CONTROL DATA

160G

GASS
PRELIMINARY
MANUAL

CONTROL DATA
160G

GASS
PRELIMINARY
MANUAL

The information contained in this
manual is preliminary and subject
to change without notice.

CONTENTS

APPENDIXES

FIGURES

CHAPTER 1

DESCRIPTION OF GASS

An assembly program has two major functions. The first is to allow the programmer to use meaningful names for the instructions; thus, an add instruction could be written as ADD rather than octal 20. The assembly program thus translates the mnemonic operation codes to the actual machine codes required. The second function is to allow the programmer to assign lables or names to locations which are to be referenced in the memory. These lables or names are called symbols. When a location is labled by a symbol, the programmer then may refer to that location by use of the symbol without concern about the actual location in the program. The assembly program takes care of the assignment of symbols to locations and the resultant translation by the construction of the symbol table which gives the absolute location and the symbol to which it has been assigned.

Using the assembly program, a programmer may write, "ADD L2". The assembly program noting the assignment of location L2 translates the above statement to the actual machine coding to perform the function. GASS provides these basic functions of an assembly program and other capabilities as defined in this manual.

DESCRIPTION

GASS accepts symbolic instructions for the 160G computer as an input. These instructions which are prepared on punched cards may be transferred to magnetic tape as an alternate input to the program. The assembly process requires two passes (readings) of the source language information. During the first pass, GASS reads each line (card) of symbolic information and, optionally, writes this information on an intermediate magnetic tape for the second pass. The information is scanned, and a partial translation of the operation code is made to determine the space which will be required in the memory of the 160G to contain the instructions and

data. This information is used to assign values to symbols which appear in the program. The symbols then are inserted into the internal and external symbol tables for use in the second pass of the assembly process.

During the second pass, GASS analyzes the input information (derived either from the input media or the intermediate magnetic tape) and converts this information to machine form. It transmits each line of output and groupings of assembled binary words to the output devices. At the completion of the assembly process, it also provides the linkage information for inclusion in the binary program output. If the information compiled in the current internal symbol table will be needed during the assembly of other programs, it may be punched on a separate, special card deck at the completion of the second pass.

Normally, GASS prepares the internal symbol table based on the input symbolic coding; however, the symbol table from a previous assembly may be loaded prior to assembly of a program. This special symbol table information must have been prepared by GASS as an output from a previous assembly.

The following output information results from the assembly process:

1. An output listing of the assembled program
2. Relocatable binary card output for subsequent loading and execution of the assembled program
3. Relocatable binary card images on magnetic tape
4. A symbol table for use in combining other programs during a separate assembly

SPECIAL FEATURES INCLUDED IN GASS

Gass includes several special features which are described in this chapter. Additional features are covered in other chapters of the manual.

SUBPROGRAM AND SUBROUTINE LINKAGES

At running time, a program is composed of one or more independently assembled subprograms. Each subprogram is composed of one or more subroutines which are assembled at the same time.

For the object program to operate correctly, all references in the various subprograms and subroutines must refer to the actual machine locations as they are assigned at run time. The following methods are used to specify and complete these linkages provided in MASS and GASS:

1. Several parts of one program may be assembled at different times by the use of the internal symbol from the previous assemblies. To do this, the first part of the program is assembled and the internal symbol table is saved. For the assembly of the second part, the symbol table from the first part is included in the input information. In this manner, all references from the second part to the first part may be made symbolically. This method is limited by the fact that there is no easy way to make a symbolic reference from the first to the second part; however, a possible method is to use the EQU statement in the first part.

2. By use of the pseudo instructions LOCAL and NONLC, the location symbols appearing in one subroutine may be declared local to the current coding until the appearance of another LOCAL or NONLC pseudo instruction. Cross references from other subroutines to be assembled at the same time are indicated in the address field of the LOCAL pseudo instruction. This technique allows the programmer to re-use symbols in various subroutines without trouble from multiple definition. Only the cross reference symbols will appear in the main symbol table. This technique also allows the use of more symbols in the program than the capacity in the symbol table. (See the description of LOCAL and NONLC pseudo instructions.)

3. Subprograms assembled separately or contained on the library tape, communicate with each other by the use of entry points and external symbols. (See ENTRY and EXTNL pseudo instructions.) If,in subprogram 1 there is a desire to make reference to the location represented by the symbol ABLE which occurs in subprogram 2, ABLE must be declared as an external symbol in Subprogram 1 and as an entry point in subprogram 2. The linkage, or insertion of the true machine location, is made by MASS at load time. Only a limited number of linkages can be made by MASS due to the limited storage available in the basic 160G system. A program is available to prelink the subprograms prior to run time.

SUBPROGRAM FORM

The first card of each subprogram must contain the IDENT pseudo instruction, and the last card must contain the END pseudo instruction.

Any subprogram may be the main subprogram -- the one to which initial control will be given on completion of the loading process. One, and only one subprogram must have an ENDT card which gives the name of the entry point in the main subprogram. When the binary deck of the assembled program has been loaded by the linking binary loader, a bank return jump is made to this entry point. If no entry point has been specified, the loader terminates the job. When the program is complete, there should be a jump to the entry point location. This jump returns control to the operating system and terminates the job.

DATA STORAGE

Data may be stored in the same area as the program and thus be relocatable with the program or non-local to the subprogram. Non-local data storage must be declared in COMN statements (see COMN pseudo instruction). Blocks of common storage declared in a subprogram may be in different banks of memory with the restriction that any one block of common storage must be wholly contained in one memory bank. A block of common storage shared by two or more subprograms or by more than two computers, must be declared in each of the subprograms which refer to that block. Further rules regarding the use of common storage are given under the COMN pseudo instruction.

PROGRAM RELOCATION

All programs assembled by GASS are relocatable (if certain address field conventions are followed) by the specification of a relocation constant at load time. This constant may be originated by the programmer under self-contained operation or by MASS under the monitor form of operation.

The relocation process depends on the contents of the word and on whether a word is relocatable. The following definitions will be used.

1. A relocatable word is a word of program or data which was assembled under control of the ORG or PRG counter.
2. A nonrelocatable word is a word which was assembled under control of the CON counter or which is defined in a COMN pseudo instruction.
3. A word which may be modified that specifies a reference to the 13-bit address of a relocatable word.

Under the above definitions, the relocation process consists of storing all nonrelocatable words at the location specified as the result of the GASS assembly. All relocatable words will be stored at the location specified in the assembly, plus the relocation constant. All words which make a 13-bit reference to a relocatable word will have the relocation constant added to the contents of the word thus making a reference to the true location of the relocated word.

The following words are not modified by the relocation constant:

1. One word instructions (for example, load direct, load relative, etc. instructions)
2. Words which consist of a numeric address field
3. Words that refer to addresses of words assembled under control of the CON counter or specified in COMN
4. Words which specify the difference of the addresses of two relocatable words

OPERATOR OPTIONS FOR GASS

By positioning jump switches on the 160G console, the operator may do any of the following during GASS assembly:

1. Suppress binary output
2. Load a special symbol table prior to assembly
3. Obtain the symbol table as output of the assembly

Further options, including the assembly of several jobs at one time, may be performed. See the GASS operating instructions for these options.

CHAPTER 2

INSTRUCTION FORMAT

Input to GASS consists of symbolic instructions for the 160G written in the form of characters punched on 80-column cards or images of these cards on magnetic tape. The cards are punched from coding lines written by a programmer on the 160G assembly system coding form. The format of the input as written on the coding form is expressed in terms of the colums of a card in which the information will be punched. The coding form is illustrated in figure 2-1.

Figure 2-1. 160G Assembly System Coding Form

FIELDS

A coding line is divided into four major fields: location field, L; operation field, O; address field, A; and comments field, C. The location field covers columns 1 through 8. Column 9 is not used. The operation field begins in column 10 and ends at the first blank column. The address field may start anywhere to the right of the blank column terminating the operation field, but must begin no later than column 20. Column 20 (the first dashed line on the coding form) is a convenient justification point for the address field. The address field terminates at the first blank column or at column 72. The remaining columns after the blank column following the address field are treated as comments. Column 41 (second dashed line) is suggested for justifying the comments if the address field ends before column 40. Columns 73 to 80 may be used only for comments or identification.

LOCATION FIELD

The location field (called an L-term, label, symbol, or identifier) may contain a symbol or 8 blanks. A symbol consists of 1 to 8 non-blank characters, with at least one character being alphabetic. Leading, imbedded, and trailing blanks will be ignored, and the symbol will be packed and left justified in the field by GASS. The following characters may __not__ be used in constructing a symbol:

$$+ - * / , = ( ) \$$$

All other characters from the 64-character set may be used.

OPERATION FIELD

The operation field may consist of the following:

1. One of the mnemonic operation codes listed in Appendix B, followed by a blank column or a comma
2. One of the pseudo operations listed in Appendix A
3. The name of a macro-instruction
4. One of the preceding, a comma, and a modifier expression

A blank in column 10 indicates that the word to be assembled does not contain an operation field. In this case, the address field may start in column 11.

