SYMBOLIC INSTRUCTIONS

The format of the symbolic instruction is similar to that of the machine instruction, consisting of an Operation Code, an Address, and an Address Control. The symbolic instruction has no definite memory location until assembly time. However, a symbolic instruction does have a symbolic location known as the Reference Symbol. When a programmer relates a Reference Symbol to a symbolic instruction, the Reference Symbol represents the memory location which the assembler will assign to the instruction.

Symbolic instructions are entered on the coding form as follows:

- The symbolic location is written in the Reference Symbol columns.
- The mnemonic operation code is written in the Operation columns.
- The Address and Address Control are written in the Operation Parameters columns.

The Address begins in the first column of the Operation Parameters. It may be absolute, symbolic, or a combination of both, (calculated) or the Address may be a literal. The Address may be terminated by a blank or by a comma; if it is terminated by a blank, the assembler assumes an Address Control Field of zero; if it is terminated by a comma, the assembler processes the remaining Operation Parameter as an Address Control Field. The Address Control Field may be an absolute, symbolic, or calculated value.

Symbolic Reference Entries

Ey using Reference Symbols, the programmer equates his assigned symbol to the data with which it is associated. The symbol becomes synonymous to the memory cell in which the data is located; the data may be referenced by its symbol, eliminating the need for assigning and keeping track of memory locations.

The following rules govern the construction and use of symbols.

 A Reference Symbol may be written in columns 9 through 16 of the coding form. It may contain from one to eight alphanumeric characters. 2. A symbol is built using any of 37 characters of the GE-425/435 character set. The allowable characters are:

Alphabetics A, B, C, . . . Z

Numerals 0, 1, 2, 9

Special Character _ ~ (tilde)

At least one character of a symbol must be alphabetic.

- 3. A Reference Symbol may be floated anywhere within the eight columns provided for the Reference Symbol. It should not contain imbedded blanks because blanks are not considered as part of a symbol. If blanks are present they will be deleted. The resulting symbol is right justified, and zeros are placed in the positions to the left of the significant characters.
 - Let Δ represent a blank; then the following all represent the same Reference Symbol on a coding sheet:

 $AZ \triangle \Delta \Delta \Delta \Delta \Delta \Delta \Delta$

 $\triangle AZ \triangle \triangle \triangle \triangle \triangle$

 $\triangle \triangle \triangle Z$

 $\triangle \triangle \triangle AZ \triangle \triangle \triangle$

 $\triangle \triangle \triangle AZ \triangle \triangle$

 $\triangle AZ \triangle \triangle AZ \triangle$

 $\triangle \triangle \triangle \triangle \triangle \triangle \triangle AZ$

 $\mathbf{A} \triangle \triangle \triangle \triangle \triangle \triangle \mathbf{Z}$

All of these Reference Symbols would result in the same symbol:

000000AZ

4. A symbol is defined when it appears as the Reference Symbol of a symbolic instruction or pseudo-operation.

- 5. A symbol must be defined only once within the program. More than one such definition results in an error indication by the assembler each time the symbol is used.
- 6. When a program is written as a series of distinct regions *, different prefix characters may be appended to the symbols used in each section.
- 7. Symbols used in the Operation Parameters field must be defined, or use of them results in an error indication on the assembler listing.
- 8. Symbols used in the Operation Parameters field may not contain imbedded blanks. A blank column in the Operation Parameters columns indicates the end of significant data. Information following a blank column is treated as comments.

ASSEMBLER PHASE

To fully understand the basic assembly language, a brief description of the assembler process seems appropriate at this point. The assembler has two passes, briefly described below. If a symbolic analysis of the assembled program is desired, it may be performed following the regular assembly operation.

Pass I

Pass I reads the symbolic source input, assigns memory locations to machine instructions and constants, processes pseudo-operations, and constructs a table containing the Reference Symbols which appear in the source program and the relative address which each symbol represents. In addition, Pass I supplies the binary machine operation code for each mnemonic code used by the programmer.

The primary functions of Pass I are the assignment and allocation of memory locations and the construction of the symbol table. In accomplishing these functions, Pass I employs a "location counter" to control all memory assignments made by the assembler. The location counter always contains a value which corresponds to the next memory location which may be assigned by the assembler.

^{*}Subdivisions within segments are referred to as regions.

As Pass I processes a symbolic record, it identifies it as either a symbolic instruction or pseudo-operation. If it is a symbolic instruction, the assembler increments the location counter by one and looks for a Reference Symbol. If there is a symbol, it is placed in the symbol table with the value which was in the location counter prior to incrementation. A pseudo-operation can affect the location counter in several different ways. It may cause it to be incremented by a particular value as in the case of the BSS; or. it may have no effect at all on the location counter as in the case of the EQU. Pass I identifies the pseudo-operation; affects the location counter in the specified manner; and, looks for a Reference Symbol. If there is a symbol, it is placed in the symbol table, with the appropriate value from the location counter, and with any flags which are necessary to identify and retrieve it in Pass II. In either case, the symbolic record and the data gathered by Pass I are written on an intermediate tape for Pass II processing.

Thus, Pass I assigns relative memory locations to all instructions and constructs a symbol table containing all Reference Symbols and the relative memory addresses which the symbols represent.

Pass II

Pass II reads the symbolic instructions from the intermediate tape and produces the binary object program and the assembly listing. Pass I has supplied a machine operation code for each mnemonic operation code. Pass II must supply an Address and an Address Control from entries in the Operation Parameters. In performing this function, Pass II relies, for the most part, upon the data in the symbol table.

Pass II searches the Operation Parameters of each instruction for symbols. When it finds a symbol. Pass II goes to the symbol table to obtain the value assigned to that symbol in Pass I. It replaces the symbol with the value from the symbol table and thus arrives at the address field of the instruction.

When a symbolic instruction has been completely processed, it is printed on the assembly listing with its octal location, the contents of that location, and an indication of any errors detected during processing. The assembled instruction is stored in a binary program card area to be punched when the area is filled.

When the final entry from the intermediate tape has been processed, listed and punched, Pass II and the assembly program are complete. An assembly listing has been printed and a binary program has been punched.

Symbolic Analyzer

The Symbolic Analyzer lists every symbol with its address and all references to each symbol. Each reference will consist of the operation code that references the symbol, the page and line number of the reference, and the address of the reference.

CLASSIFICATION OF SYMBOLS

As mentioned in the brief assembly description above, the assembler constructs a symbol table in Pass I. When a symbolic reference is encountered during a relocatable assembly it must be classified as either relocatable or absolute.

Absolute Symbols

An absolute symbolic reference in the Reference Symbol field of an instruction represents either a fixed machine location or a value. In either case, because the absolute symbolic reference will probably appear in the Operations Parameters field of subsequent instructions, the assembler will mark the symbolic reference as absolute. In the following example all symbols are treated as absolute.

Example

REFERENCE SYMBOL	OPERA	TION	OPERATION PARAMETERS
DATA NAME 16	17 14 19 20	S V C 22 E 24	PICTURE 25
	• • • • • • • • • • • • • • • • • • • •		
			
F, X, R, 3,	EQU		3
F.X. P. 2	ε φ.υ.α.		2
T, Y, P, E, C, H, A, N	E 9,0 Ø		2.0.
D.1.V.2.	E 6.10		
P.L. U.S	E q.u.k		6.0.
			* * * * * * * * * * * * * * * * * * *

Relocatable Symbols

All Reference Symbols that are not classified as absolute are marked as relocatable. Therefore, any Reference Symbol which does not represent a fixed machine address or a value is relocatable. (See Chapter IV, Relocatable Segments.)

OPERATION PARAMETER ENTRIES

Operation Parameter entries may be of four different types:

1. Absolute - a decimal integer

2. Symbolic - a symbol

3. Calculated - an expression

4. Literal - a literal

Normally all four entries appear in a program; they are described in the following paragraphs.

Absolute Entries

To form an Absolute Entry, the programmer determines the decimal value he wishes to employ in an instruction and places that value in the proper place in the Operation Parameters field (or column).

Example

	REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS
ŧ	9 DATA NAME 16	17 18 19 20 C 22 F 24	PICTURE 25
		L D.S	1.0.0.4
		5 7 S	1.0.0.6.3.
		B.R.U	1.0.2.5

1-13

Note:

Programming in absolute places the responsibility of memory allocation upon the programmer rather than upon the assembler. For this reason it is recommended that absolute programming be avoided.

Symbolic Entries

To form a Symbolic Entry, the programmer uses the symbols which he has assigned to his instructions and data. He references his instructions by their assigned symbols.

Example

Assume that the current value of the location counter is 1000.

1	REFERENCE SYMBOL	Γ	OPERATION						OPERATION PARAMETERS		
9	DATA NAME	1,,	LEVEL 18 19	20	S _Y N _C	22	u s E	24	PICTURE.		
		L	DIS			4			GRPAY		
		A.	0,5		ļ	-			D.E.D.UIC.T.		
	L.	1.7	T. 5		1	ļ	ļ		N.E.T.PIAIX, JA.		
	_1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _	В.	R.u		L-	-	ļ		* <u>* , 4, </u>		
	GRPAY	3.	SS		L.	<u> </u>		ļ	Alamana a manama mana mana basa		
	DIE DIVIET.	B	5.5		١.		ļ	: 	العاصفية فللطبية فالمنافقة الالتطابية الفاراق للم		
	N.E.T.P.A.Y	18.	5.5		L.		· 		The community of the state of t		
	A	3	UI Q.				· 		Burker and the second		
		1		-	l		+		La de la companya de		
		•									

During Pass I, the assembler assigns values to the symbols GRPAY (1004), DEDUCT (1005), and NETPAY (1006). The symbols and their assigned values are stored in the symbol table. During Pass II, symbols in the Operations Parameters are "looked up" in the symbol table and replaced by the appropriate values from the symbol table. Thus, the assembly program provides an address for the programmer's symbolic address giving:

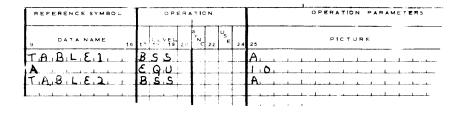
LDS	1004
ADS	1005
STS	1006,3
BRU	1007

The asterisk (*) is a special type symbol; the assembler replaces it with the current value of the location counter.

Previously Defined Symbols

Symbols are assigned values during Pass I processing. Since the Operation Parameters are normally processed during Pass II, it does not matter where within a program a symbol is defined so long as it is defined. However, in the case of several of the pseudo-operations (BSS, BSSL, LSB, LSBL, BPS, BPSL, ARP, ARPL, ACUM, ORG and EQU) the address must be calculated in Pass I since these instructions either control the Pass I location counter or cause an entry in the symbol table during Pass I.

When an entry containing one or more symbols appears in the Operation Parameters field of one of these pseudo-operations, the assembler must use the symbol table to calculate the value of the entry. Obviously the assembler cannot calculate that value unless all symbols appearing in the entry exist in the symbol table. They are in the symbol table only if they were defined prior to the pseudo-operation which uses them. Therefore, the pseudo-operations BSS, BSSL, LSB, LSBL, BPS, BPSL, ARP, ARPL, ACUM, ORG and EQU demand that symbols used in Operation Parameters be previously defined. Consider the following example:



In Pass I, the assembler will attempt to reserve the space for TABLE1 by adding the symbolic value A to the location counter. The assembler goes to the symbol table to find the value of A. However, A has not yet been defined and is not in the symbol table. Therefore, the location counter cannot be incremented and space cannot be reserved. The same procedure takes place for TABLE2; A is now in the symbol table and ten memory locations are reserved.

Calculated Entries

 $T_{\rm D}$ form a calculated entry the programmer states the entry as an expression, thus allowing the assembler to calculate the value of the entry.

An expression is a series of symbols and/or integers called elements which are connected by the operators: +, -, *, and /. These operators $r \in present$ the arithmetic operations: add, subtract, multiply, and divide, respectively. According to the normal arithmetic rule, the elements of an expression are evaluated from left to right, with multiplication and division preceding addition and subtraction.

Example Calculated Addressing

Assume that the current value of the location counter is 1000. Note that the * in this case is an operator. The * as an element represents the current value of the location counter.

		T REFERENCE SYMBOL OPE								OPERATION PARAMETE	
SEC II NOF	E ,	8	DATA NAMI	16	1,,	EVEL	S YN C 2:	U _S E	24	25 FICTURE	
. يند ا لايند .	ļ			<u> </u>	L	DS				G.R.P.A.Y.+.1. B.+.T.	
	l		L		A	DIS	-			D.E.D.U.C.T.+ 11 .T - B	
4 1. 1.			- 1 1 t		S.	Ts⊥		1		N.E.T.P.A.Y.T.I.	
A	1				8	R.U		-		*,+,2 ,0, , , , , , , , , , , , , , , , , , ,	
<u>B </u>	ļ.		· · · · · · · · · · · · · · · · · · ·		3	Q.U.				2	
G. R. L.P. A.Y.					В	S.S.	<u> </u>	1-1		B	
DED UCT	ł			<u> </u>	8	5.5		11		<u>B </u>	
NLE IT FLA Y	1		1 1 1 1	sl.	В	ی ی				<u>B </u>	
1	ł	- 1		. بلسا	3	6.0		1	i	3	
	•	1	1	باست	ļ.,		I	1			

Pass I assigns values to the symbols B (2), GRPAY (1004), DEDUCT (1006), NETPAY (1008) and T (3). The symbols and their assigned values are stored in the symbol table.

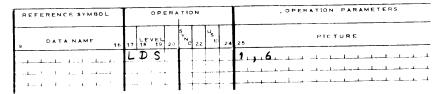
During Pass II, the symbols in the Operation Parameters are replaced by the appropriate values from the symbol table. Having evaluated the expression in the Operation Parameters, the assembler calculates an address giving

LDS	1005, 5
ADS	1007, 1
STS	1009
BRU	1023

The following examples demonstrate the various types of entries allowed in the Operation Parameters field.

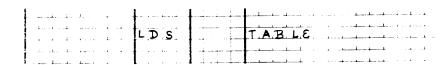
Examples

1. Absolute Address, Fixed Index



Develop an effective address by adding the address portion of index register six to the address portion of the instruction. Load the contents of this location into the single accumulator.

2. Symbolic Address, No Index



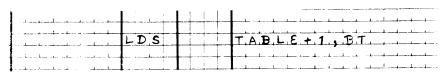
Load the contents of symbolic location TABLE into the single accumulator.

3. Calculated Address, Fixed Index

REFERENCE SYMBOL	OPERATION I	OPERATION PARAMETERS			
DATA NAME 16	17 18 19 20 C 22 E 24	PICTURE 28			
	LD,5	TABLE,+11,3,3, , , ,			

Develop an effective address by adding the address portion of index register three to the calculated address TABLE+1. Load the contents of this location into the single accumulator.

4. Calculated Address, Symbolic Index



Same as Example 3, if BT EQU 3.

5. Calculated Index



Same as Example 4, if G EQU 1, D EQU 2.

6. Next Word Indexing

REFERENCE SYMBOL	OPER -	A TION	OPERATION PARAMETERS				
DATA NAME	1 EVEL 17 18 19 20	S U S E 24	PICTURE 25				
	LDS		T.A.B.L.E.				
	×		1.0.0				

The instruction calls for next word indexing. The AMS word is an index. An effective address is developed by adding the address portion of the Address Modification Sequence (AMS) word to the symbolic address TABLE. The contents of the effective address are loaded into the single accumulator.

7. Multiple Next Word Indexing

REFERENCE SYMBOL	OPER	ATION	OPERATION PARAMETERS				
DATA NAME	LEVEL 17 18 19 20	S U S E 24	. PICTURE				
	MFM		T.A.B.L.E., E.				
i		X	D. A. T. I.				
		X	2,5, , , , , , , , , , , , , , , , , , ,				
4_4_4_4		Ø P	RATE				
		Y					

If E EQU 7, the instruction calls for next word indexing. The AMS word is an index; its symbolic address is added to the symbolic address TABLE. The AMS word specifies continuation. The next AMS word is examined. It is an index. Its address is added to the previously calculated address. Since no further continuation is specified, the effective address has been developed. One word is moved from that location to the symbolic location RATE.

Content Restrictions

The following restrictions exist on the content of Operation Parameters:

- 1. External global symbols (as described in Local and Global Symbols, page 1 31) are permitted only in the expression that indicates the address portion of Operation Parameters; they are not permitted in the Address Control.
- 2. The value that results from calculating the expression for address control must be absolute. (See Relocation Errors, page 1 35..)

Size Restrictions

The Address Control Field of the GE-425/435 instruction word may contain values which range from 0 to 7. (See word format on page 1 - 6.) It may indicate a fixed index register -- values 1 to 6. It may indicate an Address Modification Sequence -- value 7. It may indicate that no address control is required -- value 0 (this is normally indicated by terminating the Address with a blank, although "0" will produce the same result).

The Address Field of a GE-425/435 instruction word may contain values which range from 0 to 32,767. The full capacity of the Address Field is not required by all GE-425/435 instructions; for example, the value in a Shift instruction can range only between 0 and 31, while the restore value in a Branch Counter can range only between 0 and 511.

As the assembler constructs the fields for a machine instruction, it compares the binary representation of the value which the programmer has specified for a field, with that field's allowable range. When the value exceeds the allowable range, a predictable result always occurs. The assembly retains from the specified value in its binary representation, a number of the least significant bits sufficient to fill the field without overflowing.

Values which exceed the allowable ranges are truncated as follows:

- 1. All Address Control Values are truncated modulo-8
- 2. All Addresses are truncated modulo 32,768
- 3. All Shift Counts are truncated modulo-32
- 4. All counts specified for Branch Counters, Move Counters, LPW's or DCW's are truncated modulo-512.

Literals

During the course of coding a program, there are many occasions when a programmer desires to use constants. To enter these constants, the programmer uses one of the many pseudo-operations that are provided in the basic assembly language or he enters them in the Operation Parameters as literals. A literal specifies the actual value of the data that is to be operated on by the Operation. The allowable length of the literal is determined by the type of literal and the Operation with which it is associated. Any literal which exceeds the number of allowable characters or includes an invalid character will be flagged as a literal error on the assembly listing.

The four types of literals that are allowed in the basic assembly language are alphanumeric, decimal, octal, and binary.

1. Alphanumeric Literals

Alphanumeric constants are entered in a program by means of the AN pseudo-operation, or by entering them as alphanumeric literals in the Operation Parameters. An alphanumeric literal is indicated in the Operation Parameters by preceding the literal with a quotation mark ("), entering the literal, which can be a maximum of 16 characters in length, and terminating the literal with a quotation mark (").

Since the quotation mark is used as a delimiter, it is the only character of the computer's character set which may not be part of an alphanumeric literal.

The Operation normally specifies the number of words of constant information that the assembler will generate for the literal (see page 1 - 30). The assembler right justifies the

literal within the field that represents these generated words. Zeros are inserted in the left most character positions when the literal does not completely fill the field.

Example

REFERENCE SYMBOL	OPER	A TION	OPERATION PARAMETERS
DATA NAME	17 LEVEL 20	S Y N C 22 E 24	PICTURE 25
	-:		
A	L D 6		"PAYR & L. L. REC&RD
R	15 T Q	 - - - - - - - - - -	H. E. A. D. R. +. 3. ". P. A. G. E N. O "
	STD		H.E.A.D.R.+.7.
·	B,R.U	 	
1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	l. : -		
Iu c u n k	כניםן	<u> </u>	10

After executing the instruction at location A, the quadruple accumulator would appear as follows:

After executing the instruction at location B, the double accumulator would appear as follows:

2. Decimal Literals

Decimal constants are entered in a program by means of the decimal data pseudo-operations or by entering decimal literals in the Operation Parameters. A decimal literal is indicated in the Operation Parameters by preceding the literal with the pounds sign (#) and entering the literal, which can be 16 digits in length, and terminating the literal with a pounds sign or a blank. When a literal is used in a two address instruction which refers to a fixed index register, the comma or pound sign may terminate the literal.

A decimal literal must contain only the digits 0 - 9 and may be preceded by a plus (+) or minus (-) sign. If these rules are violated, a flag will appear on the assembly listing.

The Operation normally specifies the number of words of constant information that the assembler will generate for the literal. (See list of literals on page 1 - 30.) The assembler right justifies the literal within the field that represents these generated words. If the literal does not completely occupy the field, zeros are inserted in the left most character positions.

Example

REFERENCE SYMBOL	NOITARAGE	OPERATION PARAMETERS
DATA NAME: 16	1 EVEL S YN C 22 E	24 25 PICTURE
1		
I i se e constante e		
A	LDS	<u> </u>
.a. i i	A. D. S	C. Ø. U. N.T.
	STS	L. Ø. V. N. T. I.
3	L D D	# - 4 8 0 0 0 0 0
	ADD	IP A Y
		and the second s
		La Carlo Car

After executing the instruction at location A, the single accumulator would appear as follows:

0 0 0 1

After executing the instruction at location B, the double accumulator would appear as follows:

0 0 4 8 0 0 0 0

Since the literal is negative, the least significant character of the accumulator must indicate it.

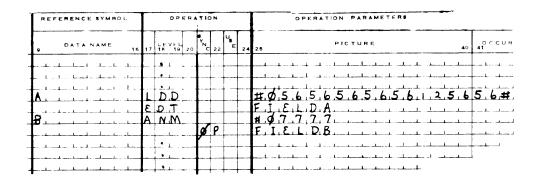
3. Octal Literals

Octal constants are entered in a program by means of the octal pseudo-operations or by entering octal literals in the Operation Parameters. An octal literal is indicated in the Operation Parameters by preceding the literal with two characters: pound sign # and the letter O. The octal literal is entered immediately following these two indicators; it may be a maximum of 32 octal numbers and is terminated by either the pound sign (#) or a blank.

An octal literal must contain only the octal numbers 0 - 7 and must not be preceded by the plus or the minus sign. If these rules are violated, a flag will appear on the assembly listing.

The Operation normally specifies the number of words of constant information that the assembler will generate for the literal. (See list of literals on page 1-30.) The assembler right justifies the literal within the field that represents these generated words. If the literal does not completely fill the field, zeros are inserted in the left most character positions.

Example



After executing the instruction at location A, the double accumulator would appear in octal as follows:

5 6 5 6 5 6 5 6	5 6 1 2 5 6 5 6
-----------------	-----------------

The instruction at location B would cause the octal mask 00007777 to modify location FIELDB.

4. Binary Literals

Binary constants are entered in a program as binary literals. These are indicated in the Operation Parameters by preceding the literal with two characters: a pound sign (#) and the letter B. A decimal integer from 0 to 16,777,215 immediately follows these indicators and constitutes the value of the binary literal. The literal is terminated by either the pound sign (#) or a blank. When a literal is used in a two address instruction which refers to a fixed index register, the comma or pound sign may terminate the literal.

The decimal number to be converted to a binary value must be composed of the digits 0 - 9 and must not be preceded by a plus (+) or a minus (-) sign. If these rules are violated, a flag will appear on the assembly listing.

The assembler constructs the final binary value for the literal by converting the indicated decimal value to binary and right justifying the result in one machine word. The assembler inserts zeros in the unused left most bits of the word.

Example

REFERENCE SYMBOL	OPE	ERATION	OPERATION PARAMETERS S PICTURE 24 25			
DATA NAME 16	LEVEL 17 17 19	S US E 24				
A	LDS		#, 8, 3, 2, #,			
3	S.T.5. A.B.M.		P. d. w. E. R. 2.			
		Ø P	T.W. Ø. 1. N. C.			

After executing the instruction at location A, the single accumulator contains the binary value 32. After executing the instruction at location B, location TWOINC has been increased by the binary value 2.

Immediate Value Literals

For each of the four types of literals just explained, the assembler generates at least one full machine word of constant information. These generated words are placed starting at the end of the main body of coding.

In addition, the complete literal capability can be utilized by the following immediate value instructions:

> AIM AIX CMI CXI MFI LXI Ø X

When literals appear in the Operand Parameters of immediate value instructions, remote constant information is not generated by the assembler. Because of the nature of immediate value instructions, the value of the literal is placed in the address portion of the instruction itself.

The following restrictions apply to use of immediate value instructions:

- Alphanumeric literals must not exceed two characters in length.
- 2. Decimal literals must be integers within the range \pm 799.
- 3. Octal literals must not exceed 5 octal numbers in length.
- 4. Binary literals must be indicated by decimal integers which do not exceed 32,767.

If any of these restrictions are violated, a flag will appear on the assembly listing.

Example

REFERENCE SYMBOL	MBOL OPERATION OF						OPE	RA	т 10	N	PAF	N A S	ET	ERS	3				
DATA NAME	17	EVE	20	s _{>} z _c	ı	U _S	24	25						PIC	: T L	J R F	-		
		• .						L	11			-		L		1		٠	
1. L. L. L. L. L.			1	L	_	<u> </u>		L		٠	_		١	١		١.		1	١
	ļi		<u> </u>	<u> </u>	-	<u> </u>		"	1.	+		7	l		ــــــــــــــــــــــــــــــــــــــ	-1		1	
A	C	X.J		ļ			-	<u> </u>	M	<u>. </u>	_ ,	د	Ь—			-		<u>. </u>	
	B	R.E		ļ	ļ			11	1	٠.	4	ــــــــــــــــــــــــــــــــــــــ	-				-		L .
B	IC.	$\mathbf{X}_{\perp}\mathbf{I}$!	ļ	ļ		Ħ			-7	→	۲_∔	1					
المارية	B.	R.E		1	ļ	<u> </u>		L.,	١,	4		4	ι		٠		_ L		
C	A.	L		_	-	<u></u>		#	Ø		_0	i II		5					
D.	A	TX	,	1		1		ш	B	. 5	10		. 6	2 L	_				

The instruction at location A causes the letter M to be compared to index register 3. The instruction at location B causes the decimal value -14 to be logically compared to index register 4. The instruction at location C causes index register 5 to be increased by the octal number 10 (decimal 8). The instruction at location D causes the index register 6 to be increased by 50.

Literal Pooling - Assembler Action

As previously mentioned, when the assembler generates words of constant information the words are placed starting at the end of the program. The location of the last instruction in the program is known at the end of Pass I. The assembler begins building the required literals in the succeeding memory cell.

While building the required constants, the assembler insures that no duplicate constants occur when constants are identical in word length. When constants are unequal in word length, an attempt is made to match the smaller length constants with portions of the longer constants to avoid duplication.

For example, assume that the final location used by the program is 2020. When the instruction

A LDQ #1#

LOCATION	CONTENTS
2021	0000
2022	0000
2023	0000
2024	0001

And the address for the instruction at location A becomes 2024.

12/6/63

When the programmer uses the following constructions, no additions are made to the literal table:

CONSTRUCT	MON	CONVE	RSION
LDS	#Ø#	LDS	2021
LDD	#Ø#	LDD	2022
LDT	#Ø#	LDT	2023
LDS	#1#	LDS	2024
LDD	#1#	LDD	2024
LDT	#1#	LDT	2024
LDS	#Ø#	LDS	2024
•		•	
•		•	
•		•	
LDS	#B1#	LDS	2024
LDS	"1"	LDS	2024
LDD	"1"	LDD	2024

etc.

Instructions Which Permit Literals

Literals are permitted with the mnemonic operations in the table below. The number of words generated can be influenced by the length of the literal. If the literal specified is larger than the number of words specified by the mnemonic operation, a flag will appear on the assembly listing and the larger literal will be generated (up to a maximum of four words).

Instruction	Mnemonic	Number of Words Generated by the Literal
LOAD SINGLE ACCUMULATOR	LDS	1
LOAD DOUBLE ACCUMULATOR	TDD	2
LOAD TRIPLE ACCUMULATOR	LDT	3
LOAD QUADRUPLE ACCUMULATOR	тъб	4
ADD DECIMAL SINGLE	ADS	1
ADD DECIMAL DOUBLE	ADD	2
ADD DECIMAL TRIPLE	ADT	3
ADD DECIMAL QUADRUPLE	ADQ	4
SUBTRACT DECIMAL SINGLE	SDS	1
SUBTRACT DECIMAL DOUBLE	SDD	2
SUBTRACT DECIMAL TRIPLE	SDT	3
SUBTRACT DECIMAL QUADRUPLE	SDQ	4
COMPARE ALPHA ACCUMULATOR	CAA	1-4*
COMPARE DECIMAL ACCUMULATOR	CDA	1-4*
VARIABLE LENGTH MULTIPLY	VLM	2
VARIABLE LENGTH DIVIDE	ATD	2
EXPLODE	EXP	1
ADD BINARY TO MEMORY	ABM	1
ADD BINARY TO INDEX	ABX	1
SUBTRACT BINARY FROM MEMORY	SBM	1
SUBTRACT BINARY FROM INDEX	SBX	1
AND TO MEMORY	ANM	1

^{*} CAA and CDA allow 1-4 words. The number of words required depend on the length of the literal.

	Instruction	Mnemonic	Number of Words Generated by the Literal
AND TO INDEX		ANX	1
OR INCLUSIVE	TO MEMORY	RIM	1
OR INCLUSIVE	TO INDEX	RIX	1
OR EXCLUSIVE	TO MEMORY	RXM	1
OR EXCLUSIVE	TO INDEX	RXX	1
COMPARE MEMO	RY TO MEMORY	CMM	1
COMPARE INDE	K TO MEMORY	CXM	1
MOVE FROM ME	MORY	MFM	1
LOAD INDEX		LDX	1

IV. RELOCATABLE SEGMENTS

In every program, there are strings of coding which can be isolated as logical entities. For example, in a payroll application, one might choose to identify such entities as initialization, tax calculation, dollar extension, bond deduction, etc. Each of these logical entities is called a segment. A relocatable segment is a logical entity which can be placed anywhere in memory at load time. The assembler assigns relative locations to the instructions and constants, and supplies extra information in the object program to permit the loader to position the program in any convenient memory area and adjust it to operate properly wherever it is placed.

The user may organize a large program into many small independent segments. Each of these segments may be assembled independently of the main program. Since each segment is tested separately, the cause of a bad result may be rapidly isolated. If the same bad result occurred while testing the complete object program, it could be attributed to an error in any of its parts, and could result in unnecessary computer time and programmer effort to isolate the bug. The use of segments reduces the total elapsed time in program development, since the segments can be distributed among several programmers who work simultaneously.

Changes to a relocatable segment are made by correcting the symbolic coding and reassembling only the segment found to be in error. If the corrections change the size or the storage allocation of this relocatable segment, storage assignments in the main program, with which the segment is used, are not affected. Conversely, independent changes in the size and storage assignments of a relocatable main program do not affect the use of the relocatable segments called upon by the main program. In addition, if certain segments are required in other programs, the relocatable binary output can easily be incorporated without a time consuming reassembly.

LOCAL AND GLOBAL SYMBOLS

When a program is written for a relocatable assembly, a means of communication between the series of distinct but interacting relocatable segments is required. The global symbol forms the basis for this communication. A global symbol may be defined as a symbol which is referenced in more than one segment of a relocatable program. A local symbol is one which is referenced only within the segment in which the symbol is defined.

12/6/63

1-30a 12/6/63 1-31

There are two types of global symbols: internal and external. An internal global symbol is "internal" to the segment in which it is defined. An external global symbol is "external" to any segment in which it is referenced except that segment in which it is defined. See the examples and explanation following.

After a relocatable segment has been coded, the user must indicate to the assembler, by use of a pseudo-operation, those symbolic references which the segment requires for communicating with other segments at execution time. He must further indicate whether the global symbols are internal or external by his choice of pseudo-operation.

A detailed description of the pseudo-operations which provide the necessary indication to the assembler for global symbols is presented in Chapter VI, Pseudo-Operations. At this point, only the general concept of the interrelationship of segments is given.

Consider the example on the opposite page in which lines of coding appear in three separate segments.

In segment 1, there are three symbolic references: FICA, PRINT, and A. A and PRINT are referenced only within segment 1 and, hence, are not required for communication purposes. These symbolic references are local symbols. Segment 1 requires communication with segment 2; the symbol reference FICA is common to both segments. To insure proper linkage at load time, the user must indicate that FICA is global when segment 1 is assembled, and again that FICA is global when segment 2 is assembled. In segment 1, FICA is termed an external global reference because it is defined outside the segment. In segment 2, FICA is termed an internal global reference because it is defined within the segment. Appropriate pseudo-operations for indicating internal and external global symbols are found on pages 1-74 and 1-76.

i	REFERENCE SYMBOL	ı	0	PER	1				1	OPERATION PARAMETERS
	DATA NAME	17	LEV	EL 20	5 Y,	vc :	22	U _S	24	PICTURE 4
_	Segment 1	Г	• 1		Ι					
	A ES MENE				T		T			
					T		Ì			
٦		L	D .9	5	Т	Т				F.I.C.A.
٦		S	T.	S	Т	Т	\neg			P. B. L. N. T.
		S		,	1					A
	PRINT	B	S	5	T	T				
		1		_	T		T			
					1	1				
7			1	\neg	1	1				
┪	A	1	DS	₹₩	1	T	_			
┪		1		1	1	_	T			
٦		_	T	_	1					
_		1		1	T	1				
	C	1	† • <i>-</i>	_	1	T	_			
-	Segment 2 -	1	†		+	1	T	_		
-		-	1	-	1	\top	_			
_		t		+	+	-+				
	T C A	B	5.3	<	1	+	T			
	P 1 19 1TO 1	1"	013		1	\uparrow	_	_		
\exists		┢	1.		+	\top	_			
7		1	1		1	+	1			
7		P	X.1	В	T	\top				C.A.L.T.A.X.
٦		<u>'</u> -			T	\top		_		
		Г	1.		T	1	1			
_		1	1		1	1				
7					1	T				
	Segment 3 -	ļ —	b		1	1	1	_		
		1			T	7	1			
1	1	Γ	•	-	T	T	i			
		1			1	1	T			
-	C.A.L.T.A.X.	5		A	1	1	-			C. A. L. T. A. X. I. T., I.
		ĭ~	+2412	-+-	1	-	-			
		1			1	1	_			
		1			T	1	T			
7	CALTAXILT	R	RI	0	1	1				**
		۳	1		T		\neg			
-			+	\neg	1	\top	1			
٦		t	1-		+	+	-†			
		Г	T :	-	T	T	_			1 1
		1	· · · · ·		1	T	寸			

I REFERENCE SYMBOL I

1-33 12/6/63

OPERATION PARAMETERS

1-32 12/6/63

The following table lists the symbolic references listed in this example and comments as to their relationship in the segments.

Symbolic Reference	Comment
FICA	External global reference in segment 1 Internal global reference in segment 2
PRINT	Local to segment 1
Α	Local to segment 1
CALTAX	External global reference in segment 2 Internal global reference in segment 3
CALTAXIT	Local to segment 3

ASSEMBLER ACTION IN GLOBAL SYMBOLIC REFERENCES

If a symbolic reference in a segment is an internal global reference, a special flag appears in the symbol table. When the symbol table has been completed, the assembler scans the table and punches out internal global cards containing the names of the symbols and their assigned addresses within the segment. The loader uses this information when the segment is loaded into memory.

If the symbolic reference is an external global reference, the assembler performs the following actions:

- 1. Each instruction in which an external global symbol appears is flagged. The address of the instruction is calculated by assuming that the external global symbol itself has an absolute value of zero.
- 2. The assembler punches out external global cards containing the external global symbol names and the relative address(es) assigned to each.

RELOCATION ERRORS

When coding for a relocatable assembly, it is sometimes possible to inadvertently generate errors in the use of relocatable reference symbols in the Operation Parameters. When these errors are flagged by the assembler, it is important that the programmer understand the reason why.

Reference Symbols, when they are encountered by the assembler, are classified as being relocatable or absolute. Operation Parameters are most often constructed from Reference Symbols. The way that Reference Symbols are used in the construction of Operation Parameters determines whether the address in the Operation Parameter field is relocatable or absolute. When the assembler has made this determination, it can indicate the result to the loader by means of the reserved bits on the binary output.

All relocatable programs are assembled with their addresses starting at zero. When they are loaded into the machine at object time, their location in memory depends upon the current setting of the program counter. Since they will be relocated, the loader must be aware that the relocatable address will be modified (added to the base load address) and that absolute addresses must remain unaltered. For example, consider the following instructions assembled relative to zero.

REFERENCE SYMBOL					OPERATION							OPERATION PARAMETERS		
9_	DATA	NAM	E:	16	1,7	L E	VEL	20	s _{>N} C	22	U _S E	24	25	PICTURE
A_{\perp}		11.			L	D	,S						B	
					S	R	ر.	A				!	C , ,	
				1	S	T	ی						\mathcal{D}_{\cdot}	
					\mathcal{B}	R	U	l .					ε	
\mathfrak{B}_{-}		<u></u>			B	S	٠S						7	
\mathcal{D}_{\perp}					B	S	S	!					1	
ε_{\perp}		<u> </u>			L	D	D				i		2	
C,	_1 1	1		,	8	Q	U					ĺ	2	
		11]_	٠,					1			
			Transport or a]		,							
					1					1				

The binary output appears as follows:

Relative Location	REFERENCE SYMBOL	OPERATION	OPERATION PARAMETER
	DATA NAME	17 18 15 15 17 17 17	PICTURE 25
0		LDSI	(,4,),R,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
1		5. R.S. A.	(,2,),A,,,,,,,,
2		[S, T,S]	(,5,),R,,,,,,,
3		B RU L	(, b,), R, , , , , , , , , , , , , , , , ,
4		(reserved locations)	
5	<u></u>	(reserved locations)	
6		L'D.D.	(12) A

where R is a flag to the loader which indicates that the address must be relocated and A is a flag which indicates the address must remain unchanged. Therefore, if the program is loaded starting at 1501, it would appear in memory as follows:

Memory		.	1 + + -	
Location	المناصف المسلم المسلم	السلية أ		
	ا فا عالدالمالية بالأربية			
1501	and the second second	L.D.S		1 5 0 5
1502	and the transfer also the call of	S.R.S.A		[2 i
1503		5.7 5	i	11506
1504	k	B.R.V.		1507
1505	Landa de la companya	(reserved	location)	
1506	L	(reserved	location)	
1507	L	L.D.D.		2

If the program were loaded at location 1630, it would appear as follows:

.,	REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS	
Memory Location	9 DATA NAME 16	17 LEVEL SYN C 22 E 24	PICTURE 28	
1630		LDIS	1,6,3,4,	
1631		S R S A	2	
1632		ST.S	1,6,3,5,	
1633		BRU	1,6,3,6,	
1634		(reserved location)		
1635		(reserved location)		
1636		LDD	2	
	L.,	L		

When the assembler encounters a relocation error (some value which is meaningless), a flag is produced on the assembly listing. It is imperative that the programmer understands the assembly process of detecting errors so that these mistakes can be avoided or corrected.

By far, the vast majority of programs that are written contain addresses formed by connecting symbols and values with plus and minus signs only. In this environment, the programmer should check to discover whether the value of the address is absolute, relocatable, or a relocation error. The assembler also checks these values during assembly.

Procedure for expressions connected by plus and minus only

1. Examine each element of the expression. If it is a numeric value, remove it from consideration. If it is a symbol which represents an absolute value, remove it from consideration. If it is a symbol which represents a relocatable value, substitute the letter M (this value can be Moved at load time) in place of the symbol.

