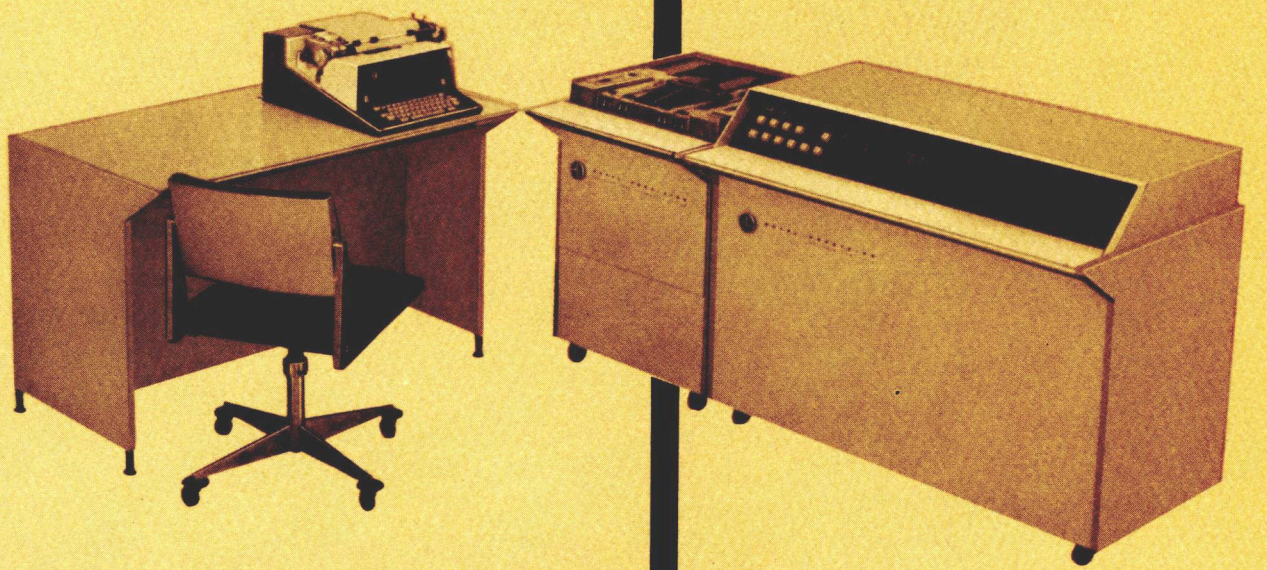
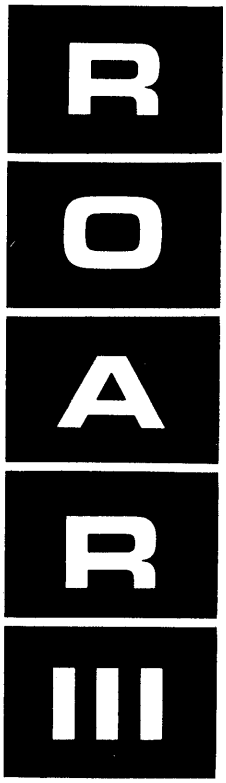
GC 4000

H2-01.2



ROAR III programming manual

GENERAL PRECISION, INC.
Commercial Computer Division



THE OPTIMIZER AND
ASSEMBLY ROUTINE

for the **RPC 4000** General Precision Electronic Computer

PROGRAM NO. H2-01.2

PREFACE

Recently a number of important modifications and changes were made in the General Precision assembler, ROAR I, program H2-01. 0. One version (ROARPACT) was developed so that ROAR could assemble programs compiled by COMPACT, program H3-01. 0. Another version (ROAR II), in addition to encompassing all the features of ROAR I and ROARPACT, produced a relocatable hexadecimal program tape, allowed selection of input-output devices, introduced a new error recovery procedure, and expanded the list of pseudo-instructions.

This publication describes ROAR III, the latest version of the assembly program. ROAR III incorporates all the features of the previous versions and provides some new ones. Information published for the first time in this manual includes

Chapter 2 - Tape Records, Transfer Code
Chapter 3 - How ROAR Optimizes
Chapter 4 - Delayed Address
Chapter 5 - HED, NXT, PAS, SLT, SRT, SBT, SUB, SBE
Chapter 6 - PROGRAMMING TECHNIQUES
Chapter 7 - Procedures for Punching Input Tapes
Chapter 8 - Sense Switch Options, Error Printouts, Error Recovery, Operating Procedures for Assembled Program
Appendixes - A, B, C

ROAR III is compatible with ROAR I and ROAR II; that is, any symbolic program that can be assembled by the earlier versions of ROAR can also be assembled by ROAR III (but not conversely). All references to "ROAR" in this document are to ROAR III unless specifically noted as "ROAR I" or "ROAR II. "

This manual has been prepared for both novice and experienced programmer. Included are a detailed description of ROAR's optimization processes and internal configuration, an expanded discussion of the pseudo-instructions and special addresses, an explanation of tape preparation, and complete operating procedures. The appendixes supply additional general information about the internal structure of ROAR, charts, and summaries. The RPC-4000 commands and the functions of the various control switches are not defined or explained. It is assumed that the reader is familiar with the command structure and the use of the manual controls on the computer. However, a summation of the commands is provided for reference purposes.

CONTENTS

		Page
1	PROGRAM PREPARATIONS	1-1
	Introduction	1-1
	Program Organization	1-1
	Program Coding	1-2
	Program Assembly	1-3
2	THE ASSEMBLED PROGRAM	2-1
	Program Tape	2-1
	Bootstrap	2-1
	Tape Records	2-1
	Transfer Code	2-2
	Decimal Listing	2-2
3	INSTRUCTION TIMING AND OPTIMIZATION	3-1
	Instruction Cycle	3-1
	Instruction Timing	3-2
	How ROAR Optimizes	3-4
	Symbol Table	3-5
	Availability Table	3-5
	Command Code Table	3-5
	Optimizing Rules	3-5
4	ROAR CODING RULES	4-1
	Coding Sheet for ROAR	4-1
	Location Field	4-2
	Order Field	4-2
	Indexed Orders	4-2
	ROAR Addressing	4-2
	Symbolic Addresses	4-3
	Numeric Addresses	4-3
	Blank Addresses	4-4
	Special Addresses	4-4
	Artificial Sector Addresses	4-4
	Special Character Addresses	4-6
	Regional Addresses	4-6
	Recirculating Track Addresses	4-6
	Double-Access Track Addresses	4-8
	Skip Address	4-9
	Delayed Address	4-10
	Comments Field	4-11

		Page
5	PSEUDO-INSTRUCTIONS	5-1
	Allocate Memory	5-1
	Control Assembly	5-1
	Establish Constants	5-2
	Input-Output	5-2
	Shift	5-2
	Comment	5-2
	Use Subroutine Library	5-2
	RES Reserve Portion of Memory	5-2
	REG Establish a Region	5-3
	RLR Designate a Relocatable Region	5-5
	PRE Prepare ROAR	5-6
	SET Establish Global Symbols	5-6
	RST Restore Symbol Table	5-7
	RRS Relocate Regional Storage	5-8
	AVL Make a Block Available	5-9
	EQR Equate and Reserve Location	5-10
	EQV Make Equivalent	5-11
	NIX Wait a Second	5-12
	NEW Begin a New Assembly	5-13
	CLS Clear Symbol Table	5-13
	END End Assembly	5-14
	TAG Designate Header Tag	5-14
	HED Assign Sequential Header Tag	5-16
	NXT Optimize Next Instruction as Indicated	5-17
	HEX Establish a Constant from Hexadecimal Input	5-18
	DEC Establish a Constant from Decimal Input	5-19
	ALF Input Alphanumeric Characters	5-20
	PAV Punch Availability Table	5-21
	PPA Punch and Print Availability Table	5-22
	RAV Read Availability Table from Tape	5-22
	PST Print Symbol Table	5-23
	5CS Punch Five-Character Symbols	5-24
	PAS Punch All Symbols	5-25
	PRC Print a Character	5-25
	SRT Shift Right	5-26
	SLT Shift Left	5-27
	COM Comment	5-28
	$(\overline{56})(\overline{57})(\overline{56})$ Compact-Generated Comments	5-28
	SBT Enter Subroutine Library Mode	5-29
	SUB Read Subroutine from Library Tape	5-30
	SBE Exit from Subroutine Library Mode	5-30
6	PROGRAMMING TECHNIQUES	6-1
	Program Library	6-1
	Data Input-Output	6-1
	Sectional Assemblies	6-7
	Optimizing Exit from Subroutine	6-8
	Header Tags Versus Global Symbols	6-10

		Page
7	INPUT TAPE PREPARATION	7-1
	Tape Characters	7-1
	Parity Checking	7-1
	Rules for Tape Preparation	7-1
	Procedures for Punching Input Tapes	7-2
	Correcting Errors While Punching Input Tapes	7-2
	Correcting Errors After Punching Input Tapes	7-2
	Duplicating Tapes On-Line	7-3
	Correcting Errors On-Line	7-3
	Correcting Parity Errors	7-3
8	OPERATING PROCEDURES	8-1
	ROAR Master Tape	8-1
	Bootstrap Procedure	8-1
	Manual Transfer to ROAR	8-1
	Assembly Preparations	8-2
	Input-Output Selections	8-2
	Changing Input-Output Selections	8-3
	Subroutine Tape Region Storage	8-3
	Bootstrap Track	8-3
	Assembly Procedures	8-4
	Interrupting an Assembly	8-4
	Sense Switch Options	8-4
	Error Printouts	8-5
	Error Recovery	8-6
	Input Errors	8-6
	Errors During Blind Assemblies	8-8
	Parity Errors	8-9
	Operating Procedures for Assembled Program	8-9
	Bootstrap Procedure	8-10
	Correction of Checksum Error	8-10
	Correction of Parity Error	8-10
A	SUBROUTINE LIBRARY TAPE	A-1
B	SPECIAL ROAR TABLES	B-1
	Symbol Table	B-1
	Equivalence Table	B-2
	Availability Table	B-2
C	COMMAND CODE TABLE	C-1
D	MODULO-EIGHT TABLE	D-1
	Basic EXC Data-Track Settings	D-1
E	ALPHANUMERIC CODES	E-1

		Page
F	INPUT-OUTPUT SELECTION CODES	F-1
G	SUMMARY OF RPC-4000 COMMANDS	G-1
H	SUMMARY OF ROAR PSEUDO-COMMANDS	H-1

ILLUSTRATIONS

		Page
Figure 3. 1	Instruction Timing	3-2
Figure 3. 2	Instruction Timing	3-3
Figure 3. 3	Instruction Timing	3-4
Figure 4. 1	ROAR Coding Sheet	4-1
Figure 4. 2	Form of Indexed Instructions	4-2
Figure 4. 3	Addressable Fields	4-3
Figure 4. 4	Typical Symbolic Addresses	4-3
Figure 4. 5	Sectors 90-98 for Special Addresses	4-5
Figure 4. 6	Sectors 98, 99 for Special Addresses	4-5
Figure 4. 7	Sample Regional Addresses	4-6
Figure 4. 8	RECRC Address	4-7
Figure 4. 9	Preset Location Counter	4-8
Figure 4. 10	Double-Access Address	4-8
Figure 4. 11	Double-Access Addressing	4-9
Figure 4. 12	Skip Address	4-9
Figure 4. 13	Skip Addressing	4-10
Figure 4. 14	Delayed Address	4-10
Figure 4. 15	Delayed Addressing	4-10
Table 6. 1	Representative Programs Available	6-1
Figure 6. 1	Input Routine	6-2
Figure 6. 2	Decimal Output Routine	6-3
Figure 6. 3	Hexadecimal Output Routine	6-4
Figure 6. 4	Suppress Leading Zeros	6-5
Figure 6. 5	Bit Configuration	6-7
Figure 7. 1	Tape Channels	7-1
Figure 7. 2	Sample Problem Coding	7-4
Figure 7. 3	Sample Problem Input Tape Listing	7-5
Figure 8. 1	Coding Error	8-7
Figure 8. 2	Coding Error	8-8

PROGRAM PREPARATIONS 1

INTRODUCTION

ROAR is an assembly program which translates symbolic-coded programs into numeric or machine-language which can be read directly by the RPC-4000. In addition, ROAR will insert in the object program all absolute memory addresses not supplied by the programmer, and furthermore choose these addresses so that each instruction in the object program will operate in the least amount of time. ROAR also provides a number of pseudo-instructions which are the programmers' means of communicating with ROAR. The pseudo-instructions cause ROAR to perform many convenience functions, such as reserving blocks of memory for data storage, equating certain symbols with other symbols, and other bookkeeping chores.

The programmer may code his program in symbolic language, absolute numeric language, or in a combination of the two. He then reads his program into the computer under control of ROAR, whereupon the program is assembled one instruction at a time. The two end results of the process are a hexadecimal tape of the assembled program, ready to be read into the computer and operated, and a printed list of the program. Each instruction is printed out on a separate line showing a) the instruction in the form in which it was read by ROAR, b) the instruction in its assembled form, c) the comments field of the instruction.

Programming for ROAR may be considered to include three areas of activity: organization, coding, and assembly.

PROGRAM ORGANIZATION

Proper organization of a program entails selection of the best method for solving the problem, determination of the limits within which the program is to operate, and choice of the input-output format.

Next, the programmer must determine the amount of storage area needed. It is desirable to restrict a program to the lower half of memory (tracks 00 through 52) whenever possible, thus leaving ROAR in memory during the check-out of the program. Then, if it is necessary to re-assemble the program, ROAR will still be available and need not be re-loaded.

When a program is especially large, it is often advantageous to divide the program into sections and code each section separately. Then each section can be assembled and checked out before attempting to run the whole program. After all sections have been successfully tested, they can be combined into a single tape.

Many programs and subroutines require regional storage, i.e., sequential memory locations, for the storage of tables, data, etc. The programmer must determine the amount of region storage area to reserve for his own program (the "object" program) and also for any subroutines he intends to use. The information concerning region storage for the subroutines is available in the respective

program descriptions. If the programmer plans to use a Subroutine Library Tape (i.e., a single tape containing numerous subroutines from which ROAR will assemble only those that have been referenced in the object program), he must reserve region storage for the subroutines at the beginning of the assembly when ROAR queries "Subroutine Tape Region Storage." Space for regions in the object program may be reserved in the same manner, but it is usually more desirable to establish these regions individually through the use of REG pseudo-commands. (See REG, Chapter 5.) A detailed description of how to make a Subroutine Library Tape is given in Appendix A.

Among the first outputs from ROAR will be the bootstrap necessary to load the assembled program. The bootstrap occupies one track and uses the Recirculating Track for temporary storage. The programmer must reserve the track where the bootstrap is to be stored since ROAR does not automatically reserve the track when it is allocated. (See "Assembly Preparations," Chapter 8.) To conserve space, the programmer may choose to locate the bootstrap in an area that will later be used for data storage.

PROGRAM CODING

The assembly program permits coding in a symbolic language whereby each machine instruction is represented by one symbolically noted assembly instruction. ROAR will interpret symbolic (mnemonic) and numeric commands and will assign optimum absolute memory locations for addresses which are expressed symbolically. ROAR will also act upon a number of pseudo-instructions. A pseudo-instruction is an instruction word containing a pseudo-command directed to ROAR. It takes the same form as an instruction and is reproduced on the decimal listing but in most instances is not punched in the program tape. Generally, pseudo-instructions affect only the assembly process - e.g., reserving areas of memory - and thus are not output as part of the assembled program. However, a few pseudo-instructions do result in instructions in the assembled program, namely those used for printing and shifting. These special pseudo-instructions are handled by ROAR in the same manner as RPC-4000 commands and follow the same rules that govern commands. (See "Blank Addresses," Chapter 4.)

The tape produced by ROAR will be a relocatable, hexadecimal program tape. That is, the program may be positioned anywhere in memory and is not restricted solely to the area where it is assembled. If the program is to be used as a relocatable program, absolute addresses should be avoided whenever possible. Absolute addresses, Double-access Track addresses, and Recirculating Track addresses cannot be relocated when loading the program. Symbolic addresses, including Regional Addresses, are made relocatable. If it is necessary to have a region that must not be relocated, absolute addresses should be used to refer to the locations rather than regional notation. (See "Regional Address," Chapter 4.)

Usually the first items listed on the coding sheet are those used to establish regions, e.g., for data storage; to set up equivalents, e.g., for entry points; and, to reserve areas, e.g., for limiting location of the program. If a header tag is to be used, it is also among the first items.

The first instruction word containing an RPC-4000 command must have the location given. Thereafter, the general rules governing blank fields are applicable. (See "Blank Addresses," Chapter 4.) Pseudo-instructions may appear anywhere in a program. General pseudo-instructions are not governed by, nor do they affect, the rules for blank addresses. For example, assume instruction A is

followed by a general pseudo-instruction, which is followed by instruction B. If instruction A has no blank address field, ROAR will except a filled location in the next instruction. ROAR does not consider the general pseudo-instruction to be an instruction, but rather instruction B.

Program checkout and correction will be simplified if symbols that look like pseudo-commands or mnemonic commands are avoided. Using HLT, SLC, HEX, etc. as location symbols can cause much confusion for the person checking the program.

The final item listed should be an END or NIX pseudo-command.

PROGRAM ASSEMBLY

When the ROAR-language program has been completed, a tape of it should be made for use during the assembly. (This tape is referred to as the symbolic input tape.) After the ROAR program is stored in memory, it will read one symbolically coded instruction at a time, assemble it as a machine-language word, and output that hexadecimal word as part of the assembled program. At the beginning of the assembly, ROAR requires certain specific information as explained under "Assembly Preparations" in Chapter 8. Throughout the remainder of the assembly there should be little need for intervention by the programmer unless there is an error printout. Then, following the assembly, the hexadecimal tape produced by ROAR can be read into memory and executed.

During an assembly ROAR reads the first 4 fields of an instruction in their entirety before any information is output. If all 4 fields of input are acceptable, ROAR will punch the hexadecimal output and will type the decimal equivalent of the assembled instruction. Then the Comments field is copied on the decimal listing. However, if an error other than parity occurs in any one of the first 4 fields, ROAR will clear from memory the instruction that it was building, execute an error printout, and halt. Error printouts and error recovery procedures are discussed in Chapter 8.

To provide the greatest latitude possible in relocating programs, the program should be assembled relative to location zero. That is, if a program requires 8 tracks, it should be assembled on tracks 000 through 007. Thus assembled, the program can be loaded on any 8 sequential tracks on the drum. If the program had been assembled on tracks 058 through 065, it could never be loaded on tracks 000 through 057.

When a large program has been coded in sections, it is advisable to assemble the most frequently used routines first in order to give them the best optimization. After all sections are operating correctly, they can be combined on a single tape or re-assembled into a single program. A detailed discussion on sectional assemblies is given in Chapter 6.

THE ASSEMBLED PROGRAM 2

During the assembly process ROAR produces a relocatable, hexadecimal program tape and, if desired, a decimal listing of the program.

PROGRAM TAPE

The hexadecimal tape consists of the following:

1. A bootstrap, followed by a length of blank tape.
2. Several tape records that make up the program.
3. A transfer code, if an END pseudo-command was used.

Bootstrap

When the assembled program is read into the computer, it may be positioned anywhere in memory by incrementing the track portion of addresses in the program. The value by which the addresses are incremented is called the modifier. Since the process of locating the program in a specified area of memory is the function of the bootstrap, it will add the modifier to all appropriate addresses as it loads the program. The operator supplies the desired modifier to the bootstrap at the time the bootstrap is loaded into the computer. (See "Operating Procedures for Assembled Programs," Chapter 8.) The bootstrap enters 200 words per minute with or without a modifier.

To verify the accuracy of information being input into the computer, a one-word checksum is inserted by ROAR after every 100 words on the hexadecimal tape. This is followed by a portion of blank tape. The format of the words being entered is 12 hexadecimal characters plus a stop code (*). The bootstrap forms a checksum as it loads the program by summing the first 4 characters, summing the last 8, and adding the two sums. Overflow is ignored. The bootstrap compares the checksum it formed with the previously computed checksum that is punched on tape. If the comparison is successful, reading continues. Otherwise, an error indication is typed and the computer halts.

Tape Records

The format of each tape record (except the final one) is 100 tape words; each word containing 12 hexadecimal characters followed by a stop code (*). The first 4 characters contain the modifier flags and the Location; the last 8 characters are the data (usually an instruction word) to be stored in the specified location. Bits 1, 2, and 3 of the first hexadecimal character are the modifier flags which indicate to the bootstrap whether or not a particular address field (Location, Next-address, Data-address, respectively) can be modified.

When the word is read into the Double-length Accumulator, the data is in the Lower Accumulator, and in the Upper Accumulator are the Location (at a "q" of 31) and the modifier flags (in bit positions 16, 17, and 18).

The last tape record will consist of 100 tape words or less. The tape words in this record are in the same format as the previous words except that the last one will be a 16-character transfer code.

Transfer Code

The transfer code is an instruction output by ROAR as the result of an END pseudo-instruction. It indicates to the computer the location of the next instruction to be executed, i.e., the transfer location. The transfer code is a double-length, hexadecimal word containing:

sign-bit	at q = 0
modifier flag	at q = 3
99 (decimal)	at q = 17
transfer location	at q = 62

The decimal 99 is an indication to the bootstrap that the next word is the last word on the tape. That last word is the final checksum.

DECIMAL LISTING

Unless the programmer dictates otherwise, ROAR will type a decimal list of the program as it is being assembled. The list will contain one instruction per line, each showing the following:

1. The symbolic instruction as it was read by ROAR.
2. The decimal equivalent of the assembled instruction.
3. The Comments field of the symbolic instruction.

ROAR allows the programmer to select the input and/or output devices that will be used during an assembly. Thus, he may have ROAR bypass the typewriter output completely or, if an RPC-4600 Auxiliary Tape Typewriter System is available, have the decimal listing punched on tape to be typed at a later time. Use of either option will reduce the time required for an assembly by one-half. (Selection of input and output devices is explained in Chapter 8.)

INSTRUCTION TIMING AND OPTIMIZATION 3

In the RPC-4000 the sequence in which instructions are executed is controlled by addresses contained within the instruction word. The time required for completing the operation specified is likewise dependent, in part, on the addresses.

INSTRUCTION CYCLE

A complete instruction cycle begins with the memory search for the instruction word and ends with the commencement of the search for the next instruction word.

The complete cycle consists of four phases:

Phase 1 - Search for memory location specified in Next-address field of Command Register. Requires 1 to 64 word times.

Phase 2 - Transfer contents of this location to the Command Register. Requires 1 word time.

Phase 3 - Search for memory location specified in Data-address field of Command Register. Requires 1 to 64 word times.

Phase 4 - Execution of operation. Requires basically 1 word time but some instructions require additional Phase 4 cycles to complete the operation.

A "word time" is the basic timing unit in the RPC-4000. It is equivalent to 1/64 drum revolution or . 26 milliseconds.

Basically the computer requires a complete cycle— a minimum of 4 word times— to obtain and execute instructions. However, some instructions do not require a memory search for data (Phase 3); some require an extended Phase 3 operation; and some require an extended Phase 4.

The following instructions do not require a memory search for data and require only 1 word time in Phase 3.

HLT	(Halt)	SRL	(Shift Right or Left)
SNS	(Sense)	SLC	(Shift Left and Count)
CXE	(Compare Index Equal)	PRD	(Print from Data-address)
LDX	(Load Index Register)	PRU	(Print from Upper Accumulator)
INP	(Input)		

An exception to the timing of Phase 3 for the PRD and PRU instructions occurs when the I/O interlock indicates non-readiness of the output device. The computer will hold in Phase 3 until the interlock releases; then the output command is executed.

The instructions EXC (Exchange) and MPT (Multiply by Ten) do not require a memory search for data either. However their Phase 3 time depends on the value in the Data-sector field of the instruction word. That is, the Data-sector value may specify that the operation is to be performed on a particular word of the 8-word Lower Accumulator. Consequently a Phase 3 wait for the required word in the Lower may be necessary.

