22
8,23-74
25 apts NTS

BDX-613-1053 (Rev.)

24535

READ-ONLY MEMORY IMPLEMENTATION STUDY

PDO 6984754, Final Report

J. J. Zimmerman, Project Leader

Project Team:
J. D. Anderson
M. J. Devaney
R. L. Johnson
M. E. Martin

August 1974

Include
doc in
packet
in back

MASTER

Prepared for the
United States Atomic Energy Commission
Under Contract Number AT(29-1)-613 USAEC

Kansas City
Division

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

Available from the National Technical
Information Service, U. S. Department of
Commerce, 5285 Port Royal Road, Springfield,
Virginia 22151.

Price: Paper Copy \$4.00
Microfiche \$1.45

READ-ONLY MEMORY IMPLEMENTATION STUDY

August 1974

**Project Leader:
J. J. Zimmerman
Department 864**

**Project Team:
J. D. Anderson
M. J. Devaney
R. L. Johnson
M. E. Martin**

**PDO 6984754
Final Report**

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Technical Communications

**Kansas City
Division**

MASTER

READ-ONLY MEMORY IMPLEMENTATION STUDY

BDX-613-1053 (Rev.)
August 1974

Prepared by J. D. Anderson, D/864, under PDO 6984754

Electrically-alterable read-only memory (ROM) units have been incorporated into production test equipment. Production quantity costs of ROM's enable practical use of these large scale integrated (LSI) circuits in design of small-quantity equipment. One production tester is fully automated with six 4000-bit memories and twenty-two 256-bit memories as the control feature.

THE BENDIX CORPORATION

KANSAS CITY DIVISION

KANSAS CITY, MISSOURI

A prime contractor for the
Atomic Energy Commission
Contract Number AT(29-1)-613 USAEC

CONTENTS

Section	Page
SUMMARY	6
DISCUSSION.	7
SCOPE AND PURPOSE	7
PRIOR WORK.	7
ACTIVITY.	7
<u>Initial Investigations</u>	7
<u>Incorporation of ROM's</u>	8
<u>Coupler/Controller Functions</u>	8
<u>ROM Program Card</u>	15
<u>Programming Read-Only Memories</u>	24
ACCOMPLISHMENTS	30
FUTURE WORK	35
APPENDICES	
A. ROM SUPPORT PROGRAM	39
B. HP2570A ASSEMBLER	48
C. ROM CARD PROGRAMMING	53
DISTRIBUTION	56

ILLUSTRATIONS

Figure		Page
1	Structures of Intersil Memories (Polaroid)	9
2	Exterior of HP2570A Coupler/Controller (P84378). . .	10
3	Interior of HP2570A Coupler/Controller (P84375). . .	11
4	Block Diagram of HP2570A Coupler/Controller Interfaced With Several Instruments	12
5	Diode Pinboard Programmer Card (P84376).	13
6	Schematic Example of Programming	14
7	Read-Only Memory (ROM) Card (P84399)	16
8	Schematic of a ROM Programming Subroutine.	22
9	ROM Card Monitor (P84707).	24
10	INTEL 1702 Programmer (P84382)	25
11	Programmer and Teletype Interfaced to Coupler/ Controller (P84379)	26
12	Spectrum Dynamics 550 ROM Programmer (P84380). . . .	28
13	Typical Printout, HP2570A Program.	29
14	Typical Input Card Deck, Mode = ϕ	30
15	Truth Tables for ROM's of Figure 14.	31
16	Circuit for an Intersil 32-Word ROM.	32
17	Typical Input Source Deck (Mode = 1)	33
18	Output Truth Table for Input Source Deck	34
19	Control Section of TE5017 Tester (P84377).	35
20	TE5017 Tester (P84381)	36
21	PT4007 ROM-Controlled Tester (P84708).	37
22	ROM Control Chassis of PT4007 (P84709)	38
C-1	Block Diagram of ROM Card Programming.	55

TABLES

Number		Page
1	Typical Program on HP2570A Control Card	15
2	Typical ROM Program Card Information	17
3	ROM Starting Address Pointers	19
4	ROM Branch Statement Pointers	20
5	Typical Subroutine Loop	22

SUMMARY

Techniques for incorporating read-only memory (ROM) units into production test equipment have been developed in several areas.

- Logic circuits using ROM's replace conventional logic modules.
- The use of ROM's has reduced package count and the number of wire interconnects and has streamlined the logic definition.
- A computer program automates the programming definition of ROM's.
- A hardware programmer unit has been designed to accept the definition of the ROM from a computer-punched paper tape to program the ROM automatically.
- A ROM control card has been designed and fabricated for the Hewlett Packard 2570A Coupler/Controller to give it an instruction capability of 4096 eight-bit words, as opposed to the 60 supplied with the unit.
- A computer assembler program has been written for the ROM control card to simplify the programming definition of the ROM's.
- The TE5017 tester uses the HP2570A coupler/controller and ROM card for its automatic testing sequence.
- The PT4007 uses ROM's to fully automate its testing sequence.
- ROM's are now emerging in the design of many different pieces of test equipment.

DISCUSSION

SCOPE AND PURPOSE

This project covers the development of techniques to incorporate ROM's into production test equipment. An ROM is a digital integrated circuit which generates a set of preprogrammed logic levels at its output terminals in response to all possible combinations of logic levels at its input terminals. It is essentially an electronic truth table array with address decoding (Figure 1). Electrically alterable ROM's are used in a variety of ways in test equipment design, thus giving the designer availability of large scale integrated (LSI) circuits at production-quantity costs to fabricate small-quantity equipment, with the advantage of complete flexibility in defining the logic function.

Since the introduction of ROM's to the market, their cost (256-bit ROM) has decreased from \$38.40 (when they were first used in 1971) to \$13.50 by 1974. The ROM makes available to the designer of unique equipment, or of equipment where production is in limited quantities, a mass-produced LSI integrated circuit of general configuration which can be tailored to his particular application. In addition, the designer is afforded the low cost and short lead time inherent in competitively marketed mass-produced LSI devices.

Also during this project new developments in memory-erasing technology in ROM's were introduced, making their application still more appealing. This project gives Bendix the capability to program ROMs automatically, and affords Bendix engineers the facilities for incorporating ROM designs with minimal difficulty.

PRIOR WORK

No formal reports have been published on this project; only internal memorandums have been issued.

ACTIVITY

Initial Investigations

Approximately 30 technical articles describing ROM applications in logic systems were collected, and a survey of available electrically-alterable ROM's was made. These ROM's are manufactured in bit capacities ranging from 256 to over 4000. At the beginning of this project, a programmed ROM could not be corrected if a mistake occurred in programming. If a 2000-bit ROM was being programmed and a mistake occurred on the last bit, much time

and the cost of the ROM were lost. During the project new developments in memory-erasing technology were introduced to make ROM's reprogrammable. The Intersil 256-bit memory and the Intel 2048-bit memory were chosen for use in this investigation. The larger memory is erasable by use of high-intensity ultra-violet light, and the unit is reprogrammable. The structure of the two memories is shown in Figure 1.

The smaller memory was programmed with an Intersil Programmer. No facilities were available to program the larger memory until a manual programmer printed circuit card was purchased from Intel. Components were selected and added to the card to provide programming capability. Manually programming 256 eight-bit words proved tedious. It was found to be much easier and faster to automatically program the memory from punched paper tape.

Incorporation of ROM's

ROM's were incorporated in test sequencing controllers and more specifically in the control card of the Hewlett Packard Model 2570A Coupler/Controller (Figures 2 and 3). The coupler/controller is a frame box containing a control card and a common bus connected to eight plug-in slots for circuit interface cards. This instrument provides a common code so that different instruments connected to it can communicate with each other even though their respective codes are different. It allows a binary-coded decimal (BCD) input unit to communicate with a unit which has a straight binary output. It performs this task by changing all codes to ASCII (American Standard Code for Information Exchange). An interface card for each instrument converts that instrument's particular code to ASCII and this card is then plugged into the controller. If one instrument (such as a digital voltmeter) has inputs and outputs to the controller, two interface cards are required: one for converting the output code to ASCII and one for converting ASCII to the input code.

Coupler/Controller Functions

The block diagram of the controller is shown in Figure 4. The control card controls the flow of information to and from the ASCII bus and the interface cards, and converts all input and output codes to the common ASCII code. The control logic section of the controller can be programmed to make the coupler controller perform a series of instructions. The programming language is ASCII and the actual program is entered into the control logic section of the coupler/controller by any of three devices; ASCII tapes inserted by a teletype tape reader, a teletype keyboard, or a diode pin-board programmer as shown in Figure 5. The diode pinboard programming is described in detail in a subsequent section of this report. A program instruction, as entered from a teletype keyboard or tape reader, would consist of three ASCII characters in the format: @ a c, where

Figure 1. Structures of ROM Memories

- @ - informs the HP2570A that this is an instruction and causes the next two characters to be interpreted as an address and command,
- a - is an ASCII character in the numerical range ϕ -7 and denotes the address of the slot being commanded, and

c - is an ASCII character in the alphabetical range A-O which represents a specific command to be performed by the device.

As an example of programming, assume a digital voltmeter (DVM) interfaced into the HP2570A through Slot 4 and a printer interfaced into Slot 1 (Figure 6). Further assume that the digital voltmeter interface card is coded so that an E causes it to tell the DVM to take a reading and that an O causes it to output that reading to the HP2570A bus. Further assume that the printer interface card is coded so an I causes it to output data from the bus to the printer and causes the printer to print that data. To program the system to take a reading and then print it, a teletype tape containing the following would be required: @1I @4E @4O. If a teletype is interfaced into the HP2570A system and the above tape is put on the tape reader, the following events would occur.