- 2. This will leave an expression in terms of one unknown, M.
- 3. Combine like terms in this expression
 - a. if the result is 0 or nothing, the address is absolute and cannot be moved.
 - b. if the result is M, the address is relocatable and must be moved.
 - c. if the result is anything else, the address is a relocatable error.

Examples

1. Absolute

.
.
.
SRSA
.
.

The expression for the address of the Shift instruction contains only one term, i.e., the absolute value 3. Because it is an absolute value, it is ignored. Since the result has produced nothing, the address is considered absolute.

Since the symbol ALTER represents an absolute value, it is ignored. The remaining procedure is the same as Example 1.

3.	Relocatable	
	•	
	•	
EENØ	BSS	1
	•	
	•	
	LDS	EENO

The symbol EENO represents a relocatable memory cell which will be moved at load time. The letter M is substituted for the symbol EENO. Since the result is M, the address is relocatable.

4.	Relocatable	2
	•	
	•	
GRØS	BSS	1
	•	
	•	
	LDS	GRØS+7

M is substituted for the symbol GROS and the 7 is ignored. Since the result is M, the address is relocatable.

5.		Absolute	
		•	
		•	
	BEGTAB ENDTAB	BSS BSS	99 1
		•	
		•	
		BCTR	ENDTAB - BEGTAB

The symbols ENDTAB and BEGTAB both represent relocatable elements. When M is substituted, the equation M-M=0 results. Since the equation has produced 0, the address is absolute. This example illustrates the fact that the difference between two relocatable symbolic references is absolute.

6. Relocatable Error ... BEGTAB BSS 99 ENDTAB BSS 1 ... BCTR ENDTAB + BEGTAB

The symbols ENDTAB and BEGTAB both represent relocatable elements. When M is substituted, the equation M+M=2M results. Since the result is not 0 or M, the address is flagged as a relocation error.

These examples cover some of the more common types of expressions that can appear in the Operation Parameters. On occasion, a programmer will wish to combine symbolic references by connecting them with the multiplication (*) and division (/) operators. In this environment, the rules for determining the nature of the Operation Parameters become slightly more complex. A description of this procedure may be found at the end of this section of the manual in the Supplement to Chapter IV, on page 1-151.

V. COMPUTER OPERATION CODES

The GE-425/435 instruction repertoire is listed on the following pages. A detailed description of the functions and operation of these instructions is found in the GE-425/435 Reference manual. In the listing which follows, all two-address instructions are indicated by (2A) in the left margin.

LISTING OF COMPUTER OPERATION CODES

	Data Transfer Instructions	Mnemonic	Octal
	Load Single	LDS	40
	Load Double	LDD	41
	Load Triple	LDT	42
	Load Quadruple	LDQ	43
(2A)	Move From First Memory	MFM	30
(2A)	Load Index	LDX	30
(2A)	Move From Immediate	MFI	31
(2A)	Load Index with Immediate	LXI	31
(2A)	Store Single	STS	44
	Store Double	STD	45
	Store Triple	STT	46
	Store Quadruple	STQ	47
(2A)	Move To First Address Field	MTA	32
(2A)	Store Index in Address Field	SXA	32
	Move	MOV	06
(2A)	Move on Index Control	MXC	06
	Move Counter	MCTR	

1-40 12/6/63 1-41 12/6/63

	Arithmetic Instructions	Mnemonic	Octal	Shift Instructions					
	Add Decimal Single	ADS	50	(See Tables 1 and 2 on page 1-54)	Mnemonic	Octal			
	Subtract Decimal Single	SDS	60	Shift Right Single Decimal	SRSD .	. 22			
	Add Decimal Double	ADD	51	Shift Right Single Decimal and Set	SRSDS	22			
	Subtract Decimal Double	SDD	61	Shift Right Single Alpha	SRSA	22			
	Add Decimal Triple	ADT	5 2	Shift Right Single Alpha and Set	SRSAS	22			
	Subtract Decimal Triple	SDT	62	Shift Right Double Decimal	SRDD	22			
	Add Decimal Quadruple	ADQ	5 3	Shift Right Double Decimal and Set	SRDDS	22			
	Subtract Decimal Quadruple	SDQ	63	Shift Right Double Alpha	SRDA	22			
(2A)	Add Immediate to Memory	AIM	33	Shift Right Double Alpha and Set	SRDAS	22			
(2A)	Add Immediate to Index	AIX	33						
` ,	Variable Length Multiply	VLM	27						
	Variable Length Divide	VLD	26						
	Add to Memory Single	AMS	54						
	Add to Memory Double	AMD	55						
	v				ons, bits 14 through 7 determine the length				
	Add to Memory Triple	AMT	56	* In all shift instructions, bits 14 through					
	Add to Memory Quadruple	AMQ	57	of the accumulator, the direction, an	of the accumulator, the direction, and the type of the shift.				
(2A)	Add Binary to Memory	ABM	34						
(2A)	Add Binary to Index	ABX	34						
(2A)	Subtract Binary from Memory	SBM	35						
(2A)	Subtract Binary from Index	SBX	35						

1-42

Shift Instructions (Continued)	Mnemonic	Octal	Shift Instructions (Continued)	Mnemonic	Octal
Shift Right Triple Decimal	SRTD	22	Shift Left Quadruple Alpha	SLQA	22
Shift Right Triple Decimal and Set	SRTDS	22	Shift Left Quadruple Alpha and Set	SLQAS	22
Shift Right Triple Alpha	SRTA	22	Rotate Right Single Decimal	RRSD	22
Shift Right Triple Alpha and Set	SRTAS	22	Rotate Right Single Decimal and Set	RRSDS	22
Shift Right Quadruple Decimal	SRQD	22	Rotate Right Single Alpha	RRSA	22
Shift Right Quadruple Decimal and Set	SRQDŠ	22	Rotate Right Single Alpha and Set	RRSAS	22
Shift Right Quadruple Alpha	SRQA	22	Rotate Right Double Decimal	RRDD	22
Shift Right Quadruple Alpha and Set	SRQAS	22	Rotate Right Double Decimal and Set	RRDDS	22
Shift Left Single Decimal	SLSD	22	Rotate Right Double Alpha	RRDA	22
Shift Left Single Decimal and Set	SLSDS	22	Rotate Right Double Alpha and Set	RRDAS	22
Shift Left Single Alpha	SLSA	22	Rotate Right Triple Decimal	RRTD	22
Shift Left Single Alpha and Set	SLSAS	22	Rotate Right Triple Decimal and Set	RRTDS	22
Shift Left Double Decimal	SLDD	22	Rotate Right Triple Alpha	RRTA	22
Shift Left Double Decimal and Set	SLDDS	22	Rotate Right Triple Alpha and Set	RRTAS	22
Shift Left Double Alpha	SLDA	22	Rotate Right Quadruple Decimal	RRQD	22
Shift Left Double Alpha and Set	SLDAS	22	Rotate Right Quadruple Decimal and Set	RRQDS	22
Shift Left Triple Decimal	SLTD	22	Rotate Right Quadruple Alpha	RRQA	22
Shift Left Triple Decimal and Set	SLTDS	22	Rotate Right Quadruple Alpha and Set	RRQAS	22
Shift Left Triple Alpha	SLTA	22	Rotate Left Single Decimal	RLSD	22
Shift Left Triple Alpha and Set	SLTAS	22	Rotate Left Single Decimal and Set	RLSDS	22
Shift Left Quadruple Decimal	SLQD	22	Rotate Left Single Alpha	RLSA	22
Shift Left Quadruple Decimal and Set	SLQDS	22	Rotate Left Single Alpha and Set	RLSAS	22

1-44 12/6/63 1-45 12/6/63

Shift Instructions (Continued)	Mnemonic	Octal		Compare Instructions	Mnemonic	Octal
Rotate Left Double Decimal	RLDD	22		Compare Alphanumeric Accumulator to Memory	CAA	03
Rotate Left Double Decimal and Set	RLDDS	22		Compare Decimal Accumulator to Memory	CDA	02
Rotate Left Double Alpha	RLDA	22	(2A)	Compare Memory to Immediate	CMI	01
Rotate Left Double Alpha and Set	RLDAS	22	(2A)	Compare Index to Immediate	CXI	01
Rotate Left Triple Decimal	RLTD	22	(2A)	Compare Second to First Memory	CMM	04
Rotate Left Triple Decimal and Set	RLTDS	22	(2A)	Compare Index to Memory	CXM	04
Rotate Left Triple Alpha	RLTA	22				
Rotate Left Triple Alpha and Set	RLTAS	22		Branch Instructions	Mnemonic	Octal
Rotate Left Quadruple Decimal	RLQD	22		Branch Unconditionally	BRU	10
Rotate Left Quadruple Decimal and Set	RLQDS	22	(2A)	Store Program Counter and Branch	SPB	17
Rotate Left Quadruple Alpha	RLQA	22	(2A)	Program Counter to Index and Branch	PXB	17
Rotate Left Quadruple Alpha and Set	RLQAS	22		Branch if Equal	BRE	13
Shift Right Single (Binary)	SRS	22		Branch if Less	BRL	14
Shift Right Single (Binary) Test	SRST	22		Branch if Greater	BRG	12
Shift Right Double (Binary)	SRD	22	(2A)	Branch on Count	BRC	16
Shift Right Double (Binary) Test	SRDT	22	(2A)	Branch on Index Count	BXC	16
Rotate Right Single (Binary)	RRS	22		Branch Counter	BCTR	
Rotate Right Single (Binary) Test	RRST	22		Branch if Zero	BRZ	15
Rotate Right Double (Binary)	RRD	22		Branch if Minus	BRM	11
Rotate Right Double (Binary) Test	RRDT	22				

1-46 12/6/63 1-47 12/6/63

	Logical Instructions	Mnemonic	Octal		Miscellaneous Instructions	Mnemonic	Octal
(2A)	AND to Memory	ANM	24	(2A)	General	GEN	07
(2A)	AND to Index	ANX	24		Halt	HLT	00
(2A)	OR Inclusive to Memory	RIM	23		Load Accumulator Location and length	LAL	3/3
(2A)	OR Inclusive to Index	RIX	23		Store Accumulator Location and length	SAL	3'7
(2A)	OR Exclusive to Memory	RXM	25		Explode	EXP	2)
(2A)	OR Exclusive to Index	RXX	25		Implode	IMP	21
	Edit	EDT	05		Low Bit Test	LBT	22
					Reset Accumulator Length Single	RALS	22
	Central Processor Instructions	Mnemonic	Octal		Reset Accumulator Length Double	RALD	22
(2A)	Central Processor Operation	СРО	67		Reset Accumulator Length Triple	RALT	22
(2A)	Set Status by ANDing	SSA	67		Reset Accumulator Length Quadruple	RALQ	22
(2A)	Set Status by ORing	SSO	67		Index	.X	60
(2A)	Set Status by Loading	SSL	67		Index Pointer	XP	01
(2A)	Request Status of Processor	RQSP	67		Index Link	ХL	02
					Index Indirect	X*	10
					Index Pointer Indirect	XP*	11
					Index Link Indirect	XL*	12
					Operand	O	CO
					Operand Pointer	OP	(1
					Operand Link	OL	02
					Indirect Address Word	IAW	00

1-48 12/6/63 1-49 12/6/63

Miscellaneous Instructions (Continued)	Mnemonic	Octal	Printer	Mnemonic	Command
Data Control Word Character Count	DCWC		Print in the Edited mode (data controls slewing)	PRE	30
Data Control Word Word Count	DCWW		Print in the Edited mode-slew Single line	PRES	31
List Pointer Word	LPW		Print in the Edited mode-slew Double line	PRED	32
			Paint in the Edited mode to Top of page	PRET	33
Input/Output Commands			Point in the Non-edited mode-slew no lines	PRN	10
All Input/Output Commands are Two-Address Instr	uctions.		Print in the Non-edited mode-slew Single line	PRNS	11
			Print in the Non-edited mode-slew Double line	PRND	12
Common to All Peripherals	Mnemonic	Command	Print in the Non-edited mode-slew to Top of page	PRNT	13
Request Status	RQS	00	Slew Printer Single line	SPRS	61
Reset Status	RSS	40	Slew Printer Double line	SPRD	62
			Slew Printer to Top of page	SPRT	63
Card Reader	Mnemonic	Command			
Read Card Decimal	RCD	02	Console Typewriter	Mnemonic	Command
Read Card Binary	RCB	01	Request Status of Typewriter	RQST	00
Read Card Mixed mode	RCM	03	Reset Status of Typewriter	RSST	40
			Type Input-Octal	TIO	01
Card Punch	Mnemonic	Command	Type Input-Alphanumeric	TIA	03
Punch Card Decimal	PCD	12	Type Output-Octal	TOO	11
Punch Card Binary	PCB	11	Type Output-Alphanumeric	TOA	13
Punch Card in Edited mode	PCE	13			

1-50 12/6/63 1-51 12/6/63

Paper Tape	Mnemonic	Command	Disc Storage Unit	Mnemonic	Command
Read Paper Tape	RPT	02	Seek/Write File	SWF	10
Punch Paper Tape	PPT	11	Seek/Write File and Release seek	SWFR	11
Punch Edited Tape-system mode	PET	31	Seek/Write File and Increment address	SWFI	12
Punch Single character mode Tape	PST	16	Seek/Write File and Verify	SWFV	13
Punch Double character mode Tape	· PDT	13	Seek/Read File	SRF	14
			Seek/Read File and Release Seek	SRFR	15
Magnetic Tape	Mnemonic	Command	Seek/Read File and Increment address	SRFI	16
Read Tape Binary	RTB	05	Seek/Compare	SCPR	17
Read Tape Decimal	RTD	04	Read Buffer	RB	24
Write Tape Binary	WTB	15	Read File Continuous and Release seek	RFCR	25
Write Tape Decimal	WTD	14	Write Buffer	WB	30
Write End of File	WEF	55	Write File Continuous and Release seek	WFCR	31
Backspace one Record	BSR	46	Accept Buffer Address	ABA	32
Backspace one File	BSF	47	Write File Continuous, Verify and release seek	WFCV	33
Set Density Low	SDL	61	Seek File	SF	34
Set Density High	SDH	60	Seek/Link	SLNK	35
Forward Space one Record	FSR	44	Load Buffer For Compare	LBFC	36
Forward Space one File	FSF	45	Move Data	MVDT	37
Rewind	RWD	70	Write File	WF	`50
Rewind and Standby	RWS	72	Write File and Release seek	WFR	51
Erase	ERS	54	Write File and Increment address	WFI	52
			Write File and Verify	WFV	53

1-52

12/6/63 1-53 12/6/63

<u>Disc Storage Unit</u> (Continued)	$\underline{\textbf{Mnemonic}}$	Command
Read File	\mathbf{RF}	54
Read File and Release seek	RFR	55
Read File and Increment address	RFI	56
Compare	CPR	57
Link	LNK	75

Document Handler	Mnemonic	Command
Read Document	RDOC	01
Feed Document	FDOC	41
Pocket Document	PDOC	43

Table 1. Shift Mnemonics For Characters

S	R	S	A	S
		SINGLE		
SHIFT	RIGHT	DOUBLE	ALPHA	SET ACCUM LENGTH
R	L	T TRIPLE	D	Δ
ROTATE	LEFT	Q QUADRUPLE	DECIMAL	NO ACCUMU- LATOR LENGTH CHANGE

Table 2. Shift Mnemonics For Binary

S	R	S	Т
SHIFT		SINGLE	TEST
R ROTATE	RIGHT	D DCUBLE	△ NO TEST

Using these two tables, the programmer can form any of the Shift Mnemonic Operation Codes.

COMPLEMENTARY INSTRUCTION WORDS

In order to accomplish their function, certain GE-423/435 instructions require information in addition to that contained within the instruction word. The words supplying this additional data are called complementary instruction words. Such words must be provided in the proper format when using the instructions requiring them.

The basic assembly language includes the mnemonic operation codes designed to form the complementary instruction words. These mnemonic codes are described on the following pages.

BRANCH COUNTER	DOWN
	BCTR

Function

BCTR enables the programmer to specify the count required by the Branch on Index Count (BXC) and the Branch on Count (BRC).

Format

BCTR may contain

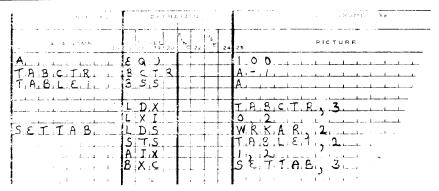
- A symbol or blanks as the Reference Symbol
- BCTR as the operation
- An expression in the Operation Parameters for the count of the branch counter.

·..

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. A branch counter is formed by using the value of the expression in the Operation Parameter as the count.

Example



This coding causes 100 words of data to be moved from a work area in memory, WRKAR, to a table in memory, TABLE 1. The SETTAB coding is executed once before the BXC command is encountered and 99 additional times on the BXC command. The BCTR controls the associated BXC command.

Notes

- The Branch Counter specifies the number of times that the associated BXC or BRC will branch before falling through to the next instruction.
- 2. The largest value which a Branch Counter can contain is 511.

 Therefore, if the value specified for a Branch Counter exceeds 511. it will be truncated modulo 512.

- 3. The value that results from calculating the expression in the Operation Parameters, must be absolute (See Relocation Errors, page 1-35.)
- 4. The Operation Parameters may not contain external global symbols. (See Local and Global Symbols, page 1-31.)

MOVE COUNTER

MCTR

Function

MCTR enables the programmer to specify the count required by the Move (MOV) and the Move or Index Control (MXC) instructions.

Format

MCTR may contain

- 1. A symbol or blanks as the Reference Symbol
- 2. MCTR as the operation
- 3. An expression in the Operation Parameters for the address.
- 4. An expression in the Operation Parameters for the count of the Move Counter.

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. A Move Counter is formed using the first expression in the Operation Parameters as the Address Filed, and the value of the second expression as the count of the number of words to be moved.

Example

REFERENCE SYMBOL	٠	, OF	ERA	1. T10	N		OPERATION PARAMETERS	
DATA NAME	16 1	1 L T V E 1	20	\$ ZC	22	U _S	24	PICTURE
A	[8	9 0						1,0,0, , , , , , , ,
TABILIELL	1	5 5 5						A
T.A.B.CTR.		ICI	R					W.R.K.A.R.
		1.	1					
		1.	ļ					
		1						
S.E.T.I.A.B	N	√			_			T.A.B.L. E.
	1	L	!	Ø	P			T.A. E.C. T.R.

This coding causes 100 words to be moved from a work area in memory, WRKAR, to a table in memory, TABLE1. The MCTR supplies the address from which data is moved and controls the number of words moved.

Notes

- 1. The largest number of words which can be moved using a Move Counter is 512. Therefore, if the value specified for a Move Counter exceeds 512, it will be truncated modulo 512.
- 2. The value that results from calculating the expression for the count must be absolute. (See Relocatable Errors, page 1-35.)
- 3. The expression for the address may contain external global symbols, the expression for the count may not. (See Local and Global Symbols, page 1-31.)

LIST POINTER WORD	LPW

Function

LPW enables the programmer to specify the list pointer words which indicate the location and size of associated data control lists.

Format

LPW may contain

A symbol or blanks as the Reference Symbol

- LPW as the Operation
- An expression in the Operation Parameters for the address
- An expression in the Operation Parameters for the count of the List Pointer Word.

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. A List Pointer Word is formed using the first expression in the Operation Parameters as the Address Field of the LPW and the value of the second expression as the count of the number of words in the associated data control list.

Example

REFERENCE SYMBOL		OPI	ERA	TIC	N			İ	OPERATION PARAMETERS
DATA NAME 16	,,	LEVEL		s > 2 C	22	U _S	24	25	PICTURE
	L	₽W						بط	1.5.1.1.,.7

The data control list is at symbolic location LIST1 and contains seven words. Note that the two expressions in the Operation Parameters field are separated by a comma.

- A single data control list may not contain more than 512 words.
 If the value specified for the count of an LPW exceeds 512, it will be truncated modulo 512.
- 2. The value that results from calculating the expression for the count must be absolute (see Relocation Errors, page 1-35).
- 3. The expression for the address may contain external global symbols; the expression for the count may not (see Local and Global Symbols, page 1-31).
- 4. The count in an LPW should be the count of the DCW's +1.

DATA CONTROL WORDS

DCWC-DCWW

Function

DCWC (Data Control Word-Character Count) and DCWW (Data Control Word-Word Count) enable the programmer to construct input and output data control lists by indicating the individual data control words which comprise such lists.

Format

DCWC and DCWW may contain

- A symbol or blanks as the Reference Symbol
- DCWC or DCWW as the Operation
- An expression in the Operation Parameters for the address
- An expression in the Operation Parameters for the count of the data control word.

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. A data control word (DCW) is formed using the first expression in the Operation Parameters as the Address Field of the DCW and the value of the second expression as the DCW character count. (The assembler multiplies the DCWW count by four before using it as a character count.)

Example

١	HELEGENET TOMBOL	DEFRATION	OPENALION MAMANITERS
İ	4 DAYANAME		28 PICTURY 40 ACCUPY II.
		L P.W.	L. 1. 5. T. 1. , 7. 1
	LitsTil	o c ww	T.N. P. V. T
1		DCWC	Ting Proctof this factor of the control of the Cont
ł		DIC W (I.N.P.O.T. * . 8
1		D C WW	INPUT-15., 1.0. L. I.N.P. U.T.+1.5. L. I.S. T. I., 7. L. I.S. T. I., 7. L. I.S. T. I. P. O. I.N.T. E. R. R. C.S. T. O. R. E. M. O. B. D
1			ا الاستاملين المنظم المنظم المستطيعة والمنظمية الله المنظمية المنظم المنظم المنظم المنظم المنظم المنظم المنظم ا المنظم المنظم

Note that the expressions in the Operation Parameters field are separated by commas.

Notes

- The DCWC contains a character count while the count in the DCWW is a word count.
- 2. A single DCW cannot control more than 512 characters. If the value specified for a DCW count exceeds 512, it will be truncated modulo 512.
- 3. The value that results from calculating the expression for the count must be absolute (see Relocation Errors, page 1-35).
- 4. The expression for the address may contain external global symbols; the expression for the count may not (see Local and Global Symbols, page 1-31).

ADDRESS MODIFICATION SEQUENCE WORD

Function

X (Index), XP (Index Pointer), and XL (Index Link) enable the programmer to specify Address Modification Sequence (AMS) words and to indicate their class. (Address Modification is discussed in detail in the GE-425/435 Reference Manual.)

Format

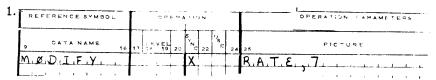
- X, XP, and XL may contain
- A symbol or blanks as the Reference Symbol
- X, XP, or XL as the operation
- An expression in the Operation Parameters for the address.
- If desired an expression in the Operation Parameters for the address control of the AMS word.

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. An AMS word is formed using the first expression in the operations Parameters as the Address Field of the AMS word. If there is a second expression, it is used as the Address Control field of the AMS word. The class is established by the mnemonic operation code employed by the programmer.

1-61

Examples



Example 1 forms an AMS Index whose address is RATE and whose Address Control Field specifies continuation MODIFY is defined as the location of the AMS word.



Example 2 forms an AMS Index Pointer whose address is RATE.



Example 3 forms an AMS Index Link whose address is MODIFY.



Example 4 forms the same AMS words as example 1, 2, and 3; however, the Indirect Address Indicator is set forth in each of the AMS words. Refer to the GE-425/435 Reference Manual.

Notes

- For documentation purposes, these instructions should be left-justified in the second half of the operation columns of the coding form.
- 2. If any other value besides 7 is specified for the Address Control Field, continuation will not take place. Values greater than 7 are truncated modulo-8. Such values will cause a flag to appear on the appropriate line in the assembly listing.

- 3. The value that results from calculating the expression for the address control must be absolute. (See Relocation Errors, page 1-35.)
- 4. The expression for the address may contain external global symbols; the expression for the address control may not. (See Local and Global Symbols, page 1-31.)

SECOND ADDRESS SEQUENCE WORDS

Function

O (Operand), OP (Operand Pointer) and OL (Operand Link) enable the programmer to specify Second Address Sequence (SAS) words and to indicate their class.

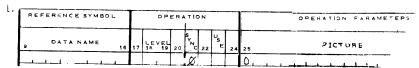
Format

- O, OP, and OL may contain
- A symbol or blanks as the Reference Symbol
- O, OP, or OL as the operation
- An expression in the Operation Parameters for the address of the SAS word.

Assembly Action

The Reference Symbol, if any, is defined as the current value of the location counter. An SAS word is formed using the expression in the operation Parameters as the Address Field of the SAS word. The class is extablished by the mnemonic operation code employed by the programmer.

Examples



Example 1 forms an SAS operand.

2.	REFERENCE SYMBOL		OPE	RATIO	N			OPERATION PARAME	TERS
	DATA NAME 16	17 LE	VEL 19 2	S YN	22	U _S E	24	PICTURE	_
	A.D.R.2			0	₽			R.A.T.E.	

Example 2 forms an SAS operand Pointer whose address is RATE. ADR2 is defined as the location of the SAS word.



Example 3 forms an SAS operand Link whose address is ADR2.

Notes

- 1. For documentation purposes, the mnemonics for these instructions should be left-justified in the second half of the operation.
- 2. The value that results from calculating the expression for the address control must be absolute (see Relocation Errors, page 1-35).
- 3. The expression for the address may contain external global symbols; the expression for the address control may not (see Local and Global Symbols, page 1-31).

INDIRECT ADDRESS WORDS IAW

$\boldsymbol{F}unction$

IAW enables the programmer to specify Indirect Address Words.

Format

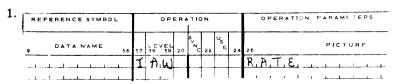
IAW may contain

- A symbol or blanks as the Reference Symbol
- IAW as the operation
- An expression in the Operation Parameters for the address.
- An expression in the Operation Parameters for the address control of the IAW.

Assembly Action

If there is a Reference Symbol, it is defined as the current value of the location counter. An Indirect Address Word is formed using the first expression in the Operation Parameters as the Address Field of the IAW. If there is a second expression, it is used as the Address Control Field of the IAW.

Examples



Example 1 forms an Indirect Address Word whose address is RATE.



Example 2 forms an Indirect Address Word whose address is RATE and whose Address Control Field specifies continuation.

- 1. If the value specified for the Address Control Field is other than 7, continuation will not take place. Values greater than 7 are truncated modulo-8. Such values will cause a flag to appear in the appropriate line on the assembly listing.
- 2. The value that results from calculating the expression for the address control must be absolute (see Relocations Errors, page 1-35).
- 3. The expression for the address may contain external global symbols: the expression for the address control may not (see Local and Global Symbols 1-31).

INPUT/OUTPUT MNEMONIC OPERATION FORMATS

The Basic Assembly Language separates the input/output operation codes into two broad classes according to the information which must be supplied in the Operation Parameters. The first class consists of those instructions which involve channel, device and Address Control. The second class consists of those instructions which involve only the Address Control. (Refer to Basic Input/Output System manual.)

CLASS 1 INPU	T/	OUTPUT	MNEMONICS

Function

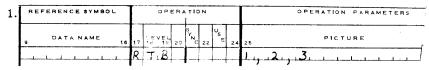
Class 1 contains all instructions which deal with input/output devices other than the console typewriter.

Format

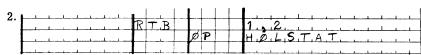
A Class 1 instruction may contain

- A Symbol or blanks as the Reference Symbol
- A Class 1 mnemonic operation code as the operation
- An expression in the Operation Parameters for the channel
- An expression in the Operation Parameters for the device
- An expression in the Operation Parameters for the address control of the Class 1 instruction.

Examples



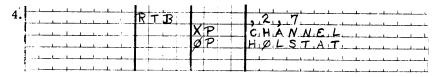
Example 1 shows a Read Tape Binary command to Channel 1, magnetic tape unit 2. Status is stored in fixed index register 3.



Example 2 shows a Read Tape Binary command to Channel 1, magnetic tape unit 2. The operand pointer sends status to HOLSTAT.

'	REFERENCE SYM	во ц		OP	ERA	T 10	N			OPERATION PARAMETERS
٠	DATA NAME	16	17	LEVEL 18 19	20	a YZC	22	U S E	24	PICTURE 25
L			R	TE						1
						X	P			DEVICE
						Ø	P			H. C. L. S. T. A.T.
						ľ				

Example 3 shows a Read Tape Binary command to Channel 1. The index pointer establishes a device code. The operand pointer sends status to HOLSTAT.



Example 4 shows a Read Tape Binary command to magnetic tape unit 2. The index pointer establishes the channel. The operand pointer sends status to HOLSTAT.



Example 5 shows a Read Tape Binary command. The first index pointer establishes the channel; the second the device code. The operand pointer sends status to HOLSTAT.

- 1. Expressions in the Operation Parameters must be separated by commas. The number of commas which precede a particular expression establish its function. Thus, an expression for the device must always be preceded by one comma and an expression for the address control by two commas.
- 2. The value that results from calculating the expression for the channel, device and address control must be absolute (see Relocation Errors, page 1-35).

3. External global symbols may not be used in the Operation Parameters (Local and Global Symbols, page 1-31).

CL	ASS	2

INPUT/OUTPUT MNEMONICS

Function

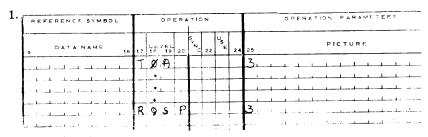
Class 2 input/output mnemonics contains all instructions which deal with the console typewriter and the central processor channel.

Format

A Class 2 instruction may contain:

- A symbol or blanks as the Reference Symbol
- A Class 2 mnemonic operation code as the Operation
- An expression in the Operation Parameters for the address control of the Class 2 instruction.

Examples



Example 1 shows two Class 2 commands, a Type Output Alphanumeric and a Request Status of Processor, which sends status to fixed index 3.

2.	F	REI	E	RE	N	E	5	YM	В	Ļ		Г		OP	ERA	TIC	7			OPERATION PARAMETERS
	9			D A	Τ,	. ,	- A	ME			16	17	L E	√EL 19	20	5 × × C	2.2	U _S E	24	PICTURE 25
				_		.1						I.	Ø	A						
				_	_	1						L_	Ĺ		-	Ø	P	<u> </u>		H. Ø. L. S. T. A. T.
	L				L	_		_				L_	•	1		_		<u> </u>		_!
	L.		<u></u>		L_			٠				ļ	١.	1		ļ	<u> </u>	<u>i —</u>	-	
	L	٠			<u>.</u> .	,	_			_		L		٠		ļ.,	į	+	-	
		,			1							IR	Q	S	P	1.			! _	
		_			_											10	P	-		$H_{\mathcal{B},L}$, $S_{1}T_{1}A_{1}T_{1}$

Example 2 shows two Class 2 commands, a Type Output Alphanumeric and a Request Status of Processor. The Operand Pointers send status to HOLSTAT.

- The value that results from calculating the expression for the address control must be absolute (see Relocation Errors, page 1-35).
- 2. External global symbols may not be used in the Operation Parameters. (see Local and Global Symbols, page 1-31).

VI. PSEUDO-OPERATIONS

Pseudo-operations are mnemonic codes which have the same general form as computer operations; however, they perform their function during the running of the assembly and are never executed by the computer as machine instructions. These instructions are used to control and instruct the assembly of a program.

Pseudo-operations may be used to communicate with the various portions of the GE-425/435 software system, to indicate constants, to influence memory assignments, to annotate the assembly listing etc.

Available pseudo-operations are listed and explained on the following pages.

SYSTEM ORIENTED PSEUDO-OPERATIONS	MNEMONIC
Segment Name	SGMT
Define Internal Global symbols	DIG
Define External Global symbols	DXG
Define Global Reference Remotely	DGRR
Define Global Reference Ends	DGRE
Define Global Reference	DGR
Equals Global symbol	EQUG
Call segment at load time	CALL
Include Library segment	INCL
Include Symbolic segment	INCS
Block Started by Symbol in the Loader area	BSSL
Block Preceded by Symbol in the Loader area	BPSL
Last Symbol in Block in the Loader area	LSBL
Accumulator Reference Point in the Loader area	ARPL

MEMORY ALLOCATION PSEUDO-OPERATIONS	MNEMONIC
Block Started by Symbol	BSS
Block Preceded by Symbol	BPS
Last Symbol in Block	LSB
Accumulator Reference Point	ARP
Accumulator Reference Point in the Loader area	ARPL
Accumulator	ACUM
Fill word	FILL
CONSTANT PRODUCING PSEUDO-OPERATIONS	MNEMONIC
Decimal Constant Single word	DECS
Decimal Constant Double word	DECD
Decimal Constant Triple word	DECT
Decimal Constant Quadruple word	DECQ
Alphanumeric constant	AN
Last Symbol Alphanumeric	LSAN
Octal Constant Single word	OCTS
Octal Constant Double word	OCTD
Octal Constant Triple word	OCTT
Octal Constant Quadruple word	OCTQ

ASSEMBLY OUTPUT CONTROL PSEUDO-OPERATIONS	MNEMONIC
Title	TTL
Eject page	EJT
Identify binary output	IDEN
Full binary cards	FULL
Symbolic Analyzer	SYAL
MISCELLANEOUS PSEUDO-OPERATIONS	MNEMONIC
Onigin	OBC

MISCELLANEOUS PSEUDO-OPERATIONS	MNEMONIC
Origin	ORG
Origin Octal	ORGO
Equals	EQU
Equals Octal	EQUO
Prefix	PRFX
Transfer Control Card	TCD
End of program	END

SYSTEM ORIENTED PSEUDO-OPERATIONS

SEGMENT NAME	CONTR
DEGMENT NAME.	SGM T
	20111

Function

All segments must be given a unique name to be punched on the output program header card. (See Section III, Assembly Functions.)

SGMT allows the programmer to name his segments (program). In addition, SGMT allows the programmer to specify an absolute origin for the segment, thus giving him the ability to assemble in the absolute mode.

Format

SGMT may contain

- The segment name as the Reference Symbol
- SGMT as the operation
- An expression or blanks in the Operation Parameters.

Assembler Action

When the assembler recognizes the SGMT pseudo-operation, it isolates the segment name and retains it for the program header card.

The Operation Parameters can contain three types of entries:

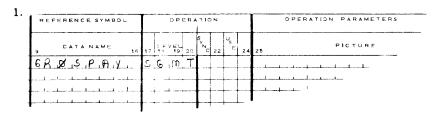
- 1. Blanks: Blanks in the Operation Parameters indicate the relocatable mode of assembly. All instructions are assembled relative to zero; the assembly listing is relative to zero; relocatable errors are flagged.
- 2. An expression containing one external global symbol: In addition to the general restrictions placed on external global symbols on page 1-79, the external global must be either preceded by a plus sign, or the plus sign must be implied. The condition, then indicates the relocatable mode of assembly.
- 3. An absolute decimal value: Because the absolute value is by definition the origin of the segment (program), its presence indicates the absolute mode of assembly. All instructions are listed relative to the starting origin. Relocatable errors are flagged for warning purposes only, since the program can be moved by changing the starting address on the origin.

Restriction

SGMT must be the first card of every segment or program that is presented for assembly. In addition, if the segment is to be placed in the library, it must be given a unique name. This will insure that both the assembler and librarian will be able to locate the segment at some future time.

12/6/63

Example



The programmer indicates that the name of the segment is GROSPAY. Further, since the Operation Parameter field is blank, a relocatable assembly is indicated.



The programmer indicates that the name of the segment name is FISCAL. Since the Operation Parameter field contains a symbolic reference (by definition, an external global symbol), a relocatable assembly is indicated.



The programmer indicates that the segment name is TIMCAL. Since the Operation Parameter field contains a decimal value, an absolute assembly is indicated.

DEFINE INTERNAL GLOBAL SYMBOL	DIG

Function

DIG indicates those symbolic global references which are defined within

the segment being assembled. The names of these references are required at load time to insure that separately compiled segments are properly linked.

Format

DIG must contain

- DIG as the Operation
- Internal global references that are listed in the Operation Parameters. These symbolic references must be eight characters or less. They must be separated by a comma.
- All internal global symbols listed must also be found in the Reference Symbol field.

Assembler Action

When the assembler encounters a DIG, the Operation Parameters are scanned and each symbolic reference is stored on the symbol table with a unique flag that defines it as an internal global symbol.

Example

REFERENCE SYMBOL	OPER	TION	OPERATION PARAMETERS		
DATA NAME 16	17 (P 19 20	S N C 22 E 24	FICTURF 25		
· L - L - L - L - L - L - L - L - L - L			<u> </u>		
	DIG		F.I.C.A., 5.A.L.A.R.Y.		
+ + + + + + + + + + + + + + + + + + + +	DIG		L.N.S.U.R.		
· · · · · · · · · · · · · · · · · · ·					
F.I.C.A.	L S.B		2.		
SALARY	LSB				
I N.S.U.R.	L.S.B.		1		
			L		

By using DIG, the three symbols, FICA, SALARY, and INSUR, are set forth as internal global symbols. They will be entered on the symbol table and marked as such. At the end of Pass I of the assembler, each internal global symbol together with its value and type (absolute or

relocatable) is punched into an internal global card so that the loader can effect the proper linkage among segments at load time.

Note

DIG should appear near the beginning of the program deck with the other pseudo-operations that effect linkage. When DIG is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

DEFINE EXTERNAL GLOBAL SYMBOLS	DXG

Function

DXG must contain

• DXG as the operation

• A list of external global symbolic references that will be found within the segment in the Operation Parameters. These symbolic references must be eight characters or less. They are separated by a comma.

Assembler Action

When the assembler encounters a DXG, the Operation Parameters are scanned and each symbolic reference is stored on the symbol table with a unique flag that defines it as an external global symbol.

1 - 77

REFERENCE SYMBOL	OPER	ATION	" " GRATION PARAMETERS				
DATA NAME	6 17 15 7 EL	S U S E 24	PICTURE.				
		1					
1 4 4 4 4 4 .		·	بالمستقل المستقد المست				
1 4 .1 (.4	- National III	1					
	D X G	1	B. W. D. E. D. T. I. M. E.				
. i	DXG	1	P.R.T. U.H. & C.K.				
1 . 1 . 4							
المنتبك المناب المناب			kaj la skoj draj kajaka i kojaka kajaka kajaka kajaka kajaka				
i	P.X.B.		B. &. N. D. E. D. J. B.				
12 dk(1) (1		l					
**** *	-	I					
_ t 1	PXB.	f	T.1.M.E. 6				
فللسب فالتدلسك	1 A . U .	1	Tilimie, ja				
	_	1					
	1	1					
and a state of the	PXB		P.R.T.C.H.E.C.K., 6				
	1.	I					
	1	i .	I				

The three symbols, BONDED, TIME, and PRTCHECK, are declared to be External Global Symbols. They will be entered on the symbol table and marked as such. At the termination of Pass II of the assembler, each external global symbol will appear in the binary output. Further, every location within the segment that requires the external global for completeness will also be indicated.

