RCA ELECTRONIC 501 DATA PROCESSING SYSTEM

AUTOMATIC ASSEMBLY SYSTEM

A PROGRAMMER'S REFERENCE MANUAL

RADIO CORPORATION OF AMERICA
Electronic Data Processing Division
Camden, New Jersey

RCA ELECTRONIC 501 DATA PROCESSING SYSTEM

AUTOMATIC ASSEMBLY SYSTEM

A PROGRAMMER'S REFERENCE MANUAL

RADIO CORPORATION OF AMERICA
Electronic Data Processing Division
Camden, New Jersey

The data herein presented is subject to minor change without notice.

Supplements may be provided to advise of such revisions or additions.

PREFACE

This manual is presented as a programmers' description of the RCA 501 Assembly System. It is intended to furnish information in sufficient detail to enable programmers to use Assembly language coding. Reference is made to the RCA 501 Electronic Data Processing System Programmers' Reference Manual in this text and it is assumed that the reader is familiar therewith. Operating procedures and description of the subroutines and macro-instruction library are not within the scope of this manual, but will be the subject of separate publications.

Particular emphasis has been placed on the structure and uses of the RCA 501 Assembly System language. Master instruction reference charts are included, for example, as Appendix B.

Acknowledgement is given for help received from several quarters including the Methods Training Group. The creation and development of the Assembly System is the work and responsibility of the Advanced Programming Organization, as is the writing and editing of this manual.

TABLE OF CONTENTS

Page

INTRODUCTION 1

DATA DESCRIPTIONS 3
RCA 501 Automatic Coding Data Sheet 3

THE PROGRAM SHEET AND ITS GENERAL USAGE 12
RCA 501 Automatic Assembly Program Sheet 12
General Methods of Writing Pseudo-Code 12

ADDRESSING 25
Machine Addresses 25
Data Addresses 25
Forward Read Data Addressing 26
Reverse Read Data Addressing 29
General Working Storage Addresses 30
Special Working Storage Addresses 32
Constant Addresses 34
The Define Constant Descriptor Verb 38
Tape Station Addresses 39
Instruction Addresses 40

SPECIAL ASSEMBLER FEATURES 49
Special Mnemonic Instructions 49
The Stash (STSH) Descriptor Verb 52
Variable Addresses 53
The ADV Mnemonic Instruction 54

SPECIAL ASSEMBLY OPTIONS 56
The Tape Sense Instruction 56
Special Addresses with Unwind and Rewind Instructions 57
Special Addresses with Locate nth Symbol 57
Special Addresses with Paper Advance Instruction 58
Other than "Set" Generated by "T" Entries 58
Random Distribute Special Options 60
Multiple Sector Write Options 66
Address Modifier Addresses 69
Special Addressing of Registers 72
Symbolic Addressing of Standard Locations 76

MEMORY LAYOUT 78
Automatic Format 78
Programmer Control of Memory Layout 79

MACRO-INSTRUCTIONS 86
Calling Macro-Instructions 86
Defining Macro-Instructions 86
Programmer-Written Preset 92
Macro-Instruction Segmentation 93
The Replace Macro-Instruction Descriptor Verb 94
The Delete Macro-Instruction Descriptor Verb 95

Page

SUBROUTINES 97
Calling Subroutines 97
Entering Subroutines 98
The Define Subroutine Descriptor Verb 101
The Programmer-Written Preset 102
Subroutine Segmentation 103
Dynamic Parameters 104
The Replace Subroutine Descriptor Verb 106
The Delete Subroutine Descriptor Verb 106

ASSEMBLY SYSTEM OUTPUTS 112
Magnetic Tape Outputs 112
Printed Outputs 112

APPENDICES

A. Programmer Check List Prior to Assembly 115
B. Instruction Reference Tables
Table I – 501 Mnemonic Instructions 118
Table II – Special Assembler Mnemonics 125
Table III – Descriptor Verbs 126
C. Addresses substituted for symbolic names 129
D. Object Program Examiner 130

INDEX 131

I. INTRODUCTION

The RCA 501 Computer is potentially capable of solving a wide variety of problems. Before it can perform this task, however, it must be provided with an appropriate program.

Unfortunately, human beings and computers do not naturally use the same kind of language. A human being, when solving a problem, prefers to express it in his native language, or at the very least, a version of his language which, even with certain restrictions imposed upon it, will remain clear and concise.

The computer on the other hand, being an electronic device, must receive its instructions in a special machine language, consisting of various numeric or alpha-numeric codes. The preparation by human beings of these coded instructions (called "programming") is accomplished by breaking down the various operations connected with the problem and arranging them into a series of basic machine operations which must be performed in order for the computer to solve the problem. Since the computer will do only what it is directed to do, the framing of the instructions must be in precise detail and must state the exact sequence in which data is to be processed, how it is to be manipulated, where computation is to take place, and where the results are to be stored. Provision must also be made for the allocation of available storage space. The process of manual coding is both time consuming and costly.

A plan for alleviating this problem is suggested by the fact that coding is mainly a clerical task and the computer is well suited to handle clerical functions. A method of programming has been designed which utilizes this circumstance and is generally known as "automatic programming". Automatic programming is defined as the use of a language suitable to human beings for problem preparation together with a computer program capable of translating this language to that language which will be acceptable to the computer.

The programmer thus describes problems in a language convenient to himself. This is known as the "subject" program. The subject program is then processed in the computer by the translating program. Depending upon the type of translating program used, the output may be in one of two forms. In brief, these types are:

Interpretive – In this type of system, conversion of the subject program into machine language and the actual manipulation of the problem is performed concurrently. The output is the processed data which results from the solution of the problem. This procedure must be repeated each time a problem is to be performed.

Compiler – This system is one of pure translation or compilation. The output is the "object" program, or the machine code version of the subject program. The object program may then be used to perform the problem on the computer. Translation takes place but once for each program.

Interpretive systems are impractical for data processing applications because the translation and problem solving functions are intermixed. In non-interpretive systems, on the other hand, compilers perform translation and then problem solving (by means of the object program) takes place without the need to again refer to the translating program. The RCA 501 Assembly System is of the compiler type.

Compilers may be further classified by the types of languages available to users. These range from languages which are akin to machine code, to those which resemble basic English. Sub-classifications of compilers may be conveniently defined as either the machine-oriented language type or problem-oriented language type. The terms used to refer to these types are "Assembly" and "Narrative", respectively.

From this general discussion of automatic programming systems, a description of the RCA 501 Assembly System specifically, may better be appreciated.

The RCA 501 Assembly System is a machine-oriented, automatic programming system designed to simplify, and thus expedite, the writing of coding for the RCA 501 Electronic Data Processing System. This Assembly System is said to be "machine-oriented", because the format in which instructions are written with this system (its pseudo-code) is close to the format of actual machine instructions.

Some of the highlights of the RCA 501 Assembly System are:

1. *Relative Instruction Addresses* – The user can specify instruction addresses which are *not* fixed High Speed Memory locations, but are, instead, keys which indicate the order in which instructions are to be placed in memory. Relative instruction addresses are automatically translated into appropriate machine addresses by the Assembly System.
2. *Symbolic Addresses* – The user can refer to data, working storage areas, and constants by symbolic notations. Symbolic addresses are translated into appropriate machine addresses by the Assembly System.
3. *Machine Addresses* – The user may also specify actual machine addresses.
4. *Mnemonic Operation Codes* – For more easily readable programs, mnemonic operation codes, rather than octal number codes, are used; e.g., TC for 71, TCA for 44, BA for 41, etc.
5. *Machine Code* – Actual machine code may also be used *anywhere* in the program.
6. *Macro-instructions* – In most programs there are a number of common and frequently recurring steps which are necessary to accomplish a specific programming task. These groups of instructions have been extracted and written into short open sub-routines which may be inserted into the program by the use of a single macro-instruction. This has the effect of greatly reducing the amount of coding necessary to write a program.
7. *Descriptor Verbs* – There are a variety of verbs which fall into the category of Descriptor Verbs. They are used to intervene and modify the process of translating or compiling the object program. While their use is optional, they offer a convenient means for the programmer to obtain greater efficiency in the final running program.
8. *Variable Address* – Certain addresses may not be known at the time a program is being written. The user may arbitrarily assign a temporary (variable) address and thus provide for future definition of this address, when it becomes known. This provision is accomplished with a given instruction (DEFV) and may be inserted *anywhere* in the program. Thereafter, whenever the designated symbol (variable address) appears in the program, the definition of the address is automatically substituted, added or subtracted by the computer at the proper places in the program.
9. *Flexible Format* – Assembly System pseudo-instructions may be written as 2, 3, 4, 5 or 6 address instructions.
10. *Sub-Routines* – Sub-routines are generally larger bodies of coding than macro-instructions. They perform some major, often recurring, programming function. The Assembly System user may call upon any one of the sub-routines in the system's library with one line of pseudo-code anywhere within his program. Sub-routines may be open-ended or closed and they may call upon macro-instructions. New sub-routines may be added to the library, new sub-routines may replace outdated sub-routines, or old sub-routines may be removed from the library, all during the course of assembly.

The RCA 501 Assembly System pseudo-code thus provides the user with greater flexibility and simplicity in writing coding than does machine code. Moreover, the Assembly System enables its user to employ *all* techniques and coding tricks available to the machine code user.

II. DATA DESCRIPTIONS

To guide the Assembly System in generating a running program from the pseudo-code, the user provides the system with a description of the data files which the program was designed to process. A "file", as defined in the Programmers' Reference Manual, ". . . consists of any number of related information units, in message or block format; it may consist of several tapes (reels) or any part of one tape. A file is terminated by an End File (EF) Symbol, preceded and followed by an Intermessage Gap.

"In a multi-tape file, all but the last tape are terminated by an End Data (ED) symbol alone, preceded and followed by an Intermessage Gap. The end of the file on the last tape is indicated by an EF, preceded and followed by an Intermessage Gap. If this is a full tape, or a partially filled tape with no other valid information following the file data, an ED follows the EF, separated from it and followed by an Intermessage Gap."

The Assembly System recognizes this definition of a file. Files are described on forms RCA IE 241 and RCA IE 241-1.

RCA 501 AUTOMATIC CODING DATA SHEET, FORM RCA IE 241

The first page, and only the first page, of each file description is written on form RCA IE 241. See sample at end of this chapter.

Heading

The heading identifies the file and gives some general information about the file. This information is entered as follows:

Name of File

The file name, abbreviated to a maximum of five characters, is entered in the space provided. The five-character maximum file name is preceded by an "F" which is pre-printed on the form.

If, for instance, a master employee file were being described, the file name might be abbreviated, "MASEM". The entry would then appear as follows:

NAME OF FILE < ● FMASEM

Page ● 1 of ●

After the ISS following the "of", enter the number of pages used to describe this file.

Example:

PAGE● 1 of● 1

No. of Msgs.

Enter the number of messages in the complete file. This need not be an exact count, but should be a good approximation. A maximum of eight decimal digits can be used.

Example:

No. of Msgs.
●30000

Max. Msg. Size

This refers to the number of characters in the largest message in the file. The Assembly System will compute this number from the information given in subsequent entries on these forms, and use this number to assign input areas in memory. *Unless the analyst has definite knowledge that the number which will be computed by the Assembly System will, for some reason, be too small, he should leave this box blank.* If the analyst makes an entry in this box, his entry will take precedence over the figure computed by the Assembly System. A maximum of five decimal digits may be entered.

No. of Reels

An estimate of the number of reels in the file is entered in this box. Two decimal digits may be used. This information is for analyst reference only.

Example:

No. of Reels
● 11

No. of Stations

Enter a number specifying how many tape stations will be used for this file. This information is for analyst reference only.

Example:

No. of Stations
● 2

Keys

This box may be left blank, or the analyst may list from one to five possible sorting keys. Each key listed can contain a maximum of five characters preceded by a "D" which is pre-printed on the form. This information is for analyst reference only.

Example: *

Keys
● DFLNO̲ ● DENO̲ ● DSTNO̲ ● DSHNO̲ ● DMANO̲

* Note that in entries on the Assembly System forms, since it is important that the alpha O be distinquished from the numerical O. The alpha O is always written with an underscore.

Message Format

Is the file in message format? Enter a "Y" for "yes", a "N" for "no" in the provided box.

Example:

Message Format Yes or No ● Y or N
□Y

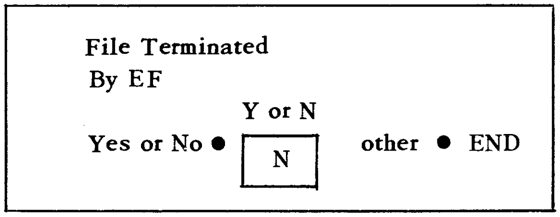
Files Terminated by EF

Is this file terminated by an End File (EF) symbol? Enter a "Y" for "yes",a "N" for "no" in the provided box. If no:

1. If the file is in message format, enter the sentinel that is used after the ISS following "other". A maximum of sixty-four characters may be used to specify the end file sentinel. The Assembly System will assume that the sentinel, as it appears on tape, will be preceded by a Start Message Symbol and be followed by an End Message Symbol; however, these symbols are not entered on the data description form.
2. If the file is in block format, the entered sentinel must be at least eight characters, and no more than sixty-four characters long.

In either event no control symbols, SM, EM, EF, or ED, may be entered as the "other" end file sentinel or part of the "other" end file sentinel.

Example:



File Terminated
By EF
Y or N
Yes or No ● N other ● END

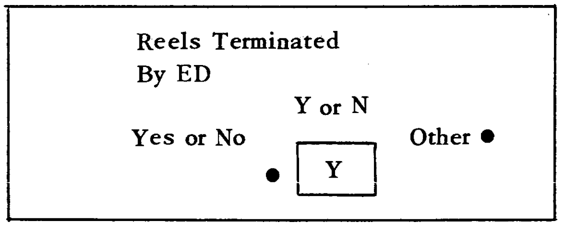
Reels Terminated by ED

Is each reel in the file terminated by an End Data (ED) symbol? Enter a "Y" for "yes", an "N" for "no" in the provided box. If no:

1. If the file is in message format, enter the sentinel that is used after the ISS following "other". A maximum of eighty characters may be used to specify the end data sentinel. The Assembly System will assume that the sentinel, as it appears on tape, will be preceded by a Start Message Symbol and followed by an End Message Symbol; however, these symbols are not entered on the data description form.
2. If the file is in block format, the entered sentinel must be at least eight characters, and no more than eighty characters long.

In either event no control symbols, SM, EM, EF, or ED, may be entered as the "other" end data sentinel or part of the "other" end data sentinel.

Example:



Reels Terminated
By ED
Y or N
Yes or No Other ●
● Y

Remarks

Provisions are made for as many as 230 characters (including space between words) of remarks. These are analyst remarks and may be used at the discretion of the analyst.

Body Message

The body message describes the individual items and sub-items that comprise the messages within the file. Every item which may appear in any message of the file must be described in the body message. Items must be described in the same order in which they may appear in messages on magnetic tape, and must be numbered in that order. Entries are made in the body message as follows:

Item No.

Enter the number of the items. Items are numbered sequentially from 1 to 999.

Sub-Item

If the item described on a line of the form is *not* preceded by an ISS, enter "NOIS" in the Sub-Item column of that line. If an ISS precedes the item, leave this column on the line blank. If the item is composed of sub-items, the sub-item column is used to assign letters, in alphabetical order, to sub-items on *following lines.* Sub-items are parts of an item which one may wish to consider separately from the other parts of the item. For instance, consider a Master Employee file which contains, as the first item in each message, an eleven digit employee number. The number may be considered in five parts:

a. A two-digit plant code, designating the particular plant in which the employee works,

b. A two-digit department code, designating the department within the plant in which the employee works,

c. A two-digit station code, designating the station or machine at which the employee works,

d. A one-digit shift code, designating the shift the employee normally works,

e. A four-digit man number, which is unique to this employee.

For cost accounting purposes, it may be desirable to charge the man's salary against his department and plant individually. It would then be convenient to be able to address the sub-items individually. However, for the purpose of matching his time card information against his master employee file message, the entire employee number is used, and it would be convenient to be able to address the item as a whole.

To gain this facility with the use of the RCA 501 Assembly System, the user specifies such multiple code items as a single item which is further divided into sub-items. Both the item and sub-items are given individual names in the "ABBREVIATION" column.

If an item is divided into sub-items, the LHE of the first sub-item will be the LHE of the item (the ISS, if present). There may be up to 26 sub-items per item.

Abbreviation

Enter an abbreviated name for each item and sub-item described on the form. The abbreviation may be five characters preceded by the pre-printed "D".

Each item of the file must be assigned a unique abbreviation. Moreover, the same item in different files must have different abbreviations.

Example:

Item No.	*Sub-Item*	*Abbreviation*
< ● 1	●	●D E M P N O̲
●	●A	●D P L N O̲
●	●B	●D D E N O̲
●	●C	●D S T N O̲
●	●D	●D S H N O̲
●	●E	●D M A N O̲

Description

Up to twenty-five characters of description for each item and sub-item may be entered in this column for the analyst's convenience.

FAA

If the item being described on a line is of fixed character length, in a fixed position with relation to the first character of the message, and always appears in the message, an "X" is placed in this column on the line. It is not necessary to complete this column for sub-items, since, if the item itself is fixed and always appearing, it will be assumed that its sub-items are also fixed and always appearing. Up to 240 FAA items may appear in any one file.

The Assembly System assumes that "fixed and always appearing" means that the right hand end and left hand end of an FAA item is *always* a fixed character distance from the first character of the message. Therefore, once a variable item is described, *all* succeeding items will be considered variable, regardless of size.

JY

If the item or sub-item being described on a line is justified right, enter an "R" in this column on that line.

If the item or sub-item being described on a line is justified left, enter an "L" in this column on that line.

Sign

If an item has a sign associated with it, enter an "X" in the column on the line describing the item. If not, leave this column blank.

If a sign is indicated, the Assembly System automatically allots an extra location in that field to accommodate the sign. Note, however, that the sign is *not* to be included in the number of characters specified for this item.

No. Chars. – Max.

Enter the maximum number of characters (1 to 999) that can appear in the item or sub-item. Note that only information characters are counted; *sign and ISS are not counted as part of an item or sub-item.*

No. Chars. – Avg.

Enter the average number of characters (1 to 999) that will appear in the item or sub-item. Note that only information characters are counted; *sign and ISS are not counted as part of an item or sub-item.* The entries in "MAX." and "AVG." should be equal for all FAA items and their sub-items.

% Use

Enter the percentage of the messages in the file in which this item or sub-item will appear. Note that this must be "100" for all FAA items.

Wtd. Avg.

The weighted average is the "NO. CHARS.-AVG." times the "% USE". This figure will be computed by the Assembly System, and need not be computed by the analyst. If, however, the analyst wishes to compute this figure and enter it on this form, he may do so; however, *weighted averages must not be punched and entered as input to the Assembly System.*

Totals

At the bottom of the form are spaces for totals. The Assembly System will compute these totals. They need not be computed by the analyst. If, however, the analyst wishes to compute these figures and enter them on this form, he may do so; however, *totals must not be punched and entered as input to the Assembly System.*

RCA 501 AUTOMATIC CODING DATA SHEET, FORM RCA IE-241-1

As stated previously, the first page, and only the first page, of each file description is written on form RCA IE-241. All succeeding pages of a file description, if necessary, are written on form RCA IE-241-1, shown at end of this chapter.

Heading

The heading of form RCA IE-241-1 is a short heading containing merely identification information. All general information about the file will have been described on form RCA IE-241.

Name of File

Enter the same file name abbreviation that was entered on the first page of this file description.

Page● of●

After the ISS following "Page", enter the number of this page in relation to all other pages describing the file.

After the ISS following "of", enter the total number of pages used to describe this file.

Remarks

Up to 230 characters of remarks may be made by the analyst concerning the data described on this page.

Body Message

The body message is filled out in exactly the same manner as in the body message of form RCA IE-241.

ENDING PAGES

Pages may be ended anywhere on the physical page. The analyst, for instance, may elect to put only a full item (with all its sub-items) description on one page, or he may elect to fill all spaces on one page before going on to the next. The amount of information he chooses to put on a page is left to his discretion; however, he must follow certain rules for ending pages:

1. If the page *does not* end a file description, place an EM symbol after the "% USE" entry of the last item described.
2. If the page ends a file description, and this is *not* the last file described, place an EM symbol followed by an EF symbol after the "% USE" entry of the last item of the file.
3. If the page ends the file description, and this is the last file described, place an EM symbol followed by an ED symbol after the "% USE" entry of the last item described.

A form RCA IE-241 showing a single item page, completely entered, is presented at the end of this chapter.

NUMBER OF FILES

The maximum number of files that can be described for a pseudo-code program is 20.

KEYPUNCHING DATA SHEETS

When punching data sheets the following rules must be observed:

Header:

1. All preprinted ISS's in the Header message must be punched.
2. Y or N must be punched in the MESSAGE FORMAT block.
3. Except for the F in *Name of File* entry and the D's in *Keys* entry, preprinted words on the Data Sheet Header are *not* to be punched.

Body:

1. All preprinted ISS's on the form must be punched.
2. Weighted Average column and the Totals at the bottom of the form are *not* to be punched.

501 AUTOMATIC CODING DATA SHEET

NAME OF FILE <•F ______ PAGE• 1 OF•

NO. OF MSGS.	MAX. MSG. SIZE	NO. OF REELS	NO. OF STATIONS	KEYS
•	•	•	•	• D______ • D______ • D______ • D______ • D______

MESSAGE FORMAT YES OR NO • Y OR N □

FILES TERMINATED BY EF YES OR NO • Y OR N □ OTHER •

REELS TERMINATED BY ED YES OR NO • Y OR N □ OTHER •

REMARKS •

> X OR BLANK | L OR R | X OR BLANK

ITEM NO.	SUB ITEM	ABBREVIATION	DESCRIPTION	F A A	JY	S I G N	NO. CHARS. MAX.	NO. CHARS. AVG.	% USE	WTD. AVG.
<•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	

TOTAL CHARACTERS OF INFORMATION			
ADDED CONTROL CHARACTERS			
TOTAL CHARACTERS			

501 AUTOMATIC CODING DATA SHEET

NAME OF FILE <• F __ __ __ __ __

PAGE• OF•

REMARKS •

			>	X OR BLANK	L OR R	X OR BLANK				
ITEM NO.	SUB. ITEM	ABBREVIATION	DESCRIPTION	FAA	JY	SIGN	NO. CHARS.		% USE	WTD. AVG.
							MAX.	AVG.		
<•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	

TOTAL CHARACTERS OF INFORMATION			
ADDED CONTROL CHARACTERS			
TOTAL CHARACTERS			

501 AUTOMATIC CODING DATA SHEET

NAME OF FILE <•F MASEM

PAGE• 1 OF• 8

NO. OF MSGS.	MAX. MSG. SIZE	NO. OF REELS	NO. OF STATIONS	KEYS
• 30000	•	•	•	• D____ • D____ • D____ • D____ • D____

MESSAGE FORMAT YES OR NO • Y OR N: Y

FILES TERMINATED BY EF YES OR NO • Y OR N: □ OTHER •

REELS TERMINATED BY ED YES OR NO • Y OR N: □ OTHER •

REMARKS •

Employee sp No

> X OR BLANK / L OR R / X OR BLANK

ITEM NO.	SUB ITEM	ABBREVIATION	DESCRIPTION	FAA	JY	SIGN	NO. CHARS. MAX.	NO. CHARS. AVG.	% USE	WTD. AVG.
<• 1	•	• D EMPNO	• Employee No.	•X	• L	•	• 11	• 11	• 100	
•	• A	• D PLNO	• Plant No.	•X	• L	•	• 2	• 2	• 100	
•	• B	• D DENO	• Department No.	•X	• L	•	• 2	• 2	• 100	
•	• C	• D STNO	• Station No.	•X	• L	•	• 2	• 2	• 100	
•	• D	• D SHNO	• Shift No.	•X	• L	•	• 1	• 1	• 100	
•	• E	• D MANO	• Man No.	•X	• L	•	• 4	• 4	• 100	>
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	
•	•	• D	•	•	•	•	•	•	•	

TOTAL CHARACTERS OF INFORMATION

ADDED CONTROL CHARACTERS

TOTAL CHARACTERS

III. THE PROGRAM SHEET AND ITS GENERAL USAGE

An RCA 501 Automatic Assembly Program Sheet, form RCA IE-240 is provided for writing programs to be assembled by the Assembly System. A sample coding sheet is shown at the end of this chapter.

HEADING

The heading of this sheet provides space for the user to enter the title of the project, the programmer's name, the date, and a page number. Entries in Title, Programmer, Date and page number may be made in any way the user desires.

GENERAL METHODS OF WRITING PSEUDO-CODE

Pseudo-code is written in the body of the form. General use of the columns on this form is as follows:

Instruction Number

Numbers, indicating the relative order in which instructions are to be placed in memory, are entered in this column.

Instruction numbers are composed in the following format:

PAGdd

where: PAG is three alphabetic characters, the first character of which is always "P".

dd is a two-digit numeric specifying the sequence of this instruction relative to the other instructions within this block of pseudo-coding. It is advisable to make "dd" assignments in multiples of ten.

Instruction numbers are written in ascending sequence in consecutive order; however, each line on the form need not contain an instruction number, in which case the Assembly System will assign a relative instruction number as explained in Chapter IV.

For example, instruction numbers may be entered as follows:

```
PABØØ
______
______
PAB 1Ø
______
______
______
PAB 2Ø
______
______
______
______
PAC 1Ø
```

An Instruction number having "PAGdd" format is often referred to as being an "explicit P-address." This terminology will be used throughout the remainder of this manual. An "explicit P-address" *must* appear in the instruction number column of the first line on each page of pseudo-coding, except where the first line is the continuation of a pseudo-code instruction from the previous page. In this case, an explicit P-address must appear for the first instruction following the continued one.

When the program is initially written, or when corrections are to be applied to an initial assembly, only explicit P-addresses or addresses using decimal point insertion may appear in the instruction number column. Decimal point insertion is explained in detail in Chapter IV.

Certain pseudo-code instructions require that an explicit P-address appear in the instruction number column. These will be so specified as they are discussed.

Except for these special cases, the user enters instruction numbers at his own discretion.

Comments

Under "Comments", the user may enter any brief descriptive remarks he wishes to make for each pseudo-instruction. Any EMP printing characters may be used. Control symbols, themselves, must not be used, but may be specified by the letters "SM", "EM", "ED", or "EF".

An entry in the Comments column may never exceed 44 punchable characters.

The comments column of the first P-address of the sorted pseudo-coding should contain the name of the program and the date that the program was written. This information, when provided, must be a fixed field containing exactly forty-four characters in the following format:

(Positions)	1 – 35	36	37 – 44
	NAME	sp	DD/MM/YY

OP.

Under "OP", the user specifies the operation code of the pseudo-instruction. Operation codes may be mnemonic 501 operation codes, machine operation codes, special mnemonic operation codes, Descriptor Verbs, macro-instructions, or subroutines. The latter three categories will be discussed in detail in later chapters.

Mnemonic codes available are shown in the following chart with their corresponding octal machine codes.

Mnemonic	*Octal Notation*	*Instruction Name*
PES	01	Programmed Error Stop
PR	02	Print
PA	03	Paper Advance
LRR	04	Linear Read Reverse
BRR	05	Block Read Reverse
UNS	06	Unwind n Symbols
TCW	10	Transcribing Card Punch Write
SSW	11	Single Sector Write
LW	12	Linear Write
MSW	13	Multiple Sector Write
LRF	14	Linear Read Forward
BRF	15	Block Read Forward
RNS	16	Rewind n Symbols
RWD	17	Rewind to BTC
IT	21	Item Transfer
OCT	22	One Character Transfer
STC	24	Sector Transfer by Character
TCT	25	Three Character Transfer
STT	26	Sector Transfer by Tetrad
RD	27	Random Distribute
LNS	31	Locate *n*th Symbol in Sector
ZS	32	Zero Suppress
JR	33	Justify Right
SCC	34	Sector Clear by Character
SCR	35	Sector Compress Retain Redundant ISS's
SCT	36	Sector Clear by Tetrad

(Continued on following page)

Mnemonic	*Octal Notation*	*Instruction Name*
SCD	37	Sector Compress Delete Redundant ISS's
BA	41	Binary Add
BS	42	Binary Subtract
SC	43	Sector Compare
TCA	44	Three-Character Add
TCS	45	Three-Character Subtract
LO	46	Logical "or"
LA	47	Logical "and"
DA	51	Decimal Add
DS	52	Decimal Subtract
DM	53	Decimal Multiply
DD	54	Decimal Divide
CTC	61	Conditional Transfer of Control
SSM	62	Sense Simultaneous Mode
TS	63	Tape Sense
SSG	65	Sense Simultaneous Gate
TA	66	Tally
TC	71	Transfer Control
SET	72	Set Register
STR	73	Store Register
CSG	75	Control Simultaneous Gate
ST	76	Stop
RAI	77	Return After Interrupt

Most of the mnemonic instructions listed in the above chart perform exactly the same functions in a similar manner, and are written in similar formats to that described in the RCA 501 Electronic Data Processing System Programmers' Reference Manual. The exceptions to this statement are described in a later chapter.

Special Mnemonic Operation Codes

The Assembly System also permits the use of four other mnemonic operation codes which have no counter-parts in machine code language. These are:

IGN – Ignore
ADV – Add Variable
RES – Reserve
DUP – Duplicate

The functions and format of these mnemonic operation codes are described in Chapter V.

Mnemonic Operation Codes

The Assembly System is designed to be used primarily with mnemonic operation codes. If mnemonic operation codes are used, all of the options available with the Assembly System may be used.

Machine Operation Codes

If machine operation codes are used, the octal operation code notation must be preceded by an "M" or "G".

When an M operation code is used, the *entire instruction* must be written in *machine code,* in exact machine code format. When a "G" operation code is used, the A and B columns may contain symbolic addresses and/or M-addresses. No entries may be made in columns to the right of the "B-Address" column, and none of the various options generally afforded to the Assembly System user can be used.

Thus, to perform a Conditional Transfer of Control,

CTC
M61
or
G61

may be written in the "OP:' column.

A Address

If a mnemonic or "G" operation code is used, the user may specify either a relative instruction address or an actual octal machine address under "A-address". The various addresses and addressing systems will be described in detail in the chapter devoted to addressing. In all but the exceptional cases (to be discussed in Chapter VI), the A-address entry will perform the same function in mnemonic instructions as in machine instructions.

If an M operation code is specified, the A-address must be an actual machine address written in format specified in the Programmers' Reference Manual, and will perform the same function as described in that manual.

N_A

An entry in "N_A" will specify the particular address modifier whose contents will be added to the A-address when the instruction is executed in the object program.

If a mnemonic or "G" operation code is used, a symbolic notation (see Chapter VI) is used to specify the address modifier location; if no modification is desired, N_A may be left blank.

If an M operation code is used, octal notation must be used to specify the address modifier location; if no modification of the A-address is desired, an octal zero must be entered under "N_A".

N_B

An entry in "N_B" will specify the address modifier whose contents will be added to the B-address when the instruction is executed in the object program.

If a mnemonic or "G" operation code is used, a symbolic notation is used to specify the address modifier location; if no modification is desired, N_B is left blank.

If an M operation code is used, octal notation must be used to specify the address modifier location; if no modification of the B-address is desired, an octal zero must be entered under "N_B".