Some instructions require an extended Phase 4 to complete their operations. The total Phase 4 time for these is as follows:

DVU	(Divide Upper)	67 word times
DIV	(Divide)	67 word times
SRL	(Shift Right or Left)	4 word times plus 1 word time for each bit position shifted
SLC	(Shift Left and Count)	4 word times plus 1 word time for each bit position shifted
MPY	(Multiply)	67 word times

The Phase 4 time required for the INP (Input), PRD (Print from Data-address), and PRU (Print from Upper) instructions depends on the speed of the selected input/output devices.

There are two instructions which do not follow the normal phase sequence: TMI (Transfer on Minus) and TBC (Transfer on Branch Control). When a successful test is made during Phase 3 of a TMI or TBC instruction, Phase 4 of that instruction and Phase 1 of the following instruction are bypassed, and the computer advances directly to Phase 2. Consequently transfer instructions may require as little as 2 word times. Unsuccessful tests require all 4 phases to complete their operations.

All instructions other than those discussed require 1 word time for completion of Phase 4.

INSTRUCTION TIMING

From the foregoing explanation of the computer's command execution cycle it can be seen that the Data-address and Next-address portions of the instructions greatly influence the speed with which they operate. Consider the following instruction:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
9,1,0	R,A,U	9,1,2	9,1,4	

FIGURE 3.1 INSTRUCTION TIMING

Phase 2 in the execution of this instruction consists of loading the instruction into the Command Register. Since the instruction is in Sector 10 of a track, Phase 2 occurs during Sector 10 time. Phase 3 consists of a wait while the drum rotates until the next sector to be read is the same as the one specified in the Data-address part of the instruction. Since the minimum time for Phase 3 is 1 word time, the earliest the computer can read a word in the location specified by the Data-address is at a sector time that is 2 greater than the sector in which the instruction is located. The sample instruction has a Data-address

such that the Phase 3 time is at a minimum, and the instruction with respect to the Data-address is said to be optimum. Considered step-by-step, the timing of the instruction looks like this:

Phase	Sector (time)	Activity
1	09	Search for instruction in sector specified by Next-address of preceding instruction. The next sector is it.
2	10	Transfer contents of Sector 10 to Command Register.
3	11	Search for operand in sector specified by Data-address of this instruction. The next sector is it.
4	12	Contents of 912 to Upper Accumulator.

Note that the Next-address referred to in Phase 1 belongs to the instruction preceding the instruction under consideration. It is in the Command Register as a result of the previous instruction having been executed. It should not be confused with the Next-address belonging to the instruction. in the example; its Next-address will be in the Command Register at the completion of Phase 2. This can be made clearer if one considers two consecutive instructions beginning with Phase 2 of the first.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
910	RAU	912	914	
914	ADU	918	920	

FIGURE 3.2 INSTRUCTION TIMING

Phase	Sector (time)	Activity
2	10	Load contents of 910 into Command Register.
3	11	Search for 912. Next sector is it.
4	12	Contents of 912 into Upper Accumulator.
1	13	Search for 914. Next sector is it.
2	14	Load contents of 914 into Command Register.
3	15	Search for 918. Wait.
	16	Wait.
	17	Wait. Next sector is 18.
4	18	Add contents of 918 to Upper Accumulator.
1	19	Search for 920. Next sector is 20.

And so on.

In the example just explained, Phase 3 and 4 of the first instruction and Phase 1 and 2 of the second instruction have a minimum wait time or latency of zero, and these portions of their execution cycles are said to be optimum. Phase 3 of the second instruction has a latency of 2 (word times), and is not optimum. Although "optimum" is grammatically an absolute degree, in the discussion of programming techniques it is considered to be relative; i. e., certain conditions can cause some instructions to be more or less optimum, with regard to their timing, than other instructions.

The preceding examples contain instructions that require a full computer cycle for their operation. Those instructions which do not require a full cycle vary only slightly. Assuming the contents of Location 707 to be a negative value, consider the following instructions:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
705	RAU	707	709	
709	TMI	711	715	
711	ADU	713	715	

FIGURE 3. 3 INSTRUCTION TIMING

Phase	Sector (time)	Activity
2	05	Load contents of 705 into Command Register.
3	06	Search for 707. Next sector is it.
4	07	Contents of 707 into Upper Accumulator
1	08	Search for 709. Next sector is it.
2	09	Load contents of 709 into Command Register.
3	10	Content of Upper is negative. Search for 711. Next sector is it.
2	11	Load contents of 711 into Command Register.
3	12	Search for 713. Next sector is it.
4	13	Add contents of 713 to contents of Upper.
1	14	Search for 715. Next sector is it.

HOW ROAR OPTIMIZES

Fortunately for the programmer ROAR takes care of most of the routine work of substituting optimum addresses for symbolic or blank addresses. For the majority of instructions in the assembled program, the addresses ROAR assigns will be truly optimum. There will be instances, however, when ROAR's optimization can be improved. It is obviously impossible for ROAR to anticipate all the ways in which an instruction is used in a program; and since ROAR is a one-pass assembler, which completely assembles one instruction before going on to the next, it cannot determine by inspection how an instruction will interact with those that follow.

Later on, this manual will explain a number of convenient features that permit the programmer to intervene in ROAR's normal optimization process. For now, it is important to see how ROAR goes about optimizing a program. Basically, ROAR uses a few simple rules built into its logic, a Symbol Table in which each symbol is stored, a Command Code Table for classifying instructions as to optimizing type, and an Availability Table that tells it whether an address has already been assigned to an instruction.

Symbol Table

The Symbol Table (Appendix B) is a block of memory where all the symbolic addresses used in a program are stored by ROAR during the assembly of that program. By referring to this table, ROAR determines whether a symbol is a new one or has been previously encountered. If the symbol is new, it is stored in the table according to a formula which enables ROAR to determine the absolute address that should be assigned to that symbol. If the symbol is not new, ROAR will find it present in the Symbol Table and will be able to locate its equivalent absolute address.

Availability Table

The Availability Table (Appendix B) consists of a block of words wherein each bit position of each word corresponds to a memory address, and the contents of the position signifies the availability status of the address. During assembly of a program, ROAR computes the optimum addresses to substitute for the symbolic addresses in each instruction. Before making the substitutions, ROAR checks the availability status of the addresses. If the Availability Table shows these addresses have already been assigned to another instruction or have been reserved for some other purpose, ROAR will repeat the process of computing new addresses and checking their availability until a successful assignment is made.

Command Code Table

The Command Code Table (Appendix C) shows the optimizing classification ROAR assigns various RPC-4000 commands and the values used in computing optimum addresses. The commands are classified as Type 1, 2, or 3. Type 1 indicates that the Data-address field contains the location of data rather than data itself. Type 2 indicates that the Data-address field contains data instead of a location. Type 3 indicates a shift command. The three classifications determine how the Data-address and Next-address fields will be optimized.

Also listed in the Command Code Table are the special pseudo-commands that ROAR handles in the same manner as RPC-4000 commands. They are HEX, DEC, ALF, PRC, SRT, and SLT. (See Chapter 5, "PSEUDO-INSTRUCTIONS.")

Optimizing Rules

ROAR optimizes each field individually, starting with the Location and continuing through the Next-address field. If the Location field contains a symbol which has been encountered previously, ROAR will assign the same address it assigned before. If the symbol has not been encountered previously, ROAR will assign it the same sector (but a different track) as was assigned to the preceding Next-address field. If the Location field is legally blank, ROAR will assign it the same address as was assigned the preceding blank Data-address or Next-address field. If the Location field is blank when it should be filled, ROAR checks the Order field to determine if a general pseudo-command (i.e., a non-special pseudo-command) is present. If one is present, ROAR will perform as required by the pseudo-command; if one is not present, ROAR will type an error indication and then halt.

After the Location field has been assembled, ROAR processes the Order field. The content of the Order field is compared with the Command Code Table. If the comparison is successful, an RPC-4000 command or a special pseudo-command is present. ROAR will note its classification, assemble the command, and continue. If the comparison is not successful, the Order field did not con-

tain a command or a special pseudo-command; and ROAR checks to see if it contains a general pseudo-command. If so, ROAR will perform the required function and then continue; if not, ROAR will type an error indication and halt.

Following the assembly of the Order field, the Data-address field is optimized relative to the Location. If the command is a Type 1 and the Data-address field is blank or contains a symbol not previously encountered, ROAR will optimize it as Location plus 2. If the symbol had been encountered before, ROAR will assign it the same address as had been assigned previously. If the command is a Type 2 or 3, the Data-address field is assembled directly and is not subject to the optimization rules. For example, in a shift instruction the Data-address field contains a value indicating the number of positions to be shifted. ROAR does not replace this value with an address, but merely converts it to a hexadecimal number and proceeds to the Next-address field.

The Next-address field is the last to be assembled. If the Next-address field contains a symbol which had been encountered previously, it is assigned the same address as before. If the field is blank or contains a symbol not previously encountered, the Next-address field is assembled according to the classification of the command. A Type 1 command results in the Next-address being optimized as Data-address plus Constant. (The Constants, which vary depending on the specific command, are listed in the Command Code Table, Appendix C.) A Type 2 command causes the Next-address to be optimized as Location plus Constant. A Type 3 command results in the Next-address being optimized as Location plus Data-address plus Constant, except the SLC command which is optimized as Location plus Constant. (The programmer should provide a delay to account for the maximum number of shifts anticipated.)

The Comments field is not considered since it is only copied from the symbolic input tape to the decimal listing, and does not influence the assembled instruction.

CODING SHEET FOR ROAR

Figure 4. 1 shows a coding sheet which was designed specifically for ROAR-language programming. It is divided into columns corresponding to the five fields which most ROAR instructions require: Location, Order, Data-address, Next-address, and Comments.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS

FIGURE 4. 1 ROAR CODING SHEET

LOCATION FIELD

ROAR will accept as a Location any symbol having no more than 6 characters, including regional notation and absolute decimal addresses. In some cases the Location must be left blank, i. e., when the instruction word contains a pseudo-command and also when either the Data-address field or the Next-address field of the previous instruction word was blank. A stop code must follow this field even when it is blank.

ORDER FIELD

Two types of commands may be written for this field: commands for computer instructions and pseudo-commands. Pseudo-commands are so-called because they are not interpreted by the computer but are directed to the ROAR program. They may result in an instruction to the object program, supplement other computer instructions, or cause ROAR to perform certain utilitarian functions which facilitate the writing of a program. The pseudo-commands are explained in Chapter 5.

Commands for computer instructions may be written on the coding sheet in either symbolic (mnemonic) form or in absolute machine language (numeric) form. ROAR will accept both forms of commands.

Indexed Orders

If an instruction is to be indexed, write an X immediately to the left of the 2- or 3-character order code.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	,0,2			RESET ADD UPPER
	X,0,2			INDEXED RESET ADD UPPER
	,R,A,U			RESET ADD UPPER
	X,R,A,U			INDEXED RESET ADD UPPER

FIGURE 4. 2 FORM OF INDEXED INSTRUCTIONS

ROAR will accept indexed numeric orders from X00 through X31. The numbers 0 through 9 may be used in place of the decimal order codes 00 through 09, but may not be indexed if written in that form.

ROAR ADDRESSING

The Location, Data-address, and Next-address fields may contain an absolute address or a symbol representing an address to be assigned by ROAR. All addresses in the object program will be converted by ROAR to track and sector form. In Figure 4. 3 the addressable fields are marked with a left-hand character "A" followed by five "B" characters. The B characters show the placement of either an absolute address or a symbolic address containing no more than five characters. If an address contains both an A character and B characters it is interpreted by ROAR as a special six-character address. There is also a form of special address which contains an artificial sector number 90 through 99. Special ROAR addressing will be explained further on in the Chapter. First the two types of non-special addresses, symbolic and numeric, shall be examined.

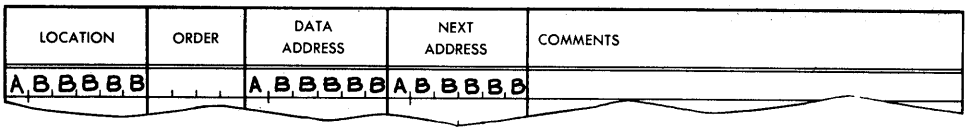


FIGURE 4.3 ADDRESSABLE FIELDS

Symbolic Addresses

Symbolic addresses must consist of one to five characters, at least one of which must not be numeric, that is, 0 through 9. The non-numeric characters are shown in the table of alphanumeric codes (Appendix E) as codes greater than 25.

Symbolic addresses are assigned absolute track and sector equivalents by ROAR when they are first encountered, and they retain this equivalence throughout the assembly of the program. When a track and sector has been assigned to a symbolic address it is made unavailable for assignment to any other symbolic address unless, of course, the subsequent symbolic address is a duplicate of one already encountered.

Some typical symbolic addresses are shown in Figure 4.4.

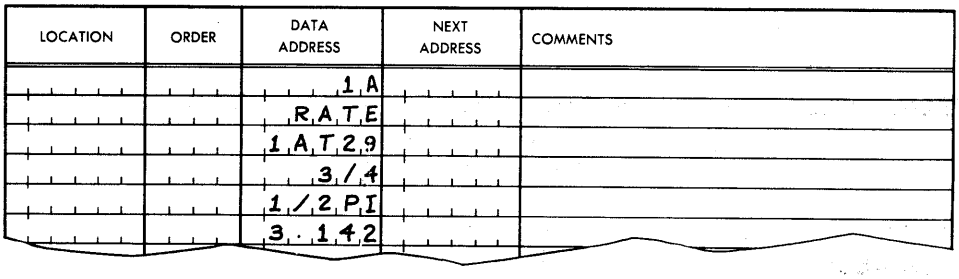


FIGURE 4.4 TYPICAL SYMBOLIC ADDRESSES

Numeric Addresses

Numeric addresses are always in track and sector form and, excepting some special cases, refer to absolute memory locations. It is not necessary to include leading zeros in numeric addresses. ROAR will accept any five digits as a legal address except that track numbers will be assembled modulo 128, and sector numbers modulo 64. A track and sector address of 12373 will be converted by ROAR to a mod-64 sector number on the next higher track, namely 12409. An address of 12764 will yield track 0, sector 0; 13176 will yield 00412, etc.

If numeric addressing of data or instructions is employed in coding a program, certain precautions are necessary. ROAR will not automatically refrain from assigning these absolute numeric addresses to symbolic addresses it encounters. Consequently absolute addresses must be reserved in advance of the assembly of the program by employing the RES pseudo-command. Stated simply, it will cause ROAR to change the status of those bits in the Availability Table corresponding to the address, so that the address becomes "unavailable" for assignment by ROAR. The RES pseudo-instruction is explained in Chapter 5.

A hexadecimal address will not be accepted as such by ROAR, but will be interpreted as some symbolic or absolute address.

BLANK ADDRESSES

Address fields may be left blank subject to the following rules:

1. For any instruction word, either the Data-address or Next-address field may be left blank, but not both.

2. When one of these is left blank the Location field of the following instruction must also be left blank.

3. If both the Data-address and Next-address fields are occupied, the Location field of the following instruction cannot be blank, but must contain some symbolic or absolute location.

When either the Data-address or Next-address is left blank, ROAR will assign it an available memory location (which is then made "unavailable"), and also will assign that address to the blank Location of the following instruction. These rules do not apply to the general pseudo-instructions, but to the special pseudo-instructions (which result in an output to the assembled program tape), namely those containing the pseudo-commands HEX, DEC, ALF, PRC, SLT, or SRT.

Obviously, if a location is to be referred to from more than one place, it must be assigned a symbol rather than being left blank. Blank addresses, in effect, are symbolic addresses with the symbol implied.

SPECIAL ADDRESSES

Special addresses are of two types: those employing an artificial sector number 90 through 99, and those having a character in the left-most position of the Location, Data-address, or Next-address fields.

Artificial Sector Addresses

When an instruction involving the Lower Accumulator is executed and the Lower is in Eight-Word Mode, the particular Accumulator word used will correspond to the modulo-8 equivalent of the Data-sector specified by the instruction. To put it another way, if L_0, L_1, $L_2 \ldots L_7$ designate the eight Lower Accumulators, then an instruction with a Data-address of 10300 would use or affect L_0, as would 10308, 10316, etc. Likewise, instructions with Data-address of 02703, 05403, 06711, 11819, etc., would affect L_3.

If a program is being coded in absolute numeric language, there is no difficulty in choosing the proper Data-sector to write on the coding sheet. One merely specifies the sector number which corresponds to the particular word of the 8-word Lower desired, which will result in minimum latency during the instruction's Phase 3 time. If the program is coded in ROAR language, however, it is not possible to specify beforehand the optimum sector number for the Data-sector. This depends on the sector ROAR assigns to the location part of the instruction.

To circumvent these and similar difficulties, provision has been made in ROAR for specifying which sector it will assign to an address or location. It is unnecessary to control ROAR's assignment of track numbers since this part of an address has no bearing on an instruction's timing. As far as the execution of instructions is concerned, switching from one track to another by the computer occurs instantaneously. This means that two instructions having latency zero with respect to their sector locations, but on different tracks, will operate at the same speed as if they were on the same track.

The artificial sector numbers 90 through 99 are used for controlling ROAR's assignment of sector numbers. When read by ROAR in the Data-sector field

of an instruction, as one example, they will be interpreted as follows:

90-97 The second digit refers to a prime sector (corresponding to L_0, L_1, etc. of the Lower Accumulator and to 00, 01, etc. of the Recirculating Track). ROAR will assign the next optimum, modulo-8 equivalent of this sector to the Data-sector part of the instruction word. As an example, if the ROAR-computed optimum sector for the Data-address of a given instruction is 34 and the given sector is 93, ROAR will assign 35 to the Data-sector because this is the next sector (after 34) which is a modulo-8 equivalent of prime sector 3. (See Modulo-8 Table, Appendix D.)

98 ROAR will assign the most optimum sector to the Data-sector part of the instruction word (Figure 4.5).

Assume data is stored in Region A. Word 3 is to be brought to the Upper Accumulator and Word 5, to the Lower. It is desired to accomplish this in less than 1 drum revolution.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	E,X,C	1,6,9,8		SET LOWER TO 8-WORD MODE
	L,D,C	T,H,R,E		LOAD COUNT OF 3
	R,A,L	/ 0,0,0,0,3		BRING SECTORS 3 - 6 OF REGION
	E,X,C	2,9,3		EXCHANGE WORD 3 TO UPPER
	E,X,C	3,2,9,5		SET LOWER TO 1-WORD MODE LEAVING
				WORD 5 IN L.

FIGURE 4.5 SECTORS 90-98 FOR SPECIAL ADDRESSES

99 ROAR will assign the same sector to the Data-sector part of the instruction as it computes for the Location field of the instruction. Therefore the Data-sector will have a latency 64. Sector 99 is often used in the Data-address of "print" instructions to make the I/O interlock operative and thereby prevent the execution of a subsequent print order until execution of the first is complete. If sector 99 is not used in coding a Print instruction, ROAR will assign an optimum sector to the Data-address and the interlock will not be operative (Figure 4.6).

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	P,R,D	1,9,9		CARRIAGE RETURN
	P,R,D	3,0,9,9		PRINT E

FIGURE 4.6 SECTORS 98, 99 FOR SPECIAL ADDRESSES

Note that these artificial sector numbers are normally used in the sector part of the Data-address or Next-address fields. If used in the sector part of the Location field, that location cannot be referred to by some other instruction. As an example, if one instruction is given location "NOW94", and another in-

struction uses NOW94 as a Next-address, ROAR will not assign to this Next-address the absolute address of the first instruction. Instead ROAR will interpret both as special sector 94 addresses and will assign each a different optimum, modulo-8 equivalent of prime sector 4.

Special Character Addresses

The second category of special addresses is made up of those containing a special character in the left-hand column of the Location. Data-address, or Next-address fields. These special addresses are:

Regional address
Recirculating Track address
Double-access Track address
Skip address
Delayed address

REGIONAL ADDRESSES A region is a block of memory between and including specified initial and final locations (see REG, Chapter 5). ROAR contains a one-track Region Table in which the 64 symbols which can be entered into the computer are stored. When a region is defined, the location of the first word of the region is stored in the Region Table beside the specified symbol. Whenever reference is made to a location within a region, ROAR searches the Region Table, finds the base address designated for the region, and computes the desired absolute address. A member of a region is addressed relative to the first location in the region.

Addressing relative to the first location in the region consists of writing a single character region symbol and a five digit sequence number. The symbol enables ROAR to locate in the Region Table the initial location of the region. The five-digit number indicates the word location relative to that first location. Thus a Regional Address T00012 indicates word number 12 in Region T. If, for example, Region T has been set up as locations 02500 through 02763, some acceptable Regional Addresses would be as follows:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
		T,0,0,0,0,1		02500
		T,0,0,0,6,4		02563
		T,0,0,1,9,2		02763
		T,0,0,2,0,0		WOULD BE ASSIGNED AS 02807 EVEN
				THOUGH IT IS 8 SECTORS BEYOND
				THE LIMITS OF THE T REGION.
		T,0,0,0,0,0		ASSIGNED AS 02463.
		T,0,8,1,9,2		BECAUSE REGION STORAGE IS CALCULATED
				MODULO 8192, THIS IS ALSO 02463.
		T,0,8,1,9,1		ASSIGNED AS 02462.

FIGURE 4.7 SAMPLE REGIONAL ADDRESSES

If an instruction using a Regional Address contains an undefined region symbol, ROAR will indicate an error and then halt.

RECIRCULATING TRACK ADDRESSES Addresses referring to the Recirculating Track 127 are coded in ROAR language by writing the symbol "RECRC" followed by a prime, modulo-8 sector number. Figure 4.8 illustrates the form of a RECRC address.

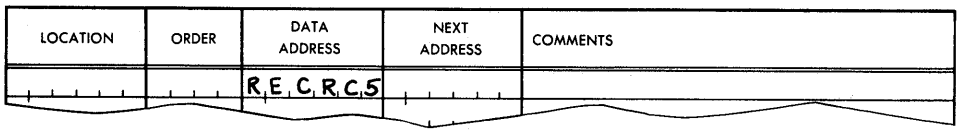


FIGURE 4.8 RECRC ADDRESS

Upon reading an address in this form ROAR will assign to the address the next, optimum, modulo-8 equivalent of the prime sector. If RECRC5 is coded as the Data-address of an instruction, and the ROAR-computed optimum sector for this address is 18, it will assign an address of 12721. This is because sector 21 is the next sector (following 18) which is a modulo-8 equivalent of prime sector 5. (See Modulo-8 Table, Appendix D). The modulo-8 equivalents of prime sector 5 are 5, 13, 21, 29, 37, 45, 53, 61. Assume the optimum sector for the address is 30. Therefore the modulo-8 5 sector following the optimum sector of 30 is sector 37, and ROAR would assign an address of 12737.

The RECRC(n) notation may not be used as the location of an instruction with the anticipation that the location may be referred to in the address of another instruction. When ROAR reads the RECRC address it will not equate it with the RECRC location of the prior instruction. Instead ROAR will treat the RECRC address as it always does, and assign a mod-8 equivalent of the prime sector specified.

Furthermore, when ROAR reads RECRC(n) in the Location field of an instruction, the instruction will not be punched into the hexadecimal, program tape being produced. However, the instruction will be printed on the program listing. With these restrictions borne in mind it is possible to use RECRC(n) in the Location field to achieve some special results in the assembled program.

The RECRC(n) notation when written in the Location field has the effect of setting the sector part of ROAR's internal Location Counter to a predetermined value. ROAR assigns optimum addresses based on the value in the Location Counter. When ROAR assigns an address it puts the number of the sector in the Counter and thereby keeps track of what sector it assigned last. Assume that one wishes to write an instruction that is to use the Recirculating Track, and it is desired to have the instruction optimized so that it operates in the least amount of time.