2-2

Modifier Expression

A modifier expression is contained in the operation field and is separated from the operation code by a comma. The modifier may take on any form which is described under address field. A restriction on the modifier expression is that it must be of the correct magnitude to be acceptable to GASS.

ADDRESS FIELD

The address field is evaluated to get either the address of an operand in memory or a constant which will enter into the calculations in the 160G. The address field may be any of the following:

1. Blank. In this case, it will be evaluated as zero.
2. A decimal number less than 2^{17}. A number will be interpreted as decimal unless it is suffixed with a B.
3. An octal number no greater than 377777_8. A number to be interpreted as octal must be suffixed with the character B.
4. A symbol
5. A symbol prefixed with the character $. This symbol will be interpreted as "the number of the bank which contains the location specified by the symbol". For example, The symbol ABLE is assigned to location 0077 in bank 5. $ABLE will be interpreted as the number 0005.
6. An array element expressed as A. For example, A(i,j,k) where A is an array name defined in COMN and i,j, and k are element numbers in the array. The element numbers must be numeric.
7. A special symbol (*). The value of * is the current value of the location counter. This value specifies the address at which the operation code portion of the current instruction is being assembled.
8. A special symbol (**). The value of ** is always the octal value 17777. This symbol may not appear in an arithmetic expression. It is normally used in a field which is going to be preset.
9. An arithmetic expression combining any combination of the preceding items 2 through 7.

Arithmetic Expression

An arithmetic expression may combine symbols and constants using the four operations of addition (+), subtraction (-), multiplication (*), and division (/). The expression is evaluated from left to right, performing all multiplications and divisions and then all additions and subtractions. The use of parentheses for grouping is not permitted.

For example, the expression:

$$15^*a + 5/2^* \ C-5$$

is evaluated as

$$(15 \cdot A) + ((\frac{5}{2}) \cdot C) -5$$

The following rules apply to the evaluation of an arithmetic expression:

1. The modulus of the arithmetic is $2^{13}-1$.
2. In a divide operation, only the integer portion of the quotient is retained.
3. Symbols which appear in an arithmetic expression must be defined, thus external symbols must not appear in an arithmetic expression.

COMMENTS FIELD

The comments field begins with the first column after the blank column which terminates the address field and ends with column 80. The comments field is ignored by the assembler, but it is printed on the output listing.

OSAS OPTION OF OPERATION

If a line of coding is indicated to be in the OSAS format by the previous appearance of the OSAS pseudo instruction, the format and general rules of operation are as given in Control Data Publication 507a "OSAS-A 160-A Assembly System".

CHAPTER 3

RULES OF OPERATION

During the first pass of an assembly process, GASS forms the symbol table which gives a symbol and its numeric equivalent. The numeric equivalent is determined by where the symbol appears in the location column of a coding form or in the COMN statement. The numeric equivalent is stored in the symbol table as a 17-bit number of which the upper 5 bits represent a bank number (4096-word bank) and the lower 12 bits represent the location within the bank. In the G mode of operation, the programmer may consider this number to be a 5-bit bank number, where the banks are numbered 0,2,4,6, etc., and a 13-bit address within a 8192-word bank.

The numeric equivalent is formed in three ways:

1. It is obtained from an EQU pseudo instruction. In this case, the number contained in the location term is converted to a 17-bit number and placed in the symbol table, or the expression in the location term is evaluated and placed in the symbol table.

2. The equivalent is implied by the current value of the program location counter. In this case, the value of the counter is placed as the numeric equivalent, including the bank number.

3. The equivalent is determined by the COMN statement.

The symbol table also classifies a symbol as to whether it may be relocatable or non-relocatable. A relocatable symbol is any symbol which has its numeric equivalent assigned from a setting of the ORG-PRG program location counter or which is defined by an equivalence statement to a relocatable symbol. A non-relocatable symbol is any symbol which has its numeric equivalent assigned from a setting of the CON program location counter or from an EQU statement which has a numeric location term only. A relocatable symbol also may be defined by an equivalence statement to a relocatable symbol plus or minus a constant.

The value of the address term is also flagged on the binary card output as being relocatable or non-relocatable to permit the relocating binary loader routines to position the program correctly. Generally, an address term will be flagged as non-relocatable unless it consists

of a single relocatable symbol or a single relocatable symbol plus or minus a constant. Any other form of relocatable symbols appearing in an address term will be evaluated. Generally, the resulting program will operate correctly only at the position it was assembled, and the program itself will not be relocatable.

In constructing the final machine instruction from the symbolic statement, the evaluated address term, and also modifier terms, must be reduced to 3, 6, 13, or 18 bits. The following rules of operation are followed by GASS in producing the address portion of the instruction or the instruction.

1. A line that contains information only in the remarks field will appear on the listing, but it will not reserve space in the binary object program.

2. If the operation field is blank (column 10 is a blank code), the address field will be evaluated if it is an arithmetic expression. A 13-bit number will be produced and stored. The operands will all consist of the lower-order 13 bits of the numeric equivalent without regard to the bank number assigned in the numeric equivalent.

3. If the operation field specifies no address, direct, or indirect addressing (N,D,I), the address field is evaluated to form a 13-bit address. If this value is greater than 77_8, the line is flagged as a range error (R) on the listing, and the lower 6 bits of the instruction are set to zero. If the value is less than or equal to 77_8, it is used to form the one-word instruction.

4. If the operation field specifies 6-bit relative addressing (F,B,R) and a numeric address field is given, this address is inserted in the 6-bit E portion of the instruction, if it is the right size. If the address field is symbolic, it is evaluated, and the contents of the location counter is subtracted from the evaluated address field to determine the relative increment. For F type operation codes, a positive result is directly inserted in the lower-order 6 bits of the instruction. If the result is negative or the number is greater than 77_8, the lower 6 bits of the instruction are taken as zero, and the line is flagged as a possible range error in t he listable output. For B-type operation codes, a negative result is complemented, and the process is as for forward operation.

The possibility of range errors may be reduced by substituting an R-type code for the F- or B-type operation codes. In this case, GASS determines the correct relative machine operation code and selects the correct operation if the address specified is within range. (JPR and ERR, which are not considered R-type operation codes, will not be recognized as such by GASS.)

5.   If the operation code specifies a 13-bit address portion (M,C,IB,MX), the evaluated address field will be stored as the 13-bit G portion of the instruction.

6.   If the operation code specifies relative entire bank (RB), the location of the operation code F portion of the instruction will be subtracted from the evaluated address field and stored as a 13-bit number.

7.   If the operation code specifies entire memory (no modifier), the 17-bit value of the evaluated address field is stored as 18 bits in the instruction. The 5-bit bank designator is stored as the lower 5 bits of the E portion of the instruction. The 12 bits of the bank location plus the low-order bit of the bank designator are stored as the 13-bit G portion of the instruction, as shown in figure 3-1.
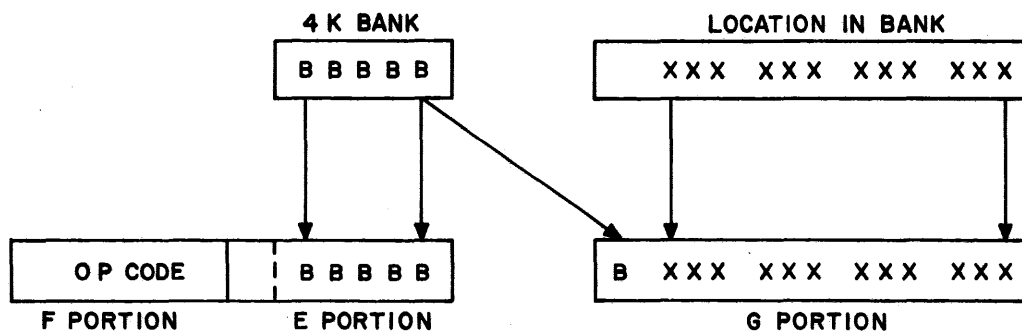

Figure 3-1.   Entire Memory Instruction

CHAPTER 4

PSEUDO INSTRUCTIONS

Pseudo instructions are non-machine language instructions which are used in preparing a subprogram for GASS assembly, a program for inclusion on the library tape, and the output of the GASS assembly for operation under the MASS monitor system. Some of the pseudo instructions provide required information to GASS, others provide information which is passed on to MASS for operation of the program, and still others are programmer aids which provide a more convenient means for defining portions of a program. The pseudo instructions are grouped according to functions. Following several of the pseudo instructions are comments on the function and programmer use of the pseudo instructions.