B Address

If a mnemonic or "G" operation code is used, the user may specify either a relative instruction address, an actual octal machine address, or a symbolic address under "B-address". In all but the exceptional cases, the B-address entry will perform the same function in mnemonic instructions as in machine instructions.

If an M operation code is specified, the B-address must be an actual machine address, written in the format specified in the Programmers' Reference Manual, and will perform the same function as described in that manual.

T Address

The "T-address" column and all other columns to the right may only be used with mnemonic operation codes. In general, an entry in the T-address column will cause the Assembly System to generate a "Set T" instruction and place it before the instruction specified in the operation code. For example, let us look at how a T-address entry may be used with the Decimal Multiply. The instruction may be written as follows:

OP	*A*	$N_A N_B$	*B*	*T*
DM	WPAY		WRATE	WTAX

where: WPAY = a symbolic address specifying the rightmost character of multiplicand

WRATE = a symbolic address specifying the rightmost character of the multiplier

WTAX = a symbolic address specifying the destination location of the sign of the product.

The Assembly System will generate the following machine instructions as a result of this one pseudo-instruction:

OP	*A*	N_AN_B	*B*
72	WTAX	00	600000
53	WPAY	00	WRATE

Notice that this option (the T-Address column on the form) allows the user to write what is, in effect, a three address instruction. In the Decimal Multiply instruction shown above, the A-address specifies the multiplicand, the B-address specifies the multiplier, the T-address specifies the product location.

N_T

An entry in "N_T" will usually specify the address modifier whose contents will be added to the A-address of the generated "Set T" instruction, when the instruction is executed in the object program.

A symbolic notation is used to specify the address modifier; if no modification is desired, N_T is left blank.

For example, if the T-address of the previous example is to be modified by the contents of address modifier 1, the pseudo-instruction is written as follows:

OP	*A*	N_AN_B	*B*	*T*	N_T
DM	WPAY		WRATE	WTAX	B1

The machine instructions generated would be:

OP	*A*	N_AN_B	*B*
72	WTAX	10	600000
53	WPAY	00	WRATE

Note that if *any entry* is made in either the T-address or N_T columns, except for the special cases to be discussed in Chapter VI, a "Set T" instruction will be generated preceding the instruction specified under "OP".

If the T-address is left blank, but an entry is made in the N_T column, a "Set T" instruction with an A-address of $(000000)_8$ is generated preceding the instruction specified under "OP".

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T
DM	WPAY		WRATE		B1

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72 53	000000 WPAY	10 00	600000 WRATE

If a zero is entered in the N_T column, and the T-address column is left blank, a "Set T" instruction, with an A-address of $(000000)_8$ and a N octal digit of 0, is generated preceding the instruction specified under "OP ".

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T
DM	WPAY		WRATE		0

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72 53	000000 WPAY	00 00	600000 WRATE

These options provide the user with the ability to generate "Set T" instructions which may be set to any A-address with any A-address modifier by later programming steps.

CSG

The CSG column stands for "Control Simultaneous Gate". If a Control Simultaneous Gate instruction is desired after the instruction specified in the "OP." column, an entry is made under CSG. If the letter "O" is entered, the generated Control Simultaneous Gate instruction will open the simultaneous gate. If the letter "C" is entered, the generated Control Simultaneous Gate instruction will close the gate. The Control Simultaneous Gate instruction, if called for, will always immediately follow the instruction specified in the "OP." column, whether or not a T-address entry is made.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C S G*
DM	WPAY		WRATE	WTAX	B1	C

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72 53 75	WTAX WPAY 000000	10 00 00	600000 WRATE 010000

IF – GO TO

The right hand nine columns are used to specify conditional or unconditional transfer of control instructions. Three conditions may be specified in the three sets of "IF – GO TO" columns. The "IF" column is used to specify the condition; the "GO TO" column specifies the address of the instruction to which control will be transferred if the condition stated in the previous "IF" column prevails.

Either one or two of the following instructions will be generated as a result of entries in the "IF – GO TO" columns.

TC
CTC
TS
SSG
SSM

The instructions generated will be placed immediately following the CSG instruction, if one was called for, or if no CSG was called for, immediately following the instruction specified in the "OP." column.

TC

A single Transfer of Control instruction may be specified by:

1. Entering TC in the first "IF" column, and entering in the first "GO TO" column the relative or machine instruction address of the instruction to which control is to be transferred if no breakpoint bits are to be set to one.

2. If any breakpoint bits are to be set to one, enter "SW" followed by numbers of the breakpoint bits which are to be set to one in the first "IF" column. In the first "GO TO" column, enter the relative or machine address of the instruction to which control is to be transferred. SW3 would cause a test of breakpoint 3, SW025 would cause a test of breakpoints 0, 2 and 5.

Example:

Pseudo-code:

OP	*A*	$N_A N_B$	*B*	*T*	N_T	*CSG*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
DM	DPAY		DTAX	DNET			TC	PAB 25							

Generated Machine Code:

OP	*A*	$N_A N_B$	*B*
72	DNET	00	600000
53	DPAY	00	DTAX
71	PAB25	00	000000

CTC

The insertion of a Conditional Transfer of Control instruction may be specified by the use of two (or three) of the "IF – GO TO" columns in accordance with the following:

1. In one of the first two "IF" columns enter a "+". In the "GO TO" column to the right of the "+", enter the relative or machine instruction address of the instruction to which control is to be transferred if PRP is set.

2. In the other of the first two "IF" columns enter a " – ".

In the "GO TO" column to the right of the "–", enter the relative or machine instruction address of the instruction to which control is to be transferred if PRN is set.

3. If PRZ is set, control is transferred to the next instruction. If, however, the user wants a TC instruction to be inserted as the next instruction, the third set of "IF – GO TO" columns is used in the following manner:

   a. If no breakpoint bits in the inserted TC instruction are to be set to one, enter "0" in the third "IF" column, and in the third "GO TO" column, enter the relative or machine instruction address of the instruction to which control is to be transferred.

   b. If any breakpoint bits of the inserted TC instruction are to be set to one, enter in the third "IF" column an "SW" followed by the bit numbers which are to be set to one. In the third "GO TO" column, enter the relative instruction address to which control is to be transferred.

   c. When "+", "–" and "0" are used together, their order is immaterial. However, if "SW" is used it must appear in the 3rd column.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C* *S* *G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
SC	DCODE		DCODE	DMCODE			+	PAB 10		–	PAB 20		0	PAB 30	

where: DCODE = address of left-most character of the item, DCODE.

DCODE = address of right-most character of the same item.

DMCODE = address of right-most character of the comparison quantity, DMCODE.

PAB 10 = the address of the instruction to which control is to be transferred if PRP is set.

PAB 20 = the address of the instruction to which control is to be transferred if PRN is set.

PAB 30 = the address of the instruction to which control is to be transferred if PRZ is set.

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	DMCODE	00	600000
43	DCODE	00	DCODE
61	PAB 10	00	PAB 20
71	PAB 30	00	000000

TS

The Tape Sense instruction tests to see if any number of six possible tape conditions prevail. The 501 Assembly System user can specify the insertion of a Tape Sense instruction through the use of the first two "IF – GO TO" sets of columns in the following manner:

1. In the first "IF" column, enter letter codes indicating the condition, or combination of conditions, to be tested for. The following code letters are used:

B = Is the tape positioned on BTC?
E = Has ETW been sensed?
F = Is the tape now stationery or moving forward?
R = Is the tape now moving in the reverse direction?

(Cont'd)

M = Is the tape now in motion?
N = Is the Tape Station non-operable?

Any one or a combination of these code letters may be entered in the first "IF" column.

2. The relative or machine instruction address of the instruction to which control is to be transferred if any one of the conditions specified in the "IF" columns is found to be present, is entered in the first "GO TO" column.

3. A symbolic or machine Tape Station number (Tape Station Addressing is covered in Chapter V , Addressing) indicating the station to be tested is entered in the second "GO TO" column.

4. The third set of "IF – GO TO" columns may be left blank, or can be used to generate a transfer of control instruction – see TC.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C* *S* *G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
DM	DPAY		DTAX	DNET			RN	PAC43			IMAS				

where: IMAS = Tape Station 10

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	DNET	00	600000
53	DPAY	00	DTAX
63	PAC43	00	105000

An entry in N_1, where a Tape Sense instruction is called for, will cause address modification of the A-address of the generated machine instruction.

An entry in N_2, where a Tape Sense instruction is called for, will cause address modification of the B-address of the generated machine instruction. The B-address of the generated machine instruction will be generated from the entry made in the second "GO TO" column (the Tape Station) and from the entry in the first "IF" column (the condition to be tested). An N_2 entry in this case may, therefore, effect a change in these two specifications.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C* *S* *G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
DM	WPAY		WRATE	WTAX			B	PAB40	B1		T10	B2			

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	WTAX	00	600000
53	WPAY	00	WRATE
63	PAB40	12	100100

SSG

Two conditions are tested by the Sense Simultaneous Gate instruction. Is the simultaneous gate opened, or is the simultaneous gate closed? To generate an SSG instruction, "SGO" is entered in one of the first two "IF" columns, and SGC is entered in the other of the first two "IF" columns. The order in which they are entered is immaterial. The third "IF – GO TO" set of columns may be left blank, or can be used to generate a TC instruction.

In the "GO TO" column to the right of the "IF" column in which SGC was entered, enter the relative or machine instruction address of the instruction to which control is to be transferred if the simultaneous gate is found to be closed.

In the "GO TO" column to the right of the "IF" column in which SGO was entered, enter the relative or machine instruction address of the instruction to which control is to be transferred if the simultaneous gate is found to be opened.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C* *S* *G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
DM	WPAY		WRATE	WTAX			SGO	PAB 10		SGC	PAB 20				

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	WTAX	00	600000
53	WPAY	00	WRATE
65	PAB 10	00	PAB 20

SSM

The Sense Simultaneous Mode Instruction tests for one of four conditions:

1. Is the simultaneous mode occupied by a read instruction?
2. Is the simultaneous mode occupied by a write instruction?
3. Is the simultaneous mode occupied by a Paper Advance?
4. Is the simultaneous mode unoccupied?

A Sense Simultaneous Mode instruction which makes special provisions for all but condition 3 (i.e., the Paper Advance is occupying the simultaneous mode) may be entered in the "IF – GO TO" columns in the following manner:

1. Enter "SMR" (simultaneous mode occupied by read) in one of the first two "IF" columns.

   In the "GO TO" column to the right of the SMR entry, enter the relative or machine instruction address of the instruction to which control is to be transferred if the SSM finds a read instruction in the simultaneous mode.

2. Enter "SMW" (simultaneous mode occupied by write) in the other of the first two "IF" columns. The order is immaterial.

   In the "GO TO" column to the right of the SMW entry, enter the relative instruction address of the instruction to which control is to be transferred if the SSM finds a write instruction in the simultaneous mode.

3. If the simultaneous mode is found to be unoccupied, control will normally be transferred to the next instruction. If this is desired, leave the third set of "IF – GO TO" columns blank.

4. If a transfer of control to some instruction other than the next instruction is desired, when the simultaneous mode is found to be unoccupied, a TC instruction may be inserted as the next generated instruction through the use of the third set of "IF – GO TO" columns. Enter either "SW" followed by from one to six digits or "TC" in the third "IF" column according to the following:

   a. If no breakpoint bits in the inserted TC instruction are to be set to one, enter a "TC" in the "IF" column.

   b. If any breakpoint bits are to be set to one, enter "SW" followed by the numbers, from zero through five, of the bits that are to be set to one. Any number of the bits may be set to one; e.g., SW2, SW4, SW245, are all legitimate entries.

5. If a "TC" or "SW" entry has been made in the third "IF" column, enter, in the third "GO TO" column, the relative or machine instruction address of the instruction to which control is to be transferred by the generated TC instruction.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C S G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
DM	WPAY		WRATE	WTAX			SMR	PAB10		SMW	PAB20		SW234	PAB 30	

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	WTAX	00	600000
53	WPAY	00	WRATE
62	PAB 10	00	PAB 20
71	PAB 30	00	340000

N_1, N_2, N_3

Entries in the N_1, N_2, and N_3 columns, in all cases but the Tape Sense instruction, specify the address modifier locations whose contents are to be used to modify the "GO TO" addresses immediately to the left of the particular "N" column in which an entry is made.

Example:

Pseudo-code:

OP	*A*	N_AN_B	*B*	*T*	N_T	*C S G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
SC	DPAY		DPAY	DTAX	B1		+	PAB22	B2	–	PAB 25	B2	0	PAM 19	B3

Generated Machine Code:

OP	*A*	N_AN_B	*B*
72	DTAX	10	600000
43	DPAY	00	DPAY
61	PAB 22	22	PAB 25
71	PAM 19	30	000000

ADDRESS LENGTHS

Entries in the A-, B-, and T-address columns may not exceed nineteen characters in length.

THE "N" ADDRESS IN GO TO COLUMNS

The Assembly System will recognize an N entered in a "GO TO" column on the coding sheet. It will substitute for the N the address of the next pseudo-instruction. Note that this feature *cannot* be used in other columns.

GO TO COLUMNS LEFT BLANK

If the programmer has made an entry in an "IF" column and has left its corresponding "GO TO" column blank, the assembler will substitute a $(000000)_8$ address in the generated machine instruction.

RCA 501 AUTOMATIC ASSEMBLY PROGRAM SHEET

TITLE

PROGRAMMER

DATE

PAGE

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
INSTRUCTION NUMBER	COMMENTS	OP.	A ADDRESS	N_A	N_B	B ADDRESS	T ADDRESS	N_T	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

IV. ADDRESSING

Various methods and systems for addressing instructions, data, constants, working storage areas and address modifier locations are available to the RCA 501 Assembly System user. This chapter describes the available systems and their usage.

MACHINE ADDRESSES

Machine addresses may be used to address all five types of information:

1. Instructions,
2. Data,
3. Constants,
4. Working Storage Areas,
5. Address Modifiers.

Format

When used with mnemonic operation codes, machine addresses are written as six-digit octal numbers preceded by the letter "M". They may be entered in the "A-address", "B-address", "T-address" and "GO TO" columns of the coding sheet. If a mnemonic or "G" operation code is used for a particular pseudo-instruction, any combination of addresses of that pseudo-instruction may be machine addresses; however, each machine address used must be preceded by an "M".

If the octal operation code is preceded by an "M", all other entries (A, N_A, N_B, B) must be machine addresses, *not* preceded by an "M".

Example:

Pseudo-Code: *Machine Code:*

OP	*A*	$N_A N_B$	*B*		*OP*	*A*	$N_A N_B$	*B*
IT	M002000		M002050	=	21	002000	00	002050
M21	002000	00	002050	=	21	002000	00	002050

Addresses preceded by an "M" are called "M-addresses". Note that M-addresses may be entered for all addresses except address modifier locations. If a mnemonic operation code is used, address modifier locations for that instruction must be specified symbolically. If an octal operation code (preceded by an "M") is used, address modifier locations for that instruction are specified octally and are not preceded by an "M".

DATA ADDRESSES

The data descriptions entered on forms RCA IE-241 and IE-241-1 are entered as input to the Assembly System. The information provided on these forms will be used by the system to allocate read-in areas in memory for the various input files to the run being assembled. The amount of memory allocated by the Assembly System for each file read-in area will be based upon the maximum message size as determined from the data sheets.

FORWARD READ DATA ADDRESSING

To cause data from a file to be read into its allocated read-in area, the file is addressed by its file name. The file name for each file is the name entered on the data sheet describing that file (an F followed by a maximum of five characters).

Example:

To effect a Linear Read Forward of the next message on the Master Employee File tape from Tape Station 10 into the allocated read-in area, the Assembly System user writes:

OP	*A*	$N_A N_B$	*B*
LRF	FMASEM		T10

If the Assembly System has allocated memory locations 037000 through 037143 as the read-in area of this file, the generated machine code will be:

OP	*A*	$N_A N_B$	*B*
14	037000	00	100000

Notice that the file name, FMASEM, corresponds to the memory location into which the SM of the message will be placed (037000).

The following general statement may, therefore, be made:

If data is read into memory with a Linear Read Forward instruction, the symbolic file name, when placed in an address calling for an LHE specification, will address the memory location into which the SM is placed by the read instruction. The LHE of the read-in area will always be a C0 position to properly accommodate the SM. The RHE of the file area will always be the C3 position of the last tetrad in that area. It will *not* necessarily be the last character read in.

Symbolic Addressing

If a message is read in with a Forward Read Instruction, FAA (fixed and always appearing) items and sub-items in the read-in area may be symbolically addressed by item names. Item names are those names entered under "Abbreviation" on the data sheets. They are a maximum of five characters preceded by a "D".

For instance, the Employee Number item of a Master Employee file can be addressed as DEMN$\underline{O}$. When items are so addressed, the Assembly System will insert the address of either the right or left hand end of the item in the generated instruction according to the requirements of the particular instruction.

Example:

Assume that the first two items of every message in a Master Employee file are Employee Number, abbreviated DEMN$\underline{O}$, always fixed at eleven digits preceded by an ISS and Social Security Number, abbreviated DSSN$\underline{O}$, always fixed at nine digits preceded by an ISS.

The message is read into memory with a Linear Read Forward Instruction and the SM of the message is placed in memory location 037000.

A Sector Transfer of these two items from the read-in area may be written as follows:

OP	*A*	$N_A N_B$	*B*	*T*
STC	DEMN$\underline{O}$		DSSN$\underline{O}$	WBAL

where WBAL is the destination address.

Generated Machine Code:

OP	*A*	$N_A N_B$	*B*
72	041510	00	600000
24	037001	00	037026

where 037001 is the left hand end of the Employee Number item, its ISS; 037026 is the right hand end of the Social Security Number item; and 041510 is the right hand end of the work area, WBAL.

Character Addressing

Particular characters of any FAA item or sub-item are addressable by use of the following format:

$$\text{NAME}\begin{pmatrix} R \\ L \end{pmatrix} \text{ or NAME} \begin{pmatrix} R \pm n \\ L \pm n \end{pmatrix}$$

where: NAME is the address of the reference item or sub-item.
R specifies the right hand end of the item, the sign if present,
L specifies the left hand end of the item, the ISS if one is present,
+ specifies that the addressed character is n places to the right.
– specifies that the addressed character is n places to the left.
n is the number of places that the addressed character is removed from the specified end.

Note that character addressing must be enclosed by parentheses.

Examples:

Assume that the FAA item DNAME is 10 characters in length, including an ISS. The following references may be made:

DNAME (R) = addresses rightmost character of field.
DNAME (L+2) = addresses *third* character in field.
DNAME (R–4) = addresses *sixth* character in field.

Characters of FAA items within a message read in with a forward read may also be addressed relative to the LHE in the following format:

FILE NAME(L+n)

where: FILE NAME represents the address of the LHE, n is the number of characters to the right of the LHE.

Example:

FMASEM(L+10) addresses the 11th character of the message
FMASEM(L+25) addresses the 26th character of the message

Relative Addressing

Any FAA item or sub-item read into a file area with a forward read instruction may be addressed relative to the left hand end of the file area, or relative to other FAA items.

Relative to LHE of the file area:

Since the LHE of a file is addressed by the file name, and all FAA items are fixed with respect to the LHE, it is possible to address these items by their relative position to the LHE in the following manner:

FILE NAME + k

Where k is the number of the addressed item within the file. Thus, in the Master Employee file where the Social Security Number is the second item, it may be addressed as follows:

FMASEM + 2

The second sub-item within Social Security Number may be addressed:

FMASEM + 2B

Relative to Other FAA Items:

FAA items within a message read into memory with a forward read instruction may also be addressed by their relative position to other FAA items. FAA item addresses which are relative to other FAA items are written in the following format:

ITEM NAME ± k

where: k is the number of items the addressed item is removed from the reference item.
+ means to the right of the reference item.
– means to the left of the reference item.

Thus, if the Stock Number is the second item and Date is the sixth item, Date can be addressed as follows:

DSTKNO + 4

The first sub-item within Date may be addressed:

DSTKNO + 4A

Character addressing may also be used with relative addresses; for example:

FMASEM + 6 (L + 3)

This will address the fourth character of the sixth item of the file FMASEM.

As with symbolic addressing, the Assembly System will supply the address of the right or left hand end of the relative item according to the requirements of the instruction in which it is used. *It should be noted that an ISS (if present) is considered as the left hand end; a sign (if present) is considered as the right-hand end.*

ISS Considered As LHE

When an item contains an ISS, the ISS location is considered as the left-hand end of the item. Therefore, if DITEM is used in an address requiring the LHE, the Assembler will supply the location of the ISS. Similarly, DITEM(L) would address the same location.

If an item contains an ISS, the LHE of its *first* sub-item will be the ISS location, also.

Alternate Read-In Area Addressing

Since the RCA 501 Data Processing System has simultaneous operation features, the Assembly System provides for up to four possible read-in areas to facilitate alternate read programming.

1. The "F" read-in area
2. The "X" read-in area
3. The "Y" read-in area
4. The "Z" read-in area

The primary, or "F", read-in area is addressed by the methods previously discussed.

To read a message into the second read-in area, the file name preceded by an "X" is used as the A-address of the Linear Read Forward instruction. FAA items, sub-items and characters may be addressed symbolically and relatively within the "X" read-in area in the same format described for the "F" read-in area except that file names and item names are prefixed with X's.

Similarly, data is read into the Y and Z read-in areas by prefixing the file name in the A-address of the LRF with the letter corresponding to the read-in area desired. Data is addressed within these read-in areas by referring to file names and item names prefixed with Y or Z (depending upon the read-in area being used).

All addressing procedures previously described also apply to the alternate areas.

Examples:

The following Linear Read Forward instructions will read successive messages from the same file into the four different read-in areas.

OP	*A*	N_AN_B	*B*
LRF	FMASEM		T10
LRF	XFMASEM		T10
LRF	YFMASEM		T10
LRF	ZFMASEM		T10

The following Item Transfer instructions transfer the Employee Number item from the four different read-in areas to four work areas.

OP	*A*	N_AN_B	*B*
IT	DEMNO		WAREA 1
IT	XDEMNO		WAREA 2
IT	YDEMNO		WAREA 3
IT	ZDEMNO		WAREA 4

Use of Data Addressing

Note that the symbolic and relative addressing systems described herein apply to data in original read-in areas only. Once data is removed from its original read-in area, it is no longer addressable by these methods. W-addresses, to be described under Working Storage Addressing, are assigned to data transferred from read-in areas to other areas of memory.

Addressing Non-FAA Items

Two systems may be used to address non-FAA items:

1. The message may be transferred into working storage by use of the Random Distribute instruction. The items in working storage may then be addressed symbolically by working storage addresses. A complete description of the use of working storage addressing is given in this chapter under Working Storage Addresses.

2. The Locate *n*th Symbol instruction may be used to obtain the machine address of any desired item. The address thus determined may be used in subsequent instructions.

Any attempt to make direct symbolic or relative reference to a non-FAA item in the read-in area will cause the Assembly System to leave that address blank, and an error print-out will indicate this occurrence.

REVERSE READ DATA ADDRESSING

If files are read into memory with reverse read instructions, several options are available to the programmer for addressing data within the file.

Reverse Reads of Complete FAA Files

If a file is composed of all FAA items, *and the RHE falls in the C3 position of a tetrad on a forward read,* all the rules of data addressing applying to data read with a forward read, would apply to this file if it were read with a reverse read.

Reverse Reads of Data Containing Non-FAA Items

Data read with reverse reads will be placed in memory such that the RHE will always be in a C3 position of a tetrad. If the file contains non-FAA items, the LHE of the file (and all items designated as being FAA) will no longer be fixed. Thus, it will not be possible to address these items relative to the LHE. The Assembly System, therefore, makes the following options available to the user for addressing data read with a reverse read.

Character Relative with Respect To the RHE

Individual characters within a message read into memory with a reverse read may be addressed in the following format:

FILE NAME (R – n)

where: FILE NAME is the name of the file.
n is the number of characters to the left of the RHE.

The % Address

When a file is read into memory with a reverse read, the machine address of the last character read is left in the A Register. The contents of the A Register may then be stored in an address modifier location. This address modifier location can then be used to modify all addresses in subsequent instructions referring to FAA items. In addresses modified in this manner, the names of the addressed FAA items, preceded by a % symbol is entered; e.g., %DEMNO for Employee Number, %DSSNO for Social Security Number, etc.

The Assembly System will substitute, for all % addresses, the octal number of characters the LHE or RHE of the item is removed from the left-hand end of the file, depending upon the requirements of the instruction.

Example:

If the Employee Number item is the first item of a message, and is composed of eleven digits preceded by an ISS:

%DEMNO will be replaced by 000001 in instruction addresses requiring the LHE.

%DEMNO will be replaced by 000014 in instruction addresses requiring the RHE.

GENERAL WORKING STORAGE ADDRESSES

Working storage addresses are always written with a "W" as the first character of the address, and are therefore referred to as "W-addresses". There are two classes of W-addresses:

1. General Working Storage Addresses
2. Special Working Storage Addresses

General working storage addresses are written in the following format:

WANNNNN

where: W must be the first character of all W-addresses, A specifies the particular section of working storage memory into which the working storage specified by this address is to be placed. "A" may be any letter or number except 0, 1, D, F, X, Y, or Z. "A" equal to 1 refers to print working storage areas covered later in this chapter. "A" equal to D, F, X, Y or Z designates Distributed Data Working Storage areas created by the Random Distribute Instruction as explained in Chapter VI.

NNNNN may be any combination of five numbers or letters naming the particular location to which this address applies.

Length of General Working Storages

The length of all general working storages must be defined as part of a W-address once and only once within the program. Working storage lengths are specified by writing, after the W-address, a decimal point followed by a decimal number of up to four digits which specifies the number of character locations to be reserved for this working storage. Having once specified the length of a particular working storage, the user makes reference to that working storage by the W-address, alone, without the length designation.

Example:

Suppose that it is desired to multiply two items, place the product in working storage, and zero suppress the product. Further suppose that the multiplier is an FAA three-digit item named DMULTR, the multiplicand is an FAA four-digit item named DMULTD, and the product is to be placed in a working storage location in the "A" working storage block. The pseudo-coding to accomplish this may be written as follows:

OP	*A*	$N_A N_B$	*B*	*T*
SCC	WAPROD.8		WAPROD	
DM	DMULTD		DMULTR	WAPROD
ZS	WAPROD		WAPROD	

Note that in the above pseudo-code "WAPROD" is used as both the left and right hand address in the SCC and ZS instructions. The Assembly System will generate the left or right hand end of the working storage area from the W-address according to the requirements of the address of the particular instruction.

Note also that the length of the working storage area into which the product is placed is specified only once (in the above case in the A-address of the SCC instruction). Should the length be specified more than once, more than one memory allocation will be made; however, only one of the allocated memory areas will be consistently referred to by this W-address.

Positioning Working Storage Within Tetrads

In the above example, an SCC instruction was used to clear the product destination area. Notice, however, that eight character locations were reserved for the product destination area. If the programmer could be assured that the eight characters reserved by the Assembly System for "WAPROD" were two complete tetrads, rather than one full tetrad and two partials, he could use the faster SCT instruction rather than the SCC instruction to clear the product destination area.

The Assembly System, therefore, provides the user with the option of specifying which particular character within a tetrad is to be the left hand end of the working storage area. Left-hand-end-character-within-tetrad-designation is written in the following format:

W-ADDRESS.NC (Cn)

where: W-ADDRESS is the W-address the user has assigned to this working storage area.
.NC is the number of characters to be reserved for this working storage area preceded by a decimal point.
(Cn) is the character position within the tetrad which is to be the left hand end of the working storage area, where n may equal 0, 1, 2 or 3.

The (Cn) notation need only be specified if it is important that a working storage area be positioned properly with respect to tetrads in memory. If such positioning is important the (Cn) notation must be specified in the same W-address as the length specification. All other references to the same working storage areas are made by the W-address, alone.

Example:

With the use of this option, the previous example may be coded as follows:

OP	*A*	N_AN_B	*B*	*T*
SCT DM ZS	WAPROD.8(CO) DMULTD WAPROD		WAPROD DMULTR WAPROD	WAPROD

Character Addressing Within Working Storage

Individual characters within working storage areas may be addressed by following the W-address with (R ±n) or (L ±n) in a similar manner to character addressing within FAA items.

Note that a W-address referring to a particular character within a working storage must not be used in a W-address which also specifies the length of the working storage.

SPECIAL WORKING STORAGE ADDRESSES

There are two types of special working storage addresses:

1. Print working storage addresses.
2. Distributed data working storage addresses.

Print Working Storage Addresses (On-Line Printer)

The Assembly System makes special provisions for setting up working storages from which information is to be printed. Print working storages are addressed in the following format:

W1dd

where: W must be the first character of all working storage addresses.
1 indicates that this is the address of a print working storage.
dd is a two-digit number ranging from 01 to 99, inclusive. This is the number assigned to this particular print working storage.

Length and Position of Print Working Storages

The Assembly System automatically assigns print working storages. The programmer does not; he implies these specifications to the Assembly System when he writes a "1" as the second character of a W-address. However, to cause the Assembly System to act upon this implication, for each unique print working storage address, the programmer must place a decimal point after the address once within the program.

Example:

W101.

The RCA 501 Print instruction always prints the contents of 120 successive character locations in memory. Furthermore, this instruction requires that the address of the first of the 120 characters to be printed have an even number in its C2 digit and $(00)_8$ in its C3 digit. The Assembly System always reserves $(128)_{10}$ = $(200)_8$ character positions in memory for each unique print working storage addressed in the pseudo-code. Each of the reserved print working storages will begin in a memory location whose C2 digit is even and whose C3 digit is $(00)_8$.

Print Working Storage References

It was stated above that 128 character positions were reserved for each print working storage, but only 120 characters are printed with each print instruction. W1 addresses, therefore, when used in instructions where the right hand end of a sector is required, will generate the machine address corresponding to the 120th character of the print working storage rather than the 128th character. W1 addresses, when used in instructions where the left hand end of a sector is required, will generate the machine address corresponding to the first character set aside for the print working storage.

Example:

To clear a print working storage, designated as W105, the programmer may write:

OP	*A*	N_AN_B	*B*
SCT	W105.		W105

If W105 were assigned to memory locations 031200 to 031377 the generated machine code would be:

OP	*A*	N_AN_B	*B*
36	031200	00	031367

Character Addressing Within Print Working Storage

Characters within working storage areas may be addressed with (L ±n) or (R ±n) notation following the W1 address. "L" refers to the first character of the print working storage. "R" refers to the 120th character of the working storage. To refer to the normally unused eight characters (characters 121 through 128) of the 128 characters reserved by the Assembly System for print working storage areas, (L+120) to (L+127), or (R+1) to (R+7) may be used.

Example:

To address the 121st character of print working storage location W105, the following addresses may be used:

W105(R+1)
W105(L+120)

Note that the (R ±n) and (L ±n) notation must not be used in a W-address with a period in it.

Use of W1 Addresses with Print Instructions

Assume it is desired to print the following FAA items from the read-in area in the indicated print positions on one line.