Notes

1. All symbols preceded by a lozenge are assumed to be symbols which are communicating with the various portions of the GE-425/435 software system. When the assembler encounters such references in the Operation Parameters, it automatically classifies them as external global symbols. For example, when a programmer wishes to communicate with the Basic Input/Output System such references as □ CARD, □ TAPE, □ TYPE, □ DEMND, etc., are all classified as external global symbols. They need not be declared as such by a DXG.

2. DXG should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When DXG is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

Restrictions

The following restrictions must be observed when using external global symbols:

- 1. The external global symbol may be preceded by either a plus (+) or a minus (-) sign; no other operators are allowed. If the external global symbol is the first term in the Operation Farameters and is not preceded by a sign is implied.
- 2. The external global symbol may be followed by a plus (+) or a minus (-) sign, or in the case of the last term of an expression by a blank or a comma (,). No other operations are allowed.

Thus, external symbols may not be used as factors in either multiplication or division. The assembly listing will flag any violations of these rules.

DEFINE GLOBAL REFERENCE REMOTELY DGRR

Function

The DGRR pseudo-operation helps to reduce the number of global symbols required for communication between segments.

One or more of the segments of a program may contain "chains" * of data which must be referenced by other segments, for example, the individual files of a working storage area. Since, in most cases, segments

* "Chains" may be defined as related records in a file, which each contain similar elements of data. For example a Personnel File

EMPNO	1	EMPNAM	- 1	EMGROPAY			- 1
111111111		THE INVINI	- 1	EMGROPAI	:	 	. [
						 	- 1

deal with the individual elements of the chain rather than the chain as a whole, the segments must be able to reference each element individually, as well as the chain itself.

The normal method of obtaining data from another segment requires that an external global symbol be assigned to each element of such data. However there is a limit to the number of external global symbols which the loader and the assembler can handle efficiently. Assigning a global symbol to each element in a chain of elements tends to produce a large number of external global symbols.

The pseudo-operation DGRR helps to reduce the number of external global symbols required for the efficient operation of a program.

Consider the following example:

REFERENCE SYMBOL		O F	ER	L T FO	N		l		OPERATION PARAMETERS								
DATA NAME 16	17	LEVE 18 19	20	S _N C	22	s E	24	26					PI	ст	UF	E	
INREC	ם	GR	R	Γ							,		,	_			
EMPNØ,	В	SIS	T	Ī				۵.									
E.M. P.N.A.M.	R	SS		Ι_				6.	,			i de la composición della comp					 _
EMGROPAY	B	5.5	Ĺ					3.									
		•		I							1						-
											,		1	,			

The DGRR assigns the global symbol INREC to the chain of data which follows. Further, it instructs the assembler to define the elements of the chain, EMPNO, EMPNAM and EMGROPAY, as INREC+0, INREC+2 and INREC+8, respectively. Any time the assembler recognizes a reference to an element from the DGRR, it converts that reference to the global symbol INREC with the proper increment.

Format

DGRR must contain:

- A symbol as the Reference Symbol
- DGRR as the Operation.

Assembler Action

When the assembler encounters a DGRR, it realizes that it is dealing with a storage allocating section which will be defined in one of the other segments (see DGR). This implies that no actual storage allocation will take place in the segment that is currently being assembled.

The Reference Symbol in the DGRR is treated as an external global symbol. A new location counter is initiated so that increments from the base external global can be calculated. As Reference Symbols are encountered after the DGRR, they are placed in the symbol table along with their increment and a marker which points at the base external global. This process continues until one of the following occurs:

- 1. the assembler encounters another DGRR which reinitiates the entire process with a new external global, or
- 2. the assembler encounters a DGRE (Define Global Reference Ends) which terminates the reference area, or
- 3. the assembler encounters an END or TCD which terminates the assembly, or
- 4. the assembler encounters a machine operation or illegal pseudo-operation. Legal pseudo-operations are: BSS, LSB, AN, LSAN, OCT (S, D, T, Q), DEC (S, D, T, Q), BPS, TTL, and EJT.

Example

REFERENCE SYMBOL	OPERA	TION	OPERATION PARAMETERS					
DATA NAME	17 16 19 20	S Y N C 22 E 24	PICTURE 25					
	• 1							
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		1						
		+++++						
w. <u>s</u>	DGRR							
E.E.N.,8	8 5 5		2					
E.M.P.N.A.M.E.	BSS		b					
E.M.P.B.Y.	BSS		2					
	DGRC		بالمستلم بالمستان المستلم المستان المس					
<u> </u>	LDD.		1. + Y & P.M 3					

The DGRR indicates that the subsequent block reservation is not defined within the segment; therefore, no storage is to be allocated. The symbolic reference WS is treated as a basic external global symbol. The symbols: EENO, EMPNAME, and EMPAY, are converted respectively to WS+0, WS+2 and WS+8. When the instruction at location A is converted, the assembler communicates the following information to the loader:

- 1. An external global symbol (WS) appears at this point.
- 2. The value 9 must be added to the value WS since EMPAY+1 equals (WS+8)+1 or WS+9.

Note

DGRR should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When DGRR is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

DEFINE GLOBAL REFERENCE ENDS

DGRE

Function

When a global reference is defined remotely, the programmer must indicate when the definition has STOPPED, i.e., when to resume normal assembly and begin stepping the location counter. DGRE performs this function.

Format

DGRE must contain

- A symbol or blanks in the Reference Symbol
- DGRE as the Operation

Assembler Action

When the assembler encounters a DGRE, it ceases to operate in a remote mode and begins the normal assembly process.

Example

REFERENCE SYMBOL	OPERA	T10N		OPERATION PARAMETE					
9 DATA NAME 16	17 18 19 20	S U _S C 22 E 24	25	PICTURE					
	•								
للل في علي السابطال فالداف الفاد .									
<u> </u>	BKU		L						
<u>w.s.</u>	D. S. R. R.								
	B 5 5		`						
	B 5.5		1						
¥.A. 9	8 5,5		1 \						
	D GIR E								
E.	s Kid		i .						
Panik			8						

When the assembler encourters the DGRR, it begins to process subsequent instructions actor and to the manner described under the DGRR pseudo operation. When NGRE is encountered, normal assembly is resumed. If the reference symbol A was assigned to location 1000, reference symbol B is assigned to contion 1001.

Note

When INFE is used, it is always used with a DGRR. Both instructions should be placed near the beginning of the program deck with other pseudo-operations that effect segment linkage. In this way, the coding will contain centralized documentation which describes the interaction between segments.

DEFINE GLOBAL REFERENCE	DGR

Function

One segment in a program must insure that storage is allocated for data that must be available to other segments. The DGR pseudo-operation is used to indicate the beginning of common storage in this segment.

Format

DGR must contain

- A symbol in the Reference Symbol
- DGR as the Operation

Assembler Action

When the assembler encounters a DGR, it automatically classifies the Reference Symbol as an internal global symbol. The assembly then continues in its normal way; subsequent cards contain information about storage which must be allocated within the current segment.

Example

PERINING E MEO.	OFERA	TION		OPERATION PARAMETERS
Section Assessed	17 U 19 2'	E U ₃	2 K	PECTUR
5.M. 8 1	BSS		2	5
				to the state of the same
			ä	

Then the assumbler encounters the DGR, the symbol WS is defined as internal global symbol. The assembler then proceeds in the normal canner, making the following address assignments:

Reference Symbol	Address Assignment (Decimal Notation)
A	1000
ws	1001
EENO	1001
EMPNAME	1003
EMPAY	1009
В	1011

Note

DGR should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When it is used this way, the coding will contain centralized documentation which describes interaction between segments.

EQUALS GLOBAL SYMBOL

EQUG

Function

EQUG equates a local symbol to an external global symbol.

Format

EQUG must contain

- A local symbol as the Reference Symbol.
- EQUG as the Operation
- An expression containing one external global symbol in the Operation Parameters.

Assembler Action

The assembler enters the local symbol in the Symbol Table with a flag indicating that all references to that symbol are actually referencing the external global symbol to which it has been equated.

Example

REFERENCE SYMBOL		o P	ER,	ATIO	N							OPE	R A	TIO	N	PAI	MAS	ETE	Rs
DATA NAME 16	,,	LEVEL 18 19	20	s _{YN} _C	22	U _S	24	25						PI	C T I	JRE			
	-			ļ.,					L	ı		.	L	1	1	1			1
P.R.I.I.N.T.	ε	<u> </u>	G					S	Y_	, ,	F	ıR.	I	.N	T		4l	1	l
r.L.M.E.C.A.L.	€	٠ ١ ١	G					G	٤	١N	J	I	M	ـــــــــــــــــــــــــــــــــــــ		-1	ر ا 5		 1
. Ø Ø P	Ĺ	Dis						T	I	M	ξ.	 	Α	1 1 1	+	; <u>2</u>		1	2
	Ρ	x.B.						Ρ	R		N	I	,	6	1 1	<u>. </u>	L1	+	
											<u> </u>			1					_

Assume that two different programmers were writing separate segments of a production program, the following situation might occur. One programmer might define the symbol for his time calculation routine as TIMECAL. The other programmer might refer to the same routine in his segment as GENTIME-15. Similar referencing might occur with PRINT and SYSPRINT. To equate the two sets reference symbols, the pseudo-operation EQUG is used as shown.

The first EQUG instructs the assembler to substitute the external global symbol SYSPRINT for the symbol PRINT. The second EQUG instructs the assembler to substitute GENTIME-15 for TIMECAL, i.e., the instruction at symbolic reference LOOP will be treated as if it were written:

LOOP I.DS GENTIME-15+20, 2

which results in

LOOP LDS GENTIME+5, 2

where GENTIME is now an external global symbol.

Note

EQUG should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When it is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

Restriction

The following restrictions must be observed when equating a local symbol to an external global symbol.

- 1. The external global symbol may be preceded by a plus (+) sign; no other operators are allowed. If the external global symbol is the first term in the Operation Parameters and is not preceded by a sign, the plus sign is implied.
- 2. The external global symbol may be followed by a plus (+) or a minus (-) sign, or in the case of the last term of an expression, by a blank or a comma (,). No other operations are allowed.

CALL SEGMENT AT LOAD TIME

CALL

Function

Any given segment may contain a CALL pseudo-operation. The CALL pseudo-operation indicates that the segments named in the Operation Parameters are needed for the successful running of the given segment. In contrast to INCL, CALL indicates that the required segments are requested from a library tape and loaded by the loader at execution time.

Format

CALL must contain

- CALL as the Operation
- In the Operation Parameters, the names of the segments which the loader is to provide. These names are separated by commas.

ASSEMBLER ACTION

When the assembler recognizes a CALL, it extracts the segment names from the Operation Parameters and places them in the program header card.

Example

REFF	HCE	SYMBO			0	PER	ATI	0 N			Γ	OPERATION PARAMETERS	
9	·/ TA	AME	• 6	,_	ĻEV	EL 19 20	SYZ	22	U _S	24	25	PICTURF 40	OCCURS 41 4
	1	1 1.=	L		_ :		Γ.	L					
1	1.		ţ.	D.,	X .(,	╁	<u> </u>	Ļ_		I.	L.M.E.T.P J., 26	<u> </u>
	1		i	١,	. •	🛊	\perp	-	<u> </u>		<u> </u>		ļ
1	4	. 1				+	+	-	├	-	Ͱ		
		* 1. 4	4	ī	N.		\dagger	-	-		Í	ØTAPE, 1 ØCARD, PR	1.N.T.
		4	L	T = '	ΑЦ			+			I	INECARD, B. W.D. E.D.U	
			1		•,		1		Ļ.,		L		
			4				·	-			<u> </u>		
				50	- 1		-	+-					
.A		سيك ، د		\mathcal{L}	X	ゴ	-	+		-	1	I.M.E.TP 1 , b	
.				 				+-	}		┨	<u> </u>	+ -

In this example, the assembler will inform the loader that two previously assembled segments are required from the library tape before execution. In this case, the names of the segments are TIMECARD and BONDEDUC; these names are placed on the program header card.

The instruction at location A illustrates one important point, if the symbol TIMETP1 is assumed to be an entry point in the segment named TIMECARD. The name of the segment need not be the same as one of its entry points since entry points are defined globals in the "called" segment.

Note

CALL should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When it is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

INCLUDE LIBRARY SEGMENT

INCL

Function

Any given segment may contain an INCL pseudo operation. The INCL pseudo-operation indicates that the segments named in the Operation Parameters are needed for the successful running of the given segment. In contrast to CALL, INCL causes that the required segments to be supplied at assembly time.

Format

INCL must contain

- INCL as the Operation
- In the Operation Parameters, the names of the segments which are to be supplied. These names are separated by commas.

Assembler Action

When the assembler encounters an INCL, it extracts the segment names from the Operation Parameter and places them in the program header card. In addition, the system tape is searched for the named routines which are then appended to the segment begin assembled. An indication is given to the user if the segment cannot be found.

Example

ĵ .	FERRICAL STREET	, 6, 114	FISH	BECHATION SARAMETERS										
	CATA WANG	· 5.5	5. 7. F. 24	25 PIC URA										
	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	2. X.G.		I,Ø,1, , , , , , , , , , , , , , , , , ,										
	in the first of th	,												
1		INCL CALL		I Ø TA P.E. I Ø C.P.P.D. P.P. I.N. T.I. M.E.C.A.R.C., G.Z.V.L.E. L.U.C.										
				ه المحسدين الطبارية الشعيدية الكرياطية للطبيكية الطراح الرامين لا <u>يريد الرامية الرامية الرامية الرامية الرامية</u> السام الرامية المحدد الحددة السامة السطينية السطينية الرامية الرامية الرامية الرامية الرامية الرامية الرامية ال										
,Н		нхв		I.Ø.1., 6.										
				ing the second of the second o										

In this example, the programmer requests three segments at assembly time: IOTAPE, IOCARD and PRINT. These segment names are extracted and placed on the program header card. When the segment has been assembled, the system tape is searched and the previously assembled segments are punched.

The instruction location A illustrates one important point if the symbol IO1 is assumed to be an entry point in the segment named IOTAPE. The name of the segment need not be the same as one of its entry points since entry points are defined as internal globals within the "included" segment.

Note

+

INCL should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When it is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

INCLUDE SYMBOLIC SEGMENT

INCS

Function

INCS allows the programmer to specify the names of segments which he wishes to have reassembled with his program. Sections of the program library may be kept in symbolic form and assembled with the current segment.

Format

INCS must contain

- INCS as the Operation
- In the Operation Parameters, the names of the segments which are to be assembled with the current program. These names must be separated by commas.

Assembler Action

When the assembler encounters an INCS, it extracts the segment names from the Operation Parameters and retains them in a table. At the end of Pass I, these symbolic segments are located on the system tape. The assembler then continues to process these segments just as if they had been placed within the programmer's input.

Example

REFERENCE SYMBOL			OP	ER,	A T 10	N			OPERATION PARAMETERS					
9 DATA NAME 9 16		LEV 18	/EL	20	2××C	22	U _S	24	25 PICTURE					
	ļ			<u> </u>										
	l	يـــــــــــــــــــــــــــــــــــــ			L.		_							
					L_	ļ								
	I	N.	C	L					I.Ø.T.A.P.E. I.Ø.C.A.R.D					
	I	N.	C	ڪ					GR. Ø. S. PARTS.					
. المسلم ما المسلم الما الما الما الما ال														
1 1 1 1 1 1 1														

In this example, the programmer requests two segments at assembly time: GROS and PARTS. The INCS pseudo-operation specifies that these segments reside on the system tape in symbolic form. At the end of Pass I, the system tape is searched and the two segments are located. These segments are then assembled with the current segment.

Note.

INCS should appear near the beginning of the program deck with the other pseudo-operations that effect segment linkage. When it is used in this way, the coding will contain centralized documentation which describes the interaction between segments.

BLOCK STARTED BY SYMBOL IN THE LOADER AREA BSSL

Function

BSSL allows the programmer to allocate storage in the memory that is occupied by the loader. BSSL is described under Memory Allocation Pseudo-Operations.

BLOCK PRECEDED BY SYMBOL IN THE LOADER AREA BPSL

BPSL performs a function similar to BSSL. BPSL is described under Memory Allocation Pseudo-Operations.

LAST SYMBOL IN BLOCK IN THE LOADER AREA

LSBL

LSBL performs a function similar to BSSL. LSBL is described under Memory Allocation Pseudo Operations.

ACCUMULATOR REFERENCE POINT IN THE LOADER AREA ARPL

ARPL performs a function similar to BSSL. ARPL is described under Memory Allocation Pseudo-Operations.

MEMORY ALLOCATION PSEUDO-OPERATIONS

BLOCK STARTED BY SYMBOL

BSS

Function

BSS reserves a block of consecutive memory locations and defines a Reference Symbol as the first of the reserved locations.

Format

BSS must contain

- A symbol or blanks as the Reference Symbol
- BSS as the Operation.
- An expression in the Operation Parameters for the number of words to be reserved by the BSS.

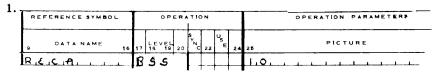
Assembler Action

The Reference Symbol is defined as the current value of the location counter. The location counter is then increased by the value of the expression in the Operation Parameter; thus, the assembler reserves a

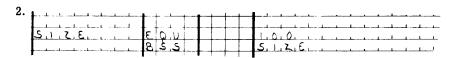
block of memory which is equal in length to the value of the expression. The symbol, therefore, represents the first location in the reserved block. BSS reserves memory; however, it does not clear the reserved area. Therefore, the programmer may not assume that an area reserved by a BSS will initially contain zeroes.

Examples

Assume that the current value of the location counter is 100.



In example 1, the BSS defines RECA as 1100 and reserves 110 locations (1100-1109).



In example 2. SIZE equals 100; therefore the BSS reserves 100 locations (1110-1209).



In example 3, the BSS defines CODE as 210, the current value of the location counter. Since no locations are to be reserved, the location counter remains at 1210.

Notes

- 1. All symbols used in the Operation Parameters of a BSS must have been previously defined (see Previously Defined Symbols, page 1-15).
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page 1-35).

BLOCK STARTED BY SYMBOL IN THE LOADER AREA

Function

BSSL reserves a block of consecutive memory locations and defines a Reference Symbol as the first of the reserved locations. Unlike BSS, the block of reserved locations is not allocated "in line"; the block is reserved in the area occupied by the loader. Thus, BSSL allows the programmer to make use of the entire computer memory for his applications because it gives him the facility to overlay the loader.

Format

BSSL must contain

- A symbol as the Reference Symbol
- BSSL as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the BSSL.

Assembler Action

The expression in the Operation Parameters is evaluated to determine the number of words to be reserved. This value and the Reference Symbol are placed in the symbol table with a special flag. At the end of Pass I, the assembler punches this information so that the storage can be allocated when the program is loaded. When the loader reserves this memory, it does not clear the allocated area. Therefore, the programmer must not assume that the area reserved by a BSSL will initially contain zeroes.

BSSL

Example

REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS								
DATA NAME	17 LE VEL 17 N C 22 E 24	PICTURE 25								
										
A. I	85.5	2								
В	B 5.5									
1 N	BSSL	5.0.0.								
Ø.U.T.	BSSL	1.0.0								
D	B 5.5	4								
		▊ ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀								

In this example, if we assume that symbol A is assigned to location 7500, symbol B will be assigned to location 7502 and symbol D will be assigned to location 7503. Symbolic reference IN will be punched in the binary deck with an indication that a block of 500 memory locations are to be reserved by the loader. The symbol IN will be assigned to the first location of this block.

Symbolic reference OUT will be punched in the binary deck with an indication that a block of 100 memory locations are to be reserved by the loader. The symbol OUT will assigned to the first location of this block.

In this example, a block of 500 locations is reserved starting at symbolic location OUT. If there are other "loader area" pseudo-operations in the program, there is no guarantee that the two blocks of reserved locations will be next to each other in memory. Thus, the programmer may not assume that IN+500 is location OUT.

Notes

- All symbols used in the Operation Parameters of aBSSL must have been previously defined (see Previously Defined Symbols, page 1-15).
- 2. The results of evaluating the expression in the Operation Parameters must be absolute (see Relocatable Errors, page 1-35).
- 3. BSSL should appear near the beginning of the program deck with the other pseudo-operations that allocate storage in the area occupied by the loader. When it is used in this way, the coding will contain centralized documentation that describes the overlay of the loader.

BLOCK PRECEDED BY SYMBOL

BPS

Function

BPS reserves a block of memory locations within a program and defines a Reference Symbol as the location which immediately precedes the block.

Format

BPS must contain

- A symbol or blanks as the Reference Symbol
- BPS as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the BPS.

Assembler Action

If there is a Reference Symbol, it is defined as one less than the current value of the location counter. The location counter is then increased by the value of the expression in the Operation Parameters; thus, a block of memory is reserved which is equal in length to the value of that expression. BPS reserves memory; however, it does not clear the reserved area. Therefore, the programmer may not assume that an area reserved by a BPS will initially contain zeroes.

Examples

Assume that the current value of the location counter is 2000.

3						2000
REFERENCES	YMBGL		Dere.	1100		OPERATION PARAMETER:
DATA Na	, M E 16	17	FVEL 2.	0, 10 22	U _E 24	FICTURE
TIP GILLE		B 1	5 .S. P. S.			1.0,

The BSS defines TABLE1 as 2000 and feserves 10 locations (2000 - 2009). The BPS defines TABLE2 as 2009 and reserves 10 locations (2010 - 2019).

1.

RE	FERENCE SYMB	O L		OP	ERA	. т I С 1	N			OPERATION PARAMETERS					
9	DATA NAME	16	17	LEVEL	20	s N _C	22	U _S	24	25	PICTURF				
				- 1							<u> </u>				
٠.			<u> </u> _		-	Ĺ	-	_							
	.4				! 	!		L.							
4	عاد بدينيت بدرستا		L	XI	<u> </u>	<u> </u>		ļ.,		٥	4444 _ 				
	 		Ь.	ے ط		ļ		Ļ.,		I.	<u> </u>				
			_			 	<u>.</u>	<u> </u>		L					
			L.,		-	L		<u> </u>							
						!	_	<u> </u>		┡					
						!	_	-		L					
					+	ļ	_	!	-	L.					
	+		-		!	▙	<u> </u>	-	<u> </u>	ļ					
				I I	<u>. </u>	İ		<u> </u>	L	L.					

Since TABLE1 was defined by a BSS and represents the first location in a reserved block, fixed index register 1 must be set to zero if the programmer desires to obtain the first item from TABLE1; one if the second item is desired, etc.

2.

	1 :	1 1 1 1	
			
1-	1		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	LXI		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
	L DS		T,A,B,L,E,2,11
		1-1-1-	
.1			

Since TABLE2 was defined by a BPS and represents the location which precedes the first position of the reserved block, fixed index register 1 must be set to one to obtain the first item from TABLE2; two to obtain the second item, etc. Because TABLE2 has been defined by a BPS any of the 10 items from the table may be obtained by loading index register one with the table position of the desired item.

Notes

- 1. All symbols used in the Operation Parameters of a BPS must have been previously defined (see Previously Defined Symbols. page 1-15).
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page 1-35).

BLOCK PRECEDED BY SYMBOL IN THE LOADER AREA

BPSL

Function

BPSL reserves a block of consecutive memory locations and defines a Reference Symbol, as the location which immediately precedes that block. Unlike BPS, the block of reserved locations is not allocated "in line"; the block is reserved in the area occupied by the loader. Thus, BPSL allows the programmer to make use of the entire computer memory for his applications since it gives him the facility to overlay the loader.

Format

BPSL must contain

- A symbol as the Reference Symbol
- BPSL as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the BPSL.

1-97 12/6/63 1-98 12/6/63

Assembler Action

The expression in the Operation Parameters is evaluated to determine the number of words to be reserved. This value and the Reference Symbol are placed on the symbol table with a special flag. At the end of Pass I, the assembler punches this information so that the storage can be allocated when the program is loaded. When the loader reserves this memory, it does not clear the allocated area. Therefore, the programmer may not assume that the area reserved by a BPSL will initially contain zeroes.

REFERENCE # MBOL	I	OPERATION [OPERATION PARAMETERS											
DATA NAME 16	17	LE 18	VEL 19	20	5 × N	22	U _S	24	25						P	ст	UR	E.				
	L	ļ.			L																_	_
<u> </u>		·			L	├	_	_	ļ	-	-							-1.				
7 .	R	ċ	- 5		╂─		-		5	т	+	-		-					Ц		-	-
3	ğ	5		-	1	 	\vdash	\vdash	ŕ	<u>. </u>	 	 -						+		-	4	
.N	ß		S	<u>L</u>					5	0	10	+			1	———— 				-4		
Y.U.T	ß	P	S	L	L		-		1	٥	٥٠) [1							4 -	
) <u></u>	В	S	٤.		<u> </u>	<u> </u>			7	٠	ــــ		-4							<u>.</u> .		
	Ì		L		H		-	-	-	L	1		1	-1_			-			-1	4	
		j`i	-						-				+	-		٠		-			4	

In this example, if we assume that symbol A is assigned to location 7500, symbol B will be assigned to location 7502 and symbol D will be assigned to location 7503.

Symbolic reference IN will be punched in the binary deck with an indication that a block of 500 memory locations are to be reserved by the loader. The symbol IN will be assigned to the location which immediately precedes this block.

Symbolic reference OUT will be punched in the binary deck with an indication that a block of 100 memory locations are to be reserved by the loader. The symbol OUT will be assigned to the location which immediately precedes this block.

In this example, a block of 500 locations is reserved starting at symbolic location IN+1 and a block of 100 locations is reserved starting at symbolic location OUT+1. If there are other "loader area" pseudo-operations in the program, there is no guarantee that the two blocks of reserved locations will be next to each other in memory. Thus, the programmer may not assume that IN+500 is location OUT.

Notes

- 1. All symbols used in the Operation Parameters of a BPSL must have been previously defined.
- 2. The results of evaluating the expression in the Operation Parameters must be absolute (see Relocations Errors, page 1-35).
- 3. BPSL should appear near the beginning of the program deck with the other pseudo-operations that allocate storage in the area occupied by the loader. When it is used in this way, the coding will contain centralized documentation that described the overlay of the loader.

LAST SYMBOL IN BLOCK

LSB

Function

LSB is used to reserve a block of consecutive memory locations and to assign a Reference Symbol to the last of the reserved locations.

Format

LSB must contain

- A symbol or blanks as the Reference Symbol
- LSB as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the LSB.

Assembler Action

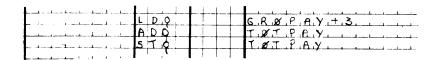
The location counter is increased by the value of the expression in the Operation Parameters; thus, the assembler reserves a block of memory which is equal in length to the value of that expression. The Reference Symbol is defined as one less than the value of the location counter, after it has been increased. The symbol, therefore, represents the last location in the reserved block. LSB reserves memory; however, it does not clear the reserved area. Therefore, the programmer may not assume that an area reserved by an LSB will initially contain zeroes.

Example

Assume that the current value of the location counter is 1100.

REFERENCE SYMBOL	OPERA	TION	OPERATION PARAMETERS
DATA NAME	17 1H 19 20	S U S E 24	PICTURE 25
TOTPAY	LISIA		4
G.R. Ø.P.A.Y.	BSS	.	4

The LSB reserves four locations (1100-1103) and defines TOTPAY as 1103. The BSS defines GROPAY as 1104 and reserves four locations (1104-1107).



In the example, the programmer may address the area reserved by the LSB symbolically. The area reserved by the BSS must be addressed relative to the symbol, GROPAY.

LAST SYMBOL IN BLOCK IN THE LOADER AREA

LSBL

Function

LSBL reserves a block of consecutive memory locations and defines a Reference Symbol to the last of the reserved locations. Unlike LSB, the block of reserved locations is not allocated "in line"; the block is reserved in the area occupied by the loader. Thus, LSBL allows the programmer to make use of the entire computer memory for his applications since it gives him the facility to overlay the loader.

Format

LSBL must contain

- A symbol as the Reference Symbol
- LSBL as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the LSBL.

Assembler Action

The expression in the Operation Parameters is evaluated to determine the number of words to be reserved. This value and the Reference Symbol are placed on the symbol table with a special flag. At the end of Pass I, the assembler punches this information so that the storage can be allocated when the program is loaded. When the loader reserves this memory, it does not clear the allocated area. Therefore, the programmer may not assume that the area reserved by an LSBL will initially contain zeroes.

R	REFERENCE SYMBOL									OPERATION													O F	ER	A T	10 N	ı F	AF	t A M	4 E .	TEF	₹ \$				
9				D	Α.	T A	. N	Ι Α	м	E			16	17	L.F	۷E	.20	57,	c	22	U _S E	24	25							Р	1 C	1 U I	RE			
				_												٠		L						ı				+-			-					
	L					_		_			_		_			1	1	L	4				L												١	
	ı		4		٦.		_	_	_			1	4	_			+-	↓.	4	_			_		+			_	_		_				1	
٨	L		1			_	_		١.	1		_		В	S	ۍ	+-	╀	4				2		-			4	-		_				_	
B.		_	_	_	_		_	_		_			_	B		چـ	4-	Ł	4	_	_	ļ.,,	1	٠	4									L	4	
<u>. </u>	u	N			4		_		_	_1				L	S	ıΒ	1	╁.	4				5				_	-						L	4	
<u>×</u>	ı	U		1	_	_	ı	_	L_	_			4	L	5	В	44	╀	+				1	٥٠	2ب	٠.		-	-		1_	_		L	4	
D	.1.		_		ユ.		4		L			٠		В	S	ي		╀	4				4		ــــ	4		١		_1.					+	
	_						L.		L_			_	_	_	٠		_ -	╀	+	_					-4-			ــــا	4	_					١	1
							1	_	_						•	1	+	╁	+			-	-			1		1	-			1		L	٠	_
	L		4	_	1		1_		L		_		-		•	1	+-	╀	+	4		-		<u> </u>				4-	_			+			-	
						_		_				_			L.	1		1	4		L		ļ	_				٠	4		1_			<u> </u>	+	

In this example, if we assume that symbol A is assigned to location 7500, symbol B will be assigned to location 7502 and symbol D will be assigned to location 7503.

Symbolic reference IN will be punched in the binary deck with an indication that a block of 500 memory locations are to be reserved by the loader. The symbol IN will be assigned to the last location in this block.

Symbolic reference OUT will be punched in the binary deck with an indication that a block of 100 memory locations are to be reserved by the loader. The symbol OUT will be assigned to the last location of this block.

In this example, a block of 500 locations is reserved starting at symbolic location IN-499 and a block of 100 locations is reserved starting at symbolic location OUT-99. If there are other "loader area" pseudo-operations in the program, there is no guarantee that the two blocks of reserved locations will be next to each other in memory. Thus, the programmer may not assume that IN+100 is location OUT.

Notes

- 1. All symbols used in the Operation Parameters of LSBL must have been previously defined. (See Previously Defined Symbols, page 1-15.)
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page 1-35).
- 3. LSBL should appear near the beginning of the program deck with the other pseudo-operations that allocate storage in the area occupied by the loader. When it is used in this way, the coding will contain centralized documentation that describes the overlay of the loader.

ACCUMULATOR REFERENCE POINT

ARP

Function

ARP insures the programmer that those symbols which he uses as accumulator references are defined as locations which are evenly divisible by four. This need occurs because of the hardware requirement that the high order word of the quadruple accumulator be a memory location which is evenly divisible by four. This concept is referred to as 0 MOD 4 and means that the number of the memory location must produce a zero remainder when divided by four.

Format

ARP must contain

- A symbol or blanks as the Reference Symbol
- ARP as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the ARP.

Assembler Action

The ARP is executed in three steps:

1. The assembler determines whether the current value of the location counter is evenly divisible by four. If it is, the

assembler goes on to Step two. Otherwise, the assembler advances the location counter to the next location which is evenly divisible by four. In this process, from one to three locations may be "skipped over". These skipped locations are filled with Fill words which are generated by the assembler (see Fill on page 1-111. It is strongly recommended that the Fill instruction be used in conjunction with ARP to prevent errors which might occur if for any reason the program should branch to a skipped location).

- 2. If there is a Reference Symbol, it is defined as the current value of the location counter as adjusted in Step one.
- 3. The location counter is now increased by the value of the expression in the Operation Parameters; thus, reserving a block of memory which is equal in length to the value of that expression. ARP reserves memory; however, it does not clear the reserved area. Therefore, the programmer may not assume that an area reserved by an ARP will initially contain zeroes.

Example

PEFFERENCE SYMBOL	OFER	ATION	OPERA JN Salent				
OA!A SAME		S U _S	PICTUR!				
MAS TICKIN	8 Q U		1.0.C.O				
DET BILLIN	ARP.		4.Q				

In this example the location counter is set to the next location which is evenly divisible by four. The intervening words are Filled and two input areas, MASTERIN and DETAILIN, are reserved. The accumulator may now be located around various portions of the input records.

Notes

- 1. All symbols used in the Operation Parameters of an ARP must have been previously defined (see Section Previously Defined Symbols, page 1-15).
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page 1-35).

ACCUMULATOR REFERENCE POINT IN THE LOADER AREA ARPL

Function

ARPL reserves a block of consecutive memory locations starting at a 0 Mod 4 address and defines a Reference Symbol as the first of the reserved locations. Unlike ARP, the block of reserved location, when using ARPL, is not allocated "in line". The block is reserved in the area occupied by the loader. Thus ARPL allows the programmer to make use of the entire computer memory for his applications since it gives him the facility to overlay the loader.

Format

ARPL must contain

- A symbol as the Reference Symbol
- ARPL as the Operation
- An expression in the Operation Parameters for the number of words to be reserved by the ARPL.

Assembler Action

The expression in the Operation Parameters is evaluated to determine the number of words to be reserved. This value and the Reference Symbol are placed on the symbol table with a special flag. At the end of Pass I, the assembler punches this information so that storage can be allocated when the program is loaded. When the loader reserves this memory, it does so after stepping its data location counter to a 0 Mod 4 state. The procedure is similar to the assembler action on ARP with one exception: Fill words are not generated by the loader. When the loader reserves this memory, it does not clear the allocated area. Therefore, the programmer may not assume that the area reserved by ARPL will initially contain zeroes.

REFERENCE SYMBOL	OPERA	TION	OPERATION PARAMETERS				
9 DATA NAME 16	17 LSVEL 17 P 19 20	S Y N C 72 E 24	PICTURE 25				
A	BSS		2, , , , , , , , , , ,				
B	B S.S						
[.N	ARIPL		5,0,0,				
8,0 T	ARPL		1.0.0.				
D	B 5 5		4				

In this example, if we assume that symbol A is assigned to location 7500, symbol B will be assigned to location 7502 and symbol D will be assigned to location 7503.

Symbolic reference IN will be punched in the binary deck with an indication that a block of 500 memory locations starting at a 0 Mod 4 cell are to be reserved by the loader. The symbol N will be assigned to the first location (0 Mod 4) of this block.

Symbolic reference OUT will be punched in the binary deck with an indication that a block of 100 memory locations starting at a 0 Mod 4 cell are to be reserved by the loader. The symbol OUT will be assigned to the first location (0 Mod 4) of this block.

In this example, a block of 500 locations is reserved starting at symbolic location IN and a block of 100 locations is reserved starting at symbolic location OUT. Further, both locations IN and OUT are assigned to 0 Mod 4 memory locations. If there are other "loader area" pseudo-operations in the program, there is no guarantee that the two blocks of reserved locations will be next to each other. Thus, the programmer may not assume that IN+500 is location OUT.

Notes

- 1. All symbols used in the Operation Parameters of an ARPL must have been previously defined. (See Previously Defined Symbols, page 1-15).
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page 1-35).
- 3. ARPL should appear near the beginning of the program deck with the other pseudo-operations that allocate storage in the area occupied by the loader. When it is used in this way, the coding will contain centralized documentation that describes the overlay of the loader.

ACCUMULATOR

ACUM

Function

The ACUM pseudo-operation allows the program to establish a working accumulator of 1, 2, 3.or 4 words and to define a symbol representing the location of that accumulator. Further, it assures, through the generation of Fill words (see FILL, page 1-111) that the area reserved for that accumulator will be properly positioned in memory.

Note

Since the most significant word of the quadruple accumulator must always be in a location which is evenly divisible by 4. it follows that:

- The most significant word of the triple accumulator must always be in a location which, when divided by 4, produces a remainder of 1.
- The most significant word of the double accumulator must always be in a location which, when divided by 4. produces a remainder of 2.

• The most significant word of the single accumulator must always be in a location which, when divided by 4, produces a remainder of 3.

Format

ACUM must contain

- A symbol as the Reference Symbol
- ACUM as the Operation
- In the Operation Parameters, an expression for the size of the accumulator.

Assembler Action

The assembler evaluates the expression in the Operation Parameters, and, based on the value of that expression, adjusts the location counter to the proper setting; producing a Fill word for each location skipped while adjusting the location counter. The Reference Symbol is then defined as the current value of the location counter and the value of the expression is added to the location counter; thus, reserving space for the specified accumulator.

Note

The expression for the accumulator size may be zero, one, two, three, or four. If the expression is zero, the ACUM will be treated as an ARPO . If it exceeds four, it will be flagged and treated as an ACUM 4. ACUM reserves memory; however, it does not clear the reserved area. Therefore, the programmer may not assume that an area reserved by an ACUM will initially contain zeroes.

Examples

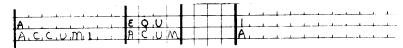
The programmer requires a single accumulator.

1.

REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS
DATA NAME	S S US S V N C 22 E	PICTURE 24 25
	ARP	3
A . C . C . U . M . 1	85.5	<u> </u>
		•

Assume that the current value of the location counter is 1010; then the ARP 3 Fills locations 1010-1011 and sets the location counter to 1015. The BSS 1 defines ACCUM1 as 1015 and reserves memory for the single accumulator. Five locations, 1010-1014 are wasted.

2.



Assume the location counter is positioned at 1010; then, the ACUM A Fills location 1010 and advances the location counter to 1011. ACCUM: is defined as 1011, a single accumulator, and memory is reserved for a single accumulator. Only one location, 1010 is wasted.

3.

F, I, C, A,	la cum L	0, , , , , , , , , , , , , , , , , , ,
	D E.C 0	4.2.0.0.0.0.0.
	▎ ▘▘▏ ▗ ▋┤┼┼	
	LAI	F.I.C.A.
	CDA	G. R. P. A. Y.

Example three sets the location to the next location which is evenly divisible by four, Fills the intervening locations, and forms a four-word constant FICA. Later instructions locate the accumulator at FICA and determine if GROPAY exceeds 480000.

Notes

- 1. All symbols used in the Operation Parameters of an ACUM must have been previously defined (see Previously Defined Symbols, page I-15).
- 2. The result of evaluating the expression in the Operation Parameters must be absolute (see Relocation Errors, page I-35).

FILL WORD FILL

Function

FILL allows the programmer to specify the contents of Fill words produced by the assembler during the processing of an ARP or an ACUM.

Format

FILL must contain

- A mnemonic machine operation code as the Reference Symbol
- FILL as the Operation
- An expression in the Operation Parameters for the address
- An expression in the Operation Parameters for the address of the Fill word.