	Location	Order	D-Address	N-Address
A. Say the instruction is	RATE	CLL	RECRC0	ANY
ROAR will assemble as	01110	27	12716	01118
B. If, however, we precede the instruction with	RECRC2	RAU	98	RATE
Which will assemble as	12718	02	01120	01122
C. Then the instruction we are concerned with	RATE	CLL	RECRC0	ANY
Will assemble as	01122	27	12724	01126

The assembled instruction in Example A is not optimum. There is a difference of 6 word times between the Location-sector and the Data-sector, instead of the optimum 2 word times. The assembled instruction in Example C is optimum.

Another way to preset the Location Counter is to write an instruction of the form

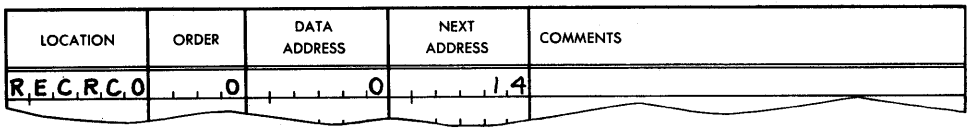


FIGURE 4.9 PRESET LOCATION COUNTER

An instruction with RECRC0 as a Location, with zero in the Order and Data-address fields, will set the Location Counter to the value contained in the Next-sector field.

DOUBLE-ACCESS TRACK ADDRESSES During the assembly of a program ROAR will not arbitrarily assign Double-access Track addresses to Location or Address fields. If it is desired to let ROAR assign these tracks in the same way it assigns other memory locations, they may be made available to ROAR with the pseudo-command AVL as explained in Chapter 5.

The normal way of referring to the Double-access Tracks is by means of the ROAR Double-access Address. This symbolic address is illustrated in Figure 4.10.

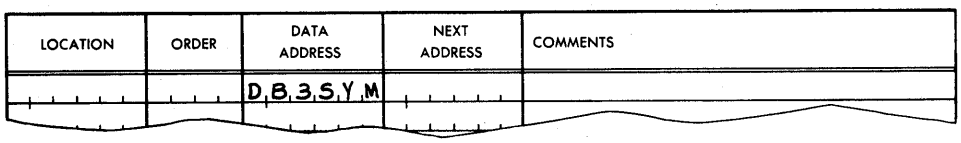


FIGURE 4.10 DOUBLE-ACCESS ADDRESS

The address consists of three parts:

A. The letters DB with the D in the special character column. This identifies the form of the address for ROAR.

B. The number following DB may be a number from 1 through 4. It represents a Double-access Track number as follows:

Number	Meaning
1	Leading head of the 16 word track (123)
2	Trailing head of the 16 word track (125)
3	Leading head of the 24 word track (124)
4	Trailing head of the 24 word track (126)

C. The letters SYM represent any 3-character symbol capable of being read into the computer.

This three character symbol has the same significance as any other symbolic location except that once it has been used in conjunction with a particular Double-access Track it should continue to be referred to in conjunction with that track or the associated track. DB Tracks 1 and 2 are associated, as are 3 and 4. (Just as Tracks 123 and 125 are associated, and 124 and 126 are associated.) Thus DB1VAR and DB2VAR refer to the same word on the 16 word track except that the DB2VAR reference is 16 word times later than DB1VAR.

To illustrate all this more clearly, consider the following sequence, given a variable in the Upper Accumulator at a q of 7.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
EX	STU	DB1VAR		STORE TEMPORARILY
	SRL	1		RIGHT ONE BIT
	STU	VAR		VARIABLE @ 8
	RAU	DB2VAR	GO ON	REPLACE VARIABLE @ 7

FIGURE 4.11 DOUBLE-ACCESS ADDRESSING

The instruction in EX temporarily stores the variable in the first available sector of the 16 word Double-access Track using the leading head.

The Accumulator content is shifted right by 1 bit and the variable, now at a q of 8, is stored in Location VAR. ROAR will not confuse this location with the VAR in DB1VAR or DB2VAR.

The fourth instruction restores the variable, at q of 7, to the Accumulator. The variable is read by the trailing head of the 16 word track, i.e., 16 word times later than it will be stored in DB1VAR.

ROAR might have assembled the sequence as follows:

```
EX*STU*DB1VAR**          00105  24  12307  00009  *
*SRL*1**                 00009  12  00001  00217  *
*STU*VAR**               00217  24  00019  00021  *
*RAU*DB2VAR*GO ON*       00021  02  12523  00125  *
```

It should be noted that ROAR assigned the address 12307 to DB1VAR and 12523 to DB2VAR. On subsequent encounters of the DB1VAR symbol ROAR may assign either 12307 or 12523, whichever results in the least latency. This is permissible because both addresses refer to the same word.

SKIP ADDRESS At times, for purposes of optimization, it is desirable to place an instruction in a location beyond the computed optimum. One example of this would be where the Data-address is variable and may be set to pick up the contents of one of several consecutive locations. We would like the following instruction to be located so that it is optimum when the last sector of the set is used, rather than the first. The Skip Address (Figure 4.12) enables this to be done.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
		SKIP12		

FIGURE 4.12 SKIP ADDRESS

The symbol SKIP with S in the special character column identifies the Skip Address. The 2 digit number specifies the number of sectors to be skipped beyond the optimum sector that would ordinarily be assigned to the field in which the Skip Address symbol appears.

The following example will illustrate a typical use of the Skip Address. Assume locations 10520 through 10525 have been reserved for a set of parameters and have been defined as Region Q. The first has been chosen for use and its address placed in Location CODE which ROAR may assign as 00318.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
CODE	XRAU	Q00001	SKIP05	PARAMETER TO U
	MPY	X	GO ON	

FIGURE 4.13 SKIP ADDRESSING

For these instructions ROAR might generate:

```
CODE*XRAU*Q00001*SKIP05*        00318  X02  10520  00027   *
*MPY*X*GO ON*                   00027   14  00029  00033   *
```

When the assembled program is operated and the instruction in CODE (00318) has been executed for the last time, the Data-address will contain 10525 and the MPY instruction will then be optimum.

DELAYED ADDRESS The last of the special 6-character addresses is the Delayed Address (Figure 4.14). It is used only with the pseudo-command NXT (see Chapter 5).

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	NXT	DELAYD		

FIGURE 4.14 DELAYED ADDRESS

DELAYD may appear in the Data-address field, Next-address field, or both; the Location field is left blank. This pseudo-instruction causes ROAR to change the optimization of the corresponding field of the instruction following it. That is, ROAR will optimize that address as the preceding Next-address sector plus 4. This is possible only if the address field is blank or contains a symbolic address which has not been previously seen by ROAR.

Consider:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
GO	RAL	INST	SQRT	
	NXT	DELAYD		
INST	STU	RTX	B	

FIGURE 4.15 DELAYED ADDRESSING

The symbol GO will be assigned an optimum address; the symbol INST will be assigned an address optimum with respect to GO. The symbol SQRT will be optimized with respect to INST and that sector value stored. When ROAR reads the Data-address field of the INST instruction, it will optimize that field as SQRT+ 4. The above example might be assembled as:

```
GO*RAL*INST*SQRT*        03633   03      03635   03737   *
*NXT*DELAYD**                                            *
INST*STU*RTX*B*          03635   24      03741   03743   *
```

In effect the DELAYD caused ROAR to delay the optimization of the Data-address field of the INST instruction (03635) and to optimize it with respect to the sector assigned to SQRT (i. e. , 37+ 4 = 41).

This special address is effective in optimizing the instruction to which a subroutine will exit. (See Chapter 6, "PROGRAMMING TECHNIQUES. ")

COMMENTS FIELD

The fifth field on the coding sheet is the Comments Field. ROAR will copy any information in this field onto the decimal listing. The comments will appear to the right of the decimal representation of the assembled instruction word. A stop code must follow this field even if the field is blank.

PSEUDO-INSTRUCTIONS 5

The description of each pseudo-instruction on the following pages includes:

1. A chart showing what each field must contain.
2. A discussion of how it is processed by ROAR.
3. An explanation of its effect on other pseudo-instructions.
4. An example of its use.
5. A list of error printouts that may result from its misuse.

Most pseudo-instructions have the standard format discussed in Chapter 4, i.e., 5 fields, each followed by a stop code. However, a few pseudo instructions deviate from this format; viz., PRE, SET, RAV, COM, (56)(57)(56), SBT, SUB, and SBE. The differences in format for these are explained in the detailed descriptions of the pseudo-instructions.

The thirty-four pseudo-instructions fall into seven classifications:

ALLOCATE MEMORY

At the beginning of the assembly, all of the computer's memory is available to the program. Normally, it will be necessary for the programmer to reserve various parts of memory for data storage, subroutines, etc. ROAR provides three pseudo-instructions to reserve areas of memory. They may be used at any time, but normally will be among the first instructions assembled. These pseudo-instructions are RES, REG, and RLR.

It is often desirable to pre-establish absolute addresses for certain symbols, for example 5-character symbols for communication between parts of a program. There are four pseudo-instructions which are used for this purpose: PRE, SET, RST, RRS.

Occasionally it may be necessary to make available a previously reserved area of memory or to set certain symbols equal to specific addresses. Three pseudo-instructions have been provided to accomplish these tasks: AVL, EQR, EQV.

CONTROL ASSEMBLY

A group of four pseudo-instructions provides the programmer with means of controlling the progress of the assembly; i.e., pausing, restarting, altering memory, terminating. This group includes NIX, NEW, CLS, and END.

When a program is constructed in many segments, or makes use of symbolically-coded subroutines, it becomes difficult to avoid duplication of symbolic addresses. Two pseudo-instructions have been provided which allow use of identical symbols in different subroutines without being ambiguous to ROAR when it makes memory assignments. They are TAG and HED.

Should a programmer desire to control the optimization of an instruction by increasing the latency in the Data-address or Next-address field, he may do so with an NXT pseudo-instruction.

ESTABLISH CONSTANTS

The assembly program provides the programmer with four methods of setting up constants for his program. He may convert the constant to the form of an instruction and place it on the coding sheet as such, or he may use any of the three pseudo-instructions HEX, DEC, ALF.

INPUT-OUTPUT

Six pseudo-instructions can be used to obtain reference material for use in program checkout and change. In some cases their output may be used to advantage in partial re-assemblies. They include PAV, PPA, RAV, PST, 5CS, and PAS.

A pseudo-instruction has been included to facilitate the coding of print instructions. It is PRC.

SHIFT

Two pseudo-instructions have been included for convenience in the coding of shift instructions: SRT and SLT.

COMMENT

Two pseudo-instructions have been added to allow unusually long statements to be inserted in the symbolically coded program. They are COM and (56)(57)(56).

USE SUBROUTINE LIBRARY

A Subroutine Library is a tape made up of frequently used, symbolically coded programs, e. g. , data input and output programs, trigonometric functions, etc. Selecting and assembling the desired programs from the Library Tape is handled entirely by ROAR. The pseudo-instructions relating to the use of a Subroutine Library are SBT, SUB, and SBE.

ALLOCATE MEMORY

RES — Reserve a Portion of Memory

Location - Must be blank.
Command - RES
Data-address - Any legal decimal address having no more than 5 digits and designating the first location of the reserved area.

Next-address - Any legal decimal address having no more than 5 digits and designating the last location of the reserved area.
Comments - Any applicable remarks.

FUNCTION This pseudo-instruction will make a designated number of sectors unavailable to ROAR for assignment to symbolic and blank addresses. The absolute address in the Data-address field designates the first location to be reserved. The absolute address in the Next-address field designates the last location to be reserved. All locations between and including these addresses are reserved. The sectors in the reserved area can be referred to by absolute addresses only.

Normally, RES will be one of the first pseudo-instructions processed by ROAR. The use of RES allows the programmer to restrict his program to desired areas or to reserve for other uses any area in memory. If the Data-address field of an RES pseudo-instruction is greater than the Next-address, ROAR will begin the reservation at the location specified by the Data-address, continue to 12763, then proceed with 00000, and continue until the location specified by the Next-address is processed. (See last example below.)

All sectors included in the bounds of an RES pseudo-instruction remain reserved until reinstated in the Availability Table by an AVL pseudo-instruction or until ROAR is initialized to begin a new assembly.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	RES	6,0,0	1,0,6,3	RESERVE ALL SECTORS IN TRACKS
				6,7,8,9, AND 10.
	RES	1,4,0,0,0	2,0,7,5	RESERVE ALL SECTORS FROM 1200
				THROUGH 2111.
	RES	1,2,0,0,0	1,6,3	RESERVE ALL SECTORS FROM
				12000 THROUGH 12763 AND FROM
				00 THROUGH 163. THIS INCLUDES
				THE DOUBLE-ACCESS TRACKS AND
				RECIRCULATING TRACK, ALTHOUGH
				IT IS UNNECESSARY TO RESERVE
				THESE TRACKS UNLESS THEY
				HAVE PREVIOUSLY BEEN MADE
				AVAILABLE WITH AN AVL.

ERROR PRINTOUT REFERENCE Address not decimal for pseudo-op REG RES or AVL. See "Error Printouts," Chapter 8.

REG —— Establish a Region

Location - Must be blank.
Command - REG
Data-address - The symbol that is to designate the region, followed by a 5 decimal digit address which will be the first location in the region.

Next-address - An address of not more than 5 decimal digits that will be the final location in the region.
Comments - Any applicable remarks.

FUNCTION An REG pseudo-instruction will establish a region whose first location will be specified by the absolute address in the Data-address field. The five-digit Data-address in the REG instruction word is stored in the Region Table and is used by ROAR to compute absolute addresses when reference is made to the region later in the program. (See "Regional Addresses," Chapter 4.) The final location in the region will be designated by the absolute address in the Next-address field of the same instruction word. All sectors between and including these addresses are reserved for the region and may only be referred to by regional addresses or by absolute addresses.

If the content of the Data-address field of an REG pseudo-instruction is greater than the content of the Next-address field, ROAR will begin the region at the location specified by the Data-address, continue to 12763, then proceed with 00000, and continue until the location specified by the Next-address is processed. However, it would be very difficult to use a region which included the Double-access Tracks or the Recirculating Track. If they were included in a region, ROAR would consider the Double-access Tracks to be 4 individual tracks, each having 64 sectors, rather than only 2 tracks of 128 sectors and would consider the Recirculating Track to have 64 individual sectors rather than 8 sectors repeated 8 times.

All sectors affected by an REG pseudo-instruction are arbitrarily included in the region regardless of prior use by the program being assembled. Any sectors included in the bounds of a subsequent AVL pseudo-instruction will be arbitrarily reinstated in the Availability Table but such sectors may still be referred to by regional addressing. Any locations affected by a prior or subsequent REG, EQR, or EQV will be treated in the manner usual for the REG, EQR, or EQV. That is, a location can be set equivalent to some address (absolute or symbolic) and can also be designated as part of a region. Thus the location could be addressed three ways: regional addressing, absolute addressing, and the address to which it has been equated.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	REG	/0105	7620	RESERVE FOR REGION SLASH (/) ALL SECTORS FROM 1056 THROUGH 7620.
	REG	31620	9798	RESERVE FOR REGION 3 ALL SECTORS FROM 3400 THROUGH 9834.
	RAU	A00001	N-AD	LOAD THE FIRST WORD OF REGION A INTO THE UPPER ACCUMULATOR.
	RAL	30036	N-AD	LOAD WORD NUMBER 360 OF REGION 3 (WHICH IS ALSO WORD NUMBER 1840 OF REGION SLASH) INTO THE LOWER ACCUMULATOR.

ERROR PRINTOUT REFERENCE Address not decimal for pseudo-op REG RES or AVL. Unassigned region. See "Error Printouts," Chapter 8.

RLR —— Designate a Relocatable Region

Location - Must be blank.
Command - RLR
Data-address - The 1-character symbol that is to designate the region.
Next-address - The number of locations to be in the region.
Comments - Any applicable remarks.

FUNCTION At the beginning of an assembly, the second printout from ROAR inquires "Subroutine Tape Region Storage." (See "Assembly Preparations," Chapter 7.) The area reserved at that time is normally for regions used by subroutines to be assembled from a Library Tape. However, the area thus reserved may be used for regions in the object program if the programmer wishes.

After the area for the regions has been reserved, the individual regions are designated with RLR pseudo-instructions. The region symbol is written in the Data-address field, and the number of memory locations required for that region is written in the Next-address field. When an RLR pseudo-instruction is encountered, ROAR will assign sequential memory locations from the subroutine region storage area to the given region. When subsequent RLR pseudo-instructions (designating the same or different region symbols) are encountered, ROAR will assign the next sequential group of locations to those regions. For example, subroutine A has 20 locations designated as Region Slash, subroutine B has 12 locations also designated as Region Slash, and both subroutines are to be assembled from a Library Tape. At the beginning of the assembly, the programmer would reserve as Subroutine Tape Region Storage 32 locations, say 4000 through 4031. Then when ROAR begins the assembly from the Library Tape, the second instruction of subroutine A would be *RLR*/*20** which would establish Region Slash as locations 4000 through 4019. When the assembly of subroutine B begins, ROAR will encounter *RLR*/*12** which would establish the next sequential group of locations as Region Slash, that is 4020 through 4031. In this way, the same region symbols may be used in numerous subroutines without conflict in the absolute addresses ROAR will assign. (See SUBROUTINE LIBRARY TAPE, Appendix A.)

If regions in an object program are established with RLR pseudo-instructions, certain precautions should be observed by the programmer when using regional addresses. Because the sectors actually assigned to a region established by an RLR are unknown, it is hazardous to use regional addressing with the 8-word Lower Accumulator. Since the words in the region might not have been stored modulo-8, the programmer could not refer to a particular word in the 8-word Lower; or if used in conjunction with the Repeat Mode, the 8 words in the Lower Accumulator might not be in the order desired.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	R,L,R	K	81	ESTABLISH REGION K AS THE
				FIRST 81 LOCATIONS IN AREA
				OF SUBROUTINE REGION STORAGE.
	R,L,R	Z	40	ESTABLISH REGION Z AS THE
				NEXT SEQUENTIAL 40 LOCATIONS.

The Subroutine Tape Region Storage area might have been reserved as locations 5600 through 5963. Then, Region K would be 5600 through 5716 and Region Z would be 5717 through 5756.

ERROR PRINTOUT REFERENCE Insufficient subroutine region storage. See "Error Printouts," Chapter 8.

PRE —— Prepare ROAR

Location - Must be blank.
Command - PRE
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION This pseudo-instruction is generated by the Compact compiler (see Compact Operating Procedure, program H3-01. 0) and is the last item output after the program compilation is complete. It contains the name of the subroutine and the information concerning the amount of storage required by that subroutine for each of the six special character regions. The PRE pseudo-instruction and its information must be the first item to enter the assembly routine.

The information output with the PRE pseudo-instruction is in hexadecimal form. The first word is the name of the subroutine. The second and subsequent words are made up of two parts: the first 4 hexadecimal characters give the name of the region; the last 4 hexadecimal characters indicate (at a q of 30) how many memory locations are required for that region.

EXAMPLE

```
*PRE*229E8BAD*34DC000E*36DC003E*35DC0000*3ADC0000*3BDC0000*37DC01DC*

Subroutine INOUT requires

      7 locations for region ,
     31 locations for region [
      0 locations for region =
      0 locations for region +
      0 locations for region -
    238 locations for region ]
```

ERROR PRINTOUT REFERENCE None.

SET —— Establish Global Symbols

Location - Must be blank.
Command - SET
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

ALLOCATE MEMORY

FUNCTION This pseudo-instruction is used to set up the global symbols, which are symbols common to all programs assembled as a single operating program unit and which are used for communication between sections of a program or between subroutines and a program. The advantage of establishing global symbols with an SET pseudo-instruction is that the symbols are not restricted to 5 characters and that the Symbol Table may be cleared of all symbols except global symbols (see RST). SET is not generated by the COMPACT compiler, but is required if any symbols are to be global.

When the SET pseudo-instruction is executed, ROAR places in the Set Table all the symbols which follow it. The Set Table is 4 tracks long and contains only those symbols established by an SET pseudo-instruction. Any number of symbols (up to 256) may be entered with a single SET pseudo-instruction. Each symbol, consisting of no more than 5 characters, is followed by a stop code. The pseudo-instruction is terminated by an extra stop code. When a global symbol is next encountered by ROAR, it is placed in the Symbol Table and is assigned an absolute address.

When assembling a COMPACT-compiled program, the SET pseudo-instruction should immediately follow the PRE pseudo-instruction. However, use of the SET pseudo-instruction is not limited to COMPACT-compiled programs. Any names which are defined as global in either a PRE or SET pseudo-instruction are preserved in the Set Table as long as the ROAR program is not initialized. If ROAR encounters a CLS (Clear Symbol Table), the global symbols will be removed from the Symbol Table, but not from the Set Table.

Global symbols consisting of fewer than 5 characters should not be used in the same sequence of coding with a header tag, since ROAR would prefix the header tag to the symbol and, therefore, would not recognize it as a global symbol. (See Chapter 6, "Programming Techniques.")

EXAMPLE

```
*SET*FBR*COSF*XMODF*MAXIF*FDC**
```

These 5 symbols will be entered as global symbols.

ERROR PRINTOUT REFERENCE None.

RST —— Restore Symbol Table

Location - Must be blank.
Command - RST
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Must be blank.

FUNCTION When an RST pseudo-instruction is encountered, ROAR will clear from the Symbol Table all non-global symbols. This does not affect the Set Table. ROAR examines the list of global symbols established by SET and PRE pseudo-instructions to determine whether any of these symbols have been assigned absolute addresses in the object program. If they have, the symbols are re-established in the Symbol Table and their absolute addresses retained. All other symbols are cleared from the Symbol Table. In this way two different

programs that are assembled together may use the same non-global symbol to mean different things without conflict. Every COMPACT-compiled program has an RST pseudo-instruction as its first instruction.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	R,S,T			

ERROR PRINTOUT REFERENCE None.

RRS —— Relocate Regional Storage

Location - Must be blank.
Command - RRS
Data-address - The symbol designating the region followed by 5 digits indicating the number of locations used by the region.
Next-address - Must be blank.
Comments - Must be blank.

FUNCTION With the exception of a region that is common to two or more programs, sequential storage from one program must not overlap sequential storage from another. The COMPACT compiler, however, can never determine whether the program it is now compiling is to be assembled with another, and yet unique sequential storage regions must be assigned as needed. The RRS pseudo-instruction relegates this task to the assembly routine by informing ROAR how many locations have been used in the particular program under consideration. Every compiler-generated program, requiring regional storage, has an RRS pseudo-instruction at the end of the program.

When ROAR encounters an RRS pseudo-instruction, it relocates the beginning location of the specified region upward by the number of locations given in the Data-address field. Thus when the next program that uses the specified region is assembled, the regional storage will commence from that point.

The function of the RRS pseudo-command is similar to that of the RLR pseudo-command with two exceptions:

1. The RLR pseudo-instruction refers only to that area of memory reserved in reponse to the preliminary query, "Subroutine Tape Region Storage"; whereas the RRS pseudo-instruction can refer to any region used by a program regardless of how it was reserved.

2. The RLR pseudo-instruction is coded by the programmer and precedes the program that will use the specified region; the pseudo-instruction is necessary to establish the region. The RRS pseudo-instruction is generated by COMPACT (but can be coded by the programmer) and follows the program that used the specified region; the pseudo-instruction does not initially establish a region, but rather relocates the base address of a previously established region.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	RRS	[00031		
	RRS	]00238		

Region [required 31 memory locations in this program.
Region] required 238 locations.

ERROR PRINTOUT REFERENCE None.

AVL —— Make a Block Available

Location - Must be blank.
Command - AVL
Data-address - Any legal decimal address having no more than 5 digits and designating the first location to be made available.
Next-address - Any legal decimal address having no more than 5 digits and designating the last location to be made available.
Comments - Any applicable remarks.