Item	*Abbreviation*	*No. Digits*	*Print Positions*
Employee Number	DEMNO	11	5-15
Social Security No.	DSSNO	9	20-28
Date	DDATE	6	35-40
Yearly Salary	DYRSAL	4	45-48

The pseudo-code to accomplish this may be written as follows:

OP	*A*	N_AN_B	*B*
SCT	W103.		W103
IT	DEMNO		W103 (L + 14)
IT	DSSNO		W103 (L + 27)
IT	DDATE		W103 (L + 39)
IT	DYRSAL		W103 (L + 47)
PR	W103		

Distributed Data Working Storage Addresses

Special WD, WF, WX, WY and WZ working Storage areas are created by the Random Distribute ALL instructions. These distributed data working storage areas are covered in Chapter VI.

CONSTANT ADDRESSES

Constants may be referred to in two different ways:

1. Literals

2. Symbolic Addressing

Literals

The Assembly System permits the programmer to write constants within the A, B or T columns of instructions having mnemonic operation codes. These constants will be generated by the Assembler and assigned specific memory locations within the segment of the program in which they are used. The generated machine instruction will contain the address of the LHE or RHE of the constant, depending on the requirements of the instruction. When the constant, itself, is written in an address, the entry is called a literal. Two types of literals may be written, RCA 501 character literals and octal literals.

RCA 501 Character Literals

Literals which the programmer wishes to use as decimal or RCA 501 characters can be written as the RCA 501 characters; they need not be written as their octal equivalents. RCA 501 character literals are written enclosed with quotes.

Assume that a transaction file contains a one-character code of either an R (Receipt) or I (Issue). Assume also that this one-character code has been defined within the file description as having a data name "DCODE", and is preceded by an ISS. If it is desired to compare this code with a constant "R", the instruction may be written as follows:

Example:

OP	*A*	N_AN_B	*B*	*T*
SC	DCODE(L + 1)		DCODE	"R"

All RCA 501 characters may be specified in an RCA 501 character literal, except the SM, EM, ED, EF, and quote symbols. These characters must be specified in an octal literal (see below). The ISS when specified is an RCA 501 character literal is represented by an asterisk. The space is represented by "sp". Algebraic sign is represented by placing a plus (+) or minus (–) sign as the rightmost character of the literal.

Examples of RCA 501 Character Literals:

"*3146+"
"*00001 –"
"*RERUN sp TAPE sp A"
"CHECK sp TAPES"

Octal Literals

Constants which are to be used in binary arithmetic operations may be specified as octal literals. Octal literals are enclosed with the number sign (#). All entries between # signs are decoded such that each two characters of the literal become one 501 or two octal digits in memory. Therefore, octal entries *must* be composed of an even number of octal digits (digits between 0 and 7 inclusive). No RCA 501 characters, as such, may be written within # signs. Symbols such as SM, EM, ED, ", and letters may be represented by their octal equivalents.

Examples:

#73# = ED
#777777# = $(777777)_8$
#4455430163405744# = End Tape

Length of Literals

The length of the literal is specified by the literal itself. The literal is stored in memory exactly as written.

The maximum size of a literal in the A, B, or T columns is 19 punchable characters, which includes quotes, number signs, and all RCA 501 control symbols.

Unique and Non-Unique Literals

When a literal is specified in an address, the Assembly System stores the literal in some memory location. The address of this location is substituted by the Assembly System for the literal itself. Should the programmer again specify this literal (the same literal in the very same format is written in another address in the program), the Assembly System will not re-store the constant. Instead, the Assembly System will recognize the repeated literal as being exactly equal to an already stored constant. It will then substitute the address in which the constant was originally stored in the instruction in which the literal was repeated.

Example:

OP	*A*	N_AN_B	*B*	*T*
TCA	M000113		#000001#	
TCA	M000133		#000001#	
TCA	M000153		#000001#	

If the #000001# literal is stored in memory locations 002001 through 002003, the generated machine code would be:

OP	*A*	N_AN_B	*B*
44	000113	00	002003
44	000133	00	002003
44	000153	00	002003

This elimination of duplicated literals is a memory space saving feature of the Assembly System; however, there will be occasions when the programmer will want to specify the same constant more than once in his program, but want the constant to be stored in different memory locations each time it is specified.

For instance, suppose two different tallies were to be set to $(777777)_8$ initially. If #777777#, alone, were written in the B-address of both TA instructions, both of these instructions would decrease the same quantity. But two unique tally quantities are desired.

To specify unique memory locations for repeated constants, the programmer writes a "U" followed by a one- or two-digit decimal number. The number specifies to which particular constant of the set of alike, but unique, constants reference is being made.

Example:

If two different tallies are to be kept, each initially set to 777777, the following pseudo-code may be written:

OP	*A*	N_AN_B	*B*
TA	PAM10		#777777#U1
TA	PAB20		#777777#U2

where: PAM10 is the instruction address of the instruction to which control is to be transferred if the first TA instruction does not find its unique tally to be exhausted.
PAB20 is the instruction address of the instruction to which control is to be transferred if the second TA instruction does not find its unique tally to be exhausted.

Fixed and Non-Fixed Literals

The Assembly System user will have the option of segmenting his program; i.e., he may choose to read one segment of coding into memory, operate from it, read another segment into the same memory area, operate from it, and so on. Each succeeding segment of coding may be read into the same memory area, erasing the previous segment.

Segmentation will be accomplished by Descriptor Verbs, to be covered in a later chapter.

Literals will normally be considered to be part of the segment in which they are specified. Therefore, constants generated by literals in one segment will normally be destroyed when another segment is read into memory.

Should the programmer wish to place a constant in a fixed constant pool, so that the constant is available to *all* coding segments, he may do so by writing the letter "F" after the literal, or if the literal is also to be unique, after the "U" designation. Literals followed by an "F" are called "fixed constants". Having once designated a literal as a fixed constant, the programmer must *always* refer to it with the suffixed "F".

Examples of Fixed Constants

"*06"F

"*06+"U3F

#000010#U1F

#000010#U7F

#000001#F

Character Position Within Tetrad

Just as it may sometimes be important for the programmer to be able to specify the position within tetrad for working storages, so it is just as important to have the same option with constants. Therefore, any literal may be followed by a (Cn) notation in parentheses, where "n" is the number (0, 1, 2, or 3) of the character position within a tetrad in which the left hand end of the literal is to be stored.

If the literal has no other notation associated with it, the (Cn) notation immediately follows the literal.

If the literal has a "U" notation, the "U" notation follows the literal, and the (Cn) notation follows the "U" notation.

If the literal has an "F" notation, the "F" follows the "U", if any, and the (Cn) follows the "F".

Examples:

```
#000001#(C0)
#000010#U2(C1)
#001000#U1F(C2)
#000100#F(C3)
```

Having once specified a (Cn) notation for a literal, the programmer should not repeat the (Cn) notation again. Further references to the same literal are written without the (Cn) notation. Specifying the same literal with a different (Cn) notation will have no effect. The constant will be considered to be oriented according to the first (Cn) designation.

Symbolic Addressing of Constants

Constants may be referred to by addressing them with "K-names". K-names are symbolic addresses, composed by the programmer in the following format.

KNNNNN

where: K must be the first character of all symbolic constant names.
NNNNN may be any combination of up to five alphabetic or numeric characters. It is advisable to make NNNNN a mnemonic representation for this constant.

Example:

In order to calculate the withholding tax to be deducted from an employee's salary, his taxable earnings are multiplied by the current tax rate. His taxable earnings were previously calculated and stored in a working storage addressed by WPAY.The current tax rate is a constant which may be addressed by a K-name. If the programmer designates KRATE as the symbolic address for the current tax rate, the multiplication operation may be specified as follows:

OP	*A*	N_AN_B	*B*	*T*
DM	WPAY		KRATE	WTAX.8

Note that the product of this multiplication is the withholding tax deduction and is stored in a location specified by WTAX.

Constant Specification

In the above example, the programmer merely specified an address for the constant. This is the symbolic address of the memory location in which this constant is to be stored. What the constant actually is must also be specified. This is done through the use of a special Descriptor Verb called the Define Constant Verb which is described below.

Should the programmer wish to refer to the same constant more than once in the program he may use the same K-name each time he refers to that constant; however, he need define the contents of that K-name, by a Define Constant Descriptor Verb, only once.

THE DEFINE CONSTANT DESCRIPTOR VERB

Format:

Inst. No.	*OP*	*A*	*B*
DPxxxx	DEFK	constant	K-NAME

where: DPxxxx is a Descriptor Verb instruction number. Each DEFK verb must have a unique DP-address.

DEFK specifies the operation code for this Descriptor Verb.

K-NAME is the symbolic name assigned to the constant appearing in the A-address – where K is the first character, followed by up to five alphanumeric characters.

Example:

Assume that the symbolic address KRATE has been used in the program to represent a tax rate. Further, the actual tax rate is defined to be 18%. When this number becomes available to the programmer the following instruction would be written:

Inst. No.	*OP*	*A*	*B*
DPAR23	DEFK	"* 18+"	KRATE

Note that when the A column is used the Comments column may be freely used for descriptive remarks.

When using the A column of a DEFK instruction, the maximum number of characters (including control symbols) is 19. If space in the A column is not sufficient, the A column may be left blank and the constant may be defined in the comments column. In this case, the constant may then contain up to 44 characters. If more than 44 characters are required, the entry in the comments column may be continued on succeeding lines provided all other columns on these lines are left blank. In no event, however, can the maximum size of the constant (including control symbols) exceed 320.

The following option is available when a constant is defined *by a DEFK Descriptor Verb.* If the first character following the quote is a slant (/), it will be converted to a SM symbol; if the last character within quotes is a slant, it will be converted to an EM symbol. In all other cases, slant symbols will produce slant symbols in the constant.

Example:

Assume that the symbolic name KERROR has been used in the program to represent an error condition when the transaction input is improper:

Inst. No.	*COMMENTS*	*OP*	*A*	*B*
DPAB10	"/TRANSACTIONspINPUTspIMPROPER/"	DEFK		KERROR

All rules discussed previously with regard to specifying a literal as unique, fixed or non-fixed, and its position within a tetrad, also apply to the constant that is defined by the DEFK verb. For example:

Inst. No.	*COMMENTS*	*OP*	*A*	*B*
DPAR16		DEFK	"125" U1F	KVALUE

Character Addressing Within K-constants

Individual characters within a K-constant may be addressed relative to the right or left hand end in the following format:

$$\text{K-name}\begin{pmatrix} \text{R} \\ \text{L} \end{pmatrix} \qquad \text{or} \qquad \text{K-name}\begin{pmatrix} \text{R} \pm \text{n} \\ \text{L} \pm \text{n} \end{pmatrix}$$

It should be noted that character addressing can *not* be used with octal or decimal literals.

Assembly Rules Applying to Constants

The following rules govern the use of constants within an Assembly program:

Fixed:	Non-Fixed:
75 fixed literals	75 non-fixed literals per segment
40 fixed K-constants	40 non-fixed K-constants per segment

Note: See Assembly Check List, Appendix A, for total number of characters that may appear in the fixed and non-fixed constant blocks.

TAPE STATION ADDRESSES

Tape stations may be addressed in two ways:

1. Directly
2. Symbolically

Direct Tape Station Addressing

Tape stations may be directly addressed by their octal numbers preceded by a "T".

Example:

To cause a linear read forward from tape station 30 into read-in area YFMASEM, the programmer writes:

OP	*A*	$N_A N_B$	*B*
LRF	YFMASEM		T30

Symbolic Tape Station Addressing

In input-output instruction addresses, in which particular tape stations are to be specified, the programmer may write symbolic notations in the following format:

$$\begin{matrix} \text{I} \\ \text{or} \\ \underline{\text{O}} \end{matrix} \ \text{NNNNN}$$

where: $\begin{matrix} \text{I} \\ \text{or} \\ \underline{\text{O}} \end{matrix}$ = the first character, which must be either an "I" or an "$\underline{\text{O}}$". "I" is used to specify input tapes. "$\underline{\text{O}}$" is used to specify output tapes.

NNNNN = Any five character maximum mnemonic name the programmer wishes to use.

Each I or $\underline{\text{O}}$ symbolic name used will later be assigned an actual tape station number by the programmer through the use of the TAPE Descriptor Verb. Use of symbolic tape station addressing allows the programmer to defer tape station allocation until after the completion of the major body of pseudo-code. In addition, it is recommended that symbolic addressing be used throughout the program if the user wishes to take advantage of the tape switching feature of the RCA 501 Sequencer Routine.

The TAPE Descriptor Verb

The TAPE Descriptor Verb has the following format:

Inst. No.	*OP*	*A*	*B*
DPxxxx	TAPE	I NNNNN O	XX

where: TAPE is the operation code; the symbolic tape designation is specified by the A address; the actual tape number, written as two octal digits is specified by the B address. Note that this verb requires a DP-address.

Example:

Assume that a program uses a reference tape designated IREF, a transaction tape designated ITRANS and produces two output tapes designated OREF and OPRINT respectively. When tape station assignments became known they were 10, 06, 30, 05. The following instruction may be written in the program:

Inst. No.	*COMMENTS*	*OP*	*A*	*B*
DPAM13		TAPE	IREF	10
			ITRANS	06
			OREF	30
			OPRINT	05

Thus, the actual tape station numbers (10, 06, 30, 05) are associated with each of the tapes used in the program. Note that the tape station numbers written in the B column are *not* preceded by the letter "T". Also, "TAPE" need only appear in the OP code on the first line.

INSTRUCTION ADDRESSES

Instruction numbers are used by the Assembly System for two purposes:

1. To permit reference to other instructions or their parts.
2. To indicate the sequence in which instructions are to be placed in memory.

Instruction References

Several options are available to the Assembly System user for addressing instructions or parts of instructions.

Direct Symbolic Addressing of Instructions

Any pseudo-instruction to which an instruction number has been assigned may be directly addressed by that instruction number. As explained in Chapter III of this manual, "explicit P-address" instruction numbers are composed in the following format:

PAGdd

where: PAG is three alphabetic characters, the first character of which is always a "P".
dd is a two-digit number specifying the sequence of this instruction number relative to the other instruction numbers within that block of pseudo-coding.

Example:

IF	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
+	PAB10		–	PAB20		0	PAG60	

The above example shows a CTC instruction specified in the "IF – GO TO" columns of the coding sheet. In this example, if the CTC instruction generated finds PRP set, control will be transferred to the pseudo-instruction numbered PAB10; if the CTC instruction finds PRN set, control will be transferred to the pseudo-instruction numbered PAB20; if the CTC instruction finds PRZ set, control will be transferred to the pseudo-instruction numbered PAG60.

Relative Instruction Addressing

In the above example, each of the instructions referred to in the "GO TO" columns, was assigned an instruction number. But each instruction written need *not* be assigned an instruction number. How, then, does one refer to instructions which have no assigned instruction numbers?

This is accomplished by indicating the number of pseudo-instructions that the addressed instruction is removed from the last instruction that has an explicit P-number. The format of relative instruction addresses is as follows:

PAGdd+k

where: PAGdd is the number of the first pseudo-instruction, preceding the one addressed, which has been assigned an explicit P- instruction number.
k is the number of pseudo-instructions the addressed instruction is removed from the reference instruction. This number may range from 1 to 99.

Example:

The following sequence of instructions are written somewhere within the program:

Instruction Number	*OP*	*A*	$N_A N_B$	*B*
PAA30	IT TC IT TC IT	"*GREATER" PAA40 "*LESS" PAA40 "*EQUAL"		M002700 M002700 M002700

It is desired to perform a CTC instruction somewhere else in the program, and transfer "*GREATER" to memory location 002700 if the CTC instruction finds PRP set, "*LESS" if the CTC instruction finds PRN set, and "*EQUAL" if the CTC instruction finds PRZ set. The CTC instruction may be written in the "IF – GO TO" columns of the coding sheet as follows:

IF	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
+	PAA30		–	PAA30+2		0	PAA30+4	

When the instruction address notations described above (PAGdd or PAGdd+k) are used, the Assembly System generates the machine address of the operation code of the *first machine instruction generated*by the addressed pseudo-instruction.

Example:

The following DM instruction may appear in a program as shown:

Instruction Number	*OP*	*A*	$N_A N_B$	*B*	*T*
PAA30	DM	WATXER.6		"* 18+"	WATXDE.8

Later in the program a TC instruction may be written as follows:

Instruction Number	*OP*	*A*	N_AN_B	*B*
PAB50	TC	PAA30		

If the Assembly System makes the following address assignments:

Pseudo-address	*Assigned Machine Address*
PAA30	002000
WATXER	037000
WATXDE	037006
"*18+"	027000
PAB50	002700

The generated machine code for the DM instruction will be as follows:

Inst. No.	*OP*	*A*	N_AN_B	*B*
002000	72	037006	00	600000
002010	53	037000	00	027000

The generated TC instruction will appear as:

Inst. No.	*OP*	*A*	N_AN_B	*B*
002700	71	002000	00	000000

Note that the TC instruction transfers control to the generated SET (72) instruction rather than to the DM (53) instruction, since the machine address assigned to PAA30 becomes the address of the *first machine instruction* generated by the pseudo-instruction on that line.

Instruction Numbers Specify Sequence

As shown above, instruction numbers are used as tags, i.e., they are used in addressing to make references to other instructions, but instruction numbers also have another important purpose. They indicate to the Assembly System the order in which instructions are to be placed in memory. To illustrate how the Assembly System uses instruction numbers to sequence instructions, consider a sample instruction number column on the coding sheet. This column may appear as follows where lines represent pseudo-instructions left un-numbered by the programmer:

```
PAA10

______
______
______
______
______
______

PAA20

______
______
______
```

```
PAA30
______
PAA40
PAA50
______
______
______
______
______
______
______
PAA60
```

When the pseudo-code is read into the computer for assembly, early in the assembly process, the pseudo-instructions are sorted in ascending sequence, with DP-addresses appearing first, followed by the P-addresses.

Sorting is accomplished by first supplying relative instruction numbers to the un-numbered pseudo-instructions, following the same scheme that the programmer uses to address un-numbered instructions. Then both programmer assigned, and Assembly System-assigned instruction numbers are used as sorting keys for their corresponding instructions.

The list of instruction numbers shown above, will, therefore, appear as follows after the Assembly System assigns un-numbered instruction numbers.

```
PAA10
PAA10+1
PAA10+2
PAA10+3
PAA10+4
PAA10+5
PAA10+6
PAA20
PAA20+1
PAA20+2
PAA20+3
PAA30
PAA30+1
PAA40
PAA50
PAA50+1
PAA50+2
PAA50+3
PAA50+4
PAA50+5
PAA50+6
PAA50+7
PAA60
```

When sorted by the Assembly System, this group of instructions will remain in the same sequence since the programmer wrote them in ascending order. However, should the programmer write the following set of instruction numbers, the Assembly System will assign instruction numbers, and sort the instruction in sequence as indicated:

Instruction Nos. As Written	*Assembly System Assignments*	*Sorted Sequence*
PAA10	PAA10	PAA10
———	PAA10+1	PAA10+1
———	PAA10+2	PAA10+2
PAA30	PAA30	PAA20
———	PAA30+1	PAA20+1
———	PAA30+2	PAA30
———	PAA30+3	PAA30+1
PAA20	PAA20	PAA30+2
———	PAA20+1	PAA30+3
PAB10	PAB10	PAB10
———	PAB10+1	PAB10+1
———	PAB10+2	PAB10+2
———	PAB10+3	PAB10+3

This feature of the Assembly System can be used to excellent advantage by the programmer to insert forgotten or additional instructions into his program.

Insertions

Two methods of effecting insertions are available to the Assembly System user:

1. Explicit P-Instruction Insertions.
2. Decimal Point Insertions.

Explicit P-Instruction Numbers

Suppose the programmer has written the following instruction numbers:

PAA10
———
———
———
PAA20
———

He then wishes to insert, between the third and fourth pseudo-instructions, a set of additional pseudo-instructions. He may replace the fourth line with an explicit P-number which falls between PAA10 and PAA20 (i.e. PAA15). The instructions to be inserted may now be written anywhere in the program provided the inserted pseudo-instructions are written in proper sequence relative to each other, and the first of the inserted instructions is assigned an explicit P-instruction number.

Example:

It is desired to insert three pseudo-instructions between lines three and four of the indicated list of instruction numbers:

Original Coding	*Revised Coding*	*Assembly System Assignments*	*Assembly System Sort*
PAA10	PAA10	PAA10	PAA10
———	———	PAA10+1	PAA10+1
———	———	PAA10+2	PAA10+2
———	PAA15	PAA15	PAA14
PAA20	PAA20	PAA20	PAA14+1
———	———	PAA20+1	PAA14+2
	PAA14	PAA14	PAA15
	———	PAA14+1	PAA20
	———	PAA14+2	PAA20+1

Note that three instructions, the instruction to which number PAA14 was assigned and the following two instructions, were inserted, by the Assembly System sort, between the third and fourth lines of the original coding.

When using this method for insertions, the programmer must bear in mind that he has to change the instruction number designation of the fourth instruction. For instance, in his original coding the fourth line would have been assigned number PAA10+3 by the Assembly System. The programmer may, therefore, have made reference to PAA10+3 as a P-address elsewhere in the program. But to effect the insertion, what was PAA10+3 in the original coding has now become PAA15. All program references to PAA10+3 will no longer address the proper instruction. It, therefore, becomes necessary, when using this insertion method, to make certain that either no relative references were made to the changed instruction numbers, or if such references were made, that these references be changed accordingly.

To avoid the necessity for this concern, the user may wish to use the second method for effecting insertions.

Decimal Point Insertions

Again, let us assume that three pseudo-instructions are to be inserted between the third and fourth pseudo-instructions whose instruction numbers are written as follows:

PAA10
———
———
———
PAA20
———

The programmer is aware that the Assembly System will assign PAA10+2 to the third instruction, and PAA10+3 to the fourth instruction. If he could, therefore, write instruction numbers which fall sequentially between PAA10+2 and PAA10+3, the Assembly System will automatically insert the pseudo-instructions associated with these numbers between PAA10+2 and PAA10+3 during the sort.

The Assembly System makes this possible by permitting the programmer to write instruction numbers for instructions to be inserted in the following format:

PAGdd+k.n

where: PAGdd+k is the instruction number the Assembly System will assign to the instruction which is to precede the inserted instructions.

.n is a number (1 through 9) preceded by a decimal point. This number indicates the relative order of the inserted instructions.

The desired insertion can therefore be specified as shown:

Original Coding	*Revised Coding*	*Assembly System Assignment*	*Assembly System Sort*
PAA10	PAA10	PAA10	PAA10
———	———	PAA10+1	PAA10+1
———	———	PAA10+2	PAA10+2
———	———	PAA10+3	PAA10+2.1
PAA20	PAA20	PAA20	PAA10+2.2
———	———	PAA20+1	PAA10+2.3
	PAA10+2.1	PAA10+2.1	PAA10+3
	PAA10+2.2	PAA10+2.2	PAA20
	PAA10+2.3	PAA10+2.3	PAA20+1

Note that PAA10+3 remains undisturbed by the insertion. Any references to PAA10+3 still address the same instruction as they did in the original coding.

Descriptor Verb Designations

The methods for specifying instruction numbers described above apply to all Assembly System instructions except Descriptor Verbs.

A characteristic of Descriptor Verbs is that they generate no equivalent machine instructions in the final or object program. They are used during compilation to assist in language translation. Hence, the manner in which Descriptor Verbs are assigned instruction numbers for sequencing is somewhat different than in the case of other types of instructions.

First, no references may be made to a Descriptor Verb by another instruction. Descriptor Verbs cannot be used as destinations for transfers of control, for example. Neither can a Descriptor Verb be the subject of modifying instructions.

For sequencing purposes, *every* Descriptor Verb must have an entry in the Instruction Number column on the program sheet. The format of this entry is DXXXXX, where XXXXX is written exactly as any standard instruction number; e.g., PAB10.

Thus, Descriptor Verbs are assigned instruction numbers which are used during compilation for purposes of identification and segmentation. It is recommended that the same scheme adopted for numbering other instructions be extended to include Descriptor Verbs.

Example:

Assume that a Descriptor Verb appears in the program:

Instruction Number	*OP*
PAB10	DM
———	——
———	——
———	——
DPAB20	Desc. Verb
———	——
———	——
———	——
PAB30	TC

Note that the number given to the Descriptor Verb (DPAB20) is in sequence with numbers given immediately before (PAB10) and after (PAB30). In this way any possibility of duplicating numbers or confusing the meaning of instruction numbers is avoided.

It is further noted that instructions following DPAB20 are addressed relative to PAB10. The Descriptor Verb is treated, for this purpose, as if it were non-existent. The same program would appear after the Assembly System had sorted the pseudo-coding and completed the Instruction Number column as follows:

Instruction Number	*OP*
DPAB20	Desc. Verb
PAB10	DM
PAB10+1	——
PAB10+2	——
PAB10+3	——
PAB10+4	——
PAB10+5	——
PAB10+6	——
PAB30	TC

Referring to Parts of Instructions

Provisions are also made to provide the user with the ability to address any part of any instruction of the generated program by writing P-addresses in the following format:

PAGdd+k (Q)

where: PAGdd+k is the P-address of the pseudo-instruction.

(Q) may be either a one- or two-valued designation, and must be enclosed in parentheses.

If (Q) is a one valued designation, it may be either:

nnn – where nnn is a three-digit maximum decimal number from 1 to 999. This number represents the relative position of the addressed digit *from* the operation code of the first machine instruction generated by line PAGdd+K. (PAM22(1) would refer to the A1 character of the first instruction).

A – refers to the A3 character of the instruction generated by the entry in the OP column.

B – refers to the B3 character of the instruction generated by the entry in the OP column.

T – refers to the A3 character of the instruction generated by the entry in the T column. This is usually a "SET T" instruction. (Note that the T notation may not be used when the T entry is associated with the TCA, TCS, DA, DS, SSM or RAI instructions).

S – refers to the operation code of the instruction generated by the entry in the OP column. If a DM instruction with a T entry is written on line PAB30, a reference to PAB30 will address the operation code of the generated "SET T" instruction; however, PAB30(S) will address the machine operation code of the generated DM instruction.

G – refers to the operation code of the generated Control Simultaneous Gate instruction (CSG)

C – refers to the operation code of the generated Condition Transfer of Control instruction (CTC).

U – refers to the operation code of the generated Unconditional Transfer of Control (TC).

If (Q) is a two valued designation, the first value must be either S, G, C or U. The second value may be any one of the following:

m – a one-digit decimal number ranging from 1 to 7. This number represents the relative position of the addressed digit *from* the operation code specified by the first value.

A – refers to the A3 character of the instruction referred to by the first value.

B – refers to the B3 character of the instruction referred to by the first value.

N – refers to the address modifier digit of the instruction referred to by the first value.

Example:

If, for the following pseudo-instruction:

Inst. No.	*OP*	*A*	N_A N_B	*B*	*T*	N_T	*C S G*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
PAB10	DM	WATXER.6		"*18+"	WATXDE. 8		C	+	PAB10+2		–	PAB10+4		0	PAB10+8	

the Assembly System makes the following address assignments:

Pseudo-address	*Machine Address*
PAB10	002000
WATXER	037000
WATXDE	037006
"*18+"	027000
PAB10+2	002070
PAB10+4	002140
PAB10+8	002270

The following machine instructions will be generated:

Inst.	*OP*	*A*			*N*	*B*		
No.	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
00200	72	03	70	06	00	60	00	00
00201	53	03	70	00	00	02	70	00
00202	75	00	00	00	00	01	00	00
00203	61	00	20	70	00	00	21	40
00204	71	00	22	70	00	00	00	00

The following pseudo-addresses will, then, correspond to the indicated machine addresses.

Pseudo-address	*Machine Address*
PAB10	002000
PAB10(8)	002010
PAB10(S)	002010
PAB10(S3)	002013
PAB10(A)	002013
PAB10(B)	002017
PAB10(3)	002003
PAB10(T)	002003
PAB10(G)	002020
PAB10(GA)	002023
PAB10(CB)	002037
PAB10(U)	002040
PAB10(U4)	002044
PAB10(39)	002047
PAB10(C)	002030

V. SPECIAL ASSEMBLER FEATURES

Thus far, this manual has covered the RCA 501 Assembly System Data Sheet, the RCA 501 Assembly System Coding Sheet, their general usages, and the various addressing systems. This chapter will discuss a group of special Assembly Operation Codes (which have no counterpart in 501 machine-coding) and additional Assembly features.

SPECIAL MNEMONIC INSTRUCTIONS

Ignore Instruction

It was mentioned in Chapter III that the Assembly System allows the use of an IGN mnemonic instruction code in the "OP" column. This instruction has no counterpart in the RCA 501 repertoire of machine instructions.

The IGN instruction, itself, generates no machine instruction in the object program. When this instruction is used all entires on that line up to and including the B-column will be ignored when machine instructions are generated. Thus, the IGN instruction allows the programmer to take advantage of the options provided by the columns to the right of the B-address column, even if the instructions to be generated by the use of these columns are to be the first instructions of a block.

The programmer may use the A and B columns of IGN instructions to define working storage areas. In addition, when a relative P-address is to be deleted it should be *replaced* by an IGN instruction. This will preserve the relative order of instructions following the one deleted, and insure that all references made to those instructions will remain unaffected by the deletion.

Example:

If the programmer wishes to close the simultaneous gate with the first instruction in his program, he can specify this as follows:

Inst. No.	*OP*	*A*	$N_A N_B$	*B*	*T*	N_T	*C S G*	*IF*	*GO TO*
PAA10	IGN						C		

Reserve Instruction

The Reserve instruction, mnemonic operation code, RES, provides the Assembly System user with the ability to reserve program space in the object program. It is written in the following format.

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
	RES	Total number of instructions to be reserved (max = 510)		

Instruction Number

May be an explicit-P, relative or decimal point P-address.

A-address:

The total number of instruction locations to be reserved is entered in the A-address. This is a decimal number, and may not exceed 510. Eight character locations will be reserved for each instruction.

N_A, N_B, B, etc.:

All columns to the right of the A-address column must be left blank.

The RES instruction causes the number of instruction lines specified in the A-address to be reserved and filled with octal zeros. These instruction lines will be reserved in the place where the RES instruction is written. To address locations in the reserved instruction lines, the programmer uses character relative designations.

Example:

Pseudo-code

Instr. No.	*OP*	*A*	N_AN_B	*B*	*T*
	IT	WDSSNO		W101(L+15)	
PAG20	RES	5			
	STC	KHEAD		KHEAD	W101(L+55)

Generated Machine Code

Inst. No.	*OP*	*A*	N_AN_B	*B*
003000	21	015750	00	037620
003010	00	000000	00	000000
003020	00	000000	00	000000
003030	00	000000	00	000000
003040	00	000000	00	000000
003050	00	000000	00	000000
003060	72	037670	00	600000
003070	21	016750	00	016764

The following character relative instruction addresses, then, refer to the indicated machine addresses.

PAG20	= 003010	PAG20(15)	= 003027
PAG20(8)	= 003020	PAG20(39)	= 003057
PAG20(24)	= 003040	PAG20(27)	= 003043

The Duplicate Instruction

The special Duplicate, DUP, mnemonic instruction provides the programmer with the ability to duplicate various portions of his program in other parts of the program. The duplicated portion will be duplicated with some modifications.

Format:

Instr. No.	*OP*	*A*	N_AN_B	*B*	*T*
explicit P-address	DUP	explicit P-address		relative P-address	Data Address Modification, if any

Instruction Number:

An explicit P-address must be entered by the programmer in the Instruction Number column of a DUP instruction. The next pseudo-instruction must also be assigned an explicit P-address. Note that only 10 DUP instructions may appear in a program.

