

PRELIMINARY EDITION

ROAR

RPC-4000

Optimizer &

Assembly

Routine

MANUAL FOR PROGRAMMERS AND OPERATORS

R O A R

The optimizer and assembly routine
for the
RPC-4000 electronic computer

TABLE OF CONTENTS

I	Introduction.....	1
II	A Brief Description of the RPC 4000.....	4
III	The Assembly Program.....	21
IV	Addresses.....	24
V	Pseudo Operations.....	31
VI	Operation Codes.....	37
VII	Programming Using ROAR.....	39
VIII	Programming Examples.....	40
IX	Input Tape Preparation.....	49
X	Output Format.....	52
XI	Operator's Instructions.....	53

I

INTRODUCTION

The object of an assembly program is to allow the programmer to code instruction for instruction like actual machine language, but to be relieved of the extra chores of optimizing, keeping track of used locations, etc. With the "one over one" addressing system of the RPC-4000, optimization is doubly necessary. It was with these considerations in mind that the first major program for the RPC-4000 should be a symbolic assembler and optimizer.

The overall scheme is this:

The programmer codes his program either in symbolic language or a combination of symbolic language and machine language. The program is then processed by ROAR.

The output of ROAR is of a dual nature. There are

1. A hex tape of the assembled program.
2. A decimal copy of the assembled program along with a reproduction of the input.

These provide the programmer with a ready means of loading his program as well as a complete record for error correction and program checkout.

On the following pages will be presented a list of definitions of the terms and abbreviations used in the rest of the manual.

U	Upper half of the double length accumulator
L	Lower half of the double length accumulator
X	Index register
C	Command register
FWA	First word address, as of table
LWA	Last word address
Lo	First location of, as a table
LOC	Location, as of an instruction
D-ADDR	Data address; bits 5 through 17 of the command word
N-ADDR	Next address, the address where the next instruction is located. Also bits 18-30 of the command word
ORDER	The instruction code. Bits 0-4 of the command word
SUBROUTINE	A part of a program. A program used as a part of a larger program to perform a specific function
SYMBOLIC ADDRESS	A symbol to which ROAR assigns a machine address. A symbol contains at least one character that is not numeric
RECR	Refers to track 127, the 8 word recirculating track
DB	Double Access code
BLOCK	An area of sequential drum storage
PROGRAM	Unless specified as assembly program, program refers to the object program - the program being coded
BC	Branch control, an internal sense toggle and overflow indicator
→	Replace (Replaces)
NU	Not used

PSEUDO OPERATION An instruction to the assembly program. It takes the same form as an instruction and is reproduced on the printed output. It is not output into the program tape.

AVAILABLE An available memory location may be used by ROAR for assignment as an instruction or constant, at which time it is made unavailable.

UNAVAILABLE A memory location is unavailable when it has been reserved through a pseudo-op or assigned by the assembler. Double access tracks and RECRC are unavailable unless specifically made available.

RESERVE To make a memory location unavailable

REPEAT MODE The mode of operation in which the execution of an instruction is repeated as specified

q Applies to the scaling convention we use. Q represents the bit position in the data word that separates integer from fraction. Mathematically: the number to be represented is equal to the data word (which is considered as a fraction) multiplied by 2^q

II

A BRIEF DESCRIPTION OF THE RPC-4000

This chapter is designed to provide a programmer's view of the RPC-4000 and to acquaint the reader with some of the terms used by the assembler.

MAIN MEMORY

The memory consists of 8192 addresses broken into 128 tracks of 64 sectors. Some of these addresses are redundant as we shall see. Tracks 000 through 122 contain nothing special and may be referred to directly. Tracks 123 and 124 are the leading heads of two double access tracks and, if double access is not to be used, may be used as two more tracks of main memory. Track 125 is the trailing head 16 word times behind the head of track 123; track 126 trails the head of track 124 by 24 word times.

Track 127 is an eight word recirculating line (RECRC). In other words, the group of eight words represented by sectors 00 - 07 is duplicated 8 times as 08 - 15, 16 - 23, etc.

The following table will show the correspondence between the various sectors of track 127 as well as provide a presentation of modulo 8 equivalence for later consideration. The RECRC notation will be discussed in Chapter IV. All the sectors on each line refer to a single word of memory.

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

Figure 1

COMMAND EXECUTION

The normal operation of the "one over one" address system is as follows:

As an instruction enters the C register, the indicated operation is performed upon the memory location specified by the D address. The next instruction to be executed is located in the memory location given in the N address.

TIMING

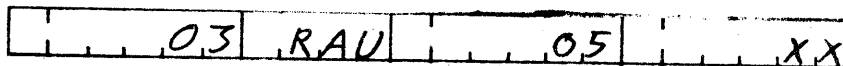
Instruction timing is generally discussed in terms somewhat removed from direct application to programming. We speak of n milliseconds access time or an addition requiring X microseconds. For the purposes of the programmer this is cumbersome. We plan to relate instruction timing to optimization and, for this purpose, two time units are sufficient.

- They are:
1. The drum revolution $1/60$ second
 $16\ 2/3$ millisecond
 2. The word time $1/64$ drum revolution
 260.4 microsecond
 The time required for word transfer
 between registers and memory

For this discussion we will neglect the track since nothing in timing depends on it.

Let us begin, as in **COMMAND EXECUTION**, with an instruction entering the C register. At this point the computer is entering phase 3 of a four phase operation. During phase 3 the computer is concerned with waiting while the drum rotates until the next sector is the one specified in the D address. The minimum time for phase 3 is one word time, so that for best optimization the D sector will be two greater than the sector in which the instruction is located.

As an illustration



SECTOR	OCCURRENCE
03	Instruction came from sector 3
04	Phase 3 Search for "next sector is it" NOW!
05	Operand sector
06	

This is the most optimum case. Consider a case which is less optimum:



SECTOR	OCCURRENCE
03	Instruction came from sector 3
04	Phase 3 Search for "next sector is it" Wait
05	Wait
06	Wait
07	Now
08	Operand Sector
09	

Once the computer establishes that the next sector is the operand sector, phase 4 is entered. Phase 4 is the execution phase of the instruction (see below).

Phase 3 is limited to one word time on the following instructions:

HLT-SNS	00	SRL	12
CXE	01	SLC	13
LDX	07	PRD*	16
INP	08	PRU*	17

so that the sector used is generally immaterial.

The time in phase 4 depends upon the instruction being executed:

Shift orders take 4 word times plus 1 word time for each bit position shifted.

The multiply and both divide orders require 67 word times of execution.

The input order is variable, depending upon the number of characters read.

All other orders require 1 word time in phase 4.

After execution, phase 1 is entered. Phase 1 is the search for the sector of the next instruction and requires a minimum of one word time in the same manner as phase 3. In phase 2, the instruction enters the command register and the cycle is complete.

As an example, consider the following instruction sequence:

	Location	Order	Data Address	Next Address
(1)	5,0,0	R,AV	3,0,7	3,1,2
(2)	3,1,2	A,DU	1,4	5,1,5

*Sectors of print orders must not be first optimum unless it is desired to bypass the print interlock.

SECTOR TIME	ACTIVITY
00	Phase 2 Command 1 enters C register
01	Phase 3 Search for sector 07 to be next
02	
03	
04	
05	
06	07 is next
07	Phase 4 Contents of 0307 to upper
08	Phase 1 Begin search for sector 12
09	
10	
11	12 is next
12	Phase 2 Command 2 to C register
13	Phase 3 Begin search for sector 14: Next
14	Phase 4 Add C(0014) to upper
15	Phase 1 Begin search for sector 15: Long wait

Note

1. Command 1 is moderately optimum .
2. Execution of command 2 is first optimum.
3. The N address of command 2 calls for the least optimum sector.

Print orders require one word time for execution. Further printing will be held up by an interlock until the previous print stroke is completed.

To summarize instruction timing for the RPC-4000, the following table will illustrate in terms of the phase structure.