Samples of the way that the pseudo instructions are written on the GASS coding form are included with the description of each pseudo instruction. The pseudo instructions have certain specifications on the symbols which may appear in the location field and the address field. The following conventions have been followed in preparing the samples:

1. If a symbol <u>must</u> appear in a field, the symbol REQUIRED, REQ, or some variation is written in the field.
2. If a symbol <u>may</u> occur, but is not necessary, the symbol OPTIONAL, OPNL, OP, or some variation is written in the field.
3. If a symbol <u>must not</u> appear in a field, the field is left blank.
4. Other self explanatory information may appear in the address and comments field.
5. All examples are shown with the address field starting in column 20 of the coding form. This is not necessary; the address field may start following the first blank column following the operation field.

SUBPROGRAM LINKAGE

These pseudo instructions define the name, beginning, and end of a subprogram. They define the logical input/output units which are used by the subprogram if the subprogram is to be operated under MASS. They also define the subprograms which will be called

during the execution of the subprogram under MASS control. In-structions are also provided to incorporate other subroutines during the assembly of a given subprogram.

IDENT

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	IDENT REQUIRED	

Description

IDENT causes the symbol contained in the address field to be established as an identifier lable for the assembled subprogram. This control operation must be physically located prior to any instruction, data definition, insertion, or storage allocation statement in a subprogram. REM pseudo operations may precede the IDENT card for a program which is not to be included on the library tape in a symbolic form.

Comments

IDENT must appear as the first meaningful card of each subprogram; otherwise, "NO IDENT CARD" will appear as the first line on the output listing. If an IDENT occurs anywhere but the first line of a subprogram, it will be flagged on the output listing with the error code I. An L term is meaningless and will be ignored. The address field must contain a symbol. If the subprogram is to be operated under MASS and makes use of the MASS standard input/output routines, the standard unit designation of the I/O units used in the subprogram must also appear on the IDENT card as shown in the following example.

Example

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	IDENT REQUIRED, 1, 11, 3, 25	

System use of IDENT

The identifier lable provided in the address field of the IDENT card has the following uses throughout the 160G programming systems:

4-2

1. The subprogram may be included on the system library tape in a symbolic form for inclusion with other assemblies of subprograms. In this case, the program may be called a subroutine. For this use, the IDENT card must be the first card of the subroutine deck. The program identifier lable from the address field is used by GASS in searching for the subroutine from the library tape. This calling of a subroutine from the library tape is provided by the LIBA pseudo instruction.

2. The subprogram may be assembled to a relocatable binary form and included on the MASS library tape. In this case, IDENT provides the identifier for locating the program on the library tape and information on the length of the program and the standard I/O units which are required.

3. The subprogram may be combined with other subprograms at object program load time. The other subprograms may be loaded from the same media as the original subprogram, or they may be called from the library tape. The effect of the IDENT card is to cause an IDC (Identification Card) to be produced as the first card of a binary program deck. The format will be that prescribed by the MASS relocatable linking loader. The symbol from the address field will appear on the IDC card as the name of the subprogram. Also appearing on this IDC card will be the length of the subprogram and the standard input/output units which may be called on by the subprogram.

The identifier lable specified in the address field of the IDENT card is unique and may or may not be the same as any location lable defined in the subprogram or any other subprogram. This lable will be printed out (on option) at object program load time, to tell what subprograms are being used. The lable is also used in the address field of the LIBA and LIBS pseudo operations to specify what subprograms are to be included in the final operating program.

END

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	END	OPTIONAL	

`1 |2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10 |11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |20 |21 |22 |23 |24 |25 |26 |27 |28 |29 |30 |31 |32 |33 |34 |35 |36 |37 |38 |39 |40 |41 |42 |43 |44 |45 |46 |47 |48 |49 |50`

Description

END marks the end of the current subprogram to the assembly pro-
cess. It causes the assembly operation to prepare for the next pass
or to terminate the program assembly process. Since END marks
the end of the subprogram, it causes a binary transfer card (TRA
card) to be produced as the last card in the relocatable binary deck.
If a symbol appears in the address field, the TRA card will contain
the relocatable transfer address within the subprogram. If there is
no symbol, there will not be a transfer address in the TRA card.
A location term, if present, will be ignored.

Comments

END must always be the last card of a subprogram.

System use of END

END is used by the assembler to mark the end of an assembly pro-
gram and to produce a binary transfer card on the relocatable output.
The binary transfer card is used by the relocatable loader routine to
identify the end of a binary deck. When the relocatable loader is used
independently of the MASS system, the transfer address given in the
TRA card is set up as the starting point of the program. If there is
no transfer address, the loader will stop and leave the starting of the
program up to the operator.

Within MASS, the transfer address given in the TRA card is ignored
during the loading, and the MASS linking relocatable loader depends
on the symbolic transfer card to designate the starting point in an ob-
ject program. For the MASS loader, two successive binary trans-
fer cards must appear to signal the end of the loading process.

ENDT

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
¹₁²₁³₁⁴₁⁵₁⁶₁⁷₁⁸	⁹	E N D T ¹R E Q U I R E D	¹

Description

ENDT causes a symbolic transfer card (STRA) to be produced in the relocatable binary deck. Normally, the ENDT is the card preceding the END card of a subprogram. The address field must contain a symbolic transfer address. This address must appear in an ENTRY statement in the current subprogram or in one of the subprograms which are loaded together with the current subprogram in obtaining the final object program. The symbolic transfer card will be in the format specified by the linking relocatable binary loader used in MASS. A location term will be ignored.

Comments

The ENDT pseudo command is used to specify a transfer of machine control to the MASS linking relocatable loader. On completion of the loading of all programs provided and called for at the running time, a transfer is made to the symbolic location specified. The transfer address must be specified as an ENTRY point to one of the subprograms which are loaded since the transfer is made by an examination of the entry point table at load time.

WAI

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
¹₁²₁³₁⁴₁⁵₁⁶₁⁷₁⁸	⁹	W A I C O M M E N T S T O O P E R A T O R	¹

Description

WAI causes the program assembly process to stop. The contents of theWAI statement starting at column 10 will be typed on the typewriter to provide instructions to the operator. Typing a space and then a carriage return causes the assembly process to continue. Typing a carriage return simulates the END pseudo operation.

Comments

An END or WAI pseudo operation must be the last card of the program to be assembled. If a WAI is the last instruction, the END pseudo must be generated by the operator as given in the preceding description in order to complete the assembly process.

ENTRY

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	ENTRY	IREQ, ØPTIONL2	I

Description

ENTRY causes the symbols appearing in the address field to be placed in an Entry Point Name Table. These symbols must be defined within the subprogram. If a symbol has not been defined, the error flag U will appear next to the ENTRY pseudo operation in the program listing. A location term will be ignored. The address field is of the form SYMBOL1, SYMBOL2, etc., with the field terminated by the first blank column. ENTRY pseudo operations may appear any place in a subprogram.

Comments

The ENTRY pseudo operation is used to define locations in a subprogram which will be referenced by other subprograms.

System use of ENTRY

The entry point name table is punched out as a part of the binary program deck. The information in the table is combined with information on the entry point name table obtained from other subprograms that are loaded at the same time. This allows the linking relocatable loader to provide the linkage which permits all subprograms to work together. The information in the entry point name table is carried as the symbolic name of a location and the relocatable binary location which corresponds to the symbolic name in the subprogram. At load time, the symbols specified in the external symbol table are matched with symbols specified in the entry point name table. The corresponding substitution of absolute machine locations is made in the subprograms to complete the linkage.

EXTNL

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 EXTNL	REQUIRED, OPTIONAL, O?M1, OP,Y2	

Description

EXTNL causes the symbols appearing in the address field to be entered into the external symbol table. The ENTRY and EXTNL control operations are used to establish linkage controls for the object program linking loader in the system monitor program. The form of the address term in EXTNL is the same as that in ENTRY. The symbols appearing in the address field are entered in order into the external symbol table. Duplicate external symbols are deleted. A location term will be ignored.

Comments

The symbols in the address field represent entry point names of subprograms and library subroutines which will be called in at load time. If the symbols were not named as external, they would appear to the GASS assembler as undefined symbols. A symbol appearing in the address field of EXTNL must not appear anywhere in the location column or otherwise be defined in the subprogram which contains the EXTNL pseudo operation.

In the output listing of the subprogram where these symbols appear in address fields, the machine language will contain the location of a previous reference to that external symbol. The first reference to the external symbol will contain the machine value of 17777. In the column preceding this reference octal value, the letter X will appear.

System use of EXTNL

The external symbol table is punched out as a part of the binary program deck. The information in the table is added to information on external symbols from other subprograms which are loaded at the same time. When all subprograms are loaded, each entry in the external symbol table is matched with the corresponding symbolic member from the entry point name table. The absolute location of the named entry point is then placed in the object program and replaces the chain of references to the given external symbol.

Caution

Each reference to a location defined as external by the EXTNL pseudo operation must appear as the symbol alone, and the instruction which references the external symbol must be a two-word instruction. An external symbol may appear as a constant. The following coding shows legal and illegal use of an external symbol.