OP:

DUP is entered in the "OP" column.

A-address:

The P-address of the first instruction in the section to be duplicated is entered in the A-address. This address must be an explicit P-address which the programmer has assigned to that instruction.

B-address:

A P-address, *which must* be relative to the P-address entered in the A-column, is entered in the B-address column. The entry in the B-column, therefore, is the address of the last relative instruction in the section to be duplicated. (Note that no other explicit P-address can appear in the area to be duplicated.)

T-address:

The T-address is left blank if no data address modification of the duplicated coding is desired.

Data Address Modification

The option is given to the programmer to ask for the following data address modifications by appropriate entries in the T-address column of the DUP instruction.

Entry in the T-address Column	*Modification Effected*
F	All data addresses suffixed with dollar signs ($) in the duplicated area referring to data in the X, Y, and Z read-in areas will be changed to address the same item and file names in the F read-in area. All WX, WY, and WZ addresses suffixed with dollar signs ($) will be changed to W addresses.
XF	All data addresses suffixed with dollar signs ($) in the duplicated area referring to data in the F, Y, and Z read-in areas will be changed to address the same item and file names in the X read-in area. All W, WY, and WZ addresses suffixed with dollar signs ($) will be changed to WX addresses.
YF	All data addresses suffixed with dollar signs ($) in the duplicated area referring to data in the F, X, and Z read-in areas will be changed to address the same item and file names in the Y read-in area. All W, WX, and WZ addresses suffixed with dollar signs ($) will be changed to WY addresses.
ZF	All data addresses suffixed with dollar signs ($) in the duplicated area referring to data in the F, X, and Y read-in areas will be changed to address the same item and file names in the Z read-in area. All W, WX, and WY addresses suffixed with dollar signs ($) will be changed to WZ addresses.

Special Treatment of RD and MSW Instructions in Duplicated Areas

As shown in Chapter VI, if a programmer wishes to Random Distribute messages from alternate read-in areas to a common working storage, he uses the Random Distribute All option. He then uses the machine format Random Distribute option to accomplish the distribution of data from other read-in areas to the same working storage area by specifying the same L-address that was named in the Random Distribute All option. In this way he avoids the Assembly System generation of two separate, but exactly alike, destination lists.

Suppose, however, that a Random Distribute All instruction is included in the section to be duplicated. Will this cause two or more Random Distribute All instructions for the same file, to the same list, to be generated? No, because the Assembly System scans all of the coding to be duplicated and eliminates the word, ALL, in the T-addresses of any Random Distribute instruction in which it appears.

Similarly, the Assembly System user is provided with various other Random Distribute and Multiple Sector Write options, some of which specify destination lists. The Assembly System makes special provisions for these options when they are duplicated, such that two separate lists are not generated as a result of the duplicated instruction.

Modification of References to First P-address

Any reference, within the area to be duplicated, to the first, and only the first, line of the area to be duplicated, will be changed to refer to the first line of the duplicated area; i.e., if an area to be duplicated begins in PDQ50 and is duplicated by a DUP instruction on line PJW40, all reference within the duplicated coding to PDQ50 will be changed to PJW40.

Summary of DUP Modifications

The DUP instruction makes the following modifications automatically:

1. Provides for non-duplication of lists specified in duplicated Random Distribute and Multiple Sector Write instructions.
2. Changes all references to the first line within the duplicated area to address the first line of the new area.
3. The option is left to the programmer to change data addresses to alternate read-in area data addresses and distributed W-addresses to alternate read-in distributed W-addresses.
4. Duplicated coding may be addressed character relative to the P-address of the DUP instruction in exactly the same manner as reserved instructions and their parts may be addressed character relative to the P-address of the RES instruction. Duplicated coding may also be addressed relative to the beginning P-address in the same manner as other P-relative addressing, e.g. PAB10+13(CA).

The last instruction of an area to be duplicated cannot be followed by an ADV, nor can the last instruction contain a list. Multiple duplications of the same area must all be of the same length, i.e., the A and B addresses of the DUP verb must all be the same.

THE STASH DESCRIPTOR VERB

The Stash, STSH, Descriptor Verb provides the user with the ability to store the machine address associated with any P-address or a special tally quantity in a symbolically addressable memory location. The Stash Descriptor Verb is especially useful when combined with some of the special options to be discussed in the following sections.

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	STSH	A-Symbolic		P-address to be stored or Tally Quantity

Instruction Number:

A DP-address Descriptor Verb instruction number must be entered in the Instruction Number column. Each STSH instruction must have a unique DP-address and up to 49 STSH verbs may appear in any one segment.

OP:

STSH is entered in the "OP" column.

A-address:

An A followed by up to five alphanumerical characters is entered in the A-address column. This entry then becomes the symbolic name of the location in which the machine address or tally quantity is stored and is called an A-symbolic.

B-address:

Enter in the B-address, the P-address or tally quantity whose corresponding machine address is to be stored.

If a tally quantity is specified, it is entered as a decimal number which is *not* enclosed in quotes. The tally quantity will be stored as a six octal digit number in a memory location addressable by the A-symbolic.

Example:

Instr. No.	*OP*	*A*	N_AN_B	*B*
DPAL10	STSH	ATA2		2
PAL20	TCT	ATA2		WATA2
●	●	●		●
●	●	●		●
●	●	●		●
	TA	PAL20+1		WATA2
	TCT	ATA2		WATA2

In the above example the computer will go through the coding loop three times. Upon emergence from the loop the tally quantity is reset.

Note that addresses stored by use of the STSH verb are considered non-fixed constants; i.e., they are considered to be part of the segment in which they are specified and may be overlaid by a following segment.

VARIABLE ADDRESSES

The RCA 501 Assembly System provides the programmer with the option of deferring the naming of some addresses until after the major body of pseudo-code is completed.

For instance, a run may be split into two sections, each section being written by a different programmer. One programmer may have need to address an instruction in the section written by the other programmer; however, the address of that instruction is not yet known. The programmer may then write a variable address instead of the P-address. Once the exact P-address becomes known the variable address is then defined by the Define Variable Descriptor Verb.

A variable address consists of at most six characters, the first of which must be a "V". The other five may be an alphanumeric mnemonic composed by the programmer for his own convenience.

For example, a programmer wishes to transfer control to some entry point in a section of coding written by another programmer. The instruction number of this point, however, is not yet established. The programmer may write:

OP	*A*	N_AN_B	*B*
TC	VENTRY		

Once it is established that the entrypoint is PLZ80, a Define Variable verb may be written defining VENTRY as PLZ80.

Define Variable Descriptor Verb

The format of the Define Variable Address instruction is:

Inst. No.	*OP*	*A*	*B*
DP-address	DEFV	VNNNNN	Machine (M) or any Symbolic Address

where: DEFV is the operation code; V specifies a variable address; NNNNN is an arbitrary designation of up to five alphanumeric characters assigned by the programmer; the actual address, machine or symbolic, which is to be substituted for the V-address is specified in the B column. This may *not* be another V-address.

Note that each Define Variable Instruction must have a DP-address; also, up to 99 variables may be defined in a program.

Example:

VENTRY will be defined as PLZ80 by the following Descriptor Verb.

Inst. No.	*Comments*	*OP*	*A*	*B*
DPAN62		DEFV	VENTRY	PLZ80

V-addresses may be used to define *any* symbolic (constants, P-addresses, work-areas, etc.) except another V-address. Note that when a variable is defined it *CANNOT* have appendages. For example: PLZ80+2, WPAY (L+1), etc. are incorrect. Once defined, however, appendages may be used with the V-address in the same manner as other symbolic addresses. (These appendages must comply with the rules governing relatives used with the substituted value.)

Assume that VENTRY has been defined as PLZ80, and the programmer desires to transfer to the second relative instruction following VENTRY.

```
TC     VENTRY+2
```

The assembler will substitute for this entry:

```
TC     PLZ80+2
```

THE ADV MNEMONIC INSTRUCTION

The ADV mnemonic instruction is a special option which permits the programmer to automatically increment or decrement addresses appearing in a pseudo-instruction by other symbolic addresses. When this function is desired a line is written immediately following the instruction to be modified. The letters ADV are placed in the OP column of this added line, and the incrementing or decrementing addresses are placed in the column(s) below the addresses to be modified.

If a minus sign appears to the left of the modifying address, that address will be subtracted from the address appearing above. A plus sign would cause an addition.

Example:

Assume that a routine is to be written such that it is self-relative. A transfer of control in this routine must have as its destination address the distance between this instruction and the destination instruction. Further, this address must be modified by the contents of the program register (BP, B4, RP). If the TC appears at PAB10 and the destination is PAB20 the instruction is written:

Instruction Number	*OP*	*A*	N_A
PAB10	TC ADV	PAB20 –V1(8)	BP

Somewhere in the program V1 is defined:

OP	*A*	*B*
DEFV	V1	PAB10

The effect of this coding is to subtract the machine addresses given to PAB10 and PAB20 and to store the result as the address of the TC instruction. Note that a minus sign to the left of V1 indicates that the operation is a subtraction. A plus sign would have caused the addresses to be added. Further note that V1 was adjusted in the example by (8) to compensate for the setting of the program register to the address of the next instruction.

No entry may be made in the Instruction Number column on a line that has ADV as the OP code. Such a line is considered part of the previous pseudo-instruction for purposes of relative instruction addressing.

Instruction Number	*OP*
PAB10	TC
	ADV
	DM
	IT
	TC
PAB20	LRF

When the Assembly System fills out entries in the Instruction Number column they appear:

Instruction Number	*OP*
PAB10	TC
	ADV
PAB10+1	DM
PAB10+2	IT
PAB10+3	TC
PAB20	LRF

The ADV may be used to modify a V address as well as any other symbolic or machine address. Only one ADV line can be used after a pseudo-instruction. Any pseudo-instruction except a Descriptor Verb can be modified with ADV. ADV may not be used with call lines and entrance lines. It can, however, be used within subroutines. Finally, the V entries on an ADV line may be made in one or more of the address columns (A, B, T, GO TO) but not in the Instruction Number, OP, N, CSG, IF or Comments columns.

The ADV verb may also be used to modify addresses by other known addresses.

For instance, in the previous example, where the A-address of a TC instruction was to be decremented by PAB10+1, it could have been written directly on the ADV line:

Instr. No.	*OP*	*A*	*N_A*
PAB10	TC	PAB20	BP
	ADV	–PAB10+1	

In this case no DEFV verb is necessary.

VI. SPECIAL ASSEMBLY OPTIONS

The Assembly System user may write most 501 instructions in a form close to that of machine coding, taking advantage, of course, of symbolic addressings and other Assembly features.

This chapter will discuss the use of special Assembly options available with certain 501 instructions. Some of these options will involve a change in format only; others will introduce special addresses not covered in the chapter on addressing. All options have been designed to provide the Assembly user with greater versatility and more powerful pseudo-instructions.

SYMBOLIC ADDRESSES IN TC INSTRUCTION

The B-addresses of TC instructions written on their own lines may be entered with the SWO ... 5 notation described in Chapter III.

Example:

OP	*A*	N_AN_B	*B*
TC	PAB10		SW235

THE TAPE SENSE INSTRUCTION

As shown in Chapter III, a Tape Sense instruction may be specified in the "IF–GO TO" columns of the coding sheet; however, the Tape Sense instruction may also be written in the following format:

OP	*A*	N_AN_B	*B*	*T*
TS	Jump Address		Tape No.	Sense Condition(s)

A-address:

An M-address or P-address is entered in the A-address of this instruction. The A-address entry is the address of the instruction to which control is to be transferred if the condition, or one of the conditions, specified in the T-address is found to prevail.

B-address:

An I, O, or T Tape Station designation is entered in the B-address of this instruction. The B-address entry designates the Tape Station to be tested.

T-address:

The T-address entry specifies the condition or combination of conditions to be tested by entering any one, or combination, of the following letters:

B for Is the tape positioned on BTC?
E for Has ETW been sensed?
F for Is the tape now stationary or moving forward?
R for Is the tape now moving in the reverse direction?
M for Is the tape now in motion?
N for Is the Tape Station non-operable?

SPECIAL ADDRESSES WITH UNWIND AND REWIND INSTRUCTIONS

The Unwind n Symbols and Rewind n Symbols instructions may be written in the following format:

OP	*A*	$N_A N_B$	*B*
UNS or RNS	Symbol, No. of Symbols		Tape No.

A-address:

The symbol and the number of symbols to be unwound or rewound is designated in the A-address in the following format:

SYM,n

where: SYM is the designated symbol.
Symbols are designated by one of the following notations:

EM	=	End Message Symbol
ED	=	End Data Symbol
EF	=	End File Symbol
SM	=	Start Message Symbol
GAP	=	Intermessage Gap

,n is a comma followed by a decimal number up to 4095 indicating the number of symbols to be rewound or unwound.

B-address:

An I, O̲, or T tape station designation is entered in the B-address.

Example:

OP	*A*	$N_A N_B$	*B*
RNS	EM,29		T10

SPECIAL ADDRESSES WITH LOCATE *n*th SYMBOL

The Locate *n*th Symbol instruction may be written in the following format:

OP	*A*	$N_A N_B$	*B*	*T*
LNS	LHE of Sector		RHE of Sector	Symbol, Number

A-address:

An M-address or any symbolic address designating the left hand end of the sector to be searched is entered in the A-address.

B-address:

An M-address or any symbolic address designating the right hand end of the sector to be searched is entered in the B-address.

T-address:

The symbol and the number of the designated symbols to be located is entered in the T-address in the following format:

SYM,n

where: SYM may be any RCA 501 character. The Start Message, End Message, Item Separator, End Data and

End File Symbols are represented by:

SM
EM
ISS
ED
EF
respectively.

,n is a comma followed by a decimal number up to 4095 indicating the number of symbols to be located.

Examples:

OP	*A*	N_AN_B	*B*	*T*
LNS	FMASEM		FMASEM	ISS,5
LNS	FMASEM		FMASEM	M,20
LNS	FMASEM		FMASEM	1,5
LNS	FMASEM		FMASEM	L,30

It should be noted that the symbol to be located can only be shown *symbolically* in the T column of a LNS instruction. Symbols *cannot* be shown symbolically within a separate SET instruction.

SYMBOLIC ADDRESSES FOR PAPER ADVANCE INSTRUCTION

The Paper Advance instruction may be written in the following format:

OP	*A*	N_AN_B	*B*
PA	PC VT Decimal Number or M-address		

A-address:

Either PC, VT, a Decimal Number (not in quotes) or an M-address may be entered in the A-address.

If PC, the PA instruction will cause a loop controlled page change.

If VT, the PA instruction will cause a loop controlled vertical tabulation.

If a decimal number, the instruction will cause the paper to advance the number of lines specified by the decimal number.

GENERATED SET INSTRUCTIONS

It was stated in Chapter III that generally any entry in the T-address or N_T columns of the coding sheet will cause the Assembly System to generate a "Set Register T" instruction and place it before the instruction specified by the mnemonic operation code.

OTHER THAN "SET" GENERATED BY "T" ENTRIES

"T" Entries Generating TCT Instructions:

A T-address or N_T entry on a line with the following four mnemonic instructions will generate a TCT instruction rather than a SET instruction:

TCA	SSM
TCS	RAI

The generated TCT instruction will immediately precede the instruction generated by the mnemonic operation code.

Examples:

Pseudo-Code						*Corresponding Machine Instructions*			
OP	*A*	N_AN_B	*B*	*T*	N_T	*OP*	*A*	*N*	*B*
TCA	PAB20		#000030#	PMA74		25 44	PAB20 PMA74	00 00	PMA74 #000030#
TCS	PAC30		#000020#	PFA10		25 45	PAC30 PFA10	00 00	PFA10 #000020#

Note from the above chart, that the use of the T-address column allows the programmer to write what is, in effect, a three-address instruction.

For the TCA instruction, the A-address specifies the augend, the B-address specifies the addend, and the T-address specifies the sum location.

For the TCS instruction, the A-address specifies the minuend, the B-address specifies the subtrahend, and the T-address specifies the difference location.

Pseudo-Code						*Corresponding Machine Instructions*			
OP	*A*	N_AN_B	*B*	*T*	N_T	*OP*	*A*	*N*	*B*
SSM	PAC22		PAC26	AS3		25 62	AS3 PAC22	00 00	000203 PAC26
RAI	DPAY		WPAY	AS3		25 77	AS3 DPAY	00 00	000003 WPAY

As stated in Chapter III, the SSM instruction may be generated by entries in the "IF–GO TO" columns. This method, however, does not provide for the possibility of a Paper Advance occupying the simultaneous mode. If the programmer wishes to make special provision for this possibility, the SSM instruction may be written as shown on the above chart. If this option is exercised, the A-address must specify the instruction to which control is to be transferred if a "read" instruction is occupying the simultaneous mode; the B-address must specify the instruction to which control is to be transferred if a "write" instruction is occupying the simultaneous mode; and the T-address must contain the *A-Symbolic* name of the location containing the address of the instruction to which control is to be transferred if a Paper Advance is found to be in the simultaneous mode.

Note that to accomplish this, operation code 71 must be placed in memory location 000200. This may be accomplished by a prior One Character Transfer instruction, or through the use of the Assign Descriptor verb.

For the RAI instruction, the A-address specifies the address to appear in Register A when the program is re-entered, the B-address specifies the address to appear in Register B when the program is re-entered, and the T-address specifies the *A-Symbolic* of the instruction to which control will be transferred by the RAI instruction.

"T" Entries Generating JR Instructions

Entries in the T-address or NT columns of DA and DS instructions generate preceding Justify Right instructions:

Pseudo-Code						*Corresponding Machine Instructions*			
OP	*A*	N_AN_B	*B*	*T*	N_T	*OP*	*A*	*N*	*B*
DA	DPAY		DTAX	DAMT		33 51	DPAY DAMT	00 00	DAMT DTAX
DS	DPAY	B1B3	DTAX	DAMT	B5	33 52	DPAY DAMT	15 53	DAMT DTAX

SPECIAL NOTE: In the TCA, TCS, DA and DS instructions the T-address is repeated in both the instruction generated by the T-address or N_T entry, and in the instruction generated by the operation code. Because of this repetition of the T-address in two of the generated instructions, the programmer may *not* refer to the T column entry of these special instructions in the PAGdd+k(T) format.

It should also be noted that if an address modifier is specified for the repeated address, the address modifier will be associated with that address in *both* instructions.

RANDOM DISTRIBUTE SPECIAL OPTIONS

Several options are available to the Assembly System user for writing Random Distribute instructions.

Random Distribute All

The Random Distribute All option causes the Assembly System to automatically assign working storage addresses for every item distributed by the instruction. This option is written in the following format:

OP	*A*	$N_A N_B$	*B*	*T*
RD	File name		L-address	ALL

A-address:

FILE NAME is the name of the file read-in area from which all items in the message are to be distributed. This may be an F, X, Y, or Z read-in area. If the file name is used alone the SM will also be distributed.

If it is not desired to distribute the SM, or any of the first several FAA items, the File Name+k notation may be used in the A-address. When this notation is used, the kth item, and all following items, will be distributed. Note that item names cannot be used with this option.

B-address:

L-ADDRESS is a list name provided by the programmer. The first character must always be "L", followed by up to five alphanumeric characters. The L-address becomes the symbolic address of where the list of addresses is stored in memory.

T-address:

ALL specifies that this particular Random Distribute option is being used. Use of the "ALL" option causes the Assembly System to automatically generate a destination list for all items to be distributed. Items will be placed in the destination area in contiguous memory locations, with the amount of memory reserved for each item depending on the *maximum* item size (including sign) as given in the data sheets.

A W-address is assigned to each item placed in the destination area. This W-address is the symbolic name of the item, preceded by the read-in area letter, preceded by a "W". For example, if XFMAST is distributed, individual items in the destination area can be addresses as:

WXDNAME
WXDPAY
WXDATE
etc.

(If FMAST were distributed, individual items would be addressed in the destination area as WDNAME, WDPAY, WDATE, etc.)

The entire distribution area is also assigned a symbolic name which will be the name of the read-in area preceded by a "W". For example WXFMAST would address the entire distribution area. When used in an address requiring the LHE of the area, WXFMAST would address the distributed SM. When used in an address requiring the RHE of the area, WXFMAST would address the distributed EM — this location will contain an ISS symbol at the completion of the RD instruction (see 501 Programmer's Reference Manual).

The size of the distribution area will be the sum of the maximum number of characters in all of the items of the message, plus 2. The extra two locations are used for the SM and EM.

The generated address list is automatically stored by the Assembly System and may be later referred to by its L-name.

Example:

Suppose it is desired to process the Master Employee File shown on the data sheet on page 62. To read a message from this file into memory, and Random Distribute it, the programmer may write the following instructions:

OP	*A*	$N_A N_B$	*B*	*T*
LRF	FMASEM		IMAS	
SCC	WFMASEM		WFMASEM	
RD	FMASEM		LMAS	ALL

Note, from the above example that "WFMASEM" is the address of the working storage area into which the "Random Distribute All" instruction distributes the file. "WFMASEM" can be used for either the left or right hand address of that area according to the requirements of the address of the particular instruction in which it is used. Having distributed a message through the use of the "Random Distribute All" instruction, *the programmer may make reference to all items in the message, both FAA and non-FAA items, symbolically by their destination addresses; e.g.*

WDEMPNO addresses Employee Number
WDPLNO addresses the sub-item, PLANT NO.
WDNAME addresses Employee Name
WDCEARN addresses Cumulative Earnings
etc.

Having once used the "Random Distribute All" instruction the programmer can cause the distribution of messages from alternate read-in areas of the same file to the same work area by specifying the same L-address in the B-address of a machine format Random Distribute instruction.

Example:

The Random Distribute All instruction, shown below, will distribute all items of the message from the primary read-in area to a work area.

OP	*A*	$N_A N_B$	*B*	*T*
RD	FTEST		LMAS	ALL

To distribute the same items of a message of the same file to the same work area from the X read-in area, the following instruction is written:

OP	*A*	$N_A N_B$	*B*	*T*
RD	XFTEST		LMAS	

The "Random Distribute All" option *is not used for this second transfer,* because if it were used, the Assembly System would assign a different working storage area to the items distributed from the X read-in area. The machine format Random Distribute instruction is used with the B-address specifying the list generated by the initial "Random Distribute All" instruction instead.

Having done this, the programmer can address all items in the work area by their names as they appear on his data sheets, preceded by a "W", regardless of which read-in area they were distributed from.

501 AUTOMATIC CODING DATA SHEET

NAME OF FILE <•F MASEM	PAGE• 1 OF• 1

NO. OF MSGS.	MAX. MSG. SIZE	NO. OF REELS	NO. OF STATIONS	KEYS
• 30000	•	•	•	• D PLNO • D DENO • D STNO • D SHNO • D MANO

MESSAGE FORMAT YES OR NO • Y OR N: Y	FILES TERMINATED BY EF YES OR NO • Y OR N: Y OTHER •	REELS TERMINATED BY ED YES OR NO • Y OR N: Y OTHER •

REMARKS •

MASTER EMPLOYEE FILE >

X OR BLANK   L OR R   X OR BLANK

ITEM NO.	SUB ITEM	ABBREVIATION						DESCRIPTION	F A A	JY	SIGN	NO. CHARS.		% USE	WTD. AVG.
												MAX.	AVG.		
<• 1	•	• D	E	M	P	N	O	• EMPLOYEE NO.	• X	• L	•	• 11	• 11	• 100	
•	• A	• D	P	L	N	O		• PLANT NO.	• X	• L	•	• 2	• 2	• 100	
•	• B	• D	D	E	N	O		• DEPARTMENT NO.	• X	• L	•	• 2	• 2	• 100	
•	• C	• D	S	T	N	O		• STATION NO.	• X	• L	•	• 2	• 2	• 100	
•	• D	• D	S	H	N	O		• SHIFT NO.	• X	• L	•	• 1	• 1	• 100	
•	• E	• D	M	A	N	O		• MAN NO.	• X	• L	•	• 4	• 4	• 100	
• 2	•	• D	S	S	N	O		• SOCIAL SECURITY NO.	• X	• L	•	• 9	• 9	• 100	
• 3	•	• D	N	A	M	E		• EMPLOYEE'S NAME	•	• L	•	• 40	• 30	• 100	
• 4	•	• D	D	E	P			• NO. OF DEPENDENTS	•	• L	•	• 2	• 1	• 100	
• 5	•	• D	H	R	R	T		• HOURLY RATE	•	• L	•	• 3	• 3	• 100	
• 6	•	• D	C	E	A	R	N	• CUMULATIVE EARNINGS	•	• L	•	• 6	• 6	• 100	
• 7	•	• D	C	W	T	H		• CUMULATIVE WTH TAX	•	• L	•	• 5	• 5	• 100	
• 8	•	• D	C	S	S	T	X	• CUMULATIVE SS TAX	•	• L	•	• 4	• 4	• 100	
• 9	•	• D	B	N	D	D	E	• BOND DEDUCTION	•	• L	•	• 5	• 4	• 30	
• 10	•	• D	B	N	D	E	N	• BOND DENOMINATION	•	• L	•	• 3	• 2	• 30	
• 11	•	• D	C	B	O	N	D	• CUMULATIVE BOND DED.	•	• L	•	• 5	• 4	• 30	
• 12	•	• D	C	C	C			• COMBINED CHAR. CONTR.	•	• L	•	• 4	• 2	• 25	
• 13	•	• D	H	O	S	P		• HOSPITALIZATION DED.	•	• L	•	• 4	• 3	• 25	
• 14	•	• D	A	U	T	I	N	• AUTO INSURANCE	•	• L	•	• 4	• 4	• 10	>EF
•	•	• D						•	•	•	•	•	•	•	
•	•	• D						•	•	•	•	•	•	•	
•	•	• D						•	•	•	•	•	•	•	
•	•	• D						•	•	•	•	•	•	•	
•	•	• D						•	•	•	•	•	•	•	
•	•	• D						•	•	•	•	•	•	•	

TOTAL CHARACTERS OF INFORMATION			
ADDED CONTROL CHARACTERS			
TOTAL CHARACTERS			

IE 241   1/59

Random Distribute All+1

Format:

OP	*A*	N_AN_B	*B*	*T*
RD	File name		L-address	ALL+1

When this option is used, the Assembly System will allot one extra position for all items placed in the destination area *if the item contains an Item Separator symbol.*

In all other respects, this option is identical to the Random Distribute All.

Random Distribute to Specified List

The "Random Distribute to Specified List" option provides the user with the ability to specify a destination list, containing up to 57 entries, in the Random Distribute instruction. It is written in the following format:

OP	*A*	N_AN_B	*B*	*T*
RD	LHE of Area to be Distributed		L-Address	List of Destination Addresses (limit = 57) END

A-address:

The A-address may contain an M-address, a Symbolic Data Address (either read-in area name, an address relative to the read-in area name or an Item Name) or a W-address. The address will specify the LHE of the area to be distributed.

B-address:

In the B-address the programmer specifies some L-address. This will become the symbolic address of the list specified in the T-address. Having once specified an L-address for a list, the programmer may make further references to the same list by its L-name.

T-address:

In the T-address column, the programmer lists, on successive lines, the destination addresses. No other entries may be made on the succeeding lines taken up by the destination list. Entries which may appear in the destination list are as follows:

W-address – Some working storage
M-address – An actual machine location
D-name – Some symbolic data address
P-address – Some instruction address
TA – Indicating throw away addresses. The Assembly System will substitute $(777777)_8$ for TA.
END – Must end each list. The Assembly System will substitute $(000000)_8$ for END.

Example:

OP	*A*	N_AN_B	*B*	*T*
RD	DBAL		LBAL	WAMT WNAME TA TA MO37775 DNET END

Random Distribute to Unspecified List

This option allows the programmer to write a Random Distribute instruction where the list is unspecified. It is assumed that the list will be generated by the program itself. The Random Distribute to an Unspecified List is written in the following format:

OP	*A*	N_AN_B	*B*	*T*
RD	LHE of Area to be Distributed		L-address	SAVE J

A-address:

The A-address may contain an M-address, a Symbolic Data Address, or a W-address. The address will specify the LHE of the area to be distributed.

B-address:

Since this option is designed to allow the programmer to generate his own destination list in other parts of the program, an L-address must be specified in the B-address so that reference can be made to the list area to load the generated addresses into the list.

T-address:

SAVE J must be entered in the T-address for this option, where "J" equals the decimal number of tetrads to be reserved for this list. Caution should be exercised in specifying the value of "J", since enough room should be left for throw away addresses $(777777)_8$ and ending addresses $(000000)_8$.

The Assembly System will reserve the number of tetrads specified by J, filling them initially with $(000000)_8$. The programmer may load addresses into the list by referring to the L-address using character relative notation, (L+n)(R−n), to specify particular tetrads in the list.

Example:

Suppose the following "Random Distribute to Unspecified List" instruction is written:

OP	*A*	N_AN_B	*B*	*T*
RD	FMASEM+1		LMASEM	SAVE 5

Further, suppose the destination addresses of items, one, two, and four of the message are generated by other parts of the program and, after generation, are to be found in working storages as follows:

Item one	–	WA1
Item two	–	WA2
Item four	–	WA4

Item three is to be a throw away item.

The list would then be loaded as follows:

OP	*A*	N_AN_B	*B*
TCT	WA1		LMASEM(L+3)
TCT	WA2		LMASEM(L+7)
TCT	#777777#(C1)		LMASEM(L +11)
TCT	WA4		LMASEM(L+15)

Random Distribute to Combination List

A Random Distribute option in which the programmer specifies some addresses and reserves room for others to be program-generated is also available. The list may contain up to 57 entries, and is written in the following format:

OP	*A*	*B*	*T*
RD	LHE of Area to be Distributed	L- address	Address Address or Save J Address or Save J Address or Save J etc. END

A-address:

The A-address may contain an M-address, a Symbolic Data Address or a W-address. The address will specify the LHE of the area to be distributed.

B-address:

In the B-address the programmer specifies some L-address.

T-address:

In the T-address column, the programmer lists, on successive lines, destination addresses or SAVE J entries. (Note that SAVE J cannot appear as the first entry.) Destination addresses may be any one of the following:

W-address – Some working storage
M-address – An actual machine location
D-name – Some symbolic data address
P-address – Some instruction address
TA – Indicating throw away addresses. The Assembly System will substitute $(777777)_8$ for TA.
END – Must end each list. The Assembly System will substitute $(000000)_8$ for END.

SAVE J entries will cause the number of tetrads specified by J to be reserved in the list in the place where the SAVE J was entered.

Random Distribute-Machine Format

The mnemonic Random Distribute instruction, just as all instructions, may be specified in a format very close to the format of machine code.

OP	*A*	$N_A N_B$	*B*
RD	LHE of Area to be Distributed		L-address

A-address:

The A-address specifies the left-hand end of the area to be distributed. This address may be specified by an M-address, a W-address, or some symbolic data address. If an entire message is to be distributed from the read-in area, the read-in area name may be entered in the A-address.

For example, FMASEM, XFMASEM, YFMASEM, or ZFMASEM, as entries in the A-address will cause a distribution from the corresponding read-in areas. Naming the read-in area will cause the SM to be distributed as well as the data.