FUNCTION This pseudo-instruction is the opposite of RES. That is, it causes ROAR to modify the Availability Table, making available for use all sectors between and including the locations indicated by the Data-address and Next-address fields. All locations within the bounds of an AVL pseudo-instruction are arbitrarily made available regardless of prior use by the program being assembled. The only time it would be meaningful to employ the pseudo-instruction AVL at the beginning of an assembly is in the event that one or both of the Double-access Tracks will not be used for double access but are to be assigned by ROAR in the same way that it assigns other memory locations. Only one head of a pair on each Double-access Track should be made available. If both heads were available, ROAR could assign two words to the same drum location because each head would then be assumed to represent 64 unique locations. (See "Double-access Track Addresses," Chapter 4.)

If the address in the Data-address field of an AVL pseudo-instruction is greater than the address in the Next-address field, ROAR will make available the location specified by the Data-address, continue to 12763, then proceed with 00000, and continue until the location specified by the Next-address is reached.

An AVL that includes locations mentioned in a previous RES, REG, or EQR will cause those locations to be restored to the Availability Table and thus will permit ROAR to use them for assignment to undefined symbolic and blank addresses. Any sectors affected by a subsequent REG, RES, or EQR will be treated in the manner usual for the REG, RES, or EQR.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	AVL	1,2,3,0,0	1,2,3,6,3	MAKE AVAILABLE THE LEADING HEAD
				OF TRACK 123/125, ie., SECTORS
				12300 THROUGH 12363.
	AVL	1,2,6,0,0	1,2,6,6,3	MAKE AVAILABLE THE TRAILING HEAD
				OF TRACK 124/126, ie., SECTORS
				12600 THROUGH 12663.
	AVL	1,3,2,0,0	6,8,3	MAKE AVAILABLE ALL SECTORS FROM
				400 THROUGH 719.
	AVL	1,0,0,0	5,0,6,3	MAKE AVAILABLE ALL SECTORS FROM
				1000 THROUGH 5063.

ERROR PRINTOUT REFERENCE Address not decimal for pseudo-op REG RES or AVL. See "Error Printouts," Chapter 8.

EQR —— Equate and Reserve Location

Location - Must be blank.
Command - EQR
Data-address - Any legal address.
Next-address - Any legal address.
Comments - Any applicable remarks.

FUNCTION This pseudo-instruction causes ROAR to make one address equivalent to some other address. It is possible to equate one symbol with another symbol or with an absolute address, but either the Data-address or the Next-address field (if not both) must contain a symbolic address. An error halt will occur if an absolute address appears in both the Data-address and Next-address fields. A blank address field and the SKIP(nn) address are not legal addresses to use with this pseudo-command.

If one address field of an EQR pseudo-instruction contains a symbolic address and the other a numeric address, the memory location thus assigned is made unavailable. ROAR would store the symbol in the Symbol Table and the absolute address in the Equivalence Table. The Equivalence Table is a block of memory where ROAR stores the absolute addresses that are assigned to symbolic addresses. (See "Equivalence Table," Appendix B.) If a numeric address is in either the Data-address or the Next-address field, the sector portion must be within the range $00 \leq N \leq 63$.

Symbolic addresses may be made equivalent even if neither has been previously assigned an absolute address. Both symbols are stored in the Symbol Table. When ROAR subsequently encounters either of the symbols, an optimum address is computed and assigned to both the symbols. Any number of symbols may be made equivalent through the use of several EQR pseudo-instructions.

If two symbolic addresses which have already been assigned absolute addresses by ROAR are to be made equivalent, the Next-address field is considered to be predominant. That is, the absolute address of the symbol written in the Next-

address field of the EQR pseudo-instruction will be assigned to the symbol in the Data-address field; the absolute address thus replaced is left unavailable. If several symbols have been made equivalent and have been assigned an absolute address, changing one of the symbols by making it equivalent to some other location does not affect the remaining symbols. For example, assume A, B, C, and D are all equivalent and have been assigned the location 02354; also, assume F has been assigned location 04623. If A is made equivalent to F, then the absolute address of A would be changed to 04623, but B, C, and D would continue to have location 02354.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	EQR	START	1000	EQUATE START WITH THE LOCATION
				1000 AND RESERVE THAT LOCATION.
	EQR	1001	NOW	EQUATE NOW WITH LOCATION
				1001 AND RESERVE IT.
	EQR	A	B	EQUATE A, B, Z, AND D.
	EQR	Z	D	
	EQR	A	Z	
	EQR	/00008	LOD	EQUATE LOD WITH THE ABSOLUTE
				ADDRESS ASSIGNED TO THE
				8TH WORD OF REGION SLASH.

ERROR PRINTOUT REFERENCE Impossible address. See "Error Printouts," Chapter 8.

EQV —— Make Equivalent

Location - Must be blank.
Order - EQV
Data-address - Any legal address.
Next-address - Any legal address.
Comments - Any applicable remarks.

FUNCTION This pseudo-instruction is handled in the same manner as EQR with one exception: when a symbol is equated with an absolute address, the memory location thus assigned is not reserved. It is assumed that when this pseudo-instruction is used the location will have been reserved previously or else that there is no need for reservation.

With an EQV pseudo-instruction it is possible to equate one symbol with another symbol or with an absolute address, but either the Data-address or the Next-address field (if not both) must contain a symbolic address. An error halt will occur if an absolute address appears in both address fields. When a numeric address is used in an EQV pseudo-instruction, the sector portion must be within the range $0 \leq N \leq 63$. A blank address field and the SKIP(nn) address are not legal addresses to use with this pseudo-command. Symbolic addresses may be made equivalent even if neither has been previously assigned an absolute address. If two symbolic addresses which have already been assigned absolute addresses by ROAR are to be equated, the symbol in the Next-address field is given precedence over the one in the Data-address field.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	EQV	ABC	9600	EITHER WILL EQUATE ABC WITH LOCATION
	EQV	9600	ABC	9600. LOCATION 9600 IS NOT RESERVED.
	EQV	105Q	/0003	EQUATE 105Q WITH THE 3RD WORD
				OF REGION SLASH.
	EQV	105Q	ABC	EQUATE 105Q WITH ABC; i.e.,
				CHANGE 105Q TO 9600.

ERROR PRINTOUT REFERENCE Impossible address. See "Error Printouts," Chapter 8.

CONTROL ASSEMBLY

NIX —— Wait a Second

Location - Must be blank.
Command - NIX
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION When NIX is encountered, ROAR copies the Comments field, removes any header tag, indicates the number of memory words used, and halts. This is useful as an end of tape code, or for any time it is desirable to have the computer halt during an assembly, e.g., at the end of each of several subroutines assembled as a unit. Following the printing of the comments, ROAR types a number, in track and sector notation, indicating the number of locations assigned since the beginning of the assembly if this is the first NIX, or since the previous NIX if there are more than one. To resume assembly, depress START COMPUTE, and ROAR will continue as if the interruption had not occurred. The internal structure of ROAR and its tables are unaltered.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	NIX			CHANGE TO TAPE B.

Typewriter Output:

```
*NIX***                                   CHANGE TO TAPE B*  103
```

The 103 indicates the number of locations used: 1 track and 3 sectors = 67 locations.

ERROR PRINTOUT REFERENCE None.

NEW —— Begin New Assembly

Location - Must be blank.
Command - NEW
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION The NEW pseudo-instruction causes ROAR to execute its initialization routines. The Availability and Symbol Tables are cleared, and ROAR is prepared to begin a new assembly. The NEW pseudo-instruction will not terminate the assembly in progress, if any. The effect of NEW is the same as if ROAR were entered from the bootstrap or by a manual transfer to GOTOROAR.

This pseudo-command is normally used shortly following an END pseudo-command. The instruction immediately following the NEW pseudo-instruction is assumed to be the first instruction of a new assembly.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	NEW			ROAR WILL INITIALIZE ITS TABLES,
				AND BEGIN A NEW ASSEMBLY.

ERROR PRINTOUT REFERENCE None.

CLS —— Clear Symbol Table

Location - Must be blank.
Command - CLS
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION The execution of this pseudo-instruction causes ROAR to remove all symbols from the Symbol Table. It does not affect the Availability Table. The CLS pseudo-instruction is used when it is desired to continue assembly with the Availability Table left intact but with a cleared Symbol Table. This situation usually arises when separate programs are being assembled to be in memory at the same time, or when major portions of a program have been coded by more than one person and duplicate symbols have been used inadvertently.

EXAMPLE

LOCATION.	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	CLS			REMOVE ALL SYMBOLS FROM THE
				SYMBOL TABLE.

ERROR PRINTOUT REFERENCE None.

END —— End Assembly

Location - Must be blank.
Command - END
Data-address - Must be blank.
Next-address - Any legal address.
Comments - Any applicable remarks.

FUNCTION When the END pseudo-instruction is detected, ROAR will read the Next-address field and will punch on the hexadecimal output tape a transfer instruction to the address given in that field. The Comments field is copied, and ROAR punches the final checksum followed by a length of blank tape. ROAR then types, in track and sector notation, the total number of locations it has assigned during the assembly. This includes all blank and symbolic addresses and any absolute address that is part of an EQV pseudo-instruction, but specifically excludes any regional reservations and any other absolute addresses used in the coding. Finally, ROAR initializes the word-count and the checksum, removes any header tag, and halts. The END pseudo-instruction does not cause ROAR to initialize its control tables, i.e., Symbol Table, Availability, etc., in preparation for a new assembly.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	END		BEGIN	PUNCH ON THE HEXADECIMAL OUTPUT
				TAPE THE INSTRUCTION TO TRANSFER
				TO THE ADDRESS EQUIVALENT TO
				THE SYMBOL BEGIN.

Typewriter Output:

```
*END**BEGIN*                          PUNCH TRANSFER CODE*  119
```

The number of locations used during the assembly is 83 (1 track and 19 sectors).

ERROR PRINTOUT REFERENCE Blank N address. See "Error Printouts," Chapter 8.

TAG —— Designate Header Tag

Location - Must be blank.
Command - TAG
Data-address - The symbol to be used for the tag.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION When a large program is being written in sections or by several people or when many subroutines are to be assembled with an object program as a unit, it is virtually impossible to avoid duplication of symbolic addresses.

To alleviate this situation, header tags were designed. After the header tag has been established, it is prefixed to any symbolic address having fewer than 5 characters. Symbols having 5 characters are not tagged, in order that they may be used as linkage symbols between sections of a program and between subroutines and a program. These 5-character symbols are similar to the global symbols established by SET. However, certain precautions must be taken when using global symbols and header tags in the same program. These precautions are discussed in Chapter 6, "PROGRAMMING TECHNIQUES."

To establish a header tag, the pseudo-instruction TAG is used with the Data-address field containing the single character symbol that is to be the tag. The Location and Next-address fields are blank. Once established, the tag will be used until it is replaced, removed, or until initialization for a new assembly is effected.

When the TAG pseudo-instruction is encountered by ROAR, the symbol is stored. Thereafter, all symbolic addresses (having fewer than 5 characters) are assigned the additional symbol. This tag will not appear on the decimal listing. An entry is made in the Symbol Table so that ROAR will always differentiate between the tagged symbol and any similar symbol. (See "Symbol Table," Appendix B.) To remove a header tag during an assembly, use the pseudo-instruction TAG with a blank Data-address field. (The NIX and END pseudo-instructions also remove header tags.)

A header tag established by TAG remains in effect until a subsequent TAG or HED assigns a different header tag. If an assembly is made from a Subroutine Library Tape, ROAR will enter what is called the "Subroutine Library Mode." Then an SBT will terminate assignment of header tags established by TAG or HED, and SUB will initiate the assignment of a different header tag. However, when ROAR exits from the "Subroutine Library Mode," i.e., returns to Normal Mode of operation, the header tag in force before SBT was encountered will be reinstated. (See SBT and SUB.)

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	T,A,G	M		SET A HEADER TAG M.
	R,A,U	K,O,N	N,-,A,D	THESE SYMBOLIC ADDRESSES WOULD
				BE STORED AS (M)KON AND (M)N-AD.
	S,A,U	M,N,-,A,D	K,O,N	THESE ADDRESSES WOULD BE STORED
				AS MN-AD AND (M)KON.
	T,A,G	A		TAG NOW BECOMES A.
	T,A,G			REMOVE TAG.

NOTE: (M)N-AD represents the symbolic address N-AD preceded by the tag M; MN-AD is simply a 5-character, symbolic address.

ERROR PRINTOUT REFERENCE None.

HED —— Assign Sequential Header Tag

Location - Must be blank.
Command - HED
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION This pseudo-command has the same effect as TAG with one exception: the programmer has no control over the symbol assigned as the header tag. To establish a header tag with HED, place the pseudo-command in the Command field and leave all other fields blank. ROAR will assign the next available header tag to all subsequent symbolic addresses having fewer than 5 characters. As explained in the description of TAG, 5-character symbols are used for links between different parts of programs and are similar to the global symbols established by SET. Certain precautions must be observed when using such global symbols and header tags in the same program. (See Chapter 6, "PROGRAMMING TECHNIQUES.") Once established, the tag will be used until it is removed or replaced or until initialization for a new assembly is effected. A maximum of 63 header tags can be assigned by HED during a single assembly.

An advantage of TAG relative to HED is that the condition of being under the influence of a particular tag can be restored at a later time, e.g., for program correction during assembly, etc. An advantage of HED is that it may be used at any time during the coding of a program with the assurance that conflict will not develop later through inadvertent use of the same header symbol.

A header tag established by HED is in effect until a subsequent HED or TAG assigns a different header tag. To remove a header tag during an assembly, use the pseudo-instruction TAG with a blank Data-address field. (The NIX and END pseudo-instructions also remove header tags.) If an assembly is made from a Subroutine Library Tape, ROAR will enter what is called the "Subroutine Library Mode." Then an SBT will terminate assignment of header tags established by TAG or HED, and SUB will initiate the assignment of a different header tag. However, when ROAR exits from Subroutine Library Mode, i.e., returns to Normal Mode of operation, the header tag in force before SBT was encountered will be reinstated. (See SBT and SUB.)

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	HED			ESTABLISH A HEADER TAG.
	SBL	ONE	N-AD	STORED AS (TAG) ONE AND (TAG)N-AD.
	TAG	B		HEADER TAG NOW BECOMES "B".
	HED			ASSIGN NEXT SEQUENTIAL HEADER
				TAG.
	TAG			REMOVE HEADER TAG.

ERROR PRINTOUT REFERENCE None.

NXT —— Optimize Next Instruction as Indicated

Location - Must be blank.
Command - NXT
Data-address - A special 6-character address, a blank address, or a number which will change the latency of the following Data-address field.
Next-address - A special 6-character address, a blank address, or a number which will change the latency of the following Next-address field.
Comments - Any applicable remarks.

FUNCTION At times it is advantageous to be able to control the optimization of instructions. This can be accomplished with the NXT pseudo-command. There is a special 6-character address that is used with the NXT pseudo-command: DELAYD. The DELAYD address appears in the Data-address or Next-address field (or both) of the pseudo-instruction. This will cause ROAR to change the normal optimization of the corresponding address field of the instruction that immediately follows the NXT pseudo-instruction. That field will be assigned the preceding Next-address sector plus 4. For a detailed description of this special address see Chapter 4.

A number may appear in the Data-address field, the Next-address field, or both fields of the NXT pseudo-instruction. That number is added to the optimum sector which ROAR computes for the absolute address of the corresponding field in the instruction following the NXT pseudo-instruction. Thus the latency of an instruction can be incremented by any desired amount from zero through 63, or since addressing is modulo-64, the latency of an instruction can be decremented: 1 word time by using the address 63, 2 word times by using 62, etc. Obviously incrementing or decrementing addresses is possible only if the instruction following the NXT pseudo-instruction contains a blank address field or symbolic addresses which have not been assigned locations. If the symbol in the corresponding address field has been assigned an absolute location, that portion of the NXT pseudo-instruction will have no effect.

The NXT pseudo-instruction must immediately precede the instruction whose optimization is to be changed. This rule has one exception: an NST pseudo-instruction may be followed by other NXT pseudo-instructions, and they will all be effective. If 2 or more NXT pseudo-instructions appear in sequence, they are accumulative. For example, assume information to be assembled is entered directly from the typewriter keyboard and one item is an NXT pseudo-instruction to provide a latency of 8 in the Next-address field. If the operator were to inadvertently type a "7" instead of the desired "8", the error could be easily corrected by entering another NXT pseudo-instruction with a "1" in the same field. The values are added. An NXT pseudo-instruction with a DELAYD address may be followed by an NXT pseudo-instruction with a numeric address, and both will be effective. That is, the address field of the next instruction will be assembled as the previous Next-address sector plus 4 plus the value indicated by the second NXT pseudo-instruction.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
RTD	RAL		INP	
	NXT	DELAYD		OPTIMIZE +Z AS INP PLUS 4.
	STU	+Z	COMP	
	NXT		8	INCREASE LATENCY OF LOOP BY 8;
	SLC	0	LOOP	ASSUME MAXIMUM NUMBER OF SHIFTS
				WILL BE 8.
	NXT	61		DECREMENT LATENCY OF HOLD BY
	RAU	HOLD	REPT	3 WORD TIMES.
	RAU	/00008	OUT	OPTIMIZE YES AS BCK+11 (2 WORD
	NXT	5	2	TIMES NORMALLY ALLOWED PLUS 9
	NXT	4		INDICATED BY NXT); OPTIMIZE NO
BCK	CLL	YES	NO	AS YES+4 (2 WORD TIMES NORMALLY
				ALLOWED PLUS 2 INDICATED BY NXT)
	RAL	EXIT	SQRT	
	NXT	DELAYD	1	OPTIMIZE SQX AS SQRT PLUS 4
	NXT	5		PLUS 5; OPTIMIZE COMP AS SQX
EXIT	STU	SQX	COMP	PLUS 3.

ERROR PRINTOUT REFERENCE None.

ESTABLISH CONSTANTS

HEX —— Establish a Constant from Hexadecimal Input

Location - Any legal address.
Command - HEX
Data-address - From 1 to 4 hexadecimal characters.
Next-address - From 1 to 4 hexadecimal characters.
Comments - Any applicable remarks.

FUNCTION A constant may be converted to its hexadecimal equivalent and entered with the HEX pseudo-instruction. The Location field may contain any legal address. The left four characters of the hexadecimal value are written in the four, low-order positions of the Data-address field. The right four characters are written in the low-order portions of the Next-address field. Leading zeros are unnecessary. A blank address is assumed to be zero.

Since the HEX pseudo-instruction may have a blank Location field, the instruction preceding it does not have to be a closed instruction (i.e., both the Data-address and the Next-address fields filled). However, if a HEX pseudo-instruction does have a blank Location field, it cannot be referred to from any other instruction. The instruction following a HEX pseudo-instruction must have a filled Location field.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
/0010	HEX	7735	9400	10^9 AT Q OF 30
MASK	HEX	F	C000	MASK FOR COMPARISON IN DATA-SECTOR
KON	HEX		1	1 AT Q OF 31

ERROR PRINTOUT REFERENCE None.

DEC —— Establish a Constant from Decimal Input

Location - Any legal address.
Command - DEC
Data-address - A value and its sign, if negative, indicating the "q" at which the number is to be held.
Next-address - The decimal number to be entered, including the decimal point and sign if applicable.
Comments - Any applicable remarks.

FUNCTION By using the DEC pseudo-instruction the programmer can write his constants in decimal notation, and ROAR will convert the number to the equivalent hexadecimal value at the indicated "q". The Location, symbolic or absolute, is where the value is to be stored. The Data-address field contains the q at which the number is to be held. The sign, necessary only if the q is negative, precedes the q. The Next-address field contains the decimal number, including its sign and decimal point if applicable. The sign, necessary only when the value is negative, must precede the number; the decimal point is included at its proper place.

The decimal number to be entered may consist of more than 16 characters, but only the least significant 16 will be considered. It may contain a maximum of 13 digits following the decimal point. When writing the program, start the decimal number in the Next-address column and allow it to extend into the Comments column. When punching the tape, place the stop code for the Next-address field after the complete number.

Since the DEC pseudo-instruction may have a blank Location field, the instruction preceding it does not have to be a closed one. However, reference cannot be made to any instruction or pseudo-instruction that has a blank Location field. The instruction following a DEC pseudo-instruction must have a filled Location field.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
KON	DEC	21	-18563	.8543* ENTER -18563.8543 AT A Q OF 21 IN LOCATION KON.
NUM	DEC	-3	.062583	4* ENTER .0625834 AT A Q OF -3 IN LOCATION NUM.
INT	DEC	17	1847	ENTER THE NUMBER 1847 AT A Q OF 17 IN LOCATION INT.

ERROR PRINTOUT REFERENCE Incorrect entry for DEC. See "Error Printouts," Chapter 8.

ALF —— Input Alphanumeric Characters

Location - Any legal address.
Command - ALF
Data-address - Alphanumeric characters.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION The alphanumeric characters in the Data-address field of the ALF pseudo-instruction are read in Six-Bit Mode and are assigned the Location indicated by the Location field. The Next-address field must be blank. Any number of characters may be in the Data-address field, but only 5 complete characters can be stored in any given location. If 5 characters are entered, they will occupy bit positions 2 through 31 and bit positions 0 and 1 will contain zeros. If fewer than 5 characters are entered, they are assembled in the low-order portion of the memory location with zeros in the high order portion. If more than 5 characters are entered, the last 5 characters and the final 2 bits of the preceding character will be assembled. Any alphanumeric characters, codes 16 through 62, may be used. (See Appendix E, "ALPHANUMERIC CODES.")

Since an ALF pseudo-instruction may have a blank Location field, it does not have to be preceded by a closed instruction. However, reference cannot be made to any instruction or pseudo-instruction that has a blank Location field. The instruction following an ALF pseudo-instruction must have a filled Location field.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
608	ALF	I=PRT		ENTER THE FIVE ALPHANUMERIC CHARACTERS I, =, P, R, T IN LOCATION 608.
PNT	ALF	X=2Z/Y		ENTER THE BIT CONFIGURATION (01) AND THE FIVE CHARACTERS =, 2, Z, /, Y.

ERROR PRINTOUT REFERENCE None.

PAV —— Punch Availability Table

Location - Must be blank.
Command - PAV
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION Should it be desirable to know the status of the Availability Table, the programmer can have its contents punched on tape by using the PAV pseudo-instruction. The PAV pseudo-instruction is usually input to ROAR directly from the typewriter keyboard after the assembly is completed. It should be entered after the END pseudo-instruction, in order that the tape thus obtained will be separate from the program tape. This pseudo-instruction causes ROAR to turn off and bypass the Copy Mode, and to bypass the typewriter output. The only output from ROAR resulting from the pseudo-instruction is a hexadecimal tape of the Availability Table.

The tape consists of the following:

1. An area of blank tape.
2. An RAV pseudo-instruction.
3. The Availability Table, consisting of a hexadecimal representation of the contents of the stored table interspersed with initial adresses in decimal notation. Each hexadecimal word represents a particular sector on 32 consecutive tracks; the decimal address denotes the sector concerned together with the first of the 32 tracks. (See "Availability Table," Appendix B.)
4. A checksum.

NOTE: ROAR will read *PAV* and immediately begin punching the tape. After completing the tape, ROAR will read the remaining 3 fields.

In this discussion of the PAV pseudo-instruction the output unit is assumed to be the RPC-4500 Punch. However, if other units are used for output during an assembly, the output from this pseudo-instruction will be on the same unit as the hexadecimal program. (See "Input-Output Selections," Chapter 8.)

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	PAV			PUNCH THE AVAILABILITY TABLE.

ERROR PRINTOUT REFERENCES None.

PPA —— Punch and Print Availability Table

Location - Must be blank.
Command - PPA
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION This pseudo-command is handled in the same manner as PAV with one exception: in addition to the hexadecimal tape which is punched, a typed, decimal listing of available sectors is provided. The hexadecimal tape is exactly the same as the one produced following a PAV. The typed listing will have 15 items per line, consisting of a two-digit sector number followed by three-digit numbers indicating the tracks on which that sector is available. If a specific sector is not available on any track, that sector is not listed.

The first item punched on the hexadecimal tape resulting from the PPA is the pseudo-instruction RAV (Read Availability Table), which is necessary in order to read the tape into the computer. The last item punched on the tape is a checksum. The RAV pseudo-instruction and the checksum do not appear on the typed listing.