Legal Example

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8 9	LDM	FØØ	10...50
	STMX,5	RES	
	JPR	FUNCTM	
	LDF	A	
	JFI	/	
		START	
A		FUNCTM	
	EXTNL	FØØ, FUNCTM, RES	
START	HLT		

Illegal Example

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8 9	EXTNL	FØØ, FUNCTN, RES	10...50
	LDD	FØØ (ØNE WØRD CØMMAND)	
	STM	RES+5 (MØDIFIED)	
	LDRB	FUNCTN (DANGERØUS)	

LOCAL

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
REQUIRED	LOCAL	OPTIONAL, OPN	

Description

LOCAL causes ensuing symbols which appear in the location field to
be considered local to the coding which follows the LOCAL pseudo
operation. Symbols which will be referenced from outside of the
local region are specified in the address field of the LOCAL pseudo
instruction. A location symbol must appear with the first LOCAL
pseudo instruction of a local region. If the number of symbols refer-
enced from outside of the local region will not fit in the address field
of a card, the address field can be continued by another LOCAL
pseudo operation which does not have a symbol in the location field.

Comments

LOCAL defines a region of coding which extends from the LOCAL
pseudo operation to the next LOCAL pseudo operation with a location
symbol, or to the NONLC pseudo operation. Symbols which appear
in the location field are unique to the local region. This feature allows
a programmer to reuse symbols as often as desired, provided each
use occurs only once in each local region. Correspondingly, a pro-
grammer can freely use symbolic library programs without any
trouble due to duplicate symbols between the main program and the
library routines.

Any reference to a symbol within a local region from outside of the
region is made by writing the symbol in the address field of the
LOCAL pseudo operation.

System use of LOCAL

Each LOCAL pseudo operation which contains a symbol in the loca-
tion field establishes a local symbol table and also closes out the pre-
vious local symbol table. The address field of the LOCAL pseudo
operation contains a list of symbols, which if encountered in the loca-
tion field in the local area, will be entered into the common symbol
table for the main program. For a program with a large number
symbols which may overflow the internal symbol table, the local
symbols will be put on tape and called back as needed.

4-9

NONLC

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
`1 2 3 4 5 6 7 8`	`NØNLC` `9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50`		

NONLC terminates a local region which was initiated by LOCAL.
If there was no LOCAL pseudo operation, the NONLC will be ig-
nored by the assembler.

LIBA, LIBI

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
`1 2 3 4 5 6 7 8`	`LIBA, n` `9 10 11 12 13 14 15 16 17 18 19`	`NAME1, NAME2, ..., NAMEn` `20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50`	

Description

LIBA (include library routine in assembly form) causes the symbolic
library routines specified in the address field to be included as input
to the assembly process for inclusion in the current program. The
operation modifier, n, specifies the number of routines which are to
be called in. The names in the address field are the identifiers
which occur on the IDENT cards in the library routines. The
library routines which are carried on the system library tape in
assembly coding, conform to the rules for writing library routines.

In case more routines are required than there is space on one card
to name them, a continuation of the LIBA pseudo instruction is made
by giving the modifier, n, on the first card the value which will in-
clude all routines desired. The continuation card will contain LIBA,
without a modifier, and a continuation of the names of routines. On
each card of a continued LIBA, the last routine named must be
followed by a comma and a blank column. A routine name must not
be split between two cards.

Example

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	LIBA, n	NAME1, NAME2, NAME3, NAME4,	
	LIBA	NAME5, •••, NAMEn,	

The library routines will be included in the program at the point that the LIBA pseudo instruction occurs. GASS will treat them as a continuation of the programmers own coding.

Comments

One pass of the symbolic library tape will occur when an LIBA pseudo instruction is encountered. All routines named on the LIBA card will be included in the programmers coding at the point the cards occur. The routines will occupy the bank and location indicated by the present setting of the program location counter.

The library routines include a size indication. The assembly process will have an error indicated if the library routines included cause a program to overflow one bank of memory.

Routines on the library tape may also call for other routines from the library tape. In this case, an extra pass of the library tape may be caused for each level of routine calling within the library tape. The programmer can prevent the extra passes of the tape by including all routines which may be called in his LIBA instruction.

Several LIBA pseudo instructions may be given at different places in the main program. Each occurance of LIBA causes a pass of the library tape. If the same routine is called in two different LIBA pseudo instructions, it appears in the program twice with the resulting duplication of symbols.

System use of LIBA

The function of the LIBA pseudo instruction is to include the given library routine names into a library search table. This table is filled until the end of the LIBA address field, (as indicated by a symbol followed by a blank column (whether the first or the end of a group of continuation cards). Then the library tape is searched, and each IDENT card from the library tape is compared with the library search table. If a match is found, the library routine is

4-11

then treated as normal input up to the END card of the library routine. Then the search continues. The library routine names in the address field of the LIBA may occur in any order; however, all routines within one LIBA pseudo instruction will appear in the program in the order they appear on the library tape.

The required library routines are copied on the intermediate tape as they go through the first pass of the assembly; therefore, an intermediate tape must be used if LIBA occurs in a program.

LIBS

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 LIBS, n 10 11 12 13 14 15 16 17 18 19	NAME1, . . . , NAMEn 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	

Description

LIBS (library routine from the system (binary) tape) causes the routines named in the address field to be included in the program at running time when the program is to be run under MASS control. LIBS has the same form as the LIBA pseudo instruction. The LIBS pseudo instruction must be the last card before the END card of the subprogram.

System use of LIBS

LIBS causes a system library call card to be punched as a part of the binary program deck. This card is used by MASS to determine which binary programs are to be included from the binary system tape.

4-12

STORAGE ALLOCATION

The storage allocation pseudo instructions give the programmer control of the positioning of the program in the core storage of the computer and also enable him to allocate data storage to be associated with the program. The data storage may be unique to one program (BSS, BLR, and BES) or it may be in an area common to a number of subprograms (COMN).

BSS

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
OPTIONAL	BSS	# OR SYMBOL	

BSS reserves a block of consecutive addresses and assigns the location symbol, if present, to the first location of the block. The symbol in the location field is optional. The evaluation of the address fields specifies the number of locations to be reserved. Any symbols which occur in the address field expression must be previously defined. A negative value for the address field is considered as an error, and the BSS will be ignored. A zero address field leaves no space, but it assigns a value to the symbol appearing in the location field.

BLR

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
OPTIONAL	BLR	# OR SYMBOL	

BLR operates the same as BSS.

BES

LOCATION		OPERATION	ADDRESS FIELD	COMMENTS
ØPTIØNAL		BES	# ØR SYMBOL	

`1|2|3|4|5|6|7|8 | 9 | 10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38|39|40|41|42|43|44|45|46|47|48|49|50`

BES reserves a block of consecutive addresses and assigns the location symbol, if present, to the last location of the block. The symbol in the location field is optional. The evaluation of the address fields specifies the number of locations to be reserved. Any symbols which occur in the address field expression must be previously defined. A zero value for the address field leaves no space, but it assigns a value to the symbol appearing in the location field.

> NOTE: The BSS, BLR, and BES pseudo oper-
> ations will result in relocatable addresses
> if they are used under control of the
> ORG-PRG counter. They will result
> in non-relocatable addresses if they are
> used under control of the CON counter.

LOCM

LOCATION		OPERATION	ADDRESS FIELD	COMMENTS
		LØCM	REQUIRED	

`1|2|3|4|5|6|7|8 | 9 | 10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38|39|40|41|42|43|44|45|46|47|48|49|50`

Description

LOCM (Locate common) specifies the beginning address assignment for the following COMN statements. The address term is required and may take two basic forms:

1. Assign a bank and location with a bank. This form would be written as LOCM,# LOCN where # is a bank number and LOCN is a location within a bank.
2. Assign a bank and location within a bank from the implicit bank assignment which is available.

The general form of the address field of the LOCM is the same as applied to entire memory addressing forms for computer instructions.

Comments

The LOCM specifies the beginning value for the assignment of common storage. The address term may be any value which is defined. Uses of LOCM in addition to specifying the beginning of the common area include defining an overlay of common name assignments, defining sub-arrays within another array, etc.

COMN

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8 9	CØMN	A(2),B(2,3),C(2,3,4),D,E	10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Description

COMN defines the data to be included in a common block for reference from several subprograms. The data may be expressed as arrays or as single symbols. A location symbol will be ignored. The address field defines the arrays to be included in the block. The address field is terminated by the first blank character encountered. Array definitions are separated by commas, the general form of the address field is as follows:

$$A(i,j,k),B(1,m,n), \text{ etc.},$$

where i,j,k are the dimensions of the array. An array is restricted to a maximum of three dimensions. In the example and in practice, i,j,k l,m, and n must appear as integer constants. For a two-dimensional array, the third subscript should be omitted. For a one-dimensional array, only one subscript should appear. For a single element, the entire term within the parentheses and the parentheses enclosing this term are omitted.