If a part of a message is to be distributed, the item of the file from which distribution is to start, *provided* *the item is an FAA item,* may be specified by the item name or by its relative position with respect to the SM. Thus, if DNAME is the first item in the file, FILE+1 or DNAME will cause distribution of the entire message except the SM.

B-address:

The B-address specifies the address of the first tetrad of the destination list; this must be an L-address. It will be assumed that the list which is stored at that L-address has been defined elsewhere in the program.

GENERAL STATEMENTS ON RANDOM DISTRIBUTE OPTIONS

A. The programmer must realize that the Random Distribute instruction depends upon the ISS of the next item for all medial items, and the EM for the last item of a message, to key it to the fact that it has completed the transfer of a full item. Therefore, if some items of the file as described on the data sheets, do not have ISS's, they will not be distributed to discrete destination locations. They will be considered to be part of the preceding item and will be transferred as such.

B. All references to SM and EM apply only to data that is in message format. When distribution begins with a relative address, a character location is not allowed for the SM. In addition, if the A-address specifies an item in a file area, the item must be FAA.

C. Entries are not to be made in any columns to the right of RD instructions, or on any line associated with a RD instruction.

D. There is a maximum of 20 L-lists for any Assembly program. This limit includes lists created by RD *and* MSW instructions.

MULTIPLE SECTOR WRITE OPTIONS

The Assembly System provides the user with various options for writing the Multiple Sector Write instruction.

Multiple Sector Write From Specified List

The Multiple Sector Write from Specified List instruction may contain up to 57 entries, and is written in the following format:

OP	*A*	$N_A N_B$	*B*	*T*
MSW	L-address		Tape No.	
			J	Address
			J	Address
			etc.	
			END	

A-address:

An L-address entry must be made in the A-address in the first line. The first character must be "L", followed by up to five alphanumeric characters.

B-address:

On the first line, the Tape Station number is entered. This may be an I, O, or T Tape Station designation.

On *succeeding* lines, the number of characters in each sector to be written is entered on individual lines. Decimal notation is used. The number of characters in each section is *not* limited to 64.

The last entry in the B-address column for this instruction must be "END". No other entries are made on the line in which "END" is entered.

T-address:

The first line (the line in which "MSW" is entered) of the T-column is always left blank.

The T-column on succeeding lines contains the LHE address of each sector to be written. Note that the number of characters to be written is entered in the B-column on the same line.

Entries in the T-column may be M-addresses, W-addresses, K-addresses, P-addresses, or Symbolic Data addresses. If a symbolic address is used the Assembler will automatically supply the LHE address.

Note that if the B-address is left blank the Assembly System will assume that the sector specified by the related T entry consists of one character. If the T-address is left blank, the Assembly System will substitute the machine address $(000000)_8$.

Example:

OP	*A*	N_AN_B	*B*	*T*
MSW	LWRITE		OPRINT	
			12	WNAME
			73	WAAD(L+5)
			4	KTWO
			END	

Multiple Sector Write From Unspecified List

An option is provided by the Assembly System to permit the programmer to write a Multiple Sector Write instruction from an unspecified list. It is assumed that the list will be generated in other parts of the program. The Multiple Sector Write Unspecified List option is written in the following format:

OP	*A*	N_AN_B	*B*	*T*
MSW	L- address		Tape No.	No. of Tetrads

A-address:

The A-address entry must be an L-address to provide the programmer with a reference address for use in loading the list with program-generated addresses.

B-address:

The tape station is specified by an I, O, or T Tape Station address.

T-address:

A decimal number indicating the number of tetrads to be reserved for the list is entered under T-address. Note that the terminating tetrad $(000000)_8$ must be provided for. In loading program-generated addresses into the list, the L-address with (R – n) (L + n) character relative notation is used. All tetrads in the list will be initially filled with octal zeros.

Example:

OP	*A*	N_AN_B	*B*	*T*
MSW	LWRITE		OPRINT	17

Multiple Sector Write From Combination List

The Multiple Sector Write from Combination List option provides the Assembly System user with the ability to write a Multiple Sector Write instruction in which some addresses in the list are specified, and some are left unspecified. The unspecified addresses are to be generated by the program. This option is written in the following format:

A-address:

An L-address, which the programmer will use as a reference address to load program-generated addresses into the list, is entered in the A-address of the first line of this instruction.

B-address:

An I, O, or T symbolic Tape Station number is entered in the B-address of the first line of this instruction.

On succeeding lines, either J or SAVE is entered. "J" is the number of characters in each sector whose address is specified in the T-column. SAVE is entered for those sectors whose addresses will be generated by the program.

The last entry in the B-column must be "END". No other entries may be made on the "END" line.

T-address:

The first line of the T-address column is left blank. Succeeding lines are entered as follows:

1. Opposite each "J" entry in the B-column, enter the address of the sector that is defined by that "J". These entries may be M-addresses, W-addresses, K-addresses, P-addresses or Symbolic Data Addresses. If a symbolic address is entered, the Assembler will substitute the LHE address of that symbolic.
2. Opposite each "SAVE" entry in the B-column enter the number of tetrads to be reserved by the Assembly System for program-generated addresses. All save tetrads will be initially filled with octal zeros.

Example:

The following sample MSW instruction may be written:

OP	*A*	N_AN_B	*B*	*T*
MSW	LLIST		T20	
			15	WA3
			13	WA6(L+4)
			12	M003000
			5	K123
			SAVE	3
			END	

Note that the program must generate the octal character count and also the addresses to be inserted in the three tetrads designated by SAVE 3. For example, assume that these addresses have been generated and stored in the following work areas:

First address	– WB1
Second address	– WB2
Third address	– WB3

To load the list with these addresses, and the octal number of characters in each sector, the following instructions may be written.

OP	*A*	N_AN_B	*B*
OCT	#12#		LLIST(L+16)
TCT	WB1		LLIST(L+19)
OCT	#04#		LLIST(L+20)
TCT	WB2		LLIST(L+23)
OCT	#05#		LLIST(L+24)
TCT	WB3		LLIST(L+27)

Note that character relative notation is used to load program-generated addresses into the list, and that the L-address was considered as corresponding to the first character of the *entire* list, not the first character of the saved tetrads.

Multiple Sector Write-Machine Format

The Multiple Sector Write instruction may be written in a format similar to machine code format as follows:

OP	*A*	N_AN_B	*B*
MSW	L-address		Tape No.

A-address:

An L-address entry must be made in the A-address of this option. The L-address is the address of the list designating the sectors to be written. The list referred to by this L-address ***must be defined elsewhere in the program.***

B-address:

An I, O, or T Tape Station specification is entered in the B-address.

GENERAL STATEMENT ON MULTIPLE SECTOR WRITE OPTIONS

When more than 64 characters are specified for a single sector, the list will contain a tetrad address for each multiple of 64. For instance, if 132 characters are specified in the B-column, the list will contain three tetrads for that sector.

The Assembler will supply the LHE address of symbolics entered in the T-column. If other addresses are required, character addressing must be employed.

A specified list of addresses may contain up to 57 entries. It should also be noted that there is a maximum of 20 L-lists for any Assembly program (this includes lists created by RD ***and*** MSW instructions).

When SAVE is used, the programmer must transfer the octal character count and the starting address into each SAVE tetrad of the list.

Entries are not to be made in any columns to the right of MSW instructions, or on any line associated with a MSW instruction.

ADDRESS MODIFIER ADDRESSES

The seven address modifiers may be specified by the following symbolic names:

Symbolic Name	*Address Modifier*	*Machine Address*
B1	AM1	000111–000113
B2	AM2 (STA)	000221–000223
B3	AM3	000131–000133
B4	AM4 (P Register)	
B5	AM5	000151–000153
B6	AM6 (T Register)	
B7	AM7	000171–000173

These symbolic names may be used to specify the address modifiers in the A, B, T or N columns of all mnemonic instructions. However, the following special formats must be observed.

Setting Address Modifiers

The SET instruction may be used to place information into address modifier 2 (STA), address modifier

4 (P Register) and address modifier 6 (T Register).

Examples:

pseudo-code:

OP	*A*	$N_A N_B$	*B*
SET	M000010		B2

equals in machine code:

```
72        000010        00        200000
```

pseudo-code:

OP	*A*	$N_A N_B$	*B*
SET	PAB20		B4

equals in machine code:

```
72        023000        00        400000
```

where: 023000 is the machine address the Assembly System has assigned to PAB20.

pseudo-code:

OP	*A*	$N_A N_B$	*B*
SET	#000010#		B6

equals in machine code:

```
72        033000        00        600000
```

where: 033000 is the address in which the Assembly System has stored the literal, #000010#

Since the other address modifiers (B1, B3, B5 and B7) are actual HSM locations, information *cannot* be placed in them through the SET instruction.

The Assembly user, however, may transfer information into these locations with a TCT instruction; or he can add to or subtract from these locations using the TCA or TCS instructions. When these instructions are used in conjunction with these modifiers the symbolic addresses B1, B3, B5 or B7 may appear in the A or B-addresses.

Examples:

pseudo-code:

OP	*A*	$N_A N_B$	*B*
TCT	B1		B3

generated machine code:

```
25        000113        00        000133
```

pseudo-code:

OP	*A*	$N_A N_B$	*B*
TCA	B5		#001000#

generated machine code:

44 000153 00 003750

where: 003750 is the machine address of the location in which the Assembly System has stored the RHE of the literal, #001000#.

pseudo-code:

OP	*A*	$N_A N_B$	*B*
TCS	B7		B5

generated machine code:

45 000173 00 000153

Note that the address of the C3 character of the indicated location is always substituted for B1, B3, B5 and B7.

Storing the Contents of Address Modifier Locations

All symbolic address modifier addresses may be used as the B-address of an STR instruction.

If address modifier 4 (P Register) or address modifier 6 (T Register) is specified, a machine STR(73) instruction is generated.

Examples:

pseudo-code:

OP	*A*	$N_A N_B$	*B*
STR	M005000		B4

generated machine code:

73 005000 00 400000

pseudo-code:

OP	*A*	$N_A N_B$	*B*
STR	M003000		B6

generated machine code:

73 003000 00 600000

If the other address modifiers (AM 1, 2, 3, 5 or 7), are entered in the B-address, a machine TCT(25) instruction is generated.

Examples:

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M004000		B1

generated machine code:

```
25     000113     00     004000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M004000		B2

generated machine code:

```
25     000223     00     004000
```

Use of Address Modifiers with TA Instruction

B1, B3, B5 and B7 may also be used as the B-address of the TA (66) instruction.

Example:

pseudo-code:

OP	*A*	N_AN_B	*B*
TA	PAA20		B5

generated machine code:

```
66     022100     00     000153
```

where: 022100 is the machine address the Assembly System has assigned to PAA20.

SPECIAL ADDRESSING OF REGISTERS

The Assembly System provides the user with the ability to address the five registers and the PRI's symbolically. The symbolic addresses of the registers are as follows:

Symbolic Addresses	*Addressed Location*
RP	Register P
RA	Register A
RB	Register B
RS	Register S
RT	Register T
RPRI	Previous Result Indicator
RPRN	Previous Result Indicator Negative
RPRP	Previous Result Indicator Positive
RPRZ	Previous Result Indicator Zero

Use of Symbolic Register Addresses With SET Instruction

The following symbolic register addresses may be used with the SET instruction:

RP
RA
RT
RPRN
RPRP
RPRZ

The format in which the SET instruction is written is as follows:

OP	*A*	N_AN_B	*B*
SET	Quantity to be Placed in Register		Symbolic Address of Register

A-address:

The A-address entry may be an M-address, any symbolic address or a literal.

If an M-address is specified, the actual quantity following the "M" will be placed in the register designated in the B-address when this instruction is executed.

If some symbolic address is specified, the address assigned by the Assembly System to that symbolic address will be placed in the designated register when this instruction is executed.

If a literal is specified, the machine address of the memory location in which the literal is stored (the RHE) is placed in the designated register when this instruction is executed.

If RPRN, RPRP or RPRZ is entered in the B-address, the A-address must be left blank.

B-address:

Either RP, RA, RT, RPRN, RPRP or RPRZ may be entered in the B-address.

Example:

pseudo-code:

OP	*A*	N_AN_B	*B*
SET	#000100#		RP

generated machine code:

```
72        032003        00        400000
```

where: 032003 is the machine address of the location in which the Assembly System has stored the RHE of the literal, #000100#.

pseudo-code:

OP	*A*	N_AN_B	*B*
SET	M000100		RA

generated machine code:

```
72        000100        00        200000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
SET	PAA20		RT

generated machine code:

```
        72          022100          00          600000
```

where: 022100 is the machine address the Assembly System has assigned to PAA20.

pseudo-code:

OP	*A*	N_AN_B	*B*
SET			RPRN

generated machine code:

```
        72          000001          00          100000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
SET			RPRP

generated machine code:

```
        72          000004          00          100000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
SET			RPRZ

generated machine code:

```
        72          000002          00          100000
```

Use of Symbolic Register Addresses With Store (STR) Instruction

All of the symbolic register addresses, except RPRP, RPRN and RPRZ, may be used with the STR instruction in the following format:

OP	*A'*	N_AN_B	*B*
STR	Location in which contents of designated register are to be stored.		Symbolic address of register whose contents are to be stored.

A-address:

An M-address, or any symbolic address may be entered in the A-address of this instruction. This address designates the memory location to receive the contents of the register designated in the B-address.

B-address:

Any one of the symbolic register addresses, except RPRP, RPRN and RPRZ may be entered in the B-address of this instruction.

If RP, RB, RS, RT or RPRI is addressed in the B-address, the Assembly System will generate an STR machine instruction as follows:

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M020000		RP

generated machine code:

```
73      020000      00      400000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M020000		RB

generated machine code:

```
73      020000      00      300000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M020000		RS

generated machine code:

```
73      020000      00      500000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	PAA20		RT

generated machine code:

```
73      022100      00      600000
```

where: 022100 is the machine address the Assembly System has assigned to PAA20.

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M020000		RPRI

generated machine code:

```
73      020000      00      100000
```

If RA is addressed in the B-address, the Assembly System will generate a TCT machine instruction from the STA location as follows:

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M020000		RA

generated machine code:

```
25      000223      00      020000
```

Note that the C3 location of the STA location is addressed.

SYMBOLIC ADDRESSING OF STANDARD LOCATIONS

The Assembly System user may also write symbolic addresses for standard memory locations.

Symbolic Addresses	*Standard Memory Locations*
STP	Standard STP Location (000241 – 000243)
RRAI	Return After Interrupt Setting (000001 – 000003)
RPAJ	Paper Advance Jump Location (000201 – 000203)

Use of Symbolic Standard Memory Locations

The following instructions may symbolically address the standard memory locations in either their A- or B-addresses:

STR
TCT
TCA
TCS

Examples:

pseudo-code:

OP	*A*	$N_A N_B$	*B*
TCT	PAB20		RPAJ

generated machine code:

```
25        023000        00        000203
```

where: 023000 is the machine address the Assembly System has assigned to PAB20.

pseudo-code:

OP	*A*	$N_A N_B$	*B*
TCA	STP		#000010#

generated machine code:

```
44        000243        00        003200
```

where: 003200 is the machine address in which the Assembly System has stored the literal, #000010#.

If a symbolic standard memory address is entered in a STR instruction, the Assembly System generates a TCT instruction as follows:

pseudo-code:

OP	*A*	$N_A N_B$	*B*
STR	PAA20		STP

generated machine code:

```
25        000243        00        022100
```

where: 022100 is the machine address the Assembly System has assigned to PAA20.

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M007000		RRAI

generated machine code:

```
25        000003        00        007000
```

pseudo-code:

OP	*A*	N_AN_B	*B*
STR	M007000		RPAJ

generated machine code:

```
25        000203        00        007000
```

Note that, in all instances, the address of the C3 character of standard memory locations is substituted by the Assembly System for symbolic standard memory location addresses.

Alternate Symbolic Addresses

1. In a symbolic SET or STR instruction the following symbolic addresses are interchangeable:

   B2, BA or RA
   B4, BP or RP
   B6, BT or RT

2. When address modification is specified in the "N" columns, 1, 2, 3, 4, 5, 6, and 7 may be used in lieu of B1, B2, B3, B4, B5, B6, and B7. (Note that this does not apply to the A and B columns.)

VII. MEMORY LAYOUT

RESERVED MEMORY AREAS

The following memory areas are normally reserved, as indicated:

HSM LOCATION	*RESERVED FOR*
000000 – 000277	Standard HSM memory locations
000300 – 001277	Running Program Insertion Routine
001300 – 002507	Rollback Routine
002510 – 002777	Program Sequencer Working Area
003000 – 003777	Sequencer Call Table Area

It is therefore recommended that all programs be assembled starting at memory location 004000, or higher. If lower memory addresses are used certain restrictions are placed on the use of the RCA Program Sequencer.

MEMORY ALLOCATIONS AT ASSEMBLY TIME

A paper call tape is required at assembly time which specifies the beginning and ending memory limits for the program being assembled.

The LHE of memory should be specified as 004000, or higher. (This address must always end with a "zero.")

The RHE is specified as any HSM memory location ending in "7." *Note, however, that if any W1 print areas are defined in the program, this ending address must end in X77, where X is an odd number.*

AUTOMATIC MEMORY LAYOUT

Based on the memory limits specified in the call tape at assembly time, the Assembly System will then allocate memory in the following manner:

The Fixed Constant Area

Starting at the LHE address, the Assembly System will reserve memory for the fixed constants area -- those constants specified with an "F" notation. If fixed constants are not used, two tetrads (starting at the LHE) will be reserved for a dummy fixed constant block – program block 2.

Working Storage

The working storage area is reserved starting at the RHE memory address specified in the call tape.

Before assigning work area addresses, the Assembler first sorts all W-addresses in inverse alphanumerical order, *on the second character.* Thus, all W1-addresses will be grouped in a W1 work-area "block;" all WA-addresses will be grouped in a WA "block," etc.

The W1 "block," if present, is placed at the end of memory (RHE). The W2 "block," if any, will precede the W1 locations, etc. See illustration.

Each working storage "block" will begin in a Co position of a tetrad. However, the order of individual working storage areas within each block depends on the order in which they are defined in the *sorted* program. Therefore, the first WA-area defined will be placed at the *LHE* of the WA block; the second WA-area defined will be placed to the right of the first area, and so on.

Once an individual working storage area has been defined in the program, the programmer may refer to the entire working storage "block." For example, *if one or more WA working storage areas have been defined,* the programmer may clear the entire WA block by the following instruction:

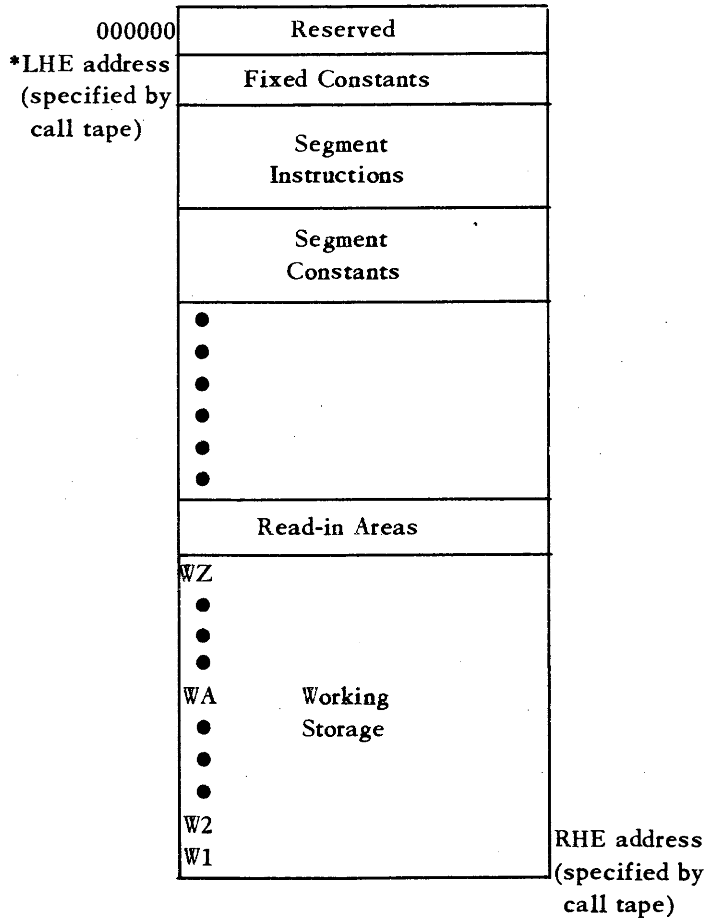
OP	*A*	*B*
SCT	WA	WA

Read-In Areas

Preceding the working storages, read-in areas (F, X, Y, Z) are reserved. Each read-in area is assigned so that the LHE of the area *always* falls in a Co position and the RHE in a C3 position, to accommodate either forward or reverse reads.

The Program

The memory space remaining between the last constant in the fixed constant area and the first read-in area is the memory available for the program. Each segment is considered to consist of the instructions within the segment and the non-fixed constants for that segment. Instructions will be placed into memory starting with the first available last-digit-zero memory location. The segment constants will be placed into memory immediately following the last instruction of the segment. Thus, the automatic memory layout will appear as indicated below:



*LHE should be 004000, or greater.

PROGRAMMER CONTROL OF MEMORY LAYOUT

The following Descriptor Verbs provide the programmer with the ability to control the automatic memory layout:

LEVE - Leave

The LEVE Descriptor Verb permits the programmer to instruct the Assembly System to reserve certain memory locations. The programmer can then make whatever use he wishes of these unassigned areas.

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	LEVE	No. of Characters to be Left Unassigned. (max. = 9999)		File Name or Wa or BOM

Instruction Number:

Enter a DP-address according to the rules of Descriptor Verb instruction numbering.

OP:

Enter the Descriptor Verb operation code, LEVE.

A-address:

Enter the total number of characters to be reserved for the area specified in the B-address. (If the size of the File or working storage "block" is greater than the number in the A-address, the Assembler will assign the larger number to the area).

Enter the total number of characters to be reserved for the area specified in the B-address.

B-address:

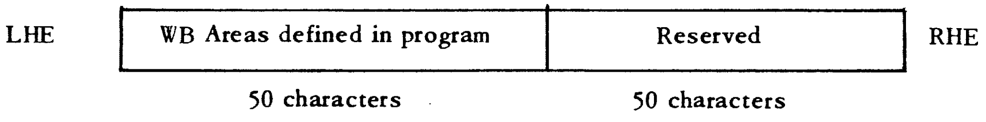
Enter either: File Name, BOM or Wa. *Note that an M-address cannot be used.*

where: File Name is the name of the file whose read-in areas (F, X, Y, and Z areas, if used) will be set at the number of characters specified in the A-address. Note that this use of the LEVE verb may override any length specification for read-in areas made on the data sheets.
BOM specifies the beginning of memory as indicated by the LHE address in the call tape. Thus, if the call tape specified 004000 as the LHE, and 100 is entered in the A-address and BOM is entered in the B-address, memory locations 004000 through 004143 will be left untouched by the Assembly System. The fixed constant area will commence with location 004144.
Wa indicates that a working storage *"block"* length is specified by this LEVE verb, where "a" is the block designator.

If 100 is entered in the A-address and WB is entered in the B-address, the Assembly System will allocate 100 character locations to the WB working storage block. If references to WB addresses within the program specify only 50 characters of working storage, the first 50 characters of the reserved WB area will be used for those working storages specified in the program. The remaining 50 characters will remain untouched by the Assembly System.

INST. NO.	*OP*	*A*	$N_A N_B$	*B*
DPAF22	LEVE	100		WB

WB Working Storage Block



To address the reserved locations in the WB block, the programmer may refer to them as WB(L+50) to WB(L+99) or WB(R) to WB(R−49).

Note:

If the programmer wishes to use the LEVE verb to reserve locations in a working storage block, he cannot make reference to this area by W-name unless at least one work area has been defined in the program within that block.

ASGN – Assign

The ASGN Descriptor Verb provides the programmer with the ability to place constants in the standard memory locations between 000100 and 000277.

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	ASGN	Literal		M-address between M000100 and M000277

Instruction Number:

A DP-address Descriptor Verb instruction number must be entered in the Instruction Number column.

OP:

The Descriptor Verb operation code, ASGN, is entered in the "OP" column.

A-address:

An octal or decimal literal is entered in the A-address. The first character of this literal (LHE) will be stored *starting* at the machine location indicated in the B-column.

B-address:

An M-address between M000100 and M000277 is entered in the B-address column.

The ASGN Descriptor Verb causes the specified literal to be placed in the designated memory location as a result of reading in the program block. This verb can be used to set up various necessary constants in standard memory locations.

Thus, the programmer may assign an initial value of $(000020)_8$ for address modifier 1, by the following instruction:

INST. NO.	*OP*	*A*	$N_A N_B$	*B*
DPAA04	ASGN	#000020#		M000111

Special Note:

An Assign block (Program block 1) will *always* be generated for the object program, regardless of whether or not an ASGN verb was used in the pseudo-coding. If ASGN verbs have not been used, the Assign block will load spaces $(01)_8$ into memory locations 000100 through 000277. When ASGN verbs have been used, the specified literals will be placed in the locations specified, and the remaining locations will be filled with space symbols.

OVLY – Overlay

The OVLY Descriptor Verb provides the programmer with the ability to conserve memory space by directing the Assembler to overlay instructions, read-in areas, and working storage "blocks" with other read-in areas or working storage "blocks."

An example of the use of this verb would be a case where a WA working storage block has been reserved as a result of references to WA working storages. In another part of the program WB working storage references have created a WB working storage block. If the WA and WB areas are never used simultaneously, there is no reason why these two working storage blocks may not occupy the same memory area. The OVLY verb may be used to accomplish this double use of memory area.

Instr. No.	*OP*	*A*	N_AN_B	*B*
DP-address	OVLY	P-address F, X, Y, or Z read-in address or Wa		F, X, Y, or Z read-in address or Wb

Instruction Number:

A DP-address must be entered in the Instruction Number column.

OP:

The Descriptor Verb, OVLY, is entered in the "OP" Column.

A-address:

The entry in the A-address column specifies the area which will be overlaid.

If a P-address, this address indicates the beginning of the program area to be overlaid.

If an F, X, Y, or Z read-in area address, this indicates the particular read-in area to be overlaid.

If Wa, the particular block of working storage to be overlaid is specified where "a" is the second character block indicator.

B-address:

The entry made in the B-address column specifies the overlaying area.

If an F, X, Y, or Z address, this address specifies a particular read-in area.

A Wb entry specifies a particular working storage block.

Examples:

Instr. No.	*OP*	*A*	N_AN_B	*B*
DPKD10	OVLY	WA		WB
DPKD20	OVLY	FSALE		FMASEM
DPKD30	OVLY	PZG50		WK

The following rules must be observed when using the O̲VLY verb:

1. The area which is being overlaid (A-address) must be the same size or greater than the overlaying area (B-address).
2. The programmer may overlay one area with any number of other areas. However, if an area is used as an "overlaying" area it cannot also be used as an "overlaid" area. The following example illustrates these points showing the correct and incorrect methods.

Example:

Programmer wishes to overlay working storage blocks WL, WM, WN and WP; WL is the largest area:

CORRECT

INST. NO.	*OP*	*A*	N_AN_B	*B*
DPZZ10	O̲VLY	WL		WM
DPZZ11	O̲VLY	WL		WN
DPZZ12	O̲VLY	WL		WP

INCORRECT

INST. NO.	*OP*	*A*	N_AN_B	*B*
DPZZ10	O̲VLY	WL		WM
DPZZ11	O̲VLY	WM		WN
DPZZ12	O̲VLY	WN		WP

SGMT – Segment

The SGMT Descriptor Verb provides the programmer with the ability to specify program segmentation.

Instr. No.	*OP*	*A*	N_AN_B	*B*	*T*
DP-address	SGMT	Explicit P-address		P-address ESGMTJ or M-address	J

Instruction Number:

Enter A DP-address in the Instruction Number column.

OP:

Enter SGMT in the "OP" column.

A-address:

Enter the explicit P-address of the first instruction in the segment being defined. Note that a relative P-address cannot be used.

B-address:

Enter the location where the first instruction in this segment is to be placed:

M-address = Any machine location with an address ending in "O."
P-address = P-address of an instruction that appears in another segment.
ESGMTJ = If this segment is to be placed in memory following another segment, enter ESGMTJ, where J is the segment number of the other segment.

T-address:

Enter the segment number assigned to this segment. Segments must be numbered consecutively as they appear in the program. All segments of a program, except for the first segment, must be defined by a SGMT verb. Each segment must be defined individually by a separate SGMT verb.

Example:

Instr. No.	*OP*	*A*	$N_A N_B$	*B*	*T*
DPZM50	SGMT	PBA10		PAA10	2
DPZM60	SGMT	PCA10		PBA10+1	3
DPZM70	SGMT	PDA10		ESGMT3	4
DPZM80	SGMT	PEA10		ESGMT3	5

Use of ESGMTJ

1. In the previous example, it was specified that segment 4 is to be placed in memory following segment 3. The programmer must remember that each segment includes both instructions and constants for that segment. Therefore, these segments will appear in memory as follows:

Segment 3 instructions Segment 3 constants
Segment 4 instructions Segment 4 constants

There is no automatic linkage, then, between the last instruction in segment 3 and the first instruction in segment 4. This must be provided by the programmer.

2. Because of the manner in which the Assembly System processes segment information, the following rule must be adhered to when using ESGMTJ:

The A-address of the SGMT verb must be GREATER than the P-address of the first instruction of Segment "J."

CORRECT

INST. NO.	*OP*	*A*	$N_A N_B$	*B*	*T*
DPZZ10	SGMT	PAL10		PAA10	2
DPZZ11	SGMT	PAM10		PAL10	3
DPZZ12	SGMT	PAR10		ESGMT2	4

INCORRECT

INST. NO.	*OP*	*A*	$N_A N_B$	*B*	*T*
DPZZ10	SGMT	PAL10		PAA10	2
DPZZ11	SGMT	PAM 10		PAL10	3
DPZZ12	SGMT	PAC10		ESGMT2	4

The Effect of Segmentation on Non-fixed Constants, L-Lists and Stashes

Non-fixed K-constants and Stashes:

All non-fixed K-constants and stashes must be defined in the segment in which they are used. The Assembly System determines the segment definition in the following manner: If segment 2 starts at PAL20 and ends at PAN14, all DP-addresses from DPAL20 thru DPAN14, *that define non-fixed K-constants and stashes,* will be considered as applying to segment 2.

If the programmer, then, refers to a particular K-name in segment 2, but the K-constant was not *defined* in the DP-address range for segment 2, it will be considered as undefined for that segment. This may be overcome by making the constant "fixed," or by defining this constant in each segment in which it is used.

Non-fixed Literals:

Normally, non-fixed literals present no segmentation problems since they are written as they are used, and are therefore defined at that time.

However, the positioning of literals within a tetrad may be affected. For example, if #000030#(C1) is used in the first segment, all further references to the same literal are made without C notation. Should the same literal be used in the second segment, *and the "(C1)" notation was **not** specified again,* the constant in the second segment would be considered as *not* having a C-notation.

L-lists:

Once an L-list has been defined in the program, it need not be defined again, regardless of the number of segments. The Assembler will duplicate the address list in all segments which refer to it. The location of the list will ***not*** be the same for all segments since lists are carried in the non-fixed constant section *after* the instructions for that segment.

STRT — Start

The STRT Descriptor Verb informs the Assembly System of the starting point of the program.

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	STRT	P-address of First Instr. to be Executed.		Segment No.

Instruction Number:

Enter a DP-address in the Instruction Number column.

OP:

Enter the operation code STRT in the "OP" column.

A-address:

Enter the P-address of the first instruction to be executed in the subject program.

B-address:

Enter the segment number of the segment in which the first instruction appears. This is a decimal number. The B-address may be left blank if the program consists of only one segment.

The STRT Descriptor Verb must appear once in each program.

VIII. MACRO-INSTRUCTIONS

A macro-instruction is a single instruction designed to perform some task which would normally require the writing of several machine or pseudo-instructions.

A single macro-instruction will generate all of the instructions necessary to perform the specified task. The coding generated by the macro-instruction is an open-ended sub-routine; i.e., there are no special entrance lines and no special exits. This sub-routine is inserted directly into the main body of coding in place of the macro-instruction call-line.

Macro-instructions usually perform some often recurring programming steps. For example, the assembler macro-instruction, ALTP, performs the commonly recurring programming task of switching, i.e., it causes the computer to alternate between two different paths through the program each time it comes to a specified point.

This alternating path function would normally involve the repetitious writing of several lines of coding. However, this routine may be specified by the one macro-instruction.

CALLING MACRO-INSTRUCTIONS

A macro-instruction "call line" is written at the point in the program where the function performed by the macro is desired. The format of this call line is as follows:

INST. NO.	*COMMENTS*	*OP*	*A*	*B*
Explicit P-address		Macro-name (4 characters)		

Instruction Number:

All macro "call lines" must have explicit P-addresses.

OP:

A four-character macro name is entered in the OP column. This name may be alphanumeric.

Other columns:

The macro user enters information in the remaining columns of the call line according to the specifications for that particular macro.

DEFINING MACRO-INSTRUCTIONS

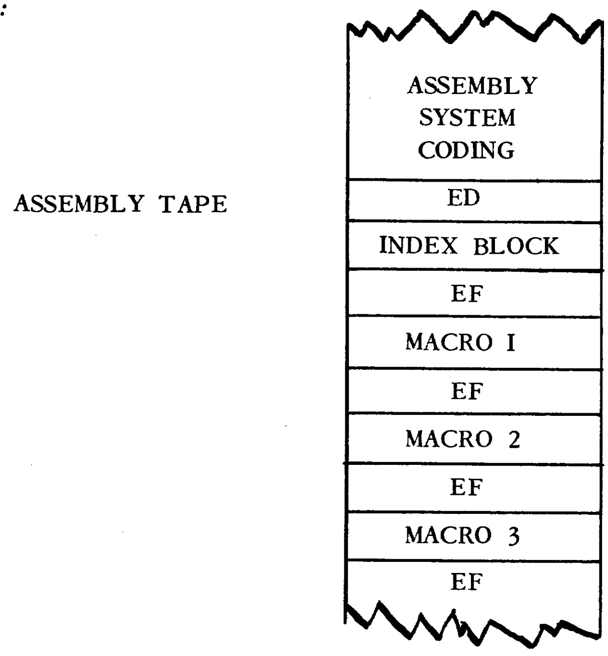
Since an understanding of the method which the Assembly System uses to process macro-instructions is essential to the proper creation of macro-instructions, let us first examine this method.

The Macro-Instruction Library

An ED appears following the main body of coding on the Assembly System Tape. (The phrase, "coding on the Assembly System tape" refers to the actual coding that will perform the function of assembling pseudocode into machine code.) Following this ED is a macro-instruction index block. This index block contains the name and block number of all macro-instructions incorporated in this Assembly System Library Tape.

Following the index block is an EF, followed by all the macro-instruction routines listed in the index block. Each macro-instruction routine is followed by an EF.

Example:



Macro-Instruction Processing

When a macro call-line is encountered during assembly, the index block is searched for that macro-instruction name. Associated with the name, is the macro-instruction block number. The Assembly System is thus referred to the proper macro-instruction block. This macro-instruction block is then read into the computer, and control is transferred to the first preset instruction.

Composition of Macro-Instructions

All macro-instruction routines, as they appear on the library tape, are composed of two parts:

1. Preset
2. Macro-Instruction Symbolic Coding (hereafter referred to as "macro-coding").

The Preset

The preset part of the macro-instruction routine precedes the macro-coding and is the part to which control is immediately transferred when the macro block is read into the computer from the library tape. The preset may have been prepared by the Assembly System when the macro-instruction routine was originally included in the library, or it may have been written by the programmer when the macro-instruction was designed.

If prepared by the Assembly System, the preset will consist of two machine instructions which 1) places the address of the first instruction of the macro-coding into Register T, and 2) transfers control to location 003310$_{(8)}$, where a portion of the Assembly System called the "Operator" is located.

If programmer-prepared, the preset may be written such that it operates on the macro-coding to modify it in some way. These modifications are usually those details necessary to make the general macro-coding specific to the program into which it is to be incorporated. When the programmer writes the preset, he must include in the preset coding the placing of the first macro-coding instruction address into Register T and a transfer of control to the "Operator" 003310$_{(8)}$.

The Operator, having been provided with the address of the first instruction in the macro-coding will then process the special macro-instruction symbolic format into Assembly System pseudo-code format and incorporate it into the program in place of the call-line.

Macro-Instruction Symbolic Coding

The macro-coding section is written in a special format peculiar to macro-instructions. The following rules apply:

1. Only a two-address system may be used, i.e., only the OP, A, N_A, N_B and B columns can be used.
2. The macro-instruction, when originally written, *cannot* contain the following:
   a. literals or K-names.
   b. P-addresses
   c. V-addresses
   d. W-addresses
   e. Descriptor verbs.
   f. DUP, ADV or RES operation codes.
   g. "Call-lines" for subroutines or other macro-instructions.

   Note, however, that "a" through "d" may appear in the call line and can be incorporated in the macro-coding through the special C-notation addressing explained in note 4.
3. If a ST operation code appears in the macro-coding, the Assembly System will *not* recognize this as a program exit line for the RCA Sequencer Routine.
4. The macro-coding section may retrieve data from the "call-line" through a special symbolic addressing system that permits the addressing of individual columns in the call line. This system is illustrated below:

C1	*Not Available*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*	*C13*	*C14*	*C15*	*C16*	*C17*	*C18*	*C19*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

where the information entered in each column may be addressed by the C notation shown. For example, C8 addresses the data appearing in the T column.

5. All references in the macro-coding section to other parts of that section must be C1 relative. C1(8), for example, would address the OP code of the second instruction in the section.
6. After the last instruction in the macro-coding, the word END must be entered in the next OP column. Note that the END line does not generate an instruction in the object program.

The Define Macro-Instruction Descriptor Verb

The Define Macro-Instruction Descriptor Verb, DEFM, is used to enter macro-instructions into the macro-instruction library. The use of this verb only defines the macro onto the library tape, it does not generate instructions in the object program. To include the macro-coding as part of a program, a "call-line" must be used. The general format of the DEFM verb is as follows:

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	DEFM	Macro Name		S or P

Instruction Number:

A DP-address must be specified in the instruction number column.

OP:

The operation code DEFM is entered in the "OP" column.

A-address:

The name which is to be assigned to this macro-instruction is entered in the A-address. Note that this must be a four-character name which must be different from all other macro-instructions.

B-address:

Enter either S or P. S means that the programmer has written no special preset part. This is a Simple Substitution type of macro-instruction. A two line preset for this macro-instruction will be generated by the Assembly System before inclusion in the library.

P means that the programmer has written his own special preset for this instruction. No preset will be generated by the Assembly System.

Note that the first pseudo-instruction following the definition of the macro (i.e., the instruction after the END line) must have an explicit P or DP-address.

Example:

The following example shows the DEFM coding to incorporate the macro-instruction, ALTP, onto an Assembly System Library tape.

DEFM coding:

Instr. No.	*OP*	*A*	N_A	N_B	*B*
DP-address	DEFM	ALTP			S
	TA	C4	C5		C1(23)
	SCC	C1(23)			C1(23)
	TC	C7	C6		M000001
	END				

When this Descriptor Verb is encountered during Assembly, the new ALTP macro-instruction name will be added to the index together with the block number that will be assigned to this macro-instruction routine.

Because an S is specified in the B-address of the DEFM verb, two lines of preset coding (preceded by the macro-instruction name) will be generated, and prefixed to the symbolic coding. The entire routine – name, preset, and symbolic coding – will then be stored in the library, and two copies of the updated tape will be generated.

The form in which this macro-instruction is stored in the library is as follows:

OP	*A*	N_A	N_B	*B*
ALTP				
72	000010	4	0	600000
71	003310	0	0	000000
TA	C4	C5		C1(23)
SCC	C1(23)			C1(23)
TC	C7	C6		M000001
END				

Now, let us assume that an ALTP call-line in the following format is encountered somewhere within the program:

C1	*C3*	*C4*	*C5*	*C6*	*C7*
Instr. No.	*OP*	*A*	N_A	N_B	*B*
P-address	ALTP	P1	N1	N2	P2

Having encountered this call-line, the Assembly System will search the index for ALTP and will be referred to the block containing the ALTP macro-instruction routine.

The routine, as shown above, will be read into the computer and control will be transferred to the first instruction of the routine. This will be the (72)$_8$ instruction of the Assembly System generated preset. The (72)$_8$ instruction will place the address of the first symbolic macro-instruction (the TA instruction) in Register T. The (71)$_8$ instruction will then transfer control to memory location 003310, the address of the first instruction of the Operator.

Having been provided with a starting point, stored in Register T, the Operator will then scan the symbolic coding, looking for symbolic C notations. For each C notation found, the contents of that column in the call-line will be substituted for the C notation appearing in the symbolic instruction. This will continue until the END in the ''OP'' column is encountered. The operator then inserts the modified macro-coding section into the pseudo-program in place of the call-line. The entire sequence of coding in all its phases is presented below so that the reader can see which operations occur during each phase.

Define Macro-Instruction Coding:

Instr. No.	*OP*	*A*	N_A	N_B	*B*
DP-address	DEFM	ALTP			S
	TA	C4	C5		C1(23)
	SCC	C1(23)			C1(23)
	TC	C7	C6		M000001
	END				

As Placed in the Library:

ALTP			
72	000010	40	600000
71	003310	00	000000
TA	C4	C5	C1(23)
SCC	C1(23)		C1(23)
TC	C7	C6	M000001
END			

If the Following Call-Line is Encountered:

C1	*C3*	*C4*	*C5*	*C6*	*C7*
PAB10	ALTP	PAA10	B1	B3	PDQ40+8

The Following Pseudo-Code is Generated at PAB10:

INST. NO.	*OP*	*A*	N_A	N_B	*B*
PAB10	TA	PAA10	B1		PAB10(23)
	SCC	PAB10(23)			PAB10(23)
	TC	PDQ40+8	B3		M000001

Note from the above example that:

1. The Operator substitutes for the C notations the information entered in the indicated column of the call-line.

Parenthetical expressions that follow the C notations are appended to the entries taken from the call-line. For example, in the SCC instruction C1(23) appeared in the A-address.The Operator substituted for C1 the information in the Instruction Number column of the call-line, i.e., PAB10. The modified A-address then became PAB10(23).

The symbolic macro-instruction coding must therefore be such that when entries in the macro-instruction call-line are substituted for corresponding C notations, the result is a two-address system of coding whose addresses are symbolic addresses recognizable to the Assembly System.

2. Since the P-address of the call-line becomes the P-address of the first instruction of the generated macro-instruction, reference may be made to other parts of the macro-coding by addressing them character relative to the call-line P-address. This is done by using the C1(n) notation, where C1 will be replaced by the call-line P-address, and (n) designates the position of the addressed character as it will appear in the final generated coding.

Defining Macro-Instructions With Presets

Let us look at a possible macro-instruction, the TEST macro-instruction. It would have the following call-line:

Instr. No.	*OP*	*A*	N_A	N_B	*B*	*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
C1	*C3*	*C4*	*C5*	*C6*	*C7*	*C11*	*C12*	*C13*	*C14*	*C15*	*C16*	*C17*	*C18*	*C19*
P-address	TEST	FILE NAME				SM	P1	N1	EF	P2	N2	ED	P3	N3

where: 1. The P-address associated with the call-line is entered in C1.

2. The macro-instruction name, TEST, is entered in C3.

3. The name of the file whose first character is to be tested is entered in C4.

4. SM is entered in C11.

5. The address to which control is to be transferred (P1) if the first character is found to be SM is entered in C12.

6. The address modifier location to be used to modify P1, if any, is entered in C13.

7. An EF is entered in C14.

8. The address (P2) to which control is to be transferred if the first character of the file is found to be an EF is entered in C15.

9. The address modifier location to be used to modify P2, if any, is entered in C16.

10. An ED is entered in C17.

11. The address (P3) to which control will be transferred if an ED is found to be the first character is entered in C18.

12. The address modifier location to be used to modify P3, if any, is entered in C19.

This macro-instruction would be incorporated into the macro-instruction library by the following DEFM verb:

Instr. No.	*OP*	*A*	N_A	N_B	*B*
DP-address	DEFM	TEST			S
	SET	C4(L)			M607300
	SC	C1(6)			C1(6)
	CTC	C15	C16	C13	C12
	TC	C18	C19		
	END				

It will be incorporated into the library in the following format:

OP	*A*	N_A	N_B	*B*
TEST				
72	000010	4	0	600000
71	003310	0	0	000000
SET	C4(L)			M607300
SC	C1(6)			C1(6)
CTC	C15	C16	C13	C12
TC	C18	C19		
END				

If, later in the same program in which the TEST was defined, or in some other program, the following call-line is encountered:

Instr.No	*OP*	*A*		*IF*	*GO TO*	N_1	*IF*	*GO TO*	N_2	*IF*	*GO TO*	N_3
C1	*C3*	*C4*		*C11*	*C12*	*C13*	*C14*	*C15*	*C16*	*C17*	*C18*	*C19*
PAB10	TEST	FMASEM		SM	PSM10	B1	EF	PEF10		ED	PED10	

The coding generated by the operator and substituted for the call-line will appear as follows:

Instr. No.	*OP*	*A*	N_A	N_B	*B*
PAB10	SET	FMASEM(L)			M607300
	SC	PAB10(6)			PAB10(6)
	CTC	PEF10		B1	PSM10
	TC	PED10			
	END				

The macro-instruction creator, however, may wish to generalize the macro-instruction still further. He may wish to allow the user the option of specifying either SM or EM in C11, where an EM specifies that the read wās a read reverse. In order to accomplish this, the macro-instruction writer must provide a set of coding which tests for the presence of EM in C11. If EM is specified, the macro-instruction coding must be revised to test the right-hand-end of the area instead of the left-hand-end.

This will have to be done *before* the Operator translates the symbolic macro-instruction into Assembly System pseudo-code, and the coding to accomplish this must not itself be incorporated into the pseudo-program. This is the role of the preset. The preset, therefore, must have the ability to refer to the call-line, the symbolic macro-instruction coding, and to other parts of itself.

THE PROGRAMMER-WRITTEN PRESET

When the preset is provided by the programmer, the following rules must be observed:

1. On the DEFM line a "P" must be written in the B column to indicate that the preset is written by the programmer.
2. The Preset block of coding is written in strict machine format. (The Operation code and addresses are *NOT* to be preceded by "M".)
3. The first instruction in the preset will be read into HSM location $011010_{(8)}$. Other parts of the preset, or instructions in the macro-coding, are written relative to this location.
4. The preset coding must place into Register T the address of the SM symbol of the first macro-coding instruction, and must also transfer control to the operator, located at $003310_{(8)}$.

5. The Assembler reads the macro "call-line" into location $035700_{(8)}$. See figure A.

6. The location of the symbolic instructions in the macro-coding will vary according to the size of the preset. The SM of the *first* instruction, however, will always immediately follow the last preset line. The SM of the next instruction will be 100 octal locations greater. (To determine the starting address of subsequent instructions, the programmer adds $000100_{(8)}$ to each SM location.) The format of a macro-coding line, as it is placed in memory, is shown in Figure B.

To illustrate the preceding rules, let's examine a more general version of the TEST macro defined below:

	INST. NO.	*COMMENTS*	*OP*	*A*	N_A	N_B	*B*
	DP-address		DEFM	TEST			P
011010	PAZ00		72	011017	0	0	604454
011020			43	036031	0	0	036032
011030			61	011050	0	0	011050
011040			22	011056	0	0	011125
011050			72	011100	0	0	606100
011060			71	003310	0	0	000000
011070			00	000000	0	0	000000
011100			SET	C4(L)			M607300
011200			SC	C1(6)			C1(6)
011300			CTC	C15	C16	C13	C12
011400			TC	C18	C19		
011500			END				

Notes:

A. The first two preset instructions test the first IF column in the call-line for the characters EM. If not EM the macro-coding remains unchanged. If EM appears, an "R" is substituted for the "L" in the first macro-coding line.

B. After the call-line has been tested, and the macro-coding modified (if necessary), the location of the first line of the macro-coding is placed in the T register and control transferred to the Operator. The Operator will then substitute data in the call-line for all C notations, and will incorporate the modified pseudo-coding into the program.

C. It should be noted that a dummy instruction was added to the preset so that the first instruction of the macro-coding would begin at a location ending in "00". Although this is not required, it will make it easier for the programmer (when using the format in figure B) to determine the exact machine locations in a symbolic instruction line.

D. For reference ease, the macro writer should list the HSM addresses of each preset instruction and macro-coding instruction to the left of the Inst. No. column. (This information is not to be punched.)

E. A dummy instruction number, PAZ00, was used in the first preset line to facilitate possible corrections at the time the macro-instruction is defined. For example, if the programmer wishes to change the second line of the macro-coding during definition, he could refer to that line as PAZ00+8.