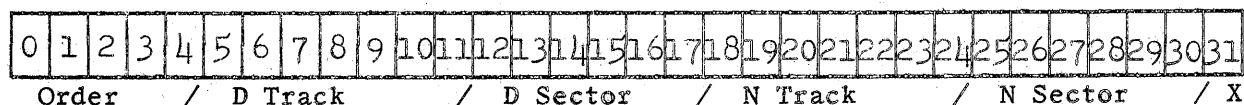
<u>Phase</u>	<u>Activity</u>
1.	Search to determine that the next sector contains the instruction: 1 to 64 word times
2.	Copy instruction into command register, adding indexing portion of X register if indexed: 1 word time
3.	Search to determine that the next sector contains operand: 1 to 64 word times (phase 3 is limited to 1 word time on certain instructions)
4	Execution of command. This is a fixed or variable amount and depends upon the order.

Note About Repeat Mode

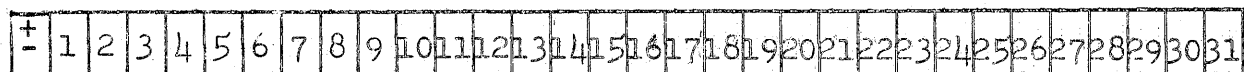
Repeat mode operation means extended phase 4 operation. In repeat mode, the execution portion (phase 4) is extended (or repeated) as many times as the count indicates (see repeat mode of operation page 11).

WORD STRUCTURE

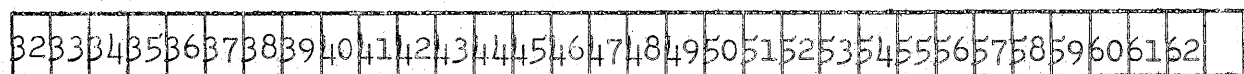
Command Word



Data Word

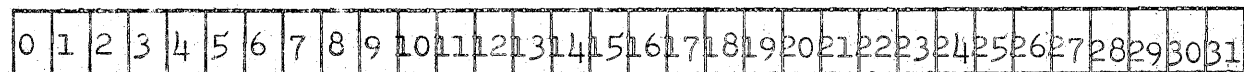


Lower Accumulator when used as lower $\frac{1}{2}$ of product, numerator for division. When used for other functions it uses the data word form.



NU

INDEX REGISTER



Not used / X Track / X Sector / Repeat count / Found sector / NU
 + 1 from compare memory instructions

REGISTERS

The RPC-4000 has four registers:

1. Command Register C
2. Upper Accumulator U
3. Lower Accumulator L
4. Index Register X

Command Register C

The command register analyzes the instruction word. The general interpretation of most commands is: Perform the indicated operation upon the contents of the memory location specified by the D address and find the next instruction in the location specified by the N address.

Upper Accumulator U

The Upper is a one word accumulator capable of arithmetic use. It holds the multiplicand and high order product for multiplication, the high order portion of the dividend and the quotient for division and its sign bit is checked by the test on minus instruction.

Lower Accumulator L

The lower accumulator may be specified by programming as one or eight words in length. It may be added to, subtracted from, and shifted in conjunction with the upper. During addition and subtraction there is no carry between upper and lower, nor between units of the 8 word lower.

For division L holds the lower half of the dividend, then the remainder. For multiplication it holds the lower half of the double length product. After a shift and count its D address bits hold the number of places shifted.

L is utilized as a mask holder for the compare memory instructions.

Index Register X

The X register has several varied uses as follows:

1. Its primary use is as an index register:

If an instruction is indexed, i. e., it contains the index bit (a one at 31), the D address portion of the index register is added to the D address of the instruction as it goes into the command register. The index register portion may be loaded by use of the LDX (07) order. It may be incremented by an indexed LDX order: "Load the X register with the D address of the LDX order, after adding in the previous contents of the X register."

The index register portion of the X register may be tested by use of the CXE (01) order followed by a TBC (23) order.

2. For repeat mode instructions, e. g., using the compare memory orders as a table look-up, adding a column of figures, etc., the N track holds and counts down the repeat count.
3. On the compare memory instructions, one greater than the sector where found, or one greater than the last sector searched (if not found), is placed in the N sector of the X register.
4. When not otherwise in use, the X register, through the use of the EXC (09) order, becomes a word of immediate access, high speed storage.

Branch Control

The Branch Control is not truly a register but an internal toggle which is capable of being in an on or off position.

The following conditions will turn it on:

- A. Overflow conditions resulting from:
 - Add and subtract in either upper or lower
 - Either divide
- B. Shifting a bit left beyond the normalized position in the upper.
- C. Successful compare (memory, X) instructions
- D. A successful match on SNS (00) instruction

The Branch Control is turned off in the following ways:

- A. As the first step in executing these instructions:
 - SNS (00) Sense (not halt)
 - CXE (01) Compare index register
 - CME (20) Compare memory equal
 - GMG (21) Compare memory greater than or equal
- B. Upon execution of the TBC (23) instruction (if on)

SPECIAL FEATURES

Lengthened Lower Accumulator

The lower accumulator may be either one or eight words in length as determined by the EXC (09) order. The 16's bit in the D track of the exchange

command changes it to 8 word length; the 32's bit changes it to one word length. The lengthened Lower may be read into, and may perform all arithmetical instructions that the one word Lower is capable of, either one word at a time by judicious placing of an operand in memory, or several words at a time by making use of the repeat mode of operation.

The programmed timing for the 8 word Lower is quite straightforward. It is timing modulo 8. A word entering the 8 word Lower from memory sector zero, for example, may only be operated on, stored into or replaced from a sector zero location, modulo 8; that is sector 0,8,16,24,32,40,48, or 56, of a track. (See figure 1.)

It is suggested that shifting be not done when the lower is lengthened. If programmed, left shifting into the upper will be a bit in turn from each of the lowers for the duration of the shift.

Repeat Mode

If the LDC (06) instruction is executed, the N track of the X register will be replaced by the corresponding bits from the memory location specified in the D address, and the execution of the succeeding instruction will be repeated as many times as given in that count.

An example of this operation will be given in Programming Example 1 on page 40 following the discussion on coding. This tool is expected to prove very powerful, for it enables us to perform table look-up (repeated CME or CMG), add a column of numbers (repeated ADU), move blocks of data (repeated RAL, STL coupled with the 8 word Lower), etc..

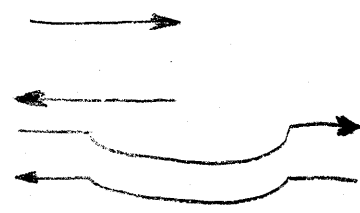
Track switching is not possible in repeat mode. For example, assume a count of 5 is loaded, followed by a CME (20) 0960 XX. The locations searched are 0960, 0961, 0962, 0963, 0900, 0901 in that order.




Command Structure

The following table provides a brief description of the RPC 4000 command structure:

ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	EFFECT ON		
		TRACK	SECTOR		U	L	X
HLT	00	000	Any	HALT	N.U.	N.U.	Index
SNS	00	≠000	Any	SENSE No operation. Turn the Branch Control on if a track bit (or more) corresponds to a depressed sense switch on the console. The track 64 bit will always turn the Branch Control on.	N.U.	N.U.	Index
CXE	01	A	B	COMPARE X EQUAL 1. Turn the BC off. 2. Compare the bits of the D address with the corresponding bits of the X register. 3. If equal, turn BC on.	N.U.	N.U.	D addr. compared Indexing is redundant
RAU	02	A	B	RESET - ADD UPPER Replace the contents of U with the contents of memory location A B	C(AB) → U	N.U.	Index
RAL	03	A	B	RESET - ADD LOWER Replace the contents of L with the contents of memory location AB	N.U.	C(AB) → L	Index
SAU	04	A	B	STORE ADDRESS FROM UPPER Store the D portion of U in the location specified by AB, leaving the rest of the word unchanged	D copied Register left unchanged	N.U.	Index
MST	05	A	B	MASKED STORE Where U contains 1's store L into memory location AB; where U contains 0's, leave the memory location unaltered	Mask	Data word	Index

ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	U	EFFECT ON	
		TRACK	SECTOR			L	X
LDC	06	A	B	LOAD COUNT Replace the contents of the N track of the X register with the corresponding bits of memory location AB. Execute the next instruction in repeat mode, <u>repeating by</u> the amount placed in X.	N.U.	N.U.	C(AB) → C(N track) Index
LDX	07	A	B	LOAD X Load the indexing portion of the X register (the D address) with AB, the D portion of the LDX order	N.U.	N.U.	AB → C(D addr.) Index
INP	08	000	N.U.	INPUT Read: 4 bit			
INP	08	064	N.U.	Read: 6 bit If L is at 1 word length, read into the double 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. This read is different from LGP-30 in that there is no shift prior to the first digit entering			Index
EXC	09	01	B	EXCHANGE Replace the contents of the Lower with the contents of the Upper. L → U U → X X → U Lower to 8 word length Lower to 1 word length Unspecified as yet Any combination of D-track bits is acceptable			Index



ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	EFFECT ON		X
		TRACK	SECTOR		U	L	
DVU	10	A	B	DIVIDE UPPER divide the Upper by C(AB): retain the quotient in U and the remainder in L	Dividend, Quotient	0, Remainder	Index
DIV	11	A	B	DIVIDE Divide the double length accumulator by C(AB)	High dividend, Quotient	Low dividend, Remainder	Index
SRL	12	000	B	SHIFT RIGHT OR LEFT Shift the double length accumulator right by B bits.			Index
	12	001	B	Shift the double length accumulator left by B bits. If an overflow is detected, turn BC on.			Index
SLC	13	N.U.	N.U.	SHIFT LEFT AND COUNT Normalize the double length number. Shift left until the most significant bit is in the high order position of the upper. After shifting clear the lower to zeroes and place the number of bits shifted in D sector of L.			Index

ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	EFFECT ON		
		TRACK	SECTOR		U	L	X
MPY	14	A	B	MULTIPLY Clear the lower to zero. Multiply the upper accumulator by the contents of memory location AB and develop the product in the double length accumulator.	Multiplicand, High order product	Zero, Lower half of product	Index
MPT	15	000	B	MULTIPLY BY TEN Multiply the upper accumulator by 10.	$C(U) \times 10 \rightarrow C(U)$	N.U.	Index
		064	B	Multiply the lower accumulator by 10.	N.U.	$C(L) \times 10 \rightarrow (L)$	
PRD	16	A<64	B	PRINT DATA ADDRESS Print the character represented by A on the selected output device.	N.U.	N.U.	Index
		A=64	B	Select the input and/or output device(s) indicated by A If the automatic interlock is to be made use of, B must not be the first optimum sector	N.U.		
PRU	17	A<64	B	PRINT FROM UPPER Print the left four bits of the upper as channels 1-4 and take channels 5 and 6 from D track of print order	Sign and left 3 bits	N.U.	Index
		064	B	Print the left six bits of the upper accumulator	Sign and left 5 bits.	N.U.	Index
EXT	18	A	B	EXTRACT Make the upper zero where memory location AB contains zeroes; do not change the upper when C(AB) has ones (LGP-30 extract)	$C(AB)EXT \rightarrow U$	N.U.	Index

ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	U	EFFECT ON		
		TRACK	SECTOR			L	X	
MML	19	A	B	MASKED MERGE LOWER (formerly Collate) "Masked Bring." For the bit positions where the upper contains zeroes, retain the contents of the lower. For the positions where the upper contains ones, replace the contents of the lower with the contents of memory location AB	Mask		Left alone or replaced	Index
CME	20	A	B	COMPARE MEMORY EQUAL Compare the upper with memory location AB in the bits where the lower contains ones. If they are equal, turn BC on and place the sector plus one in the N sector of the X register. BC is turned off immediately before comparison begins	Control word	Mask		Index Sector "found" + 1 → N sector
CMG	21	A	B	COMPARE MEMORY GREATER Compare upper with memory under the same conditions as CME except that BC will be turned on if C(AB) is algebraically greater than or equal to C(U) in the bits compared	Control word	Mask		Sector "found" +1 → N sector
TMI	22	A	B	TEST MINUS If the sign bit of the upper contains a one, take the next instruction from AB. If not, take the next instruction from the N addr. as usual	Sign checked	N.U.		Index
TBC	23	A	B	TEST BRANCH CONTROL If BC is on, turn it off and take the next instruction from AB. If not proceed to N addr. as usual	N.U.	N.U.		Index

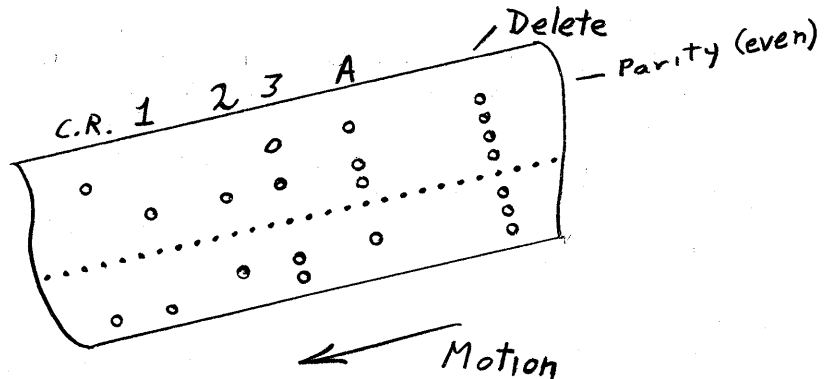
ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	EFFECT ON		
		TRACK	SECTOR		U	L	X
STU	24	A	B	STORE UPPER Replace the contents of memory location AB with the contents of the upper accumulator; leave the upper unchanged.	Stored, unaltered	N.U.	Index
STL	25	A	B	STORE LOWER Replace the contents of memory location AB with the contents of the lower; leave the lower unchanged.	N.U.	Stored unaltered	Index
GLU	26	A	B	CLEAR UPPER Replace the contents of memory location AB with the contents of the upper accumulator, then clear the upper to zeroes.	$C(U) \rightarrow C(AB)$ zero $\rightarrow C(U)$	N.U.	Index
GLL	27	A	B	CLEAR LOWER Replace the contents of memory location AB with the contents of the lower, then set the lower to zeroes	N.U.	$C(L) \rightarrow C(AB)$ zero $\rightarrow C(L)$	Index
ADU	28	A	B	ADD UPPER Add algebraically the contents of memory location AB to the contents of the upper, leaving the sum in the upper. An overflow will turn BC on.	$C(U) + C(AB) \rightarrow C(U)$	N.U.	Index
ADL	29	A	B	ADD LOWER Add algebraically the contents of memory location AB to the contents of the lower, leaving the sum in the lower. An overflow will turn BC on.	N.U.	$C(L) + C(AB) \rightarrow C(L)$	Index

ORDER SYMBOL	ORDER NUMBER	ADDRESS		OPERATION	EFFECT ON		
		TRACK	SECTOR		U	L	X
SBU	30	A	B	SUBTRACT FROM UPPER Subtract algebraically the contents of memory location AB from the upper; leave the difference in the upper. An overflow turns BC on.	$C(U) - C(AB)$ $\rightarrow C(U)$	N.U.	Index
SBL	31	A	B	SUBTRACT FROM LOWER. Subtract algebraically the contents of memory location AB from the contents of the lower, leaving the difference in the lower. An overflow turns BC on.	N.U.	$C(L) - C(AB)$ $\rightarrow C(L)$	Index

ALPHANUMERIC CODES

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

NUMERIC	DEFINITION	BINARY	NUMERIC	DEFINITION	BINARY
00	Tape feed	000000	32	G	100000
01	Carriage return	000001	33	H	100001
02	Tab	000010	34	I	100010
03	Backspace	000011	35	J	100011
04	Color Shift	000100	36	K	100100
05	Upper Case	000101	37	L	100101
06	Lower Case	000110	38	M	100110
07	Line feed	000111	39	N	100111
08	*Stop Code	001000	40	O	101000
09		001001	41	P	101001
10		001010	42	Q	101010
11		001011	43	R	101011
12		001100	44	S	101100
13	Photo Reader	001101	45	T	101101
14		001110	46	U	101110
15	End of Block	001111	47	V	101111
16	0)	010000	48	W	110000
17	1)	010001	49	X	110001
18	2 "	010010	50	Y	110010
19	3 #	010011	51	Z	110011
20	4 Σ	010100	52	, \$	110100
21	5 Δ	010101	53	= :	110101
22	6 @	010110	54	[;	110110
23	7 , &	010111	55] %	110111
24	8	011000	56		111000
25	9 (011001	57		111001
26	A	011010	58	+ ?	111010
27	B	011011	59	- -	111011
28	C	011100	60	. .	111100
29	D	011101	61	Space	111101
30	E	011110	62	/ *	111110
31	F	011111	63	Code delete	111111



III

THE ASSEMBLY PROGRAM

CODING SHEET FOR THE ASSEMBLY PROGRAM

Figure 2 shows a sample of the coding form for the RPC 4000. It may be used equally well for machine language programs and symbolically coded programs.