Assembler Action

The basic assembly language maintains a simulated line of coding with which it fills location skipped during the processing of an ARP or ACUM. Initially, this line is set to HLT O.

When a Fill word is required, the assembler supplies this line of coding. FILL instructs the assembler to reconstruct the simulated line of coding. The mnemonic operation code from the Reference Symbol of the FILL becomes the Operation of the simulated line of coding; the Operation Parameters from the FILL complete the simulated line of coding. Any Fill words produced by the assembler, prior to another FILL, will contain the command specified in the FILL.

Examples

1.

REFERENCE SYMBOL	ł	OPERA			N			OPERATION PARAMETERS
DATA NAME 16	17	LEVEL 18 19	20	SYNC	22	U _S	24	PICTURE 28
	L	D.S	!-					f)
	A	D, 5					` '	B
P.X.B.	F	IL	L					D.U.M.P. 16.
	A	R.P						1,0,0,0,

In example one, the Fill line is set to PXB DUMP, 6 to be used if locations are skipped during execution of the ARP. Assume that the location counter for LDS is 1000. The coding above would result in the assembler assigning the following location values:

1000	LDS	A
1001	ADS	В
1002	PXB	DUMP, 6
1003	$\mathbf{P}\mathbf{X}\mathbf{B}$	DUMP, 6
1004	ARP	1000

2.

	-			-	 	
IPX B	15 1	.1 1	1		1	M. Ø. N. I J. Ø R. I b.
2	1.	-	-	-	 	
A. C. C. U.M.I.	IALC	u.M				13
	1		T		_	**************************************

Example two sets the Fill line to a PXB MONITOR, 6 and uses the PXB as Fill if locations are skipped during execution of the ACUM.

CONSTANT PRODUCING PSEUDO OPERATIONS

Often in writing a program, the programmer will find that some of the items with which he is working remain static during the duration of his program. Such items are called constants, and they occur in nearly all programs.

A constant may be classified in terms of its composition as numeric, alphabetic, or alphanumeric.

A numeric constant is composed of some combination of the numerals zero to nine and may contain a leading plus sign or minus sign. An alphabetic constant is composed of some combination of the 26 alphabetic characters; it has no sign. An alphanumeric constant may contain numeric, alphabetic, or special machine characters; it has no sign.

In terms of its use, a constant may be classified as either numeric or alphanumeric. A constant is numeric when it is used computationally, that is, when it is involved in an arithmetic or logical operation. If a constant is not used computationally, it is considered to be alphanumeric.

For example, the programmer is producing a printed report, each page of which will contain a page number and a date. The page number will change with each page; however, the remaining data is constant:

PAGEA NO. AXXXX

DATE 011563

To take care of the requirements, the programmer creates four constants:

- PAGE∆NO
- 2. DATE
- 3. 011563
- 4. 1

In terms of composition, the first constant is alphanumeric - letters and symbols. The second constant is alphabetic - all letters. The third and fourth are numeric.

In terms of their use, constants one, two and three are alphanumeric not computational. Constant four is numeric; it is used to update the page count and is, therefore, computational. The pseudo-operations which are described on the following pages may be used by the programmer to generate the constants required in his program. They may also be used to set areas reserved for variable data to an initial state.

DECIMAL CONSTANTS

DECS-DECD-DECT-DECQ

Function

These four mnemonics represent

DECS	Decimal Single
DECD	Decimal Double
DECT	Decimal Triple
DECQ	Decimal Quadruple

These four pseudo-operations are used to create decimal constants of one, two, three or four words respectively.

Format

A decimal constant contains

- A symbol or blanks as the Reference Symbol.
- DECS, DECD, DECT, or DECQ as the Operation.
- In the Operation Parameters, a decimal number, with or without a leading sign, as the constant.

The limitation on the number of digits which may appear in a decimal constant is as follows:

DECS	One to four digits
DECD	One to eight digits
DECT	One to twelve digits
DECQ	One to sixteen digits

Assembler Action

The specified constant is right-justified and assembled into one, two, three or four computer words depending upon the mnemonic operation code. If it is preceded by a minus sign, the constant is assembled as a negative quantity. If there is a leading plus sign or if there is no sign, the constant is assembled as a positive quantity.

If there is a Reference Symbol, it is defined as the location of the word containing the least significant digit of the decimal constant. Decimal points may not appear in constants generated by Decimal Instructions.

Example

Assume that the current value of the location counter is 1000.

REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS								
DATA NAME 16	17 18 19 20 C 22 E 24	25 PICTURE 40 4								
K. I.	D E C S	T,2,0,0,0,								
K.3.	D.E.C.S.	+11.2.3.4.5.6.7.8.								
K 5	DECD	1.2								
K 6	D E C T	+,1,2,3,4,5,5,7,8,9,6,1,2; -,1,2,3								
K.7		+11.2.3.4.5.6.7.8 9 0 1.2.3.4.5 6								
N.A	DECO	-1.2.3.4								

This coding produces the following results

Decimal Location	Content	
1000	2000	K1 is defined as 1000
1001	$000\overline{1}$	K2 is defined as 1001
1002	1234	
1003	5678	K3 is defined as 1003
1004	0000	
1005	0012	K4 is defined as 1005
1006	1234	
1007	5678	
1008	$901\bar{2}$	K5 is defined as 1008
1009	0000	
1010	0000	
1011	$012\bar{3}$	K6 is defined as 1011
1012	1234	
1013	5678	
1014	9012	
1015	3456	K7 is defined as 1015
1016	0000	
1017	0000	
1018	0000	
1019	$\boldsymbol{123\bar{4}}$	K8 is defined as 1019
ALPHANUMERIC CONSTA	NT	AN

Function

 $\ensuremath{\mathrm{AN}}$ enables the programmer to form alphanumeric constants within his program.

Format

An alphanumeric constant may contain

- A symbol or blanks as the Reference Symbol
- AN as the Operation
- A word count (1-13) in the second half of the Operation
- The characters of the alphanumeric constant in the Operation Parameters.

Assembly Action

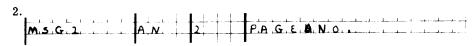
If there is a Reference Symbol, it is defined as the current value of the location counter, i.e., as the first word of the constant which is formed. The assembler forms the number of words indicated in the second half of the Operation. Each word contains four of the specified characters; the first word contains the first four characters, the second word the next four characters, etc. Since the Operation Parameters contains a maximum of 52 columns (25-76), the maximum word count permissible in an AN is 13 (52 \pm 4). An alpha-numeric constant produced by AN may contain any combination of valid machine characters including commas and blanks. Commas may immediately follow the last character of the constant in the Operation Parameters.

Examples

Assume that the location counter is 1000



In example one, MSG1 will be assigned to 1000 and form four words (1000-1003) containing 16 characters from the Operation Parameters.



In example two, MSG2 will be assigned to 1004 and form two words (1004-1005) containing eight characters from the Operation Parameters.



In example three, MSG3 will be assigned to 1006 and form two words (1006-1007) containing eight characters from the Operation Parameters.

	REFERENCE SYMBOL		OP	T10	N				-	 OPERATION PARAMETERS									
4.	DATA NAME	17	LEVEL		s ^{>} N°C	22	U _S	24	25				PI	СТ	נו ז	RE	-		
		A	N	-	I	3				-	 							 	1

In example four 13 words of blanks will be formed.

LAST SYMBOL ALPHANUMERIC	LSAN

Function

LSAN enables the programmer to enter alphanumeric constants into his program. It is identical in use to the AN pseudo-operation which was just described, with the exception that the Reference Symbol is assigned to the last word produced by the assembler.

Format

LSAN may contain

- A symbol or blanks as the Reference Symbols
- LSAN as'the Operation
- A word count (1-13) in the second half of the Operation
- The characters of the alphanumeric constant in the Operation Parameters.

Assembler Action

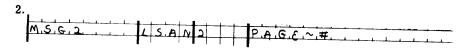
If there is a Reference Symbol, it is assigned to the last word of a multiword constant which is formed by the assembler. The assembler constructs the number of words indicated in the second half of the operation. Each word will contain four characters. The first word is taken from columns 25-28 of the Operation Parameters; each succeeding word is constracted from the succeeding four columns of the Operation Parameters. An LSAN Operation allows for a maximum of 13 words to be entered in the 52 columns (25-76). Any combination of valid symbols may be used in the Operation Parameters.

Examples

Assume that the location counter is at 1000.

REFERENCE SYMBOL	OPERATION							OPERATION PARAMETERS
9 DATA NAME 16		EVEL 8 19	20	s YNC	22	U _S	24	4 25 PICTURE
M. S. G. L.		SA	Ν	5				Ø.P.E.R.A.T. Ø.R.A.E.R.R.Ø.R.

In example one, MSG1 will be assigned to 1004 and form five words taken from the Operation Parameters. The last word will contain blanks since there are only 14 characters specified.



In example two, MSG2 will be assigned 1006 and the six characters in PAGE \sim # will be found in the six high order position of location 1005 and 1006 with two trailing blanks.



In example three MSG3 will be assigned to 1009 and the numbers 1---0 will appear in the high order ten characters position of words 1007-1009.



In example four, blanks will be assigned to location 1013 and four words of blanks will be placed in words 1010-1013.

OCTAI	CONTORAN	TOTAL	
OCTAL	CONSTAN	N 15	OCTS-OCTD-OCTT-OCTQ
Arr name.		11 70 11 Management	OCTS-OCTD-OCTT-OCTQ

Fu action

These four mnemonics represent

OCTS	Octal Single
OCTD	Octal Double
OCTT	Octal Triple
OCTQ	Octal Quadruple

These four pseudo-operations are used to create octal constants of one, two. three, or four words respectively.

For mat

An octal constant may contain

- A symbol or blanks as the Reference Symbol
- OCTS, OCTD, OCTT, or OCTQ as the Operation
- An unsigned octal number as the constant in the Operation Parameters.

The limitation on the number of digits which may appear in an octal constant is as follows:

OCTS	One to eight octal digits
OCTD	One to sixteen octal digits
OCTT	One to twenty-four octal digits
OCTQ	One to thirty-two octal digits

Assembly Action

The specified constant is right justified and assembled into one, two, three or four computer words depending upon the mnemonic operation code. If there is a Reference Symbol, it is defined as the location of the word containing the least significant digit of the octal constant.

Example

Assume that the current value of the location counter is 2000.

REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS
DATA NAME	S US YN S E 24	PICTURE 43
Ø.K.L.	Ø C T S	1, 7, 7, 7, 7, 7, 7, 7, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
Ø. K. 2	Ø C T S	1.2
Ø1 K. 3	Ø. C. T. D.	1.7.1.7.1.7.1.7.1.4.4.4
Ø. K. 4	X CTT	7.7.7.7.6.6.6.6.5.5.5.5
Ø.K.6.	Ø C.T.T.	1, 2, 3, 4, 5, 6
Ø.K.7.	ØCTO	7.7.6.6.5.5.4.4.3 3.2.2.1.1
Ø. K. 8	Ø.C.T.O.	1,2,3,4,5,6,7.0

This coding produces the following results:

Decimal Location	Content	
2000	1777777	OK1 is defined as 2000
2001	00000012	OK2 is defined as 2001
2002	17171717	
2003	17171717	OK3 is defined as 2003
2004	00000000	
2005	00001234	OK4 is defined as 2005
2006	00000000	
2007	00007777	
2008	66665555	OK5 is defined as 2008
2009	00000000	
2010	00000000	
2011	00123456	OK6 is defined as 2011
2012	00000000	
2013	00000000	
2014	00776655	
2015	44332211	OK7 is defined as 2015
2016	00000000	
2017	00000000	
2018	00000000	
2019	12345670	OK8 is defined as 2019

ASSEMBLY OUTPUT CONTROL PSEUDO-OPERATIONS

COTOOT TO	T TL
TITLE	
111111	

Function

TTL allows the programmer to specify the heading line printed at the top of each page of the assembly listing.

Format

TTL must contain

- TTL as the Operation
- A page heading in columns 25-76 of the coding form.

Assembler Action

When the assembler encounters a TTL in Pass II, it prints the card containing the TTL, slews the assembly listing to the top of page; and prints columns 25-76 of the TTL as the page heading. This same heading will be printed on all subsequent pages of the listing until a new TTL is received. Prior to encountering a TTL, the assembler will supply the heading line.

Example

		0 5	ERA	TIC	N																		c	PE	R A 1	011	N	A FR	AME	TER	5		
17	LE	VEL.	20	S _{YN} C	22	U _S E	24	25						F	· I C	ΤU	RE							40	41	c c	U R	5	45				
ī	T	L						P	A	И	R	Ø	1.1	L	L		M	A.	S	T	3	R	L	F	I	L	3.		υ.	PJ	D. £	LF	3.
1									<u>. </u>			ı												_			·	L	<u></u>				
ì				ı		li																		. 1					Ι.		- 1	,	1

EJECT PAGE

EJT

IDENTIFY BINARY OUTPUT

IDEN

Function

EJT causes the assembler to slew the assembly listing to the top of the page before reading the next instruction.

Format

EJT contains EJT as the Operation.

Assembler Action

When the assembler encounters an EJT in Pass II, it prints the card containing the EJT and slews the assembly listing to the top of the page.

Example

REFERENCE SYMBOL	OPERATION	OPERATION PARAMETERS
DATA NAME	17 16 19 20 C 22 E 24	PICTURE 25
	EJT	
-		

1-123

Function

IDEN allows the programmer to specify the identification, in columns 77-80, for his binary cards.

Format

IDEN must contain

- IDEN as the Operation
- Four identification characters in columns 25-28.

Assembler Action

When the assembler encounters an IDEN, it causes the characters contained in columns 25-28 from the IDEN to be punched in all binary output cards. Any four characters selected by the programmer may be used as identification. If the source program does not contain an IDEN, the identification field in the binary output cards is blank.

Example

REFERENCE SYMBOL	OPERATION				N	* 1 10		OPERATION PARAMETERS
DATA NAME 16	17	LEVEL 18 19	20	8×20	22	U _S	24	PICTURE 25
	I	J,E	Ν					G, E, C, D, , , , , , , , , ,
				_	_	_	_	

Restriction

The source program may not contain more than one IDEN.

FULL BINARY CARDS

FULL

SYMBOLIC ANALYZER

SYAL

Function

The normal binary output card contains such data as card type, word count, checksum, instruction flags, instructions, and identification. Occasionally, as in the case of a memory load program, it may be necessary to produce output cards which contain only instructions. FULL controls the format of the binary output card, by instructing the assembler to produce its output in the full card format - 40 instructions per card.

Format

FULL contains FULL in the Operation.

Assembler Action

When the assembler encounters a FULL it produces the binary program in the full card format.

Example

REFERENCE SYMBOL	Ι	0 P	E.R.	1 T A	N				OPERATION PARAMETERS
DATA NAME	17	LF/EL	20	s _× _N _C	22	U S E	24	25	PICTURE
	F	UL	L					-	
					L			1	 <u> </u>

Function

SYAL instructs the assembler to produce a symbolic analyzer tape during Pass II and to engage the Symbolic Analyzer Program upon termination of Pass II.

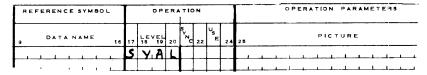
Format

SYAL contains SYAL as the Operation.

Assembler Action

When the assembler recognizes the SYAL in Pass I, it signals Pass II to produce the symbolic analyzer tape and to engage the Symbolic Analyzer upon termination of Pass II. The symbolic analyzer portion of the assembler is optional but it is highly recommended. Its use requires an additional tape unit.

Example



MISCELLANEOUS PSEUDO-OPERATIONS

ORIGIN	ORO

Function

ORG sets the location counter to a specific value during the assembly of a program.

Format

ORG may contain

- A symbol or blanks as the Reference Symbol
- ORG as the Operation
- An expression in the Operation Parameters to indicate the value at which the location counter is to be set.

Assembler Action

If there is a Reference Symbol, it is defined as the value of the expression in the Operation Parameters. Since ORG sets the location counter to a specified value, it causes the next assembled instruction to be assigned to the location represented by that value. The process is completed in Pass II where the assembler informs the loader of the fact that a change in the location counter has occurred. This is accomplished by means of flags on the binary output.

Examples

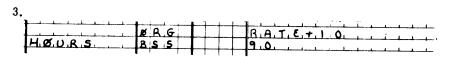
1.

REFERENCE SYMBOL	OPERAT	10 N	OPERATION PARAMETERS		
DATA NAME	LEVEL 17 18 19 20	NC 22 E 24	PICTURE 25		
RIAITIEL	ØRG		1,0,0,0,		
<u> </u>	B 5 5		1,0,		

The ORG sets the location counter to 1000 decimal and defines RATE as 1000. The BSS reserves 10 locations (1000-1009).

									•	
2.	REFERENCE SYMBOL			OP	ER,	TIC	N			OPERATION PARAMETERS
۷.	DATA NAME 16	17	L E	VEL	20	2 × × C	22	U S E	24	26 PICTURE
		L	-		_	L _				
		M	R	ıG.	<u> </u>	_				1.1.0.0
	T. L.M. E.	8	5	ڪر	-	L	_	_	Щ	1.0.

The ORG sets the location counter to 1100 decimal. The BSS reserves 10 locations (1100-1109) and defines TIME as 1100.



The ORG sets the location counter to RATE+10 (1010 decimal). The BSS reserves 90 locations (1010-1099) and defines HOURS as 1010.



The ORG sets the location counter to 200 decimal and defines START as 200. The LDS 1000 is assigned to location 200 and BEGIN is defined as 200.

Notes

- 1. All symbols appearing in the Operation Parameters of an ORG must have been previously defined (see Previously Defined Symbols, page 1-15).
- 2. The expression in the Operation Parameters may not contain an external global symbol (see Local and Global Symbols, page 1-31). This function is accomplished by means of the SGMT pseudo-operation (see description of SGMT, page 1-35).

ORGO

Function

ORGO sets the location counter to a specific octal value during the assembly of a program.

Format

ORGO must contain

- A symbol or blanks as the Reference Symbol
- ORGO as the Operation
- An unsigned octal number in the Operation Parameters as the setting of the location counter.

Assembler Action

The ORGO performs the same function as the ORG, however, the setting of the location counter must be expressed as an octal number in the Operation.

Example

	REFERENCE SYMBOL	0 P	ERA	Tic	N			OPERATION PARAME ERS
dramana, sug	DATA NAME	1 EVE	20	s _Y _N _C	22	U _S	24	P1CTURE 25
Ī	I NIR E C	Ø, R 16	Ø				1	2,6,0,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
PARTIES SEL	L							
			-					استناست المستناسية سياسي المستانيين المستانين المستانيين لمستانيين لمستانيين لمستانيين لمستانيين ال

The ORGO sets the location counter to 200 octal (128) decimal and defines INREC as 128 decimal.

EQUALS

Function

EQU instructs the assembler to define the symbol which appears in the Reference Symbol as the value specified in the Operations Parameters.

Format

EQU must contain

- A symbol as the Reference Symbol
- EQU as the Operation
- An expression in the Operation Parameters for the value of the Reference Symbol.

Assembler Action

The assembler defines the Reference Symbol as the value of the expression in the Operation Parameters. If the expression is an asterisk, the Reference Symbol is defined as the current value of the location counter.

Examples

REFERENCE SYMBOL	OPER#	TION	OPERATION PARAMETERS		
DATANAME 16	17 LEVEL 17 18 19 20	5 NC 22 E 24	PICTUR. 25		
PIALYICIDI II	εφιυ		* <u> </u>		

In example 1, the EQU defines PAYCD as the current value of the location counter 1000.

1-130

EQU

REFERENCE SYMBOL	I	OP	ERA	TIC	N			OPERATION PARAMETERS
DATA NAME 15	17	LEVEL 18 19	20	S _Y _N C	22	u S E	24	PICTURE 25
T.L.M.C.D.	8	Q.U	ļ	L				P.A.Y.C.D

In example 2, the EQU defines TIMCD as the symbolic value PAYCD (1000).

3.



In example 3, the EQU defines FICA as 200.

Notes

- All symbols appearing in the Operation Parameters of an EQU must have been previously defined. (see Previously Defined Symbols, page 1-15).
- 2. The expression in the Operation Parameters may not contain an external global symbol (see Local and Global Symbols, page 1-31). This function is accomplished by means of the EQUG pseudo-operation (see page 1-85).

EQUO

Function

EQUO instructs the assembler to define the symbol which appears in the Reference Symbol as the octal number specified in the Operation Parameters.

Format

EQUO must contain

- A symbol as the Reference Symbol
- EQUO as the Operation
- An unsigned octal number in the Operation Parameters as the value of the Reference Symbol.

Assembler Action

EQUO performs the same function as EQU.

Example

REFERENCE SYMBOL		OP	ER	T 10	N		OPERA						I A F	ION PARAMETERS					
DATA NAME	17	LEVEL 18 19	20	SYNC.	22	U _S	24	25						PI	ст	UR	E		
A,G.E	ε,	O 'O	K					L	٥	L									
_ + + - + - + - + - + - +			-	L.	<u> </u>	L.			ـــــا			Щ.							
	_		-	ļ				<u> </u>				L	-4						
	ļ		,	L								٠					L.		

The EQUO defines AGE as 101 octal (65 decimal).

A programmer may decide to further subdivide his segments into smaller regions, to code each of the regions as an entry, and to combine the regions as a single source segment for assembly.

Since each of the regions is written separately, there is a possibility that similar symbols exist in two or more of the regions. If similar symbols do exist, multiple defined errors will occur during the assembly. PRFX minimizes this problem by allowing the programmer to specify a unique prefix character for each of his regions.

Format

PRFX must contain

- A single alphanumeric or numeric prefix character in card column nine, the first column of the Reference Symbol.
- PRFX as the Operation.

Assembler Action

When the assembler encounters a PRFX, it saves the prefix character. During assembly, the symbols which appear in subsequent instructions in either the Reference Symbol or the Operation Parameters are prefixed with that prefix character. The assembler stops prefixing instructions when it encounters a new PRFX. Subsequent instructions are prefixed with the new prefix character. This procedure minimizes the possibility of multi-defined symbols between regions.

The assembler right justifies all symbols and replaces leading blanks with zeroes. Thus, the symbol PAY becomes 00000PAY. When the assembler encounters a symbol in a prefixed region, it examines the high order character of that symbol. If the character is a zero, the assembler replaces it with the prefix character. Thus, in a region prefixed by X, PAY becomes X0000PAY.

Note:

Eight character symbols are not prefixed and a region prefixed by a zero is equivalent to an unprefixed region.

Communication Between Regions

Although the regions of a segment are treated as distinct units by the programmer, there will be times when communication between them is required; one may contain data, constants or subroutines used by another. Since eight character symbols are not prefixed, they may be used whenever communication is necessary. If the programmer does not desire to use eight character symbols, he may employ the \$ character to facilitate communication.

When using the \$ method, the programmer should append the proper prefix character and the \$ to the high order side of the symbol before placing it in the Operation Parameters of his instruction. The symbol will then be assembled using the character which precedes the \$ as the prefix character for that symbol.

Examples

1.

REFERENCE SYMBOL	ELLE SYMBOL OPERATION							OPERATION PARAMETERS					
DATA NAME	,,	Ļē	V E L.	20	S NC	22	U _S	24	PICTURE.				
0	ρ	R	F	X									
E.M.P.N.Ø.	8	5	S						2				
NAME	В	5	S		Г				1.0				

Example 1 shows a region prefixed by a zero, an unprefixed region.

REFERENCE SYMBOL	OPERATIO	ON	OPERATION PARAMETERS					
DATA NAME 9 16	17 LEVEL SYN	U _S E 24	PICTURE 25					
X	PRFX							
EMPNO.	EQU		SEMPNE OF SEMPNE					
RATE	E 0.U		Z.S.RATE					
TØTPAY	LSB		4					
GRØPAY	LSB		4					
FICA	DECD		4800000					
START.	LDD		RATE.					
			. 3 <u> </u>					
	LDD		E.M.P.N.Ø.					

Example 2 shows a region prefixed by X and a method of communicating with other regions using the EQU.

3.

F		
IZ	PRFX	<u> </u>
EMPN.	U.03	SEMPNO OSEMPNO
NAME	E Q U	IS.E.M.P.N.X . D.S.E.M.P.N.X
TØTPAY	E Q U	X, S, T, & T, P, A, Y
GROPAY	£Qυ	X.S.G.R.D.P.A.Y
RATE	!LSB	2
START	LDD	T. Ø. T. P. A. Y.
	ADD	RATE
	STD	GRØPAY
	LDD	EMPNØ
i		The state of the s

Example 3 shows a region prefixed by a Z and a method for communicating with other regions using the EQU.

4.

PRFX
LDD X\$TØTPAY
ADD Z\$RATE
STD X\$GRØPAY
LDD \$EMPNØ D\$EMPNØ

Example 4 shows a region prefixed by Y and a method for communicating with other regions without using the EQU.

TCD

Function

TCD indicates that an interruption in the loading process is desired so that the programmer may execute some instructions in his program and then return control to the loader.

Two functions are performed by TCD.

- 1. It terminates the assembly of this segment.
- 2. It prepares the loader transfer card (TCD-Type 60).

Format

TCD must contain

- TCD as the Operation
- An expression in the Operation Parameters for the address to which control will be transferred during loading.

Assembler Action

When the assembler encounters a TCD it completes the processing and punching of all previous instructions and punches a transfer card to the address indicated in the Operation Parameters.

Example



Note

Control must be returned to the loader by the object program. This is accomplished by an unconditional branch to zero on index register 6 (BRU 0.6).

Restriction

The expression in the Operation Parameters may contain one external global symbol. In addition to the restrictions placed on external global symbols in DXG page 1-79, the external global must be either preceded by a plus sign or the plus sign must be implied.

Care must be exercised if this external global symbol feature is used. When the object program receives control, the programmer must be sure that all necessary subroutines are in memory and that all required external global symbols have been defined.

END OF PROGRAM	

Function

The END pseudo-operation terminates the assembly of this segment and prepares an END transfer card for the loader (card type 70).

Format

END may contain

- END as the Operation
- An expression or blanks in the Operation Parameters.

Assembler Action

When the END is encountered in Pass II, the Operation Parameters is scanned. If the field is blank, no transfer control card is produced. If the field contains an expression, the expression is evaluated and the resulting address is placed in the transfer control card.

1-137 12/6/63

Examples

REFERENCE SYMBOL				OPERATION [OPERATION PARAMETERS																		
	DA	TA	NA	ME		16	17	17 18 19 20 C 22 E 24 25		LEVEL 20 YN C 22 E 24 25		PICTURE 4 25				PICTURE											
M.A. L.N.P.R.Ø.G			L	۵	DIS						G	R	ø	P	Α	·γ				_			_				
	_		ш.			 _	┖		L	<u> </u>	┖	<u> </u>			L.	L		.	ı	<u>. </u>	٠						
				-		1	L	1		ļ	L.	<u> </u>	<u> </u>	_	L_		.	-						٠	_1		_
			L				L				L	L			L										1		_
		DA	DATA	DATA NA	DATA NAME	DATA NAME	DATA NAME	DATA NAME 16 17	DATA NAME 16 17 18		DATA NAME 16 17 LEVEL 20	DATA NAME 16 17 LEVEL 20 VNC	DATA NAME 16 17 LEVEL 5 YN C 22	DATA NAME 16 17 LEVEL 20 F N C 22 E	DATA NAME 16 17 16 19 20 V C 22 E 24	DATA NAME 16 17 16 19 20 V C 22 E 24 25	DATA NAME 16 17 16 19 20 V C 22 E 24 25	DATA NAME 16 17 16 19 20 VS VN C 22 E 24 25	DATA NAME 16 17 16 19 20 VS VN C 22 E 24 25	DATA NAME 16 17 18 19 20 SYN C 22 E 24 25	DATA NAME 16 17 18 19 20 S Y N C 22 E 24 25	DATA NAME 16 17 18 19 20 VS VN C 22 E 24 25 PI	DATA NAME 16 17 16 19 20 VS VN C 22 E 24 25 PICTO	DATA NAME 16 17 18 19 20 E 24 25 PICTURE	DATA NAME 16 17 18 19 20 V C 22 E 24 25 PICTURE	DATA NAME 16 17 18 19 20 VS VN C 22 E 24 25 PICTURE	DATA NAME 16 17 18 19 20 N C 22 E 24 25 PICTURE

Example 1 indicates the assembly of a main segment. A transfer of control to symbolic location MAINPROG is effected at the end of the loading process.

2.												
			1		 1-1	ш				 		 -
	S.T.A.R	T.S.E	G	LDS		P.	A.	Y		 		 ì.
				1.						 		
		1 (1				Γ			1	 		
				E N.D		Ι.				 	1 1	

Example 2 indicates the assembly of a subsegment. There will be no address in the transfer control card produced.

Note

END

When an expression is present in the Operation Parameters, it may contain one external global symbol. In addition to the restrictions placed on external global symbols on page 1-79 , the external global must be either preceded by a plus sign or the plus sign must be implied.

1-138 12/6/63

TABLES

A.ALPHABETIC LISTING - COMPUTER INSTRUCTION REPERTOIRE

Mn	emonic Code	*	Computer Operation	Octal
	ABA	(2A)	Accept Buffer Address	32 x
	ABM	(2A)	Add Binary to Memory	34
	ABX	(2A)	Add Binary to Index	34
	ADD		Add Decimal Double	51
	ADQ		Add Decimal Quadruple	53
	ADS		Add Decimal Single	50
	ADT		Add Decimal Triple	52
	AIM	(2A)	Add Immediate to Memory	33
	AIX	(2A)	Add Immediate to Index	33
	AMD		Add to Memory Double	55
	AMQ		Add to Memory Quadruple	57
	AMS		Add to Memory Single	54
	AMT		Add to Memory Triple	56
	ANM	(2A)	AND to Memory	24
	ANX	(2A)	AND to Index	24
	BCTR		Branch Counter	
	BRC	(2A)	Branch on Count	16
	BRE		Branch if Equal	13
	BRG		Branch if Greater	12
	BRL		Branch if Less	14
	BRM		Branch if Minus	11

^{* 2}A in this column indicates a two-address instruction.

Mnemonic Code	*	Computer Operation	Octa	1
BRU		Branch Unconditionally	10	
BRZ		Branch if Zero	15	
BSF	(2A)	Backspace one File	47	x
BSR	(2A)	Backspace one Record	4 6	x
BXC	(2A)	Branch on Index Count	16	
CAA		Compare Alphanumeric Accumulate to Memory	or 03	
CDA		Compare Decimal Accumulator to Memory	02	
CMI	(2A)	Compare Memory to Immediate	01	
CMM	(2A)	Compare Second to first Memory	04	
CPO	(2A)	Central Processor Operation	67	
CPR	(2A)	Compare	57	x
CXI	(2A)	Compare Index to Immediate	01	
CXM	(2A)	Compare Index to Memory	04	
DCWC		Data Control Word Character Count		
DCWW		Data Control Word Word Count		
EDT		Edit	05	
ERS	(2A)	Erase	54	x
EXP		Explode	20	

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

^{* 2}A in this column indicates a two-address instruction.

x An x following an Octal Code indicates that the octal value is a command code, not an opera on code.

Mnemonic Code	<u>*</u>	Computer Operation	Octal	
FDOC	(2A)	Feed Document	41	x
FSF	(2A)	Forward Space one File	45	x
FSR	(2A)	Forward Space one Record	44	x
GEN	(2A)	General	07	
HLT		Halt	00	
IAW		Indirect Address Word	00	
IMP		Implode	21	
LAL		Load Accumulator Location and	36	
1.000	(0.4.)	Length	36	
LBFC	(2A)	Load Buffer for Compare		х
LBT		Low Bit Test	22	
LDD		Load Double	41	
LDQ		Load Quadruple	43	
LDS		Load Single	40	
LDT		Load Triple	42	
LDX	(2A)	Load Index	30	
LNK	(2A)	Link	75	x
LPW		List Pointer Word		
LXI	(2A)	Load Index with Immediate	31	
MCTR		Move Counter		
MFI	(2A)	Move From Immediate	31	

^{* 2}A in this column indicates a two-address instruction.

Mnemonic Code	*	Computer Operation	Octal	
MFM	(2A)	Move from First Memory	30	
MOV	(2A)	Move	06	
мта	(2A)	Move to first Address Field	32	
MVDT	(2A)	Move Data	37	x
MXC	(2A)	Move on Index Control	06	
O		Operand	00	
OL		Operand Link	02	
OP		Operand Pointer	01	
PCB	(2A)	Punch Card Binary	11	x
PCD	(2A)	Punch Card Decimal	12	x
PCE	(2A)	Punch Card in Edited Mode	13	x
PDOC	(2A)	Pocket Document	43	x
PDT	(2A)	Punch Double character mode Tape	13	x
PET	(2A)	Punch Edited Tape - system mode	31	x
PPT	(2A)	Punch Paper Tape	11	x
PRE	(2A)	Print in the Edit mode (data controls slewing)	30	x
PRED	(2A)	Print in the Edit mode - slew Double line	32	x
PRES	(2A)	Print in the Edit mode - slew Single line	31	x
PRET	(2A)	Print in the Edit mode to Top of page	33	х
PRN	(2A)	Print in the Non-edited mode - slew no lines	10	х

^{* 2}A in this column indicates a two-address instruction.

 $[\]boldsymbol{x}$ An \boldsymbol{x} following an Octal Code indicates that the octal value is a command code, not an operation code.

 $[\]boldsymbol{x}$ An \boldsymbol{x} following an Octal Code indicates that the octal value in a command code, not an operation code.

Mnemonic Code	<u>*</u>	Computer Operation	Octa	<u>.1</u>
PRND	(2A)	Print in the Non-edited mode - slew Double line	12	х
PRNT	(2A)	Print in the Non-edited mode - slew to Top of page	13	x
PRNS	(2A)	Print in the Non-edited mode - slew Single line	11	x
PST	(2A)	Punch Single character mode Tape	16	x
РХВ	(2A)	Program counter to Index and Branch	17	
RALD		Reset Accumulator Length Double	22	
RALQ		Reset Accumulator Length Quadruple	22	
RALS		Reset Accumulator Length Single	22	
RALT		Reset Accumulator Length Triple	22	
RB	(2A)	Read Buffer	24	x
RCB	(2A)	Read Card Binary	01	x
RCD	(2A)	Read Card Decimal	02	x
RCM	(2A)	Read Card Mixed mode	03	x
RDOC	(2A)	Read Document	01	x
$\mathbf{R}\mathbf{F}$	(2A)	Read File	54	x
RFCR	(2A)	Read File Continuous and Release Seek	25	x
RFI	(2A)	Read File and Increment Address	56	x
RFR	(2A)	Read File and Release seek	55	x
RIM	(2A)	OR Inclusive to Memory	23	
RIX	(2A)	OR Inclusive to Index	23	

^{* 2}A in this column indicates a two-address instruction.

Mne	monic Code	*	Computer Operation	Octa	<u>al</u>
	RLDA		Rotate Left Double Alpha	22	
	RLDAS		Rotate Left Double Alpha and Set	22	
	RLDD		Rotate Left Double Decimal	22	
	RLDDS		Rotate Left Double Decimal and Set	22	
	RLQA		Rotate Left Quadruple Alpha	22	
	RLQAS		Rotate Left Quadruple Alpha and Set	22	
	RLQD		Rotate Left Quadruple Decimal	22	
	RLQDS		Rotate Left Quadruple Decimal and Set	22	
	RLSA		Rotate Left Single Alpha	22	
	RLSAS		Rotate Left Single Alpha and Set	22	
	RLSD		Rotate Left Single Decimal	22	
	RLSDS		Rotate Left Single Decimal and Set	22	
	RLTA		Rotate Left Triple Alpha	22	
	RLTAS		Rotate Left Triple Alpha and Set	22	
	RLTD		Rotate Left Triple Decimal	22	
	RLTDS		Rotate Left Triple Decimal and Set	22	
	RPT	(2A)	Read Paper Tape	02	х
	RQSP	(2A)	Request Status of Processor	67	
	RQS	(2A)	Request Status	00	x
	RQST		Request Status of Typewriter	00	
	RRD		Rotate Right Double (binary)	22	
	RRDA		Rotate Right Double Alpha	22	

²A in this column indicates a two-address instruction.

12/6/63

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

Mnemonic Code	*	Computer Operation	Octal	
RRDAS		Rotate Right Double Alpha and Set	22	
RRDD		Rotate Right Double Decimal	22	
RRDDS		Rotate Right Double Decimal and Set	2 2	
RRDT		Rotate Right Double (binary) Test	22	
RRQA		Rotate Right Quadruple Alpha	22	
RRQAS		Rotate Right Quadruple Alpha and Set	22	
RRQD		Rotate Right Quadruple Decimal	22	
RRQDS		Rotate Right Quadruple Decimal and Set	22	
RRS		Rotate Right Single (binary)	22	
RRSA		Rotate Right Single Alpha	22	
RRSAS		Rotate Right Single Alpha and Set	2 2	
RRSD		Rotate Right Single Decimal	2 2	
RSDS		Rotate Right Single Decimal and Set	22	
RRST		Rotate Right Single (binary) Test	22	
RRTA		Rotate Right Triple Alpha	2 2	
RRTAS		Rotate Right Triple Alpha and Set	2 2	
RRTD		Rotate Right Triple Decimal	2 2	
RRTDS		Rotate Right Triple Decimal and Set	2 2	
RSS	(2A)	Reset Status	40	x
RSST	(2A)	Reset Status of Typewriter	40	x
RTB	(2A)	Read Tape Binary	05	x
RTD	(2A)	Read Tape Decimal	04	x

^{* 2}A in this column indicates a two-address instruction.

1-145

Mnemonic C	Code <u>*</u>	Computer Operation		_
RWD	(2A)	Rewind	70	x
RWS	(2A)	Rewind and Standby	72	x
RXM	(2A)	OR Exclusive to Memory	25	
RXX	(2A)	OR Exclusive to Index	25	
SAL		Store Accumulator Location and length	37	
SBM	(2A)	Subtract Binary from Memory	35	
SBX	(2A)	Subtract Binary from Index	35	
SCPR	(2A)	Seek/Compare	17	Х
SDD		Subtract Decimal Double	61	
\mathtt{SDL}	(2A)	Set Density Low	61	х
SDH	(2A)	Set Density High	60	х
SDQ		Subtract Decimal Quadruple	6 3	
SDS		Subtract Decimal Single	60	
SDT		Subtract Decimal Triple	62	
$_{ m SF}$	(2A)	Seek File	34	х
SLDA	Δ	Shift Left Double Alpha	22	
SLDA	\S	Shift Left Double Alpha and Set	22	
SLDD)	Shift Left Double Decimal	22	
SLDD	os	Shift Left Double Decimal and Set	22	
SLNK	(2A)	Seek/Link	35	X
SLQA	A	Shift Left Quadruple Alpha	22	
SLQA	AS	Shift Left Quadruple Alpha and Set	22	
SLQ)	Shift Left Quadruple Decimal	22	
SLQD	DS	Shift Left Quadruple Decimal and Set	22	

^{* 2}A in this column indicates a two-address instruction.