If desired, a dummy P-address may also be assigned to the first macro-coding instruction. These P-addresses, however, will not appear in the macro as placed on the library tape.

MACRO-INSTRUCTION SEGMENTATION

If the coding for a macro-instruction will extend beyond $037777_{(8)}$, it must be segmented.

Segmentation is accomplished by a special line in which "LIM" appears in the "OP" column and XXXXXX (a machine address) in the A-address column.

	INST. NO.	*COMMENTS*	*OP*	*A*	N_A	N_B	*B*
	DP-address		DEFM	NAME			P
011010	PAZ00		71	011040	0	0	000000
011020			22	000215	0	0	011035
011030			15	011100	0	0	000000
011040			72	011100	0	0	600000
011050			71	003310	0	0	000000
011060			00	000000	0	0	000000
011070			00	000000	0	0	000000
011100	PAZ01		SCC				
011200			IT				
037400			TCT				
037500			LIM	011020			
011100	PAZ02		DA				
011200			DS				
011300			IT				
011400			END				

In the above example the programmer segmented the macro-coding after the TCT instruction. When the "Operator" reads in the macro-instruction from the library the ***first*** block (up to and including the LIM line) will be brought into memory. The first preset instruction is then performed.

When the "Operator" encounters the LIM notation, the macro-coding processed up to that point is incorporated into the pseudo-code program. Control is then transferred to the address (XXXXXX) entered in the A-column. The address in the A column ***must*** be some instruction in the preset; control is usually transferred to that part of the preset which reads in the next segment from the library tape. In this example, a One Character Transfer was made fromlocation $000215_{(8)}$ to the B-address of the Block Read instruction. Note that the Assembler Library tape is always located at $000215_{(8)}$.

Special Notes

1. Each macro instruction segment may contain its own preset instructions, if desired.
2. The XXXXXX in the LIM line ***must*** ***not*** be preceded by an "M".
3. Only the first segment of a macro-instruction is brought in by the Assembler. Additional segments must be read in by instructions in the preset.

THE REPLACE MACRO-INSTRUCTION DESCRIPTOR VERB

Just as it was shown that the first TEST macro-instruction written was later improved by making it more general, so it is expected that macro-instructions will constantly be improved. To facilitate the replacement of obsolete macro-instructions with improved versions, the Assembly System user is provided with the following Descriptor Verb.

Instr. *No.*	*OP*	*A*	N_A N_B	*B*
DP-address	REPM	Macro- Name		S or P

Note that, in the B-address, an S is entered if the new version is a simple substitution type; a P is entered if the new version contains a programmer-written preset, regardless of what type of macro-instruction is being replaced. Following the REPM verb the new version of the macro-instruction is entered in exactly the same format as is used with the DEFM verb.

The REPM verb will cause the Assembly System to delete the macro-instruction listed in the library under the name indicated in the A-address of the REPM verb line. The new version, as written following the REPM verb line will be inserted in its place. The next instruction in the program must have an explicit P-address.

The REPM verb will also cause the Assembly System to generate two new updated library tapes.

THE DELETE MACRO-INSTRUCTION DESCRIPTOR VERB

The ability to delete macro-instructions from the library is provided by the DELM Descriptor Verb:

Instr. No.	*OP*	*A*	$N_A N_B$	*B*
DP-address	DELM	Macro-Name		

The DELM verb causes the Assembly System to delete the named macro-instruction from the library and from the index block. Two updated library tapes are generated as a result of the DELM function.

FIGURE A.

THE MACRO-INSTRUCTION CALL-LINE IN MEMORY

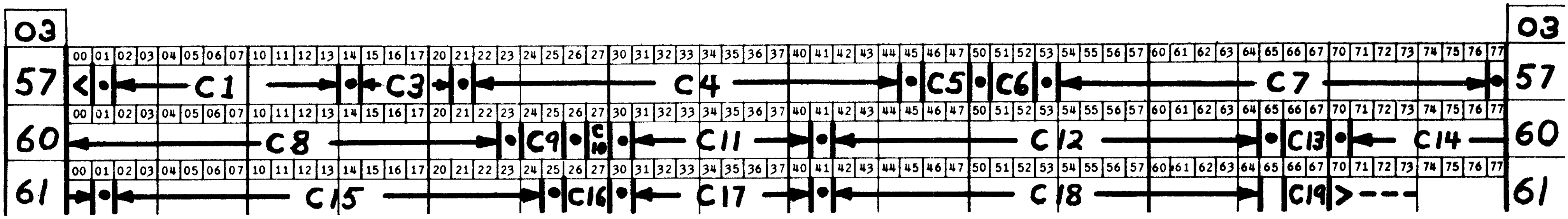
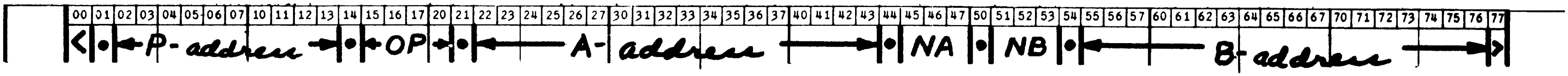


FIGURE B.

FORMAT OF A LINE OF MACRO-INSTRUCTION SYMBOLIC CODE IN MEMORY



IX. SUBROUTINES

Subroutines differ from macroinstructions in two respects. First, subroutines create coding in full Assembly format. That is, they do not turn out two-address symbolic coding as do macroinstructions, but create coding in the expanded pseudo-code form. Consequently, all Assembly features are available to the subroutine in so far as subroutine instructions may include Descriptor Verbs, macroinstructions and the like. Second, a subroutine may be an "open" or "closed" routine. That is, it may be inserted into the pseudo-code program each time the function is desired ("open"); or it may be inserted once, with references in the program to the subroutine being made by transferring to it ("closed").

CALLING SUBROUTINES

The programmer may extract a subroutine from the Assembly Library tape and enter it in his program through a subroutine "call-line." If the subroutine will be performed in the program by entering it from the coding preceding the subroutine, it is considered an "open" subroutine. If, on the other hand, the subroutine will be entered via a transfer of control, it is considered a "closed" subroutine. As explained later in this chapter, all subroutines are originally *written* as "closed" subroutines. The determination of whether a subroutine will be *used* in the open or closed form is made from the format of the subroutine call-line.

Call-lines for open subroutines are written in the following format:

1. An explicit SP-address must be placed in the instruction number column. (The Assembly System will remove the "S" from this address, and will assign the remaining P-number to the first line of the subroutine coding.)

   The last two characters of the SP-address *must always* be 00. For example:

   SPAB00

   The P-address, then, of the first line of the subroutine included at this point will be PAB00.

2. The Comments column must be left BLANK.
3. The subroutine name (4 alphanumeric characters) is entered in the OP column.
4. The remaining columns on the call-line are used to supply the subroutine with various data (called "parameters"). Any number of lines following the call-line may be used to specify additional parameters. These lines must be identified as parameter lines by inserting "PARA" in the OP column of each line. (Instruction numbers are *not* to be used in the Inst. No. columns of these PARA lines.)

   Note: The subroutine write-up will specify what information is to appear in the call-line and PARA lines.

5. The instruction following a subroutine call-line must have an explicit P-address that is 100 greater than that of the call-line.
   For example:

SPAB00	Subroutine call
PAC00	Next instruction

Call-lines for closed subroutines are identical to those for open subroutines, with one exception. The Comments column of the call-line must contain the words:

INCLUDE+COPY+n

where "n" is a number, ranging from 1 to 9, that identifies the particular subroutine copy that has been entered in the pseudo-code program. Suppose, for example, that the same subroutine may take several forms according to the parameters given in the call-line. Each new form must be identified by a different "n" in its call line. INCLUDE+COPY+1 is used for the first version, INCLUDE+COPY+2 for the second, and so on.

Example A: Calling in an "open"subroutine from Library

INST. NO.	*COMMENTS*	*OP*	*A*	*B*
SPAD00		SUB1		
PAE00		LW	KNAME	T77

Example B: Calling in a "closed" subroutine from Library

INST. NO.	*COMMENTS*	*OP*	*A*	*B*
SPAD00	INCLUDE+COPY+1	SUB1		
PAE00		LW	KNAME	T77

ENTERING SUBROUTINES

Since "open" subroutines are placed in the coding at the point where they will be executed, no special provision has to be made to enter them. "Closed" subroutines, however, are entered from other points in the program. To facilitate this procedure, an entrance line to a closed subroutine is written in the following format:

1. An explicit SP-address must be placed in the Instruction Number column. *The next pseudo-code line must have an explicit P-address.*
2. The term COPY+n must be placed in the Comments column, where "n" indicates the particular subroutine copy that the programmer wishes to enter at this point. Care should be taken that the proper copy of the subroutine will be in memory when this entrance line is executed in the object program.
3. The four-character subroutine name is entered in the OP column.
4. No other entries may be made on the entrance line.

When this entrance line is encountered during Assembly, the following events occur:

a) "S" is removed from the SP-number.

b) "COPY-n" is deleted from the Comments column.

c) "TC" is substituted for the subroutine name in the OP column.

d) The P-address of the first line of the requested subroutine copy is inserted in the A-column. This address depends on the subroutine name in the OP.column and the COPY-n notation in the Comments column. It is important, therefore, that these entries be accurate.

Example:

Entrance line:

INST. NO.	*COMMENTS*	*OP*	*A*	*B*
SPAN24	COPY+1	SUB6		

As Modified by Assembly System:

INST. NO.	*COMMENTS*	*OP*	*A*	*B*
PAN24		TC	PAB00	

where PAB00 is the location of the first instruction in the subroutine coding.

CONSTRUCTION OF SUBROUTINES

General Rules

Certain general rules apply to subroutines that are to be used with theAssembly System. First, subroutines are written in Assembly System language. However, an imposed restriction is that subroutine call-lines may not be included within subroutines. Otherwise, macro-instructions, Descriptor Verbs, variable addresses, etc., may all be used.

A second rule concerns the fact that subroutines must be written as "closed." This means that the first pseudo-instruction in all subroutines must store the return point (TCT of STP to AM 7); and the last pseudo-instruction must provide an exit from the subroutine (TC, modified by AM 7).

1st instruction:

OP	*A*	N_A	N_B	*B*
TCT	STP			B7

last instruction:

OP	*A*	N_A	N_B	*B*
TC		B7		

If the subroutine user desires an "open" routine (i.e., leaves the Comments column of the call-line blank), the Assembly System automatically deletes the first and last instructions before incorporating the subroutine-coding into the program. If "INCLUDE+COPY+n" appears, these instructions remain unchanged.

Composition of Subroutines

All subroutines, as they appear on the library tape, are composed of two parts:

1. Preset
2. Subroutine Symbolic Coding (hereafter referred to as "subroutine-coding").

The Preset

The preset part of the subroutine precedes the subroutine-coding and is the part to which control is immediately transferred when the routine is read into the computer from the library tape. The preset may have been generated by the Assembly System when the subroutine was included in the library, or it may have been written by the programmer when the subroutine was designed.

If prepared by the Assembly System, the preset will consist of two machine instructions which 1) places the address of the first subroutine-coding instruction into Register T, and 2) transfers control to location $004200_{(8)}$, where a portion of the Assembly System called the "Operator" is located.

If programmer-prepared, the preset may be written such that it operates on the subroutine-coding to modify it in some way. These modifications are usually those details necessary to make a general subroutine-coding specific to the program into which it is to be incorporated. *When the programmer writes the preset, he must include in his preset coding the placing of the first subroutine-coding instruction address into Register T and a transfer of control to the "Operator" at* $004200_{(8)}$.

The Operator, having been provided with the address of the first instruction of the subroutine-coding, then processes the subroutine symbolic format into Assembly System pseudo-code format and incorporates it into the program in place of the call-line.

Using the Call-Line and PARA Lines as Data for Subroutine

The subroutine-coding section may use the call-line and PARA lines as data through the use of a special symbolic addressing system illustrated below. Note that a similar system was used for macro-instructions.

Call line:

C1	*Not Available*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*	*C13*	*C14*	*C15*	*C16*	*C17*	*C18*	*C19*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

1st PARA line:

C20	*C21*	*C22*	*C23*	*C24*	*C25*	*C26*	*C27*	*C28*	*C29*	*C30*	*C31*	*C32*	*C33*	*C34*	*C35*	*C36*	*C37*	*C38*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

2nd PARA line:

C39	*C40*	*C41*	*C42*	*C43*	*C44*	*C45*	*C46*	*C47*	*C48*	*C49*	*C50*	*C51*	*C52*	*C53*	*C54*	*C55*	*C56*	*C57*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

3rd PARA line:

C58	*C59*	*C60*	*C61*	*C62*	*C63*	*C64*	*C65*	*C66*	*C67*	*C68*	*C69*	*C70*	*C71*	*C72*	*C73*	*C74*	*C75*	*C76*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

4th PARA line:

C77	*C78*	*C79*	*C80*	*C81*	*C82*	*C83*	*C84*	*C85*	*C86*	*C87*	*C88*	*C89*	*C90*	*C91*	*C92*	*C93*	*C94*	*C95*
Instr.	Comments	OP	A	NA	NB	B	T	NT	CSG	IF	GO TO	N_1	IF	GO TO	N_2	IF	GO TO	N_3

The information in these columns may be addressed by the C-notation shown. For example, C8 addresses the data in the T-column of the call line; C61 addresses the data in the A-column of the third PARA line. Parenthetical expressions following C-notations will be appended to entries taken from the call-line. Thus, if C8(L+6) appears in the subroutine-coding, and WPAY appears in the T-column (C8) of the call-line, the Operator will substitute WPAY (L+6) in the subroutine-coding.

Note that the Comments column of the call-line, and the Inst. No. and OP columns of ALL lines, are not available for data (parameters). In addition, the subroutine writer cannot use C notation in the Inst. No., Comments or OP columns of instructions in the subroutine-coding section.

If more than five lines are required for parameters, additional PARA lines may be used. However, the Assembler will only process five lines at a time, and it will be necessary for the programmer to call in additional groups of PARA lines when needed. This is accomplished by the following machine instruction which must be written in the preset:

OP	*A*	*N*	*B*
71	010320	00	000000

Location 010320 is a part of the Operator that will bring in the next group of PARA lines. After this is done, the Operator returns control to the next preset instruction. When subsequent groups of PARA lines are brought into memory, they will *overlay* the previous parameter lines and will be addressable by C1 to C95.

The Define Subroutine Descriptor Verb (DEFS)

A subroutine may be defined anywhere in the program, but it must be remembered that "defining" it does not incorporate it into the object program. This is done through the call-line described previously. The following format is followed when defining a subroutine onto the Assembly Library Tape:

INST. NO.	*OP*	*A*	N_AN_B	*B*
DP-address	DEFS	Subroutine name (4 characters)		S000 or P000

Instruction Number:

A DP-address must be specified.

OP:

The Operation code DEFS is entered.

A-address:

The name of this subroutine is entered; the name may be alphanumeric, and must be four characters.

B-address:

Enter either S000 or P000. S means that the preset is to be generated by the Assembly System. P means that the programmer is providing the preset. "000" is used if the subroutine will *not* use "dynamic" parameters. Dynamic parameters are discussed in a later section.

On the following lines the coding for the subroutine is written.

Rules Governing Subroutine Symbolic Coding Section

All Assembly features, except a subroutine call-line, may be used in the subroutine-coding section. In addition:

1. The word END must appear in the OP column immediately following the exit TC instruction. This END line will not generate an instruction in the object program.
2. The first instruction (i.e., TCT of STP to B7) *must* have an address of PSR00. All other instructions in this section must be written relative to PSR00, with the highest usable address, PSR19+99.

For example:

INST. NO.	*COMMENTS*	*OP*	*A*	N_A	N_B	*B*	*T*
PSR00		TCT	STP			B7	
		TCA	PSR01(B)			PSR01+1(B)	
PSR01		SCC	C12			C12	
		TC		B7		M000006	
		END					

As stated previously, one of the functions of the "Operator" is to substitute data for the C-notations. In addition, the Operator will also change all PSR-addresses to the address (minus S) that appears in the call-line. Thus, if a programmer calls for this subroutine at SPAM00, all PSR-addresses in the above example will be changed to PAM-addresses by the Operator.

Sample Subroutine Definition

INST. NO.	COMMENTS	OP	A	N_A	N_B	B	T
DPAC18		DEFS	RWND			S000	
PSR00		TCT	STP			B7	
		RWD				C7	
		RWD				C8	
PSR10		RWD				C12	
		TC		B7			
		END					

The Programmer-Written Preset

When the preset is provided by the programmer, the following rules must be observed:

1. On the DEFS line a "P000" must be written in the B-column to indicate that the preset is supplied by the programmer. (See dynamic parameters for significance of "000".)
2. The Preset block of coding is written preceding the subroutine-coding in strict machine format. (The Operation code and addresses are *NOT* to be preceded by "M".)
3. The preset coding and the subroutine-coding are written relative to memory address 000000. *The Assembly System will automatically place into AM1 the location of where the first instruction in the preset was placed.* Therefore, to specifically address other instructions in the preset, or parts of the subroutine-coding lines, the relative addresses must be modified by AM1.
4. The preset coding must place into Register T the address of the SM symbol of the first subroutine-coding instruction, and must also transfer control to the Operator, located at $004200_{(8)}$.
5. The call-line and PARA lines are always read into memory starting at location $013700_{(8)}$. To determine actual machine addresses for information in these lines, refer to Figure B, page 110.
6. The first instruction of the subroutine symbolic coding begins immediately following the last preset instruction. Thereafter, the SM symbol for each succeeding pseudo-line will be $(400)_8$ locations greater. See format of symbolic lines in Figure A, page 109.

After the subroutine has been brought into memory from the Library tape, control is immediately transferred to the first instruction in the preset. Since the preset is written in strict machine-coding, the subroutine writer may use his preset coding to directly address information in the call-line and PARA lines; he may also address other parts of the preset and subroutine-coding lines by modifying zero-relative addresses by AM1.

To illustrate these various addressing systems, let us define a hypothetical subroutine, OCTS, that is designed to allow the user to transfer one character to two destinations, or to transfer one character to five destinations. The user must include the following data (parameters) in the call-line:

C4 (A-address) = location of character to be transferred.
C5 (N_A) = enter a "2" or "5" to indicate the number of transfers.
C7 (B-address) = 1st destination location
C26(B-address) = 2nd destination location
C45(B-address) = 3rd destination location
C64(B-address) = 4th destination location
C83(B-address) = 5th destination location

The following subroutine is then defined:

	INST. NO.	COMMENTS	OP	A	NA	NB	B	T
	DPAF14		DEFS	OCTS			P000	
000000	PAA20		72	000007	1	0	600025	
000010		Check for 2.	43	014044	0	0	014044	
000020		No modification	61	000050	1	1	000050	
000030			72	002477	1	0	600000	
000040		Move TC-END lines	26	003100	1	1	004077	
000050			72	000100	1	0	600000	
000060		TC to Operator	71	004200	0	0	000000	
000070			00	000000	0	0	000000	
000100	PSR00	Set exit	TCT	STP			B7	
000500		to 1st location	OCT	C4			C7	
001100		to 2nd location	OCT	C4			C26	
001500	PSR10	to 3rd location	OCT	C4			C45	
002100		to 4th location	OCT	C4			C64	
002500	PSR15	to 5th location	OCT	C4			C83	
003100		Exit	TC		B7			
003500			END					

The first preset instructions test the character in the NA column of the call-line for a "2." If a "2" appears, the TC and END lines of the subroutine-coding are moved up to the lines following the second OCT instruction. If a numeric two is not present, the subroutine-coding remains unchanged.

The "Operator" is given the address of the first pseudo-line and control transferred to it. The Operator then substitutes data in the parameter lines for the C-notations, and changes all PSR-addresses to the address in the call-line. When END is encountered, the Operator inserts the modified subroutine-coding into the program.

It should be noted that a dummy preset instruction was added so that the beginning address of each pseudo-line would end in "00." Also, a dummy P-address was used in the first preset line to facilitate possible corrections during definition.

For reference ease, the subroutine writer should list the *relative* addresses of each preset and pseudo-instruction to the left of the INST. NO. column. (This information is not to be punched.)

SUBROUTINE SEGMENTATION

If the coding for a subroutine will extend beyond the *relative* address of 021060, it must be segmented. Segmentation is accomplished by a special line in which LIM appears in the "OP" column and XXXXXX (a relative machine address) in the A-address column. For example:

	INST. NO.	COMMENTS	OP	A	N_A	N_B	B	T
	DP-address		DEFS	NAME			P000	
000000	PAA00		71	000030	1	0	000000	
000010			22	000215	0	1	000025	
000020			15	000100	1	0	000000	
000030			72	000100	1	0	600000	
000040			71	004200	0	0	000000	
000050			00	000000	0	0	000000	
000060			00	000000	0	0	000000	
000070			00	000000	0	0	000000	
000100	PSR00		TCT	STP			B7	

017500	PSR14		SCC	C12		C12	
020100			LIM	000010	1		PSR10(B)
000100			TCT	#000030#(C1)			
000500			TC		B7		
001100			END				

In the above example the writer had to segment the subroutine-coding after the SCC instruction. When the "Operator" reads in the subroutine-coding from the Library the *first* block (up to and including the LIM line) is brought into memory. The first preset instruction is then executed.

When a LIM notation is encountered by the Operator, the symbolic coding up to that point is incorporated into the program. Control is then transferred to the address (XXXXXX) entered in the A-address. This address *must* be some instruction in the preset; it is usually that part of the preset that reads in the next segment from the Assembler Library. *Note that the Assembly tape is always located at* $000215_{(8)}$.

Special Notes

1. Each subroutine segment may contain its own preset, if desired.
2. The XXXXXX in the LIM line *must not* be preceded by an "M".
3. A maximum of 35 pseudo-code lines (including the LIM line) can appear in a segment that does not include preset instructions.

DYNAMIC PARAMETERS

Information may be captured from the call-line and PARA lines when a subroutine is called in from the Assembly Library. Since these parameters are only available to the subroutine at this time, they are referred to as "static" parameters. However, it is often desired to use parameters *during the running program*; this is the function of "dynamic" parameters.

"Dynamic" parameters are written immediately following the <u>entrance</u> line to a "closed" subroutine, i.e., after the "COPY+n" line. For example:

INST. NO.		*OP*	*A*	*B*
SPAT43	COPY + 1	SUB1		
PAT44		G00	WPAY(L)	WPAY(R)
		G00	WNET(L)	WNET(R)
		G00	M001420	WAMT(R)

The OP column of these lines contain "G" operation codes and parameters may appear in both the A and B columns. The first G — line *must* have an explicit P-address.

When "dynamic" parameters lines are used, the subroutine writer must indicate this when the subroutine is defined. In the preceding example, then, S003 or P003 would have been entered in the B-address of the DEFS line. This informs the "Operator" that three parameter lines appear following all entrance lines. The Operator, therefore, will ***automatically modify the subroutine exit line*** (TC) ***to return control to the fourth line after the entrance line.***

If the subroutine uses dynamic parameters, the user *must* always include these lines following all "COPY-n" lines. The data in the parameter lines may, of course, vary according to program requirements. The user must also remember that dynamic parameter lines will appear in his object program.

Dynamic parameters are incorporated into the subroutine through coding appearing within the subroutine.

This information is always addressed relative to AM7, which contains the location of the first G—— line. For example, if a subroutine instruction requires the address appearing in the B-column of the second G—— line, the following instruction may appear in the subroutine:

OP	*A*	N_A	N_B	*B*
TCT	M000013	B7		PSR03(A)

Since dynamic parameters are brought into the subroutine coding at *program running time,* the subroutine writer must appreciate the fact that HSM addresses are being transferred, *not* symbolic names.

General Comments regarding Dynamic Parameters lines:

1. Dynamic parameters may only be used with "closed" subroutines.
2. Dynamic Parameter lines must have "G" operation codes, and these lines will appear as machine instructions in the object program.
3. Parameters must be addressed relative to AM7.
4. The DEFS line must specify (after S or P), the number of dynamic parameter lines.

SPECIAL OPTION FOR ADDRESSING SUBROUTINE-CODING IN PRESET

Addresses of instructions (and parts thereof) appearing in the subroutine-coding may be calculated by using the relative address for each line and the format in Figure A. Or, the following Cn notation may be used:

Cn, where "n" is the number of the desired column in the pseudo-line. This, in turn, may be followed by the number of the desired character within that column, enclosed within parenthesis.

For example, assume that the following instruction appears in the symbolic coding section:

	C1	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*
	INST. NO.	*COMMENTS*	*OP*	*A*	N_A	N_B	*B*
007530			TC	PSR15+3			

The preset may refer to the "3" in PSR15+3 as: 007530C4(7).

Cn notations not followed by parenthetical notations will address the ISS associated with that column. In addition, the beginning and ending of pseudo-lines may be addressed by using an SM or EM notation. In the above example, 007530SM would address the SM location of the TC instruction; 007530EM would address the EM symbol.

The preset shown on page 103, therefore, might also be written as follows:

relative address	*OP*	*A*	N_A	N_B	*B*
000030	72	002100EM	1	0	600000
000040	26	003100SM	1	1	003500EM

Note: When the subroutine is defined, the special symbols described above will be automatically removed from the preset coding and the proper addresses substituted.

SPECIAL HANDLING OF LITERALS WITHIN SUBROUTINES

The Operator will add a "U4" to *all* octal and decimal literals that appear in the first subroutine that is called into the program. A "U5" will be added to *all* literals appearing in the second subroutine included, etc.

Example:

	C1	C3	C4	C7
Call-line	SPAB00	SUBR	"123"	—
Subroutine instruction	PSR00	DA	C4	"456"
Adjusted line	PAB00	DA	"123"	"456"U4

Notes: a) ***Literals appearing in the call-line are not modified.***

b) If literals within the subroutine coding already carry U notation, the Assembler will substitute a new U number to minimize the possibility of duplications.

THE REPLACE SUBROUTINE DESCRIPTOR VERB

The Assembly System provides the ability to specify subroutine replacement with the REPS Descriptor Verb.

Format:

INST. NO.	*OP*	*A*	*B*
DP-address	REPS	Subroutine name	S000 or P000

Following the REPS line, the new version of the subroutine is entered in exactly the same format as is used with the DEFS verb. Note that, in the B-address, S000 is entered if the new version is a simple substitution type; P000 is entered if the new version contains a programmer-written preset, regardless of what type of subroutine is being replaced.

The REPS verb will cause the Assembly System to delete the subroutine listed in the library under the name specified in the A-address. The new version as written following the REPS line will be inserted in its place. This verb will also cause the Assembly System to generate two new updated library tapes.

THE DELETE SUBROUTINE DESCRIPTOR VERB

The ability to delete subroutines from the library is provided by the DELS Descriptor Verb:

INST. NO.	*OP*	*A*	$N_A N_B$	*B*
DP-address	DELS	Subroutine name		

The DELS verb causes the Assembly System to delete the named subroutine from the library. Two updated library tapes are generated as a result of the DELS function.

RCA 501 AUTOMATIC ASSEMBLY PROGRAM SHEET

TITLE

PROGRAMMER

DATE

PAGE

	1 INSTRUCTION NUMBER	2 COMMENTS	3 OP	4 A ADDRESS	5 N_A	6 N_B	7 B ADDRESS	8 T ADDRESS	9 N_T	10 CSG	11 IF	12 GO TO	13 N_1	14 IF	15 GO TO	16 N_2	17 IF	18 GO TO	19 N_3
	DPAA10		DEFS	SUB1			S000												
000000	PAB10	TC → SET T REG	71	000040	1	0	000000												
000010		TC → CALL FOR MORE PARA'S	71	010320	0	0	000000												
000020		TRANSFER TK# → READ	22	000215	0	1	000035	PRESET											
000030		READ IN NEXT SEG	15	000060	1	0	$\boxed{00}$0000												
000040		SET T REG	72	000100	1	0	600000												
000050		TC → OPERATOR	71	004200	0	0	000000												
000060			00	000000	0	0	000000												
000070			00	000000	0	0	000000												
000100	PSR00		TCT	STP			B7												
000500			XX	XX			XX												
"			XX	XX			XX	SEG I											
"			XX	XX			XX												
020600			LIM	000020	1														
000100	PSR05		XX	XX			XX												
000500			XX	XX			XX												
"			Xx	XX			XX	SEG II											
"			XX	XX			XX												
020600			LIM	000010	1														
000100	PSR10		XX	XX			XX												
000500			XX	XX			XX	SEG III											
"			XX	XX			XX												
017100			TC		B7														
017500			END																

Example of Segmented Subroutine which uses more than 5 "Static" parameter lines.

Note that segments one and two will use the call line and the first four PARA lines for static parameter information. Segment three will use the rest of the Parameter lines.

501 AUTOMATIC ASSEMBLY PROGRAM SHEET

TITLE

PROGRAMMER

DATE

PAGE

	1 INSTRUCTION NUMBER	2 COMMENTS	3 OP	4 A ADDRESS	5 N_A	6 N_B	7 B ADDRESS	8 T ADDRESS	9 N_T	10 CSG	11 IF	12 GO TO	13 N_1	14 IF	15 GO TO	16 N_2	17 IF	18 GO TO	19 N_3
	DPMR10		DEFS	SUBR			P002												
000000	PAA10	TC → SET T REG	71	000030	1	0	000000												
000010		SET TK# → READ	22	000215	0	1	000025												
000020		READ IN NEXT SEG	15	000050	1	0	[00]0000	PRESET											
000030		SET T REG	72	000050	1	0	600000												
000040		TC → OPERATOR	71	004200	0	0	000000												
000050	PSR00	STORE STP	TCT	STP			B7												
000450	PSR01	DYNAMIC PARA TO LHE CLR INST	TCT	M000003	7		PSR03(A)												
001050	PSR02	DYNAMIC PARA TO RHE CLR INST	TCT	M000007	7		PSR03(B)	SEG I											
001450	PSR03	CLEAR AREA	SCT	[ ]			[ ]												
"			XX																
"			XX																
021050			LIM	000010	1														
000050		DYNAMIC PARA TO DECIMAL ADD	TCT	M000013	7		PSR10(A)												
"	PSR10		DA	[ ]			"*15+"	SEG II											
"			XX																
021050			LIM	000010	1														
000050			XX																
"			XX					SEG III											
"			XX																
017450			TC		B7														
021050			END																

EXAMPLE OF SEGMENTED SUBROUTINE WHICH USES "DYNAMIC" PARAMETERS.

501 COMPUTER HSM RECORD

Figure A

FORMAT OF SYMBOLIC SUBROUTINE LINE AS RANDOM DISTRIBUTED

00	01	02	03	04	05	06	07	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37	40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57	60	61	62	63	64	65	66	67	70	71	72	73	74	75	76	77
<	●			P-ADDRESS									●																				COMMENTS																															
											●		OP			●					A-ADDRESS																		●	NA			●		NB		●							B-ADDRESS													
●						T-ADDRESS														●	NT			●	CSG			●			IF						●								GO TO															●	N1		●			
	IF					●				GO TO																	●	N2			●			IF					●							GO TO															●	N3			>

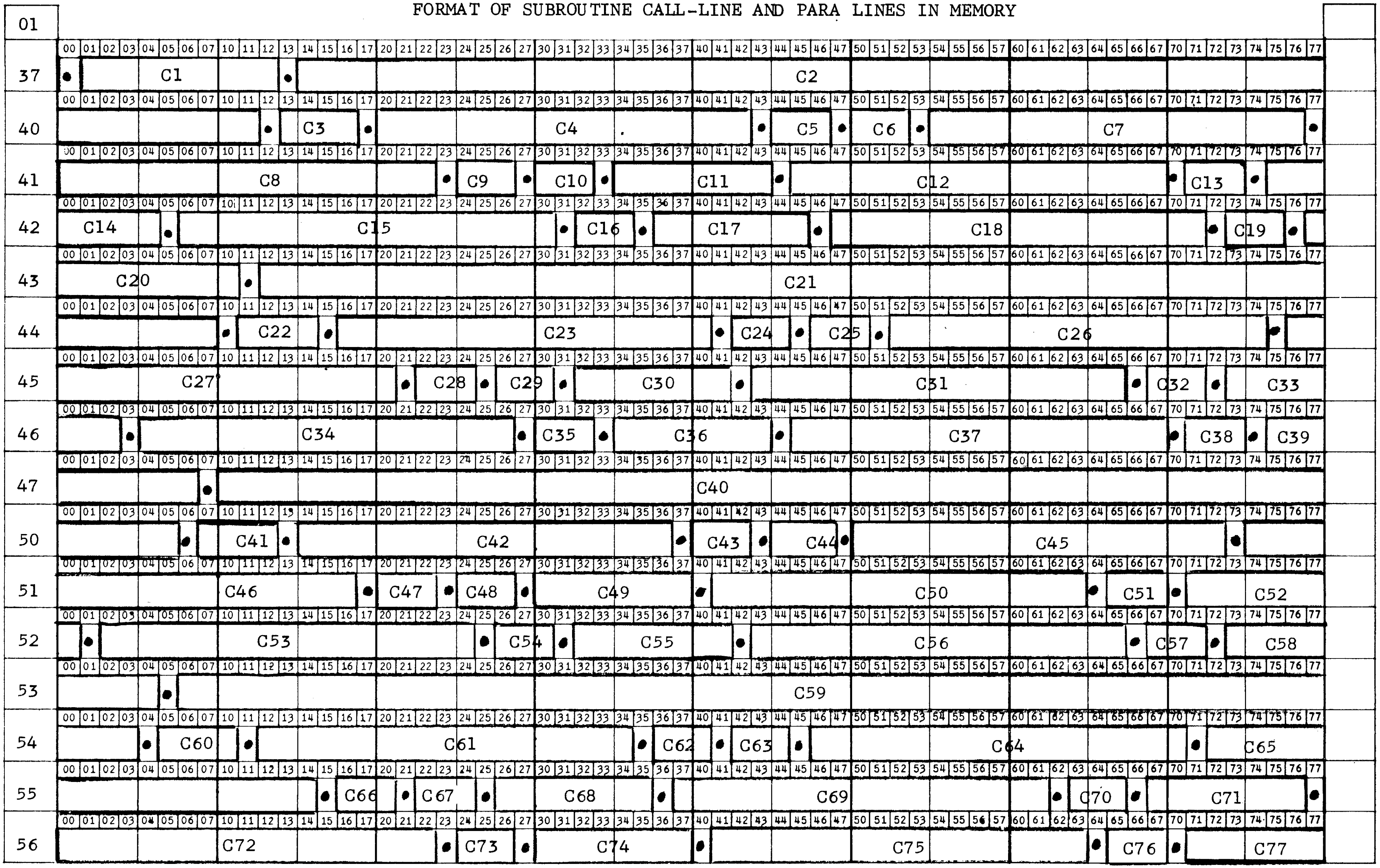
TITLE:______ BLOCK NO.:______ INDEX NO.:______ PROGRAMMER:______ DATE:______ PAGE 1 OF 1

IE 243

501 COMPUTER HSM RECORD

Figure B

FORMAT OF SUBROUTINE CALL-LINE AND PARA LINES IN MEMORY



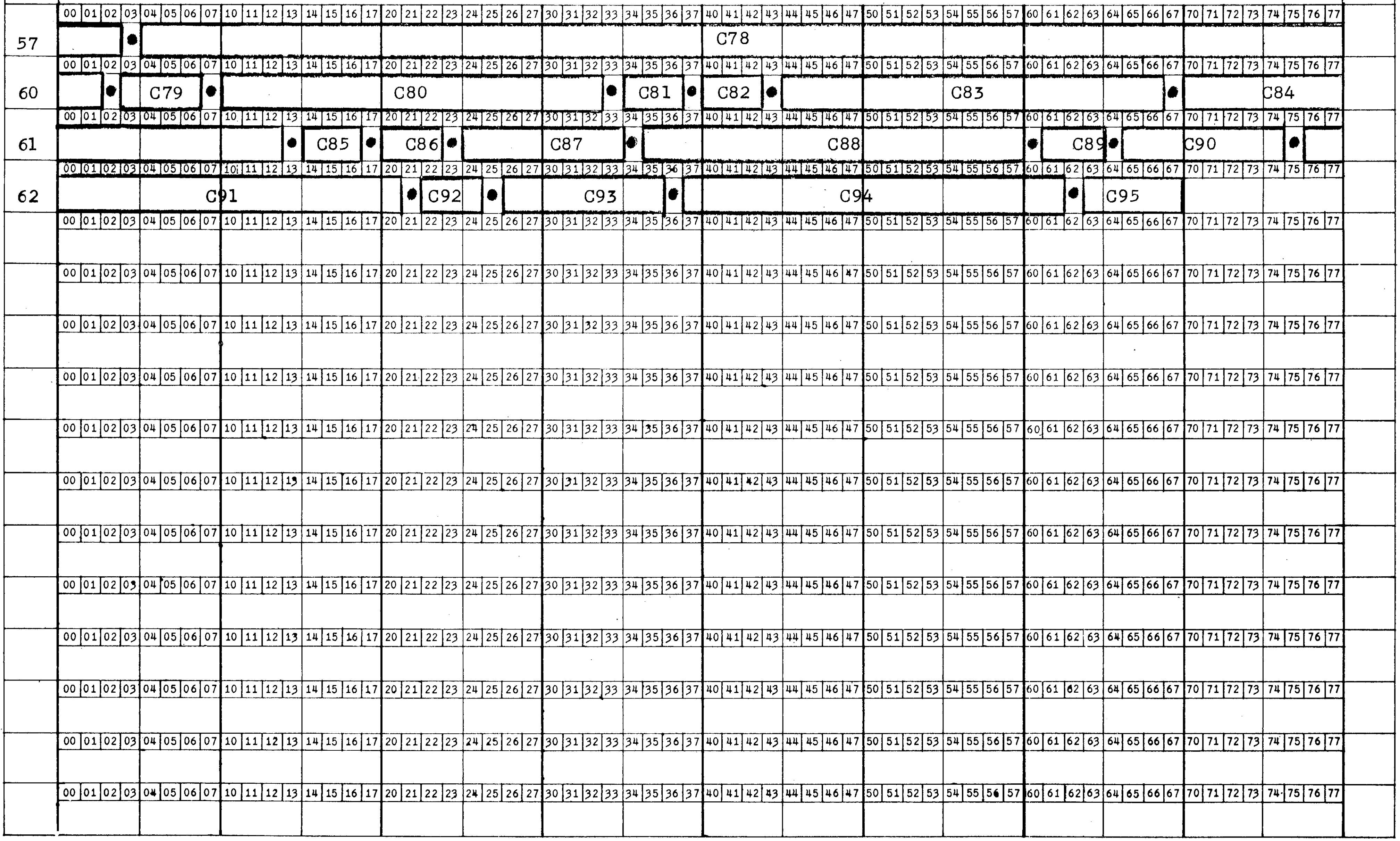
TITLE: ______ BLOCK NO.: ______ INDEX NO.: ______ PROGRAMMER: ______ DATE: ______ PAGE 1 OF 2

IE 243

501 COMPUTER HSM RECORD

Figure B (Cont'd.)

01 FORMAT OF SUBROUTINE CALL-LINE AND PARA LINES IN MEMORY



TITLE: BLOCK NO.: INDEX NO: PROGRAMMER: DATE: PAGE 2 OF 2

IE 243

X. ASSEMBLY SYSTEM OUTPUTS

Outputs of the Assembly System are of two types—magnetic tape and visual (printed). Visual outputs may be further classified into two categories—normally expected documents and error indicators. The purpose of this chapter is to provide a description of these outputs in sufficient detail to indicate the facilities available for program testing and library handling.

MAGNETIC TAPE OUTPUTS

Object Program

First and foremost, the Assembly System produces a copy of the object program on magnetic tape. This represents the machine coding which results from the translation of the program written in Assembly language.

The object program will be placed on magnetic tape in library format. That is, all conventions required by service routines for program handling are observed. Object programs may therefore, be treated exactly as if they were manually prepared. They may be tested, run, included in the Program Library, etc.

Subroutine and Macro-instruction Library

Six Descriptor Verbs were described in previous chapters which affect the content of the Subroutine and Macro-instruction Library. They are: DEFM, REPM, DELM, DEFS, REPS and DELS. If any one or more of these are used in an Assembly language program the Subroutine and Macro-instruction Library is changed accordingly. The Assembly System prepares two copies of the new tape containing the latest changes.

PRINTED OUTPUTS

Standard Documents

The documents described below are produced as the normal output of the Assembly System. In general, the purpose of these documents is to provide a permanent hard copy for use in future operations and in program debugging.

1. *Machine Assignments vs. Assembly Language*