ORDERS

Two types of orders are accepted by the Symbolic Assembler. They are machine orders and pseudo operations. Pseudo operations are instructions to the assembly program itself and are not reproduced as such into the program being processed. See the chapter on pseudo operations.

Machine orders may be entered on the coding sheet in either symbolic or numeric form. It makes no difference to ROAR which form is used. Provision has been made in the coding of ROAR for the user to add or change such symbolic codes as he desires -- LGP-30 programmers may well wish to use B for RAU, etc., if so the addition is not difficult.

Indexed Instructions

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

Examples

Location	Order	Data Address	Next Address	Comments
	O.2			Reset, Add Upper
	x.O.2			Indexed Reset, Add Upper
	R.A.U			Reset, Add Upper
	x.R.A.U			Indexed Reset, Add Upper

MEMORY SETUP FOR THE ASSEMBLED PROGRAM

In assembling an object program, the Assembler has in memory only one instruction at a time. Immediately as it is assembled, it is punched out in hex and printed in decimal. Since this is true, the entire memory of the computer is available to the assembled program.

At the beginning of assembly, all of "normal" memory, that is, tracks 000-122, is available for program. Normally, it will be necessary for the programmer to reserve various parts of memory for data storage, tables, etc.. It is also necessary for the programmer to provide one track for the bootstrap program which loads his program. This bootstrap area may be utilized later for data storage, etc.

ROAR provides two pseudo operation codes to reserve various areas of the drum. They may be used at any time, but normally will be among the first instructions assembled.

Reserve A Portion Of Memory: RES

This pseudo operation directs the Assembler to reserve the portion of memory between the FWA and the LWA. All locations between (and including) these addresses are made unavailable for symbol assignment. The FWA and LWA must be numeric addresses. (see the section on addresses which follows).

EXAMPLE

Location	Order	Data Address	Next Address	Comments
	RES	100	563	

This will make memory locations 100 through 563 unavailable.

Set Up a Region: REG

1. A region is a set of sequential memory locations which may be referred to by "regional addresses" (See Addresses).
2. Regions are distinguished by a one character code which is defined with the REG command.
3. The locations within a region are reserved in a manner similar to the RES order.

Location	Order	Data Address	Next Address	Comments
	REG B	00300	463	

will reserve region B from 300 through 463.

Make a Block Available: AVL

The function of this pseudo operation is the reverse of RES. It will serve to make the locations from FWA to LWA available for program assignment. The only time it would be meaningful to employ the AVL order at the beginning of a program is in the event that one or both of the double access tracks will not be used for double access. The leading (or trailing) head may be made available to general program use. Note should be taken that making available both heads of a double access track could lead to disastrous results. (See the section on Double Access, page 28).

Example:

Location	Order	Data Address	Next Address	Comments
	AVL	12300	12363	

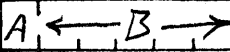
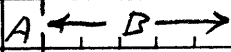
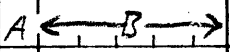
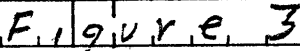
Makes available the block 12300 - 12363.

IV
ADDRESSES

Before discussing coding methods and techniques it is necessary that a complete discussion of addresses and addressing be presented.

An address in the final program will be in track and sector form. One of the major reasons for having an assembly routine, however, is to allow the programmer to be relieved of the task of assigning these machine language addresses, and to permit him to refer to operands and instructions in a convenient shorthand notation.

ROAR provides several methods for addressing memory locations in order that the programmer may allow the assembler to do as much or as little as he desires.

Location	Order	Data Address	Next Address	Comments
				
				

In the sample coding sheet (figure 3), note the columns marked Location, Data Address and Next Address. These are the columns that may contain addresses. These addresses consist of a left hand character (A) separated from the right hand five characters (B).

If a character is placed in the left hand box, it indicates a special address. Let us first consider addresses without the "special" character.

These normal or "non-special" addresses are of two types, symbolic and numeric.

Symbolic Addresses

Symbolic addresses are one through five characters in length. They contain at least one character that is not numeric. For this purpose any character that may be read into the computer is acceptable except specifically the digits zero through nine.

Symbolic addresses are assigned memory location equivalents by the assembly routine when they are first encountered and retain this equivalence throughout the program. Such equivalent locations are then made unavailable. Symbols may be preset to a particular value through the use of the pseudo operation codes EQR and EQV which are discussed later.

Some typical symbolic addresses might be:

Address
X
1A
A,1
RATE
B.E.G.I.N
D.B.L
R.E.D.B.L
1A,1
1.A.T.29
3/4
1/2.P.I
3.142

Numeric Addresses:

Numeric addresses refer to drum addresses and are in track and sector form with the exceptions mentioned below. It is not necessary to include leading zeroes in numeric addresses. The memory locations corresponding to numeric addresses are not made unavailable; if numeric addressing of data or instructions is to be used, the locations should be reserved in advance by a RES pseudo-op.

Special Sectors For Numeric Addresses:

In order to have the ability to select a particular word of the 8 word Lower, certain instructions were allowed to operate at a particular sector time: EXC (09), MPT (15). Normally we are not concerned with this feature and would like to have the Assembly Program merely assign an optimum sector.

At other times it is desirable to specify the action of these orders to operate on a particular word of the lengthed Lower.

Since we can completely describe the sectors of a track with the numbers 00-63, we have the remainder (64-99) available for these other purposes.

We have, then, assigned the sectors of the 90's as follows: the track will remain as assigned by the programmer:

- 90-97 The second character refers to a prime sector, modulo 8. ROAR will assign the next optimum occurrence of this sector as the sector address. As an example, if the next optimum sector were 34 and the given sector address, 93, the first optimum modulo eight 3 would be sector 35. See figure 1.
- 98 Assign the first optimum sector.
- 99 Assign the same sector as is contained in the location of the instruction.

Location	Order	Data Address	Next Address	Comments
	EXC	598		U→L, U→X, optimally
	PRD	199		Carriage return

The sector 99 is desirable since print interlocking is bypassed if and only if the sector of the D address of the print order is first optimum; normally this bypassing is not desirable. We can be certain that if the D sector is the same as the LOC sector, it will not be optimum.

Note that these special sectors apply only to D and N addresses; if a location were thus assigned it could never be referred to again.

Blank Addresses:

Addresses may be left blank subject to certain conditions.

If the D or N address (but not both) of an instruction is left blank, it will be assigned an available memory location (which is then made unavailable) and the location of the following instruction (or constant) will be given that address. If the D or N address of an instruction is left blank, the location of the following instruction must also be left blank.

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

ADDRESSES CONTAINING THE "SPECIAL" CHARACTER:

Regional Addresses:

A Regional Address consists of a region symbol (which must have been defined previously) and a five digit sequence number. The first memory location of a region is addressed as 00001. The sequence numbers are in true sequence,

not track and sector, so that 00105 would be the 105th word of the region.

<u>Example:</u>	Address	Comments
	T,0,0,0,0,1	2500
	T,0,0,0,6,4	2563
	T,0,0,0,6,5	2600
	T,0,0,2,0,0	would be assigned as
		2807 even though it has
		not been reserved in the
		region
	T,0,0,0,0,0	Assigned as 2463 -
		not reserved

An error stop will result if a regional address that refers to an undefined region is used.

Addressing the Eight Word Recirculating Track (127)

A special means of addressing the recirculating storage track has been devised. The eight storage locations have been assigned numbers corresponding to their prime modulo 8 sectors. They are numbered 0 through 7 or 1 through 8 depending upon the desire of the programmer; sector 0 and sector 8 in reality refer to the same sector.

Recirculating line sectors are addressed with the symbol RECRC followed by a prime sector. When the assembly program encounters a RECRC address, it assigns track 127 and the next optimum occurrence of the indicated sector. See also figure 1.

Example:

|_|_|_|_|_|_|_|_|
| R E C R C 7 |

Assume the next optimum sector to be sector 5. The next time a modulo 8 seven is available beginning with sector 5 is sector 7 itself.