NOTE: ROAR will read *PPA* and immediately begin to punch and print. After the output is completed, ROAR will read the remaining 3 fields.

In this discussion of the PPA pseudo-instruction the output units are assumed to be the RPC-4500 Punch and Typewriter. However, if other units are selected for output during an assembly, the output from this pseudo-instruction will be produced as follows: the hexadecimal output will be on the same unit as the hexadecimal program tape; the listing of sectors will be on the same unit as the decimal listing of the program. (See "Input-Output Selections," Chapter 8.)

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	PPA			PUNCH AND PRINT AVAILABILITY TABLE.

```
20   097 098 120 121 122
42   075 076 077 095 096 097 098 120 121 122
58   075 076 077 095 096 097 098 120 121 122
63   000 001 002 065 066 067 074 075 076 077 095 096 097 098
     120 121 122
```

ERROR PRINTOUT REFERENCE None.

RAV —— Read Availability Table from Tape

Location - Must be blank.
Command - RAV
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION The RAV pseudo-instruction instructs ROAR to read the hexadecimal tape of the Availability Table. This pseudo-instruction is the first item punched on the availability tape obtained by executing a PAV or PPA. When an availability tape is to be loaded, it is usually the first tape read after loading the assembly program or after using the pseudo-command NEW. After ROAR has read the availability tape, it will halt. Depressing START COMPUTE will enable ROAR to continue the assembly.

EXAMPLE

Typewriter output:

(C. R.)*RAV*(C. R.)(C. R.)

The typewriter executes a carriage return, types *RAV*, executes two more carriage returns, then is de-selected by ROAR while the tape is being read.

ERROR PRINTOUT REFERENCE Checksum wrong. See "Error Printouts," Chapter 8.

PST —— Print Symbol Table

Location - Must be blank.
Command - PST
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION The PST pseudo-instruction causes ROAR to print every symbol from the Symbol Table along with the location assigned to it. This pseudo-instruction is usually input to ROAR directly from the typewriter keyboard after the assembly is completed. In order that the list thus obtained will be separate from the decimal listing of the assembled program, the PST should be entered after the END pseudo-instruction. Three classes of symbols are possible:

1. Five character symbols.

2. Untagged symbols and those headed by the TAG pseudo-instruction are characterized by the tag (a space if untagged), a dash, and the symbol.

3. Symbols tagged by the HED pseudo-instruction and those read from the Subroutine Library tape are characterized by an asterisk (stop code), a space, and the symbol.

The symbols are printed in the same order in which they are stored in the Symbol Table.

In this discussion of the PST pseudo-instruction the output unit is assumed to be the RPC-4500 Typewriter. However, if other units are selected for output during an assembly, the output resulting from this pseudo-instruction will be on the same unit as the decimal listing of the program. (See "Input-Output Selections," Chapter 8.)

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	P,S,T			PRINT SYMBOL TABLE

Typewriter Output:

```
*PST***PRINT SYMBOL TABLE*
R-RATE  00006  R-  GO  00008  *    R  00120   - I,J  00038  * THIS  00018
*  WHY  00020   -EVER  00004  R-EVER  00104  * THIS  00022  *    A  00140
R-TIME  00010  ,-  OK  00032  * THIS  00042  *  WHY  00044  ,-  NO  00030
,- U+V  00036   START  00000  ,-THIS  00026  ,- WHY  00128  ,-EVER  00124
```

ERROR PRINTOUT REFERENCE None.

5CS —— Punch Five-Character Symbols

Location - Must be blank.
Command - 5CS
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION The 5CS pseudo-instruction will cause ROAR to search the Symbol Table for all 5 character symbols and punch them on tape in the form of EQR pseudo-instructions. The output for this pseudo-instruction and PAV or PPA makes partial re-assemblies quite simple. As with the PAV and PPA pseudo-instructions, 5CS should be input directly from the typewriter keyboard after the assembly is completed (i.e., following the END pseudo-instruction).

When the 5CS pseudo-instruction is executed, ROAR will turn off and bypass the Copy Mode and the typewriter output. The only output resulting from this pseudo-instruction is a punched tape. However, a typewritten copy may be obtained by manually selecting COMPUTER TO TYPEWRITER while the tape feeds are being punched. The last item punched on the tape is a NIX pseudo-instruction.

NOTE: ROAR will read *5CS* and immediately begin punching the tape. After completing the tape, ROAR will read the remaining 3 fields.

In this discussion of the 5CS pseudo-instruction the output unit is assumed to be the RPC-4500 Punch. However, if other units are selected for output during an assembly, the output resulting from this pseudo-instruction will be on the same unit as the hexadecimal program. (See "Input-Output Selections," Chapter 8.)

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	5,C,S			PUNCH 5 CHARACTER SYMBOLS

If listed on the typewriter, the tape could produce:

```
*EQR*RREAD*05929**
*EQR*[NEXL*05947**
*EQR*PRINT*07126**
*NIX****
```

ERROR PRINTOUT REFERENCE None.

PAS —— Punch All Symbols

Location - Must be blank.
Command - PAS
Data-address - Must be blank.
Next-address - Must be blank.
Comments - Any applicable remarks.

FUNCTION This pseudo-instruction is handled in the same manner as 5CS with one exception: all the symbols in the Symbol Table are punched.

By using an RST pseudo-instruction followed by a PAS pseudo-instruction, the programmer can have ROAR punch all the global symbols used in his program. Since the punched symbols will be in the form of EQR pseudo-instructions, the tape can be used to establish the absolute addresses of the global symbols for other assemblies.

NOTE: ROAR will read *PAS* and immediately begin to punch the tape. After completing the tape, ROAR will read the remaining 3 fields.

In this discussion of the PAS pseudo-instruction the output unit is assumed to be the RPC-4500 Punch. However, if other units are selected for output during an assembly, the output resulting from this pseudo-instruction will be on the same unit as the hexadecimal program.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	PAS			

If a typewritten copy were made, it could appear as

```
*EQR*F0A*12208**
*EQR*XMALF*12240**
*EQR*FL02*12233**
*EQR*FIX2*12237**
*NIX****
```

ERROR PRINTOUT REFERENCE None.

PRC —— Print a Character

Location - Any legal address.
Command - PRC
Data-address - A single alphanumeric character or a special 2-character typewriter control.
Next-address - Any legal address.
Comments - Any applicable remarks.

INPUT OR OUTPUT

FUNCTION The output resulting from this pseudo-instruction is the same as the output from a PRD instruction. The assembled instruction will contain the Alphanumeric Code (see Appendix E) in the Data-track portion corresponding to the character or typewriter control specified. The Data-sector will be the same as the sector portion of the Location field. The typewriter control codes used with a PRC pseudo-command are

TF = Tape Feed	LC = Lower Case
CR = Carriage Return	LF = Line Feed
TB = Tab	SC = Stop Code
BS = Backspace	SP = Space
UC = Upper Case	CD = Code Delete

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
PT2	PRC	CR		CARRIAGE RETURN
	PRC	UC		UPPER CASE
	PRC	I	COMP	PRINT I

Typewriter Output:

```
*PRC*CR**      01343  16  00143  01545  CARRIAGE RETURN*
*PRC*UC**      01545  16  00545  01847  UPPER CASE*
*PRC*I**       01847  16  03447  01949  PRINT I*
```

ERROR PRINTOUT REFERENCE Impossible Address. See "Error Printouts," Chapter 8.

SHIFT

SRT —— Shift Right

Location - Any legal address.
Command - SRT
Data-address - Number of shifts in the sector portion.
Next-address - Any legal address.
Comments - Any applicable remarks.

FUNCTION The output resulting from this pseudo-instruction is the same as the output from an SRL instruction with zero in the Data-track. The Data-sector of the SRT pseudo-instruction must contain a number from 0 through 63 to indicate the number of shifts. (A blank Data-address will result in an error halt; however, a blank address is legal in the Next-address field.) The assembled instruction will contain 000 in the Data-track portion and the indicated number of shifts in the Data-sector portion. The number of shifts may be modified by indexing.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
NOW	SRT	6	N-AD	SHIFT RIGHT 6 PLACES.
BAK	XSRT	1	INIT	INDEXED SHIFT RIGHT.

Typewriter Output:

```
NOW*SRT*6*N-AD*   07236  12  00006  07549  SHIFT RIGHT 6 PLACES*
BAK*XSRT*1*INIT*  06915  X12 00001  07023  INDEXED SHIFT RIGHT*
```

ERROR PRINTOUT REFERENCE Impossible address. See "Error Printouts," Chapter 8.

SLT —— Shift Left

Location - Any legal address.
Command - SLT
Data-address - Number of shifts in the sector portion.
Next-address - Any legel address.
Comments - Any applicable remarks.

FUNCTION The output resulting from this pseudo-instruction is the same as the output from an SRL instruction with a 1 in the Data-track. The Data-sector of the SLT pseudo-instruction must contain a number from 0 through 63 to indicate the number of shifts. (A blank Data-address will result in an error halt; however, a blank address is legal in the Next-address field.) The assembled instruction will contain 001 in the Data-track portion and the indicated number of shifts in the Data-sector portion. The number of shifts may be modified by indexing.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
CNT	SLT	4	N-AD	SHIFT LEFT 4 PLACES.
THS	XSLT	1	RND	INDEXED SHIFT LEFT.

Typewriter Output:

```
CNT*SLT*4*N-AD*   06237  12  00104  06948  SHIFT LEFT 4 PLACES*
THS*XSLT*1*RND*   07112  X12 00101  07320  INDEXED SHIFT LEFT*
```

ERROR PRINTOUT REFERENCE Impossible address. See "Error Printouts," Chapter 8.

COM —— Comment

Location - Must be blank.
Command - COM
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION When a COM pseudo-instruction is encountered, ROAR selects Copy Mode and copies from the input tape until a stop code is encountered. In this way a complete description of a program or subroutine can be given at the beginning of the assembly, limiting the necessity for any subsequent comments.

EXAMPLE

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	COM	THE	FOLLO	WING PROGRAM WILL INPUT DATA
				IN 4-BIT, CONVERT TO BINARY, SCALE,
				AND HOLD RESULTS IN LOWER AT 0=31

ERROR PRINTOUT REFERENCE None.

(56) (57) (56) —— Compact-Generated Comments

Location - Must be blank.
Command - (56) (57) (56)
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION This "comment" pseudo-instruction consists of the three characters (56) (57) (56) and instructs ROAR to ignore all characters following, including stop codes (*) unless the stop code is preceded by the character (56). In this way the original statement in the Compact-language program can be included in the ROAR symbolic output as a comment, to make the final program listing somewhat comprehensible.

Because codes 56 and 57 have no representation on the keyboard, this comment pseudo-instruction cannot be typed. However, it can be punched on tape (by the computer directly or by overpunching with the typewriter) and will enter the computer the same as any other characters. When the program containing the "comment" pseudo-instruction is assembled by ROAR, the typewriter output will show only two stop codes, a carriage return and the remarks provided, and finally a carriage return and the final stop code.

COMMENT

EXAMPLE

On tape is punched *(56)(57)(56)* Subscript and do loop test (56)*
Typed output will show:

```
**
SUBSCRIPT AND DO LOOP TEST
*
```

ERROR PRINTOUT REFERENCE None.

USE SUBROUTINE LIBRARY

SBT —— Enter Subroutine Library Mode

Location - Must be blank.
Command - SBT
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION When subroutines are to be assembled from a Library Tape, the first input from the tape is the pseudo-instruction SBT. (See Appendix A, "SUBROUTINE LIBRARY TAPE.") The SBT instructs ROAR to enter what is called the "Subroutine Library Mode" of operation. In this mode, subroutines from a Library Tape can be assembled faster than if the normal ROAR mode were used.

The Location field of this pseudo-instruction must be blank. The Data-address, Next-address, and Comment fields are non-existent for this pseudo-instruction. That is, following the Command field, ROAR will expect to read the Location field of the next instruction.

When the SBT pseudo-command is executed, ROAR will store the header tag in use, if any; establish an SBT tag; turn off and bypass Copy Mode; and bypass the typewriter output. The only output from ROAR during this mode of operation will be the hexadecimal program tape and any error printouts that might occur.

The Subroutine Library Mode will be terminated only when the pseudo-command SBE is encountered.

EXAMPLE

On a Library Tape this pseudo-instruction would be punched:

SBT

ERROR PRINTOUT REFERENCE None.

SUB —— Read Subroutine from Library Tape

Location - Must be blank.
Command - SUB
Data-address - A value indicating the number of entry points in the subroutine.
Next-address - All entry points to the subroutine.
Comments - Inapplicable.

FUNCTION The SUB pseudo-instruction must precede each subroutine on the Library Tape. The Location field must be blank. The Data-address contains a value indicating the number of entry points in the subroutine that immediately follows this pseudo-instruction. The Next-address field contains all the entry points to the subroutine; each one is followed by a stop code. If written on a coding sheet, the entry points may extend into the Comments column, since no remarks may be punched on the tape as part of a SUB pseudo-instruction. ROAR will read and store the entry points. They are then compared with the symbols in the Symbol Table to determine if any of the entry points has been called upon. If an entry point is found in the Symbol Table, the subroutine is assembled; if not, it is bypassed and the next SUB pseudo-instruction is read.

A SUB pseudo-instruction will immediately follow the SBT pseudo-instruction. The SUB pseudo-instruction is usually followed by an RLR pseudo-instruction, if applicable.

EXAMPLES

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	SUB	1	]0004	
	SUB	4	]0005*	]1005*]2005*]3005*

The first pseudo-instruction indicates that a subroutine has only 1 entry point which is]0004. The second pseudo-instruction indicates that a subroutine has 4 entry points which are]0005,]1005,]2005,]3005.

ERROR PRINTOUT REFERENCE Insufficient Subroutine Region Storage.
See "Error Printouts," Chapter 8.

SBE —— Exit from Subroutine Library Mode

Location - Must be blank.
Command - SBE
Data-address - Inapplicable.
Next-address - Inapplicable.
Comments - Inapplicable.

FUNCTION After the Library Tape has been read and all desired subroutines assembled, the pseudo-instruction SBE is used to exit from the Subroutine Li-

brary Mode. The SBE instructs ROAR to restore to its original state anything that was modified because of the SBT pseudo-instruction. That is, ROAR reinstates any header tag that was being used prior to the SBT, and re-establishes the Copy Mode and typewriter output. Following this pseudo-instruction ROAR is prepared to continue the assembly in Normal Mode.

The Subroutine Library Mode established by the SBT pseudo-instruction will be terminated only when the SBE pseudo-instruction is encountered. The SBE pseudo-instruction is the next to last instruction punched on the Subroutine Library Tape; it is followed by a NIX pseudo-instruction.

EXAMPLE

The last two instructions on the Subroutine Library Tape appear in this format:

```
*SBE*
*NIX****
```

ERROR PRINTOUT REFERENCE None.

PROGRAMMING TECHNIQUES 6

This chapter contains various programming techniques and general information about using ROAR. It is intended as instructive material for beginning programmers and reference material for others.

PROGRAM LIBRARY

The Commercial Computer Division of General Precision, Inc. maintains a library of general programs which are available to RPC-4000 users upon request. A few of the many different kinds of programs are listed here:

Classification	Example of Program
Programmed arithmetic	floating-point systems
Elementary functions	trigonometric functions
Executive routines	compiler, assembler, interpreter
Input-Output routines	program input, alphanumeric and data input-output routines
Program Test and Correction routines	trace routines, program checkout routines, memory print routines
Utility and Conversion programs	sorting, information transfer, conversion routines

TABLE 6.1 REPRESENTATIVE PROGRAMS AVAILABLE

Each program tape in the library is supplemented by a program description which explains what the program does, what information is necessary, how to provide the required information, and how to operate the program.

DATA INPUT-OUTPUT

Most programs during their execution require the input of data and result in an output of data. For these purposes standard subroutines are often used. Occasionally a programmer prefers to write his own input and output routines. The following examples illustrate the principles involved in binarization during input and decimalization during output.

In the main or source program a calling sequence is necessary each time a subroutine is to be used. A calling sequence provides any information required

by that subroutine in order to function, for example the q of a value or the exit instruction (the instruction in the source program to which control will be transferred when the subroutine has completed its operation).

Illustrated below is a program which will read a 9-digit integer and binarize it, leaving the value in the Upper Accumulator at a q of 31. The program is written as a subroutine, so that it may be used to input many different integers. The calling sequence must place the exit instruction in the Lower Accumulator before transferring to the entry location (INPUT) of the subroutine. Assume the integer 123456789 is to be input.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
INPUT	CLL	EXIT		STORE EXIT INSTRUCTION IN "EXIT"
	EXC	298		L→U (∴ ZERO→U)
	INP	98		4 BIT INPUT
	MPT	98		10×1
	CLU	RECRC7		STORE 10×1 IN RECRC 7
	SLT	4		SHIFT LEFT 4 (2→UPPER)
	ADU	RECRC7		(10×1)+2
	MPT	98		10[10×1)+2]=100×1+10×2
	CLU	RECRC7		STORE IN RECRC 7
	SLT	4		SHIFT LEFT 4 (3→UPPER)
	ADU	RECRC7		(100×1)+(10×2)+3
	MPT	98		10[(100×1)+(10×2)+3]=10^3×1
				$+10^2$×2+10×3
	CLU	RECRC7		STORE IN RECRC 7
	SLT	4		SHIFT LEFT 4 (4→UPPER)
	ADU	RECRC7		10^3×1+10^2×2+10×3+4
	MPT	98		10^4×1+10^3×2+10^2×3+10×4
	CLU	RECRC7		
	SLT	4		SHIFT LEFT 4 (5→UPPER)
	ADU	RECRC7		10^4×1+10^3×2+10^2×3+10×4+5
	MPT	98		10^5×1+10^4×2+10^3×3+10^2×4+10×5
	CLU	RECRC7		
	SLT	4		(6→UPPER)
	ADU	RECRC7		
	MPT	98		10^6×1+10^5×2+10^4×3+10^3×4+10^2×5+10×6
	CLU	RECRC7		
	SLT	4		(7→UPPER)
	ADU	RECRC7		
	MPT	98		10^7×1+10^6×2+10^5×3+10^4×4+10^3×5
				$+10^2$×6+10×7
	CLU	RECRC7		
	SLT	4		(8→UPPER)
	ADU	RECRC7		
	MPT	98		10^8×1+10^7×2+10^6×3+10^5×4+10^4×5
				$+10^3$×6+10^2×7+10×8
	CLU	RECRC7		
	SLT	4		(9→UPPER)
	ADU	RECRC7	EXIT	10^8×1+10^7×2+10^6×3+10^5×4+
				10^4×5+10^3×6+10^2×7+10×8+9
	NIX			

FIGURE 6.1 INPUT ROUTINE

The following sequence of instructions will decimalize a binary value and output that integer on the selected device. The value to be output must be in the Upper Accumulator at a q of 31, and the exit instruction in the Lower Accumulator when control is transferred to the subroutine. Preceding the number, this routine will output a sign for negative values or a space for positive ones. The output of leading zeros will be suppressed.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
9DEC0	CLL	EXIT		STORE EXIT INSTRUCTION IN "EXIT"
	LDX	0		SET INDEX REGISTER TO ZERO
	TMI		P0S	TEST FOR NEGATIVE NUMBER
	CLU	REGRC4		IF NEGATIVE, FIND ABSOLUTE
	SBU	REGRC4		VALUE.
	RAL	-SIN	J0IN	SET UP PRINT INSTRUCTION FOR "-"
P0S	RAL	+SIN	J0IN	SET UP PRINT INSTRUCTION FOR SPACE
J0IN	CLL	REGRC4		STORE PRINT (- OR +) IN RECRC4
	RAL	RND		RND = 80000000
	SBU	10T9		10T9 = 10^9 @ 31
	TMI		ERR0R	IF MORE THAN 9 DIGITS, → ERROR HALT
	ADU	TEN9		TEN9 = 10^9 @ 31
	DIV	10T8		10T8 = 10^8 @ 28
	RAL	MASK	COMP	MASK = F0000000 } SET UP TO
COMP	CME	ZERO		ZERO = 00000000 } BYPASS LEAD-
	TBC	0N	REGRC4	ING ZEROS.
REGRC4	0	0	SELC	ASSIGN SELC OPTIMUM TO RECRC 4.
0N	PRD	6199		PRINT SPACE IN LIEU OF ZERO
	MPT	98		MULTIPLY BY 10 TO PRODUCE NEXT DIGIT.
	XLDX	1		INCREMENT INDEX REGISTER
	CXE	9		TEST FOR 9TH DIGIT
	TBC	EXIT	COMP	FINISHED; OR LOOP FOR MORE DIGITS
SELC	PRU	1699		PRINT BITS 0-3 AS A DIGIT
	XLDX	1		INCREMENT INDEX
	EXT	DR0P		EXTRACT OFF LAST DIGIT PRINTED
	MPT	98		MULTIPLY BY 10 FOR NEXT DIGIT
	CXE	9		TEST FOR 9TH DIGIT.
	TBC	EXIT	SELC	FINISHED; OR LOOP FOR MORE DIGITS
ERR0R	PRD	2299		TYPEWRITER TAB
	HLT	0	EXIT	HALT; ON START SIGNAL TRANSFER TO EXIT.
-SIN	PRD	5999	SELC	PRINT -
+SIN	PRD	6199	SELC	PRINT SPACE FOR PLUS
RND	HEX	80000	0	ROUNDING CONSTANT FOR DIVISION
10T9	HEX	3B9A	CA00	10^9 AT Q OF 31
TEN9	HEX	3B9A	CA00	10^9 AT Q OF 31
10T8	DEC	28	10000	0000* 10^8 AT Q OF 28
ZERO	HEX	0	0	ZERO
DR0P	HEX	FFF	FFFF	MASK FOR EXTRACTING BITS 0-3
MASK	HEX	F0000	0	MASK FOR COMPARING WITH ZERO
	NIX			

FIGURE 6.2 DECIMAL OUTPUT ROUTINE

If a hexadecimal value is to be output rather than a decimal value, the following sequence might be used.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	COM	PRINT	CONTEN	TS OF UPPER IN HEXADECIMAL*
P8	PRU	1699		PRINT FIRST OF EIGHT CHARACTERS
	MPT	298		SHIFT UPPER LEFT 1 BIT
	MPT	198	P7	SHIFT UPPER LEFT 3 BITS
P7	PRU	1699		PRINT SECOND OF 8 CHARACTERS
	MPT	298		SHIFT UPPER LEFT 1 BIT
	MPT	198	P6	SHIFT UPPER LEFT 3 BITS
P6	PRU	1699		PRINT THIRD CHARACTER
	MPT	298		SHIFT UPPER LEFT 1 BIT
	MPT	198	P5	SHIFT UPPER LEFT 3 BITS
P5	PRU	1699		PRINT FOURTH CHARACTER
	MPT	298		
	MPT	198	P4	
P4	PRU	1699		PRINT FIFTH CHARACTER
	MPT	298		
	MPT	198	P3	
P3	PRU	1699		PRINT SIXTH CHARACTER
	MPT	298		
	MPT	198	P2	
P2	PRU	1699		PRINT SEVENTH CHARACTER
	MPT	298		
	MPT	198	P1	
P1	PRU	1699	P0	PRINT EIGHTH CHARACTER
RECRCO	0	0	P1	SET LOCATION COUNTER=P1
HEXPT	CLL	P0	P8	ENTRY TO SUBROUTINE; STORE EXIT IN P0
	NIX			

FIGURE 6.3 HEXADECIMAL OUTPUT ROUTINE

The example in Figure 6.3 is written as a subroutine which will output the contents of the Upper Accumulator as a hexadecimal word. It does not produce a stop code. The entry to the subroutine is the instruction

HEXPT*CLL*P0*P8**

It is coded as the last instruction in order that P0 can be optimized with respect to P1, thus providing an optimum exit instruction. That is, although the RECRC0 instruction is not punched in the program tape when it is assembled by ROAR, it sets ROAR's Location Counter to the value indicated in the Next-address field (in this case, to whatever address has been assigned to the symbol P1). Therefore, the symbol HEXPT will be assigned a sector equal to that assigned P1, and P0 will be assigned a sector value 2 greater. Then, after the instruction

P1*PRU*1699*P0**

is executed, the transfer to the exit instruction (stored in Location P0) will be optimum.