The assembler will sum the expressed sizes of the arrays for all common data in one block. This sum will be the total number of computer words reserved for the block. The first element of the first array in the block will occupy the location specified in the previous LOCM pseudo operation. Successive words and arrays will occupy successively higher locations. Successive COMN statements will continue filling the area started by a LOCM pseudo operation.

4-15

Address reservation under the control of the COMN pseudo operation is considered to be non-relocatable.

If the number of arrays to be specified exceeds the capacity of one card, the common specification can be extended to a second or more cards by writing COMN and continuing the arrays. An array definition must be completed on one card.

Example

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	CØMN	A(15,15),B(3,4,5),	
	CØMN	C(2,3),D,E,F	

It should be noted that the occurrence of A, B, C, D, E, and F in the preceding common statement causes the location of the first element in the array to be assigned in the symbol table. (This is the only case in which an assignment will be made to a symbol which does not appear in the location field of a card.)

Comments

The arrays defined in COMN may have up to three dimensions. The general form is $A(i,j,k)$ where the value of i is assumed to vary most rapidly, j next most rapidly, and k the slowest. For example, the array defined in common as $A(3,3,2)$ defines an array of 18 elements. The individual elements of the array would be referred to in the main part of a program as $A(1,1,1)$, $A(3,2,1)$, etc. The allocation of the members of the array would be as follows, assuming that the first element of the array appeared at location 1000_8.

Location	Array element
1000	A(1,1,1)
1001	A(2,1,1)
1002	A(3,1,1)
1003	A(1,2,1)
1004	A(2,2,1)
1005	A(3,2,1)
1006	A(1,3,1)
1007	A(2,3,1)
1010	A(3,3,1)
1011	A(1,1,2)
1012	A(2,1,2)
1013	A(3,1,2)
1014	A(1,2,2)
1015	A(2,2,2)
1016	A(3,2,2)
1017	A(1,3,2)
1020	A(2,3,2)
1021	A(3,3,2)

If a symbol is defined as an array name, e.g. MATRIX (3,2), references to its elements must be made in the same dimension or less than the one in which the array name was defined. Thus, a reference to MATRIX would refer to element (1,1) of the array. A reference to MATRIX(3) would refer to element(3,1), and a reference to MATRIX(3,2,2) would be illegal.

DATA DEFINITION

The data definition pseudo instructions cause data to be assembled into the subprogram or into a common block. A data definition pseudo operation will cause space to be reserved and data to be inserted into this space for entry into the computer when the object program is loaded into the machine. Data definitions which are made after a PRG or ORG statement (any time after the statement) are assigned to the private storage occupied by the program. This data will be moved with the program as the program is relocated. Data definitions following a CON statement are fixed in location in the memory.

A CON pseudo operation may be used to originate a sequence of data in the common area defined by a COMN pseudo operation. In this case, the address field of the CON pseudo operation is the name of an array defined within the common block. After the pre-setting operation is completed, a PRG instruction, or a CON with a symbol or number in the address field, may be used to resume the subprogram assignments.

Example

If it is desired to set the values 1 to 9 in an array A defined in a COMN statement, the coding would be as follows:

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	COMN	B(9), A(9), C(3,2)	
	CON	A	
	DEC	1, 2, 3, 4, 5, 6, 7, 8, 9	
	PRG		

DAF

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
OPTIONAL	DAF	139, 23B, $F00, A, A/2, -32	

DAF (Data Former) causes constants to be inserted into consecutive machine words. A location term is optional. If present, it will be assigned to the first word of the group of constants. The address field consists of one or more consecutive subterms, separated by commas and terminated by a blank. Each subterm specifies one constant which may take any form that is legal for the address term of a machine instruction. Thus, the subterm may be an octal number, a decimal number, the value of a symbol, etc.; or the subterm may be an arithmetic combination of quantities. Each subterm is evaluated modulo $2^{13} - 1$.

OCT

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
ØPTIØNAL	ØCT	1/05,-3,7700	

OCT causes octal constants to be Inserted Into consecutive machine words. A location term Is optional. If present, it wlll be assigned to the first word of the group of ocjal constants. The address field consists of one or more consecutive subterms, separated by commas. Each subterm specifies one constant as a slgn (+,-, or none), followed by up to 5 octal digits. Each constant Is assigned to a separate word. A negative constant Is stored as the ones complement of the positive value.

DEC

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
ØPTIØNAL	DEC	1/239,902,-481	

DEC causes decimal constants to be Inserted Into consecutive machine words. A location term Is optional. If present, It will be assigned to the first word of the group of decimal constants. The address field consists of one or more consecutive subterms, separated by commas. Each subterm specifies one constant as a sign (+,-, or none), followed by a value of up to 4 decimal digits. The decimal number must be less than 8192 In absolute value.

BCD

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
ØPTIØNAL	BCD	1/1, BCD MESSAGE	

BCD causes binary-coded-decimal characters to be inserted, two per word, into consecutive words. A location term is optional. If used, it will be assigned to the first word of the group. The address field consists of a number n where $n \leq 50$ or a previously defined symbol which is followed by a comma and n succeeding characters, including blanks. The result is n/2 computer words, each containing 2 BCD characters. The order of the characters In the

words is as follows: the first character goes to bits 11 through 6 of the first word, the second character goes to bits 5 to 0 of the first word, etc. If n is odd, the last word will contain a blank code in the lower six bits. In all cases, the highest-order bit of the words is zero. Anything after n characters is treated as remarks.

FLX

LOCATION		OPERATION	ADDRESS FIELD	COMMENTS
OPTIONAL		FLX	13, TYPE MESSAGE*R	

FLX causes Flexowriter coded characters (input/output typewriter code) to be inserted, two per word, into consecutive words. A location term is optional. If used, it will be assigned to the first word of the group. The address field consists of a number n where $n \leqq 50$ or a previously defined symbol which is followed by a comma and n succeeding typewriter code equivalent characters, including blanks.

> NOTE: Certain special functions such as carriage return, tab, etc. are not in the BCD code. They are represented by two-letter groups where the first character is asterisk (*). These special groups are counted as one character.

The result is n/2 computer words, each containing 2 typewriter characters. The order of the characters in the words is the same as in BCD. If n is odd, the last word will contain zeros as the lower six bits. This code is ignored by the typewriter or Flexowriter. In all cases, the highest-order bit of the words is zero. Anything after n typewriter characters is treated as remarks.

The assembler control pseudo instructions give the programmer control of the allocation of space in the final object program and also allow him to specify the modes of operation of the assembler.

OSAS

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	OSAS	

OSAS causes the assembler to accept cards in OSAS format until a GASS pseudo instruction is encountered. If no OSAS pseudo instruction occurs and the operator has not specified OSAS mode of operation, GASS format is assumed for all cards. An L term, if present, will be ignored. Everything beyond column 14 is treated as comments.

> CAUTION: Numbers under the OSAS format are treated as octal numbers if there is no designation to the contrary. Numbers in GASS format are treated as decimal numbers if there is no designation to the contrary.

GASS

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	GASS	

GASS pseudo instruction switches the input format to GASS if an OSAS pseudo instruction occurred previously. If the current format mode is GASS, this pseudo instruction will be ignored. Everything beyond column 14 is treated as comments. A location term, if present, will be ignored.

> CAUTION: See discussion under OSAS regarding number conventions.

BNK

LOCATION	OPERATION ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 $BNk,\#$	

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Description

BNK causes the words following this control operation to be loaded into the bank specified by the operation modifier if the location counter is set to locations in bank zero or one.

Comments

The BNK pseudo instruction allows a symbolic program to be assigned to a different bank than the one for which it was written. This action only applies if there is no bank specified or implied in the setting of the PRG-ORG counter or in the setting of the CON counter. If the program location counters are set to bank 2 or higher, the bank pseudo instruction is ignored.

This instruction has the following two operations under the preceding restriction:

1. It causes all symbols appearing in the location field of the ensuing instructions to have the bank number appended to them in the symbol table. This feature will be apparent in the entire memory mode of instructions which refer to these symbols.
2. It causes a bank specification card to be punched in the binary program deck. In the loading, BNK takes precedence over all address assignments to place data in banks 0 and 1. It does not change bank 2 and higher numbered banks.

4-22

ORG

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
₁₂₃₄₅₆₇₈	₉ ØRG	#ØR SYMBOL	

Description

ORG enters the value obtained from the address field into the ORG-PRG counter and causes this counter to assume control of the word location process. All words assembled under control of the ORG-PRG counter are relocatable. Any symbol which occurs in the address field must have been previously defined. A location term, if present, will be ignored. The address field may take a value which implies a bank setting other than zero (for example, ORG 040000B). In this case, the program is restricted to the bank implied, but the program may be relocated within that bank.