RECRC5

Assume the next optimum sector is sector 30. Modulo 8 5's are 5,13,21,29,37,45,53,61, so the next available one is 37. The assembly routine therefore assigns 12737 for RECRC5 in this case. The programmer must keep track for himself of which RECRC sectors he is using. Subroutine write-ups should provide a list of the RECRC sectors they use.

Since the bootstrap program makes use of the recirculating track, any storage therein must be done by the program itself.

If a RECRC address is used as a location, the contents will be printed as a part of the program listing, but will not be reproduced into the hex program tape.

Using Double Access Tracks:

If the double access tracks are to be used as such, they should be left unavailable to other types of addressing. This is not difficult since it requires a pseudo-op to make them available.

In this normal condition they may be addressed by special double access symbolic addressing; for this purpose, the following numeric codes have been assigned:

Number	Track
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)

The double access address consists of three parts:

 D B # E] Symbol

1. The letters DB with the D in the special column.
2. #The number as assigned above.
3. A three character symbol, which characters may be any character capable of being read into the computer.

A double access address, when assigned, may be used only by the track to which assigned or by its partner track. Tracks 1 and 2 are partners and tracks 3 and 4 are partners.

To illustrate this, consider the following program:

Location	Order	Data Address	Next Address	Comments
				Variable in U @ 7
EX	STU	DB1VAR		Store temporarily
	SRL	1		Right 1 bit : 8
	STU	VAR		Variable @ 8
	RAU	DB2VAR	GO ON	Replace variable @ 7

The instruction in EX stores the variable quantity in the first optimum sector of the first double access track (16 word time) using the leading head.

The DB symbol VAR is stored in the symbol table.

The accumulator is shifted right by one bit and the variable at $q = 8$ is stored into a location symbolized VAR. This is the first assignment of VAR so it will be optimum.

The fourth instruction replaces the variable in the accumulator at $q = 7$, read by the trailing head of the first double access track (16 word time).

The assembly routine might have translated the instruction as follows:

EX*STU*DB1VAR**	00105 24 12307 00009	Variable in U at 7*
*SRL*1**	00009 12 00001 00217	Right 1: 8*
*STU*VAR**	00217 24 00019 00021	Variable at 8*
*RAU*DB2VAR*GO ON*	00021 02 12523 00125	Replace variable at 7*

Skip Address:

At times, for the sake of optimization, it is desired to place an instruction at a distance beyond the first optimum. An example of this might be where the D address is variable and may be set to pick up the contents of one of several consecutive sectors, yet we want the next instruction to be as optimum as possible.

To enable the programmer to do this, the Skip Address has been devised. For any D or N address that might otherwise have been blank, the letters SKIP, with the S in the special column, are used followed by a two digit number of sectors to be skipped beyond first optimum.

The following example will illustrate:

Locations 10520 through 10524 have been reserved for a set of parameters, one of which will have been chosen for use and its address placed in CODE. CODE has been defined as 00318.

Location	Order	Data Address	Next Address	Comments
C, O, D, E	R, A, U	1, 0, 5, 2, 0	S, K, I, P, 0, 5	Parameter → U
	M, P, Y		X etc.	

D-ADDR

The remaining special type of address is rather limited; it was designed for a specific condition. At times it is desired to turn the branch control off. This is done by coding a TBC instruction with the D and N addresses the same. In order for this to be done optimally, the N address must be location plus four sectors, but the D address would be optimized as location plus two sectors. If we were to use the same symbolic address for both, the D address would be optimized first as location plus two and any transfer to the N address would waste a drum revolution.

To cope with this, the D address is made a SKIP02 and the N address is given the code D-ADDR with D in the special column:

Location	Order	Data Address	Next Address	Comments
	T, B, C	S, K, I, P, 0, 2	D - A, D, D, R	

The code D-ADDR in the N address column indicates that the N address is to be made the same as the D address.

Example: Assume the location to be 01235.

*TBC*SKIP02*D-ADDR* 01235 23 00539 00539

Instead of trying to place the D address in sector 37, the SKIP02 address indicates that the search should begin with 2 greater than the optimum sector, or 39. Once defined, the N address is made the same as D through the D-ADDR address.

PSEUDO OPERATIONS

A discussion of several of the pseudo-operations has already been included: REG, RES and AVL in chapter III.

PRESETTING ADDRESSES

Symbolic addresses may be fixed at any time. Generally the initial address of the program will be set. There are two pseudo operations to be used for presetting addresses.

Equivalent, Reserve: EQR

Location	Order	Data Address	Next Address	Comments
	EQR	START	1200	

Upon reading the pseudo-op EQR, ROAR will do the following:

1. Place in the symbol table the symbolic address given in the D address.
2.
 - a. If the N address is numeric, assign that address to the symbol.
 - b. If the N address contains another type of address, assign the memory equivalent of that address to the symbol.
 - c. If the N address is numeric, the sector must be in the range 00 to 63. Blank, SKIP and D-ADDR are not legal addresses for this pseudo-op.
3. The memory equivalent as assigned in (2) is made unavailable (see Availability Table).

Equivalent, Don't Reserve: EQV

Location	Order	Data Address	Next Address	Comments
	EQV	TABLE	00001	

ROAR will treat the pseudo operation in the same manner as EQR with the exception of step 3. That is, it will not reserve any location.

The D address must be symbolic, the N address may contain any type of address, (except Blank, SKIP or D-ADDR). It is assumed that when this pseudo-op is used, the location will have been previously reserved or there is no need for reservation.

Set Up A Header Tag: TAG

Location	Order	Data Address	Next Address	Comments
	TAG	A		

When a program is being written in large parts or by several people, or makes use of symbolically coded subroutines, it becomes difficult to avoid the duplication of symbolic address codes between the sections. In order to assist the maintenance of order instead of chaos, it has been made possible to head the various portions of the program, each with its own symbol, subject to the following conventions:

1. Symbolic address of five characters in length are not headed.
2. To symbolic addresses of less than five characters is added the header tag and a bit to indicate that heading has taken place.
THIS TAG WILL NOT APPEAR ON THE OUTPUT LISTING.
3. A headed four character symbol will not be confused with a five character symbol even though the characters are the same, e.g. MASKS is not equivalent to ASKS headed by M.
4. Double access symbolic codes are not headed.
5. No heading will be done by the routine until a header has been set up.
6. Since five character symbolic addresses are not headed, they may be used to provide communication between portions of the program.
7. Subroutines written by the Royal McBee Programming Group will make use of five character symbols only for entry locations. These subroutines should be headed when used.

To set up the header, the pseudo-operation TAG is used. The D address is to contain the single character header tag that is to be set up and the location and N address are to be blank. A header, when established, will be used until replaced, or initialization for a new program is entered. Should it be necessary to remove the header altogether, use the pseudo-op TAG with a blank D address.

SETTING UP CONSTANTS

The assembly routine provides the programmer with four methods of setting up constants for his program.

1. Convert the constant to the form of an instruction and place on the coding sheet as such.

2. Pseudo operation HEX. The constant may be converted to its hex equivalent and entered on the following format:

Location	Order	Data Address	Next Address	Comments
any	HEX	1,2,3,4	5,6,7,8	

The location may be as for any instruction. For order, the pseudo-op HEX is used. The left four characters are placed in the four low order characters of the D address; the right four characters are placed in the four low order characters of the N address. Leading zeroes need not be entered.

3. Pseudo operation ALF

The five characters of the D address of the ALF pseudo operation are read in six bit mode and "stored" as indicated by the location. The N address is not used.

Location	Order	Data Address	Next Address	Comments
any	ALF	I, S, I, /		

4. Decimal Numbers: DEC

By using the pseudo-op DEC, the programmer may allow ROAR to convert his constants to a given "q" and punch them out in the same manner as HEX and ALF.

DEC is used as follows:

1. The location is the desired location, coded to indicate the proper address.
2. Use the order code DEC.
3. Place the q in the D address with its sign, if negative.
4. Start the number in the N address and allow it to extend out to the right. When punching the tape, place the stop code for the N address after the complete number. Place the sign at the left of the number (necessary only if negative). Include the decimal point at its proper place.