The MPT command was used to accomplish the necessary shifts because it requires less time than the comparable SRL command. Consider the two instructions following the one in P8:

```
*MPT*298***
*MPT*198*P7**
```

These will shift the contents of the Upper Accumulator to the left 4 places and will be executed in 8 word times. The instruction

```
*SRL*104***
```

would then shift the contents of the Double-length Accumulator 4 places left, but would require 11 word times.

In Figure 6.3 several locations were assigned symbols that seem unnecessary. However, with the routine coded in this manner, it can be used with another routine, such as the one below, which will delete leading zeros.

FIGURE 6.4 SUPPRESS LEADING ZEROS

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	COM	THIS	PROGRAM	DELETES LEADING ZEROS FOR
				THE PRINT HEXADECIMAL ROUTINE *
RECRCO	0	0	P1	SET SECTOR ADDRESS OF NOZEO=P1
NOERO	CLL	P0		ENTRY TO SUPPRESS ZERO ROUTINE
	TMI	P8		IF NEG→ PRINT 8 CHARACTERS
	SBU	K1		K1= (1@19)
	TMI	OTO3	4TO8	IF NEG.→0 TO 3 CHARS; POS.→4 TO 8
4TO8	SBU	K2		K2=(1@11)-(1@19)
	TMI	4ORS	6TO8	NEG→4 OR 5 CHARS; POS→6 TO 8
4ORS	ADU	K3	.	K3=(1@11)-(1@15)
	TMI	FOUR	FIVE	NEG.→4 CHARS; POS.→5 CHAR.
6TO8	SBU	K4		K4=(1@7)-(1@11)
	TMI	SIX	7ORS	NEG.→6 CHARS; POS.→7 OR 8
7ORS	SBU	K5		K5=(1@3)-(1@7)
	TMI	SEVN	ATE	NEG→7 CHARS; POS→8
OTO3	ADU	K6		K6=(1@19)-(1@27)
	TMI	OOR1	2ORS	NEG.→0 OR 1 CHAR; POS.→2 OR 3
OORI	ADU	K7		K7=(1@27)-(1@31)
	TMI	P0	ONE	NEG.→0 CHARS. ∴ EXIT; POS.→1
2ORS	SBU	K8		K8=(1@23)-(1@27)
	TMI	TWO	THRE	NEG.→2CHARS; POS.→3
ATE	ADU	C1	P8	C1=(1@3) REESTABLISHES ORIGINAL
				VALUE.
SEVN	MPT	298		SHIFT OUT 1 LEADING ZERO (4 BITS)
	MPT	198	P7	AND GO TO PRINT 7 CHARACTERS
SIX	MPT	298		
	MPT	298		SHIFT OUT 2 LEADING ZEROS (8 BITS)
	MPT	198		AND GO TO PRINT 6 CHARACTERS.
	MPT	198	P6	
FIVE	ADU	C2		C2=(1@15) REESTABLISH ORIGINAL
				VALUE
	LDC	R1		REPEAT COUNT R1=3
	MPT	198	P5	SHIFT LEFT 12 BITS; GO TO PRINT 5

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
FØUR	LDC	R2		REPEAT COUNT R2=4
	NXT	0	4	DELAY N-ADDRESS 4
	MPT	198		SHIFT LEFT 15 BITS
	MPT	298	P4	SHIFT LEFT 1 BIT; GO TO PRINT 4
THRE	ADU	C3		C3=(1@23)
	LDC	R3		REPEAT COUNT R3=5
	MPT	198	SKIP05	SHIFT LEFT 18 BITS
	MPT	298		SHIFT LEFT 2 BITS (TOTAL 20 BITS)
	MPT	298	P3	GO TO PRINT 3 CHARACTERS
TWØ	LDC	R4		REPEAT COUNT R4=7
	MPT	198	P2	SHIFT LEFT 24 BITS; GO TO PRINT 2
ØNE	ADU	C4		C4=(1@31)
	LDC	R5		REPEAT COUNT R5=8
	MPT	198	SKIP08	SHIFT LEFT 27 BITS
	MPT	298	P1	SHIFT LEFT 1 BIT; GO TO PRINT 1
K1	HEX	0	1000	
K2	HEX	F	F000	
K3	HEX	F	0	
K4	HEX	F0	0	
K5	HEX	F00	0	
K6	HEX	0	FF0	
K7	HEX	0	F	
K8	HEX	0	F0	
C1	HEX	1000	0	
C2	HEX	1	0	
C3	HEX	0	100	
C4	HEX	0	1	
R1	0	0	300	
R2	0	0	400	
R3	0	0	500	
R4	0	0	700	
R5	0	0	800	
	NIX			

This subroutine would have to be assembled after the Hexadecimal Output Routine in order for NOZRO to be optimized with respect to P1. The number of characters to be printed is determined by a series of tests. The method shown here never requires more than 4 tests. This is in contrast with a process of straight comparison. In that method, the leading 4 bits are compared with a zero. If the comparison is successful, the value is shifted left, and the comparison continues until a non-zero configuration is found, or until the last character is tested. The comparison method may result in as many as 8 tests.

Notice that it is unnecessary to store the original value. When a constant is subtracted from the original value, a successive subtraction will in effect add the first subtrahend and subtract the next one in the same operation. That is, the first constant subtracted is 1 at a q of 11; the second should be 1 at a q of 19. The quantity actually subtracted in the second operation is the difference between these two values: (1 @ 11) - (1 @ 19). For example, if the word 0006E000 is in the Upper Accumulator to be printed, the results of the successive operations would appear as

Location	Description
	original value—
N0ZR0 2	SBU K1 (1 @ 19)
4T08	SBU K2 (1 @ 11) - (1 @ 19)
40R5	ADU K3 (1 @ 11) - (1 @ 15)
FIVE	ADU C2 (1 @ 15)
	original value

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
																			1												
													1	1	0	1	1	0	1												
											0	1	1	1	1	1	1	1	1												
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	0												
											0	1	1	1	1																
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0												
															1																
												0	1	1	0	1	1	1	0												

FIGURE 6.5 BIT CONFIGURATION

SECTIONAL ASSEMBLIES

It is frequently desirable to assemble a large program in sections, checking out and, if necessary, re-assembling each section as it is written. In order to assemble the second and subsequent portions of such a program, ROAR must be set to the same conditions as existed when the previous assembly was finished. To accomplish this, certain provisions must be made when writing each section:

1. Make all linkage between sections (utility subroutines are considered sections) either 5-character symbols or global symbols established with the SET pseudo-command. (See "Header Tags Versus Global Symbols" later in this chapter.)

2. All region reservations, EQR, and (if used) SET pseudo-instructions should be on a separate tape from the rest of the program, as this information must be read by ROAR prior to the assembly of each section.

3. With each section an END pseudo-instruction should be used to provide a checksum. Continuous loading of the completed program can be accomplished by making the transfer address a location on the bootstrap track rather than the beginning of the program. If the following section is to be loaded using the same modifier, the transfer should be to sector 39 of the bootstrap track. If a different modifier is to be used, the transfer should be to RECRC4, and SENSE SWITCH 1 must be depressed before the transfer is executed. This use of the END pseudo-command makes subsequent loading of hexadecimal tapes easier, since only the first section will have a bootstrap.

   NOTE: If this transfer is input manually at the time the program is loaded, the Lower Accumulator must be cleared before the transfer instruction is entered.

Upon completion of one portion of a sectional assembly, the information required by ROAR for the assembly of later sections can be obtained as follows:

1. Remove the hexadecimal tape of the assembled program.
2. Depress MASTER RESET.
3. Select TYPEWRITER TO COMPUTER.
4. Select COMPUTER TO PUNCH.
5. Depress ONE OPERATION.

6. Depress SET INPUT.
7. Depress EXECUTE LOWER ACCUMULATOR.
8. Depress START COMPUTE.
9. Type: NOWBEGIN*
10. Raise EXECUTE LOWER ACCUMULATOR.
11. Raise ONE OPERATION.
12. Depress START COMPUTE
13. When the light on the typewriter glows, type a PAV (or PPA) pseudo-instruction.
14. After that output, the typewriter light will glow. To have ROAR produce a tape containing the linkage symbols and their equivalent locations, proceed as follows:

    a. If only 5-character symbols have been used for linkage, type a 5CS psuedo-instruction.
    b. If only global symbols established with SET have been used, type an RST pseudo-instruction followed by a PAS pseudo-instruction.
    c. If a combination of both kinds of linkage symbols have been used, enter all 3 pseudo-instructions in this order: 5CS, RST, PAS.

When a section is to be added to a previously assembled program, read in the REG-EQR-SET tape used with the original section, followed by the output tapes obtained with the PAV, 5CS and PAS pseudo-instructions as explained above. This must be done prior to entering the symbolic tape for the current section to be assembled. If the END pseudo-instruction on the previous section provided a transfer to the bootstrap track, it is not necessary to have ROAR punch a bootstrap for the new section. When the final section is assembled, the END pseudo-instruction may be used to transfer to the beginning of the program, provided its symbolic designation has been entered as a linkage symbol or its absolute address is known (from an EQR or a previous listing).

When all checkout is complete, the various symbolically-coded sections may be reproduced onto a single tape and assembled. In this way the programmer will have one complete symbolic tape and one hexadecimal tape instead of several. An alternative is to combine all the assembled sections onto one tape. However, when that tape is read into the computer, a halt will occur preceding the transfer code at the end of each section. Depressing START COMPUTE will cause the transfer to be executed and loading to continue.

OPTIMIZING EXIT FROM SUBROUTINE

When subroutines are used with a program, it is desirable to provide optimum entries to and exits from them. Since the entry location will be assigned an absolute address the first time its symbolic location is seen by ROAR, the only time the entry to a subroutine will be optimum is the first time the calling sequence is used. However, since a different exit instruction can be used each time the calling sequence is written, the exit instruction can be optimum every time. To make the exit optimum, the programmer will use the NXT pseudo-instruction as described in the following examples.

EXAMPLE A

Assume the following instructions form a calling sequence for a subroutine which requires the exit instruction to be in the Lower Accumulator when control is transferred to the subroutine. SQRT is the entry point of the subroutine and the location of the first instruction which stores the exit instruction from the Lower Accumulator in some memory location.

The coding for the calling sequence might appear as

```
*RAU*ARG**              01513   02   01515   01517   *
*RAL*EXIT*SQRT*         01517   03   01519   01521   *
EXIT*STU*RT1*N-AD*      01519   24   01621   01523   *
```

When the subroutine is assembled, its first instruction will be something like:

```
SQRT*CLL*OUT*TEST*      01521   27   03723   03725*
```

Further on in the subroutine, the instruction which transfers control to the exit instruction might be:

```
FIN*TBC*OUT*LOOP*       03914   23   03723   03718   *
```

Thus the exit instruction of the calling sequence (*STU*RT1*N-AD*) will be executed at word time 23, but the Data-address field of that instruction has a sector 21. This means that a full drum revolution is wasted every time the subroutine is used.

However, if an NXT pseudo-instruction had been used in the calling sequence, it could be assembled as

```
*RAU*ARG**              01513   02   01515   01517   *
*RAL*EXIT*SQRT*         01517   03   01519   01521   *
*NXT*DELAYD**                                        *
EXIT*STU*RT1*N-AD*      01519   24   01525   01527   *
```

so that when the exit instruction is executed at word time 23, the Data-address has an optimum sector 25.

EXAMPLE B

In some subroutines, the first instruction does not store the exit instruction, but rather it is done several word times later. In such a case, 2 NXT pseudo-instructions are needed: one having the DELAYD address and the other having a numeric address to allow for the additional word times.

Assume the third instruction of a subroutine is the one which stores the exit instruction. The 3 instructions might appear as:

```
SINE*CLU*ARG**          01521   26   03823   03825   *
*EXC*898**              03825   09   00827   03829   *
*CLL*OUT*LOOP*          03829   27   03831   03833   *
```

Thus in order to provide an optimum exit, the calling sequence would use 2 NXT pseudo-instructions:

```
*RAU*ARG**              01513   02   01515   01517   *
*RAL*EXIT*SINE*         01517   03   01519   01521   *
*NXT*DELAYD**                                        *
*NXT*8**                                             *
EXIT*STU*SINX*N-AD*     01519   24   01533   01535   *
```

The DELAYD accounts for 1 of the 3 instructions in the subroutine, and the second NXT pseudo-instruction allows an additional latency of 8 word times to account for the other 2 instructions, making a total latency of 12 word times.

Then, when the exit instruction is executed at word time 31, the Data-address field will be an optimum sector 33.

HEADER TAGS VERSUS GLOBAL SYMBOLS

When a program is coded and assembled in sections or when numerous standard subroutines are used with a program, it is possible that the same symbol may be used inadvertently to represent more than one memory location. Header tags allow identical symbols to be used for different locations in different portions of a program without being ambiguous to ROAR. By preceding a portion of coding with a TAG or HED pseudo-command, the programmer can be sure that any symbols in that portion having fewer than 5 characters will be unique to ROAR. Since 5-character symbols are reserved for linkage symbols, no duplication of usage should occur. However, it became evident that other than 5-character symbols were needed for linkage purposes, so the SET pseudo-command was provided. This pseudo-command permits ROAR to read a list of symbols and to store them in the Set Table. Then, when a portion of a program is to be assembled, instead of using a header tag, the programmer may have ROAR clear all except the SET symbols from the Symbol Table. In this manner the SET symbols remain in the Symbol Table and retain the absolute addresses they were assigned. Since all other symbols are removed from the Symbol Table, they will be new to ROAR and will be assigned new addresses the next time they are encountered.

These two concepts—header tags and SET symbols—were not intended to be used together in the same sequence and, therefore, are not compatible in one respect: if a SET symbol of fewer than 5 characters is used in a sequence of coding that is under the influence of a header tag, the tag will be prefixed to the symbol, and ROAR will no longer recognize it as a SET symbol. Consider for example this sequence of coding:

```
*SET*FBR*FIX**
LOC*RAU*BCK*FBR*      00000   02   00002   00004   *
BCK*EXC*198*GO*       00002   09   00104   00007   *
```

The global symbol FBR has been assigned location 00004. If, later in the program, this sequence appeared:

```
*TAG*A**                                          *
NOW*RAU*BAK*FBR*      00008   02   00010   00012   *
GO*RAL*FLG*FIX*       00014   03   00016   00018   *
```

ROAR will annex the tag to the symbols and will consider them as new symbols, thus assigning new locations to them. If an RST pseudo-instruction were used to remove all except SET symbols from the Symbol Table, ROAR would remove everything except the FBR (location 00004). The symbols FBR (location 00012) and FIX (location 00018) have been tagged with an "A"; therefore, ROAR does not recognize them as SET symbols.

It is possible to use both SET symbols and header tags without conflict if the programmer observes the following rules:

1. Establish global symbols with the SET pseudo-instruction, as usual.
2. Establish header tags, as usual.
3. Do <u>not</u> use any global symbols having fewer than 5 characters in a section of coding that is under the influence of a header tag.
4. Remove the header tag at the end of the section.

INPUT TAPE PREPARATION 7

Input to ROAR is in Normal (6-bit) Mode. Any characters with a code from 16 through 62 may be entered into the computer. Input may be from the typewriter or from punched tape. Usually a symbolic program tape is prepared and is used for the input to ROAR. This reduces the possibility of input errors during the assembly since the punched tape can be listed, proofread, and corrected before the assembly begins.

TAPE CHARACTERS

The tape-reader moves the tape under a set of "read brushes" which close an electrical circuit when a tape hold (punch) passes beneath them. The tape has 7 channels. Any or all of these channels may have a punch in one horizontal row. A combination of punches in channels 1 through 6 is called a character. Channel 7 is called the parity bit.

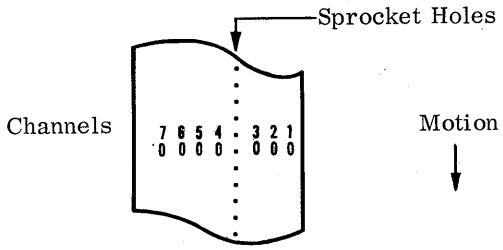


FIGURE 7.1 TAPE CHANNELS

PARITY CHECKING

The RPC-4000 uses a system of tape validating based on an even parity convention. If the number of punches in a character is odd, a parity bit will be punched so that the number of bits in all 7 channels is always even. Parity checking is performed by the input-output unit to insure maximum accuracy and occurs only during on-line operation. Operating on-line (with the computer), the input-output equipment responds to commands issued by the computer. Anything typed on the typewriter or read by the reader enters the accumulator, and therefore a parity check is performed. Operating off-line has the same effect as physically disconnecting the RPC-4500 Tape-Typewriter system from the computer. Therefore, if a program is being executed by the computer and does not use the typewriter for input or output, the typewriter may be used off-line while the computer is operating. The same is true for all input-output devices.

RULES FOR TAPE PREPARATION

The following rules for punching tapes from coding sheets apply to all instructions and pseudo-instructions, with one exception: Rule 1 does not apply to those pseudo-instructions which deviate from the standard format of 5 information fields.

1. A stop code is required following each of the five fields: Location, Order, Data-address, Next-address, Comments. It is required whether the field is filled or blank. If stop codes are not in the required places, an error halt will occur.

2. Following each instruction, a carriage return must be punched on the tape.

3. Leading zeros in a field are optional, but separating and final zeros are required.

   a. Regional addresses must contain the region symbol and 5 numeric characters which may be separating zeros, e.g., Z00008. Spaces are not acceptable in lieu of zeros.

   b. If a numeric order is to be indexed, the X must appear to the left of 2 characters, e.g., an indexed halt order could be X00, but not X0 or just X. A blank order is illegal.

4. ROAR differentiates between a blank address and a zero address; a single zero is sufficient for the distinction.

5. Place a NIX pseudo-instruction at the end of each tape if an END is not used.

PROCEDURES FOR PUNCHING INPUT TAPES

To punch tapes off-line, the following sequence should be followed:

1. Depress MASTER RESET to de-select all on-line units.
2. Depress TYPEWRITER SELECT.
3. Depress PUNCH SELECT.
4. Depress CONDITIONAL STOP.
5. Depress START READ. Light on typewriter glows.
6. Whatever is typed will be punched.

Correcting Errors While Punching Input Tapes

If an error occurs while the tape is being punched, it can be corrected as follows:

1. Depress SPECIAL bar on typewriter keyboard and hold it down.
2. Depress BACKSPACE key as many times as necessary. This moves both the typewriter carriage and the punched tape.
3. While depressing the SPECIAL bar, type X over all the characters to be deleted. The characters on the tape will be deleted, and an X will be imprinted over the characters on the typed listing.
4. Release SPECIAL bar and continue typing as before.

Correcting Errors After Punching Input Tapes

If an error is discovered after the input tape has been completed, it can be corrected by duplicating the tape that precedes the error, inserting the correct information, and duplicating the remainder of the tape. It is advantageous to duplicate tapes on-line instead of off-line so that parity checking can be done. If a character is misread or if the original tape contains a poorly punched character, a parity error is indicated and can be corrected at that time rather than during the assembly.

DUPLICATING TAPES ON-LINE To duplicate tapes on-line, the following sequence should be followed:

1. Depress MASTER RESET.
2. Depress TYPEWRITER TO COMPUTER.
3. Depress ONE OPERATION.
4. Depress SET INPUT.
5. Depress EXECUTE LOWER ACCUMULATOR.
6. Depress START READ (on-line). Light on typewriter glows.
7. Type: 40000000D0000000*. This puts a self-addressed input order in the Double-length Accumulator. The equivalent ROAR-language words would be:

```
INP   0   0
CLU   0   0
```

8. Raise EXECUTE LOWER ACCUMULATOR.
9. Raise ONE OPERATION.
10. Depress MASTER RESET.
11. Position original tape in Reader.
12. Select READER TO COMPUTER.
13. Select COMPUTER TO PUNCH.
14. Select COMPUTER TO TYPEWRITER.
15. Depress INPUT DUPLICATION SELECT.
16. Depress START COMPUTE. The original tape will be read and duplicated.

CORRECTING ERRORS ON-LINE When the tape in the reader nears the error:

When the tape in the reader nears the error:

1. Depress ONE OPERATION. The reader will stop each time it encounters a stop code.
2. Depress START COMPUTE each time another word is to be read.
3. If the error is within a word and it is desired to read one character at a time, depress SINGLE CHARACTER MODE after ONE OPERATION is depressed. Only 1 character will be read each time START COMPUTE is depressed.
4. When the last correct character (or word) prior to the error has been read, depress MASTER RESET.
5. Select READER TO COMPUTER and COMPUTER TO TYPEWRITER.
6. Depress START COMPUTE and read past the error.
7. Depress MASTER RESET.
8. Raise ONE OPERATION.
9. Raise SINGLE CHARACTER MODE if it has been depressed.
10. Select TYPEWRITER TO COMPUTER and COMPUTER TO PUNCH.
11. Depress START COMPUTE. Light on typewriter glows.
12. Type the correct information.
13. Depress MASTER RESET.
14. Select READER TO COMPUTER, COMPUTER TO TYPEWRITER, and COMPUTER TO PUNCH.
15. Depress START COMPUTE to continue duplicating the original tape.

CORRECTING PARITY ERRORS If a parity error should occur while duplicating a tape on-line, it can be corrected by the following method:

1. Depress STOP READ (on-line).
2. Depress PARITY MONITOR RESET.
3. Depress MASTER RESET.
4. Depress TYPEWRITER SELECT and PUNCH SELECT.
5. Depress START READ (off-line). Light on typewriter glows.
6. Depress SPECIAL bar on the typewriter keyboard and hold it down.
7. Depress BACKSPACE key 1 time (the error would be in the last character punched).
8. While still depressing the SPECIAL bar, type X.
9. De-select TYPEWRITER SELECT and PUNCH SELECT.
10. Select TYPEWRITER TO COMPUTER and COMPUTER TO PUNCH.
11. Depress START COMPUTE and type the correct character.
12. Depress MASTER RESET.
13. Select READER TO COMPUTER, COMPUTER TO TYPEWRITER, and COMPUTER TO PUNCH.
14. Depress START COMPUTE to continue duplicating.

Figure 7. 2 shows the coding for a sample problem; Figure 7. 3, the way a typed listing of the input tape would appear.

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
	RES	800	12263	RESTRICT PROGRAM TO TRACKS 0-7
	EQR	INP	100	DATA INPUT PROGRAM AT 100
	EQR	OUT	400	DATA OUTPUT PROGRAM AT 400
BEGIN	RAU	QUE		5 AT Q = 17
	RAL		INP	→ DATA INPUT; READ A
	STU	A		STORE A
	RAU	QUE		5@17
	RAL		INP	READ C
	STU	C		STORE C
	RAU	QUE		5@17
	RAL		INP	READ X
	ADU	A		A + X
	MPY	X		X (A+X)
	DVU	C		AX+X²/C = F (X)
	PRC	CR		CARRIAGE RETURN
	RAL	CODE		Q=5; P=8
	LDX		OUT	PRINT F (X)
	SNS	198		TEST SENSE SWITCH 1
	TBC	BEGIN		SS1 DEPRESSED: RETURN FOR NEW DATA
	HLT	0	BEGIN	HALT; DEPRESS START TO START OVER.
QUE	HLT	5	0	Q OF 5 FOR DATA INPUT
CODE	HLT	5	8	Q OF 5, P OF 8 FOR DATA OUTPUT
	END		BEGIN	

FIGURE 7. 2 SAMPLE PROBLEM CODING $\left(f(x) = \frac{ax + x^2}{c}\right)$

```
*RES*800*12263*RESTRICT PROGRAM TO TRACKS 0-7*
*EQR*INP*100*DATA INPUT AT 100*
*EQR*OUT*400*DATA OUTPUT AT 400*
BEGIN*RAU*QUE**5 AT Q = 17*
*RAL**INP*READ A*
*STU*A**STORE A*
*RAU*QUE***
*RAL**INP*READ C*
*STU*C**STORE C*
*RAU*QUE***
*RAL**INP*READ X*
*ADU*A**A+X*
*MPY*X**X(A+X)*
*DVU*C**AX+X^2/C*
*PRC*CR**CARRIAGE RETURN*
*RAL*CODE**Q=5,P=8*
*LDX**OUT*PRINT F(X)*
*SNS*198**TEST SENSE SWITCH 1*
*TBC*BEGIN**SS1 DOWN = LOOP FOR NEW DATA*
*HLT*0*BEGIN*FINISHED*
QUE*HLT*5*0*Q=5 FOR D.I.*
CODE*HLT*5*8*Q=5,P=8 FOR D.O.*
*END**BEGIN**
```

FIGURE 7.3 SAMPLE PROBLEM INPUT TAPE LISTING

OPERATING PROCEDURES 8

ROAR MASTER TAPE

The master tape containing the ROAR assembler is a hexadecimal tape consisting of a bootstrap, several tape records, and a transfer code. The bootstrap is used to load the ROAR program in memory. The tape records comprising the program itself are each followed by a checksum. The final item on the tape is a transfer code which will allow the operator to transfer control to the beginning of ROAR.