12/6/63

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

Mnemonic Code	*	Computer Operation	Octal	Mnemonic Code	<u>*</u>	Computer Operation	Octa	.1
SLSA		Shift Left Single Alpha	22	SRQDS		Shift Right Quadruple Decimal		
SLSAS		Shift Left Single Alpha and Set	22			and Set	22	
SLSD		Shift Left Single Decimal	22	SRS		Shift Right Single (binary)	22	
SLSDS		Shift Left Single Decimal and Set	22	SRSA		Shift Right Single Alpha	22	
SLTA		Shift Left Triple Alpha	22	SRSAS		Shift Right Single Alpha and Set	22	
SLTAS		Shift Left Triple Alpha and Set	22	SRSD		Shift Right Single Decimal	22	
SLTD		Shift Left Triple Decimal	22	SRSDS		Shift Right Single Decimal and Set	22	
SLTDS		Shift Left Triple Decimal and Set	22	SRST		Shift Right Single (binary) Test	22	
SPB	(2A)	Store Program counter and Branch		SRTA		Shift Right Triple Alpha	22	
SPRD	(2A)	Slew Printer Double line	62 x	SRTAS		Shift Right Triple Alpha and Set	22	
SPRS	(2A)	Slew Printer Single line	61 x	SRTD		Shift Right Triple Decimal	22	
SPRT	(2A)	Slew Printer to Top of page	63 x	SRTDS		Shift Right Triple Decimal and		
SRD		Shift Right Double (binary)	22	66.4	(- 4)	Set	22	
SRDA		Shift Right Double Alpha	22	SSA	(2A)	Set Status by ANDing	67	
SRDAS		Shift Right Double Alpha and Set	22	SSL	(2A)	Set Status by Loading	67	
SRDDS		Shift Right Double Decimal and	22	SSO	(2A)	Set Status by ORing	67	
		Set	22	STD		Store Double	45	
SRDT		Shift Right Double (binary) Test	22	STQ		Store Quadruple	47	
SRF	(2A)	Seek/Read File	14 x	STS		Store Single	44	
SRFI	(2A)	Seek/Read File and Increment		STT		Store Triple	46	
		address	16 x	SWF	(2A)	Seek/Write File	10	х
SRFR	(2A)	Seek/Read File and Release seek	15 x	SWFI	(2A)	Seek/Write File and Increment		
SRQA		Shift Right Quadruple Alpha	22	SWFR	(0.4)	address	12	
SRQAS		Shift Right Quadruple Alpha and			(2A)	Seek/Write File and Release seek	11	
SPOD		Set	22	SWFV	(2A)	Seek/Write File and Verify	13	Х
SRQD		Shift Right Quadruple Decimal	22	SXA	(2A)	Store Index in Address field	32	

^{* 2}A in this column indicates a two-address instruction.

1-148

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

^{* 2}A in this column indicates a two-address instruction.

x An x following an Octal Code indicates that the octal value is a command code, not an operation code.

B. ALPHABETIC LISTING - PSEUDO OPERATIONS

Manual

TIA (2A) Type Input - Alphanumeric 03 TIO (2A) Type Input - Octal 01	x x
TIO (2A) Type Input - Octal 01	-
TOA (2A) Type Output - Alphanumeric 13	х
TOO (2A) Type Output - Octal 11	x
VLD Variable Length Divide 26	
VLM Variable Length Multiply 27	
WB (2A) Write Buffer 30	x
WEF (2A) Write End of File 55	x
WF (2A) Write File 50	x
WFCF	
WFCR (2A) Write File Continuous and Release seek 31	x
WFCV (2A) Write File Continuous, Verify and Release seek 33	x
WFI (2A) Write File and Increment address 52	x
WFR (2A) Write File and Release seek 51	x
WFV (2A) Write File and Verify 53	x
WTB (2A) Write Tape Binary 15	x
WTD (2A) Write Tape Decimal 14	x
X Index 00	
XL Index Link 02	
XP Index Pointer 01	
X* Index Indirect 10	
XL* Index Link Indirect 12	
XP* Index Pointer Indirect 11	

^{* 2}A in this column indicates a two-address instruction.

 ${\bf x}$ An ${\bf x}$ following an Octal Code indicates that the octal value is a command code, not an operation code.

Mnemonic Code	Page	Pseudo Operation
ACUM	1-108	Accumulator
AN	1-116	Alphanumeric constant
ARP	1-104	Accumulator Reference Point
ARPL	1-106	Accumulator Reference Point in the Loader area
BPS	1-96	Block Preceded by Symbol
BPSL	1-98	Block Preceded by Symbol in the Loader area
BSS	1-92	Block Started by Symbol
BSSL	1-94	Block Started by Symbol in the Loader area
CALL	1-87	Call segment at load time
DECD	1-114	Decimal Constant Double word
DECQ	1-114	Decimal Constant Quadruple word
DECX	1-114	Decimal Constant Single word
DECT	1-114	Decimal Constant Triple word
DGR	1-83	Define Global Reference
DGRE	1-82	Define Global Reference Ends
DGRR	1-79	Define Global Reference Remotely
DIG	1-75	Define Internal Global symbols
DXG	1-77	Define External Global symbols
END	1-137	End of program
EJT	1-123	Eject page

1-151 12/6/63

12/6/63

Mnemonic Code	Manual Page	Pseudo Operation
EQU	1-130	Equals
EQUG	1-85	Equals Global symbol
EQUO	1-132	Equals Octal
FILL	1 - 1 1 1	Fill word
FULL	1-125	Full binary cards
IDEN	1-124	Identify binary output
INCL	1-89	Include Library segment
INCS	1-90	Include Symbolic segment
LSAN	1-118	Last Symbol Alphanumeric
LSB	1-100	Last Symbol in Block
LSBL	1-120	Last Symbol in Block in the Loader area
OCTD	1-120	Octal Constant Double word
OCTQ	1-120	Octal Constant Quadruple word
OCTS	1-120	Octal Constant Single word
OCTT	1-120	Octal Constant Triple word
ORG	1-127	Origin
ORGO	1-129	Origin Octal
PRFX	1-133	Prefix
SGMT	1-73	Segment Name
SYAL	1-126	Symbolic Analyzer
TCD	1-136	Transfer Control Card
TTL	1-122	Title

SUPPLEMENT TO CHAPTER IV

The possibility of generating relocation errors was discussed in Chapter IV. In that Chapter, expressions containing only the plus (+) and minus (-) operators were considered. On occasion, the programmer may wish to employ the multiplication (*) and division (/) operators and have the assembler calculate these more complex relationships. Because this is a more complex environment, it is natural to expect that the assembler's procedure for detecting relocation errors is more complex. However, it is essential that the programmer understand this procedure if errors are to be avoided during coding.

The procedure which follows is mainly applicable to complex expressions; however, it is general enough to be used on simple expressions, too. This stems from the fact that the assembler itself has only one procedure for detecting relocation errors.

The example which follows is a contrived, unreal vehicle, intended only to portray many aspects of relocation errors.

Preliminary Procedure

Step 1 Partition the expression into simple and complex groups. A simple group is one element (a symbolic reference or an absolute value) which is connected to the expression by either the plus or minus operator. A complex group contains two or more elements (symbolic references and/or absolute values) that are connected to each other by either the multiplication or division operators. The complex group, itself, is connected to the rest of the expression by either the plus or minus sign.

Therefore, in the expression

A*B/C-D-5*E*F+16+22*G

A*B/C	Constitutes a complex group,
D	Constitutes a simple group (an element),
5*E*F	Constitutes a complex group,
16	Constitutes a simple group (an element) and
22*G	Constitutes a complex group.

Step 2 Each simple group (element) is examined. If the group (element) is absolute it will have no effect on the expression and is therefore dropped from consideration. Thus, in the

1-153

previous example, the absolute value 16 is immediately ignored. If D represents an absolute symbolic reference, it is also ignored; however, if it represents a symbolic reference, it is retained.

Step 3 Each complex group is examined.

- a) If each element in the group is absolute it will have no effect on the expression and is therefore dropped from consideration. Thus, in the example above, if symbolic references A, B and C were all absolute, the complex group A*B/C would be ignored.
- b) If there is more than one relocatable symbolic reference in the complex group, the entire expression is meaningless and a relocation error is flagged on the assembly listing. Thus, in the previous example, if symbolic references E and F were both relocatable, the entire expression is in error.
- c) If the complex group contains one relocatable symbolic reference and the division operator appears anywhere in the group, the entire expression is meaningless and a relocation error is flagged on the assembly listing. Thus, in the example above, if symbolic reference A was relocatable, the entire expression is in error because of the presence of the division operator.

Substitution Procedure

The substitution procedure is performed if no errors were discovered in the preliminary procedure.

a) Each absolute symbolic reference is replaced by its absolute value. Thus, in the previous example, if the symbolic reference F has a value of 4, the result of the absolute substitution would be:

-D-5*E*4+22*G, or -D-20*E+22*G

Note:

The absolute value 16 has been dropped from consideration during the preliminary procedure and, in order to continue the example, complex group A*B/C is assumed to contain only absolute elements.

b) Each relocatable symbolic reference is replaced by the letter M since it will be moved by the loader. If symbolic references D, E, and G are assumed to be relocatable, the resulting expression will be in terms of one unknown, M:

-M-20*M+22M

Final Determination

After like terms are combined according to the rules of mathematics, the result will indicate one of three possible conditions:

- if the result is 0 or nothing, the address is absolute and cannot be moved.
- 2. if the result is M, the address is relocatable and must be moved.
- if the result is anything else, the address is a relocation error.

In the example, then, the result of combining like terms produces:

22M - 21M or

M

This result indicates that the original expression is relocatable.

CONTENTS

I.	INTRODUCTION	2- 1
	System Configuration	2- 2
II.	THE SOURCE LANGUAGE	2- 5
III.	PROGRAMMING FORM FIELDS	2- 7
IV.	IDENTIFICATION DIVISION	2-11
	Identification Division Entries SEGMENT. AUTHOR. TITLE IDEN. Example of a Typical Identification Division.	2-11 2-13 2-13 2-13 2-14 2-14
v.	ENVIRONMENT DIVISION	2-15
	Environment Division Entries SRMETHOD MEMSIZE DEFCHKPT DEFOPEN DEFCLOSE DEFECR DEFECR DEFSCHED Example of a Typical Environment Division	2-15 2-16 2-16 2-17 2-18 2-18 2-19
VI.	DATA DIVISION	2-21
	Writing the Data Division Sequence Data Name Level Picture Value	2-21 2-21 2-22 2-23 2-24 2-29

	Sync	2-30
	Type	2-32
	Use	2-35
	Occurs	2-38
	Iden	2-39
	Data Division Sections	2-40
	File Section	2-40
	File Section Conventions	2-48
	Working Storage Section	2-49
	Working Storage Section Conventions	2-49
	Memory Allocation	2-52
	Memory miscaulon	
VII.	PROCEDURE DIVISION	2-57
	Writing the Procedure Division	2-57
	Sequence	2-5 8
	Type	2-58
	Reference Symbol	2- 59
	Operation	2-60
	Operation Parameters	2-60
	Iden	2-61
	Use of Operation Parameters	2-61
	Macro Instruction Descriptions	2-65
	Input/Output Operations	2-68
	READ	2-68
	WRITE	2-69
	WRITEX	2 - 71
	OPEN	2-72
	CLOSE	2-73
	TYPE	2-74
	Extended Input/Output System Service	
	Macro-Instructions	2-74
	Data Movement	2-77
	MOVE	2-77
	LOAD	2-80
	UNLOAD	2-81
	Procedure Control Operations	2-83
	COMPARE	2-83
	IF	2-85
	SETSW	2-86
	GOTO	2-87
	GOTO DEPENDING ON	2-88
	HALT	2-88
	EOJ	2-89
	ABORT	2-90

2-ii 12/6/63

	Arithmetic Operations Add and Round ADDR Add and Truncate ADDT Subtract and Round SUBR Subtract and Truncate SUBT Multiply and Round MPYR Multiply and Truncate MPYT Divide and Round DIVR Divide and Truncate DIVT	2-91 2-92 2-92 2-93 2-94 2-95 2-96 2-97
VIII.	SEGMENTED PROGRAMS	2-97
	Internal and External Global Symbols Define Internal Global DIG Define External Global DXG Common Data Areas AREADEF Define Global Reference Remote DGRR Define Global Reference Ends DGRE Define Global Reference DGR	2-99 2-99 2-100 2-101 2-102 2-102 2-103 2-103
IX.	SAMPLE PROGRAM	2-105
Alp	phabetic Table of Macro Instructions	2-111

I. INTRODUCTION

The Macro Assembly Program offers a macro source language and the means for translating this language into the instruction format of the GE-425/435 computer. In the basic assembly language, a code exists for each operation which the computer can execute. The macro assembly language provides instructions which produce a series of these operation codes. The macro language performs operations upon data described in a field-oriented manner and is a logical extension to the basic assembly language.

Macro-instructions simplify the problem of programming, because they eliminate many of the tedious coding tasks. These instructions relieve the programmer of such details as masking and aligning data. In this way, the opportunity for introducing errors into the source program is greatly reduced. The programmer may choose to write his entire program in the macro assembly language, or he may combine basic assembly operations with macro operations.

The macro assembly language provides instructions which will accomplish the following functions:

Input/output operations

Arithmetic operations

Data movement

Procedure control

Additional macro-instructions may be added to the source language. Generator(s) required to process the new macro instruction(s) may be written by the customer in either the macro or basic assembly languages, or may be added to the Macro Assembly Program by means of a library routine.

The Macro Assembly Program achieves the following objectives. The

program includes:

A minimum but adequate set of field-oriented macro-instructions

All machine and pseudo-operations available in the basic assembly language

A detached Data Division with a fixed format

A means for making the automatic input/output routines available

A source language that is easily used by the programmer and easily assembled

An Identification Division that provides adequate source program documentation.

The Macro Assembly Program has the following capabilities. The program:

Performs a straight basic assembly without executing unnecessary phases of the Macro Assembly Program

Allows new macro instructions to be added to the system

Is upward compatible with future GE-400 series computers

Provides extensive error analysis of the source language

Performs a rapid assembly of source programs.

SYSTEM CONFIGURATION

The Macro Assembly Program operates on the following configuration:

GE-425/435 computer with 8K memory

Magnetic tape control unit

Four tape handlers

Printer

Card reader

Card punch

Additional magnetic tapes may be substituted for the card reader, card punch, or the printer. The Macro Assembly Program takes advantage of the 16K or 32K core memory options as illustrated in Figure I-1 below:

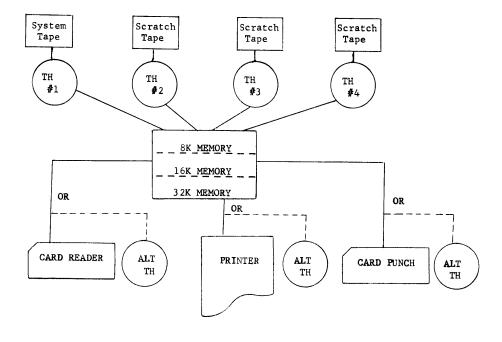


Figure I-1

II. THE SOURCE LANGUAGE

The Macro Assembly Program source language resembles English grammar and can be readily understood. It is composed of certain combinations of characters or "words" which, when written according to given conventions, describe an information processing function.

The words of the macro assembly language are made up of characters from the standard GE character set. This includes:

 Alphabetics
 A.....Z

 Numerics
 0, 1...9

plus the tilde (\sim). There must be no imbedded blanks.

When writing in the macro assembly language, there are two types of words which must be considered. First, there are the macro instructions, which generate the actual computer operations. Second, there are the words with which a programmer describes his data or allocates memory for the location of this data. These words are written by the programmer in a manner which is easily understood, but which of necessity must follow certain conventions. These will be discussed later in this section of the manual. The types of words used in the source language are illustrated below. The underlined words are the macro-instructions.

START OPEN INPUT; PAY~MSTR; TIMECDS

READ PAY~MSTR

A Macro Assembly Source Program consists of four divisions:

IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION

PROCEDURE DIVISION

The function of each of these divisions is discussed briefly in this chapter $t \in present$ an all-over view of the source language.

In the Identification Division the programmer identifies his program and labels the outputs of the assembly.

In the Environment Division, the programmer describes the characteristics of the object computer and specifies the major control routines required by the Extended Input/Output System.

In the Data Division, the programmer describes the format of the data files and allocates memory for file areas, working storage, and constants.

In the Procedure Division, the programmer writes macro-instructions and/or basic assembly language operations and pseudo-operations which define the functions to be performed. The instructions in this division operate on the data described in the Data Division.

In the section describing the basic assembly language, the concept of relocatable segments was discussed. It is possible in writing in the macro assembly language, to produce relocatable segments for use within the GE-425/435 operating system environment. Each segment written must be named in the Identification Division and be specified as to whether it is to be a relocatable or absolute assembly. More details describing the communication between segments is discussed in Chapter VIII, Segmented Programs.

2-6

III. PROGRAMMING FORM FIELDS

The macro assembly language uses both levels of the dual headings on the GE-425/435 Programming Form. This chapter discusses the placement of the four divisions of the source language on this form, and defines the fields used where necessary. The sample coding in Figure III-lis shown merely to illustrate placement; the various entries are discussed in detail in the Chapters describing the four divisions.

IDENTIFICATION∆DIVISION is written into columns 8-30. Each entry within this division starts in column 17. Descriptions of any of the options which may be used with each entry are written in the Operation Parameters, starting in column 25. Comments may be written into the Identification Division by inserting an * in column 7.

ENVIRONMENTADIVISION is written into columns 8-27. Each entry starts in column 17. Parameters for the individual entry are written in the Operation Parameters, starting in column 25.

DATA DIVISION is written in columns 8-20. This division uses the lower level of headings across the programming form. A brief definition of each of these fields follows:

Sequence (Cols. 1-6)

The Sequence number is provided as a convenient method for the programmer to maintain an order within his source deck.

Type (Col. 7)

This column describes the use of the line in which it appears.

Data Name (Cols. 9-16)

The data name is the name a programmer assigns to an entry in the Data Division.

Level (Cols. 18-19)

With a Level entry, the programmer defines the various levels of a logical record and indicates unrelated items in working storage.

GE-425/435 PROGRAMMING FORM

1	PAY	ROLL SAMPLE	PROBL	E M		PROGRAMMER	ARLIE B	8 தம் ப			DAŤE	PAGE	⊅F
	Ţ	REFERENCE SYMBOL		RATION	1			OPERA ON	AHAMETERS				
SEQUENCE	0	ļ	+	+,,,,									1
	-	DATA NAME	LEVE	y Us	1	PI^ - URI	-						10
	7 8	19 1	6 17 18 19 .		24 28		t.	40 AT CUPS	44 45		V A ⊆ U €.		, ,
0,0,0,1,0	LI	D.E.N.T.I.F.I.	ATI	DAND	VISI	Ø.N		1 1 1 1					
0,0,0,2,0			SEG	MENT	P.A.Y.~	0.0.1.							
0.0.0.3.0			TIT	1 8	P.A.Y.R	19 L.L. S	AMPIE	PRAB	L.E.M.		*	*	
اه ، ۲ ، ۵ ، ۵ ، ۵ ،			IDE	N	P.A. Y. 1				L; C. 1111		4-	en de la casa de la ca	
0.0.0.5.0	3	N.V.I.R. O.N.M.E			SILON				* * * * * * * * * * * * * * * * * * * *		the same transfer and the same transfer and the		
0, 0, 0, 6, 0		The second second	SRM		D C.A.L.L	· · · · · · · · · · · · · · · · · · ·	+ + + +					ياؤر بطب الفيد الأاليا المفاسية ببدلا	
0,0,0.7.0			DEF		E. Ø. R. I		 		+++++++				
				CIPIT	LEINER I					+			
		1 4 1 1 1 1 1 1 1 1 1	+ +						1 1 3		<u> </u>		
	i		+	-+					لتنا للاسلامة وا		4		
			+	-					· - 114				
			1	-,1	+				عادات سامانا				
0.0 1.2.0		I.V. L.C. A. A. T.A	5101	V			++-+-						
0.6.1.3.0		J LIE I SIEICIT						Andrew Control				E. Amerika kan talah dan dari	
0.0.1.4.0	•	PAY MISTR	L F D				1 1 1 1 1				. 4 L 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	In all and the second second	A A
			DEV	I C E	0.4.0.3							4 4 g i. g. t	1
and the second									· · · · · · · · · · · · · · · · · · ·		+ 	a	·
							****		·		الربطان فالتمام فالمتوافية فاستفتاه	المالية المالية المستحددة	
					1					- 1 		A	a
			1		1			4-4-4-4	+			لتوعفيتوا بشباه الغشاب	
0.0.2.5.0		PIALY R.E.C.	+ : : 	-							بيستان المساف المساف المساف المساف	4 - 4 - 4 - 4 - 4 - 4 - 4 - 4	
0 0.2.6 0		P.A.Y.~ N.Ø.	Ø-T		101/5		+					4 : 20: 4 : 1 : 1 : 1 : 1 : 1 : 1 : 1 : 1 : 1	
		F. (PL) 9 . T 1V . 92	_0.2	1	19.(.5.)		44444444444444444444444444444						L .
L	1	e a ne man a	· • • • • • • • • • • • • • • • • • • •					4-4	. 		 		
		الجداله المستناف المالات	\$ · · · · ·	-1					i		4-1		
	J	المسطيع عداليا يال	ير فيعتم أي	1					+				
TO: 0' ATO: 0		DE DIVE T	0.2				+ + + + + + + + + + + + + + + + + + + +	<u> b </u>					
0. 0.4_1.0		D.C.Ø D.E	0.3	R	9.9							A court and the all	1
	,		1 :			1 1			·		A	يها فالتابية الهياها	A 40 -4
0.0.9 0.01		Ø R K I N G S	TORRA	AGE.	E.C.T.I	Ø.N.			+		American de la companya de la companya de la companya de la companya de la companya de la companya de la compa		
0 0.9.1 0		CLASTNM	0.2	-	A. (. 1 . 6)		***			horacomorphic be and and a co	للفلية عاماله فالك	
0 0 9 2 6	Í		. 02		A.A.A.						سيرق المتاسسين فالأناف الاستادات الأرادا	يالف الوصيطية فيستطيب فيالكناناة	
				1	14111111				Δ.Δ.Δ.	رريقا سيد القالد القال الما	د	رابي الأرابية التقالية القالبية	فتا أبا المنابة
	1		1	1	1			4. -4			لليبوات ويستفيسها يسود بهالطساء للماساء المعا	يريا وعبيا والمستواد التفاسيدات	ما أيا ساد
0,1,0,10		S.W.1.	01	S			+		للمان فالسلام الأراب		الراغان الإراط والطالب والمطالب والمطالب		4 4 4 4
21.1.1.2.1.1.4	· · · · · †	9w. 1 1	, i.u	ـ د			+	+	0	للساب والمراجعين فالساب	الرازان والرافي فيساف بالقائد فالمساملة بالقا	and the state of the	ı
				+++									
				+									I
	··· · · - - 			1							*		
	- 🕌		4	1	1 +		 				t in the second		I tank k
0.1.0.4.0	_ P	RACEDURE	ADI \ Ø.P.E.N R.E.A.L	V 1 S 1								للماء المسلم المسامية المسلمانية	1 -11
0.1.0.5.0	i	START	D.P.E.N	VI	IP.A.Y.M	S.T.R .: .T	I M.E.C.D	5 . Q . P.A Y	MSTR	AVCHEC	. L		+ +
0.1.0.6.0		READM.	REAL		PAYM	S.T.R.		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			4.404 - 4 4 4 4 4 4		·
01070		R. & . A. D. T. C.	I F				T. Ø. T.E	ST	4	-4 -4-4-4-4-4-	*		فاستأساه
	•		1					i			والمستقالة والمستقالة المستقالة المستقالة المستقالة المستقالة المستقالة المستقالة المستقالة المستقالة المستقالة		4 k
				*: 4: 12 15	4 - 4 - 4 - 4	الطلمية كالمتا	ALL IN THE OTHER PROPERTY.		*		hard the first and the first section of	and the second second	, I ,
			•										

Figure III-1. Sample Programming Form

Sync (Synchronization) (Col. 21)

The Sync entry specifies the positioning of elementary items within a computer word or words.

Use (Col. 23)

The Use column specifies how a data item is to be used in memory.

Picture (Cols. 25-40)

The Picture field describes the mode, size, decimal point location, and editing characteristics of the data name entry.

Occurs (Cols. 41-44)

The entry in the Occurs columns indicates the number of times an item is to be repeated.

Value (Cols. 45-76)

The Value columns specify an initial value for data items in the Working Storage Section.

Iden (Identification) (Cols. 77-80)

The Iden columns associate the individual card with a source program deck.

PROCEDURE \triangle DIVISION is written in columns 8-25. This division uses the upper level of headings plus those headings which apply to both Procedure and Data Divisions. A brief definition of these fields follows:

Type (Col. 7)

The Type column is used to indicate a comment or continuation line.

Reference Symbol (Cols. 9-16)

The Reference Symbol field is used to assign symbols to the instructions and pseudo-operations written in the Procedure Division.

Operation (Cols. 17-24)

The Operation entry is a mnemonic code: a macro instruction, instruction irom the GE-425/435 instruction repertoire, or a pseudo-operation.

Operation Parameters (Cols. 25-76)

The parameters entered into the Operation Parameter columns complete the function indicated by the Operation.

Iden (Cols. 77-80) -- same use as in Data Division.

IV. IDENTIFICATION DIVISION

In the Identification Division the programmer identifies the source program, labels the output from the assembly, and specifies an absolute or a relocatable assembly. This division is written first on the GE-425/435 Programming Form.

Any Macro Assembly Program begins with IDENTIFICATION DIVISION written into columns 8-30. The only required entry within the division is SEGMENT. The remaining entries are optional. Comments may be included in the Identification Division by inserting an * in column 7, and then writing the comments in columns 8-80.

IDENTIFICATION DIVISION ENTRIES

The four permissible entries in the Identification Division are

SEGMENT

AUTHOR

TITLE

IDEN

Each entry is written starting in column 17. A description of the options which may be used with each entry is described in the following paragraphs. The options are written in the Operation Parameters columns starting in column 25.

SEGMENT		Required
		4

In the SEGMENT entry the programmer names his segments (programs). This entry allows the programmer to specify an absolute origin for the segment, thus giving him the ability to assemble in the absolute mode. This entry is required and must precede the other entries in the division.

Format

The format for the SEGMENT entry is as follows:

OPERATION OPERATION PARAMETERS

SEGMENT Segment name

Blanks

Expression

Absolute decimal value

The segment name should be a unique name which will be written on the output program header card. The segment name must follow the rules established for Reference Symbol on page 2-59.

The second parameter, which immediately follows the segment name should contain one of the following:

Blanks Blanks indicate the relocatable mode of assembly.
All instructions are assembled relative to zero; the assembly listing is relative to zero; relocatable errors are flagged.

Expression The expression must contain one external global symbol. The symbol must adhere to the general restrictions placed on external global symbols in the discussion of DXG in the Basic Assembly Language section, page 1-77. This condition indicates the re-

locatable mode of assembly.

The absolute value defines the origin of the segment and indicates the absolute mode of assembly. All instructions are assembled relative to the starting origin and the assembler produces an absolute listing. Relocatable errors are flagged for warning purposes only. The program may be moved by changing the starting address on the output origin card.

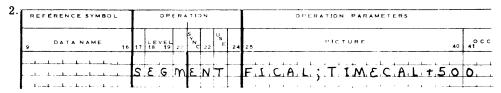
Examples

Absolute

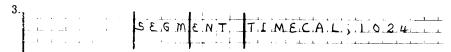
Decimal Value

1	_	RE	FE	EN	C F.	5 ٧	M Bit			Γ		0 =	FR	TIC	N			Γ				3 P I	F. ₩ Δ	٠.٥	٨.	FΔ	RAM	er to the	s	
	.,	7	С) A 1	Δ.	V A N	4 E		16	17		7 F L	20	S _X N _C	22	U _S E	24	.2 5						FI	ст	UP	F			
-	I	r D'E		<u></u>	 1.4		F.	L	د	A S	T E	I G	Ø	N E	A N	D T	1	ý	'.1 R	S	I.	Ø	N.	Y.Y	_1_			11.		i .

This example indicates that the name of the segment is GROSPAY. Since the segment name is followed by blanks, a relocatable assembly is indicated.



This example indicates that the name of the segment is FICAL. Since the segment name is followed by a symbolic reference (an external global reference), a relocatable assembly is indicated.



This example indicates that TIMECAL is to be used as the name of the segment. Since the segment name is followed by a decimal value, an absolute assembly is indicated.

AUTHOR Optional

The author's name may contain up to $30\ BCD$ characters and should be written starting in column 25.

The use of Title allows a heading line to be specified that will be printed at the top of each page of the assembly listing. The print line is written in columns 25-76.

IDEN	Optional
Address of the second s	

IDEN allows the programmer to specify the identification (columns 77-80) of the binary program cards. If used, the four identification characters written in columns 25-28 of this entry will be placed in columns 77-80 of the output deck. If IDEN is not supplied, columns 77-80 of the output deck will be blank

EXAMPLE OF A TYPICAL IDENTIFICATION DIVISION

ļ	[.]	1:					T	- 5	i ir.				ľ												000			,														
							li	- 1	1		J.	Ι.	l.															٠,	.]													
000000000000000000000000000000000000000	10	II)	ΕN	T. 1	. F.	1.€	ę.	ξŞ	, g	Ν ξ.,	Δ.E N.T	1,0	Ğ.	1.	<u>.</u>	L¢	. N	4	ì	1.0	o	٥.		ì	,								i		1							
0 1 0 0	4 0	-					f	I I	, L	É.	٠.	:	0.0	ĥ.;	r	1 0	1. L	Ä	Ť.,	u. N	.G	R.	ø	3.3		ř,	A.	, -	w	R	ť,	Τ.	ΤE	N	e e	ì	C	Ė.	6	. 3	i.	
0000	€ 5	: 2	n, in	έN	T	٤.	m.	ÃΫ		ë,	٤.	. ε		7	έ.,	ع نم	۵		1	N	۲.	ø.	L.S	'n	ı.N	s.		8	. 3	٥.									÷		1	
		i											١.				ď							ï	÷								i						,	i		
	1	1					1				÷				ì							L																		ì		

V. ENVIRONMENT DIVISION

The Environment Division enables the programmer to indicate the characteristics of the object computer and to specify the major control routines required by the Extended Input/Output System for execution of the object program. This division follows the Identification Division on the GE-425/435 Macro Assembly Program Programming Form.

The environment division must begin with a line that contains ENVIRONMENTADIVISION written in columns 8-27. The SRMETHOD entry is required for every Macro Assembly Program. If the SRMETHOD is not specified the CALL option will be assumed. All other entries in the Environment Division are optional.

ENVIRONMENT DIVISION ENTRIES

The following list of entries may be included in the Environment Division:

SRMETHOD

MEMSIZE

DEFCHKPT

DEFOPEN

DEFCLOSE

DEFEOR

DEFSCHED

The name of the entry is written starting in column 17. Parameters for the individual entries are written in the Operation Parameters columns starting in column 25. A detailed description of the entries and allowable parameters follows.

COMERNO	_
SRMETHOD	Required
	1tequii eu

The SRMETHOD entry defines the method by which any non-generated routine is to be incorporated in the object program. The parameters in the Operation Parameters columns are:

Non-generated routines are incorporated by INCL.
Include Library Segment (INCL) indicates that the
required segments are supplied from the library tape
at assembly time.

INCS
Non-generated routines are incorporated by INCS.
Include Symbolic Segment (INCS) will call routines
from the program library which are kept in symbolic
form and assemble them with the current segment.

The required non-generated routines will be incorporated by the loading program at execution time.

If CALL was specified, the latest version of each nongenerated routine will be incorporated in the program from the library tape when the program is loaded.

MEMSIZE Optional

The MEMSIZE entry specifies to the Macro Assembly Program the amount of memory available for this segment in the object program. Memory size should be written as a decimal number beginning in column 25.

DEFCHKPT Optional

The DEFCHKPT entry defines the checkpoint routine to be used. If this entry is omitted, it indicates that no checkpoint is to be taken.

There are two parameters, separated by a semicolon, required for this entry. The first parameter specifies the procedure to be performed after writing the checkpoint; the second specifies a magnetic tape unit upon which the checkpoint is to be written.

The entries for the first parameter are:

Continue processing after writing the checkpoint. PROCEED

Halt immediately after writing the checkpoint. Pro-HALT cessing will be continued when the operator depresses the "RUN" button. HALT is useful for programs that

process a large number of records on the card reader. card punch, or document handler. It allows the operator to "batch" these records to facilitate re-start.

The entries for the second parameter are

Any checkpoints will be written on the standard dump Blanks

tape (logical device 0402°). With this option, the semi-

colon is not required following the first parameter.

Where $04xx_8$ is the logical device number of a magnetic tape on which the checkpoints should be written. 04XX

The name of magnetic tape data file on which the check-FILENAME

points are written. FILENAME must be described

in the Data Division.

Any checkpoints written on data tapes are automatically by-passed by the Extended Input/Output System when those tapes are used as input.

DEFOPEN	Optional

If the DEFOPEN entry is included, it will incorporate the OPEN routine according to the SRMETHOD specified. NON~OVERLAY must be written in the Operation Parameters field starting in column 25.

Optional DEFCLOSE

When the DEFCLOSE entry is supplied the Macro Assembly Program will generate the INCL, INCS, or CALL for the CLOSE routine. CLOSE1 must be written in the Operation Parameters field starting in column 25.

DEFEOR Optional

If DEFEOR is supplied, the Macro Assembly Program will generate the INCL, INCS, or CALL necessary to incorporate the end-of-file processing routine. If the program is to process magnetic tape files, this entry is necessary, and the parameter EOR1 must be written starting in column 25.

DEFSCHED Optional

The DEFSCHED entry defines the type of buffer processing and scheduling routines to be incorporated in the object program. If DEFSCHED is omitted, the buffer processing and scheduling routines will not be included in the program. The parameters which may be used in the Operation Parameters columns are

GENERATIVE

If this option is specified, all of the buffer processing routines and parts of the scheduling routines will be generated. The generated routines will be highly time efficient at the expense of memory.

INTERPRETIVE

If the programmer desires core efficiency at the expense of execution time, this option should be selected. In this case, one interpretive buffer processing routine and one interpretive scheduling routine will be produced for the program.

VI. DATA DIVISION

EXAMPLE OF A TYPICAL ENVIRONMENT DIVISION

	Y Y P		- REFER	ENCE	SYME	10 L	ı		OPI	ERA	. T10	N			OPERATION PARAMETER
SEQUENCE 1 6	E 7		9	ATAN	AME	16	17	L.E.	V E L 19	20	نعخره	22	U _S	24	PICTURE 25
	· . –		,				ļ	-	-					_	
	-	- 1					ļ	ļ			\Box				
				L1_		1	! _	į	1						
	l						ļ	-			Ш				
0.0.0.0.7.0		٤	$N \cdot N \cdot I$. R . 9	N	3. M	N	I	Δ	\mathfrak{D}	L	٧	1	S	$1.\emptyset$, N_1
0.8.0.0.0.0							М	Ε	M	S	I	Z	$ \varepsilon $		6,0,0,0,
00,0,00					بدر المار		S	R	m	ε	7	Н	Ø	D	I.N.C.L.
0.0.1.0.0			a.	1			D	٤	E	n	H	X	Ą	П	P.R. Ø. C. E. E. D. : 4.0.6
0 1 1 1 0 0 1 0	ļ.,	_					D	ε	E	Ø	P	٤	N		N. O.N. ~ D. V.E.R. L.A.Y.
0.0.0.1.2.0		j		4.	. 4		D	ε.	F	C.	L	Ø	S	3	C.L.O.S.E.I.
0.0.0.1.3.0			L_1				D	£	F	3	Ø	R			E. Ø.R. I.
0.0.0.1.4.0			L				0	ε	F	5	ر	н	۶	Ū	INTERPRETIVE
, , ,							Γ			_		-			

The Data Division describes the input and output files to be processed and allocates memory for input/output buffers, intermediate work areas, tables, and constants. The data may be described either without regard to word boundaries, or the positioning of the fields within words may be controlled. All field characteristics of the items of data such as data name, mode, size, and initial value, if any, must be supplied by the programmer.

The Data Division consists of two sections: the File Section and the Working Storage Section. The File Section contains information about the physical characteristics of input and output files, and descriptions of the records which are contained in these files. The Working Storage Section may contain descriptions of intermediate data which is developed during the execution of the program, constants which are assigned initial values, switches which may be required by the program logic, and information that may be used to format output lines.

WRITING THE DATA DIVISION

When writing a program in the macro assembly language, the Data Division is the third division to appear on the programming form. DATA \triangle DIVISION, entered in columns 8-20, identifies this division on the form. Conventions governing the entries which appear in the programming form fields are presented in the following pages.

SEQUENCE (Cols. 1-6)

Purpose

The sequence number is provided as a convenient method for the programmer to maintain an order within his source deck.

Assembler Action

If the sequence check option is specified on a language processor option card, the sequence number assigned by the programmer will be checked,

using the GE-425/435 collating sequence. If any source program card is detected with a sequence number equal to, or less than, the preceding card, it will be flagged as a possible error in the program listing.

DATA NAME (Cols. 9-16)

Purpose

A data name is used to identify an entry in the Data Division.

Conventions

When writing a data name the following conventions must be observed. Data names--

- 1. must be one to eight characters in length.
- 2. must be formed from the alphabetic characters A-Z, the numerics 0-9, and the tilde (\sim).
- 3. must be defined only once in a program.
- must contain at least one alphabetic character.
- 5. must not contain imbedded blanks.

Data names may be floated anywhere within the eight column field. Any field that is to be referred to in the Procedure Division must have a data name. If a field is to be used only as a filler in a record description or a print line in Working Storage, the data name columns may be left blank.

Examples

PAYMASTR PAYREC PAY~NO DCODE A25 15B LEVEL (Cols. 18-19)

Purpose

The Level numbers written in these columns define the relative structure of a logical record and indicate unrelated items in Working Storage,

Concept of Levels

There are a maximum of four levels, 01 through 04 with the leading zero being optional. Level 01 indicates logical record entries in both the File Section and Working Storage, and in addition is assigned to items of data which are not a part of a logical record. This unrelated data occurs only in the Working Storage Section, because all entries in the File Section describe logical entries within the file.

Any data subordinate to another level carries a higher level number. Therefore, a field within a record would be assigned a level 02, a subfield would be assigned an 03, and any item within a subfield would be assigned an 04. Any data item which is not further subdivided is referred to as an elementary level or item. Items further divided are called groups.

Any item referred to as a group is assigned a data name at the group level and is referred to by that name. The group includes all subsequent groups and items until a level number less than or equal to the level number assigned to the group is encountered.