Figure 2. Exterior of HP2570A Coupler/Controller

Figure 3. Interior of HP2570A Coupler/Controller

Figure 4. Block Diagram of HP2570A Coupler/Controller Interfaced With Several Instruments

Figure 5. Diode Pinboard Programmer Card

- Upon receiving the @1I command, the printer interfaced to Slot 1 will begin looking for data on the bus.
- Upon receiving the @4E command the digital voltmeter interfaced to Slot 1 will take a reading. A special bus line triggered by @ notes that the @4E command is not data and inhibits the printer.
- Upon receiving the @4O command, the DVM interface card interrupts the system control and places the DVM data on the bus in ASCII. The printer interface card is inhibited from allowing the printer to print @4O, but recognizes the digital voltmeter data which follows and causes it to be printed.

Although it requires three separate ASCII characters from a teletype to program each instruction, the same instruction can be programmed within the HP2570A with one 9-bit word. The following format is used:

Figure 6. Schematic Example of Programming

Flag	Parity	Address	Command
bF	b8	b7b6b5	b4b3b2b1

where

bF is a flag bit, (If the flag bit = 1, then the HP2570A interprets the word as if @ had been presented. If the flag bit = 0 the word is treated by the HP2570A as if it were an ASCII character.)

b8 is a parity bit, (This bit is used only if required to complete odd or even parity.)

b7,b6,b5 are address bits, (Numerals 0 through 7 are coded in binary to determine the address of the slot receiving the instruction.) and

b4,b3,b2,b1 are command bits. (Letters A through O are coded the same as the four least significant bits of the ASCII letters A through O. This determines the specific command for the slot receiving the instruction.)

In the HP2570A the nine bits of each word are coded by placing diode pins in a pinboard grid. The HP2570A Control card contains a grid to program 15 words. By plugging in a diode pin a 1 is programmed whereas a 0 is programmed by the lack of a pin. To code the program of the above example on the HP2570A Control card, the placement of diode pins would occur as shown in Table 1.

Table 1. Typical Program on HP2570A Control Card

Instruction	Flag (bF)	Parity (b8)	Address b7b6b5	Command b4b3b2b1
@1I	1	0	0 0 1	1 0 0 1
@4E	1	0	1 0 0	0 1 0 1
@40	1	0	1 0 0	1 1 1 1
(additional)				
@6J	1	1	1 1 0	1 0 1 0
@3B	1	0	0 1 1	0 0 1 0
ASCII C	0	1	1 0 0	0 0 1 1
ASCII 6	0	0	0 1 1	0 1 1 0

ASCII character codes (with four least significant binary bits in parentheses) are: 302(0 010)=B, 303(0 011)=C, 305(0 101)=E, 311(1 001)=I, 312(1 010)=J, 317(1 111)=O. Parity bit is shown for even parity (i.e., the sum of all 1 bits, including the parity bit, will give an even number).

The programming capacity of the coupler/controller, with diode pinboard programming of the control card plus one additional diode programming card, is $15 + 45 = 60$ instructions. It seemed feasible to replace the diode programming card with an ROM card and thus increase the instruction capability of the coupler/controller.

ROM Program Card

A ROM card was designed to be compatible with the diode pinboard programmer and some additional features were incorporated (Figure 7). The card has Intel 1702 ROM's which are erasable and reprogrammable. A summary of the ROM card features are:

- The ROM Program Card can contain from 256 eight-bit words of memory up to more than 2000 eight-bit words of memory. The ROM's are erasable and reprogrammable. The user buys only the amount of memory he requires in increments of 256 words (1 ROM contains 256 words). If the user requires more than a 2000-word memory, he can expand up to 4000 by adding the additional memories on a remote card connected to the ROM Program Card. The ROM Program Card contains addressing for the total 4000 words of memory.

Figure 7. Read-Only Memory (ROM) Card

- Random access of memory is possible with unlimited branching capability within the full 4000-word capacity of the ROM Program Card. In addition, the ROM Program Card has conditional branch capability to all but one ROM (256 words) of the on-card 2000 words of memory and one ROM of the off-card 2000 words of memory. Each branch statement, conditional or unconditional, requires two memory words. The first word indicates that a branch will occur, whether it is conditional or unconditional, and to what ROM the branch goes. The second word indicates the actual address on the chosen ROM.
- Conditional or unconditional subroutine is possible. To perform a subroutine a total of four word locations are required; a special code word to prepare for subroutine, then a 2-word branch instruction, and finally another special code word to cause a subroutine return. An unlimited amount

of branch words may occur between the original subroutine branch words and the subroutine return word.

- The ROM Control card is wired so that an address switch register and LED readouts for ROM address, ROM content, and HP2570A bus content may be wired directly to the card through a PC card edge connector.

Operation of the Read Only Memory Program Card

As explained earlier, pinboard programming requires the use of a 9-bit word. The ROM program card uses an 8-bit word to achieve the same memory capability as the pinboard programmer. It omits the parity bit in memory, but retains parity capability by using circuitry which monitors the HP2570A bus and inserts the proper parity bit as required. In each ROM address location the bit format is standardized.

- bF (flag bit) is the most significant bit of ROM word.
- b8 (parity bit) is not programmed on the ROM word.
- b7,b6,b5 (address bits) are the 2nd, 3rd, and 4th most significant bits of the ROM word respectively.
- b4,b3,b2,b1 (command bits) are the 4th, 3rd, 2nd, and 1st least significant bits of the ROM word respectively.

The example instructions and ASCII words of Table 1 would be programmed on a ROM as shown in Table 2.

Table 2. Typical ROM Program Card Information

Instruction	Flag A8	ROM Word Address A7A6A5	Command A4A3A2A1	Octal Code
@1I	1	0 0 1	1 0 0 1	231
@4E	1	1 0 0	0 1 0 1	305
@4O	1	1 0 0	1 1 1 1	317
@6J	1	1 1 0	1 0 1 0	352
@3B	1	0 1 1	0 0 1 0	262
ASCII C	0	1 0 0	0 0 1 1	203
ASCII 6	0	0 1 1	0 1 1 0	066

To program the digital voltmeter and printer as in the previous HP2570A example, but with the ROM Program Card rather than the teletype, the following octal codes would be programmed into three consecutive address locations:

<u>Instruction</u>	<u>Octal Code</u>
@11	231
@4E	305
@40	317

If the ROM Program Card were given control of the system and then stepped through the addresses of the three octal codes listed above, the digital voltmeter would take a reading and the printer would print that reading just as if programmed by a teletype tape.

Control of the system can be given to the ROM Program Card with a two-word program sequence on the pinboard programmer of the HP2570A Control Card, or (if the teletype has control) by typing the two equivalent commands of the following format on the teletype.

@ab @aC

where

a is the address of the HP2570A slot to which the ROM Program Card is mated,

b is a pointer to the location of the read-only memory address, (Table 3 contains a list of these pointer addresses.) and

C is an ASCII C or equivalent bit pattern (0011) on the pinboard programmer.

For example, an @2A @2C sequence would give control to the ROM Program Card at Address 010, whereas an @20 @2C would give control at Address 170.

Once the ROM Program Card has control of the system it sequentially steps through the program addresses in increments of one. The memory is broken down into pages, with each page consisting of one ROM. After the last address of one page is completed, the program steps to the first address of the next page. Conditional branch, unconditional branch, and subroutine instructions give capability to jump forward or backward to different pages

Table 3. ROM Starting Address Pointers

Pointer	Address	Pointer	Address
ϕ	000	H	100
A	010	I	110
B	020	J	120
C	030	K	130
D	040	L	140
E	050	M	150
F	060	N	160
G	070	O	170

and address locations. A branch statement, conditional or unconditional, consists of two successive words in memory of the following format.

@ * a -- sets up branch

X X X -- branch location

where

@ is the presence of a ONE flag bit to signify a machine instruction.

* is the octal self-address of the HP2570A slot into which the ROM Program card is inserted.

a is the 4-bit code of an ASCII letter in the range A-O which points the page number to which the branch will occur, and also signifies whether the branch is conditional or unconditional (Table 4).

XXX is a 3-digit octal address within the previously denoted page to which branch control will be given.

An example of an unconditional branch would be the following two octal words in the memory, assuming the ROM Program Card is inserted in Slot 7 of the HP2570A.

Machine Word	Equivalent Memory Word
@7E	365
141	141

Table 4. ROM Branch Statement Pointers

Pointer		Branch Type	Page		Address Range	
Letter	Binary		Low Mem	High Mem*	Low Mem	High Mem*
φ	0000	U	0	8	000-377	4000-4377
A	0001	U	1	9	400-777	4400-4777
B	0010	U	2	10	1000-1377	5000-5377
C	0011	U	3	11	1400-1777	5400-5777
D	0100	U	4	12	2000-2377	6000-6377
E	0101	U	5	13	2400-2777	6400-6777
F	0110	U	6	14	3000-3377	7000-7377
G	0111	U	7	15	3400-3777	7400-7777
H	1000	C	0	8	000-377	4000-4377
I	1001	C	1	9	400-777	4400-4777
J	1010	C	2	10	1000-1377	5000-5377
K	1011	C	3	11	1400-1777	5400-5777
L	1100	C	4	12	2000-2377	6000-6377
M	1101	C	5	13	2400-2777	6400-6777
N	1110	C	6	14	3000-3377	7000-7377

U = unconditional
 C = conditional
 *High Memory is the off-card 2000 words of memory.

When encountered while stepping through the memory sequence, these two consecutive statements would tell the ROM Program Card to jump unconditionally to Address Location 141 on Page 5 of the ROM Program Card. The actual address location would be 2400 + 141 = 2541.

A conditional branch to the same location would be programmed with these words.