Location	Order	Data Address	Next Address	Comments
C1	DEC	5	-3.176	43

The limits are as follows:

1. For perfect accuracy, the significant figures of the number must total less than 2^{31} (2, 147, 483, 648) however, the only limit on digits is in 2.
2. The decimal point may be placed anywhere that describes 15 digits, but number, sign and decimal point must not total more than 16 characters. No decimal point is required for integers.
3. There is no limit on q except that conditions 1. and 2. must be fulfilled.

END OF PROGRAM: END

When the assembly program reads the pseudo operation code END it:

1. Computes the check sum for the hex tape and punches it.
2. Punches a transfer instruction in the hex tape, to the address given in the N address.
3. Stops computation. If started, it will read the next instruction. The END pseudo-op does not cause initialization for a new program.

Location	Order	Data Address	Next Address	Comments
	END		BEGIN	

The above sample will place on the Hex Tape the instruction to transfer to the drum equivalent of the symbol BEGIN.

THE AVAILABILITY TABLE

The Availability Table showing the availability status of each memory location may be punched out at any time through the use of the pseudo operation PAV for Punch Availability. If this is done at all, it is normally following the END code.

The punched availability table consists of the following:

1. As area of blank tape
2. The RAV code (see below) needed if the table is to be read at a future time.
3. The Availability Table itself consisting of the hex representation of the contents of the stored table interspersed with initial addresses of groups.
4. A check sum.

Punch the Availability Table

Location	Order	Data Address	Next Address	Comments
	PAV			

The Availability Table is set up in this manner so that, at a future time, additions can be made to a program without the necessity to reassemble the original program, but simply to restore the availability as it was when the original assembly was finished. If an availability table is to be read, it is normally the first tape read after loading the assembly program or using the pseudo operation NEW (See below).

CLEARING THE SYMBOL TABLE: CLS

Sometimes it is desirable to continue assembly with the same availability, but with a fresh symbol table. This is especially true when separate programs are being optimized to be in memory at the same time, or when major portions of a program have been coded by two people.

The pseudo-op CLS performs this operation. The symbol table is zeroed out, but everything else remains unchanged.

Location	Order	Data Address	Next Address	Comments
	CLS			

WAIT A SECOND: NIX

This pseudo operation has been included as an end of tape code, or for any other time it is desired to have the computer stop during assembly. No other action takes place. A depression of the start button causes the operation to continue.

Location	Order	Data Address	Next Address	Comments
	NIX			

BEGIN ASSEMBLING A NEW PROGRAM: NEW

Reading this pseudo operation causes the assembly program to transfer control to its initialization subroutines in order to make ready for a new program. This order is normally used shortly following the END order; it is unnecessary to use NEW immediately after loading the assembly program.

The following page contains a summary of the pseudo operations:

Location	Order	Data Address	Next Address	Comments
	NEW			

Location	Order	Data Address	Next Address	Comments
	R.E.G	S x,x,x,x,x	y,y,y,y,y	Reserve a region: S from xxxxx through yyyyyy
	E.Q.R	SYM	5021	Symbolize SYM with an address of 5021.
	E.Q.V	SYM	ADDR	Reserve 5021 Symbolize SYM with the same address assigned to ADDR
	R.E.S	500	520	Make unavailable the loca- tions from 500 through 520
	A.V.L	500	520	Make 500 through 520 available
	P.A.V			Punch availability table
	R.A.V			Read availability table (automatically on tape)
ANY	H.E.X	5555	6666	"Store" hex 55556666 in ANY
ANY	A.L.F	ERROR		Read ERROR in 6 Bit, place on hex tape assigned to ANY
	T.A.G	C		Tag all following symbolic addresses with C if they contain less than 5 characters
	N.I.X			Stop the computer before reading the next instruction
	E.N.D		BEGIN	End of program. Punch final check sum and set up a transfer to BEGIN
	CLS			Clear the symbol table.
	NEW			Initialize to process a new program
ANY	DEC	7-4.2361		store the number -4.2361 in ANY @ 7

VI

OPERATION CODES

The order codes along with the assembly routine's symbolic codes are given in the chapter on computer description, however, the following points are worthy of consideration.

SHIFTING

You will note that one instruction covers both directions of shift. If there is a one in the track portion of the D address, the accumulator will be shifted left by the amount contained in the sector of the D address. A zero in the track portion of the D address causes a right shift in the same manner.

EXCHANGE

The EXC (09) order may be used to execute any combination of its functions with one command. For example, let us assume it is desired to trade the contents of U and L, and copy U into X. This may be accomplished by forming the D track as the sum of the bits necessary to perform the required functions:

$$\begin{array}{r} U \rightarrow L = 1 \\ L \rightarrow U = 2 \\ U \rightarrow X = 4 \\ \hline 7 \end{array}$$

We would code track 7 for the EXC order.

If both the 16's and 32's bits are present in the EXC D track, the status of the Lower will be reversed, e.g., if it were eight words long it would become one word long.

SECTOR SEARCH

There are four instructions that the programmer should consider as to timing, two of which will normally be optimized and two of which will normally be unoptimized.

1. The two which are normally optimized: Because at times the EXC (09) and MPT (15) orders are to be used on the 8 word lower, they delay their action until the drum has turned to the sector of the D address. When used for normal operation, it is desirable for them to begin operation as soon as possible. Since they will normally be given a numeric track address, we suggest that the sector 98 (most optimum) be used for the D sector. This allows the best optimization of these instructions.

2. The two which are normally unoptimized: PRD (16) and PRU (17), Output interlocking may be bypassed by giving the print orders the first optimum sector in the D address. In normal operation this is not desirable, being for special cases only. Since these orders will normally be given a numeric D track address, we suggest the use of sector 99 for the D sector of print orders. Except for recognizing the first optimum sector, the print orders perform no complete sector search.

VII

PROGRAMMING USING ROAR

In preparing to use the Assembly Program, the following sequence will be most usual.

1. Determine the area where the bootstrap program may be placed. It requires one track plus the recirculating track. It is suggested that it be placed in an area later used for data storage.
2. Code the instructions to reserve regions and blocks for data storage, etc.
3. Set up equivalents for entry points if desired.
4. When assembling a large program, it is desirable to run the most often used routines first in order to give them the best optimization. This should be borne in mind when coding so that the linkage between various routines will be optimum.
5. Set up header tags if used.
6. Include the "Change and Transfer" program if there is room. This will enable the person doing checkout on the assembled program to get about on the computer.

VIII

PROGRAMMING EXAMPLES

The following three examples are included as a demonstration of some of the principles herein discussed. It is assumed that the reader has some coding experience. Example 3 is the simplest of the three and probably should be considered first by the more inexperienced readers.

EXAMPLE 1: LINEAR INTERPOLATION

Given: Table of X_i in Region X, 320 values in ascending order. X00001 is in Sector 00 of a track.

Table of Y_i in Region Y, each corresponding to an X_i

X in U, $X_0 < X < X_n$ of X table

Find: Y corresponding to X. Exit to FOUND with Y in the upper Accumulator.

Date 12/29/59

RPC 4000 CODING SHEET

Page 41 of

Problem Linear Interpolation

Section

Job No. Prog. No. Prep. By WJ Cr'd By

Location	Order	Data Address	Next Address	Comments
I,N,T,E,R	L,D,X	X0,0,0,0,1		Xo address → index register
	R,A,L	M,A,S,K	C,H,E,C,K	Mask for comparison
C,H,E,C,K	L,D,C	C,O,U,N,T		Setup repeat count
	X,C,M,G	0		Table lookup on Greater or equal
	T,B,C	G,O,T		→ Found X
	x,L,D,X	1,0,0		NOT found, try next track
	C,X,E	X0,0,3,2,1		Have we exceeded the table?
	T,B,C	E,R,R,O,R	C,H,E,C,K	→ Table exceeded
M,A,S,K	x,3,1	1,2,7,6,3	1,2,7,6,3	All 1's
C,O,U,N,T	0	0	6,3,0,0	Count of 63
E,R,R,O,R	H,L,T	0		stop with X in accumulator
	C,L,U	D,U,M,P	F,O,U,N,D	Exit with 0 Upper
G,O,T	E,X,C	9,9,8		X → U, U → L
	S,D,A	H,O,L,D		Store found track
	S,R,L	1,1,3		Move sector +1 to D of U
	S,B,U	D,S,E,C,1		
	E,X,T	R,I,G,H,T		Keep D sector of U, shifted X value
	A,D,V	H,O,L,D		
	S,A,U	X,2		Address of X ₁
	S,B,U	D,S,E,C,1		
	S,A,U	X,5,1		Address of X ₅
	S,A,U	X,5,2		