Comments

The ORG pseudo operation activates the use of the ORG-PRG counter in assigning locations to the following words in the assembly process. The counter is set to the value of the evaluated address field or to zero if the address field is missing. The instruction immediately following the ORG pseudo instruction is assembled to the location contained in the ORG-PRG counter as a result of the pseudo operation. If no counter setting pseudo instruction appears as the first instruction of a program, the ORG-PRG counter is assumed to be in control and it will begin assigning locations at the first location after MASS in bank 0.

PRG

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
₁₂₃₄₅₆₇₈	₉ PRG		

PRG causes the ORG-PRG counter to assume control of the word location process. The value of the ORG-PRG counter is set to the value of the counter prior to a previous CON statement or to the value specified in the previous ORG statement if no CON statement intervened.

Comments

The ORG pseudo instruction is always used to insert a starting value for assignment of storage locations for a program. As such it must always have a value in its address field, or the value of the address field will be assumed to be zero. In the process of assigning locations, the programmer may desire to assign non-relocatable, or common storage. This is done by the CON pseudo instruction. The PRG pseudo instruction allows the programmer to resume the assignment of relocatable locations from the last location assigned before the CON operation. The same effect could be obtained by using ORG with a value in the address field which defines the correct starting location.

> CAUTION: To operate correctly, the PRG pseudo operation must follow the CON pseudo operation. If no CON appears prior to the PRG pseudo operation, the PRG-ORG counter will be reset to the value given in the previous ORG pseudo instruction.

CON

LOCATION		OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	CØN 10 11 12 13 14 15 16 17 18 19	# OR SYMBOL 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

Description

CON sets the translated value of the contents of the address field into the CON counter and causes this counter to assume control of the word location process. All words assembled under control of the CON counter are non-relocatable.

Comments

The CON counter is initially set to 0 if no CON pseudo operation occurs at the beginning of a program. When a CON pseudo operation is encountered, the CON counter is set to the value of the evaluated address field. If the address field is blank, the CON counter continues counting from the previous value of the counter before an ORG or PRG pseudo operation occurred. As long as the CON counter remains active, it is incremented by one for each word assigned in storage, except when it is reset by the address field of a CON pseudo operation.

4-24

SYSTEM USE OF ORG, PRG, AND CON

The preceding pseudo operations are used to establish the starting point and the type of counter which will be used in constructing the symbol table. Each symbol in the location field of a line in a symbolic program is assigned a numeric value by GASS. This numeric value is determined by the current value of the assignment counter, or by the EQU pseudo instruction. The corresponding symbol and numeric value is stored in the symbol table. Symbols in the table are classified as relocatable and constant. The relocatable symbols are flagged as such on the binary program output and may be incremented by a relocation constant at the time they are loaded. Locations which refer to a constant symbol are not relocated or modified at the time the program is loaded.

EQU

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
REQUIRED	EQU	# OR SYMBOL	

EQU equates a symbol to the value of the address field. Thus, it provides a means of establishing a numeric equivalent for the symbol appearing in the location field. Any symbols which occur in the address field must be previously defined. A blank location term is meaningless.

MASS

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
	MASS		

Description

MASS terminates the assembly process and causes GASS to return control to the operating system. The MASS pseudo instruction should follow the END pseudo instruction of the last subprogram to be assembled by GASS. If MASS follows any card which is not END, control will be returned to MASS following the second pass of the assembly process, but a 0 error will be flagged on the output listing. A location term is meaningless and, if present, will be ignored. Everything beyond column 14 is treated as comments.

4-25

Comments

If an END card is missing from the last program to be assembled,
MASS will provide the same function as END, but an error will
be signaled as previously described.

OUTPUT LISTING CONTROL

The output listing control pseudo instructions give the programmer control over the appearance of the listing of the program which is assembled and also allow him to include additional remarks in the program.

REM

LOCATION	OPERATION ADDRESS FIELD	COMMENTS
₁ ₂ ₃ ₄ ₅ ₆ ₇ ₈	9 REM PROGRAMMER REMARKS	

REM produces an output record on the program listing which con-
tains remarks only. The pseudo instruction is ignored by GASS as
far as producing information in the binary program output. All
columns except 9 to 13 inclusive are available for remarks. On the
listing, the code REM is suppressed.

EJECT

LOCATION	OPERATION ADDRESS FIELD	COMMENTS
₁ ₂ ₃ ₄ ₅ ₆ ₇ ₈	9 EJECT	

EJECT will produce a character in column one of the next record
of the magnetic tape listing. This character causes the line printer
to eject paper to the top of the next page. Also if listing is perform-
ed on line, the eject action will take place. The input record con-
taining the EJECT pseudo instruction will be suppressed on the list-
ing.

SPACE

LOCATION	OPERATION ADDRESS FIELD	COMMENTS
₁ ₂ ₃ ₄ ₅ ₆ ₇ ₈	9 SPACE #	

SPACE will cause the listing to be spaced the number of lines
specified by the address field. If this spacing would cause an over-
flow at the bottom of the page, the page is ejected to the top of the
next page only. The input record containing the space pseudo
instruction will be suppressed on the listing.

NOLIST

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 NOLIST 10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	

NOLIST will cause GASS to suppress the output listing when it
appears. This suppression will continue until a LIST pseudo in-
struction is encountered or until the END pseudo instruction is
encountered. The input record containing NOLIST will be sup-
pressed on the output listing.

LIST

LOCATION	OPERATION	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 LIST 10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	

LIST will cause GASS to resume the output listing when it is en-
countered if the listing was previously suppressed by the NOLIST
pseudo instruction. The input record containing LIST will be
suppressed on the output listing. If NOLIST has not been encoun-
tered previously, LIST will be ignored.

CHAPTER 5

INSTRUCTIONS

The 160G instructions can be classified into six groups, depending on the number of words occupied by the instruction and the layout of the instruction. The most general form of a 160G instruction is as follows:

F	E	G
7 Bits	6 Bits	13 Bits
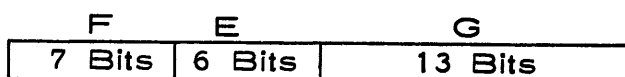
Group 1 instructions have only a 13-bit operation code. This group will be indicated as FE.

Group 2 instructions have a 10-bit operation code and a 3-bit e portion. This group will be indicated as Fe   e.

Group 3 instructions have a 7-bit operation code and a 6-bit E portion. This group will be indicated as   F   E.

Group 4 instructions have a 13-bit operation code and a 13-bit G portion. This group is indicated as FE   G.

Group 5 instructions have a 7-bit operation code and a 19-bit E and G portion. This group is indicated as F   EG.

Group 6 instructions have a 7-bit operation code, a 6-bit E portion, and a 13-bit G portion. This group is indicated as F   E   G.

GASS FORM OF INSTRUCTIONS

The instructions from each group will be listed along with the acceptable forms of writing the instruction in A coding.

GROUP 1 FE

Group 1 instructions are written as the operation codes given below. All information after the operation code is ignored and treated as comment.

The GASS form of group 1 instructions is:

ERR	LS6	LPS	SRS	ETA2	INA6	CIL3
PTA	MUT	SCS	RAS	ETA3	INA7	CIL4
LS1	MUH	LDS	AOS	ETA4	CBC2	CIL5
LS2	RS1	LCS	INA	ETA5	CBC3	CIL6
CBC	RS2	ADS	OTA	ETA6	CBC4	CIL7
ETA	CIL	SBS	HLT	ETA7	CBC5	AMOD
LS3	CTA	STS	ERRG	INA2	CBC6	GMOD
				INA3	CBC7	CTAQ
				INA4	CIL1	XAQ
				INA5	CIL2	HLTG

GROUP 2 Fe e

The general form for group 2 is as follows:

    OPN,X or OPN X

where X is a digit from 0 through 7 or a symbolic expression. The value of the expression must be in the range of 0 through 7 or the instruction will be flagged with an R(range) error on the output listing and the value 0 inserted.

This group of commands is written as follows:

NOP,X	SRJ,X	SIC,X	IRJ,X	SDC,X	DRJ,X
SID,X	ACJ,X	SBU,X	STP,X	STE,X	NOPG,X

GROUP 3 F E

Group 3 instructions are written in the general form:

    OPN X or OPN,X

where X is a number from 0 through 63_{10} (or 0 through 77_8) or a symbolic expression. The value of the symbolic expression must also fall in the range specified for the preceding number given with three exceptions. If the last letter of the mnemonic operation code is F, B, or R indicating relative addressing and the expression can be reduced to the form SYMBOL + # or SYMBOL - #, GASS will form the difference of the symbolic location and the location of the instruction. This difference will be accepted as E if it meets the limit on the size of E.