Machine Word	Equivalent Memory Word
@7M	375
141	141

In this case these two consecutive statements would cause a jump to Address Location 141 on Page 5 (Location 2541) only if the condition input line had been low for at least one memory cycle (5 ms) prior to encountering the @7M (375) statement. If this condition had been satisfied, then the branch would occur and a

condition flip-flop would be reset so the condition input must again go low to satisfy any later conditional branch statements. If the conditions input had never gone low for the minimum of one memory cycle, then the two condition branch statements would be ignored.

To cause a subroutine to occur, an octal 006 is programmed into the memory address prior to a branch (conditional or unconditional) statement. This octal 006 statement clears a memory buffer and allows it to remember the address location of the second word of the branch statement, when the branch occurs. Later, when an octal 007 is encountered while stepping through the program, control will automatically be returned to the next address location after that which had been remembered. Once an 006 and then a two-word branch instruction has occurred, any number of branch statements can occur prior to using an 007 to get a subroutine return. Only one subroutine sequence may be in effect at one time.

To give an example of using the subroutine in programming the digital voltmeter and printer sequence of previous examples may be used. In this case assume that voltage printouts are to be made only when the output voltage of a unit under test is greater than 20 volts. To accomplish this, assume that a voltage comparator is so wired that it causes the ROM Program Card condition line to go low when 20 volts is exceeded, and keeps the condition line high when voltage is 20 volts or less. With the ROM Program Card plugged into Slot 7, the digital voltmeter to Slot 4, the printer to Slot 1, and a teletype in Slot 6, the ROM Program Card memory program could be as shown in Figure 8.

To initiate action of the above program (Table 5), a teletype or pinboard program sequence of @7A @7C would be required to give the ROM Program Card control of the system at Address 010. The program would then branch to Address 1000 and begin stepping through address locations. If the condition line were low, indicating that voltage was greater than 20 volts, subroutine branch to address location 1100 would occur and operate the three-step measurement-and-print sequence programmed in address locations 1100, 1101, 1102. Address 1103 contains a subroutine return code, which would cause the control to return to Address Location 1003. The branch statement at 1003, 1004 would cause control to go to Address 1000 where the program sequence would recycle. If no condition line low were to occur, then the program would ignore steps 1001, 1002, and perform the 1003, 1004 branch to Address Location 1000. Thus, once this program were initiated, the system would go into a continuous loop, only breaking to perform measurements when the voltage output of the unit under test exceeded 20 volts.

Figure 8. Schematic of a ROM Programming Subroutine

Table 5. Typical Subroutine Loop

Address	Octal Word	Machine Word Equivalent	Function
010	362	@7B	Branch unconditionally to 000 on Page 2 = Address 1000.
011	000	000	
1000	006	006	Reset Subroutine flip-flop. Branch conditionally to 100 on Page 2 = Address 1100.
1001	372	@7J	
1002	100	100	
1003	363	@7B	Branch unconditionally to 000 on Page 2 = Address 1100.
1004	000	000	
1100	231	@1I	Set printer to look at bus for data.
1101	305	@4E	Encode DVM to read. Output DVM data to bus. Subroutine return to Address 1003.
1102	317	@4O	
1103	007	007	

When the HP2570A System is first turned on, it is normalized. In normalizing the system, the ROM Control Card is reset so that all branch statements cause branching only to addresses on the 2000 words of memory mounted on-card. If the additional 2000 words of memory mounted off-card is used, then to directly address locations in the off-card memory, an octal 005 must be programmed prior to using the branch statement. For example, an @7F, 073 will normally cause branching to address location 3073, which is on-card, whereas a sequence of 005, @7F, 073 will cause branching to address 7073, which is mounted off-card. Once an 005 statement has been programmed, all branch statements will go to the off-card memory until an octal 004 is programmed to cause all following branch statements to go to on-card memory. A fifth special code statement is the octal 000. This code causes the ROM Program Card to return system control to the HP2570 Control Card.

The special 000, 004, 005, 006, and 007 statements are valid at all times when programmed into memory except when used as the second half of a branch statement. In this case the ROM Program Card interprets them as addresses instead of octal codes. In addition, these codes are valid only when they are from programmed memory on the ROM Program Card. If a device mated to the HP2570A bus were to place one of these codes on the bus the code would be ignored by the ROM Program Card. Following is a summary of ROM card special codes. These specific octal codes were chosen from available codes which could not be mistaken as ASCII codes or information.

- 0 0 6 Sets up subroutine;
- 0 0 7 Initiates subroutine return;
- 0 0 5 Branches from low to high memory;
- 0 0 4 Branches from high to low memory; and
- 0 0 0 Returns system control to HP2570A control card.

The Read-Only Memory Program Card is wired to an output connector to allow easy cable connections to an external address switch register. With this switch register, the binary address of a memory location may be toggled in, and then a load button depressed. If the ROM Program Card is in control of the system, then that address location will become active. Additionally, the off-card 2000 words memory, as well as readouts to monitor address location, memory content, and bus content may be mated to the ROM Program Card through its output connector. Figure 9 shows a monitor and switch register chassis developed for this project.

Figure 9. ROM Card Monitor

Appendix C has a complete schematic of the ROM Card and a general description of how it works.

Programming Read-Only Memories

To obtain capability to automatically program Intel 1702 ROM's, the Intel 1702 manual programming card was interfaced to the coupler/controller, using the BCD output and teletype interface cards in the coupler/controller. The programmer box (Figures 10 and 11) allows for manual programming, programming from a teletype keyboard, or programming from a teletype tape. In addition, the programmer can be selected to do either a fast or slow programming sequence. The fast sequence allows the ROM to be programmed, but may lose its contents from electrical degradation after approximately 5 years. The slow sequence permanently programs the ROM. The teletype keyboard and tape programming is in octal, is faster, and provides fewer chances for error. The teletype tape method also may be edited before it is to program a ROM.

Figure 10. INTEL 1702 Programmer

The teletype format used for programming the ROM's is relatively simple. A starting address is given by typing a GXXX where XXX is the three-digit octal starting address. The content of that address is typed as YYY where YYY is the octal content. The ROM programmer will automatically step to the next address location to receive the next content word unless another GXXX sequence is typed to change addresses. Information is transferred with the Shift L key, and is echoed as / by the teletype.

If a ROM were to be programmed to be PAGE 2, according to the example given in the section on branch and subroutine statements, a teletype tape or manual sequence would read: G000/006/372/100/363/000/G100/231/305/317/007/

Figure 11. Programmer and Teletype Interfaced to Coupler/
Controller

If an ROM Program requires a change, or if a completely new program is desired, the old program may be erased by exposing the ROM to high-intensity ultraviolet light for approximately 10 minutes.

ROM's are manufactured by different methods and thus are programmed differently. A few of the different methods are fusible link, diode junction shorting, and floating gate avalanche-injection. It would be desirable to be able to program any of the different types and not be committed to one vendor. An investigation was made into the available equipment for programming ROM's. The Spectrum Dynamics Model 550 programmer/verifier was selected and purchased. It is a universal programmer capable of programming any of the available ROM's on the market, using a personality module which plugs into the base unit. Figure 12 shows the unit with the Intel 1702 personality module in place and the Intersil 5600 personality module alongside the base unit. The unit may be interfaced with a teletype for automatic programming.

Computer Programs

Two computer programs written in FORTRAN were generated to aid the incorporation of ROM design information. Both programs generate an 8-bit paper tape to automatically program ROM's. One program is written specifically for the HP2570A coupler/controller ROM card and the other program is for conversion of octal code from the teletype into binary code for programming an ROM.

HP2570 Assembler Program (Appendix B)

The input code of an HP2570A coupler/controller instruction is converted to an 8-bit word on a punched paper tape which is used to program the ROM memory card used in the controller. The program was written for the CDO 6600 computer and input is by punched IBM cards.

Example 1

An input IBM card statement such as @3I would be converted into the following 8-bit punched paper tape hole sequence.

Figure 12. Spectrum Dynamics Model 550 ROM Programmer

Example 2

The following IBM card statements would be converted to the corresponding punched paper tape hole sequence by the HP2570A Assembler Program.

Card	IBM Card Statement	Punched Paper Tape Hole Sequence							
		8	7	6	5	4	3	2	1
1	@ 6 E	0	0	0		0		0	
2	@ 2 E	0		0		0		0	
3	@ 5 E	0	0		0	0		0	
4	@ 4 K	0	0			0		0	0
5	@ 6 O	0	0	0		0	0	0	0
6	@ 2 O	0		0		0	0	0	0
7	@ 5 O	0	0		0	0	0	0	0
8	@ 4 N	0	0			0	0	0	

A typical printout from the HP2570A Program is shown in Figure 13.

COUPLER CONTROLLER PROGRAMMER ASSEMBLER							
EXAMPLE 1 DIGITAL VOLTMETER AND PRINTER							
PAGE	ADDRESS	CONTENT	COMMAND	DEVICE FUNCTION	COMMENT	SYSTEM FUNCTION	COMMENT
0	0	231	411	START	TEST 1	PRINTER LOOKS FOR DATA ON BUSS	1
0	1	305	44F	TEST	1	DVM TAKES A READING	2
0	2	317	440	END	TEST 1	PRINTER PRINTS DVM OUTPUT	3

Figure 13. Typical Printout, HP2570A Program

ROM Support Program (Appendix A)

This program, designed to facilitate the programming of ROM's, is written in Fortran and is available on the CDC6600 computer. Input to the program is with punched data cards, using essentially two modes of operation. Mode 0 accepts input data as three-digit octal numbers. Mode 1 accepts input data as Boolean algebraic equations where the output lines of the ROM are a function of the input lines.