Reference to a group data name by an operand in a macro-instruction dictates that each item within the group is to be included. Thus, by grouping data, the programmer may operate on several items of data with one macro-instruction, instead of operating on each item individually.

Note:

Any reference to a group refers only to a string of characters with no editing.

Example

REFERENCE SYMBOL	OPER	ATION	OPERATION PARAMETERS
DATA NAME 9 16	17 18 19 20	S Y N C 22 E 24	PICTURE 25
PALY A.C.C.	0.1		
PAY~MD	0.2		9 (5,)
DATE	02		
MO	0.3		99
PAY	೦೭.		9.9
Y K) છે		99
{ΝΑΜ ξ :	0,2 ,		
INITIALS	0.3		
HSTINIT !	04.		A.
AND INI T	<u> </u>		$m{k}_i$. The substitution of the state o
L ASTNAME	0.3	• 	A(16).

In the example just given, the logical record PAYREC is subdivided into three major parts (PAY~NO, DATE, and NAME). Any reference to PAYREC includes all of the items down through LASTNAME. DATE is subdivided into MO, DAY, and YR and is, therefore, a group. Any reference to DATE will include the three subfields. Since MO, DAY, and YR are not further subdivided, they are elementary items. Therefore, a reference to MO will apply only to that data item.

The employee's NAME is subdivided in the same manner, with 1STINIT, 2NDINIT, and LASTNAME as elementary data items. INITIALS is a group which includes 1STINIT and 2NDINIT, and the group NAME includes INITIALS and LASTNAME.

Whenever the 01 level is introduced, it terminates any association with the prior entries. Thus, unrelated items of data being written in the Working Storage Section, such as constants, switches, index words, temporary storage, should be introduced at an 01 level.

PICTURE (Cols. 25-40)

Purpose

An entry in the Picture columns describes the mode, size, decimal point location, and editing characteristics of the field represented by the data name. Picture is used in both File Section and Working Storage Section.

Picture must be specified for each elementary item unless the usage is Index or Switch. Picture can be specified only at the elementary item level.

Two types of characters may be specified in Picture: characters describing data and characters which specify editing of the data field.

Non-Edit Characters

- Data Characters -- The following non-edit characters represent data as indicated:
 - 9 Numeric character (0, 1, ..., 9)
 - X Alphanumeric character (any character)
 - A Alphabetic character (A, B, Z and blank)

These characters are repeated as many times as necessary to indicate the number of characters in an elementary item. Numeric items must not exceed 16 digits. There is no restriction to the size of alphanumeric or alphabetic items. The characters may be written in either of the following ways:

99999	9(5)
XXXXXX	X(7
AAAA	A(4

The decimal integer enclosed in parentheses following the picture character indicates the number of consecutive occurrences of that character.

- Operational Symbols -- The following Picture characters are operational symbols and do not represent character positions in the field;
 - S An S in the Picture columns means that the data field might have a sign. The S must be the first character of the Picture.
 - V An implied decimal point location is indicated by a V in the appropriate position in the Picture columns.

Edit Characters

The characters which specify editing of a data field are described on the following page.

- Zero Suppression Characters -- The following Picture characters represent leading numeric characters which are to be replaced when zero:
 - **Z** Replace zero in corresponding data character with a blank (Δ) .
 - * Replace zero in corresponding data character with an asterisk (*).
- Insertion Characters -- Each of the following Picture characters indicates that said character is to be inserted in the corresponding character position of an elementary item:
 - \$ dollar sign (two or more adjacent \$ characters indicate
 floating dollar sign)
 - . comma
 - decimal point
 - B blank
- Report Sign -- A "-" as the last Picture character indicates that a minus sign is to be inserted in the last character position of a negative field. If the field is positive, a blank is inserted in the last character position.

Conventions

Reference to the examples which follow these conventions may help in understanding them.

- 1. All characters, except the operational symbols S and V are counted in the length of the item.
- 2. A picture cannot contain more than one decimal point (either a V or a period (.)).
- 3. A Z or an * can be preceded in the Picture only by a single leading \$, a decimal point (V or .), or a comma (,).
- 4. An item whose Picture indicates zero suppression with a floating dollar sign, must contain at least one more position than the maximum number of significant digits to be stored in it. A dollar sign replaces the least significant suppressed zero indicated by a \$ in the Picture. Every other suppressed zero is replaced by a blank.

- 5. If any type of zero suppression is specified in the Picture, the suppression also applies to leading commas.
- 6. No one of the Picture characters, Z, *, or \$ may appear to the right of a decimal point (.), unless every numeric character position is represented by Z, *, or \$, respectively.
- 7. If all data characters of an item are represented by Z's, by *'s, or by \$'s, then the item is blanked when the value of the data is zero. In such a case, all other editing is overriden.
- 8. The Picture characters \$, comma (,), (.), S, and V can only be used with a numeric field.
- 9. When an item whose Picture contains editing characters is used as a source, it is treated as an alphanumeric item.
- 10. A Picture can have no more than 16 characters, including editing characters; it cannot be continued onto a new line. However, the Picture may represent more than 16 characters by using a decimal integer enclosed in parentheses to indicate the number of occurrences. (See Convention 1.)

Examples

Non-Edited

Source	Area	Receiving	, Area
Picture is:	Field as in memory:	Field will be used as:	Class is:
	<u> </u>		15.
99999	56789	56789	NUMERIC
9(3)V9(2)	56789	567.89	NUMERIC
S999V99	5678R	-567.89	NUMERIC
XXXXX	ABCDE	ABCDE	ALPHANUMERIC
AAAAA	BCDEF	BCDEF	ALPHABETIC
X(5)	12345	12345	ALPHANUMERIC
xxxxx	AB67C	AB67C	ALPHANUMERIC
99 x 99	67.89	67.89	ALPHANUMERIC
99 AA 9	23FG4	23FG4	ALPHANUMERIC
A(2)X(3)	B D 2M3	B D 2M3	ALPHANUMERIC
\$9 999	1234	1234	NUMERIC
\$ 9999	123M	-1234	NUMERIC

Edited

Sou	rce Area	Rece	eiving Area
Picture	Field as in memory:	Picture	Edited Field
9(5)	45678	\$ZZ,ZZ9.99	\$45,678.00
999 v 99	45678	\$ZZ,ZZ9.99	\$ 456.78
999 v 99	C0067	\$ZZ,ZZ9.99	\$ 0.67
999 v 99	06700	\$ ZZ , ZZ 9	\$ 67
999 v 99	00004	\$ZZ,ZZZ.99	\$.04
999 v 99	00000	\$ZZ,ZZZ.99	\$.00
9(5)	00000	\$ZZ,ZZZ.ZZ	
V9(5)	12345	\$ZZ,ZZ9.99	\$ 0.12
9(5)	12345	\$**,**9.99	\$12,345.00
9(5)	00123	\$**,***.**	\$***123.00
9 (5)	00000	\$**,***.**	
9(5)	67890	\$\$\$,\$\$9.99	\$67,890.00
999 V99	67890	\$\$\$,\$\$9.99	\$678.90
999 v 99	01234	\$\$\$,\$\$9	\$12
999 v 99	00000	\$\$\$,\$\$9.99	\$0.00
999 v 99	00000	\$\$\$,\$\$\$.\$\$	
999 v 99	00001	\$\$\$,\$\$\$.\$\$	\$.01
V9(5)	67890	\$\$\$,\$\$9.99	\$0.67
S9(5)	0056P	ZZZZZ.99-	567.00-
S9(5)	0056P	zzzzz-	567-
9(5)	56789	BBB99.99	89.00
9 (5)	56789	BBB99	89
999 v 99	567 8 9	999.BB	567.
999 v 99	56789	999.99	567.89
9(5)	00567	ZZZZZ.99-	567.00

VALUE			(Co	ls.	45-76)

Purpose

The Value columns specify an initial value for data items in the Working Storage Section.

Conventions

- 1. The value may be written either as an alphanumeric or numeric literal. However, it must agree with the class specified in the Picture.
- 2. Alphanumeric or alphabetic values must be enclosed in quotation marks. The length and mode of the data within the quotation marks must agree with the Picture supplied for this item.
- 3. The length of a numeric value written in these columns must be equal to, or less than the number of data characters specified in the Picture columns.
- 4. If a decimal point is included in a numeric value, it will be treated as an implied decimal point and, therefore, must correspond with the implied point location specified in the Picture.
- 5. The sign of a numeric literal must be the left hand character of the literal. Any unsigned literal will be assumed positive.
- 6. Value may only be specified at the elementary level. It may not be stated in an item which contains an entry in the Occurs columns or an item which is subordinate to an item containing an Occurs entry.

Examples

Picture	<u>Value</u> <u>C</u>	Contents of memory
X(5)	''12. 34''	12.34
AAA	''ABC''	ABC
A(14)	"PAYROLL RECORD"	PAYROLL RECORD
X (8)	''%NET-PAY''	%NET-PAY
9	1	1

Picture	Value	Contents of memory
S9V9 S9(5) V999	-1.5 -25673 .123	$1\overline{5}$ $256\overline{73}$ 123
S999	+345	$34\overline{5}$

The following examples illustrate what happens if the value is shorter than the picture:

999	12	012
99V99	1.23	0123
99V99	12.3	1230
999V999	12.34	012340
S999V999	-12.34	012340

SYNC (Synchronization)	(Col. 21)

Purpose

The Sync entry indicates the format of an elementary item that exists within an integral number of words. The entries which may be used are:

- Δ The item is not synchronized.
- R The item is right justified within an integral number of computer words.
- L The item is left justified within an integral number of computer words.

Sync is used in both the File Section and the Working Storage Section.

Conventions

1. In the File Section, an R or L indicates the data item is contained in an integral number of words and external format is unpacked. Unpacked data is data that is so arranged that it may be read directly into integral word lengths. The unused character positions required to fill the computer words must be zero.

2. In the Working Storage Section, an R or L specifies the internal format of the item. The original contents of the synchronized item are unpredictable unless a Value is specified.

When data is moved by a macro-instruction into an item in Working Storage or in the File Section, the contents of the character positions used to fill an integral word will be set to zero.

- 3. Whenever a synchronized item (R or L) is referenced in the source program, the original size of the item as shown in the Picture is used in determining any action which depends on size, such as truncation.
- 4. Synchronization may be specified only at the elementary level.
- 5. Synchronization of numeric items (described with 9's in the Picture) apply only to digits. The sign is carried in the sign position of the right hand word of the field, rather than over the units position of the data.
- 6. If a data item shares a given machine word with a second item, each item is referred to as packed. The data must be unpacked each time it is used.

Examples

The examples listed below are independent of each other. Assume in all examples that the previous field ended in the last character of the preceding word.

	Picture	Sync	Field in Memory
1.	x		x
2.	x	L	X 0 0 0 /
3.	x	R	0 0 0 X
4.	A(5)		AAAAA

	Picture	SYNC	Field in Memory
5.	A(5)	L	A A A A A O O O
6.	A(5)	R	000AAAAA
7.	99 AA		9 9 A A
8.	99 AA	L L	9 9 0 0 A A 0 0
9.	99 A A	L R	9 9 0 0 0 0 A A
10.	99 AA	R R	0 0 9 9 0 0 A A
TYPE			(Col. 7)

Purpose

An entry used in the Type column describes the use of the line in which it appears. The entries which may be used are:

- * denoting a comment line
- C denoting a continuation line
- R denoting a redefinition of a previously defined area

Type is used in both the File Section and the Working Storage Section.

Comment Lines

An * denotes a comment line. The comment line is carried through to the final listing for program documentation. There is no restriction on the number of consecutive lines which may be used as comment lines.

2 - 32

Continuation Lines

A C denotes continuation of the Value columns. Whenever a Value, including the delimiting quote marks, larger than the size of the Value columns (45-76), is used, the value must be written out through column 76 and continued in column 45 of the next line. The next line must be identified as a continuation line by inserting a C in column 7. On a continuation line, only the Sequence, Type, Value, and Iden columns may be used. All other columns should be left blank. From one to four consecutive continuation lines may be used.

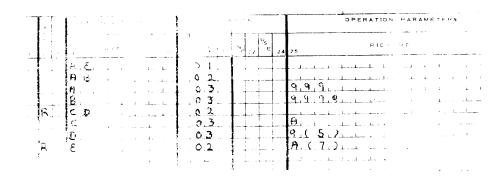
Example



Redefine Indication

An R denotes redefinition and signifies that a data item is to share its storage area with the last item of the same level.

In the following example, three items are sharing the storage allocated to data item AB.



Fields which are redefining the original area are assigned positions starting in the leftmost character of the original area.

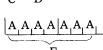
For the Data Division entries in the above example, memory will be assigned as follows:



Words N & N+1 allocated for data item AB.



N & N+1 as redefined for data item CD.



N & N+1 as redefined for data item E.

Conventions

- 1. All redefinitions of an area must occur directly below the original data item.
- 2. The number of characters in the redefining entry must not exceed the size of the original data entry.
- 3. Redefinition ends when a level number less than or equal to the level of the original data entry is encountered.
- 4. Entries redefining an area must not contain a value. It is permissible for the original data entry to contain an initial value in the Working Storage Section.
- 5. Multiple redefinitions are permitted, provided they occur at the same level, and there are no intervening entries.

USE (Usage) (Col. 23)

Purpose

An entry in the Use column specifies how a data item is to be used in memory in either File Section or Working Storage Section. The entries which may be used are:

- Δ $\;$ The field will be used as specified in the Sync $\;$ and Picture columns.
- S Data field is used as a switch.
- I Data field is used as an index word.
- U This entry is restricted to the File Section. It indicates that the data field should be unpacked and right justified as it is read into memory or packed when written from memory. U cannot be used with any fields which are involved in redefinition.

Conventions

1. The use of S (switch) and I (index) is restricted to Working Storage and must be used at an 01 level.

2 - 35

- 2. If the data field is specified as a switch, it has an implied Picture of 9(4) and may contain an initial value.
- 3. A data name specified as an index is established in the standard fixed index word format. It may contain a signed decimal integer as an initial value. The magnitude of the integer may be from 0-32,767.
- 4. The Use column may be used only at the elementary level.
- 5. If a data item has been defined as synchronized (unpacked L or R) on the external media, it is not necessary to insert a U in the Use column. It will be brought in as an unpacked item and retain the synchronization specified.
- 6. When several data items are defined between two entries with a U in the Use column, they will also be read in or written out right justified and unpacked.

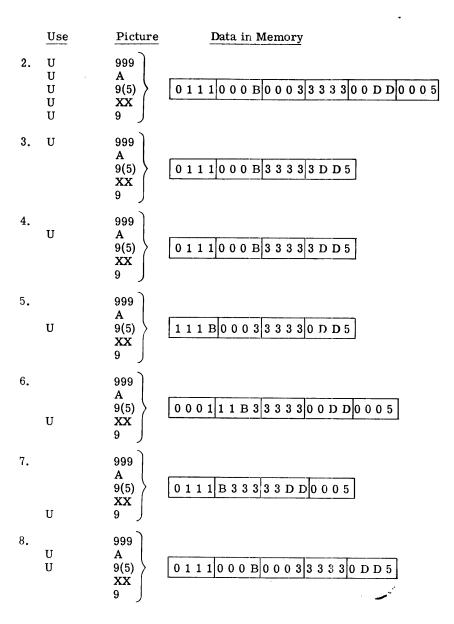
Examples

These examples illustrate the function of the U. All examples assume that the data on external media consists of --

Field No.	Data	Picture
1	111	999
2	В	Α
3	33333	9(5)
4	DD	XX
5	5	9

The same set of Pictures is used in all examples. The data on the external media is assumed to be a string of packed characters:

111B33333DD5



	Use	Picture	Data in Memory
9.	U	999 A	
	U	9(5) XX 9	0 1 1 1 0 0 B 3 3 3 3 3 0 0 D D 0 0 0 5

OCCURS	(Cols. 41-44)

${\bf Purpose}$

The Occurs columns are used to indicate the number of times an item is to be repeated in either the File Section or the Working Storage Section. If a number is not supplied, the item is assumed to occur only once.

Conventions

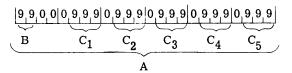
- 1. The Occurs entry must be a decimal integer, starting in column 41.
- 2. Occurs may be used at any level other than 01.
- 3. If elementary items appear within a group which is to be repeated, the complete group will be repeated. Refer to data names B and C in the second example shown.
- 4. All elementary items which occur more than once must be synchronized.

Examples

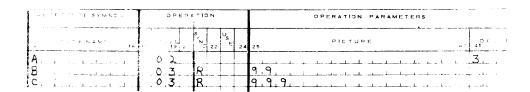
In example 1, the data name to be repeated is an elementary item.



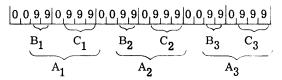
The storage will be assigned as follows:



In example 2, the group item is to be repeated. In this case, the items within the group must be synchronized.



The storage will be assigned as follows:



The group item, A, is repeated three times.

IDEN (Identification)	(Cols.	77-80)

Purpose

12/6/63

The Iden entry associates a card with a source program deck.

Conventions

1. Any four characters may be selected by the programmer.

2. The Identification is printed on the assembly listing, but is not checked by the Macro Assembly Program.

DATA DIVISION SECTIONS

FILE SECTION

The File Section contains the programmer's descriptions of the characteristics of the file, which are:

- to specify the type of processing to be performed on the file.
- to describe the areas into which the files may be read, or from which they may be written.
- to define the format of the records which are contained in the files.

The File Section begins with a line that contains FILE ASECTION in columns 8-19. Following this, there must be a description of each file that is to be used in the object program.

Each file description begins with the file name in the Data Name columns and an FD entry in the Level columns. Immediately following the FD will be a set of file parameters that specify the physical characteristics of the file, the peripheral unit, and the type of processing to be used for this file. A record description will follow the file parameters.

Example

The example on the opposite page illustrates the Data Division entries necessary to describe a file of time cards to be read by the card reader. No file parameter need be specified for any file which is not referred to by some I/O macro-instruction in this segment. In this case the record description immediately follows the FD line.

1 P. LESYMODIA	OPENATION	OPERATION SAMANET, 13
	57 19 2 C 2 E 24	26 P CTURE
FILEASECT TIMECDS TIMECDS TIMECDS PRY NØ DATE MØ DA YR KRSWKD	I B M F.D. D & V.I C.E & F.A. C.A.	0.1,0.1 E.N.D.2 9 (5.) 9 9 9 9

File Parameters

File parameters are used to supply input/output information for a file to the Macro Assembly Program. A special mnemonic, associated with each parameter is written starting in column 17 of the form. The entries which are allowed for each parameter are written in the Operation Parameters columns beginning in column 25. The standard set of file parameters are listed on page 2-42, 43; on page 2-44 , the complete list of file parameters is given.

The file parameters supplied for a given file must include DEVICE and COMMAND and must be written in the order prescribed in the File Parameter list on pages 2-44 through 2-46. The DEVICE file parameter is used to determine which additional parameters are required and to establish a standard set of file parameter entry options. Any of the standard options can be overriden by supplying file parameter name and the desired option.

Logical device codes are described in the Basic Input/Output System write-up under the Logical-Device-to-Physical-Device Translation Table. The logical device number has the following octal format:

XXYY

where XX is a device type code and YY is the number within this type

of the particular device in question. The device type codes have the following meanings:

Code (in octal)	Device
00	Typewriter
01	Card reader
02	Card punch
03	Printer
04	Magnetic tape
05	Disc storage unit
06	Magnetic reader/sorter
07	Perforated tape reader
10	Perforated tape punch

YY might be used for the number of a particular tape unit within a magnetic tape subsystem. For example, 0401, 0402, etc.

If the first characters of the device code are 02 (card punch), no additional file parameters are necessary. When the device code is 04 (magnetic tape), a FILETYPE parameter must be included to indicate the input or output option. Finally, if the device code is 01 (card reader), 03 (printer), or 04 (where magnetic tape is specified as input) an EOFADDR parameter will be required. The following table sets forth the required parameters:

Parameter	Device			
	Magnetic Tape (04)	Card Reader (01)	Card Punch (02)	Printer (03)
DEVICE COMMAND FILETYPE EOFADDR	Required Required Required Required if FILETYPE is INPUT	Required Required Required	Required Required 	Required Required Required

The following table specifies the option the Macro Assembly Program will assume for File Parameters that are not supplied.

DEVICE Code	Magnetic Tape	Card Reader 01XX	Card Punch	Printer 03XX
If this paramet	The Macro	Assembly Prog	ram will use th	nis set of options
AREADEF	GENERATE	GENERATE	GENERATE	GENERATE
IOMETHOD	INCREMENT	INCREMENT	INCREMENT	INCREMENT
FORMAT	UNPACKED	UNPACKED	UNPACKED	UNPACKED
RECFORM	FS	F	F	F
RECINDEX	*	*	*	*
MODE	BINARY	BCD	BCD	EDITED
BUFCOUNT	2	2	3	2
EROPTION	HALT	HALT	HALT	HALT
SBOPTION	H AL T	H AL T	NOT APP	LICAB LE
RWDLOCK	Y E S			
CLOSERWD	YES			
OPENRWD	YES			
B L KS IZ E	1			
ALTDEV	Device code will be used	NOT	APPLICAB LE	
DENSITY	нісн	NOT A	APPLICABLE	
PRIORITY	0			
TPMKPROC	HALT			
LABELDEF	NONE			

If the programmer specifies AREADEF DEFINED, he must also explicitly supply and specify values for DCWLIST, BLKDWNO, RECDCWNO and RECLNGTH.

A complete list of the file parameters and a detailed description of their optional entries is included in the Extended Input/Output System (EIOS) write-up. For readers convenience, the file parameters are listed below with all of their option entries.

sted below with all of their option entries.		on entries.	•	Bitor from	TYPE
	File Parameter	Option Entry			RETRY HALT
	AREADEF	GENERATE [refsym] DEFINED REMOTE	:		refsym
	DEVICE	NNNN	:	EOFADDR	refsym
	COMMAND	xxxx	i		DUMP HALT
	IOMETHOD	INCREMENT EXCHANGE SHARE	The i	following are needed only for	refsym a magnetic tape file: CHECKPOINT
	FORMAT	CONTIGUOUS]		YES NO
		UNPACKED SCATTERED	(CLOSERWD	YES NO
	RECFORM	F FS V VS	(OPENRWD	YES NO
	RECINDEX	*]	BLKSIZE	NNNN
		x	1	ALTDEV	NNNN
	FILETYPE	INPUT OUTPUT	1	RECLNGTH	NNN
	MODE	BINARY BCD]		LOW HIGH
		MIXED EDITED]	BLKDCWNO	NNN
		UNEDITED]	PRIORITY	N
	DCWLIST	refsym	1	EOREXIT	refsym

File Parameter

BUFCOUNT

RECDCWNO

EROPTION

2-44

12/6/63

2-45

Option Entry

NNN

NNN

DUMP

File Parameter Option Entry

TPMKPROC HALT

IGNORE refsym

LABELDEF NONE

STANDARD refsym

One or more of the following is needed only when LABELDEF is ${\tt STANDARD}$

LABELOPT XXXX

LABELXTS Exit 1 address; Exit 2 address;

Exit 3 address; Exit 4 address

LABELCON ZZZZNNNNFILENAMEISSSY DDDXXXXFFFF

A list of File Parameter options for perforated tape, disc storage units, and magnetic reader/sorter will be specified later.

Record Description

A description of the logical record(s) to be found in the file must be included immediately following the file parameters.

Inherent in the structure of logical records is the concept of levels of data. The programmer writing in a field-oriented language will find it convenient to divide the items of data in the record into groups, and to possibly further sub-divide the items of data within these groups. This is made possible by assigning numeric levels to each item of data within the record. Data described in a Macro Assembly Source Program may be assigned levels 1 through 4. Using level 1 for a record name, it is then possible to group data within the record into three levels subordinate to the record level.

For additional details on record structure, refer to Concept of Levels, page 2-23. The remaining columns Type, Data Name, Sync, Use, Picture, and Occurs must be filled in as necessary. For specific details about allowable entries and formats, refer to the particular entry description.

2-46

I/O Buffer Area Definition

In the Macro Assembly Program, the programmer may indicate whether the buffer area(s) and DCW list(s) are to be generated by the program, supplied by the programmer in this segment, or have been supplied in a separate segment.

The AREADEF file parameter is used to indicate which of the options is desired. This parameter has the following format:

AREADEF

\[
\begin{cases}
\text{GENERATE [; refsymbol]} \\
\text{DEFINED} \\
\text{REMOTE}
\end{cases}
\]

The options are defined below:

GENERATE The Macro Assembly Program will generate

the buffer area(s), DCW list(s), and the file table from information supplied by the file parameters. This option will be used if the AREADEF parameter is omitted. If the IOMETHOD option is SHARE, or if the FORMAT option is SCATTERED, the GEN-

ERATE option cannot be used.

refsymbol This is an optional entry which, if specified,

will cause the program to assign the specified Reference Symbol to the first word of the first buffer area generated for this

file.

DEFINED The programmer must allocate the area(s)

required for buffer(s). The DCW's and BSS pseudo-operations used to reserve the area must be written in the Procedure Division. The Macro Assembly Program will generate the file table as defined by the file para-

meters.

REMOTE

This indicates that the file parameters and record description are for reference only in this assembly. Memory will not be allocated for the file table, DCW list(s), and buffer area(s). The buffer area(s) for this file must have been established in a separate segment.

FILE SECTION CONVENTIONS

The following conventions apply to files or logical records defined in the File Section:

- 1. The file name supplied with the FD parameter will be assigned to the first word of the file table generated for this file.
- 2. All data items described within a logical record are assigned addresses relative to the beginning of the record. Therefore, reference to a given item within the record must be indexed. The first word of the record will be assigned the relative location 0. During the execution of the object program, the RECINDEX will contain the location of the first word of the current logical record. Any reference by a macro-instruction to a data item defined in the File Section is automatically modified by the index word (RECINDEX) assigned to the file.
- 3. If any macro-instruction refers to a data item within a logical record, that item must be described in the Data Division.
- 4. If a data item within the record has not been described, all references to that item must be written in the basic assembly language. The references should be relative to the start of the record and must be indexed. For example, if the following record was described with a single entry at the 01 level.



the following basic assembly language instructions would be required to unpack the record to Working Storage. Assume that fixed index word 4 has been assigned to the file.

Operation	Operation Parameters
LDD	1, 4
SRDA	1, 4 3
STD	PAY NO
LDD	2, 4
SRDA	2, 4 3
STS	ORG
LDD	3, 4
SLDA	1
SRDA	2
STD	DATE

- 5. Each buffer area generated by the Macro Assembly Program will begin at a 0 mod 4 location. The generated buffer(s) will overlay the loader program at object time.
- 6. For efficiency, if a packed data item is to be used several times, it should be unpacked by moving it to a synchronized data item in the Working Storage Section. Subsequent references should then be made to the unpacked item.
- 7. The length of a logical record described in the File Section should be a multiple of four characters. If the record is not a multiple of four characters, the FORMAT file parameter must specify UNPACKED and at least one data item within the record must contain a U in the Use column.
- 8. If CONTIGUOUS is specified in the FORMAT file parameter, it will override any U written in the Use column.
- 9. If UNPACKED is specified in the FORMAT file parameter, the RECFORM must be F or FS, and a complete logical record description must be supplied. One entry is necessary for each field to be unpacked.

WORKING STORAGE SECTION

The programmer uses the Working Storage Section as follows:

- to describe areas to be used for intermediate input/output processing
- to build constants

- to define intermediate storage to be used as tables, work areas, and temporary storage during execution of the object program
- to describe format lines with initial values to be used for output records and print lines

The first line in this section must contain WORKING△STORAGE△SECTION in columns 8-30. This will be followed by the entries required to describe the constants and data storage required by the source program.

Data items defined in Working Storage may be either unrelated elementary items or logical groups. The first entry to appear in the level columns must be an 01 (or 1). If the entries are unrelated, this will be followed by another 01 level. If it is followed by a level number higher than level 01, the 01 level data item is a logical group. The group may be further subdivided as indicated in Concept of Levels, page 2-23.

The following entries may be used to describe the data items:

Data Name Sync Use Picture Occurs

Value

WORKING STORAGE SECTION CONVENTIONS

The following conventions apply to items described in the Working Storage Section:

- Data items described in the Working Storage Section will be assigned specific addresses. Therefore, reference to items within this section need not be indexed. Indexing may be used to address items used in a table.
- 2. The number of words moved to or from the Working Storage Section with either a READ TO or WRITE FROM option is equal to the size of the input or output record.
- 3. Unrelated numeric items in the Working Storage Section should be right synchronized to produce more efficient object coding.

- 4. If an initial value is not specified, the initial content of the item is unpredictable.
- 5. The only pseudo-operations allowed in the Working Storage Section are DGR, DGRR, and DGRE. (See Common Data Areas, page 2-101.)

Example

The following example illustrates a few of the various types of storage areas, constants and print lines that may be described in the Working Storage Section.

MINISTRUMENT COMMO	1	*	Address of the Control of the Contro	
traviner :	frentreite –		and the second s	
i	k al-NaNala	200 F W1	****	
0.05.0.0.0 WE K. K. L. N.G.A.S	tan interest to	I.J. William		
0.05.0.1.0 × IN 7 € K.N.E.D.1	In the Committee of the	A.R.E.A. FOR JIO	L'actions.	A R R R R R R R R R R R R R R R R R R R
10.05.01.01.21	0.1	manda a manda a manda a manda a manda a manda a manda a manda a manda a manda a manda a manda a manda a manda a	N.C.C. P.M. E.D	
COSO SO TERRY NA	1 63 1 1 1 19 6 6			
O O S O Y O T C D AT F	1 63 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			
0 0 5 6 5 6 17 C M. Q.				
IN O.S. C. B. O. LT. C. D. B.	0.3			
6.0.5.C.7.C. IT C.Y.B.	0.3.			
16.05.0 EST TOHRS WKD				
005.090 XTIEMI BRAKI	S.T OIRAGE A.R.	F.A T.N. F. C. R.M. A. D. L.	ATE AREA FOR CALL	U.L.A.T.I.A.N
0.05.1.00 TEMPI.	61. R. 997	1. v. 9. 9	- 126.34	
0.05.1.10 C # N.S.T.	6.7 K 1899.	7. v. 9. 9.	- 1 2 6 . 3 . 4	
C.C.5 1.20 SWILL.	[. 6.1 [S.]		3	
La. 0.5 . 1 . 3 . 0	1 .0 t . I. I	and the second second	. 9	
0.0.5.1.4.0 C 0.V.ST.2.	0 L	1	GR # 55	
	Ιε. Ι Ι	a rated a kind a kind of	the second second second second second	
0 C 5 1 6 0 H L G I	[.0 ta] [- ΔΔΔΔΔ	
0 6 5 . 1 . 7 0			" Δ Δ Δ Δ Δ "	
OOSIBO . HDPAYNO		A	AAPHY - N.G.	the second second
0 0 5 1 9 0			ΔΔΡΗΥ-Νο	
0.0 5.1 0.0	. O Z X.X .	and a second control of the second control o	*.X.Z.****.	and the second second second
		and the second second second		and a second control of
1 k		transfer to the contract of	the second section of the second section is	and the second second
		and the second of the	the second secon	

2-51 12/6/63

12/6/63

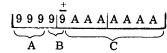
MEMORY ALLOCATION

Data described in the Macro Assembly Program Data Division is assigned memory positions, proceeding from left to right. If synchronization is not specified, consecutive data items are assigned adjacent character positions.

The following Data Division entries

REFERENCE SYMBOL	OPERATION		OPERATION PARAMETERS
9 DATA NAME 16	5 YN C 22	5 E 24	PICTURE 25
A	02		9,9,9
C.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	0.2	1	A. (, 7,)

will be assigned storage as indicated below:

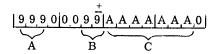


When synchronization (R or L in the Sync column) is specified, the Macro Assembly Program insures that the item is placed into an integral number of words. This may be illustrated by modifying the first example to include an R in the Sync columns of data item B. This is shown in the following example.

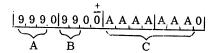
The following Data Division entries



will result in the following assignment:

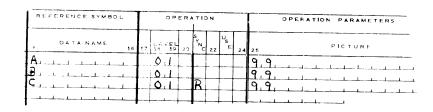


Note that data item B is unpacked in an integral number of words and that the assignment of positions to item C started at the first character position of the word following the synchronized item. This would apply even if item B included L in Sync rather than R:



Whenever the Macro Assembly Program encounters an 01 level, it will start assigning positions in the first character position of the next word. This is illustrated in the example below.

For the following Data Division entries



memory will be assigned as follows:

In the following examples, the data, as it appears on external media is shown first, followed by the way it is desired to nave it appear in memory. The coding used to arrange for this memory allocation is given last.

Examples

1. Data on the external media:

Field Field

Data in memory:

1 1 1 B 3 3 3 3 3 D D 5

Coding:

REFERENCE SYMBOL	GPERA	TION	ON OPERATION PARAMETERS		
DATA NAME	17 LEVEL 17 P 19 20	S YN C 22 E 2	PICTURE 24 25		
RIEICIØIRIDI I	011				
F. I. E. L. D. ~	0.2		9,9,9,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		
F. I. E. L. D. ~ . 2.	4.0		A		
$F.I.E.L.D.\sim 3$	0.2		9.(15.)		
FIELD - Y	0.2		XXX		
F I E. L.D.~.5	0,2		9		

2. Data on the external media:

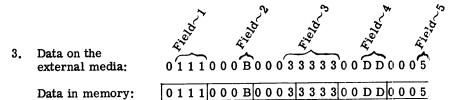
\$1 1 1 B 3 3 3 3 3 D D 5

Data in memory:

0 1 1 1 0 0 0 B 0 0 0 3 3 3 3 3 0 0 D D 0 0 0 5

Coding:

1		L	<u></u>
RECORD.	· ·		
FIELD	\$ 7	1 . Tu! I	9,9,9
***************************************			Δ
F.I.E.L.D.~12		: :	9.(15.).
E.I.E.L.D.~3.	• • • • • • • • • • • • • • • • •	i	7.5.2.7
FILD ~ Y	9.2		XIX
F. I. E. L. D. ~ 5	O.	<u> </u>	9
			1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
The state of the s		1	1



Coding:

REFERENCE SYMBOL	OPERATION			OPERATION PARAMETERS		
DATA NAME	17 18 19	20 N C 22 E	24	PICTURE 25		
R,E,C,D,R,D.	0.1					
F.I.E.L.D.~	1 2 2	B		9,9,9,		
F. [.E.L.D.~12]	0.2	R		A		
F. I. E. L. D. ~ 3	0.2	ß		9,(,5,),,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		
F. 1. E. L. D. ~ 4	0.2	R		X. X		
FILE LD~5	0.2	_ R	L	9		
			<u> </u>			
		: []		1		

Data on the external media: 1110B3333300DD5

Data in memory: 1110

1 1 1 0 0 0 0 B 0 0 0 3 3 3 3 3 0 0 D D 0 0 0 5

Coding:

R.E.C. Ø.R.D.		*				1 1	 	11	 	
FIELD.	L			9.9	9.	ļ t	1	L	 	
F.T. E.L.D.~,21		U		Α		1	 1		 	٠
F. I. E. L. D ~ 3		U		9.6	5.)	 ٠		 	ــــــــــــــــــــــــــــــــــــــ
F. T. E. L. D. ~ . 4.	R			$\mathbf{X}_{-}\mathbf{X}_{-}$			 		 	
F.I.E.L.D.~ 5		U		9			 		 	ــــ نــ
			<u> </u>	<u></u>		ب	 4		 	
							 		 <u> </u>	
<u> </u>	 -		-				 		 	ــــــــــــــــــــــــــــــــــــــ
	 		-				 		 -1-	ــــــــــــــــــــــــــــــــــــــ
		-	-				 		 	

VII. PROCEDURE DIVISION

The Procedure Division is that portion of the source program in which the programmer writes the operations that process the data and control the program flow. In the Macro Assembly Program, these operations may be written as macro-instructions or basic assembly language symbolic instructions and pseudo-operations.

The Macro Assembly Program will process all basic assembly language symbolic instructions and pseudo-operations which are written in the Procedure Division. For a complete description of the instruction formats, Operation Parameter entries, and a listing of the GE-425/435 Instruction Repertoire, as well as a description of all pseudo-operations, refer to the Basic Assembly Language section.

Macro-instructions allow the programmer to specify the operation to be performed rather than supplying the detailed coding. A description of the function, mnemonic operation code, and operation parameter format for each macro-instruction available in the Macro Assembly Program will follow a discussion on the Procedure Division entries and use of the Operation Parameters.

WRITING THE PROCEDURE DIVISION

The Procedure Division is the fourth division to be written on the GE-425/435 Macro Assembly Program Programming form. The first line of this division must contain PROCEDUREADIVISION in columns 8-25. This is followed by the macro-instructions, symbolic instructions, and pseudo-operations which describe the flow of the program. Finally, the division must be terminated by a line that contains END or TCD in the Operation columns.

Comment lines denoted by an * in column 7, may be interspersed throughout the Procedure Division.

Since the same programming form is used for the Data Division, it contains two sets of columnar headings. The headings that pertain to the Procedure Division are discussed in the next several paragraphs.

SEQUENCE (Cols. 1-6)

Purpose

The sequence number is provided to aid in maintaining the correct order of the source program deck.

Assembler Action

If the sequence check option is indicated in the Language Processor option card, the sequence number assigned by the programmer will be checked using the GE-425/435 collating sequence. Any source program card with a sequence number equal to, or less than, the preceding card, will be flagged as a possible error on the program listing.

TYPE (Col. .7)

Purpose

This column is used to indicate the use of the line in which it appears. The entries which may be used are:

- * denoting a comment line. The comment line is written on the final listing for program documentation. There is no restriction on the number of comment lines that may be included.
- denoting a continuation of the Operation Parameters columns. If a list of parameters required by a macro-instruction is larger than the Operation Parameters columns, the list may be continued on the following line. The continuation of the parameter list must begin in column 25. The Reference Symbol and the Operation code should not be repeated on the continuation line. The last parameter (literal, data name, reference symbol, or file name) must not be split between the two cards The line to be continued must terminate with a comma or a semicolon.

Example

Туре	Ref.Sym.	Operation	Operation Parameters
		GOTO	refsym~1;refsym~2;refsym~8,
C			refsym-9 DEPENDINGAON TC-NO
		GOTO	refsym~l;;DEPENDING∆ON TC~NO,
С			х8
*	THE FOLLOWING E	XAMPLE IS INC	ORRECT:
		GOTO	refsym_l;refsym_2;ref
С			sym~9 depending∆on Tc~No

Note: The refsym cannot be split between lines.

	(0-1-	0 10)
REFERENCE SYMBOL	(Cols.	8-101
Ith Bullion bimbon	(/

Purpose

The Reference Symbol enables the programmer to assign symbols to instructions and pseudo-operations written in the Procedure Division.

Conventions

The symbolic Reference Symbol entered in columns 9-16 must follow these conventions. Reference Symbols --

- 1. may be one to eight characters in length
- may be formed from the alphabetic characters A-Z, the numerics 0-9, and the tilde, written ~
- 3. may be defined only once in a segment
- 4. must contain at least one alphabetic character
- 5. may be floated anywhere within the eight column field
- 6. must not contain imbedded blanks.