Problem Linear Interpolation - 2 Section _____

Job No. _____ Prog. No. _____ Prep. By JJ Ck'd By _____

Location	Order	Data Address	Next Address	Comments
	S,B,U	X,0,0,1		
	A,D,U	Y,0,0,1		
	S,A,U	Y,S,1		Address of Y _s
	S,A,U	Y,S,2		
	A,D,U	D,S,E,C,1		
	S,A,U	Y,L		Address of Y _L
	S,R,L	1,1,9	X,S,1	X → Upper
X,S,1	S,B,U	99		
	S,T,U,R,E,C,R,C,1		X,L	X - X _s
X,L	R,A,U	99	X,S,2	
X,S,2	S,B,U	99		X _L - X _s
	S,T,U,R,E,C,R,C,5		Y,L	
Y,L	R,A,U	99	Y,S,1	
Y,S,1	S,B,U	99		Y _L - Y _s
	M,P,Y,R,E,C,R,C,1			$(Y_L - Y_s)(X - X_s)$
	D,I,V,R,E,C,R,C,5		Y,S,2	$(Y_L - Y_s)(X - X_s) / (X_L - X_s)$
Y,S,2	A,D,U	99	FOUND	Y
D,S,E,C,1	0	1	0	
H,O,L,D	0	0	0	
R,I,G,H,T	X,0,0	63	12,7,6,3	
X,0,0,1	0	X,0,0,0,1	0	
Y,0,0,1	0	Y,0,0,0,1	0	

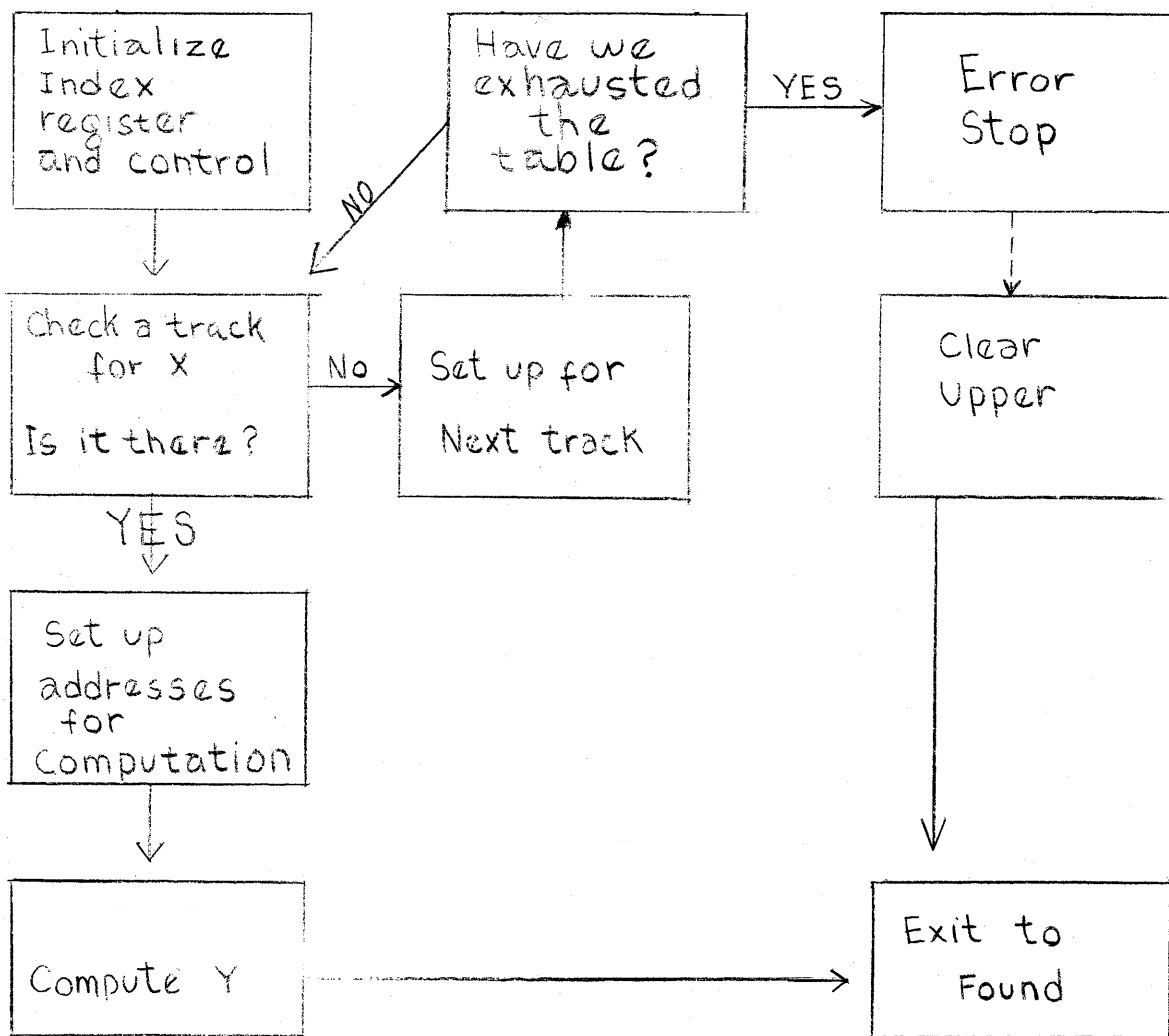
The Method Used Is:

1. Find the next larger X_i from the table X_L
2. Find the surrounding X and Y values from the table.
3. Interpolate using the formula $Y = Y_s + (X - X_s) \frac{(Y_L - Y_s)}{(X_L - X_s)}$

where subscript S indicates smaller and L larger.

4. Exit

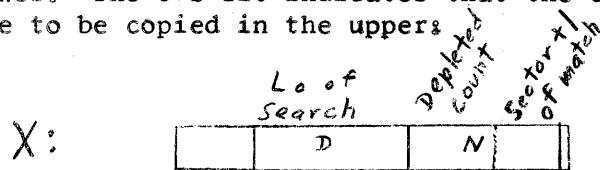
A Flow Chart Might Look Like:



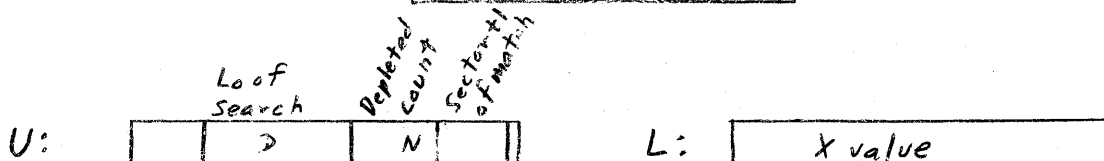
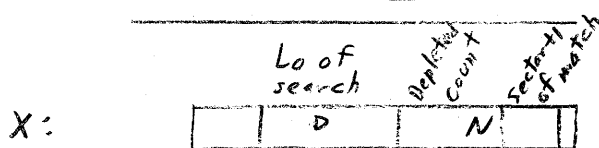
Programming Notes

1. The index register is initialized to the first location of the table of X's which is in sector 0 of a track specified previously by a REG pseudo-op.
2. The N address of the instruction in INTER has been left blank; the location has been left blank on the next instruction.
3. A mask for comparison is placed in the lower, leaving X in the upper.
4. The N address is symbolized "CHECK" because of a later reference to the same address.
5. A count of 63 is placed in the count portion of the X register and the next instruction will be repeated as specified by the command in CHECK.
6. A table look up is executed by the CMG order and checked by the TBC order. If X has been found, control is transferred to the instruction in GOT, if not, a setup for the next track occurs in the indexed LDX order.
7. For safety, a check for the end of the X table occurs in the CXE and TBC orders; If the table has been exceeded, an error stop is executed; if not we go back to CHECK another track.
8. When X is found we transfer to GOT where an EXC (Exchange) order is executed. The track is nine -- we have bits one and eight. The 1's bit indicates that the contents of the upper are to be copied in the lower. The 8's bit indicates that the contents of the X register are to be copied in the uppers.