The programmer has a paper-tape option in either mode of operation. If the option is selected, a punched paper tape can be used with the programmer box and teletype to automatically program the ROM.

In Mode 0 the program accepts the ROM definition in three-digit octal code and converts it to binary; after conversion to binary the truth table for the ROM is printed. An example of a typical programming data deck is shown in Figure 14, and Figure 15 shows the truth tables for the ROM's.

In Mode 1 the program accepts the ROM definition as Boolean algebraic expressions. Figure 16 is an example of a circuit that is to be incorporated into an Intersil 32-word ROM. After the circuits outputs are defined, they are punched on data cards and inserted into the source program deck along with the data cards indicating the type of ROM that is to be programmed. Figure 17 is an example of a typical input source deck with the resulting output truth table shown in Figure 18.

JAG64, 8864-87978-21-ROM PROGRAM, ANDERSON, CM40088, T5B, P1.
ATTACH, ROM, ROM1.

ROM.

2 TWO ROM'S TO BE PROGRAMMED

5 FIVE INPUT PINS

8 EIGHT OUTPUT PINS

32 32 WORDS

0 0 MODE 0 AND NO TAPE IS TO BE GENERATED

EXAMPLE 1					OCTAL INPUT ADDRESS PINS				INTERJIL 1M5600				
14	13	12	11	10	9	8	7	6	5	4	3	2	1
000	32												
	007		017		007		007		007		007		007
	007		007		007		007		007		007		007
	006		007		007		007		007		007		007
	007		007		027		027		027		027		000

END

EXAMPLE 2					OCTAL INPUT ADDRESS PINS				INTERJIL 1M5610				
14	13	12	11	10	9	8	7	6	5	4	3	2	1
000	32												
	077		037		077		077		077		077		077
	077		077		076		077		067		077		077
	057		077		073		077		077		077		077
	077		077		077		077		077		077		000

END

Figure 14. Typical Input Card Deck, Mode = 0

ACCOMPLISHMENTS

Described below are examples of the development progress achieved in incorporating ROM's into production testers.

The TE5017 tester (Figures 19 and 20) uses the HP2570A coupler/controller and the ROM card to provide approximately 80 percent automation in its testing sequence. The Program and associated punched paper tape.

The PT4007 tester (Figure 21) is fully automated with the use of ROM's as the control feature. The control section (Figure 22) consists of six 2048-bit memories and twenty-two 256-bit memories. All ROM programming was done at Bendix.

Text is continued on Page 35.

EXAMPLE 1

OCTAL INPUT

INTERFIL 5600

OCTAL ADDRESS	OCTAL ADDRESS	OCTAL CONTENT	A 1 4 3 2					OCTAL OUTPUT							
			A(4) (14)	A(3) (13)	A(2) (12)	A(1) (11)	A(0) (10)	O(0) (0)	O(1) (1)	O(2) (2)	O(3) (3)	O(4) (4)	O(5) (5)	O(6) (6)	O(7) (7)
0	0000	007	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0001	017	0	0	0	0	1	0	0	0	0	0	1	1	1
2	0002	007	0	0	0	1	0	0	0	0	0	0	1	1	1
3	0003	007	0	0	0	1	1	0	0	0	0	0	1	1	1
4	0004	007	0	0	1	0	0	0	0	0	0	0	0	1	1
5	0005	007	0	0	1	0	1	0	0	0	0	0	0	1	1
6	0006	005	0	0	1	1	0	0	0	0	0	0	0	1	1
7	0007	007	0	0	1	1	1	0	0	0	0	0	0	1	1
8	0010	007	0	1	0	0	0	0	0	0	0	0	0	1	1
9	0011	007	0	1	0	0	1	0	0	0	0	0	0	1	1
10	0012	007	0	1	0	1	0	0	0	0	0	0	0	1	1
11	0013	007	0	1	0	1	1	0	0	0	0	0	0	1	1
12	0014	007	0	1	1	0	0	0	0	0	0	0	0	1	1
13	0015	007	0	1	1	0	1	0	0	0	0	0	0	1	1
14	0016	007	0	1	1	1	0	0	0	0	0	0	0	1	1
15	0017	007	0	1	1	1	1	0	0	0	0	0	0	1	1
16	0020	006	1	0	0	0	0	0	0	0	0	0	0	1	0
17	0021	007	1	0	0	0	1	0	0	0	0	0	0	1	0
18	0022	007	1	0	0	1	0	0	0	0	0	0	0	1	1
19	0023	007	1	0	0	1	1	0	0	0	0	0	0	1	1
20	0024	007	1	0	1	0	0	0	0	0	0	0	0	1	1
21	0025	007	1	0	1	0	1	0	0	0	0	0	0	1	1
22	0026	007	1	0	1	1	0	0	0	0	0	0	0	1	1
23	0027	007	1	0	1	1	1	0	0	0	0	0	0	1	1
24	0030	007	1	1	0	0	0	0	0	0	0	0	0	1	1
25	0031	007	1	1	0	0	1	0	0	0	0	0	0	1	1
26	0032	007	1	1	0	1	0	0	0	0	0	0	0	1	1
27	0033	007	1	1	0	1	1	0	0	0	0	0	0	1	1
28	0034	007	1	1	1	0	0	0	0	0	0	0	0	1	1
29	0035	007	1	1	1	0	1	0	0	0	0	0	0	1	1
30	0036	007	1	1	1	1	0	0	0	0	0	0	0	1	1
31	0037	000	1	1	1	1	1	0	0	0	0	0	0	0	0

EXAMPLE 2

OCTAL INPUT

INTERFIL 5610

OCTAL ADDRESS	OCTAL ADDRESS	OCTAL CONTENT	A 1 4 3 2					OCTAL OUTPUT							
			A(4) (14)	A(3) (13)	A(2) (12)	A(1) (11)	A(0) (10)	O(0) (0)	O(1) (1)	O(2) (2)	O(3) (3)	O(4) (4)	O(5) (5)	O(6) (6)	O(7) (7)
0	0000	077	0	0	0	0	0	0	0	1	1	1	1	1	1
1	0001	077	0	0	0	0	1	0	0	1	1	1	1	1	1
2	0002	077	0	0	0	1	0	0	0	1	1	1	1	1	1
3	0003	077	0	0	0	1	1	0	0	1	1	1	1	1	1
4	0004	077	0	0	1	0	0	0	0	1	1	1	1	1	1
5	0005	077	0	0	1	0	1	0	0	1	1	1	1	1	1
6	0006	077	0	0	1	1	0	0	0	1	1	1	1	1	1
7	0007	077	0	0	1	1	1	0	0	1	1	1	1	1	1
8	0010	077	0	1	0	0	0	0	0	1	1	1	1	1	1
9	0011	077	0	1	0	0	1	0	0	1	1	1	1	1	1
10	0012	076	0	1	0	1	0	0	0	1	1	1	1	1	0
11	0013	077	0	1	0	1	1	0	0	1	1	1	1	1	1
12	0014	067	0	1	1	0	0	0	0	1	1	0	1	1	1
13	0015	077	0	1	1	0	1	0	0	1	1	1	1	1	1
14	0016	075	0	1	1	1	0	0	0	1	1	1	0	1	1
15	0017	077	0	1	1	1	1	0	0	1	1	1	1	1	1
16	0020	057	1	0	0	0	0	0	0	1	1	1	1	1	1
17	0021	077	1	0	0	0	1	0	0	1	1	1	1	1	1
18	0022	077	1	0	0	1	0	0	0	1	1	1	0	1	1
19	0023	077	1	0	0	1	1	0	0	1	1	1	1	1	1
20	0024	077	1	0	1	0	0	0	0	1	1	1	1	1	1
21	0025	077	1	0	1	0	1	0	0	1	1	1	1	1	1
22	0026	077	1	0	1	1	0	0	0	1	1	1	1	1	1
23	0027	077	1	0	1	1	1	0	0	1	1	1	1	1	1
24	0030	077	1	1	0	0	0	0	0	1	1	1	1	1	1
25	0031	077	1	1	0	0	1	0	0	1	1	1	1	1	1
26	0032	077	1	1	0	1	0	0	0	1	1	1	1	1	1
27	0033	077	1	1	0	1	1	0	0	1	1	1	1	1	1
28	0034	077	1	1	1	0	0	0	0	1	1	1	1	1	1
29	0035	077	1	1	1	0	1	0	0	1	1	1	1	1	1
30	0036	077	1	1	1	1	0	0	0	1	1	1	1	1	1
31	0037	000	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 15. Truth Tables for ROM's of Figure 14

Figure 16. Circuit for an Intersil 32-Word ROM

J4864,0864-07978-21-ROM PROGRAM,ANDERSON,CH50000,750,PI.

FTN.

LGO.

*00

OVERLAY(ROM,0,0)

PROGRAM MIKE(INPUT,OUTPUT,TAPES=INPUT,TAPES=OUTPUT)

C *****

C *

C * HEAD-ONLY-MEMORY PROGRAM SUPPORT *

C *

C *****

C *A* ENTER NUMBER OF DATA SETS ((12) FORMAT)

C ROMPS OR. M.J.DEVANEY D/864 7/15/71 LOC6600

C *2* ENTER NUMBER OF ADDRESS LINES ((12) FORMAT) (N)

C

*

*

*

*

S ,NOT,LA.AND.,NOT,LO.AND,LC

DATA (A=10*,FALSE.), (U=0*,FALSE.)

DO 50 I=1,NWORDS

C *A* ENTER OUTPUT FUNCTIONS O(1),O(2),...,O(N) AS LOGICAL EXPRESSIONS

C WHICH ARE FUNCTIONS OF THE ADDRESS INPUTS A(1)=A(0),A(2)=A(1),...,

C A(N-1)=A(N)

C I.E.