If a Reference Symbol is written in a macro-instruction, the symbol will be attached to the first instruction produced by the macro-generator.

	 		
OPERATION		(Cols.	17-24:)

Purpose

The operation is a mnemonic code which represents a macro-instruction, machine instruction, or a pseudo-operation.

Conventions

- 1. The operation may be written anywhere within the Operation columns.
- 2. The mnemonic operation code must not contain imbedded blanks.

OPERATION PARAMETERS (Cols. 25-76)

Purpose

The parameters necessary to complete the function indicated by the Operation are written in the Operation Parameters columns.

Conventions

- 1. Parameters written in the Operation Parameters columns must begin in column 25.
- 2. If the Operation code specifies a basic assembly language symbolic instruction or pseudo-operation, the parameter may be a reference symbol, a data name, an absolute address, an expression, a literal, or a value. The Operation Parameters columns of basic assembly operations may not contain imbedded blanks. The format of these basic instructions and of the pseudo-operations is described in the Basic Assembly Language section of this manual.

- 3. With a macro-instruction, a parameter may be a reference symbol, a data name, a file name, a literal, an expression, or an absolute value, depending on the specific Operation being used. The parameters follow the rules set forth in Use of Operation Parameters.
- Any address modification will appear in the Operation Parameters columns.
- 5. Comments may be included in the Operation Parameters columns, providing that at least two blanks separate the comments from the last parameter on the line.
- 6. The normal delimiter is a semicolon, however, in the description of each macro-instruction it is specifically stated what the delimiters are.

IDEN (Identification)

(Cols. 77-80)

Purpose

These columns are used to associate a card with a source program deck.

Conventions

- 1. Any four characters may be selected by the programmer.
- 2. The Identification is printed on the assembly listing but is not checked by the Macro Assembly Program.

USE OF OPERATION PARAMETERS

The number and types of parameters vary depending on the operation code. The description of the specific macro-instructions indicate which type, or combination of types, may be used as parameters.

The following parameter types may be used in either macro-instructions or basic assembly language instructions in the Procedure Division.

Reference Symbol

Reference Symbols may be assigned to macro-instructions, basic assembly language instructions, and pseudo-operations defined in the Procedure Division.

When a Reference Symbol is used as a parameter in a macro-instruction, it may use address modification as specified in Macro-Instruction Address Modification, page 2-64.

A Reference Symbol used in a basic assembly language instruction or pseudo-operation may be part of an expression as described in Calculated Entries, page 1-16 of the Basic Assembly Language section.

Data Name

Data names are associated with data items and records described in the Data Division. Any reference by a macro-instruction to a data item described in the File Section is automatically modified by the index word (RECINDEX) assigned to the file unless a specified index is supplied.

If a macro-instruction references a data name, the description of the item is used to generate the instructions specified by the operation; address modification may be indicated as described in Macro-Instruction Address Modification, page 2-64.

When a basic assembly language instruction or pseudo-operation refers directly to data defined in the Data Division, the Macro Assembly Program provides the address of the word which contains the least significant character of the data. An expression and index may be specified for address modification.

File Name

The file name is defined with an FD entry in the File Section of the Data Division. When a file name is referenced by a macro-instruction it specifies which file is to be operated upon; no address modification may be indicated.

If he file name is referenced by a basic assembly language instruction, the Macro Assembly Program will provide the address of the first word of the file table. Any word within the file table may be referenced by placing a data name on the file parameter entry or an increment may be appended to the file name to address entries within the file table.

Literal Entries

A literal specifies the actual value of data to be operated on by a macroinstruction. Literals are stored as unpacked values, synchronized left or right depending on their use in the instruction. If a literal with the same value and alignment has been previously defined, the assembler uses the address of the original literal.

The literals described below are for use with macro-instructions only. For a description of the basic assembly language literals, refer to Literals, page 1-21 of the Basic Assembly Language section.

Two types of literals may be used in macro-instructions: numeric and non-numeric.

Numeric Literals -- A numeric literal may contain the numbers 0-9, the decimal point (.), the leading plus sign (+), or the leading minus sign (-). A decimal point may be included anywhere within the numeric literal except as the right hand character. The decimal point will be treated as an implied decimal point by the assembler. A numeric literal may contain only one decimal point and/or one sign.

Examples

13602

3

1.2

-3.9

+4235.06

The length of a numeric literal may be from 1-16 characters not including the sign and/or decimal point. The sign, if included, is carried internally in the standard sign position. It is not established as a separate character.

Non-Numeric Literals -- Non-numeric literals must be enclosed by quotation marks.

An alphanumeric literal consists of any combination of characters in the GE-425/435 character set, except the quotation mark. An alphabetic literal consists entirely of alphabetic and blank characters. A non-numeric literal may be from one to forty characters in length and must not be split between two lines.

Examples

"%NET-PAY"

" END OF JOB"

"TOTAL TAX"

"-16.83"

The "-16.83" is considered an alphanumeric literal because it is enclosed in quotation marks. It is carried internally as

-16.83

and therefore may not be used in calculations.

If -16.83 is needed in an arithmetic operation, it is entered as a numeric literal (without quotation marks) and is carried internally in one word as

 $168\overline{3}$

Absolute Entries

If a specific macro-instruction allows, a parameter may be a decimal integer. This may specify either a fixed index word or a control count. If used with a basic assembly language instruction, it may refer to any fixed machine location or value.

Macro-Instruction Address Modification

• Using Indexes -- An index may be specified as a modifier to a data name or a reference symbol used in the Operation Parameters of a macro-instruction. The index may be a fixed index word or a reference symbol and must immediately follow the data name or reference symbol. The index is separated from the data name by a comma. As many as three indexes may be specified, separated by commas. Literals may not be indexed. If any index is specified in a macro-instruction, the specified index overrides the index assigned to the file. If a data name which was described in the File Section requires any index other than the index assigned to the file, all necessary indexes (including the file's RECINDEX) must be specified following the data name.

Examples

Operation	Operation Parameters
GOTO	refsym, 4
LOAD	data-name-1, ABLE
LOAD	data-name-1, ABLE, BAKER, 6
MOVE	data-name-1, 4;data-name-2, 4

<u>Using Calcualted Addresses</u> -- In macro-instructions, simple expression may be formed by following the reference symbol parameters with an arithmetic operation (+ or -) and a decimal integer.

It is difficult to predetermine the number of instructions that will be generated by a macro-instruction. Therefore, calculated addresses (i.e., *+2 or refsym-3) should never be used to refer past a macro-instruction.

MACRO-INSTRUCTION DESCRIPTIONS

Macro Assembly Source Language functions include

Input/output

Data movement

Procedure control

Arithmetic operations.

Descriptions and usage of each of the macro-instructions in these four categories are presented in this chapter. An alphabetic list of all macro-instructions is given in Table A at the end of this section of the manual.

Use of Hardware by Generated Coding

During execution of the object coding produced by a macro-instruction, the contents of location 0 and the fixed index words 1, 2, 3, and 6 plus the hardware accumulator words 12 through 15 may be destroyed. This is sometimes necessary to produce the most efficient object coding.

Therefore, prior to using a macro-instruction, the programmer must make the following provisions:

- 1. Insure that the accumulator is located at 0-3.
- 2. Save the contents of location 0, 1, 2, 3, 6, and 12 through 15 if the contents will be needed following execution of the macroinstruction.

The use of the comparison indicators will be indicated in the description of the macro-instructions that may modify their settings.

Operation Parameters Format Notation

The notation shown in the chart below is used to describe the variable format of the Operation Parameters columns.

Operands	indicated by lower case words. They indicate the type of operands that a programmer may use.
data-name-	associated with an entry in the Data Division.
file-name-	defined with an FD entry in the File Section of the Data Division.
refsym-	associated with an instruction or pseudo-operation in the Procedure Division.
literal	the literal must be specified as described in Literal Entries, page
Choices	are enclosed in braces { }. One entry must be selected from those shown within a set of braces.
Options	are enclosed in brackets . Information contained within the brackets may be included or omitted.

KEY WORDS	are those which must be used in a macro- instruction. They are indicated by under- lined upper case words.
NOISE WORDS	are those which may be used to clarify (in English language) the meaning of a macro-instruction on the programming form. When used, they must be correctly spelled. They are indicated by upper case words.
Δ¹s	are indicated in the macro-instruction formats and must be included on the programming form.

INPUT/OUTPUT OPERATIONS

In the Macro Assembly Program, the programmer may include any of the logical record macro-instructions and service macros made available by the Extended Input/Output System. For a general discussion of the design objectives and operating philosophy (including label processing and error recovery procedures) refer to the Extended Input/Output System write-up.

To incorporate the Input/Output macros in a Macro Assembly Program, the programmer must also provide

- 1. a description of each file to be used by the macro-instructions.

 These files are defined in the File Section of the Data Division.
- 2. a list of the major input/output control routines to be included with this segment. The routines are defined with System Definition Parameters in the Environment Division.

A description of Input/Output macro-instructions follows.

READ		
Function:	To obtain the	next logical record from an input file.
Format:		
	Operation	Operation Parameters
Option 1.	READ	file-name-1
Option 2.	READ	file-name- $1\triangle \underline{TO} \triangle \left\{ \begin{array}{l} data-name-l \\ refsym \end{array} \right\}$

Notes:

 In Option 1, READ causes the location of the first word of the next logical record to be placed in the specified RECINDEX.
 The records may then be processed in the input buffer by using RECINDEX to modify all references to the logical record.

- 2. In Option 2, READ causes the location of the first word of the next logical record to be placed in the specified RECINDEX and, in addition moves the record into working storage or an output file buffer area. The second parameter may specify the reference symbol of a work area defined with a BSS in the Procedure Division, the data name of a logical record defined in the Working Storage Section, or a file name. The length of the record in file-name-1 determines the number of words to be moved.
- 3. The data record last accessed by a READ macro continues to be available in the input buffer via RECINDEX until the next READ for that file has been issued. However, once the current input data record has been released to an output file by a WRITE macro, further modification of the record in the input file will not be reflected in the output record. If the current input record has been released to an output file by a WRITEX macro, any further modification of the record in the input area, up to the point that another WRITE or WRITEX is given to the same output file, will be reflected in the output record.
- Once a record has been obtained by a READ file-name, it cannot be moved to Working Storage by a READ file-name TO refsym, since the latter macro will access the next data record instead of the record desired.
- 5. The contents of **RECINDEX** must not be changed by the object program.
- 6. An OPEN macro-instruction for the file must have been executed before executing the first READ macro-instruction for the file.

TITOITE	
WRITE	

Function: To release a record to an output file.

Format:

	Operation	Operation Parameter	<u>'s</u>
Option 1.	WRITE	file-name-1 FROM	refsym data-name file-name-2
Option 2.	WRITE	file-name-1	

Notes:

1. In Option 1, the WRITE macro-instruction moves the data record into the next available space in the current buffer area of file-name-1. The second parameter specifies the source of the data record to be moved to the output file buffer. The parameter may specify the reference symbol of a work area defined with a BSS in the Procedure Division, the data name of a logical record in the Working Storage Section, or the file name of an input file whose current record is to be moved. The programmer must insure that the source record conforms with the record format specified in the output file. If the output file is composed of F or FS¹ form records, the length is specified in the RECLNGTH entry in the file table. For V or VS² records, the length will be indicated in the record length word of the record to be moved.

The RECINDEX assigned to the file always references the first word of the last record placed in the current output buffer. The record remains available for processing in the output buffer via the RECINDEX until the next WRITE or WRITEX is issued for this file.

2. For Option 2, the WRITE macro-instruction involves no data movement by the Extended Input/Output System. It is a request for the next available space in the current buffer for file-name-1, in which the object program may build an output record. The location of the first word of the space made available will be in

Note: (1) F -- Fixed-Length Records. FS -- Fixed Length Record with Block Serial Number. (2) V -- Variable-Length Records. VS -- Variable-Length Records with Block Serial Number.

the named file's RECINDEX. The size of the space made available is determined by the contents of the RECLNGTH entry in the file table, for both fixed and variable length records. For V or VS, the programmer must set RECLNGTH in the file table to the desired value before giving the WRITE.

In contrast to the normal method of placing records in an output file with the first form of the WRITE macro-instruction (after processing of the record is completed), WRITE file-name-1 must be issued before data is moved to the output area. This is to insure the proper allocation of space and to avoid loss of data. The programmer must be careful with this form of the WRITE macro-instruction to avoid issuing a WRITE after the last output record of the file has been placed in the output area; a WRITE at this time would reserve an extra record space, resulting ir an additional output record of unknown content. Both forms of the WRITE macro-instruction may be used within a program to place records in the same output file. However, if both WRITE and WRITEX macro-instructions are used with the same output file, the file parameter IOMETHOD must specify EXCHANGE.

- 3. The contents of RECINDEX must not be modified by the object program.
- 4. An OPEN macro-instruction must have been executed for this file before the first WRITE is issued for the file.

WRITEX

Function:

To release a record without internal movement of the data

record.

Format:

Operation Operation Parameters

WRITEX file-name-1∆ FROM∆file-name-2

Notes:

- The IOMETHOD indicated for file-name-1 and file-name-2, must be EXCHANGE.
- 2. A data record can be released to only one file by the WRITEX macro-instruction. If the user desires to place the same data in more than one file, the WRITEX macro-instruction may only be used for one of the files. The records must be physically moved into the other files 'y the WRITE macro-instruction.
- 3. The WRITEX macro may only be used with form F or FS records.
- 4. An OPEN macro must be executed for this file before the first WRITEX is issued for this file.

(See the GE-425/435 Extended Input/Output System Reference Manual for details of WRITEX.)

OPEN	

Function:

To ready an input or output file for processing by Input/ Output macro-instructions.

Format:

Operation Operation Parameters

OPEN file-name-1;file-name-2;;file-name-n

Notes:

- An OPEN macro-instruction must be executed for a file before any Input/Output macro-instruction is processed for that file.
- 2. This macro-instruction produces a call to the OPEN Subroutine to ready the named files for processing. The Operation Parameters specify the file-names of all files to be opened at this time.

3. The OPEN subroutine performs all necessary functions such as initial rewind and header label processing of tape files. This subroutine establishes the proper linkage and controls necessary for the proper functioning of the Logical Record Macro Routines, Buffer Control Routines, and the Scheduling System.

CLOSE

Function:

To terminate the processing of input and output files.

Format:

Operation Operation Parameters

CLOSE file-name-1;file-name-2;...;file-name-n

Notes:

- 1. A CLOSE macro-instruction should be issued after completing the logical processing of one or more files.
- The macro-instruction produces a call to the CLOSE Subroutine.
- The "closing" of a file allows the Extended Input/Output System to write out any remaining blocks or partial blocks of records, to perform designated trailer label processing of output files. and to perform designated rewind procedures for both input and output files. The closing of a file removes it from the "active" list, and disengages it from the Scheduling System.
- 4. Files may be closed individually, or all at once. One advantage to closing the files as soon as their processing is completed is to allow the operator to stagger the physical work necessary to remove that file from the computer system, thereby reducing the time between jobs.
- The user must avoid issuing any macro-instruction naming a closed file, as the proper linkage and controls are not established. If this situation occurs, the Extended Input/Output System types an error message and halts the program.

TYPE

Function:

To perform requested typewriter Input/Output operations.

Format:

Operation Operation Parameters TYPE

Notes:

- 1. If a reference symbol is indicated, it must be assigned to a DCWC which points to the word containing the first character of the message to be typed. The number of DCWC's in the list beginning at refsym is specified by N. The DCWC's must be written by the programmer in the Procedure Division.
- 2. If the third parameter specifies a data-name, the Macro Assembly Program builds the required DCWC. The size of the field defined by data-name determines the number of characters to be typed.

EXTENDED INPUT/OUTPUT SYSTEM SERVICE MACRO-INSTRUCTIONS

Function:

To provide the logical software equivalents of peripheral hardware functions necessary to maintain the logical environment established by the Extended Input/Output System.

Format:

The following Service Macro-Instructions may be used

in the source program:

Operation	Operation Parameters	(Meaning)
BKSPTM *	<pre>file-name-1 [;(character)]</pre>	(Backspace to tape mark)
RELEASE	file-name-1	
FDSPTM *	<pre>file-name-1[;(character)]</pre>	(Forward space to tape mark)
BKSPNBLK *	file-name-1;N	(Backspace N blocks)
REWIND *	file-name-1	
WRITETM *	<pre>file-name-l[;(character)]</pre>	(Write tape mark)
FORCE *	file-name-1	
RDLABEL	file-name-l <u>\INTQ\r</u> efsym;N	
WRLABEL	file-name-l <u>AFROMA</u> refsym;N	
CHKPT		

For a complete description of the Service Macro-Instructions listed above, refer to the Extended Input/Output System write-up.

DATA MOVEMENT

MOVE

Function:

To transfer data from one item to another conforming to the description of the receiving item. The receiving field picture determines the format of the data.

Format:

MOVE Coperation Parameters

data-name-1
literal; data-name-2

Notes:

- 1. If the receiving field is non-numeric (alphabetic or alphanumeric), data is transferred left-justified with space fill or truncation provided at the low order end as required.
- 2. If data is transferred to a numeric area, MOVE automatically aligns decimal point position and provides truncation or zero fill at either, or both ends as required. If the decimal point position is not supplied, the data is right justified in the receiving area.
- 3. If the description of the receiving area specifies editing, movement of data takes place as described above, the data is edited according to the image of the receiving area (refer to Picture, page 2-24).
- 4. If an numeric area is being moved to a data item defined as an index, conversion from decimal to binary is performed; data being moved from an index to a numeric item is converted from binary to decimal.
- 5. The contents of data-name-1 are not disturbed.
- 6. Table-1 indicates the types of moves that can be performed on elementary items by the MOVE macro-instruction. An entry in the table indicates a valid MOVE; the number in parentheses gives the number of an example in Table-2 that shows the content of the receiving field after the move.

2 - 77

^{*} Precede with a RELEASE macro-instruction.

Table 1. Types of moves Performed on Elementary Items by MOVE

MOVE to data-name-2 MOVE from data-name-1	Numeric- Assumed Decimal Point	Numeric- Integer	Numeric- Edited	Alpha- numeric	Alpha- numeric Edited	Alpha- betic
Numeric-Assumed Decimal Point	(1)	(2)	(3)			
Numeric-Integer	(4)	(5)	(6)	(7)	(8)	
Numeric-Edited				(9)	(10)	
Alphanumeric	(11)*	(12)*	(13)*	(14)	(15)	(16)
Alphanumeric-Edited				(17)	(18)	(19)
Alphabetic				(20)	(21)	(22)

^{*} The contents of the field being moved must be a positive numeric integer.

If any MOVE is attempted that is not indicated by an entry in Table-1, an error message will be issued. Table-1 is also applicable to types of literals that may be moved to data-name-2.

Table 2. Content of the Receiving Field after MOVE

EXAMPLE	FROM:	data-name-1	TO: da	ta-name-2
NO.	Picture	Value	Picture	Value
1(a)	9 v 99	1V23	99 v 9	01V2
(b)	999 v 99	123V45	99999799	00123 V 45
2 (a)	9 v 99	1V23	999	001
(b)	999 v 99	123 V 45	999	123
3(a)	9 v 99	1V23	9.99	1.23
(b)	999 v 99	123 V 45	Z,ZZ Z.99	123.45
(c)	999 v 99	00000	Z,ZZZ.99	.00
(d)	999 v 99	123 V 45	Z,ZZZ	123
4(a)	999	123	9999 V 9	0123V0
5(a)	999	123	9(4)	0123
(b)	9(4)	1234	999	234
6	999	123	ZZZ .99	123.00
7	999	123	X(4)	123∆
8	999	123	XXBX	12∆3
9	9.99	1.23	X(4)	1.23
10	9.99	1.23	ххвх	1.Δ2
11	XXX	123	9799	3V00
12	XXX	123	9(4)	0123
13	XXX	012	ZZZ	12
14	xxx	123	X(4)	123∆
15	XXX	123	XXBX	12∆3
16	XXX	ABC	A(4)	АВС∆
17	XXBX	AB∆C	X(5)	AΒΔCΔ
. 8	XXBX	AB∆C	XXBX	AB
.9	XXBX	AB∆C	A(6)	ΑΒΔΟΔΔ
20	AAA	ABC	X(4)	ABC∆
1	AAA	ABC	XXBX	AB∆C
2(a)	A(5)	ABCDE	AAA	ABC
(p)	A(5)	ABCDE	A(7)	ABCDEAA

^{7.} When a group is being moved, it is treated as an alphanumeric to alphanumeric MOVE (see Note 1).

12/6/63

LOAD

Function:

To transfer the contents of the specified data area to the standard accumulator (words 0-3).

Format:

Operation Operation Parameters

LOAD data-name-1

Notes:

1. Data-name-1 may not contain more than 16 characters.

2. The size of the accumulator is determined by data-name-1. Transferred data is right justified in the accumulator, and unused portions of the accumulator are set to zero.

3. Data-name-1 is not affected.

Examples:

Source Field		Accumulator
Picture	Value	After Execution Of Load
A(7)	ABCDEFG	[OABC DEFG]
xx	AB	<u>[00AB]</u>
99 v 999	12345	[0001 [2345]
9999₹9	12345	[0001 [2345]
A(4)	1234	[1234]
A	A	[000A]
X(6)	-12.34	[00-1[2.34]
X(8)	ABCD-123	[ABCD [-123]

UNLOAD

Function:

To transfer the contents of the standard accumulator (words 0-3) to a data area.

Format:

Operation

Operation Parameters

UNLOAD

data-name-1

Notes:

1. Transferred data is right-justified in the receiving area.

- 2. Data is transferred beginning at the low order end of the accumulator. The size of the receiving area determines the number of characters which are transferred. The remaining working accumulator characters must be zero.
- 3. If the receiving area is greater in length than 16 characters, all characters beyond the 16th character of the receiving area are set to zero.
- 4. If the receiving area contains editing characters, the data from the accumulator will be edited according to the edit image of the receiving area. The accumulator is assumed to contain the same number of characters and decimal places as appear in the receiving area.
- 5. The contents and size of the accumulator may be changed by this operation.

Example:

Accumulator	Receivi	ng F ield
Before Unload	_	Value
See Note 5	Picture	After Unioad
0001 2345	\$*, ***. * *	\$* *1 23. 45
[0000 0000]	\$*, ***. **	
[0000 [0001]	\$*, ***. **	\$****. 01
[0000 0001	\$\$, \$ \$\$. \$ \$	\$. 01
[0012 3456]	\$ \$, \$ \$\$. \$\$	\$1, 234. 56
0123	999	123
0012	999	012
00AB CDEF	X(6)	ABCDEF

PROCEDURE CONTROL OPERATIONS

COMPARE

Function:

To compare the contents of two data areas and to transfer control to one of three specified reference symbols based upon the result of that comparison. The possible results of the compare are:

if data-name-1 is greater than data-name-2 transfer control to refsym ~ 1

if data-name-1 is $\underline{\text{equal to}}$ data-name-2 transfer control to refsym~2

if data-name-1 is $\underline{\text{less than}}$ data-name-2 transfer control to $\texttt{refsym}{\sim}\,3$

Format.

Operation Operation Parameters COMPARE data-name-1 literal-1 compared (data-name-2); literal-2 compared (data-name-2); literal-2 compared (data-name-2); literal-2 compared (data-name-2); literal-2 compared (data-name-2); literal-2 compared (data-name-2); literal-2 compared (data-name-2); literal-3

Notes:

- 1. The collating sequence of the Computer Department Standard Character Set is the basis for all comparisons.
- 2. Comparison of numeric items. A numeric item may be compared only to another numeric item. The comparison of numeric items is based upon their algebraic values. The item length, in terms of the number of digits, is not itself significant. Zero is a unique value regardless of the length or sign of an item. A numeric item used in a Compare instruction must not exceed 16 characters.

3. Comparison of non-numeric items.

The following descriptions are written as one would logically think of comparing the two fields, not as it is being implemented by the object program.

- Items of equal length. The relationship between two nonnumeric items is determined by comparing their corresponding characters, beginning with the high order character of each item.
- Items of unequal length. The relationship is determined as described above. If the shorter item is exhausted, it is considered to be less than the other item, unless the remaining characters in the other item are spaces; in which case, the two items are considered equal.
- 4. The reference symbol for a particular result may be omitted by placing its delimiting semicolon immediately after the previous delimiting semicolon. If the reference symbol for a particular condition has been omitted and that condition is met, the program proceeds to the next line of coding.

Example:

Operation	Operation Parameters
COMPARE	FLD1;FLD2;RS1;RS2;RS3
COMPARE	FLDl;FLD2;RSl;;RS3
COMPARE	FLD1;FLD2;;;RS3
COMPARE	FLD1;FLD2;;RS2
COMPARE	FLD1;FLD2;RS1

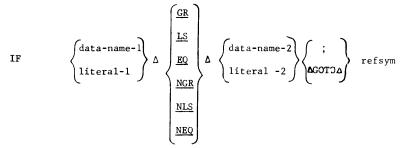
5. The setting of the comparison indicators following the execution of this macro-instruction is unpredicable.

IF

Function: To branch to a specified reference symbol or to continue in-line coding based upon the existence of a specified condition.

Format:

Option 1. Operation Operation Parameters



Option 2.

Option 3.

IF
$$sw-name-J$$
 $\Delta \left\{ \begin{array}{c} \underline{ON} \\ \underline{OFF} \end{array} \right\} \left\{ \begin{array}{c} \vdots \\ \underline{\Delta GOTO} \Delta \end{array} \right\}$ refsym

Notes:

1. Option 1

IF, Option 1, follows the rules specified for COMPARE

2. Option 2.

IF, Option 2, operates only upon numeric items.

3. Option 3.

IF Option 3, operates only upon items which have been described as switches. (See Use, page 2-35.) A switch is off when it contains zero and on when it contains any other value.

4. The word entries in the Operation Parameters must be separated from one another by a single space.

Examples

Operation	Operation Parameters
IF	PAY~NOAEQATCPAV~NOAGOTOAPROCESS
IF	PAY~NOAEQATCPAY~NO;PROCESS
IF	99999AEQAPAY~NO;END
IF	GROSSAMINUSAGO TO A ERROR
IF	SW14ON4GOTO4RDTC
IF	SW1AOFF; READM

5. The setting of the comparison indicators following the execution of this macro-instruction is unpredictable.

SETSW

<u>Function:</u> To set a programmer-specified switch to an on or our condition.

Format:

Operation	Operation Pa	rameters		
SETSW	sw-name- $1\triangle$ $\left\{\begin{array}{c} \underline{ON} \\ \underline{OFF} \end{array}\right\}$	\Rightarrow ; sw-name-2 \triangle $\left\{ \begin{array}{c} \underline{ON} \\ \underline{OFF} \end{array} \right\}$	} ;;sw-name-10∆	ON OFF

Notes:

- Any number of switches from one through ten may be set on or off in any combination.
- 2. SETSW operates only upon items which have been described as switches. (See Use, page 2-35.)
- 3 "SETSW ON" sets the specified switch to a value of one; "SETSW OFF" sets the specified switch to a value of zero.
- 4 The switch name must be separated from the specified condition by a single space.

GOTO

<u>Function:</u> To transfer control to a specified reference symbol.

Format:

Operation	Operation Parameters
GOTO	refsym

Notes

- 1. The GOTO results in an unconditional branch to the reference symbol specified in the Operation Parameters.
- 2. The reference symbol can be modified by only one index word. This index must be a fixed index word.

GOTO... DEPENDING ON

Function:

To transfer control to one of n specified reference symbols based upon the value of a specified data name.

Format:

Operation	Operation Parameters
GOTO	refsym-1;;refsym-100 AD EPENDING∆ON ∆data-name-1

Notes:

- 1. From one to 100 reference symbols may be specified as parameters.
- 2. Data-name-1 must be described as a numeric integer.
- 3. If the value of data-name-1 is less than one or greater than the number of specified reference symbols, the next "in-line" operation is executed.
- 4. Otherwise, control is transferred to the reference symbol whose position within the macro-instruction corresponds to the value of data-name-1. If data-name-1 equals one, a branch to refsym-1 is executed; if 15, a branch to refsym-15 is executed; etc.
- Each reference symbol can be modified by only one index word.
 This index must be a fixed index word.

HALT

Function:

To halt the object program, after insuring that all critical computer operations have terminated.

Format:

Operation Operation Parameters

HALT XXXX

Notes:

HALT transfers control to the Basic Input/Output System Halt Routine, which releases control to the operator, after typing the message: HLTAXXXX. The operator may continue processing by depressing the Run switch. Control will be transferred to the instruction immediately following the HALT. (See the Basic Input/Output System write-up, Halt Routine.)

END OF PROGRAM

EOJ

Function: To terminate processing by the object program.

Format:

Operation Operation Parameters
EOJ

Notes:

- EOJ transfers control to the Standard Job-Termination Routine, which--
 - insures that all I/O operations have terminated
 - types a standard end-of-job message
 - searches the system tape for the Program Monitor
 - loads and transfers control to the Program Monitor.

ABORT

Function:

To terminate processing by the object program under error conditions.

Format:

Operation

Operation Parameters

ABORT

Notes:

- 1. ABORT types a standard message, takes a memory dump, and transfers control to the Program Monitor.
- 2. ABORT is provided as a standard, error-condition exit from the object program.

ARITHMETIC OPERATIONS

ADD AND ROUND

ADDR

Function:

To add two numeric items and place the rounded result in the second item, or a third item, if specified.

Format:

Operation

Operation Parameters

Option 1.

ADDR

{data-name-1}; data-name-2 [;data-name-3]

Option 2.

ADDR

data-name-1; literal; data-name-3

Notes:

- 1. Any data name used as a parameter must be defined as a numeric elementary item in the Data Division.
- 2. Any data name used as parameter one or parameter two must not contain editing characters and must not exceed 16 characters in length.
- 3. Any literal used as a parameter must be numeric and must not exceed 16 digits.
- 4. The result of an addition may be sent to a third numeric item specified as data-name-3. If the description of data-name-3 contains editing characters, editing takes place as described in the MOVE (see page 2-77.)
- 5. If a data-name-3 is not specified, the result is placed in data-name-2 (Option 1).
- 6. Required data conversion and decimal point alignment are automatically provided and the result is rounded to the description of the receiving item.
- 7. If the magnitude of the result exceeds that of the receiving item, the excess high order characters are truncated.

8. If the second parameter is a literal (Option 2), then data-name-3 must be specified to receive the sum.

ADD AND TRUNCATE

ADDT

Function:

To add two numeric items and place the result in the second item or a third item, if specified.

Format:

Operation

Operation Parameters

Option 1.

ADDT

{data-name-1};data-name-2[;data-name-3]

Option 2.

ADDT

data-name-1; literal; data-name-3

Notes:

The notes under ADDR also apply to ADDT, except that under note 6, the result is truncated to the description of the receiving area.

SUBTRACT AND ROUND

SUBR

Function:

To subtract one numeric item from a second numeric item and place the rounded result in the second item and/or third item, if specified.

Format:

Operation

Operation Parameters

Option 1.

SUBR

 $\left\{ \frac{\text{data-name-1}}{\text{data-name-2}} \right\}$; data-name-2 [; data-name-3]

Option 2.

SUBR

data-name-1; literal; data-name-3

12/6/63

Notes:

- Parameter one is subtracted from parameter two.
- Any data name used as a parameter must be defined as a numeric elementary item in the Data Division.
- 3. Any data name used as parameter one or parameter two must not contain editing characters and must not exceed 16 characters in length.
- 4. Any literal used as a parameter must be numeric and must not exceed 16 digits.
- The result of a subtraction may be sent to a third numeric item specified as data-name-3. If the description of data-name-3 contains editing characters, editing takes place as described in the MOVE (see page 2-77).
- If data-name-3 is not specified, the result is placed in dataname-2 (Option 1).
- Required data conversion and decimal point alignment are automatically provided and the result is rounded to the description of the receiving area.
- 8. If the magnitude of the result exceeds that of the receiving area, the excess high order characters are truncated.
- 9. If the second parameter is a literal (Option 2), then data-name-3 must be specified to receive the difference.

SUBTRACT AND TRUNCATE

SUBT

Function:

To subtract one numeric item from a second numeric item and place the result in the second item or a third item, if specified.

Format:

Option 1. SUBT SUBT Substitute data-name-1; data-name-2; data-name-3

Option 2. SUBT data-name-1; literal; data-name-3

Notes:

The notes under SUBR also apply to SUBT, except that under note 7, the result is truncated to the description of the receiving area.

MULTIPLY AND ROUND

MPYR

Function: To multiply two numeric items and place the rounded product in the second item or a third item, if specified.

Format:

	Operation	Operation Parameters
Option 1.	MPYR	{data-name-1};data-name-2[;data-name-3]
Option 2.	MPYR	data-name-1; literal; data-name-3

Notes:

- 1. Any data name used as a parameter must be defined as a numeric elementary item in the Data Division.
- 2. Any data name used as parameter one or parameter two must not contain editing characters and may not exceed eight characters in length.
- 3. Any literal used as parameter must be numeric and must not exceed eight digits.

- 4. The product may be sent to a third numeric item specified as data-name-3. If the description of the third item contains editing characters, editing takes place as described in the MOVE (see page 2-77).
- 5. If data-name-3 is not specified, the product is placed in data-name-2 (Option 1).
- 6. Required data conversion and decimal point alignment are automatically provided and the product is rounded to the description of the receiving area.
- 7. If the magnitude of the product exceeds that of the receiving area, the excess high order characters are truncated.
- 8. If the second parameter is a literal (Option 2), then data-name-3 must be specified to receive the product.

MULTIPLY AND TRUNCATE

MPYT

Function: To multiply two numeric items and place the product in the second item or a third item, if specified.

Format:

	Operation	Operation Parameters
Option 1.	MPYT	data-name-I; data-name-2 [; data-name-3]
Option 2.	MPYT	data-name-1;literal;data-name-3

Notes:

The notes under MPYR also apply to MPYT except that under note 6, the product is truncated to the description of the receiving area.

DIVIDE AND ROUND

DIVR

Function:

To divide one numeric item by another and place the rounded quotient in the second item or a third item, if specified.

Format:

	Operation	Operation Parameters
Option 1.	DIVR	data-name-1;data-name-2[;data-name-3]
Option 2.	DIVR	data-name-1;literal;data-name-3

Notes:

- Parameter one is the divisor. Its data description may not contain editing characters and must not exceed eight characters in length.
- 2. Parameter two is the dividend. Its data description may not contain editing characters and must not exceed sixteen characters in length.
- 3. Any data name used as a parameter must be defined as a numeric elementary item in the Data Division.
- 4. Any literal used as a parameter must be numeric.
- 5. The quotient may be only eight characters long. Because of this, the divisor must be greater than any portion of the dividend which occupies the two high-order words of the quadruple accumulator, when the dividend has been right justified.

Example:

The following division is not allowed:

Dividend	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	9
Divisor	0	0	0	0	0	0	0	5								

This division would result in a nine-character quotient.

The following division is allowed:

Dividend 0 0 0 0 0 1 2 3 4 5 6 7 8 9 9

Divisor 0 0 0 0 0 5 0

This division would result in an eight character quotient.

- 6. The quotient may be sent to a third item specified as data-name-3. If the description of data-name-3 contains editing characters, editing takes place as described in the MOVE (see page 2-77).
- 7. If data-name-3 is not specified, the quotient is placed in data-name-2.
- 8. If the second parameter is a literal (Option 2), then data-name-3 must be specified to receive the quotient.
- 9. Required data conversion and decimal point alignment are automatically provided and the quotient is rounded to the description of the receiving area.
- 10. If the number of characters in the quotient is smaller than the number of characters in the receiving field, the quotient will be placed in the receiving field according to the decimal point alignment. Zeros will be filled in any vacant positions of the receiving field. (See Picture, Page 2-24). If the number of characters in the quotient exceeds that of the receiving area, the excess high order characters are truncated.

DIVIDE AND TRUNCATE

DIVΤ

Function:

To divide one numeric item by another and place the truncated result in the second item or a third item, if specified.

Format:

Notes:

The notes under DIVR also apply to DIVT except that under note 9, the quotient is truncated to the description of the receiving area.

VIII. SEGMENTED PROGRAMS

In the Macro Assembly Program, provisions are made to produce relocatable segments for use within the GE-425/435 Operating System environment. Each segment assembled must be identified by a name punched on the output header card. The SEGMENT entry in the Identification Division enables the programmer to supply this name and to specify a relocatable or absolute assembly. (See Chapter IV, Identification.)

Within a segmented environment, a method to indicate communication between segments must also be provided. This is accomplished by including basic assembly language pseudo-operations to indicate the reference symbols that will be used for references between segments, and to indicate the data descriptions that are common to the segments. Reference should be made to Chapter IV, Relocatable Segments and Chapter VI, Pseudo-Operations in the Basic Assembly Language section of this manual. A brief description of the pseudo-operations used for segmented programs is included in this chapter.

INTERNAL AND EXTERNAL GLOBAL SYMBOLS

To indicate the symbols used to transfer from one segment to another, the Macro Assembly Program utilizes two pseudo-operations which are included in the Procedure Division: DIG and DXG.

DEFINE INTERNAL GLOBAL	DIG

DIG is used to indicate the global reference symbols which are defined in this segment. The names of these references are required at load time to enable separately assembled segments to be linked.

Format:

Operation	Operation Parameters		
DIG	refsym-1,refsym-2,		

The Operation Parameters columns contain a list of internal global references that are defined in the Reference Symbol columns of instructions or pseudo-operations written in this segment. The symbols, separated by commas, must follow the conventions established for reference symbols (see page 2-59).

Example:

Ref. Sym.	Operation	Operation Parameters
	•	
	DIG	PRTCHECK
	DIG	BONDED, TIME
	•	
BONDED	MOVE	DECODE; TEMP4
	•	
	•	
PRTCHECK	IF PA	$Y \sim NO \triangle EQ \triangle TCPAY \sim NO \triangle GOTO \triangle EXIT$
	•	
minari	DEAD	D AXIMOM D
TIME	READ	PAYMSTR

DXG is used to indicate the global reference symbols which are defined in other segments. The names of these references are required at load time to enable separately assembled segments to be linked.

Format:

DEFINE EXTERNAL GLOBAL

Operation	Operation Parameters
DXG	refsym-1,refsym-2

The Operation Parameters columns contain a list of external global reference symbols that are referred to as Operation Parameters in the Procedure Division of this segment. The reference symbols, separated by commas, must follow the conventions established for reference symbols (see page 2-59.)

Example:

Operation	Operation Parameters		
•			
•			
DXG DXG	BONDED, TIME PRTCHECK		
•			
•			
P X B	BONDED, 6		
•			
•			
P X B	TIME, 6		
•			
•			
P X B	PRTCHECK, 6		

COMMON DATA AREAS

If a file, record, or work area is referred to by macro-instructions in several separately assembled segments, their descriptions must appear in each segment. Repetition of this description is necessary in order to provide the macro-generators with the characteristics of the data during the assembly of each segment. These characteristics (i.e., mode, size, relative position within the word) determine the patterns of instructions to be generated. Since it is only necessary to allocate storage for the areas within one segment, the AREADEF file parameter (see page 2-47) and the DGRR, DGRE, and DGR pseudo-operations (see Basic Assembly Language section, Chapter VI, Pseudo-Operations) are included in the macro assembly source language.