Bootstrap Procedure

The procedure for loading the ROAR master tape is as follows:

1.	Place	the tape in the Reader.
2.	De-select	all off-line units.
3.	Depress	MASTER RESET to de-select all on-line units.
4.	Select	READER TO COMPUTER.
5.	Depress	ONE OPERATION.
6.	Depress	SET INPUT.
7.	Depress	EXECUTE LOWER ACCUMULATOR.
8.	Depress	START COMPUTE. Wait for the Reader to stop.
9.	Depress	START COMPUTE.
10.	Raise	EXECUTE LOWER ACCUMULATOR.
11.	Depress	SET INPUT.
12.	Raise	ONE OPERATION.
13.	Depress	START COMPUTE. The bootstrap will now read in and will load the hexadecimal program tape without stopping.

After ROAR is stored in memory, the computer will halt. This halt occurs before the transfer code is executed to facilitate loading the symbolic tape of the object program. Depressing START COMPUTE will cause the transfer code to be executed and ROAR to begin the assembly.

Manual Transfer To Roar

If ROAR is already in memory, the following procedure may be used to transfer control to the beginning location:

1.	Select	TYPEWRITER TO COMPUTER.
2.	Depress	ONE OPERATION.
3.	Depress	SET INPUT.
4.	Depress	EXECUTE LOWER ACCUMULATOR
5.	Depress	START READ (on-line).
6.	Type	GOTOROAR*
7.	Raise	EXECUTE LOWER ACCUMULATOR.
8.	Raise	ONE OPERATION.
9.	Depress	MASTER RESET.
10.	Select	READER TO COMPUTER. Position input tape in the reader.
11.	Depress	START COMPUTE.

The computer will transfer to the entry point GOTOROAR, and ROAR will begin the assembly. When the entry point GOTOROAR is used, ROAR executes its

initialization routines. Therefore, use of this entry point clears the Symbol Table, the Availability Table, and all internal control tables. ROAR is thus prepared to begin a new assembly.

If it is desired after stopping to continue assembling without clearing the Symbol Table and Availability Table, the entry point NOWBEGIN should be typed in step 6 above. This entry point bypasses the initialization routines and preliminary printouts (see step 3 of ASSEMBLY PREPARATIONS below). NOWBEGIN is used when it is desired to output the Availability Table, the Symbol Table, or the 5-character symbols after the assembly is terminated following an END code.

ASSEMBLY PREPARATIONS

Prior to assembling the object program certain preparations must be made:

1. Two tab stops are required. Relative to the left margin stop, they should be placed at increments of 27 and 57 spaces.

2. If a bootstrap is not desired on the hexadecimal tape that ROAR will punch, depress SENSE SWITCH 8.

3. Provide preliminary information to ROAR regarding:

    a. Input-Output Selections.
    b. Subroutine Tape Region Storage.
    c. Bootstrap Track.

Input-Output Selections

The operator can select the input-output units ROAR will use during an assembly. Certain selections can be made by Sense Switch settings, for example bypassing the listing of input-output information. (Sense Switch options are explained later in this chapter.) However, on many occasions an operator may desire to make input-output selections other than (or in addition to) those available through Sense Switch settings. For example, he may wish to use an auxiliary unit to input the symbolic tape and to punch the hexadecimal program tape. Such selections can be made at the beginning of the assembly in response to a preliminary printout.

After START COMPUTE has been depressed to begin the assembly, ROAR will cause the typewriter to execute a carriage return and to type:

I/O SELECTIONS-

If no changes from the normal selections are to be made, type only a stop code. (Input-output devices normally used by ROAR are listed under "Assembly Procedures" in this chapter.)

Different selections are made by entering the numeric input-output selection codes (see Appendix F). The order in which they must be entered is as follows:

1. Select the input device for entering the symbolic program.
2. Select the output device for the hexadecimal output.
3. Select the output device for the decimal listing.
4. Select the output device for the preliminary questions and error printouts.
5. Select the input device for answering the preliminary questions.

If any one unit is to be changed, all <u>preceding</u> fields must be filled. For example, if the output device for the hexadecimal output is to be changed and all

other devices are to remain as normal selections, both field 1 and field 2 must be entered. The question and answer might appear as

I/O SELECTIONS-64*113**

Code 64 selects the normal input unit (RPC-4500 Reader), and code 113 selects the RPC-4600 Punch for hexadecimal output. It is not necessary to enter all 5 fields. An extra stop code indicates that the last selection has been made. If all 5 fields are entered, the extra stop code is unnecessary.

This method of selecting input-output units does not invalidate or override Sense Switch options. For example, if the RPC-4600 Typewriter is selected for output of the decimal listing, depressing SENSE SWITCH 32 will still bypass listing the decimal output.

CHANGING INPUT-OUTPUT SELECTIONS Once the selections have been made, the devices remain selected until changed. They may be changed by one of the following:

1. Transfer to entry point GOTOROAR and enter new selection codes in response to the question. (Typing only a stop code will leave previously selected devices still selected.)

2. Re-load ROAR.

Subroutine Tape Region Storage

Following the selection of input-output devices, the typewriter will execute a carriage return and will type:

SUBROUTINE TAPE REGION STORAGE-

If a Subroutine Library Tape is not to be used and regions in the object program are not to be designated with RLR pseudo-instructions, such storage is not needed. Type only a stop code, and ROAR will proceed.

If a Subroutine Library Tape or RLR pseudo-instructions are to be used, the area of memory for that type of region must be reserved at this time. Only one sequential group of locations may be reserved. Type, in track and sector notation, the first location of the area to be reserved followed by a stop code. After the typewriter executes a space, type the final location of the area, again in track and sector notation, followed by a stop code. ROAR will make unavailable all locations between and including these addresses.

Bootstrap Track

After the "Subroutine Tape Region Storage-" printout and response, the typewriter will execute a carriage return and a line feed and will type:

BOOTSTRAP TRACK-

(If SENSE SWITCH 8 was depressed prior to depressing START COMPUTE to begin the assembly, this printout will be omitted.)

Type the track where the bootstrap program is to be stored. Care must be taken that the bootstrap track is one where no part of the object program will be stored, preferably a track which is not otherwise used. However, if space does not permit this, use a track that the object program will later use for data storage.

After the bootstrap program has been punched, ROAR will transfer control to the input device selected for entering the symbolic program. Since all further

communication with ROAR can be made through the use of pseudo-commands, a manual restart to modify standard operation is not normally necessary.

ASSEMBLY PROCEDURES

The input-output devices normally used by ROAR during an assembly are as follows:

1. RPC-4500 Reader (Input symbolic program) Code 64.
2. RPC-4500 Punch (Output hexadecimal tape) Code 97.
3. RPC-4500 Typewriter (Output decimal listing) Code 98.
4. RPC-4500 Typewriter (Output preliminary questions and error printouts) Code 98.
5. RPC-4500 Typewriter (Input answers to preliminary questions) Code 68.

Different selections may be made as previously explained under "Assembly Preparations."

Interrupting An Assembly

The operator may want to use the normal input-output devices for the assembly but enter initial reservations, etc., from the keyboard. To accomplish this, the operator would indicate no changes in input-output selections in response to the preliminary question, and at a time when ROAR is on an input order (e.g., as the blank tape preceding the first punched character is moving under the read head) he would proceed as follows:

1. Depress STOP READ.
2. Depress MASTER RESET.
3. Select TYPEWRITER TO COMPUTER.
4. Select COMPUTER TO TYPEWRITER.
5. Depress START READ. Light on typewriter will glow.

ROAR will now accept input from the typewriter keyboard. To return to reader input, repeat the same procedure except at step 3 select READER TO COMPUTER.

If ROAR is halted by a NIX pseudo-instruction, assembly may be continued by depressing START COMPUTE.

The assembly of every program should be concluded with an END pseudo-instruction which provides a transfer to a desired location and a final checksum.

NOTE: This transfer may be made to the bootstrap if further program read-in will be necessary. (See "Sectional Assemblies," Chapter 6.)

Sense Switch Options

Various options for input and output are available through Sense Switch settings:

Sense Switch	Use when Depressed
1	Functions only after an error halt. Each time START COMPUTE is depressed, ROAR will read the symbolic input tape until a stop code is sensed and list that information on the RPC-4500 typewriter, but will not assemble it. After SENSE SWITCH 1 is raised, depressing START COMPUTE prepares ROAR for an input to allow error recovery.

2	Bypass listing of input.
4	Use Photo-Reader for input.
8	Do not output a bootstrap.
16	List decimal output on RPC-4500 Typewriter in addition to the selected output device.
32	Bypass decimal output.

NOTE: With both SENSE SWITCH 2 and 32 depressed, the only output is the hexadecimal program tape. This shortens the assembly time for already checked-out programs.

Error Printouts

When ROAR detects an error in the instruction it is assembling, it will type a notification of the kind of error and will halt. Listed below are the various error printouts:

Printout	Meaning
SYMBOL TABLE IS FULL	The Symbol Table can hold 2048 symbols during a single assembly. If more than that number are encountered, this printout is listed.
IMPOSSIBLE ADDRESS	More than 6 characters or an illegal special address was entered in an address field.
UNASSIGNED REGION	Use of a regional address before the region is designated or an erroneously typed region symbol will cause this listing.
DB FULL	The Double-access Tracks can hold 128 words. If more than that number are encountered, this printout occurs.
LOCATION NOT BLANK	The Location field of an instruction was filled when it should have been left blank.
LOCATION IS NECESSARY	The Location field of an instruction was left blank when it should have been filled.
PSEUDO-OP NOT IN TABLE	An order or a pseudo-command is input incorrectly, e. g. , RAX for RAU, or NXI for NIX, or the assembly program is out of phase and reading an address as an order.
D AND N BOTH BLANK	Both the Data-address and Next-address fields were left blank when one or both should have been filled.
BLANK N ADDRESS	The Next-address field was left blank when it should have been filled, e. g. , following END.

Printout	Meaning
ADDRESS NOT DECIMAL FOR PSEUDO-OP REG RESORAVL	A symbolic or blank address was entered for one of these pseudo-instructions instead of a decimal address.
INCORRECT ENTRY FOR DEC	The value given with the DEC pseudo-instruction cannot be held at the indicated "q."
DRUM IS FULL	All available memory has been used or there is not enough space for the indicated reservation.
INSUFFICIENT SUBROUTINE REGION STORAGE	During the assembly of the subroutines from the Library Tape, the space reserved for regional storage is exceeded.
CHECKSUM WRONG	If a checksum error occurs while ROAR is reading an Availability Tape, this printout occurs.

Error Recovery

There are two kinds of errors which might occur during an assembly: input errors and parity errors. An input error is either a punching or coding error which ROAR will detect and will indicate with an error printout. A parity error occurs when a character containing an uneven number of bits (counting all 7 channels) is read. This error is indicated by the PARITY MONITOR RESET light. It is possible to recover from these errors without loss of the hexadecima program tape.

INPUT ERRORS When an error is encountered in an input, ROAR clears from memory the instruction it was constructing, re-establishes internal settings (to their condition before the Location field of the instruction was read), types an indication of what caused the error, and halts. Any new symbolic address read by ROAR prior to the field in which the error occurred will be entered in the Symbol Table and will be assigned a location. For example, if the error occurred in the Next-address field, any symbols used in the Location and Data-address fields will have been processed before the computer halts.

Following an error halt, ROAR will accept either a filled or a blank Location field, depending upon the instruction that caused the error. If the Location field in the instruction which caused the error halt should be blank (i.e., was preceded by a blank Data-address of Next-address field), than a blank Location field must be entered during the recovery procedure. Conversely, if the Location field in the erroneous instruction should be filled (i.e., not preceded by a blank Data-address or Next-address field), then a filled Location field must be entered during the recovery procedure.

The following procedure should be used to correct input errors encountered during the assembly:

1. Depress MASTER RESET.
2. Select TYPEWRITER TO COMPUTER.
3. Depress START COMPUTE.
4. Type correctly the symbolically coded instruction word that contained the error. Begin with the Location of the instruction to be corrected and continue to the point where ROAR

stopped reading. The Location field may be either filled or blank as explained above.

5. Depress STOP READ (on-line).
6. Depress MASTER RESET.
7. Select READER TO COMPUTER.
8. Select COMPUTER TO TYPEWRITER.
9. Depress START READ (on-line).

ROAR will continue assembling with the input from the tape.

An error printout indicates that according to ROAR's standards the last instruction read was erroneous. However, it is possible that from a programming point of view the error actually occurred in the preceding instruction. Consider the following sequence of instructions:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
REPT	RAU	PAY		BRING AMT. SALARY
INC	ADU	BON		ADD BONUS

FIGURE 8.1 CODING ERROR

The assembly would appear as:

```
REPT*RAU*PAY**     01002   02   01004   01006   Bring amt. salary*
LOCATION NOT BLANK
```

and ROAR would halt. ROAR would consider instruction B to be incorrect because the Location field was filled. If the error actually is in instruction B and the Location field should be blank, the correction can be made simply by following the procedure outlined and by typing at step 4:

```
*ADU*
```

(It is necessary to type the Order field because ROAR will have already read that field. Anytime a blank Location field is encountered when ROAR expects to find a filled one, the Order field is read to see if it contains a pseudo-command. If it does not contain a pseudo-command, then an error halt occurs. If it does contain a pseudo-command, then the Location field of the instruction following that pseudo-command must be filled.)

However, if the error were actually in instruction A because the Next-address should not have been blank, the correction should be made as follows:

1. Follow steps 1 through 3 as outlined.
4. Type:
```
*EQV*INC*1006**
*ADU*
```
5. Follow steps 5 through 9 as before.

The EQV will establish 1006 as the absolute address for the symbol INC. Since 1006 was assigned to the blank Next-address field of instruction A, it will automatically be assigned to the blank Location field of instruction B and to any subsequent field containing INC. Obviously, this method of recovery is applicable only if the symbol INC had not been previously encountered by ROAR.

If the symbol INC had already been assigned a numeric address by ROAR, the recovery procedure would be as follows:

1. Follow steps 1 through 3 as outlined.
4. Refer to the coding sheet and locate the last instruction (prior to the one where ROAR stopped) which has a blank Location field. Starting there, type the symbolically coded instructions until the error has been corrected.
5. Follow steps 5 through 9 as before.

The corrected instructions will be made relocatable if these recovery procedures are used.

ERRORS DURING BLIND ASSEMBLIES If an error occurs during a blind assembly (i. e. , an assembly without a decimal listing), ROAR will type an indication as to what caused the error, will re-establish necessary internal settings, and will halt. For example, assume a blind assembly is being made on a program containing the following instructions:

LOCATION	ORDER	DATA ADDRESS	NEXT ADDRESS	COMMENTS
L,O,O,P	R,A,U	X,/,Y	S,Q,R,T	
	S,B,U	X,2		

FIGURE 8. 2 CODING ERROR

After instruction A has been assembled as

LOOP*RAU*X/Y*SQRT* 00117 02 00119 00121 *

ROAR would expect a filled Location field to follow. Since the Location field of instruction B is blank, ROAR would type:

LOCATION IS NECESSARY

and halt. In order to know what instruction was being assembled and to locate the error on the coding sheet, it will usually be necessary to list an instruction or two. If the RPC-4500 Reader is the input device, it can be switched to off-line mode, the typewriter activated, and as many instructions as necessary listed. Then to correct the error, follow the procedure described previously, and at step 4 type the corrections and all the instructions that were read off-line. If the RPC-4410 Photo-Reader is the input device, the tape can not be listed off-line without removing it from the unit. It is difficult if not impossible to remove the tape from the Photo-Reader and later replace it in exactly the same position. Therefore, the following procedure is recommended:

1. Depress SENSE SWITCH 1.
2. Depress START READ. ROAR will select the RPC-4500 Typewriter and will copy from the input tape—without assembling—until a stop code is sensed. This step may be repeated as many times as necessary to locate the error.
3. Manually reverse the direction of tape movement in the Photo-Reader and depress START COMPUTE repeatedly to reposition the tape at the instruction where the assembly is to be resumed. Then reverse the direction of tape motion again, i. e. , re-establish the original direction.

4. Raise SENSE SWITCH 1.
5. Depress MASTER RESET.
6. Select TYPEWRITER TO COMPUTER.
7. Depress START COMPUTE.
8. Enter the necessary corrections.
9. Depress STOP READ (on-line).
10. Depress MASTER RESET.
11. Depress SELECT switch on the Photo-Reader.
12. Depress START READ (on-line).

ROAR will continue the assembly with input from the Photo-Reader.

PARITY ERRORS When a parity error occurs during an assembly, it is often possible to continue the assembly without loss of the output tape. The procedure is as follows:

1. Do not move the input tape.
2. Depress STOP READ (on-line) and PARITY MONITOR RESET.
3. Consult the typed listing to determine where the error occurred.

    a. If the error is in the Comments field, no corrections are necessary since this field is only copied on the typed listing. To resume the assembly, depress START READ.
    b. If the error is in any field other than Comments, it can be rectified by entering the correct information through the typewriter.

4. Depress READER SELECT and TYPEWRITER SELECT.
5. Make sure CONDITIONAL STOP is raised.
6. Depress START READ (off-line). The reader will halt at the first stop code. The tape will now be positioned at the first character of the field immediately following the one where the error occurred.
7. Raise READER SELECT and TYPEWRITER SELECT.
8. Depress MASTER RESET.
9. Select TYPEWRITER TO COMPUTER.
10. Depress START READ (on-line). The light on the typewriter will glow.
11. To cause an error printout, type any 6 hexadecimal characters, other than zero, followed by a stop code.
12. Depress START COMPUTE. This will cause ROAR to transfer to the input order for error recovery.
13. Type corrections. Begin with the Location field of the instruction that caused the error and continue to the point where the tape is positioned in the reader.
14. Depress MASTER RESET.
15. Select READER TO COMPUTER and COMPUTER TO TYPEWRITER.
16. Depress START READ (on-line).

NOTE: The rule governing the choice of blank or filled Location field during error recovery (as explained under "Input Errors") is also applicable for recovery from parity errors.

OPERATING PROCEDURES FOR ASSEMBLED PROGRAM

After ROAR has completed its assembly, the object program may be read into the computer and executed.

Bootstrap Procedure

The bootstrap procedure for loading the hexadecimal tape of the assembled program is the same as the procedure for loading the ROAR master tape except that a Sense Switch option has been provided. The setting of SENSE SWITCH 1 indicates to the bootstrap whether a modifier is to be entered:

1. Raised - The bootstrap will assume the modifier to be zero, will select the RPC-4500 Reader for input, and will load the program in the locations given on the hexadecimal tape.

2. Depressed - The bootstrap will select the RPC-4500 Typewriter for input and will halt on an input order. Type the modifier, in track notation only, followed by a stop code. The bootstrap will then select the RPC-4500 Reader as the input device for loading the program.

After the program is stored in memory, the computer will halt. This occurs before the transfer code is executed. Depressing START COMPUTE will cause the transfer code to be executed and the program to begin operation.

NOTE: The bootstrap will always select the RPC-4500 Reader as the input device for loading the program. If another device is to be used, the RPC-4500 Reader will have to be de-selected and the other device selected manually.

Correction of Checksum Error

As the program tape is being read in, the bootstrap forms a checksum. After every 100 words this checksum is compared against the checksum that ROAR computed and punched on the tape. If the comparison is successful, reading continues. If the comparison is not successful, an error printout occurs. The typewriter executes a carriage return and prints: NO. The bootstrap then initializes the word count and the checksum location and halts. To continue loading, re-position the tape to the blank space preceding the erroneous tape record and depress START COMPUTE. (If a checksum error occurs in one portion of a sectionally assembled program, the entire program should be reloaded, not just the one section containing the error.)

WARNING: A checksum error indicates the possibility that a misread word could have been incorrectly stored in memory, thus destroying something previously stored. It is better procedure to start over, following a checksum error.

Correction of Parity Error

Should a parity error occur while loading the assembled program, the PARITY MONITOR RESET light glows, and the computer halts. Examine the tape to determine whether the character on the tape was incorrect or only misread. If the character on the tape was incorrect, obtain a correct tape. If the tape was correct and the bootstrap was entirely stored in memory, use the following procedure.

1. Depress STOP READ.
2. Depress PARITY MONITOR RESET.
3. Read the 4 low-order bits in the Lower Accumulator to determine whether the last character read (the one that caused the parity error) entered correctly. If it did enter correctly, depress START READ and the loading will continue. If it did not enter correctly:

    a. Depress MASTER RESET.
    b. Select TYPEWRITER TO COMPUTER.
    c. Depress START READ (on-line).
    d. Type zeros until the Upper and Lower Accumulator are clear.

e. Depress MASTER RESET.
f. Position the tape so the character under the brushes is the first character after the last stop code that was read.
g. Depress STOP READ.
h. Select READER TO COMPUTER.
i. Depress START READ (on-line).

If the computer does not stop on a checksum error at the end of the tape record, the recovery was successful. If the checksum error does occur, the entire tape should be re-loaded.

APPENDIX A

SUBROUTINE LIBRARY TAPE

A Subroutine Library Tape may contain any number of subroutines. The order in which the subroutines are listed is governed by only one rule:

> If one subroutine calls upon another, the subroutine called must not precede the subroutine calling on it.

For example, if subroutine A calls on subroutine C, subroutine A must be on the Library Tape preceding subroutine C. Then any time subroutine A is assembled from the Library Tape, subroutine C will also be assembled.

A Subroutine Library Tape is constructed as follows:

1. *SBT*
2. Photo-Reader Search Code (13)
3. *SUB*(Number of entry codes for this subroutine)*(List of entry codes, each followed by a stop code)
4. *RLR*(Region symbol)*(Number of sectors used)*(Any remarks)* for each subroutine when applicable.
5. Subroutine.
6. Repeat items 2-5 for each subroutine.
7. To complete the tape place *SBE* and *NIX**** after the last subroutine.

If new subroutines are to be added to a Library Tape, it is recommended that they precede the subroutine already on the Library Tape. In this way there should be no conflict with the rule governing the order in which the subroutines are listed.

To add new subroutines to a Library Tape:

1. Delete the *SBT* that is on the Library Tape.
2. Construct a new tape consisting of the subroutines to be added by following steps 1-6 of the procedure described above.
3. Either splice the original Library Tape to the new one or duplicate the original Library Tape as part of the new one.