O(1)=A(1).AND,A(2)

O(2)=A(3).OR,A(4)

O(3)=NOT,(A(1).AND,A(2))

O(4)=NOT,(A(3).OR,A(4))

O(5)=EXOR(A(4),A(5))

O(6)=EXOR3(A(3),A(4),A(5))

O(7)=NOT,((A(3).AND,A(5)).OR,(A(1).AND,A(4)))

O(8)=((A(1).AND,A(2)).OR,NOT,A(3)).AND,((A(1).AND,A(2)).OR,A(4))

MWORDS(I)=0

DO 10 J=1,N0

MWORDS(I)=SHIFT(MWORDS(I),1)

*

*

*

*

RETURN

END

"

1

5

8

32

1

ROM PDD BOOLEAN EXPRESSION INPUT

INTERSIL IM5610

14 13 12 11 10 9 8 7 6 5 4 3 2 1

"

Figure 17. Typical Input Source Deck (Mode=1)

READ ONLY MEMORY PROGRAM SPECIFICATION

ROM PBO

BOOLEAN EXPRESSION INPUT

INTERFIL IM5610.

PTN NUMBER	OCTAL ADDRESS	OCTAL CONTENT	R T N A P Y												
			A(4)	A(3)	A(2)	A(1)	A(0)	O(8)	O(7)	O(6)	O(5)	O(4)	O(3)	O(2)	O(1)
			(14)	(13)	(12)	(11)	(10)	(9)	(7)	(6)	(5)	(4)	(3)	(2)	(1)
	0	0000	114	0	0	0	0	0	1	0	0	1	1	0	0
	1	0001	114	0	0	0	0	1	0	1	0	0	1	1	0
	2	0002	114	0	0	0	1	0	0	1	0	1	1	0	0
	3	0003	311	0	0	0	1	1	1	0	0	1	0	0	1
	4	0004	146	0	0	1	0	0	0	1	0	0	1	1	0
	5	0005	146	0	0	1	0	1	0	1	0	0	1	1	0
	6	0006	146	0	0	1	1	0	0	1	0	0	1	1	0
	7	0007	343	0	0	1	1	1	1	1	0	0	0	1	1
	8	0010	346	0	1	0	0	0	1	1	1	0	1	1	0
	9	0011	256	0	1	0	0	1	0	1	1	0	1	1	0
	10	0012	346	0	1	0	1	0	1	1	1	0	1	1	0
	11	0013	243	0	1	0	1	1	0	1	1	0	0	1	1
	12	0014	126	0	1	1	0	0	0	1	0	1	1	1	0
	13	0015	026	0	1	1	0	1	0	0	1	0	1	1	0
	14	0016	126	0	1	1	1	0	0	1	0	1	1	1	0
	15	0017	223	0	1	1	1	1	1	0	0	1	0	1	1
	16	0020	174	1	0	0	0	0	0	1	1	1	1	0	0
	17	0021	174	1	0	0	0	1	0	1	1	1	1	0	0
	18	0022	174	1	0	0	1	0	0	1	1	1	1	0	0
	19	0023	371	1	0	0	1	1	1	1	1	1	0	0	1
	20	0024	026	1	0	1	0	0	0	0	1	0	1	1	0
	21	0025	026	1	0	1	0	1	0	0	1	0	1	1	0
	22	0026	026	1	0	1	1	0	0	0	1	0	1	1	0
	23	0027	223	1	0	1	1	1	1	0	0	1	0	1	1
	24	0030	306	1	1	0	0	0	1	1	0	0	0	1	0
	25	0031	206	1	1	0	0	1	1	0	0	0	0	1	0
	26	0032	306	1	1	0	1	0	1	1	0	0	0	1	0
	27	0033	203	1	1	0	1	1	0	0	0	0	0	1	1
	28	0034	006	1	1	1	0	0	0	0	0	0	1	1	0
	29	0035	006	1	1	1	0	1	0	0	0	0	1	1	0
	30	0036	006	1	1	1	1	0	0	0	0	0	1	1	0
	31	0037	203	1	1	1	1	1	1	0	0	0	0	1	1

Figure 18. Output Truth Table from Input Source Deck

Figure 19. Control Section of TE5017 Tester

FUTURE WORK

Future testing equipment will be evaluated to determine specific application for ROM incorporation.

The Spectrum Dynamic ROM Programmer will be interfaced to the Bendix PDP8/e laboratory computer system to permit the programming of ROM's from any of the computer's input devices.

Figure 20. TE5017 Tester

Figure 21. PT4007 ROM-Controlled Tester

Figure 22. ROM Control Chassis of PT4007

Appendix A
ROM SUPPORT PROGRAM

JAB64,0864-07978-21-ROM PROGRAM,ANDERSON,CHS08000,150,P1.

FJN.

LED.

PDR

```

OVERLAY(ROM,0,0)
PROGRAM MIKE(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
*****
C      *
C      *      READ-ONLY=MEMORY PROGRAM SUPPORT
C      *
C      *
C      *****
C  *A*  ENTER NUMBER OF DAT A SETS ((12) FORMAT)
C      RUMPS   DR, M,J,DEVANEY D/864 7/15/71 C0C6680
C
C  *2*  ENTER NUMBER OF ADDRESS LINES ((12) FORMAT)      (N)
C
C  *4*  ENTER NUMBER OF DATA LINES ((12) FORMAT)      (M)
C
C  *6*  ENTER NUMBER OF DATA WORDS 14 FORMAT = TOTAL MEMORY WORD SIZE
C  *7*  ENTER PROGRAM MODE CONTROLS
C      MODE=0 DATA READ IN OFF CARDS
C      MODE=1 DATA GENERATED FROM EQUATIONS ENTERED IN SUB FUNGEN
C      MODE FORMAT 11,TAPE=1 IN COL. 10 ON SAME CARD,NO TAPE OTHERWISE
C  *1*  ENTER 88 CHAR. LABEL-CENTERED ON CARD-TITLE WILL APPEAR ON DOCUMENT
C  *3*  ENTER PIN DESIGNATIONS FOR ADDRESS LINES A(N=1),A(N=2),...,A(8) =
C      2B14 FORMAT
C  *5*  ENTER PIN DESIGNATIONS FOR DATA LINES D(M=1),...,D(U)
C      2B14 FORMAT
C  *8*  ENTER STARTING ADDRESS (MODE=0 ONLY) AND NO. WORDS TO BE ENTERED
C      FORMAT D3,2X,13 STARTING ADDRESS ENTERED IN OCTAL
C  *9*  ENTER DATA,MODE=0 ONLY.
C      DATA MAY BE ENTERED IN THREE FORMATS. THESE FORMATS MAY BE INTERMIXED
C      WITHIN A SET. DATA FORMAT TYPE #1 - THE CONTENTS OF EACH ADDRESS MAY BE
C      SPECIFICALLY DEFINED. THE DATA IS THEN ENTERED IN (8010) FORMAT.
C      DATA FORMAT TYPE#2 - A REPEAT STATEMENT MAY BE USED TO REPEAT A GIVEN
C      OCTAL CONTENT FOR A SPECIFIED NUMBER OF ADDRESSES. THE WORD REPEAT
C      MUST APPEAR IN CARD COLUMNS 1-6 FOLLOWED BY A COMMA AND THE OCTAL CONTENT
C      TO BE REPEATED FOLLOWED BY A COMMA AND THE DECIMAL NUMBER OF WORDS TO BE
C      FILLED WITH THE OCTAL CONTENT.
C      EXAMPLE = REPEAT,164,29
C      DATA TYPE#3 - AN 'INIT' STATEMENT MAY BE SUPPLIED TO INDICATE
C      THAT THE USER WISHES TO START LOADING TYPE #1 OR TYPE#2 DATA AT A
C      SPECIFIED DECIMAL ADDRESS THE FORMAT OF THIS STATEMENT IS -
C      INIT,129
C      WHERE THE WORD 'INIT' APPEARS IN CARD COLUMNS 1-4 FOLLOWED BY A
C      COMMA FOLLOWED BY THE DECIMAL ADDRESS. THE INTERIM ADDRESSES BETWEEN THE
C      LAST LOADED ADDRESS AND THE ADDRESS SPECIFIED BY THE INIT STATEMENT WILL
C      BE FILLED WITH NO-OPS.
C  *10* ENTER THE WORD END TO INDICATE THAT ALL DATA FOR A SET HAS BEEN SUPPLIED.
C      IF THE NUMBER OF OCTAL CONTENTS SUPPLIED IS LESS THAN THE MEMORY WORD
C      SIZE, THE REMAINING LOCATIONS WILL BE PADDED WITH 1'S.
C
C
C      DIMENSION MAPINS(16),NOPINS(16),MWORDS(4096),LABEL(6),LINE(13) ---
C      INTEGER CARD(75)
C      DATA LINE/13=18H,...../
C      READ(5,27) NSETS
C      27 FORMAT(12)
C      IER=0
C      READ(5,29)NA
C      READ(5,29)NU
C      29 FORMAT(12)
C      READ(5,79) MWORDS
C      79 FORMAT(14)