Before:



After:



so that we can retain X without using temporary storage while performing address modification.

9. The SAU order after GOT places Lo of the found track in temporary storage.
10. The found sector plus one is shifted, trimmed, reduced by one and the address of X_L is compiled and stored.
11. From the address of X_L are found the addresses of X_S , Y_L and Y_S which are stored appropriately.
12. X is restored, Y is computed and control is transferred to FOUND.
13. Note the selection of RECRCl locations. There is no assurance that the first reference to RECRCl will be optimum; it might be as far as 7 word times out. However, once we have optimized to RECRCl every subsequent reference to recirculating locations is optimum through the choice of Sectors 1 and 5 which are 4 word times apart.

EXAMPLE 2: POLYNOMIAL EVALUATION

Fixed coefficients with variable X.

Determine f(X).

The routine is entered with X in the Upper @1; the instruction to be executed in order to exit is in L. The table of a_i @1 is to be in 0700 - 0706 stored a_0 through a_6 . In order to exit, the computer is to execute the instruction that was found in the Lower upon entry, leaving f(x) in the Upper @1.

Method

$$f(x) = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 + a_5X^5 + a_6X^6$$

The values are such that $f(X) < 2$ for all $X < 1$.

$$f(1) = 2$$

∴ for this program: $0 \leq X < 1$

To accomplish the evaluation, we may rewrite the equation in a form that provides an easy method of solution:

$$f(X) = a_0 + X(a_1 + X(a_2 + X(a_3 + X(a_4 + X(a_5 + X(a_6))))))$$

The evaluation will demonstrate the use of the index register, however in the reverse fashion to normal use.

Location	Order	Data Address	Next Address	Comments
	REGA	00700	706	A ₀ - A ₆
	EQR	ADD	62	
	EQR	EVALU	100	
EVALU	LDX	0	STOR	Initialize X register
ADD	xADUA	00007		Add a _i
	CXE	12758		check X for a ₀ addition
	TBC	END		Did we add a ₀ ? Yes → End
	MPY	X		
	SRL	101		
	ADL	ROUND		Add .5 to low order
	TBC	ADD R	NEXT	
NEXT	xLDX	12763	ADD	
ADD R	ADU	1AT31	NEXT	.
STOR	CLL	END		
	CLU	X	ADD	
ROUND	HEX	8000	0	-1@0
1AT31	X00	0	0	1@31

Programming Notes

1. A region is assigned for the a_i . Values and addresses are assigned for critical locations.
2. EVALU initializes the index register. Note that the initialization is placed on the coding sheet following the main loop. This allows END and X to be optimized from major use instead of initialization which is incidental.
3. The major loop of 7 instructions develops $f(X)$ in the upper accumulator.
 - a. The test-out occurs after adding an a_i .
 - b. The summation is multiplied by X and rounded.
 - c. The index register is incremented by 12763, which results in decrementation by 1.
 - d. Exit is to location END which is filled in the initialization.

EXAMPLE 3:

Evaluate the function: $f(X) = \frac{aX + X^2}{C}$

Given X in X @5

a in a @5

C in C @5

Store f(X) in f(X) @5

Begin at BEGIN

The exit is in the index register

Problem Example 3 $f(x) = \frac{x^2 + ax}{c}$

Section _____

Job No. _____ Prog. No. _____ Prep. By JJ Ck'd By _____

Location	Order	Data Address	Next Address	Comments
BEGIN	RAU	X		X
	MPY	X		X ²
	STU DB1 X SQ			X ² → C(XSQ)
	RAU	X		
	MPY	A		AX
	ADU DB2 X SQ			AX + X ²
	DVU	C		AX + X ² / C = f(X)
	STU	f(X)		
	X CXE	0		Turn BC on
	X TBC	0	ERROR	Exit to address in X
				Note:
				This could have been done more efficiently with the method of example 2, but that wouldn't demonstrate double access, would it?

IX

INPUT TAPE PREPARATION

The format for punching tapes from coding sheets is fixed and rigid for all instructions and pseudo operations. This was done purposely in an effort to reduce the possibility of tape-punch errors, and so that the puncher has relatively few rules to remember.

TAPE PUNCHING FROM CODING SHEETS IS AS FOLLOWS:

1. There is required a stop code following each of the following fields, no matter if it is blank or filled: Location, Order, Data Address, Next Address, Comments.
2. Following each instruction, a carriage return must be on tape.
3. Leading zeroes need not be punched, except that separating zeroes must be punched.
 - a. Regional addresses contain the region symbol followed by five numeric characters.
 - b. Spaces are not accepted in lieu of zeroes.
 - c. If a numeric order is to be indexed, the X must appear to the left of two (not one) characters, otherwise no leading zeroes need be punched, except that a blank order is illegal. There is a difference between a blank address and a zero address; a single punched zero is sufficient for distinction.

EXAMPLE: Punch the Following

Problem Sample Coding for Tape Punching Section _____

Job No. _____ Prog. No. _____ Prep. By _____ Ck'd By _____

Location	Order	Data Address	Next Address	Comments
	.RES	5.00	1.000	
	.REG	A.00.0.00	1.63	
	.EQR	BEGIN	15.00	
BEGIN	.RAVA	A.00.0.01		
	.TMI		NO	
	.RAU	AB	YES	
	.NO	.SBU	AC	YES
	.END			Stop for availability punch
	.PAV			Punch availability
YES	.CLU	CODE		
	.LDC	COUNT		
	.ADVA	A.00.0.01		
	.LDC	COUNT		
	.ADVA	A.00.0.65		
	.STU	TOTAL	GO. ON	
COUNT	.0	.0	6.30.0	
	.END			

The Tape Produced Must List Like This:

*RES*500*1000**
*REG*A00000*163**
*EQR*BEGIN*1500**
BEGIN*RAU*A00001***
*IMI**NO**
*RAU*AB*YES**
NO*SBU*AC*YES**
*END***Stop for availability punch*
*PAV***Punch Availability*
YES*CLU*CODE***
*LDC*COUNT***
*ADU*A00001***
*LDC*COUNT***
*ADU*A00065***
*STU*TOTAL*GO ON**
COUNT*O*O*6300**
*END****

OUTPUT FORMAT

The Assembly Program makes use of both punch and typewriter output.

The punch delivers:

1. A bootstrap program to be stored as indicated by the operator.
2. The hex tape of the program with check sums at specified intervals.
3. End and transfer codes.

The availability table is output on the punch.

The typewriter output consists of a listing of the input instruction, numeric assembled instruction and the punched comments.

The typewriter output may be bypassed, if desired as in the case of a subroutine, by the depression of a sense switch.

XI

OPERATOR'S INSTRUCTIONS

1. Load ROAR by means of the bootstrap included on the tape. The computer will stop after loading.

The method of reading the bootstrap is as follows:

- a. Place the tape in the reader
 - b. Select reader input
 - c. Depress One Operation
 - d. Depress Execute Lower
 - e. Depress Set Input
 - f. Depress Start Read on the reader
 - g. Raise Execute Lower
 - h. Depress Set Input
 - i. Raise One Operation
 - j. Depress Start Read on the reader
2. Place the tape of the program to be assembled in the reader and depress the start button. The computer will soon stop to be given the track for the Bootstrap Program. If no bootstrap is desired, sense switch 8 must be depressed.
 3. All further communication with the program should be through the use of pseudo operations. These may be entered manually by selecting typewriter entry after an END order on tape, or by stopping on tape, or by stopping the computer while reading a carriage return.
 4. If ROAR detects an error, it will print out an indication as to the type and stop. If started, assembly will continue from the point where location is read. The safest place to restart after correcting an error is at the nearest previous instruction which contains a non-blank location.
 5. TAB STOPS
Two typewriter carriage tab stops are required. All others should be cleared. Place the tab stops in relation to the left margin at increments of 27 and 50.
 6. For your convenience, the following options have been placed on console sense switches. The normal condition is represented by the UP position in each case.

Sense Switch

Use In Depressed State

2

Do not list the input on the typewriter

- 4 Use the photo reader for input instead
 of the reader
- 8 Do not output a bootstrap
- 16 Use the output device selected by print
 code 106 for the decimal output instead
 of the typewriter (may be changed to any
 other selection code)
- 32 Bypass the decimal output