The group 3 instructions are written as follows:

```
LPN X   SCN X   LDN X   LCN X   ADN X   SBN X   LPD X   LPI X
LPF X   LPB X   LPR X   SCD X   SCI X   SCF X   SCB X   SCR X
LDD X   LDI X   LDF X   LDB X   LDR X   LCD X   LCI X   LCF X
LCB X   LCR X   ADD X   ADI X   ADF X   ADB X   ADR X   SBD X
SBI X   SBF X   SBB X   SBR X   STD X   STI X   STF X   STB X
STR X   SRD X   SRI X   SRF X   SRB X   SRR X   RAD X   RAI X
RAF X   RAB X   RAR X   AOD X   AOI X   AOF X   AOB X   AOR X
ZJF X   NZF X   PJF X   NJF X   ZJB X   NZB X   PJB X   NJB X
ZJR X   NZR X   PJR X   NJR X   JPI X   JFI X   INP X   OUT X
OTN X   EXF X   HWI X   SLS X   ARS X   ALS X   QRS X   QLS X
LRS X   LLS X   SDCG,X  SICG,X
```

GROUP 4   FE   G

Group 4 instructions are written in the general form:
     OPN X   or   OPN,X
where x is a number or symbolic expression with a value of 0
through 8191_{10} or 0 through 17777_8.

The group 4 commands are written as:

```
BLS X   ATE X   ATX X   LPM X   LPC X   SCM X   SCC X
LDM X   LDC X   LCM X   LCC X   ADM X   ADC X   SBM X
SBC X   STM X   STC X   SRM X   SRC X   RAM X   RAC X
AOM X   AOC X   JPR X   IBI X   IBO X   EXC X   BBC2 X
BBC3 X  BBC4 X  BBC5 X  BBC6 X  BBC7 X  ATE2 X  ATE3 X
ATE4 X  ATE5 X  ATE6 X  ATE7 X  ATX2 X  ATX3 X  ATX4 X
ATX5 X  ATX6 X  ATX7 X
IBI2 X  IBI3 X  IBI4 X  IBI5 X  IBI6 X  IBI7 X
IBO2 X  IBO3 X  IBO4 X  IBO5 X  IBO6 X  IBO7 X
EXC2 X  EXC3 X  EXC4 X  EXC5 X  EXC6 X  EXC7 X
RCJP X  ZJRB X  NZRB X  PJRB X  NJRB X  UJRB X
BITJ X  JRIB X  JPIB X  LPIB X  LPRB X  SCIB X
SCRB X  LDIB X  LDRB X  LCIB X  LCRB X  ADIB X
ADRB X  SBIB X  SBRB X  STIB X  STRB X  SRIB X
SRRB X  RAIB X  RARB X  AOIB X  AORB X  LQIB X
LQRB X  SQIB X  SQRB X  SQC X   MUIB X  MURB X
MUC X   DVIB X  DVRB X  DVC X
```

5-3

GROUP 5 F EG

Group 5 instructions may be written in either of the following forms:

1. OPN X
2. OPN,x X

In form 1, X may be a number less than 2^{17} or a symbolic expression which is evaluated to less than 2^{17}. The bank designation which appears in the E portion of these instructions is supplied by the symbol table in GASS.

In form 2, the address is split to bank designation and location within bank. In this case, x is a bank designator which is a number from 0 through 37_8 or an expression which has a value in the same range. The quantity X may be a number less than 2^{13} or a symbolic expression which is evaluated to a number less than 2^{13}.

Under option 1, the group 5 instructions are written as follows:

JPRG X   DRJP X   SRJP X   ZJ X   NZ X   PJ X   NJ X
LP X   SC X   LD X   LC X   AD X   SB X   ST X
SR X   RA X   AO X   LQ X   SQ X   HW X   MU X
DV X

Under option 2, the Group 5 instructions are written as follows:

JPRG,x X   DRJP,x X   SRJP,x X   ZJ,x X   NZ,x X PJ,x X
NJ,x X   LP,x X   SC,x X   LD,x X   LC,x X AD,x X
SB,x X   ST,x X   SR,x X   RA,x X   AO,x X LQ,x X
SQ,x X   HW,x X   MU,x X   DV,x X

GROUP 6 F E G

Group 6 instructions which include indexing are written in the general form

    OPN,I X

where I is a number in the range of 2 through 63_{10} (or 2 through 77_8) or a symbolic expression which when evaluated is in the same range as the number. X is a number or symbolic expression which has a value of 0 through 8191_{10} (or 0 to 17777_8).

The group 6 instructions are written as follows:

```
LPMX,I X   SCMX,I X   LDMX,I X   LCMX,I X   ADMX,I X
SBMX,I X   STMX,I X   SRMX,I X   RAMX,I X   AOMX,I X
LQMX,I X   SQMX,I X   MUMX,I X   DVMX,I X
```

APPENDIX A

PSEUDO INSTRUCTIONS

APPENDIX B

MACHINE INSTRUCTIONS

NOTE: EB - Entire Bank; EM - Entire Memory

F	E	G	Mnemonic	Operation
000	00		ERR	Error Stop
000	0X		NOP	No OP
000	1X		SRJ	Set Relative Bank Control; Jump
000	2X		SIC	Set Indirect Bank Control
000	3X		IRJ	Set Indirect and Relative Bank Controls; Jump
000	4X		SDC	Set Direct Bank Control
000	5X		DRJ	Set Direct and Relative Bank Controls; Jump
000	6X		SID	Set Indirect and Direct Bank Controls
000	7X		ACJ	Set Direct, Indirect, and Relative Bank Controls; Jump
001	00	X	BLS	Block Store
001	01		PTA	P to A
001	02		LS1	Left Shift One
001	03		LS2	Left Shift Two
001	04		CBC	Clear Buffer Controls
001	05	X	ATE	A to BER
001	06	X	ATX	A to BXR
001	07		ETA	BER to A
001	10		LS3	Left Shift Three
001	11		LS6	Left Shift Six
001	12		MUT	Multiply A by 10_{10}
001	13		MUH	Multiply A by 100_{10}
001	14		RS1	Right Shift One
001	15		RS2	Right Shift Two
001	20		CIL	Clear Interrupt Lockout
001	30		CTA	Bank Controls to A
001	4X		SBU	Set Buffer Bank Control
001	5X		STP	Store P at Location 5X
001	6X		STE	Store BER at 6X, A to BER
002	XX		LPN	Logical Product No Address
003	XX		SCN	Selective Complement, No Address
004	XX		LDN	Load No Address
005	XX		LCN	Load Complement, No Address

F	E	G	Mnemonic	Operation
006	XX		ADN	Add No Address
007	XX		SBN	Subtract No Address
010	XX		LPD	Logical Product Direct
011	00	X	LPM	Logical Product Memory
011	XX		LPI	Logical Product Indirect
012	00	Y	LPC	Logical Product Constant
012	XX		LPF	Logical Product Forward
013	00		LPS	Logical Product Specific
013	XX		LPB	Logical Product Backward
014	XX		SCD	Selective Complement Direct
015	00	X	SCM	Selective Complement Memory
015	XX		SCI	Selective Complement Indirect
016	00	Y	SCC	Selective Complement Constant
016	XX		SCF	Selective Complement Forward
017	00		SCS	Selective Complement Specific
017	XX		SCB	Selective Complement Backward
020	XX		LDD	Load Direct
021	00	X	LDM	Load Memory
021	XX		LDI	Load Indirect
022	00	Y	LDC	Load Constant
022	XX		LDF	Load Forward
023	00		LDS	Load Specific
023	XX		LDB	Load Backward
024	XX		LCD	Load Complement Direct
025	00	X	LCM	Load Complement Memory
025	XX		LCI	Load Complement Indirect
026	00	Y	LCC	Load Complement Constant
026	XX		LCF	Load Complement Forward
027	00		LCS	Load Complement Specific
027	XX		LCB	Load Complement Backward
030	XX		ADD	Add Direct
031	00	X	ADM	Add Memory
031	XX		ADI	Add Indirect
032	00	Y	ADC	Add Constant
032	XX		ADF	Add Forward
033	00		ADS	Add Specific
033	XX		ADB	Add Backward
034	XX		SBD	Subtract Direct
035	00	X	SBM	Subtract Memory
035	XX		SBI	Subtract Indirect
036	00	Y	SBC	Subtract Constant
036	XX		SBF	Subtract Forward
037	00		SBS	Subtract Specific