```

```

      READ(5,49) MODE,ITAPE
42  FORMAT(I1,0X11)
      DO 101 JX=1,NSETS
      READ(5,9) LABEL
      9  FORMAT(0A10)
      WRITE(6,19) LABEL
19  FORMAT(1H,/,47X,30HREAD ONLY MEMORY PROGRAM SPECIFICATION,/,20X,
      1 0A10,/)
      READ(5,57) (NAPINS(I),I=1,NA)
      READ(5,57) (NOPINS(I),I=1,N0)
57  FORMAT(10I5)
      IF (MODE.EQ.1) GO TO 50
      READ(5,89) ISTART,LWORDS
89  FORMAT(05,I5)
      JSTART=ISTART
      IFILL=3770
      IF (LWORDS.EQ.32) IFILL=00
      IF (ISTART.EQ.0) GO TO 120
      DO 555 I=1,ISTART
555  MWORDS(I)=3770
      IF (LWORDS.EQ.256) GO TO 122
      DO 55 I=1,ISTART
      55  MWORDS(I)=0
      JSTART=0
122  ISTART=ISTART+1
120  READ(5,39) CARD
      ISTART=ISTART+1
39  FORMAT(A6,74A1)
      IF (CUP(5)) 500,121
121  IF (CARD(1).EQ.0HREPEAT) GO TO 200
      MASK=CARD(1).AND.77777770000000000000,OR.000000005555555555550
      IF (MASK.EQ.4HINIT) GO TO 400
      IF (CARD(1).EQ.3HEND) GO TO 500
      BACKSPACE 5
      LAST=ISTART+7
      READ(5,40) (MWORDS(I),I=ISTART,LAST)
40  FORMAT(0A10)
      DO 41 I=1,0
      41  R=1
      J=0-I
      IF (MWORDS(ISTART+J).NE.1H) GO TO 43
41  CONTINUE
      GO TO 120
43  IC=1H+(J+1)+0.5
      JJ=ISTART+J
      DECODE(IC,42,MWORDS(ISTART))(MWORDS(I),I=ISTART,JJ)
42  FORMAT(0I10)
      DO 44 I=ISTART,JJ
      44  CALL OCTDEC(MWORDS(I),MWORDS(I))
      MWORDS(I)=MWORDS(I).AND.7770
      ISTART=ISTART+J
      GO TO 120
200  DO 201 I=0,00
      201  J=1
      IF (CARD(I-5).EQ.1H,) GO TO 203
201  CONTINUE
210  WRITE(6,202) CARD
202  FORMAT(1X,A6,74A1/7H *****IMPROPER FORMAT FOR REPEAT STATEMENT*)
      JEN=1
      GO TO 120
203  IC=J-0
      JJ=J-0
      IF (IC.GE.10) GO TO 205
      ENCODE(4,204,1FM?) IC
204  FORMAT(* (I=,I1,=) *)
      GO TO 207

```



```

205 ENCODE(5,206,IFMT) IC
206 FOMHAT(= (I1,I2,*)*)
207 ENCODE(IC,216,KWORD) (CARD(I),I=3,JJ)
216 FOMHAT(10A1)
DECODE(IC,IFMT,KWORD) KWORD
CALL OCTOEE(KWORD,KWORD)
DO 208 I=1,75
K=76-I
IF (CARD(K),NE,1H ) GO TO 209
208 CONTINUE
209 IF (K,LE,JJ+1) GO TO 210
IC=K-(JJ+1)
IF (IC,GE,10) GO TO 212
ENCODE(4,211,IFMT) IC
211 FOMHAT(= (I1,I1,*)*)
GO TO 214
212 ENCODE(5,213,IFMT) IC
213 FOMHAT(= (I1,I2,*)*)
214 JJ=JJ+2
ENCODE(IC,216,NX) (CARD(I),I=JJ,K)
DECODE(IC,IFMT,NX) NX
DO 215 I=1,NX
215 MWORDS(ISTART+I-1)=KWORD
ISTART=ISTART+NX-1
GO TO 120
500 I=WORDS=LWORDS
IF (ISTART,EQ,LWORDS) GO TO 70
IF (LWORDS,EQ,32) GO TO 123
I=WORDS=ISTART-1
DO 124 I=ISTART,LWORDS
124 MWORDS(I)=377B
GO TO 70
123 DO 501 I=ISTART,LWORDS
501 MWORDS(I)=0B
JSTART=0
GO TO 70
400 BACKSPACE 5
READ(5,401) CARD
401 FOMHAT(5X,75A1)
KCOL=0
DO 402 I=1,75
J=1
IF (CARD(I),EQ,1H ) GO TO 403
402 CONTINUE
404 WRITE(6,405) CARD
405 FOMHAT(1X,*INIT,*,75A1,7H ***** ,*IMPROPER FORMAT FOR INIT STATE*
/NT*)
GO TO 999
403 IC=J-1
ENCODE(4,211,IFMT) IC
ENCODE(IC,216,NX) (CARD(I),I=1,IC)
DECODE(IC,IFMT,NX) NX
IF (NX,LT,ISTART) GO TO 122
IF (NX,EQ,ISTART) GO TO 120
I=NX
DO 406 I=ISTART,IF
406 MWORDS(I)=IFILL
ISTART=IF
GO TO 120
50 CALL FUNGEN(MWORDS,MWORDS,NA,NO)
70 WRITE(6,59)LINE
59 FOMHAT(1X,13A10,/)
IF (IER,NE,0) GO TO 101
CALL OUTPUT(NAPINS,NOPINS,MWORDS,NA,NO,MWORDS,LABEL,LINE)
IF (ITAPE,EQ,0) GO TO 100
CALL PPTAPE(LABEL,MWORDS,IWORDS,JSTART)

```

```

100 WRITE(6,69)
69. FORMAT(1H1)
101 CONTINUE
--- IE(1TAP,NE,1) GO TO 999
--- CALL DLIN(50,DUM,5,IRG,0)
--- CALL BXDISP(1MU,0,1H2)
999 CONTINUE
STOP
END
SUBROUTINE FUNGEN(NWORDS,MWORDS,NA,NO)
INTEGER SHIFT
LOGICAL LA,LB,LC
LOGICAL A(10),O(8),C,SUM
LOGICAL EXOR,EXOR3
DIMENSION MWORDS(MWORDS)
EXOR(LA,LB)=(LA.AND..NOT,LB.OR,LB.AND..NOT,LA)
EXOR3(LA,LB,LC)=LA.AND..NOT,LB.AND..NOT,LC.OR,
--- NOT,LA.AND,LB.AND..NOT,LC.OR,
--- NOT,LA.AND..NOT,LB.AND,LC
DATA (A=10,FALSE),(O=8,FALSE)
DO 50 I=1,MWORDS
C ** ENTER OUTPUT FUNCTIONS O(1),O(2),...,O(NO) AS LOGICAL EXPRESSIONS
C WHICH ARE FUNCTIONS OF THE ADDING INPUTS A(1)=A(0),A(2)=A(A),....
C A(NA-1)=A(NA)
C I.E.
O(1)=A(0).AND,A(1)
O(2)=A(2).OR,A(3)
O(3)=.NOT,(A(0).AND,A(1))
O(4)=.NOT,(A(0).OR,A(1))
O(5)=EXOR(A(3),A(4))
O(6)=EXOR3(A(2),A(3),A(4))
O(7)=.NOT,((A(2).AND,A(4)).OR,(A(0).AND,A(3)))
O(8)=((A(0).AND,A(1)).OR,.NOT,A(2)).AND,((A(0).AND,A(1)).OR,A(3))
MWORDS(I)=0
DO 10 J=1,NO
MWORDS(I)=SHIFT(MWORDS(I),1)
IF (O(NO-J+1))MWORDS(I)=MWORDS(I)+1
10 CONTINUE
C=.TRUE.
DO 20 K=1,NA
SUM=A(K).AND..NOT,C.OR,C.AND..NOT,A(K)
C=C.AND,A(K)
A(K)=SUM
IF (.NOT,C)GO TO 50
20 CONTINUE
50 CONTINUE
RETURN
END
SUBROUTINE OUTPUT(NAPINS,NOPINS,MWORDS,NA,NO,MWORDS,LABEL,LINE)
DIMENSION NAPINS(NA),NOPINS(NO),MWORDS(MWORDS),FMTA(10)
1,K(10),K(8),FMTB(8),FMTC(7),JA(10),JO(8),LABEL(8),LINE(13)
DATA FMTA/10H(11X,27HADD,10HRESS ADD,10HRESS CONT,4MENT,1H,
2 10H(2H(,11,2,8MH),5X,1H,10H(2H(,11,2,5MH))1/,IPAGE/1/
DATA FMTB/10H(11H PIN N,10HNUMBER,27X,1H,10H(1H(,12,2H,7H),5X,,
1 1H,10H(1H(,12,2H,5H))1/ /
DATA FMTC/10H( 12X,14,,10H5X,04,6X,0.5H1,2X,,1H,AM(19),5X,,1H,
15H(15)) /
ENCODE(2,9,FNA)NA
ENCODE(2,9,FNO)NO
9. FORMAT(12)
FMTA(5)=FNA
FMTA(8)=FNO
WRITE(6,19)
19. FORMAT(11X,24HDECIMAL OCTAL OCTAL,25X,11H8 I N A R Y )
DO 10 I=1,NA
10 KA(I)=NA-I