DXG

AREADEF

The AREADEF file parameter is used to indicate whether the buffer area(s), DCW list(s), and File Table are to be generated by the Macro Assembly Program, supplied by the programmer in this segment, or have been supplied in a separate segment. This file parameter is written in the file parameter list associated with each file described in the File Section.

The parameter has the following format:

$$\begin{array}{c} \text{AREADEF} \ \left\{ \begin{matrix} \text{GENERATE} \\ \text{DEFINED} \\ \text{REMOTE} \end{matrix} \right. \left[\begin{matrix} \text{refsymbol} \end{matrix} \right] \end{array} \right\}$$

For a discussion of the options see page 2-47.

DEFINE GLOBAL REFERENCE REMOTE

DGRR

The DGRR pseudo-operation is used to indicate a chain of common data that has storage allocated in some other segment (see DGR). The DGRR may not be used in the File Section.

Format:

Ref. Sym.	Operation	Operation Parameters
refsym-1	DGRR	

Example:

Data-name	Level	Sync	Use	Picture
PAYREC PAYDTL PAY~NO ORG DATE	DGRR 01 02 02 02			9(5) 9(4) 9(6)

In the example, because of the DGRR, the assembly program will classify its reference symbol as external. The characteristics of the items will be saved for use by the macro-generators. All the data names following PAYREC will be treated as relative to PAYREC until:

- the assembler encounters another DGRR which reinitiates the entire process with a new external global; or,
- the assembler encounters a DGRE (Define Global Reference Ends) which terminates the reference area; or,
- the Procedure Division line is encountered.

DEFINE GLOBAL REFERENCE ENDS

DGRE

When the global reference is defined remotely, an indication must be given when the definition is completed. This will resume normal assembly and allocate storage. DGRE performs this function. It may not be used in the File Section.

Format:

Ref. Sym.	Operation	Operation Parameters
	DGRE	

DEFINE GLOBAL REFERENCE

DGR

One segment of the program must allocate storage to the common data areas that are referred to in other segments. The DGR pseudo-operation is used to indicate the beginning of common storage in this segment. This pseudo-operation may not be used in the File Section.

Format:

Ref. Sym.	Operation	Operation Parameters
refsym-1	DGR	

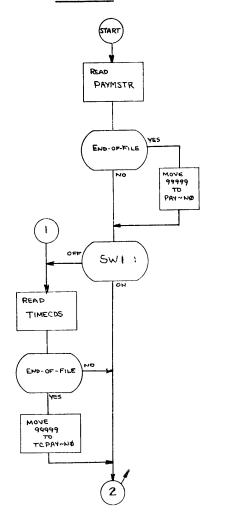
Example:

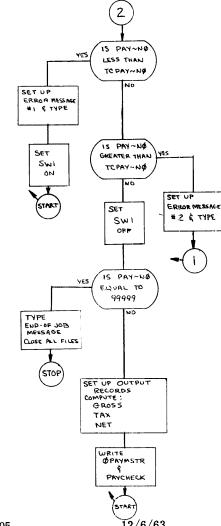
Data-Name	Level	Sync Use	Picture
PAYREC	DGR		
PAYDTL	01		
PAY~NO	02		9(5)
ORG	02		9(4)
DATE	02		9(6)

Because of the DGR, the assembly program indicates PAYREC is an internal global reference. Characteristics of the items are saved for use by the macro-generators, and storage is allocated for the logical record described following the DGR pseudo-operation. The first entry following the DGR in the Working Storage Section must be assigned level 01.

PAYROLL APPLICATION

FLOW CHART



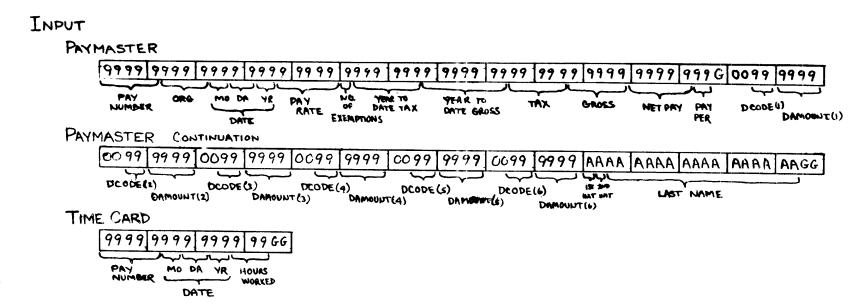


2-104

12/6/63

2-105

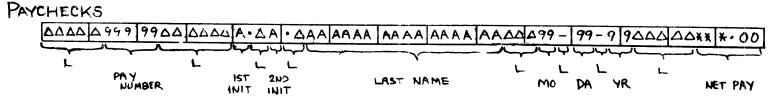
12/6/63



OUTPUT

PAYMASTER

SAME FORMAT AS INPUT PAYMASTER



LEGEND

L= LITERAL

G= GARBAGE

PROGRAM				PROGRAMMER		D	12-6-63	PAGE	° 4
PROGRAM	PAYROLI	- SAMPLE	PROBLEM	CHARLIE BROWN			12-6-63		
<u>.</u>		FERENCE SYMBOL	OPERATION		PERATION PARA	METERS			
	ř								IDE
SEQUENCE	E			PICTURE	40 41 OCCURS	V A	LSE		70 77
•	7 8 9	DATA NAME	17 LEVEL 20 C 22 E 24 28						
0.0.0.0.1.0	7.0	F.N.T.L.F.L.C	ATUONADIV	1,5,1,0,0,0				L Ll L	P-H
0.0.0.0.2.0				0 1 2 2 2 2 2			<u> </u>	LLL	
10.0.0.3.0			TITLE P.	AV RIOLL L SAMPLE	PROBLE	<u>. M </u>		<u> </u>	
2.0.0.0.0.0.	1 1 1		TDCNI	A.Y.I.	1 1 4 4 4 4	<u> </u>			
			NTADIVISI	Ø.N.			the state of the state of the	4 1	
2.0.0.0.5.0		-	SRMETH&DC.	A.I.I.			<u> </u>		
7 0 0 0 B 0			DEFSCHEDI	N.T.E.R.P.R.E.T.I.V.E.				المستنبط	
<u>, </u>		وطا القالبالا الالتا فلينتقان		O.N.~. O.Y.E.R.LAY			L.,_L.,_L., L.,_L., L.,		
0.0.0.8.0				CIVE COLUMN	1		وماليدة بطار فالطار فالطار فالتناوي والم		
0.0.0.9.0				L. &. S. E. I	· :			المتاسية التالية	
7-0-0-1-0-0			EFE ØR E	Ø R	*		and a contract to the contract	فالبيها وال	.i. i
0.0.0,1.1.0		چد عندید رید د چندچ	DEFCHKPTP		1				
2.0.0.1.2.0	T DIA.	LVIGAAL						1	
2,0,0,1,3,0	FIL	لَــــــــــــــــــــــــــــــــــــ	11 12 N - 1 - 1 - 1 - 1 - 1						
0.4.1.4.0	P	AY MISITIR							
0.0.1.5.0			DEVICE	<u>4.0.3 </u>					
0,0,0,1,5,5	1 1 1 .		COMMAND R.	T.B		<u> </u>			· I .
0,0,0,1,6,0		_ 1_1 _ 1 _ 1 _ 1 _ 1 _ 1	I OMETHODE	X.C.H.A.N.G.E			Landar alla della		
0,0,0,1,7.0			RECINDEXA			<u> </u>	<u>1</u>		1
0.0.0.1.8.0			FILETYPEI	N.P.U.T.			1.1.4.4.4.4.4.4.4.1	1 1 1 1 1	1
0.0.0.1.9.0				N.D.L.			1. 1	1. 1 1 . 1	. 1 .
0.0.0.2.0.0						 	<u> </u>		. 1
0,0,0,2,1,0			PRIGRITYL				والمفارية بنطر بالمسترك يساوي المستمسينة	an a chi fi fi	. 1 '
0.0.0.2.2.0			LABELDEFS.	T.A.N.D.A.R.D.			المفاصطيطات فالتفار والمتلكات	1. 1. 1	. 1 '
0.0.0.2.3.0				1.0.0.			According to the state of the s	1. 1 4 1 3	' 1 '
0.0.0.2.4.0				E.C.D. G.O.O.I. P.A.Y MS.J. R	1.0.0.1	0.0.0.0.0.0.0.4.0.0.1.4.	منط ملايطينا فالماطيين	1 1 1 1 1	. 1
0,0,0,2,5,6		AYREC _	0.1			_ <u> </u>	التالة عطاما فالتاب والهالها	1 1 1 1 1	- 1
0.0.0.2.5.0		A Y ~ N.O.	0.2	(15.)		Carrier and Carrie	1 1 1 1 t 1 1 t 1 1 1 1 1 1 1 1 1 1 1 1	_1 _ 1 _ 1 _ 1 _ A	· +
0 0 0 0 2 7 0		R.G.		(4) + + + + + + + + + + + + + + + + + + +		الطبيبة للقارط الانتهار والطبياة الدار ويتداب <u>نا بطالط المنتسب</u>	and a second contract of the	مالتينية بيسيين	* +
0.0.0.2.8.0		ATE	+				المائينات عددسيف الباعد الاالا	$z=\pm 1, \pm 2, \ldots, t$	
				9		. <u> </u>	Education of the Community of the Commun	$x:=1,\ldots,4,\ldots,r:=-3$	
2.0.0.2.9.0			1 1 1 1 1 1 1 1 1 1	9		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	بالما فالطالف للمراعات فالتنفيذان		
وبفيعتونه		<u>A </u>		9			اعتاد فالمتاهينية فياسا	لأشتاب با	- i.
2,0,0,3,1,0		R		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	المائينية المستقيلة الا	.i. i 1 i i	1
ة تختفتونونو		AXRATE					and the second s	به مداد به از	
0.0.0.3.3.0		<u> </u>		9,9,9,V,9,			4	4 4 A A A	1 4 4
0, 4, <u>5,0,0,</u>		Ť D T A X		(9,9,9,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0			المساعة عاد معدان العدادات عاملات		
0,0.0.3.5.0		T.D.G.R.B.S.S						i. j., 1	
0,0,0,3.6,0		A.X.		19.9.V.9.9				ستنايا بوريو	
2,0,0,3,7,0		ساسك، كبلاهسكا		1 4.4.1 1.4.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.		in a late of the first terms of the second control of the second c		A. A. A. Marie	
0.0.0.3.8.0		ET PAY		'd' 4 ' A ' A ' A ' A ' A ' A ' A ' A ' A	+++	and the second control of the second control		4	
0.0.0.3.9.0		A.Y.P.E.R.		19	++++++			4 . 4 . 4	1
0,0,0,4,0,0		E.D.U.C.T.					المتالم المالية المساها المسالم المسالم		T.L.
0.0.0.4.1.0		C. Ø. D. E		<u> </u>	L. C. L.	والأراز والطاع والانتفار والمستور المستور المستور	4 3 4		
0.0.0.4.2.0		AMOUNT		.9V.9.9.	•				
0.0.0.430		SIINIT	0 2 A	The second forms of the second					
		ND INIT	0 2 IA						

Sample Problem Written in Macro Assembly Language

PROGRAM				PROGRAMMER		DATE	PAGE	OF .
SF	m P	LE PAYROLL PR	COBLEM			12-6-63	2	~ 4
	Ţ	REFERENCE SYMBOL	OPERATION	OPE	RAT ON FARAMETERS			1
SEQUENCE	9							IDENT.
Jugoznez	1	DATA NAME	17 18 19 20 C 22 E 2	PICTURE	O CURS	VALUE		IDENT.
1	7 8	9 16		25 40			7.6	77 0
0,0,0,4,5,0	\perp	LASTNAME	0,2	A.(,1,6,), , , , , , , , , , , , , , , , , ,	<u> </u>		4 1 1 1 1	P.A.Y.
0.0.0.4.5.5			Δ.2	X.X		1 4 1 1 4 1 1 1 1 1 1 1 1	1 1 1 1 1	
0.0.0,4.6.0		TI ME CDS	FD					T .
0.0.0.4.7.0		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		0.1,0.1.	<u> </u>			1
0,0,0,4,8,0			COMMAND	R.C.D.				
0.0.0.4.9.0			EØFADDR	E.N.D.2				1
0.0.0.5.0.0			0.1					+ + +
0, 0, 0,5,1,0		TCPAY ~ NO	0.2	9.(.5.)				
0.0.0.5.2.0	1	TCDATE	0,2 U	9.(.6.)				
0.0.0.5 3.0	1-1-	H.R.S.W.K.D.	0.2	9.9.			1 _1	
0.0.054.0	1	OPAY MSTR	F.D.	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	- I		and the day day of the a	4 - 4 - 4 - 4
0.0.0.55.0	1		DEVICE	0.4.0.4	+ + -+ + - 1 1 +111111111-			· · · · · · · · · · · · · · · · · · ·
0.0,0560	1		C MM AN D	W.T.B.	+ + - + - + - + - + - + - + - + - + - +	والمرازين ويلف فللباب والمتلاء بالتلامية	4 4 4	
0.0,05.7.0	1	t t t t t t t t -		EXCHANGE	+ +			4 <u>+</u> :
0.0.05.8.0	1		FILETYPE	& U.T.P.U.T	# #	L	CO () A 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	
0.0.05.9.0	+						the second of the	41 A
0.0.0.5.0.6	 			3	+ + +- , b-+-b-+-1 i d			
	+		PRIORITY	2		والما الماست الاسطار لماسات الهاسات	- 7 1 a 1 a	1 . 1
0.0.0.6.1.0		and the section of the section of the section of		S.T.A.N.D.A.R.D.		يا والقالم والمستقد فالعالقالية للأسطال فالك		1.4
0.0.0.4.3.0	+ ,		LABELØPI	11.0.1	<u> </u>			1
0,0,0,6,3.0				G. E. C. D. O. O. O. I . P. A. Y . M. S. T. R.	1,2,0,1,0,0,0,0,0,0,0,4,0,0,1,4,,,	ورايين والور والصياطة فمستسطستك للالما	. 4 .4 . 5 . 7 . 4	Land Bridge
0.0.0.640	ļ	P.A.Y.R.E.C.	0.1	X.(.1.2.0.)	- a			L
0.0,0,6.5.0	1	₽ъ₽Ŋ С.H.E. C.K	F.D.		<u> </u>			1 1
0.0.0.6.6.0			D E V I C E	0.3.0.1.				1
0,0,0,6,7,0		والمستوالم المناف والمساهدا		PRE	A -4			
0.0.0.6.8.0		بالمستع بمالية	E Ø F A D D R	P.R.E.R.R.Ø.R.	<u> </u>		4 1. 4 1. 14	11
0,0,0,6,9,0	1	PAYCHK.	ــــــــــــــــــــــــــــــــــــــ	X, (, 6, 4,), , , , , , , , , , , , , , , , ,		ennek lakula kanak ki ki ki ki akula lakula kana		I 1
0.0.0.7.0.0	W	ØRKING 5	T 9 R A G E S	E.C.T.1.0.N.	<u> </u>			I . I.
0.0.0.7.1.0		E.R.M.S.G.L.L.	01					1
0 0.0,7,2,0			0.2	X.(.3.1.)	ME, CARD, M	I S S I N G F OR P	6 Y - N.O.	1 . 1.
0.0 0.7.3.0	٤.	1						TTT
0, 0, 0, 7,4,0	1	ERM.S.G.I.A.	0.2	9.(.5.)				T . I.
0,0,0,7,5,0	J		0.2	IX.X	· · · · · · · · · · · · · · · · · · ·			
0.0.0.7.6.0		E.R.M.S.G.Z.						T-T
0, 1, 1, 0, 0, 0	i	I	0.2	X.(.2.2.)	" A L U N M A T C H C D T			
0,0,0,7,80	1	ERMG2A	0,2	9 (, 5 ,)				1 .
0, 9, 7, 9, 0	L		0.2	IX.X.	" A.1. "			
010.0.8.0.0	L	m.s.G.3.	0.1	X (C.1.4.)	" ALEND, OF JOBA			
0.0.0.8.1.0		T.E.M.P.4.	0.1 R	9,9,9,9,9,				+ ++
0.0,0,8,2,0	I	CH.E.C.K.	6.1		* * * * * * * * * * * * * * * * * * *			┨┈┤
0.0.0,8,3,0	T		0,2	P (15.)				╂╌┵╂╼┷╌┙┄
0,0,0,8,4,0	1	CPAY~NO	0.2	9 (.5 .)				
0,0,0,8,5,0	1		0,2	9: (.6.)				
0.0.0.8.6.0	1	CISTINIT	0.2	9.				
0.0.0.8.7.0			02			بالاراف والمتطلب والمتاب المتطلب المتاب		
000880		CZNDINIT	0.2	(A) A	L L L L L L L L L L L L L L L L L L L	and the state of t		
0,0,0,8,9,0		, C 2 . N . D 1 1 V 1 (0.2	A I I I I I I I I I I I I I I I I I I I	and the second of the second o	 4. (4) (4) (1) (1) (1) 	10 miles 10	. J.
4 ~ 1 ~ 1 ~ 0 ~ 1 ~ 0		4 4	u.s	4P P		and the second of the second o		. ▼

GE-425/435 PROGRAMMING FORM

				PROGRAMMER	DATE	PAGE	,
PROGRAM	PANO	OLL SAMPLE	PROBLEM		12-6-63	3	4
	# M Y K	REFERENCE SYMBOL	DPERATION	OPERATION PARAMETERS			i
	ř	REFERENCE 3, MODE					IDENT.
SEQUENCE	E			PICTURE 40 41 45	VALUE		
1	0 7 0	DATA NAME 10	17 18 19 20 N 22 E 2	20		7.6	P.A.Y.
0.0.0.9.0	.0	C.L.A.S.T.N.M.	0.2	A.(.1,6,),			F-14-13
0,0,0,9,1	0		0.2	Α.Β.Β			
0,0,0,9,2		C.M.Ø.	0.2	9,9, , , , , , , , , , , , , , , , , ,			
0.0.0.9.3			0.2	X			
0,0,0,9,4		C.D.A.	0.2	9.9	* * * * * * * * * * * * * * * * * * *		
0,0,0,9,5	0		0.2	X	<u> </u>		
0,0,0,9,6		C.Y.R.	0.2	9.9	<u> </u>		
0.0.0.9.7			0.2	$A_1(.5.)$	NA''		
0.0.0.9.8		C.N.E.T	0,2	×,×,×, ·, 9, 9 , , , , , , , , , , , , , , ,			
0,0,0,9,9		1.	0.2	Χ.Χ			 -
0.0.1.0.0		$s_{\omega_{i,1}}$	0.1				
0,0,1,0,1		X.8.8	0.1	<u> </u>			
0.0.1.0.2		T,E,M,P,I	0.1 R	9,9,9,9,0,9,9,			
		I		<u></u>	i h h d d d d d d d d d d d d d d d d d 		
0.0,1.0.3	10 P	R. O.C. E.D. U.R.E	ADJVISI &	N			
0,0,1,0,4	. 0	LSTART	0 P.E.N		, P.A , Y, C, H, E, C , K, , , , , , , , , , , , , , , , ,		- +
0.0.1.0.5	ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ ـ	RIEIA, DIM.	IR E A DI	P.A.Y.M.S.T.R.			
0.011.0.6		R.E.A.D.T.C.	I F R E A D	S.W.I. D.N. G. T. F. T.E. S.T.	<u> </u>		
0.0.1.0.7	ــــــــــــــــــــــــــــــــــــــ	RDIC	READ	T.I. M.E. C.D.S.			1
0.0.1.0.8		T E S T	COMPARE	P.A.Y.~, N. Ø; T. C. P.A.Y.~, N. Ø; ; , T. E.S. T. Z.; E.E.	R.R.Ø.R.L.		tt
0,0,1,0,9	1.0		MAVE	TICIPAY ~ N. O. ERMS G. 2.A			1
0.0.1.1.0	0.10		T Y,P E	BUT PUT, ALPHIA, ERMS.G.2			1
0,0,1,1,1	ا ا		GOTØ	R.D.T.C.			1
9.9.1.1.2	ا ما	ERR ØRI	M & V E	P.A.Y.~N.Ø.; E.R.M.S.G.I.A.			1 11
0,0,1,1,3	اها		TYPE	12.12.1.11.12.1.11.11.11.11.11.11.11.11.			1
0.0.1.1.4	1.0		S.E.T.S.W	S1W11 191N1 11 11 11 11 11 11 11 11 11 11 11 11			1
5,0,1,1,5	ـــا ۵۰۵		G Ø T Ø	RIEIA DIMILLA INTERNATIONALIA			1-1
0.011116	الما	T.E.S.T.2	S E T S W	S.W.I. P.F.F.			1 1 1 1 1 1 1
تا المان		\$ - 4 -4 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 	I E	P.A.Y.~.N.O. E.O. 19.9.9.9.) E.O.J.			1
0.0.1.1.8			War E	T.C.D.A.T.E. J.D.A.T.E.			
وبالديمية	ـــا ا ميا		MP.Y.R				
0.0.1.2.0			MPYR		. <u> </u>		
0,0,1,2,1			SUBT	T.E.M.P.A.; G.R. Ø.S.S.; T.E.M.P L.			1 1
0.0.1.2.2			MPYR				
0,0,1,2,3	.0		SUBT	T.A.X.; G.R. Ø.S.S.; T.E.M.P. 1			
4. حـــا . فـــفـــ			MEM	B.E.T.A.			
0.0.1.2.5			, ØP	ALPHIA			
0,0,1,2,1			M F I				
דיריסיס		 	, ØP	XIRI8			
0.0.1.2.1		L.ø.ø.P.	IF	DICIONDE, 4, XIRISI ZIERIA GOTALEXI	<u> </u>		
0.0.1.2			SUBT	D.A.M.D.U.N.T., A., X.R.81; T.E.M.P.I.			
0+0.1.3.0			AIM	200			1
0.0.1.3.1			ØP.	X.R.8			
0.0.1.3.7			BRC	L. Ø. Ø. P.			1 7
10.0113.3		1	ØP	A. L.P.H.A.			

12/6/63

GE-425/435 PROGRAMMING FORM

PROGRAM				Р	ROGRAMMER			DATE	PAGE	OF.
	PAYF	ROLL SAMPLE	PROBLEM					12-6-63	4	4
	Ţ	REFERENCE SYMBOL	OPERATION			OPERATION PARAMETERS			(1
SEQUENCE			 	-						- IDEN
	- 1	DAT'A NAME	6 17 18 19 20 C 22 E	· I	PICTURE	40 41 44 45	v	ALUE		I I DEN
1	6 7 8	1 9 1				40 41 48				5 7.7
4.5.4 به به		E.X.J.T.	MØVE				 			P.A.Y
جـدنـيميه	.0		A D D T A D D T	GRASS	Y.T.D.G.R. Ø.S.S.		 			14.
0.0.1.3.6	-0		A D D T	LAXYA	T.D.T.A.X.					
0.0.1.3.7	-0		MONE	$PAY \sim N$	Ø ; C. P. A.Y .~ N. Ø.					
ه د د و و	.0		M &.V E		.f.f., C. 1.3.T.I.N.		- - - - - - - - - - 			
0.0.1.3.5	-01		M Ø N E		I.T. J. C. 2.N. D.I.N.					
0.0.1.4.0			M GOVE	LASTN	AMÉGCLAST	N.M	 		i	
0.0.1.4.1			M O V E	M. Ø. ; . CIM	· Ø · · · · · · · · · · · · · · · · ·					
0.0.1.4.2			MONE	صعرفاه						
0.0.1.4.3			MOVE	YRIGIY						
0.0.1.4.4			MØVE		Y ; CNET.					
حامد ما			WRITE	P A Y CH	E.E.K. FROM	CIH E CIK	 		·	
0.0.1.4.6			WRITEX			PIPY MISTR	1			1.1.
0.0.1.4.7	-0		GØTØ	RIE A.D.M			I +		 	
0.0.1.4.8		END I	Ma v E	9.9.9.9.9	P.A.Y.~ N.Ø.		 			
0.0.1.4.9			G ø.T ø	RIE AID T	211111111111111111111111111111111111111					$\mathbf{L}\mathbf{L}$
0.0115.0		END 2	MOVE	9.9.9.9	; TCPAY~NØ			1-	1 1 1 1 1	Т.Т.
0,0,1,5,1			GOTO	T.E.S.T.						$\perp \perp$
0.0.1.5.2	.0	EOJ	TY PE	O.G. T.U.B	$T \rightarrow A \perp P + A \rightarrow M$	5.6-3				\perp
0.0.15.3	0	1	CLOSE	PAYMS	TI, AL PHA, M	SOPFYMST	R. DAYCHECK			I . I .
0.0.1.5.4	.0.	1	103				<u> </u>			$\Gamma \Pi$
0,0,1,5,5	10	ALPHA	B C T R	5						1 .1.
0,0,1,5,6		BETA PRERROR	BCTR	5. 9.9.9.9.						I I I
0.0.1.5.7		PRERROR	HALT	9,9,9,9,			 			
		<u> </u>	Q.N.3	START						1 ×
										T
		- L			<u> </u>		 			
		1	1-	1	1 1 1 1 1 1 1 1 1 1 		 			L
		<u> </u>	+				11.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.			L.,
				1 1 1 1 1 1	 					1
		<u> </u>			 					
		1		1						
										
										Τ
		_	╂ ╌┼╌┼┈┨╶┟┈╁┈╁				1 1 1 1 1 1 1 1 1 1 			
	-								 	T
									· · · · · · · · · · · · · · · · · · ·	\mathbf{H}

ALPHABETIC TABLE OF MACRO INSTRUCTIONS

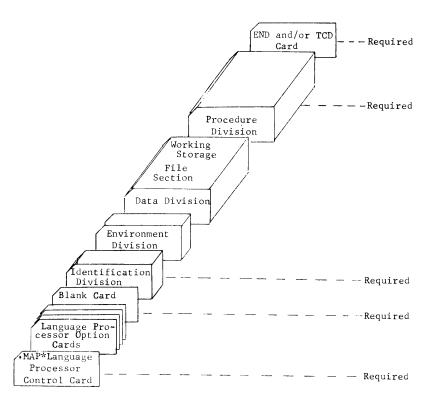
ABORT	2-90
ADDR Add and Round	2-91
ADDT Add and Truncate	2-92
CLOSE	2-73
COMPARE	2-83
DIVR Divide and Round	2-96
DIVT Divide and Truncate	2-97
EOJ	2-89
Extended Input/Output System Service Macro Instructions	2-74
GOTO	2-87
GOTO DEPENDING ON	2-88
HALT	2-88
F	2-85
LOAD	2-80
MOVE	2-77
MPYR Multiply and Round	2-94
MPYT Multiply and Truncate	2-95
OPEN	2-72
READ	2-68
SETSW	2-86
SUBR Subtract and Round	2-92

SUBT Subtract and Truncate	2-9
TYPE	2-7
UNLOAD	2-8
WRITE	2-6
WRITEX	2-6

ASSEMBLY FUNCTIONS

SOURCE PROGRAM

To prepare any Macro Assembly Program source language program for an assembly, the source deck must be organized in the following sequence:



*MAP - used here as an abbreviation for Macro Assembly Program

The Environment and Data Divisions are not required if the source program contains only basic assembly language instructions.

Language Processor Control Cards

A language processor control card causes the Program Monitor to search the systems tape, load, and transfer to the beginning of the Macro Assembly Program.

Following this control card in the source deck will be the language processor option cards. These cards, if present, specify deviations from the standard assembly procedure. They may be used to specify:

- the source language deck is to be read from tape
- the object program decks are to be written on tape, or both cards and tape
- a sequence check on the source language deck is to be performed
- the coding generated for the macro-instructions should not be printed on the assembly listing.

The standard mode of assembly and the format of the language processor (LP) option cards will be specified at a later date.

A blank card must follow the last LP option card.

ASSEMBLER PROGRAM ORGANIZATION

The Macro Assembly Program is a multi-pass macro assembler which processes a source program from punched card or magnetic tape input. The nature of the instructions which appear in the source program determines the number of passes required to produce an object program. If the source program contains macro-instructions, it is processed through the Translator, Selector, and Assembler phases of the Macro Assembly Program. If the source program does not contain macro-instructions, processing proceeds directly to the Assembler phase, bypassing phases of the Macro Assembly Program that are not essential to the assembly. See Figure 3-1.

Translator

The Translator reads and processes the source program. The following functions are executed while processing the Data Division. The Translator --

- constructs a File Table for each file encountered in the File Section.
- constructs basic assembly language pseudo-operations to allocate memory to and/or define constants for the data names encountered in record descriptions
- constructs a Data Description Table which contains complete field characteristics of each data name encountered in record descriptions.

As the Translator processes the Procedure Division, a magnetic tape is written which contains the basic assembly language symbolic instructions, pseudo-operations, and the macro-calls. A macro-call identifies a macro-instruction and its parameters. When a parameter is a data name, the macro-call includes the characteristics of the data name from the Data Description Table.

Selector

The Selector controls the generation of basic assembly language symbolic instructions, by linking the macro-calls with their required generators. The generated basic assembly language instructions are written to tape along with the input instructions. Memory available for generators is allocated by a method which takes into account frequence of use and nesting of generators so that a minimum number of passes of macro-calls is required.

Assembler

The basic assembly language instructions, both input and generated, are then assembled. The assembly listing and object program card deck are produced simultaneously.

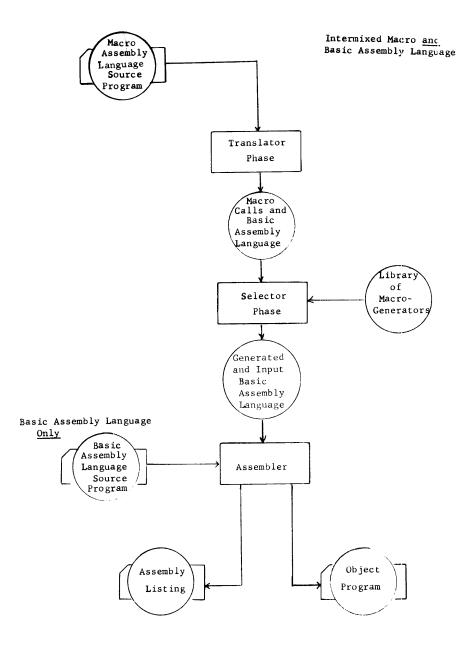


Figure 3-1. The GE-425/435 Macro Assembly Program

LOADING PROGRAM

Input to the loader consists of a relocatable binary deck for each segment to be loaded. These relocatable decks contain global symbols, relative addresses, relocation indicators, instructions, and requests for other segments. The loader's function is to combine these segments to form an operational program.

Card Types Produced by the Assembler

Seven card types are produced by the assembler and recognized and processed by the loader.

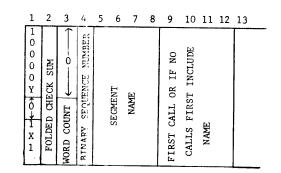
Header card
Origin card
Internal global card
Instruction card
External global card
TCD card
END card

Any segment must contain a header card and an END or TCD card. The following statements can be made for all card types.

- 1. All cards are binary and have a 7-9 punch in column 1.
- 2. Columns 74-80 are not examined by the loader.
- 3. An 8-punch in column 1 causes the checksum to be ignored. This is indicated by an X on card forms.
- 4. Checksum includes columns 1 through 73.
- 5. Card type is located in the 12-3 row of column 1.
- 6. Column 2 is reserved for a folded checksum.
- 7. Column 4 contains a segment sequence number which is checked by the loader.

The format for the seven types of cards produced by the assembler follows:

Header Card Format



68	69	70	71	72	73	74	75	76	77	78	79	80
	CALL	OR INCLUDE	NAME		# OF INCLUDE # OF CALLS	CARD	SEOUENCE	NUMBER	LLEF	I IDENTIFICATION		→

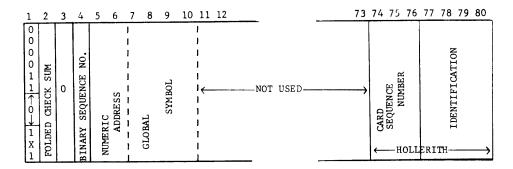
Notes

- 1. In column 1 if Y = 0 the segment is not forced to origin at 0 mod 4 if Y = 1 the segment is forced to origin at 0 mod 4.
- 2. Word count is the number of calls and includes on the card.
- 3. Column 73 contains the number of calls and includes for the segment.
- 4. If more than 1 header card is needed for a segment columns 5-8 and 73 are duplicated on the additional header cards.

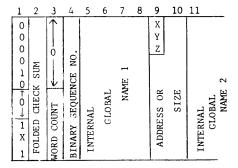
3-6

5. Card type 40 or 41.

Origin Card Format



Internal Global Card Format



70	71	72	73	74	75	76	77	78	79	80
		NOT USED		CARD	SEQUENCE	MOTIBEN		IDENTETCATION	SENT ICATION	
					—н	оцц	ERI	IH-		- >

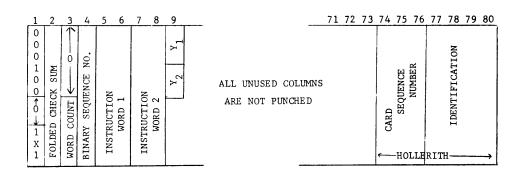
Notes

- Columns 5 and 6 contain an absolute address to which the segment may be origined.
- 2. If a global symbol is in columns 7-10, its value will be added to the value in columns 5 and 6 to form the origin.
- 3. The global must have been previously defined by the loader.
- 4. If 0 mod 4 was requested on the header, the origin is forced to 0 mod 4.
- 5. Card type 3.

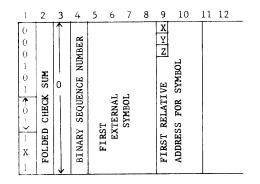
Notes

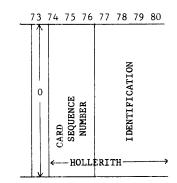
- Word count contains the number of internally defined globals on this card.
- 2. If X = Y = Z = 0, the symbol is equated to the absolute value in the associated address.
- 3. If X = Y = 0, Z = 1, the symbol is equated to the address formed by the addition of the address to the segment origin, i.e. the symbol is relocatable.
- 4. If X = 0, Y = 1, Z = 0, the block associated with the symbol is treated as a BSSL.
- 5. If X = 0, Y = Z = 1 the block associated with the symbol is treated as an ARPL.
- 6. If X = 1, Y = Z = 0, the block associated with the symbol is treated as an LSBL.
- 7. If X = Y = 1, Z = 0 the block associated with the symbol is treated as a BPSL.

Instruction Card Format



External Global Format





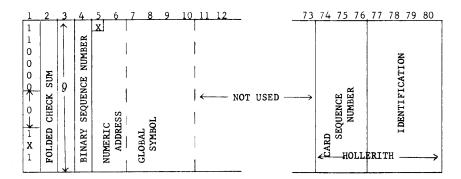
Notes

- 1. Word count is the number of instructions on the card. Maximum of 30 instructions per card.
- 2. Y is a 3-bit flag associated with each instruction of the form A. $^{22}_{+B}$. $^{21}_{+C}$. $^{20}_{-W}$ where A is presently 0.
- 3. Flag bits immediately follow the last instruction on the card.
- 4. If B = 1, the associated instruction is to be treated as an absolute increment for the loader P-counter.
- 5. If B = 0, the associated instruction is a true instruction.
- 6. If B = 0, C = 0 the associated instruction is absolute.
- 7. If B=0, C=1, relocate the address in the associated instruction by the segment origin.
- 8. Card type 4.

Notes

- A symbol must be contained in a card but reference address can be continued on the next card.
- 2. The relative address is relative to the segment origin.
- 3. If X in the relative address is 0, the value of the symbol will be added to the contents of the reference address. If X is 1 the value of the symbol will be subtracted from the contents of the reference address.
- 4. If Y is 1, an external symbol follows; if Y is 0 another relative address follows.
- 5. If Z is 1, this is the last item on this card; if Z is 0 at least one more field follows.
- 6. More than one symbol may refer to the same relative address.
- 7. Card type 5.

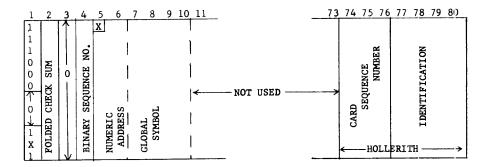
TCD Card Format



Note

- 1. Columns 5 and 6 contain an absolute address to which the loader may transfer control.
- 2. If a global symbol is in columns 7-10, its value will be added to the value in columns 5 and 6 to form the transfer address.
- 3. The global must have been previously defined by the loader.
- 4. If X = 1 in column 5, the numeric address in columns 5-6 will be added to the segment origin.
- 5. There cannot be a TCD to absolute zero.
- 6. Card type 60.

End Card Format



Notes

- 1. Columns 5 and 6 contain an absolute address to which the loader may transfer control.
- 2. If a global symbol is in columns 7-10, its value will be added to the value in columns 5 and 6 to form the transfer address.
- 3. The global must have been previously defined by the loader.
- 4. If X = 1 in column 5, the numeric address in columns 5-6 will be added to the segment origin to form the transfer address.
- 5. Card type 70.

Octal Patch Card

The loader accepts and operates on octal cards which are inserted in a segment before the segment TCD or END card. These cards are octal corrections to the segment and these corrections may be absolute or relocatable. A correction card is checked by the loader to insure that it is octal and that an error message is typed if any non-octal characters are found through column 20. Any hollerith information can exist after column 20.

The octal patch card has the following format:

	1	2	_3	- 4	_5	6_	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
	0	ĺ		oc t	а1	loc	ati	on		A	В										
				wh	ere	P	atc	h						oct	al	pa	tch	1			
ı				í	8	pla	ced						İ								

where A = 0 if columns 4-8 are an absolute address

= 1 if columns 4-8 are relocatable and the segment origin is to be added to form the location to be patched

B = 0 if columns 16-20 are an absolute address

= 1 if columns 16-20 are relocatable and the segment origin is to be added to form the desired address

All address calculations are modulo-15 bits.

Column 1 contains a 0.

Segment Deck Order

Each segment deck is sequenced in column 4. The cards within a segment must not be shifted from their punched order. The order for each format is given below with an asterisk indicating that the format must be present.

- * 1. Header card
 - 2. Origin card
 - 3. Internal global card
 - 4. Instruction card
 - 5. External global card
 - 6. Octal patches
- * 7. TCD or END card

For a more detailed discussion, see the GE-425/435 Loader write-up.

- + OUTPUTS OF THE ASSEMBLER
- ASSEMBLY SOURCE LANGUAGE ERRORS AND REMEDIES

Progress Is Our Most Important Product



COMPUTER DEPARTMENT . PHOENIX, ARIZONA