F	E	G	Mnemonic	Operation
037	XX		SBB	Subtract Backward
040	XX		STD	Store Direct
041	00	X	STM	Store Memory
041	XX		STI	Store Indirect
042	00	Y	STC	Store Constant
042	XX		STF	Store Forward
043	00		STS	Store Specific
043	XX		STB	Store Backward
044	XX		SRD	Shift Replace Direct
045	00	X	SRM	Shift Replace Memory
045	XX		SRI	Shift Replace Indirect
046	00	Y	SRC	Shift Replace Constant
046	XX		SRF	Shift Replace Forward
047	00		SRS	Shift Replace Specific
047	XX		SRB	Shift Replace Backward
050	XX		RAD	Replace Add Direct
051	00	X	RAM	Replace Add Memory
051	XX		RAI	Replace Add Indirect
052	00	Y	RAC	Replace Add Constant
052	XX		RAF	Replace Add Forward
053	00		RAS	Replace Add Specific
053	XX		RAB	Replace Add Backward
054	XX		AOD	Replace Add One Direct
055	00	X	AOM	Replace Add One Memory
055	XX		AOI	Replace Add One Indirect
056	00	Y	AOC	Replace Add One Constant
056	XX		AOF	Replace Add One Forward
057	00		AOS	Replace Add One Specific
057	XX		AOB	Replace Add One Backward
060	XX		ZJF	Zero Jump Forward
061	XX		NZF	Non-Zero Jump Forward
062	XX		PJF	Positive Jump Forward
063	XX		NJF	Negative Jump Forward
064	XX		ZJB	Zero Jump Backward
065	XX		NZB	Non-Zero Jump Backward
066	XX		PJB	Positive Jump Backward
067	XX		NJB	Negative Jump Backward
070	XX		JPI	Jump Indirect
071	00	X	JPR	Return Jump
071	XX		JFI	Jump Forward Indirect
072	00	X	IBI	Initiate Buffer Input
072	XX	Y	INP	Normal Input

F	E	G	Mnemonic	Operation
073	00	X	IBO	Initiate Buffer Output
073	XX	Y	OUT	Normal Output
074	YY		OTN	Output No Address
075	00	Y	EXC	External Function Constant
075	XX		EXF	External Function Forward
076	00		INA	Input to A
076	XX		HWI	Half Write Indirect
076	77		OTA	Output From A
077	00		HLT	Halt
077	0Y		SLS	Selective Stop
077	Y0	X	SLJ	Selective Jump
077	YY	X	SJS	Selective Stop, Jump
077	77		HLT	Halt
100	00		ERRG	Error Stop
100	0X		NOPG	No Op
100	1Y	Z	BBCY*	Set Buffer Bank Control, Channel Y
100	2Y	X	ATEY*	A to BER, Channel Y
100	3Y	X	ATXY*	A to BXR, Channel Y
100	4Y		ETAY*	BER, Channel Y, to A
100	5Y	X	MTMY*	Memory to Memory, Channel Y
100	6Y	X	IBIY*	Initiate BFR Input, Channel Y
100	7Y	X	IBOY*	Initiate BFR Output, Channel Y
101	0Y	Z	EXCY*	External Function, Channel Y
101	1Y		INAY*	Input to A, Channel Y
101	2Y		CBCY*	Clear Buffer Controls, Channel Y
101	3Y		CILY**	Clear Interrupt Lockout, Channel Y
101	60		AMOD	Select A Mode
101	61		GMOD	Select G Mode
101	62		CTAQ	Bank Controls to AQ
101	63	X	RCJP	AQ to Bank Controls; Jump
101	64		XAQ	Interchange A and Q
101	70	X	ZJRB	Zero Jump Relative - EB
101	71	X	NZRB	Non-Zero Jump Relative - EB
101	72	X	PJRB	Positive Jump Relative - EB
101	73	X	NJRB	Negative Jump Relative - EB
101	74	X	UJRB	Unconditional Jump Relative - EB
101	75	X	BITJ	Bit-by-Bit Jump
101	76	X	JRIB	Jump Relative Indirect - EB
101	77	X	JPIB	Jump Indirect - EB

* Y, which is included in the operation code, must be a digit from 2 through 7.

** Y, which is included in the operation code, must be a digit from 1 through 7.

F	E	G	Mnemonic	Operation
103	ZZ	X	JPRG	Return Jump – EM
104	ZZ	X	ZJ	Zero Jump – EM
105	ZZ	X	NZ	Non-Zero Jump – EM
106	ZZ	X	PJ	Positive Jump – EM
107	ZZ	X	NJ	Negative Jump – EM
110	ZZ	X	LP	Logical Product – EM
111	00	X	LPIB	Logical Product Indirect – EB
111	01	X	LPRB	Logical Product Relative – EB
111	YY	X	LPMX	Logical Product Index
113	YY		ARS	A Right Shift
114	ZZ	X	SC	Selective Complement – EM
115	00	X	SCIB	Selective Complement Indirect – EB
115	01	X	SCRB	Selective Complement Relative – EB
115	YY	X	SCMX	Selective Complement Memory Index
117	YY		ALS	A Left Shift
120	ZZ	X	LD	Load – EM
121	00	X	LDIB	Load Indirect – EB
121	01	X	LDRB	Load Relative – EB
121	YY	X	LDMX	Load Memory Index
123	YY		QRS	Q Right Shift
124	ZZ	X	LC	Load Complement – EM
125	00	X	LCIB	Load Complement Indirect – EB
125	01	X	LCRB	Load Complement Relative – EB
125	YY	X	LCMX	Load Complement Memory Index
127	YY		QLS	Q Left Shift
130	ZZ	X	AD	Add–EM
131	00	X	ADIB	Add Indirect – EB
131	01	X	ADRB	Add Relative – EB
131	YY	X	ADMX	Add Memory Index
133	YY		LRS	AQ Right Shift
134	ZZ	X	SB	Subtract – EM
135	00	X	SBIB	Subtract Indirect – EB
135	01	X	SBRB	Subtract Relative – EB
135	YY	X	SBMX	Subtract Memory Index
137	YY		LLS	AQ Left Shift
140	ZZ	X	ST	Store–EM
141	00	X	STIB	Store Indirect – EB
141	01	X	STRB	Store Relative – EB

F	E	G	Mnemonic	Operation
141	YY	X	STMX	Store Memory Index
143	ZZ	X	SRJP	Set Relative Bank Control; Jump-EM
144	ZZ	X	SR	Shift Replace – EM
145	00	X	SRIB	Shift Replace Indirect – EB
145	01	X	SRRB	Shift Replace Relative – EB
145	YY	X	SRMX	Shift Replace Memory Index
147	ZZ	X	DRJP	Set Direct and Relative Bank Controls; Jump-EM
150	ZZ	X	RA	Replace Add – EM
151	00	X	RAIB	Replace Add Indirect – EB
151	01	X	RARB	Replace Add Relative – EB
151	YY	X	RAMX	Replace Add Memory Index
153	ZZ		SDCG	Set Direct Bank Control – EM
154	ZZ	X	AO	Replace Add One – EM
155	00	X	AOIB	Replace Add One Indirect – EB
155	01	X	AORB	Replace Add One Relative – EB
155	YY	X	AOMX	Replace Add One Memory Index
157	ZZ		SICG	Set Indirect Bank Control –EM
160	ZZ	X	LQ	Load Q – EM
161	00	X	LQIB	Load Q Indirect – EB
161	01	X	LQRB	Load Q Relative – EB
161	YY	X	LQMX	Load Q Memory Index
162	00	X	LQC	Load Q Constant
164	ZZ	X	SQ	Store Q – EM
165	00	X	SQIB	Store Q Indirect – EB
165	01	X	SQRB	Store Q Relative – EB
165	YY	X	SQMX	Store Q Memory Index
166	00	Y	SQC	Store Q Constant
167	ZZ	X	HW	Half Write – EM
170	ZZ	X	MU	Multiply – EM
171	00	X	MUIB	Multiply Indirect – EB
171	01	X	MURB	Multiply Relative – EB
171	YY	X	MUMX	Multiply Memory Index
172	00	X	MUC	Multiply Constant
173	ZZ	X	HILO	High-Low Comparison
174	ZZ	X	DV	Divide – EM
175	00	X	DVIB	Divide Indirect – EB
175	01	X	DVRB	Divide Relative – EB
175	YY	X	DVMX	Divide Memory Index
176	00	X	DVC	Divide Constant
177	XX		HLTG	Halt

APPENDIX C

ERROR CODES

A   Address Field Error. An A error will occur if there is a format error in the address field or in an individual address term.

D   Duplicate Symbol. A D error results if a symbol occurs more than once in a location field of a subprogram. The symbol will be ignored on the second and subsequent occurrences.

F   Full Symbol Table. No assignment is made if a table entry would cause overflow of the symbol table.

I   Ident Error. An I error will occur if an IDENT record appears anywhere except as the first record of a subprogram.

L   Location Term Error. An L error occurs if an L term appears where none is allowed, if an L term is absent where one is required, or if an illegal type symbol appears.

O   Operation Code Error. An O error occurs if an illegal operation code is used. Zeros are substituted.

E   Range Error. The E term of assembled machine OP code is greater than 77_8.

U   Undefined Symbol. A symbol referenced in the address field has not been defined; zeros are substituted.

APPENDIX D

SPECIAL CODES FOR TYPE ENTRIES

BCD CHARACTERS	TYPE EQUIVALENT
*R	Carriage Return
*U	Shift to Upper Case
*L	Shift to Lower Case
*B	Backspace
*T	Tab
*X	!
*A	¡
*S	;

CONTROL DATA

CORPORATION   8100 34th Avenue South, Minneapolis 20, Minnesota