```



```

9000 FORMAT(6H1=====, * CONTENTS OF PAPER TAPE *,
10H =====)
PRINT9001,LABEL
9001 FORMAT(1X,8A10/)
IF(1SW.NE.0) GO TO 18
CALL OLINE(10,DUM,2,IRG,1)
CALL ASCII
18 1SW=1
CALL OLINE(8,LABEL,3,IRG,2)
CALL OLINE(10,DUM,2,IRG,1)
IF(MU.NE.0) GO TO 16
15=1
15=8
GO TO 17
16 15=NO
IF=NO+7
17 ENCODE(43,1,IMAGE) NO,NO,(MWORDS(I),I=15,IF)
PRINT9999,IMAGE
9999 FORMAT(1X,5A10)
DECODE(43,2,IMAGE) (IREC(I),I=1,43)
CALL OLINE(43,IREC,1,IRG,0)
1START=1F+1 8 IFIN=0
9 IF=1START+7 8 IC=34
DO 14 I=1,5
14 IMAGE(I)=1H
IF(1F.LT.MNWORDS) GO TO 10
IFIN=1
LWORDS=MNWORDS
IF=LWORDS
K=LWORDS-1START+1
IF(K.EQ.0) GO TO 15
GO TO 11
10 K=8
11 DO 12 I=1,K
12 IFMT(I+1)=7H=//+.03.
IC=K+2
ENCODE(IC,IFMT,IMAGE) (MWORDS(I),I=1START,1F)
DO 13 I=1,8
13 IFMT(I+1)=1H
PRINT9999,(IMAGE(I),I=1,8)
DECODE(IC,2,IMAGE) (IREC(I),I=1,IC)
CALL OLINE(IC,IREC,1,IRG,0)
1START=1F+1
IF(IFIN.EQ.0) GO TO 9
15 CONTINUE
ICAR=1M/
CALL OLINE(1,ICAR,1,IRG,0)
PRINT9003
9003 FORMAT(+ /+)
PRINT9002
9002 FORMAT(////+ COMMENT, / * FORM L. * = LINEFEED, $ = CARRIAGE RETURN
/M=)
RETURN
END
SUBROUTINE ASCII
C=
DIMENSION IASKE(120)
C=
C*** SET PUNCH VALUES FOR ASCII READ-OUT,..
C
IASKE(001) = 1RA
IASKE(002) = 301H
IASKE(003) = 1RB
IASKE(004) = 302B
IASKE(005) = 1RC
IASKE(006) = 303B

```

```

IASKE(007) = 1MO
IASKE(008) = 1040
IASKE(009) = 1RE
IASKE(010) = 1050
IASKE(011) = 1RF
IASKE(012) = 3060
IASKE(013) = 1RG
IASKE(014) = 1070
IASKE(015) = 1RH
IASKE(016) = 3100
IASKE(017) = 1RI
IASKE(018) = 3110
IASKE(019) = 1RJ
IASKE(020) = 3120
IASKE(021) = 1RK
IASKE(022) = 3130
IASKE(023) = 1RL
IASKE(024) = 3140
IASKE(025) = 1RM
IASKE(026) = 3150
IASKE(027) = 1RN
IASKE(028) = 3160
IASKE(029) = 1RO
IASKE(030) = 3170
IASKE(031) = 1RP
IASKE(032) = 3200
IASKE(033) = 1RQ
IASKE(034) = 3210
IASKE(035) = 1RH
IASKE(036) = 3220
IASKE(037) = 1RS
IASKE(038) = 3230
IASKE(039) = 1RT
IASKE(040) = 3240
IASKE(041) = 1RU
IASKE(042) = 3250
IASKE(043) = 1RV
IASKE(044) = 3260
IASKE(045) = 1RW
IASKE(046) = 3270
IASKE(047) = 1RX
IASKE(048) = 3300
IASKE(049) = 1RY
IASKE(050) = 3310
IASKE(051) = 1RZ
IASKE(052) = 3320

```

C*

C*** THE FOLLOWING CHARACTER IS USED FOR THE CARRIAGE RETURN...

C*

```

IASKE(053) = 1R1
IASKE(054) = 2410
IASKE(055) = 1R2
IASKE(056) = 2420

```

C* LINEFEED

```

IASKE(057) = 1R4
IASKE(058) = 2120
IASKE(059) = 1R5
IASKE(060) = 2440
IASKE(061) = 1R6
IASKE(062) = 2450

```

C* CARRIAGE RETURN

```

IASKE(063) = 1R6
IASKE(064) = 2150

```

C*

C*** THE FOLLOWING CHARACTER IS USED FOR THE LINE FEED...

C*


```

IASKE(065) = 1R*
IASKE(066) = 247B
IASKE(067) = 1R*
IASKE(068) = 250B
IASKE(069) = 1R*
IASKE(070) = 251B
IASKE(071) = 1R*
IASKE(072) = 252B
IASKE(073) = 1R*
IASKE(074) = 253B
IASKE(075) = 1R*
IASKE(076) = 254B
IASKE(077) = 1R*
IASKE(078) = 255B
IASKE(079) = 1R*
IASKE(080) = 256B
IASKE(081) = 1R*
IASKE(082) = 134B
IASKE(083) = 1R*
IASKE(084) = 60B
IASKE(085) = 1R*
IASKE(086) = 241B
IASKE(087) = 1R*
IASKE(088) = 242B
IASKE(089) = 1R*
IASKE(090) = 63B
IASKE(091) = 1R*
IASKE(092) = 244B
IASKE(093) = 1R*
IASKE(094) = 65B
IASKE(095) = 1R*
IASKE(096) = 66B
IASKE(097) = 1R*
IASKE(098) = 267B
IASKE(099) = 1R*
IASKE(100) = 270B
IASKE(101) = 1R*
IASKE(102) = 71B
IASKE(103) = 1R*
IASKE(104) = 272B

```

```

C*
C*** THE FOLLOWING CHARACTER IS USED FOR A BLANK CODE...
C*

```

```

IASKE(105) = 1R*
IASKE(106) = 240B
IASKE(107) = 1R*
IASKE(108) = 274B
IASKE(109) = 1R*
IASKE(110) = 275B
IASKE(111) = 1R*
IASKE(112) = 276B
IASKE(113) = 1R*
IASKE(114) = 277B
IASKE(115) = 1R*
IASKE(116) = 240B
CALL OLINE(50, IASKE, -1, 0, 0)
RETURN
END

```

```

1
5
0
32
1
1
NON PDD
14 13 12 11 10
BOOLEAN EXPRESSION INPUT

```

INTERFIL IN5410

1

Appendix B
HP2570A ASSEMBLER


```

C CHECK FOR INVALID SLOT ADDRESS FOLLOWING W.
IF (ISLOT,LT,4) CALL FORPR(2,1,N060)
ICOMND=ISCAN(CADD(1),A10M4,1A)
C CHECK FOR INVALID COMMAND FOLLOWING SLOT ADDRESS.
IF (ICOMND,LT,4) CALL FORPR(3,1,N060)
IF (ISLOT,LT,0,N0,ICOMND,LT,4) GO TO 100
C ENCODE SLOT & COMMAND ID.
INDEX=ISLOT+32+ICOMND
C DETERMINE IF INSTRUCTION IS NEW.
IFIND=ISCAN(INDEX,MLIST,MONTO)
IF (IFIND) 47,45,45
C COPY NEW INSTRUCTION DEVICE COMMENT FIELD INTO DITTO LIBRARY.
40 DO 41 I=1,33
41 MESSAGE(ADNTO,I)=CADD(I+39)
C INCREMENT POINTER INDICATING LOCATION OF NEXT NEW MESSAGE.
MONTO=MONTO+1
GO TO 50
C SCAN COMMENT FIELD OF OLD INSTRUCTION FOR COMMENT.
42 DO 43 I=41,72
IF (CADD(I),NE,HLANK) GO TO 48
44 CONTINUE
C DEVICE FIELD BLANK ENTER DITTO MESSAGE FROM LIBRARY.
DO 47 I=1,33
47 CADD(I+30)=MESSAGE(IFIND,I)
GO TO 50
C DEVISE COMMENT IN DITTO LIBRARY.
40 DO 49 I=1,33
49 MESSAGE(IFIND,I)=CADD(I+34)
C CONVERT TO JOE78 BACKED INSTRUCTION FORMAT
50 ICODE=128+(ISLOT-1)*16+ICOMND-1
50 MEMOBY(DEVCTD+1)=ICODE
C DETERMINE PAGE AND PAGE ADDRESS.
PAGE=DEVCTD/256
ADDRESS=DEVCTD-PAGE*256
NICEF=MIN(LEN,1)
MADES(MINSEB)=DEVCTD
C INCREMENT PROGRAM POINTER
52 DEVCTD=DEVCTD+1
GO TO(40,205,100,100,500),ISWICH
52 FORMAT(20H PAGE ADDRESS CONTENT COMMAND,7X,23HDEVICE FUNCTION COMM
1ENT,0X,23HSYSTEM FUNCTION COMMENT,5X,12HSEQUENCE ID.,//,1X,3244,/)
C OUTPUT ASSEMBLED LISTING TO PRINTER
44 WRITE(A,40) DATE,ADDRESS,ICODE,(CADD(I),I=1,33),(CADD(I),I=40,72),
1(CADD(I),I=10,24),(CADD(I),I=73,80)
C003 PR1 118R,12,1403,12,1833,14,183,15,17,15,11,14,21,12(TAMP0F 96
44 FORMAT(2X,12,4X,13,5X,03,5X,78,14X,1381,2X,30A1,2X,8A1) C005600C
C INCREMENT POINTER LINE POINTER
70 LINECT=LINECT+1
IF (LINECT,IF,50) GO TO 90
PAGECT=PAGECT+1
LINECT=1
WRITE(A,70) LINE
70 FORMAT(1X,1X,3244)
C SKIP TO TOP OF FROM AND OUTPUT PAGE AND LABEL.
WRITE(A,40) LABEL, PAGECT, LINE
80 FORMAT(1H,1/,24X,2044,10X,5HPAGE,13,//,1X,3244,/)
WRITE(A,50) LINE
90 IF (ISWICH,53,2) GO TO 100
GO TO 100
C CONTROL CHARACTER INDICATES CHARACTER STRING.
11STOP=ISCAN(CONTROL(2),IFX,71)
C DETERMINE LENGTH OF CHARACTER STRING.
IF (1STOP,LT,4) 1STOP=72
DO 200 I=2,1STOP
C CONVERT DISPLAY CODE TO ASCII
11ASCII=ISCAN(CADD(I),ACD,45)