APPENDIX B

SPECIAL ROAR TABLES

ROAR uses a major part of the first half of memory for tabular storage during assembly. The tables ROAR establishes in the lower part of memory are as follows:

Table	Tracks
Symbol Table	00 - 31
Equivalence Table	32 - 47
Availability Table	48 - 51

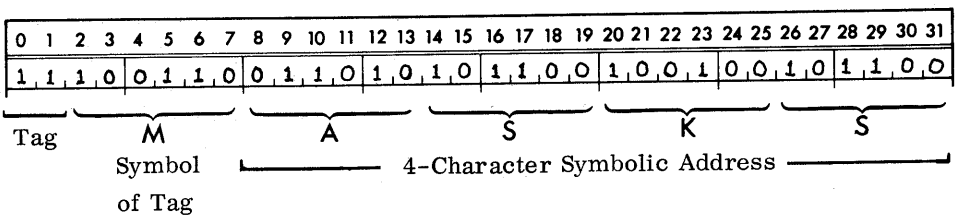
Symbol Table

The Symbol Table consists of 32 consecutive tracks and contains the symbols that are used in the program being assembled by ROAR. When a symbolic address is encountered by ROAR, it is used to compute a value representing a track of the Symbol Table. This track is then searched to determine whether the symbol had been previously encountered. If the symbol is not found, ROAR will search the track for a blank sector. If neither is present, ROAR will search the successive tracks until either the symbol or a blank sector is found. The presence of a blank sector indicates to ROAR that the symbol is new; therefore, ROAR stores the symbol in that sector and computes an optimum address for it. After determining (by reference to the Availability Table) that the address is available, ROAR will store that address in the Equivalence Table. If the desired symbol is in the Symbol Table, ROAR will refer to the Equivalence Table to find the absolute address which was previously assigned to that symbol.

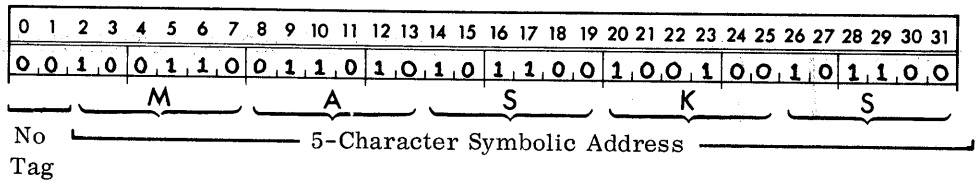
The first time the symbol is encountered by ROAR it is stored at a q of 31 in the Symbol Table. That is, a 4-character symbol would be represented by bits 8 through 31; a 5-character symbol by bits 2 through 31. Bits 0 and 1 are used to indicate the presence or absence of header tags in this manner:

Bit position 0 1	Meaning
0 0	Not tagged
0 1	Tagged by TAG
1 0	Tagged by SUB
1 1	Tagged by HED

If a header tag is present, it is stored in bit positions 2 through 7 of the word. For example, if the symbol ASKS is tagged by HED and the header tag thus assigned is "M," it would appear as



ROAR would not confuse that symbol with the 5-character symbol MASKS, which would appear as



Equivalence Table

The Equivalence Table consists of 16 consecutive tracks. In this table are stored the absolute addresses assigned to the symbolic addresses used in the object program. When a symbolic address is encountered by ROAR during an assembly, the symbol is placed in the Symbol Table; an absolute address is computed and is stored in the Equivalence Table.

If the symbol is stored in an even-numbered track in the Symbol Table, the absolute address will be stored in the Data-address field of the appropriate location in the Equivalence Table. If the symbol is stored in an odd-numbered track in the Symbol Table, the absolute address will be stored in the Next-address field of the appropriate location in the Equivalence Table.

Availability Table

The Availability Table consists of 4 consecutive tracks. Each bit position represents a memory address and the value of the position (0 or 1) signifies the availability status of the address. ROAR uses the Availability Table as a record of which locations have been used or reserved and which have not.

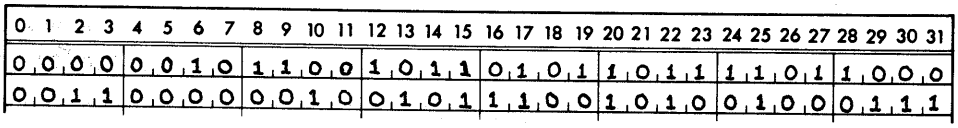
By using the PAV or PPA pseudo-command the programmer can have ROAR punch a hexadecimal tape of the Availability Table. If listed on the typewriter, the availability tape produces a list 64 lines long. Each line consists of 4 hexadecimal words, each preceded by a 5-digit decimal address. When converted to a binary configuration, each hexadecimal word shows the availability status of a given sector on 32 tracks. The presence of a 1 indicates the sector is available on that track; a zero indicates the sector is unavailable. The first hexadecimal word represents Tracks 000 through 031; the second, Tracks 032 through 063; the third, Tracks 064 through 095; the fourth, Tracks 096 through 127. Tracks 123 through 127 are represented as being unavailable unless specifically made available with an AVL pseudo-instruction. The track portion of the decimal address contains the first track of the 32-track group, and the sector portion denotes the sector concerned.

The following example represents a partial listing of an availability tape:

```
*RAV

00000 02CB5BD8*   03200 7D31FFFF*   06400 FFFFFFFF*   09600 FFFFFFE0*
00001 3025CA47*   03201 97EFFFFF*   06401 FFFFFFFF*   09601 FFFFFFE0*
```

The first hexadecimal word on each line would be converted to binary as:



Sector 00 is available on Tracks 6, 8, 9, 12, 14, 15, 17, 19, 20, 22, etc.
Sector 00 is unavailable on Tracks 0, 1, 2, 3, 4, 5, 7, 10, 11, 13, 16, etc.

Sector 01 is available on Tracks 2, 3, 10, 13, 15, 16, 17, 20, 22, etc.
Sector 01 is unavailable on Tracks 0, 1, 4, 5, 6, 7, 8, 9, 11, 12, etc.

APPENDIX C

COMMAND CODE TABLE

ORDER 0-4	CONSTANT 5-9	PSEUDO FLAG 10	OPTIMIZATION TYPE * 11-13	3 DIGIT CODE 14-31
00	04	0	2	HLT
00	04	0	2	SNS
01	04	0	2	CXE
02	02	0	1	RAU
03	02	0	1	RAL
04	02	0	1	SAU
05	02	0	1	MST
06	02	0	1	LDC
07	04	0	2	LDX
08	01	0	2	INP
09	02	0	1	EXC
10	04	0	1	DVU
11	04	0	1	DIV
12	07	0	3	SRT
12	07	0	3	SLT
12	07	0	3	SRL
13	07	0	3	SLC
14	04	0	1	MPY
15	02	0	1	MPT
16	04	0	2	PRD
16	04	0	2	PRC
17	04	0	2	PRU
18	02	0	1	EXT
19	02	0	1	MML
20	02	0	1	CME
21	02	0	1	CMG
22	04	0	2	TMI
23	04	0	2	TBC
24	02	0	1	STU
25	02	0	1	STL
26	02	0	1	CLU
27	02	0	1	CLL
28	02	0	1	ADU
29	02	0	1	ADL
30	02	0	1	SBU
31	02	0	1	SBL
	0	1	0	HEX
	0	1	0	ALF
	0	1	0	DEC

* TYPE 1 = Data-address + Constant; except, when indexed = Location + Constant + 2
TYPE 2 = Location + Constant
TYPE 3 = Location + Data-address + Constant; except, SLC = Location + Constant

APPENDIX D

MODULO-EIGHT TABLE

SYMBOLIC ADDRESS	EQUIVALENT MOD-8 SECTORS							
RECRC0 or RECRC8	00	08	16	24	32	40	48	56
RECRC1	01	09	17	25	33	41	49	57
RECRC2	02	10	18	26	34	42	50	58
RECRC3	03	11	19	27	35	43	51	59
RECRC4	04	12	20	28	36	44	52	60
RECRC5	05	13	21	29	37	45	53	61
RECRC6	06	14	22	30	38	46	54	62
RECRC7	07	15	23	31	39	47	55	63

BASIC EXC DATA-TRACK SETTINGS

COMMAND	DATA ADDRESS	EFFECT
EXC	098	No Operation.
EXC	198	Exchange Upper into Lower.
EXC	298	Exchange Lower into Upper.
EXC	498	Exchange Upper into Index.
EXC	898	Exchange Index into Upper.
EXC	1698	Change Lower to 8-word Mode.
EXC	3298	Change Lower to 1-word Mode.
EXC	4898	Reverse state of Lower.
EXC	6498	Reserved; if used, it is at present an effective No Operation.

APPENDIX E

ALPHANUMERIC CODES

The following list gives the tape codes and the computer's internal configurations of the typewriter keyboard.

NUMERIC	DEFINITION		BINARY	NUMERIC	DEFINITION		BINARY
00	Tape feed		000 000	32	g	G	100 000
01	Carriage return		000 001	33	h	H	100 001
02	Tab		000 010	34	i	I	100 010
03	Backspace		000 011	35	j	J	100 011
04			000 100	36	k	K	100 100
05	Upper Case		000 101	37	l	L	100 101
06	Lower Case		000 110	38	m	M	100 110
07	Line feed		000 111	39	n	N	100 111
08	*Stop Code		001 000	40	o	O	101 000
09			001 001	41	p	P	101 001
10			001 010	42	q	Q	101 010
11	End of Block		001 011	43	r	R	101 011
12			001 100	44	s	S	101 100
13	Photo-Reader EOM		001 101	45	t	T	101 101
14			001 110	46	u	U	101 110
15			001 111	47	v	V	101 111
16	0	)	010 000	48	w	W	110 000
17	1	°	010 001	49	x	X	110 001
18	2	"	010 010	50	y	Y	110 010
19	3	#	010 011	51	z	Z	110 011
20	4	Σ	010 100	52	,	$	110 100
21	5	Δ	010 101	53	=	:	110 101
22	6	@	010 110	54	[	;	110 110
23	7	&	010 111	55	]	%	110 111
24	8	'	011 000	56			111 000
25	9	(	011 001	57			111 001
26	a	A	011 010	58	+	?	111 010
27	b	B	011 011	59	-	_	111 011
28	c	C	011 100	60	.	.	111 100
29	d	D	011 101	61	Space		111 101
30	e	E	011 110	62	/	÷	111 110
31	f	F	011 111	63	Code delete		111 111

APPENDIX F

INPUT-OUTPUT SELECTION CODES

(SYSTEM INCLUDES AN RPC-4500 & RPC-4600)

D TRACK	INPUT SELECTED	OUTPUT SELECTED
64	4500 Reader	
65	4500 Reader	4500 Punch
66	4500 Reader	4500 Typewriter
67	4500 Reader	4500 Punch & Typewriter
68	4500 Typewriter	
69	4500 Typewriter	4500 Punch
70	4500 Typewriter	4500 Typewriter
71	4500 Typewriter	4500 Punch & Typewriter
72	4410 Photo--fwd & Search	
73	4410 Photo--rev & Search	
74	4410 Photo--fwd	
75	4410 Photo--rev	
76-79	Available	
80	4600 Reader	
81	4600 Reader	4600 Punch
82	4600 Reader	4600 Typewriter
83	4600 Reader	4600 Punch & Typewriter
84	4600 Typewriter	
85	4600 Typewriter	4600 Punch
86	4600 Typewriter	4600 Typewriter
87	4600 Typewriter	4600 Punch & Typewriter
88-94	Available	
95	Master Reset--Reset all units	
96	Available	
97		4500 Punch
98		4500 Typewriter
99		4500 Punch & Typewriter
100	Available	
101		4500 Punch
102		4500 Typewriter
103		4500 Punch & Typewriter
104	Search Mode	
105	Search Mode	
106		4440 Punch
107-112	Available	
113		4600 Punch
114		4600 Typewriter
115		4600 Punch & Typewriter
116	Available	
117		4600 Punch
118		4600 Typewriter
119		4600 Punch & Typewriter
120-124	Available	
125	Copy mode on	
126	Copy mode off	
127	Reset output units	

APPENDIX G

SUMMARY OF RPC-4000 COMMANDS

ORDER SYMBOL	ORDER NUMBER	DATA-ADDRESS		MEANING
		TRACK	SECTOR	
HLT	00	000	Any	HALT
SNS	00	≠000	Any	SENSE Turn Branch Control OFF. If a D-track bit and a depressed Sense Switch correspond, turn Branch Control ON. If D-track 64-bit is present and a) or b) is true, turn Branch Control ON: a) Photo-reader search is not completed. b) Search completed, output device not ready.
CXE	01	A	B	COMPARE INDEX EQUAL Turn Branch Control OFF. Compare the bits of the D-address with corresponding bits of the Index Register. If they are equal, turn Branch Control ON.
RAU	02	A	B	RESET - ADD UPPER Replace the content of U with the content of memory location AB.
RAL	03	A	B	RESET - ADD LOWER Replace the content of L with the content of memory location AB.
SAU	04	A	B	STORE ADDRESS FROM UPPER Store the Data-address portion of U in memory location AB, leaving the rest of the word unchanged.
MST	05	A	B	MASKED STORE Where U contains 1's, store L in memory location AB; where U contains zeros, leave memory location AB unaltered.

ORDER SYMBOL	ORDER NUMBER	DATA-ADDRESS TRACK	DATA-ADDRESS SECTOR	MEANING
LDC	06	A	B	LOAD COUNT Replace the content of the N-track of the Index Register with the corresponding bits of memory location AB. The next instruction will be executed in Repeat Mode and will be repeated as many times as the number placed in bits 18-24 of the Index Register.
LDX	07	A	B	LOAD INDEX REGISTER Replace the content of the Data-address field of the Index Register with the number AB.
INP	08	A	B	INPUT If A = 000, read in 4-bit. If A = 064, read in 6-bit. (B is not used.) If L is at 1-word length, read into Double-length accumulator. If L is at 8-word length, read into L. No other values of the D-track should be used, since any other bits present will be duplicated into every character read.
EXC	09	A	B	EXCHANGE A may be any value within the range $1 \leq A \leq 64$. B is normally optimum, but may be any sector value. 01 U → L 02 L → U 04 U → X 08 X → U 16 L to 8-word length 32 L to 1-word length 64 Unspecified. Any combination of operations may be coded with one EXC order.
DVU	10	A	B	DIVIDE UPPER Divide U by the content of memory location AB; retain the quotient in U and the remainder in L. An overflow will turn Branch Control ON.

ORDER SYMBOL	ORDER NUMBER	DATA-ADDRESS TRACK	DATA-ADDRESS SECTOR	MEANING
DIV	11	A	B	Divide the Double-length Accumulator by the content of memory location AB. An overflow will turn Branch Control ON.
SRL	12	A	B	SHIFT RIGHT OR LEFT If A=0, shift the Double-length Accumulator right by B bits. If A= 001, shift the Double-length Accumulator left by B bits. An overflow will turn Branch Control ON.
SLC	13	A	B	SHIFT LEFT AND COUNT Normalize the double-length number. Shift left until the most significant bit is in bit position 1 of U, or until the sum of B and the number of shifts equals 64. After shifting, clear L to zero and place number of bits shifted in the D-sector of L.
MPY	14	A	B	MULTIPLY Clear L to zero. Multiply U by content of memory location AB and develop product in Double-length Accumulator.
MPT	15	A	B	MULTIPLY BY TEN If A = 000 multiply U by 10. If A = 1 multiply U by 8. If A = 2 multiply U by 2. If A = 3 multiply U by 0. If A = 064 multiply L by 10. If A = 065 multiply L by 8. If A = 066 multiply L by 2. If A = 067 multiply L by 0. If A equals any other number, the result will be a multiplication of U (if A<64) or L (if A > 64) by 10, 8, 2, or zero plus some value between 1 and 8 at a q of 31.
PRD	16	A	B	PRINT DATA ADDRESS If A<64, print (on the selected output device) the character represented by A. If A≥64, select the input-output device(s) indicated by A.

ORDER SYMBOL	ORDER NUMBER	DATA-ADDRESS TRACK	DATA-ADDRESS SECTOR	MEANING
				If the automatic interlock is not to be overridden, B must be at least 1 greater than the optimum sector.
PRU	17	A	B	PRINT FROM UPPER If A < 64, print the 4 high order bits of U as channels 1-4 and take channels 5 and 6 from A. If A>64, print the 6 high order bits of U. B is normally unoptimum.
EXT	18	A	B	EXTRACT Make U zero where memory location AB contains zeros; do not change U where AB contains 1's.
MML	19	A	B	MASKED MERGE LOWER In the bit positions where U contains zeros, retain the content of L. In the positions where U contains 1's, replace content of L with the content of memory location AB.
CME	20	A	B	COMPARE MEMORY EQUAL Turn Branch Control OFF. Compare U with memory location AB in the bits where L contains 1's. If they are equal, turn Branch Control ON. When executed in Repeat Mode, the found sector plus 1 is left in the N-sector of the Index Register.
CMG	21	A	B	COMPARE MEMORY GREATER Compare with memory under the same conditions as CME except that Branch Control will be turned ON if the content of AB is algebraically equal to or greater than the content of U in the bits compared.
TMI	22	A	B	TEST MINUS If the sign bit of U contains a 1, take the next instruction from AB. If not, proceed to Next-address as usual.
TBC	23	A	B	TEST BRANCH CONTROL If Branch Control is ON, turn it OFF and take the next instruction from AB. If not, proceed to Next-address as usual.

ORDER SYMBOL	ORDER NUMBER	DATA-ADDRESS		MEANING
		TRACK	SECTOR	
STU	24	A	B	STORE UPPER Replace the content of memory location AB with the content of U; leave U unchanged.
STL	25	A	B	STORE LOWER Replace the content of memory location AB with content of L; leave L unchanged.
CLU	26	A	B	CLEAR UPPER Replace content of memory location AB with content of U; then clear U to zero.
CLL	27	A	B	CLEAR LOWER Replace content of memory location AB with content of L; then clear L to zero.
ADU	28	A	B	ADD UPPER Add algebraically the content of memory location AB to the content of U, leaving sum in U. Overflow will turn Branch Control ON.
ADL	29	A	B	ADD LOWER Add algebraically the content of memory location AB to the content of L, leaving sum in L. Overflow will turn Branch Control ON.
SBU	30	A	B	SUBTRACT FROM UPPER Subtract algebraically the content of memory location AB from the content of U, leaving difference in U. Overflow will turn Branch Control ON.
SBL	31	A	B	SUBTRACT FROM LOWER Subtract algebraically the content of memory location AB from the content of L, leaving difference in L. Overflow will turn Branch Control ON.

APPENDIX H

SUMMARY OF ROAR PSEUDO-COMMANDS

*RES*2000*2132*Any*

RESERVE A PORTION OF MEMORY
Reserve memory locations 2000 through 2132.

REG/00051*612*Any*

ESTABLISH A REGION
Assign location 51 to /00001 and reserve locations 51 through 612.

*RLR*Q*21*Any*

DESIGNATE A RELOCATABLE REGION
Reserve 21 locations for Region Q in the area designated as Subroutine Tape Region Storage.

*PRE*0000SUBR*XXXXXXXX*XXXXXXXX*XXXXXXXX*XXXXXXXX*XXXXXXXX*XXXXXXXX*

PREPARE ROAR
Supply ROAR with the name of a subroutine and the six region reservations required for that subroutine as compiled by COMPACT.

*SET*ONE*TWO*BEGIN*FBR*SUBR**

ESTABLISH GLOBAL SYMBOLS
Enter the listed symbols in the Set Table as global symbols.

*RST****

RESTORE SYMBOL TABLE
Remove all non-global symbols from the Symbol Table and restore their locations in the Availability Table.

RRS/00019***

RELOCATE REGIONAL STORAGE
Relocate the base address of Region Slash upward by 19 words.

*AVL*12030*12150*Any*

MAKE A BLOCK AVAILABLE
Restore to the Availability Table all locations from 12030 through 12150.

*EQR*I/O*1500*Any*

EQUATE AND RESERVE LOCATION
Make symbol I/O equivalent to location 1500 and reserve that location.

*EQV*1600*OUT*Any*

MAKE EQUIVALENT
Make symbol OUT equivalent to location 1600; when the symbol OUT is first encountered in the object program, ROAR will reserve the location.

*NIX***Any*

WAIT A SECOND
Wait for start signal before continuing.

*NEW***Any*

BEGIN NEW ASSEMBLY
Transfer to the initialization portion of ROAR.

*CLS***Any*

CLEAR SYMBOL TABLE
Remove all symbols from the Symbol Table.

*END**START*Any*

END ASSEMBLY
Punch transfer address "START," punch checksum, clear word counter and checksum counter, punch tape feeds, and then halt.

*TAG*A**Any*

DESIGNATE HEADER TAG
Add header tag "A" to all symbols having fewer than 5 characters.

*HED****

ASSIGN SEQUENTIAL HEADER TAG
ROAR will assign a unique tag to be added to all subsequent symbols having fewer than 5 characters.

*NXT*DELAYD*NN*Any*

OPTIMIZE NEXT INSTRUCTION AS INDICATED

Assign the Data-sector in the instruction immediately following this pseudo-instruction the value of the preceding Next-address sector plus 4 and increase the sector portion of the Next-address field in that instruction by NN.

/00007*HEX*5F5E*10000*Any*

ESTABLISH A CONSTANT FROM HEXADECIMAL INPUT

The constant 10^8 (at a q of 27) is to be assigned the address equivalent to /00007.

2T14*DEC*17*16384*Any*

ESTABLISH A CONSTANT FROM DECIMAL INPUT

The constant 2^{14} (at a q of 17) is to be assigned the address equivalent to 2T14.

COL1*ALF*I=PRT**Any*

INPUT ALPHANUMERIC CHARACTERS

Read I=PRT in Six Bit Mode and assign it the address equivalent to memory location COL1.

*PAV***Any*

PUNCH AVAILABILITY TABLE

ROAR will punch a hexadecimal tape of the Availability Table.

*PPA***Any*

PUNCH AND PRINT AVAILABILITY TABLE

ROAR will punch a hexadecimal tape of the Availability Table and will type a decimal listing of the Table.

RAV

READ AVAILABILITY TABLE

This pseudo-command is the first one punched on the availability tape resulting from a PAV or a PPA pseudo-instruction and is necessary if ROAR is to input the tape.

*PST***Any*

PRINT SYMBOL TABLE

Print all symbols that are in the Symbol Table together with their equivalent addresses.

*5CS***Any*

PUNCH FIVE-CHARACTER SYMBOLS
ROAR will search the Symbol Table for 5-character symbols and will punch them in the form of EQR pseudo-instructions.

*PAS***ANY*

PUNCH ALL SYMBOLS
ROAR will punch all symbols from the Symbol Table in the form of EQR pseudo-instructions.

NOW*PRC*A*N-AD*Any*

PRINT A CHARACTER
The output resulting from this pseudo-instruction is the same as that from a PRD with 26 in the track portion and 99 in the sector portion of the Data-address field.

DO*SRT*5*N-AD*Any*

SHIFT RIGHT
The output resulting from this pseudo-instruction is the same as that from an SRL with 000 in the track portion and 05 in the sector portion of the Data-address field.

NOR*SLT*13*N-AD*Any*

SHIFT LEFT
The output resulting from this pseudo-instruction is the same as that from an SRL with 001 in the track portion and 13 in the sector portion of the Data-address field.

*COM*any statement desired*

COMMENT
ROAR will copy the statement from the input tape until a stop code is encountered.

(56)(57)(56)

COMPACT-GENERATED COMMENTS
ROAR will ignore all characters following this pseudo-command including stop codes, unless the stop code is preceded by the character (56).

SBT

ENTER SUBROUTINE LIBRARY MODE
This pseudo-command causes ROAR to make certain internal changes in order to assemble programs from a Subroutine Library Tape.

*SUB*4*]0005*]1005*]2005*]3005*

READ SUBROUTINE FROM LIBRARY TAPE

This pseudo-instruction precedes a program on the Subroutine Library Tape and informs ROAR that the program has 4 entry points which are]0005,]1005,]2005, and]3005.

SBE

EXIT FROM SUBROUTINE LIBRARY MODE

This pseudo-command causes ROAR to restore to its original status anything altered by the SBT.

COMMERCIAL COMPUTER DIVISION

SC 4106

PRINTED IN U.S.A.