This document is the basic record of the machine code which results from translation of Assembly language. Its purpose is to indicate the high speed memory allocations which have been made and the relationship of these allocations to statements in Assembly language. Hereafter, this document will be referred to by the abbreviation MACHAS.

MACHAS will be divided into several sections. One of these is a listing of the machine-code version of the program in ascending memory address order. This listing also contains a reference to the symbolic P-addresses which generated these instructions.

Fixed constants will be listed separately as part of the basic document. This listing will appear in two parts—literals and symbolics. Each fixed literal will be shown with the memory addresses representing the left and right boundaries. Each symbolic (K designation) constant is listed with the symbolic name followed by the memory addresses representing the boundaries of the constant. The definitions of these symbolic names (i.e., the actual constants) are shown with Descriptor Verbs in the listing of DEFK instructions.

Segment or non-fixed constants are shown in exactly the same fashion as fixed constants, except that a separate list is prepared for each segment.

All input memory areas are designated. This consists of the symbolic file names, boundary addresses of the area in memory occupied by the files and the symbolic names and machine addresses for each FAA item in the files. Alternate file areas are not included in this listing.

The Multiple Sector Write and Random Distribute instructions require lists of addresses to be stored in the memory. The name and address of each such list will be printed in addition to the symbolic name and list entry for every item affected by the MSW or RD instruction.

A final listing on this document is the allocation of working areas. This consists of the symbolic name (W) and the memory address of each such area.

2. *Sorted Assembly Coding*

This document is a listing of the Assembly Language program after certain processes are effected. The Assembly coding will be sorted according to instruction number and the coding implied by the use of subroutines will have been inserted in the program.

3. *Descriptor Verbs*

A complete listing is given of all Descriptor Verbs used in the program. These verbs are listed in the order of appearance in the sorted Assembly language program.

4. *Breakpoint Bit Assignments*

A listing is given of all uses in the program of the breakpoint bits. This listing gives the location of the Assembly instructions giving rise to unconditional Transfer of Control (TC) instructions which use breakpoint bits and the location of first machine instruction generated thereby. In addition, the particular bits used in each instruction are given. The instructions are listed in the order in which they appear in the Assembly language program.

5. *Input-Output Statements*

A list is given of all input-output statements in which an I or $\underline{O}$ Symbolic tape trunk is used.

6. *Program Stops*

A list is given of all normal stops (ST) that appear in the program.

7. *Criss-Cross Listing*

A criss-cross listing is also supplied which lists each symbolic address in alphabetical order against each instruction which refers to it. Moreover, in that part of the listing in which P-addresses appear, the entire two-address symbolic instruction is also printed out, in addition to the instructions making reference to it. This is a valuable part of the listing, as this two-address symbolic listing directly corresponds with the machine code.

8. *Object Program Examiner*

The Object Program Examiner scans the generated machine-code instructions for ***questionable*** conditions, appending a unique symbol to each instruction in which a potential (suspected) error condition is found. Appendix D of this manual lists the instructions tested, showing the symbol appended in the printout denoting the possibility of a particular error condition.

Error Indicators

The Assembly System provides exceptional facilities for detecting and correcting errors in Assembly language coding. These errors fall into two categories—those which can be temporarily bypassed and those which require immediate action.

Bypassable Error Indicators

As mentioned above, one type of error, which is detected by the Assembly System, causes a printout called an "error indicator". However, the Assembly System continues to function, ignoring the "guilty" Assembly language statement until all of the program has been examined and has undergone initial processing.

This procedure introduces a high degree of efficiency in program testing. Since nearly all error indicators fall into this class, it is often possible to find all coding errors in one sweep through the program. Thus, all corrections can be made at one time, avoiding constant re-starting and, consequently, loss of machine time.

Error indicators provide sufficient information to permit immediate diagnosis of the error and the location of the "guilty" Assembly language statement.

Example:

UNDEFINED WAMT IN PAB20

This indicates that instruction PAB20 refers to work area, WAMT, which has not been defined in the program.

Many kinds of errors are detected. For example, references to undefined files and data names, improper mnemonic operation codes, etc., are typical.

Error Indicators Requiring Immediate Action

Certain errors encountered during assembly will cause the Assembly System to produce an Incomplete Assembly. In these cases, a copy of the data sheets and sorted pseudo-code will be provided so that the programmer may make the necessary corrections, and return for a reassembly. This type of error is generally caused by exceeding various Assembler limits, such as defining more than 20 files, exceeding the number of constants which can appear in the program, defining more than 20 L-lists, and so on.

When a reassembly is required, the errors must be analyzed and the necessary changes to coding determined. Next, these changes must be placed on paper tape using one or more of the following Corrective Verbs: INSERT (ISRT), DELETE (DLTE), REPLACE (REPL).

The manner and rules governing correction procedures are the subject of a separate RCA publication titled "CORRECTION TECHNIQUES."

APPENDIX A

PROGRAMMER CHECK LIST PRIOR TO ASSEMBLY

I DATA SHEETS

Format:

□ FILE NAME on all pages.
□ Number of pages in all headers (1 of 1, 5 of 8, etc.)
□ Files and pages in correct sequence for punching.
□ Message Format Box has Y or N.
□ MAX, AVG. and % USE columns filled in.
□ EM terminates all pages.
□ EM EF terminates each file (except last).
□ EM ED terminates *last* file.
□ All FAA items have X in FAA column.

Limits:

□ 20 Files.
□ 240 FAA items per file.
□ Maximum item size is 999.
□ 26 sub-items per item.
□ 30 Read-in areas (F, XF, YF, etc.)

II PSEUDO-CODE

Instruction Numbers:

□ DP-addresses *must* appear for these OP codes:

DEFK
DEFV
DEFM
DELM
DEFS
DELS
REPM
REPS
ASGN
OVLY
LEVE
SGMT
STRT
STSH
TAPE

□ *EXPLICIT* P-addresses *must* appear with these instructions:

a. All DUP verbs.
b. Line immediately following DUP verb.
c. All Macro instruction call lines.
d. Line immediately following subroutine call line (100 greater).
e. Line immediately following subroutine entrance line.
f. Next pseudo-instruction following the "END" line of a DEFS, REPS, DEFM and REPM.

Program Limits:

□ 75 Fixed literals.
*□ 75 Non-fixed literals.
(Note: literals may not exceed 19 punchable characters, *including control symbols.)*
□ 40 Fixed K's.
(Note: 320-character limit per fixed K, *including control symbols.* Total number of characters limit is 1,000–(6 times number of Fixed K's).

* See segmenting

□ 40 Non-fixed K's.
(Note: 320-character limit per non-fixed K, *including control symbols.* Total limit is 2520.

□ 63 I/O Symbolic Names defined.

□ 1 STRT verb.

*□ 1 Program Exit Line (ST) -- G76 and M76 used for other stops.

□ 10 DUP instructions.

□ 20 L-lists.

*□ 49 STSH verbs.

□ RES verb (510 limit for instructions to be reserved).

IF YOU HAVE SEGMENTED YOUR PROGRAM.

□ All non-fixed constants *must* be defined in the segment of use.
The following limits apply:
a. 75 non-fixed literals in *each* segment – see program limits for character limitations.
b. 40 non-fixed K's in *each* segment – see program limits for character limitations.

□ 49 STSH verbs in *each* segment.

□ Segments end in a TC, ST or some other break in instruction sequence.

□ Only one program exit line (ST) per segment. Limit of 30 per program.

□ Segments should be called in by programmer using RINS macro.

III GENERAL CHECK LIST

□ A STRT verb has been included.

□ Pseudo-code pages appear in proper sequence for punching.

□ All machine addresses preceded by "M" (unless "M" appears in OP column).

□ All work areas defined as to length – limit of 9999.

□ Symbolic names:
a. All symbolic names defined in data sheets or in program.
b. In read-in areas only *FAA* items referred to symbolically.
c. No control symbols (comma, parentheses,* ISS, SM, etc.) used as part of a symbolic name.
d. All symbolics beginning with "X", "Y", or "Z" must have "F" or "D" as their second character.

□ Instruction numbers:
a. On *initial* assembly relative P-addresses *cannot* appear in the instruction number column (except for decimal point insertion).
b. Instruction No's appear only once – no duplicates.

□ Literals:
a. must begin and end with quotes or number signs.
b. octal literals must have *even* number of characters which range from 0 to 7.

□ RD and MSW instructions:
a. All RD and MSW instructions must contain an L-list name.
b. When defining an RD or MSW specified list, the *maximum limit is 57 entries per list.* Each list must contain at least *one* destination address.
c. "END" must appear as the last entry in the list.
d. "SAVE" notations in the MSW must have a decimal number in the T column.

□ Alpha O's are distinguished by underlining, or numeric "zeros" are distinguished by slashes.

□ Only "C" or "O̲" can appear in the CSG column.

□ Txx cannot be used with TAPE verb – must be 10, 40, etc.

□ SET instruction can only be used with *REGISTERS,* not with standard HSM loc.

□ All DEFK instructions must have a K-name in the B-column.

□ Macro's and Subroutines:
a. All macro's and subroutines used in the program must appear on the Assembler Library tape.
b. All DEFM and DEFS instructions must have name in A-address.
c. All DEFM's, DEFS's, REPM's and REPS's must have "END" as the last OP code.
d. All DEFM's, DEFS's, REPM's and REPS's must have "S" or "P" in B address.

* See segmenting

□ Descriptor Verbs:

a) Entries cannot appear in any column on a Descriptor Verb line unless specified in the verb format.

b) DP-addresses can *only* be written in the Instr. No. column.

□ LEVE verb: maximum of 4 decimal digits in A-address.

□ SGMT verb: A-address cannot refer to a\DP-address.

□ DEFV verb: V-symbolic cannot be *defined* as having appendages (i.e., PLA1O+1, PLA20(B), WPAY(L+6) are incorrect).

CALLING IN SUBROUTINES

If you have called for a subroutine within the program, check the following:

□ Each include line must have a SP——ØØ address and the address of the next instruction must be at least 100 greater.

□ When calling in a "OPEN" subroutine the comments column must be left blank.

□ When calling in a "CLOSED" subroutine the comments column must have "INCLUDE+COPY+**n**".

□ *Call* lines for OPEN and CLOSED subroutines must have an SP———address and the next instruction must have an *explicit* P-address.

□ Call lines for CLOSED subroutines must have a "COPY+**n**" in the comments column.

APPENDIX B

INSTRUCTION REFERENCE CHARTS

TABLE I – 501 MNEMONIC INSTRUCTIONS

OP	*A*	*B*	*T*	*Remarks*

BA
(Binary Add)
	Symbolic or Literal	Symbolic or Literal	Symbolic or Literal	A = LHE } of Augend
				B = RHE } (and Sum)
				T = RHE of Addend

BS
(Binary Subt.)
	Symbolic or Literal	Symbolic or Literal	Symbolic or Literal	A = LHE } of Minuend
				B = RHE } (and Difference)
				T = RHE of Subtrahend

BRF (Block Read Fwd.)	Symbolic	Octal Tape Address preceded by T; e.g.T20 Symbolic Tape Address preceded by I (must be defined elsewhere by TAPE verb).		A = LHE of Read-In Area 1st character placed in C0.
BRR (Block Read Reverse)	Symbolic	Same as BRF		A = RHE of Read-In Area 1st character placed in C3.
CTC (Conditional Transfer of Control)	P-address of next instruction if PRP is set.	P-address of next instruction if PRN is set.		CTC will not normally be used in this manner since the "IF" and "GO TO" columns are so convenient for this use.
CSG (Control Simultaneous Gate)	Ignored	M-address B1 even = open gate B1 odd = close gate		CSG will not normally be used in this manner since the CSG column is available on the coding sheet.
DA (Decimal Add)	Symbolic or Literal	Symbolic or Literal	Symbolic T entry will generate a Justify Right of data from A to T.	A = Sign or space to right of sign of Augend B = RHE of Addend T = Sign of Sum (If no entry in T or NT, Sum overlays Augend)
DD (Decimal Divide)	Symbolic or Literal	Symbolic or Literal	Symbolic	A = Sign or space to right of sign of Dividend B = Sign or space to right of sign of Divisor T = LHE of Quotient

DM
(Decimal Mult.)
	Symbolic or Literal	Symbolic or Literal	Symbolic	A = Sign or RHE of Multiplicand
				B = Sign or RHE of Multiplier
				T = Sign of Product

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*
DS (Decimal Subtract)	Symbolic or Literal	Symbolic or Literal	Symbolic T entry will generate a Justify Right of data from A to T.	A = Sign or space to right of sign of Minuend B = RHE of Subtrahend T = Sign of Difference (If no entry in T or NT, Difference overlays Minuend)
IT (Item Transfer)	Symbolic or Literal	Symbolic		A = RHE of item to be transferred B = RHE of Destination
JR (Justify Right)	Symbolic or Literal	Symbolic		A = RHE of data to be justified B = RHE of Destination

LA (Logical "and")	Symbolic or Literal	Symbolic or Literal	Symbolic or Literal	A = LHE }
				B = RHE } of Operand to be modified (and Result)
				T = RHE of modifier (Mask)

LNS (Locate *n*th Symbol in Sector)	Symbolic	Symbolic	Symbol to be counted, comma, and decimal number of symbols to be counted (Limit of 4095) Example: ISS,23 or M,3 or sp,1289	A = LHE } B = RHE } of sector to be searched

LO (Logical "or")	Symbolic or Literal	Symbolic or Literal	Symbolic or Literal	A = LHE }
				B = RHE } of Operand to be modified (and Result)
				T = RHE of modifier

LRF (Linear Read Fwd.)	Symbolic	Octal Tape Address preceded by T Symbolic Tape address preceded by I (must be defined elsewhere by TAPE verb).		A = LHE of Read-In Area (Loc. of SM) SM placed in C0.
LRR (Linear Read Reverse)	Symbolic	Same as LRF above		A = RHE of Read-In Area (Loc. of EM) EM placed in C3.
LW (Linear Write)	Symbolic, K-name/ or *octal* literal.	Octal Tape Address preceded by T Symbolic Tape address preceded by O (must be defined elsewhere by TAPE verb).		End Message symbol must stop write.
MSW (Multiple Sector Write)				
Machine Format:	L-address	Tape No.	Blank	List must be defined elsewhere.

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*

Specified List:	L-address
	(Succeeding Lines):
	BLANK
		Tape No.
		(Succeeding Lines):
		A decimal number, not limited to 64, designnating the number of chars. in the sector (as specified in the adjacent T field) to be written out.
		(At end of List):
		END
			Blank
			(Succeeding Lines):
			Symbolic address of 1st character to be written.
				Limit of 57 entries. Any number of MSW instr. may use the same list. The list should only be given after one of the MSW instrs. The other MSW's need only refer to L-name.

Combination List:	L-address (Succeeding Lines): BLANK	Tape No. (Succeeding Lines): A decimal number, not limited to 64, designating the number of chars. in the sector (as specified in the adjacent T field) to be written out. OR SAVE (At end of List): END	Blank (Succeeding Lines): Symbolic address of 1st character to be written. OR If SAVE is in B field, place here a decimal number specifying the number of tetrads to be reserved for generation by running program.	Same as above.
Unspecified List:	L-address	Tape No.	The number (decimal, with a maximum of three chars.) of tetrads to be reserved for the list.	Octal character count and addresses must be loaded by program.

OCT (One Character Transfer)	Symbolic (usually relative character address) or Literal.	Symbolic (usually relative character address).		A = From
				B = Destination

PA (Paper Advance)	PC = Page Change
	VT = Vertical Tabulation
	If a Paper Advance (Line Shift) is desired, show number of lines as a decimal number (omit quotes).
		Ignored		

PES (Programmed Error Stop)	Ignored	Ignored		
PR (Print)	W-address from W101 through W199	Ignored		This print area must be specified, followed by a period, once in the program. The assembler will allocate 128 chars.
RAI (Return after Interrupt)	Symbolic	Symbolic		Normally the A & B fields will be BLANK.

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*
RD (Random Distribute)				
Machine Format:	Symbolic label for File (if SM is to be distributed) or relative reference to first item to be distributed. OR A W address (used for redistribution to another area such as a print line)	L-address	Blank	List must be defined elsewhere.
All:	File name or File name relative	L-address	All	
All + 1:	File name or File name relative	L-address	All + 1	Extra location for items with ISS's.
Specified List:	LHE of Area to be Distributed (may be symbolic)	L-address	The List of destination addresses (relative symbolic). Use TA for throw away items Use END to terminate list	Limit of 57 entries.
Unspecified List:	LHE of Area to be Distributed (may be symbolic)	L-address	SAVE J "J" = decimal number of tetrads to be reserved.	Addresses must be loaded by program.
Combination List:	LHE of Area to be Distributed (may be symbolic)	L-address	Address Address or SAVE J etc. END (SAVE J cannot be used on first line)	Limit of 57 entries.
RNS (Rewind "n" Symbols)	Symbol to be rewound (ED,EF,SM, or GAP) followed by a comma and decimal number of symbols to be rewound. Example: SM,18	Octal Tape address preceded by T. A symbolic Tape address preceded by O or I (must be defined elsewhere by TAPE verb).		
RWD (Rewind to BTC)	Ignored	Same as RNS above		

SC
(Sector Compare)
	Symbolic or Literal	Symbolic or Literal	Symbolic or Literal	A = LHE } of Minuend
				B = RHE }
				T = RHE of Subtrahend (A Set T Register instr. is generated)

SCC
(Sector Clear by Char.)
	Symbolic	Symbolic		A = LHE } of Sector to
				B = RHE } be cleared

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*

SCD (Sector Compress Delete Redundant ISS's)	Symbolic	Symbolic	Symbolic	A = LHE
				B = RHE of area to be compressed
				T = RHE of Destination (Set T)

SCR (Sector Compress Retain Redundant ISS's)	Symbolic	Symbolic	Symbolic	A = LHE
				B = RHE of area to be compressed
				T = RHE of Destination (Set T)

SCT (Sector Clear by Tetrad)	Symbolic	Symbolic		A = LHE B = RHE of Sector to be cleared
SET (Set Register)	Symbolic or Literal	BT B6 = T Register RT RP B4 = P Register BP BA B2 = A Register RA		
	BLANK	RPRN RPRP = PRI Settings RPRZ		
SSG (Sense Simultaneous Gate)	P-address	P-address		A = Address of next instr. if gate is open B = Address of next instr. if gate is closed
SSM (Sense Simultaneous Mode)	P-address	P-address	BLANK, unless a simultaneous paper advance may be anticipated in which case place here: An A-symbolic address representing the address of the next instr. to be performed.	A = Address of next instr. if a Simultaneous Read is sensed B = Address of next instr. if a Simultaneous Write is sensed

SSW (Single Sector Write)	Symbolic or Literal	Symbolic or Literal	Octal Tape Address preceded by T.
			Symbolic Tape Address preceded by Q (must be defined by TAPE verb).
				A = LHE
				B = RHE of write-out sector
				T = Tape Designation

ST (Stop)	Ignored	Ignored		

STC (Sector Transfer By Char.)	Symbolic or Literal	Symbolic or Literal	Symbolic	A = LHE
				B = RHE of Sector to be transferred
				T = RHE of Destination

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*

STR (Store Register)	Symbolic or Literal	BP
		B4 = P Register
		RP
		RB = B Register
		RS = S Register
		BT
		B6 = T Register
		RT
		RPRI = PRI's
				A Store Register Instruction is generated

		B1 B3 Static Address B5 Modifiers B7 BA B2 STA HSM Loc. RA 000221-223 STP = HSM Loc. 000241-243 RRAI = Ret. after Interrupt Loc. 000001-003 RPAJ = Paper Adv. Jump Loc. 000201-203		A Three Char. Transfer "B" field to "A" field is generated

STT (Sector Transfer by Tetrad)	Symbolic or Literal	Symbolic or Literal	Symbolic	A = LHE
				B = RHE of Sector being transferred
				T = RHE of Destination

TA (Tally)	P-address	Symbolic or Literal (Literal must start at C1 position in tetrad)		A = Address of next instr. if Quantity not $(000000)_8$ B = Tetrad containing Quantity
TC (Transfer Control)	P-address	If Breakpoint Switches are to be set: SW0 through SW5 or combinations such as SW245		

TCA (Three Char. Add)	Symbolic or Literal	Symbolic or Literal	Symbolic (Entry here generates a TCT of A to T)	A = Augend
				B = Addend
				T = Sum (If no entry in T or NT, sum overlays Augend)

TCS (Three Char. Subtract)	Symbolic or Literal	Symbolic or Literal	Symbolic (Entry here generates a TCT of A to T)	A = Minuend
				B = Subtrahend
				T = Difference (If no entry in T or NT, difference overlays Minuend)

TCT (Three Char. Transfer)	Symbolic or Literal	Symbolic		A = From
				B = Destination

TCW (16.7K Write)	Symbolic, K-name or Literal	Octal Tape Address preceded by T. Symbolic Tape address preceded by O (must be defined by TAPE verb).		

TABLE I – 501 MNEMONIC INSTRUCTIONS (Continued)

OP	*A*	*B*	*T*	*Remarks*
TS (Tape Sense)	P-address	Octal Tape address preceded by T A symbolic Tape address preceded by I or O (must be defined by TAPE verb).	Conditions to be sensed for: Any combination of the following letters: B BTC Test E ETW Test F FWD Test R REV Test M Motion Test N Non-Operable Test	A = Next instr. if any of conditions specified in "T" are present
UNS (Unwind "n" Symbols)	Symbol (EM,ED,EF, or GAP) followed by a comma and decimal number of symbols to be unwound. Example: EM,29	Same as TS (above)		

ZS
(Zero Suppress)
	Symbolic	Symbolic		A = LHE } of Sector
				B = RHE }

TABLE II–SPECIAL ASSEMBLER OPERATION CODES

Inst. No.	*OP*	*A*	*B*	*T*	*Remarks*
Must *not* be assigned an instruction number	ADV (Add Variable)				Place a plus or minus sign followed by a V address or other symbolic (which must be defined elsewhere by DEFV) directly under the address to be modified. Any address on the coding sheet (incl. "GO TO") can be modified.
This inst. and next inst. *must* both have Explicit P-address	DUP (Duplicate)	P-address (Must *not* be relative)	P-address (Must be relative to the address in the A field)	Blank–if no addresses within the area to be duplicated are to be modified F will remove X, Y, or Z from references followed by $ XF will cause Ref. followed by $ to be XF or XD YF will cause Ref. followed by $ to be YF or YD ZF will cause Ref. followed by $ to be ZF or ZD	Used to duplicate all coding between and including addresses in A & B fields of this instruction. See DUP instruction for explanation of special cases where Random Distribute and Multiple Sector Write are in the Dup area.
Explicit P not required	IGN (Ignore)	Ignored	Ignored		IGN will permit the use of all fields to the right of the B field on the coding sheet. Any entries in CSG, or the IF and GO TO fields will generate the instr. that they normally give rise to.
Explicit P not required	RES (Reserve Program Space)	The number of instructions to be reserved (decimal number) Limit: 510	BLANK		8 characters reserved for each instruction. The P number of this instruction determines the point at which instruction space is reserved.

TABLE III–DESCRIPTOR VERBS

Inst. No.	*OP*	*A*	*B*	*T*	*Remarks*
DP-address	ASGN (Assign)	Literals only	M address between 000100 and 000277		The literal is placed in memory as part of an initial program block.
DP-address	DEFK (Define a K Symbolic)	Literal (one line on coding sheet only)	K Symbolic	Comments Column: Any Comments	Quotes or number signs must start and end literal.
		BLANK	K Symbolic	Comments Column: Literal (may be continued on succeeding lines in comments column. When doing so, leave all other columns blank on succeeding lines).	Slant symbol first character = SM. Slant symbol last character = EM. 40 fixed K's per program; 75 non-fixed K's per segment.
DP-address	DEFV (Define a V Symbolic)	V Symbolic being defined	Definition of V: Symbolic or Machine Address (Appendages cannot be used)		Limit of 99 DEFV's
DP-address	DEFM (Define a Macro-instruction)	Name of Macro-instruction (4 alphanumeric chars.)	S (Macro-instruction only) or P (Preset and Macro-instruction)		Two-address format. END must appear in last OP code. Will produce two new copies of Macro & Sub-routine Library Tape with this new routine inserted.
DP-address	DEFS (Define a Subroutine)	Name of Subroutine (4 alphanumeric chars.)	S000 (Subroutine only) or P000 (Preset and Subroutine)		Same as DEFM with exception that Subroutine follows DEFS in complete pseudo-instruction format. END must appear in last OP code.
DP-address	DELM (Delete a Macro-instruction)	Name of Macro-instruction (4 alphanumeric chars.)	(Leave blank)		Will produce two new copies of Macro & Subroutine Library Tape with this routine deleted.
DP-address	DELS (Delete a Subroutine)	Name of Subroutine (4 alphanumeric chars.)	(Leave blank)		Same as DELM
DP-address	LEVE (Leave Memory)	Total number of chars. to be reserved. (4 digit max.)	File name Wa (working storage "block") OR BOM (LHE memory limit for program)		Overides data sheet Max. Number of chars. if File name. M-address cannot be used.

TABLE III–DESCRIPTOR VERBS (Continued)

Inst. No.	*OP*	*A*	*B*	*T*	*Remarks*
DP-address	OVLY (Overlay)	P,F,XF,YF,ZF, symbolic address (may be relative) W address (2 char. symbolic)	F,XF,YF,ZF, W address (not the same W as the "A" field) F,XF,YF,ZF symbolic		See description of OVLY verb A = Area to be overlaid B = Area overlaying
DP-address	REPM (Replace a Macro-instruction)	Name of Macro-instruction	S = (Macro-instruction only) or P = (Preset and Macro-instruction)		Will produce two new copies of Macro and Subroutine Library Tape with this new routine replacing the old version of the same routine. New routine follows REPM in two address format. Complete the new coding with the word END.
DP-address	REPS (Replace a Subroutine)	Name of Subroutine	S000 = (Subroutine only) OR P000 = (Preset and Subroutine)		Same as REPM with exception that Subroutine follows REPS in complete pseudo-instruction format.
DP-address	SGMT (Segment)	P-address of first instruction in segment.	P-address or M-address of location where instruction specified in "A" field should be placed. ESGMT J(J = segment number and is decimal)	Number of this segment. Segments must be numbered consecutively.	Used to define all segments (excluding the first one) of the program. See description of SGMT verb. limit of 50 segments
DP-address	STRT (Start)	P-address of first instruction to be executed in the object program.	Segment number of instruction identified in "A". Not required if program consists of only one segment.		This instruction must appear in each program.
DP-address	STSH (Stash)	An A-symbolic address	P-address OR Decimal Number		A = Symbolic Address being defined B = Definition (Symbolic) If B contains a decimal number, it will be converted to a three character octal equivalent and stored. Any stashed information ("B" field of STSH) is addressable by the A-symbolic. Limit of 49 STSH's per segment.

TABLE III–DESCRIPTOR VERBS (Continued)

Inst. No.	*OP*	*A*	*B*	*T*	*Remarks*
DP-address	TAPE (Define I and O Symbolics)	Symbolic Tape Address (I and O Symbolics) Limit of 63 per program	Two digit octal tape number (do not use T).		List the "A" & "B" fields for all tape references. Give op. code TAPE only on first line. The next op. code sensed will show conclusion of the list.

APPENDIX C

The following addresses are supplied by the Assembly System for symbolic names or literals that are written without character notation. It should be noted that P-addresses are excluded from this list, since the Assembler will always supply the location of the operation code of the first generated machine instruction.

Instruction	A	B
BA	LHE	RHE
BRF	LHE	RHE
BRR	RHE	RHE
BS	LHE	RHE
CSG	RHE	RHE
CTC	RHE	RHE
DA	RHE	RHE
DD	RHE	RHE
DM	RHE	RHE
DS	RHE	RHE
IT	RHE	RHE
JR	RHE	RHE
LA	LHE	RHE
LNS	LHE	RHE
LO	LHE	RHE
LRF	LHE	RHE
LRR	RHE	RHE
LW	LHE	RHE
MSW	LHE	RHE
OCT	RHE	RHE
PA	RHE	RHE
PES	RHE	RHE
PR	LHE	RHE
RAI	RHE	RHE
RD	LHE	LHE
RNS	RHE	RHE
RWD	RHE	RHE
SC	LHE	RHE
SCC	LHE	RHE
SCD	LHE	RHE
SCR	LHE	RHE
SCT	LHE	RHE
SET	RHE*	RHE
SSG	LHE	RHE
SSM	LHE	RHE
SSW	LHE	RHE
STC	LHE	RHE
STR	RHE	RHE
STT	LHE	RHE
TA	RHE	RHE
TC	RHE	RHE
TCA	RHE	RHE
TCS	RHE	RHE
TCT	RHE	RHE
TCW	LHE	RHE
TS	RHE	RHE
UNS	RHE	RHE
ZS	LHE	RHE
ADV	LHE	LHE
STSH	C_o	C_o
G ——	RHE	RHE

*When an entry is made in the T-address of a Decimal Divide, the Assembler will assign the LHE in the generated SET instruction.

APPENDIX D

OBJECT PROGRAM EXAMINER

Machine Instruction Tested	*Error Symbol Indicated*	*Suspected Error Condition*
02 (PR)	#	A address not $(XXX000)_8$.
06 (UNS)	#	Illegal symbol in A1 character.
11 (SSW)	*	A-address greater than B-address.
16 (RNS)	#	Illegal symbol in A1 character.
21 (IT)	"	Destination area overlaps ISS in original area.
24 (STC)	*	A-address greater than B-address.
26 (STT)	*	A-address greater than B-address.
32 (ZS)	*	A-address greater than B-address.
33 (JR)	"	Destination area overlaps ISS in original area.
34 (SCC)	*	A-address greater than B-address.
35 (SCR)	*	A-address greater than B-address.
36 (SCT)	*	A-address greater than B-address.
37 (SCD)	*	A-address greater than B-address.
41 (BA)	*	A-address greater than B-address.
42 (BS)	*	A-address greater than B-address.
43 (SC)	*	A-address greater than B-address.
46 (LO)	*	A-address greater than B-address.
47 (LA)	*	A-address greater than B-address.
61 (CTC)	#	A3 and/or B3 character is not $(XO)_8$.
62 (SSM)	#	A3 and/or B3 character is not $(XO)_8$.
63 (TS)	#	A3 character is not $(XO)_8$.
65 (SSG)	#	A3 and/or B3 character is not $(XO)_8$.
66 (TA)	#	A3 character is not $(XO)_8$.
71 (TC)	#	A3 character is not $(XO)_8$.
71 (TC)	:	Breakpoints set in the B1 character.
72 (SET)	#	Illegal register setting in B1 character.
73 (STR)	#	Illegal register setting in B1 character.

INDEX

Page

- **Address Modifiers**
  - Setting 69
  - Storing 71
  - Symbolic Addresses 69
- **Breakpoint Settings** 18, 56
- **Constants** 34
  - Definition of:
    - Fixed and non-fixed 36
    - Length 35
    - Positioning within tetrad 36
    - Unique and non-unique 35
  - K-constants 37
  - Literals: 34
    - Octal 35
    - 501 Character 34
- **Data Addressing** 25
- **Data Sheet** 3
  - Header 3
  - Body 5
- **Descriptor Verbs** (see Verbs)
- **Files** 3
  - Alternate read-in areas 28
  - Description of 3
  - File area 26
  - Symbolic names 3, 28
- **Instructions**
  - Addressing:
    - General 40
    - Parts of 47
  - Instruction numbers:
    - Decimal Point P-address 44
    - Explicit P-address 12, 40
    - Relative P-address 41
  - List of 501 Mnemonic Insts 13
- **Items and Sub-Items**
  - Addressing:
    - Relative character 27
    - Relative to other FAA items 28
    - Relative to LHE of File 27
    - % Addresses 30
  - FAA 6, 26
  - Forward Read addressing 26
  - Non-FAA 29
  - Reverse Read addressing 29
  - Symbolic names 5, 26
- **Literals** (see Constants)
- **Machine Addresses** 25
- **Macroinstructions** 86
  - Call-line 86
  - Composition of:
    - Preset 87
    - Macro symbolic coding 87
  - Defining Macros (DEFM verb) 88
  - Memory Layout:
    - Call-line 96
    - Symbolic inst. line 96
  - Preset, Programmer-written 92
  - Segmenting macros 93
- **Memory Layout** 78
  - Assembly Assignments: 78
    - Fixed Constants 78
    - Instructions 79
    - Non-fixed constants 79
    - Read-in areas 79
    - Working Storage areas 78
  - Programmer Control: 79
    - ASGN – Assign verb 81
    - LEVE – Leave verb 80
    - OVLY – Overlay verb 82
    - SGMT – Segment verb 83
    - STRT – Start 85
- **Object Program** 78, 112
- **Operation Codes**
  - Machine format 14
  - Mnemonic, 501 13, 14
  - Mnemonic, Special 14
- **Print Areas** 32
- **Program Sheet, General Usage** 12
- **Pseudo-Code, General** 12
- **Read-In Areas** (see Files)
- **Registers**
  - Setting 73
  - Storing 74
  - Symbolic Addressing 72
- **Segmentation**
  - Macros 93
  - Program 83
  - Subroutines 103

Special Assembler Mnemonics
- ADV – Add Variable 54
- DUP – Duplicate 50
- IGN – Ignore 49
- RES – Reserve 49

Special Format 501 Insts.
- CTC – Conditional Transfer of Control 18
- CSG – Control Simultaneous Gate 17
- DA – Decimal Add 59
- DS – Decimal Subtract 59
- LRF – Linear Read Forward 26
- LRR – Linear Read Reverse 28
- LNS – Locate nth Symbol 57
- MSW – Multiple Sector Write 66-69
- PA – Paper Advance 58
- PR – Print 32
- RAI – Return After Interrupt 59
- RD – Random Distribute 60-66
- RNS – Rewind n Symbols 57
- SET –
  - "Set T" instructions 15, 58
  - "Other than Set T" 58
- SSG – Sense Simultaneous Gate 21
- SSM – Sense Simultaneous Mode 21, 59
- STR – Store Register 71, 74
- TA – Tally 72
- TC – Transfer of Control 18, 56
- TCA – Three Character Add 59, 71
- TCS – Three Character Subtract 59, 71
- TCT – Three Character Transfer 70, 76
- TS – Tape Sense 19, 56
- UNS – Unwind n Symbols 57

Standard Memory Locations, Addressing 76

Sub-Items (see Items)

Subroutines
- Call-line 97
- Composition of:
  - Preset 99
  - Symbolic Coding 99
- Defining Subroutines 101
- Dynamic Parameters 104
- Entrance Lines 98
- Memory Layout:
  - Call-line 110
  - Para lines 110
  - Symbolic lines 109
- Preset, Programmer-written 102
- Segmenting subroutines 103

Tape Station Addressing 39

Variable Addresses 53

Verbs, Descriptor 46
- ASGN – Assign 81
- DEFK – Define Constant 38
- DEFM – Define Macro 88
- DEFS – Define Subroutine 101
- DEFV – Define Variable 53
- DELM – Delete Macro 95
- DELS – Delete Subroutine 106
- LEVE – Leave 80
- OVLY – Overlay 82
- REPM – Replace Macro 94
- REPS – Replace Subroutine 106
- SGMT – Segment 83
- STRT – Start 85
- STSH – Stash 52
- TAPE – Tape 40

Working Storage 30
- Character Addressing 32, 33
- Length 31, 32
- Positioning within tetrad 31
- Types:
  - Distributed Data 34, 60
  - General 30
  - Print 32

RCA 501 ELECTRONIC DATA PROCESSING SYSTEM

Printed in U.S.A.