```

```

C CHECK FOR INVALID ASCII CHARACTER.
  IF (IASCII.GT.0) GO TO 201
  CALL FROMR(4,1,NORG)
C INVALID CHARACTER -- MEMORY LOCATION SKIPPED.
  PGMCTR=PGMCTR+1
  GO TO 190
201 ICODE=ASCII(IASCII)
  GO TO 55
C063 MPT 1*H),1A),1H),X5,3Z,X5,3I,X4,2I,X2(TAMPOF 902
205 WRITE(4,209) PAGE,ADDRES,ICODE,CARD(I)
200 FORMAT(2X,12,4X,13,5X,03,5X,1H*,1A1,1H*) CDC6600C
  GO TO 70
190 CONTINUE
200 CONTINUE
  GO TO 100
  WRITE(4,300) ROWDEF,TEXT,ROWDEF
300 FORMAT(27X,19A4,/,27X,3H* ,714),3H *,/,27X,19A4)
  LINCNT=LINCNT+2
  GO TO 70
  LOPBNT=0
C CONVERT LOAD POINT FROM DISPLAY OCTAL TO NUMERIC.
  DO 400 I=2,5
  ITEST=ISCAN(CARD(I),MIN*,R)
C TEST FOR INVALID SYMBOL IN OCTAL LOAD POINT
  IF (ITEST.GT.0) GO TO 399
  CALL FROMR(5,1,NORG)
  GO TO 100
390 CONTINUE
400 LOPBNT=LOPBNT+4+ITEST-1
C SKIP SINGLE LINE ON LISTPOINTED INDICATING SHIFT IN ADDRESSES.
  LINCNT=LINCNT+1
  WRITE(4,400)
400 FORMAT(1/)
  GO TO 100
  OCTCON=0
C CONVERT OCTAL CONSTANT FROM DISPLAY OCTAL TO NUMERIC.
  DO 500 I=2,4
  ITEST=ISCAN(CARD(I),MIN*,R)
C TEST FOR INVALID SYMBOL IN OCTAL CONSTANT.
  IF (ITEST.GT.0) GO TO 499
  CALL FROMR(6,1,NORG)
  GO TO 100
490 CONTINUE
500 OCTCON=OCTCON+4+ITEST-1
C CHECK RANGE OF OCTAL CONSTANT.
  IF (OCTCON.LT.255) GO TO 501
  CALL FROMR(7,1,NORG)
C SKIP LOCATION OF INVALID OCTAL CONSTANT.
  PGMCTR=PGMCTR+1
  GO TO 100
501 ICODE=OCTCON
  GO TO 55
505 WRITE(4,500) PAGE,ADDRES,ICODE,(CARD(I),I=2,4)
500 FORMAT(2X,12,4X,13,5X,03,5X,3A1,1X,4A4),(2X,4A1) CDC6600C
C063 MPT 11A8,X2,1A95,X1,103,X5,37,X5,3I,X4,2I,X2(TAMPOF 905
  GO TO 70
  WRITE(4,10) LABEL,1INF
  WRITE(4,600) NORG,ICARD,NUSED
600 FORMAT(2X,17HASSEMBLY COMPLETE,/,/,25X,19,49H LEVEL 1 FROMDS FNCO1)
  INTCOD IN INPUT DATA STREAM,/,/,25X,21HASSEMBLED STATISTICS,/,/
  2 24X,10HDATA CARDS ENTERED,15,/,/,24X,24HMEMORY LOCATIONS OFFINED,
  3 15,/,/2)
  IF (NORG) 510,510,620
610 WRITE(4,610)
610 FORMAT(2X,24HPR0)GDAMMER TAPE GENERATED,1)
620 WRITE(4,620)

```



```

620 FORMAT(26X,'PROGRAMMER NAME ABORTED.')
      GO TO 1000
001 STOP
      END
      INTERFACE FUNCTION ISCAN(ITAG,LIST,LENGTH)
      FUNCTION SCANS LIST TO DETERMINE THE LOCATION OF ITAG
      IF ITAG IS NOT IN THE LIST ISCAN IS RETURNED NEGATIVE.
      DIMENSION LIST(LENGTH)
      ISCAN=-1
      DO 10 I=1,LENGTH
      IF (LIST(I).EQ.ITAG) GO TO 20
10 CONTINUE
      GO TO 10
20 ISCAN=1
30 RETURN
      END
      SUBROUTINE ERROR(NUM,LEVEL,NOGO)
C *****
C * THIS SUBROUTINE PROVIDES DIAGNOSTICS TO FACILITATE DEBUGGING *
C * ASSEMBLER INPUT. IF AN ERROR IS ENCOUNTERED IN THE INPUT DATA *
C * ASSEMBLER INPUT. IF A LEVEL 1 ERROR IS ENCOUNTERED IN THE *
C * INPUT DATA STREAM NOGO IS INCREMENTED AND WHILE THE ERROR SCAN *
C * CONTINUES TAPE GENERATION IS ABORTED. *
C *****
      COMMON ICARD,CARD(80),LINCNT
      IF (LEVEL.NE.0) NOGO=NOGO+1
      WRITE(6,9) NUM,LEVEL,ICARD,CARD
9    FORMAT(5X,5)***** ERROR ENCOUNTERED IN INPUT DATA STREAMS ERROR.
      I1=5X,5H FVF(,I1,7.5X,9HCARD NO.,(4.5X,80))
      GO TO(1,2,3,4,5,6,7),NUM
1    WRITE(6,19)
10   FORMAT(10X,39H[IVAL] IN CONTROL CHARACTER IN COLUMN 1, %I)
50   RETURN
2    WRITE(6,29)
20   FORMAT(10X,37H[IVAL] IN SLOT ADDRESS IN COLUMN 2, %I)
      GO TO 50
3    WRITE(6,39)
30   FORMAT(10X,29H[IVAL] IN COMMAND IN COLUMN 3, %I)
      GO TO 50
4    WRITE(6,49)
40   FORMAT(10X,47H[IVAL] IN SYMBOL ENCOUNTERED IN CHARACTER STRING, %I)
      GO TO 50
5    WRITE(6,59)
50   FORMAT(10X,67H[IVAL] IN SYMBOL ENCOUNTERED IN LOAD POINT SPEC, %I)
      GO TO 50
6    WRITE(6,69)
60   FORMAT(10X,49H[IVAL] IN SYMBOL ENCOUNTERED IN OCTAL CONSTANT, %I)
      GO TO 50
7    WRITE(6,79)
70   FORMAT(10X,46HOCTAL CONSTANT EXCEEDS 377 OCTAL--ABOVE RANGE, %I)
      GO TO 50
      END
00000000000000000000000000000000
      EXAMPLE 1 DIGITAL VOLTMETER AND PRINTER
011  PONTB 100KS FMT DATA ON RUSC START TEST 1
04E  DVM JAKES & READINGS YES 1
040  PONTB PUNTS DVM OUTPUT END TEST 1
5    THIS CARD FLOPS ASSEMBLY PROGRAMS
000000000000000000000000

```

Appendix C
ROM CARD PROGRAMMING

Appendix C

ROM CARD PROGRAMMING

A basic block diagram of the ROM card is shown in Figure C-1. The schematic is shown in Bendix Drawing 1459397 (attached).

The ROM control section consists of eight 256-word ROM's, U43 through U50. They are addressed by the ROM address section, which consists of U28 through U30 and U39. U28 and U29 address 256 words of memory and U30's output is decoded by U39 which determines the page addressed.

U21 is a 32-word ROM starting address pointer. It permits the ROM card to take control at a starting position other than address 0.

U25, U26, and U27 make up the subroutine setup register. When a branch to a subroutine occurs, U27 stores the page and U25 and U26 the address from which the branch to a subroutine occurred.

In a similar manner U22 through U24 make up the branch setup register. When a conditional or non-conditional branch occurs, U22 and U23 contain the word address to which the branch is to take place, and U24 contains the page number.

The switch address inputs to the ROM Card monitor are from U36 through U38. The bus inputs are from U33 through U35. The actual ROM address and outputs are obtained directly from those lines.

Some major components in the control logic section are U13, U14, U15, U16, U17 and U20. U13 permits the ROM card to take control of the system. U14 enables the ROM address section to accept count pulses and step through sequential memory locations. Branching and subroutining is controlled by U16 and U15. The branch arm F-F causes either a branch address to be stored in the branch register, or the return address of a subroutine to be stored in the subroutine register. The branch executes F-F and enables the branch register outputs, permitting the branch to transfer. U15 sub-clear F-F allows the subroutine setup register to be cleared. U15 Cond Latch is used in a conditional branch statement; if the conditions are true this F-F is set, allowing a conditional branch to take place.

U17 High Core F-F is used if a branch is to or from the 2000 words of memory located off the ROM card. If this F-F is set, the branch is to the high memory section.

Figure C-1. Block Diagram of ROM Card Programming

DISTRIBUTION

	Copy
R. Bulcock, AEC-KCAO	1
V. C. Vespe, AEC-ALO	2
J. A. Freed, LASL	3
S. J. Buginas, LLL	4
H. Brinkmeier, Monsanto	5
M. J. Devaney, University of Missouri at Columbia	6
N. J. Pollard, SLA	7
R. F. Pippert, D/100	8
J. D. Corey, D/554	9-13
L. Stratton, D/554	14-16
R. P. Frohmberg, D/800	17
D. H. Hax, D/800	18
J. D. Anderson, D/864	19-25
R. L. Johnson, D/864	26
R. E. Kessler, D/864	27
W. E. Simes, D/864	28
G. A. Yocum, Jr., D/864	29
J. J. Zimmerman, D/864	30