pacer USER MANUAL

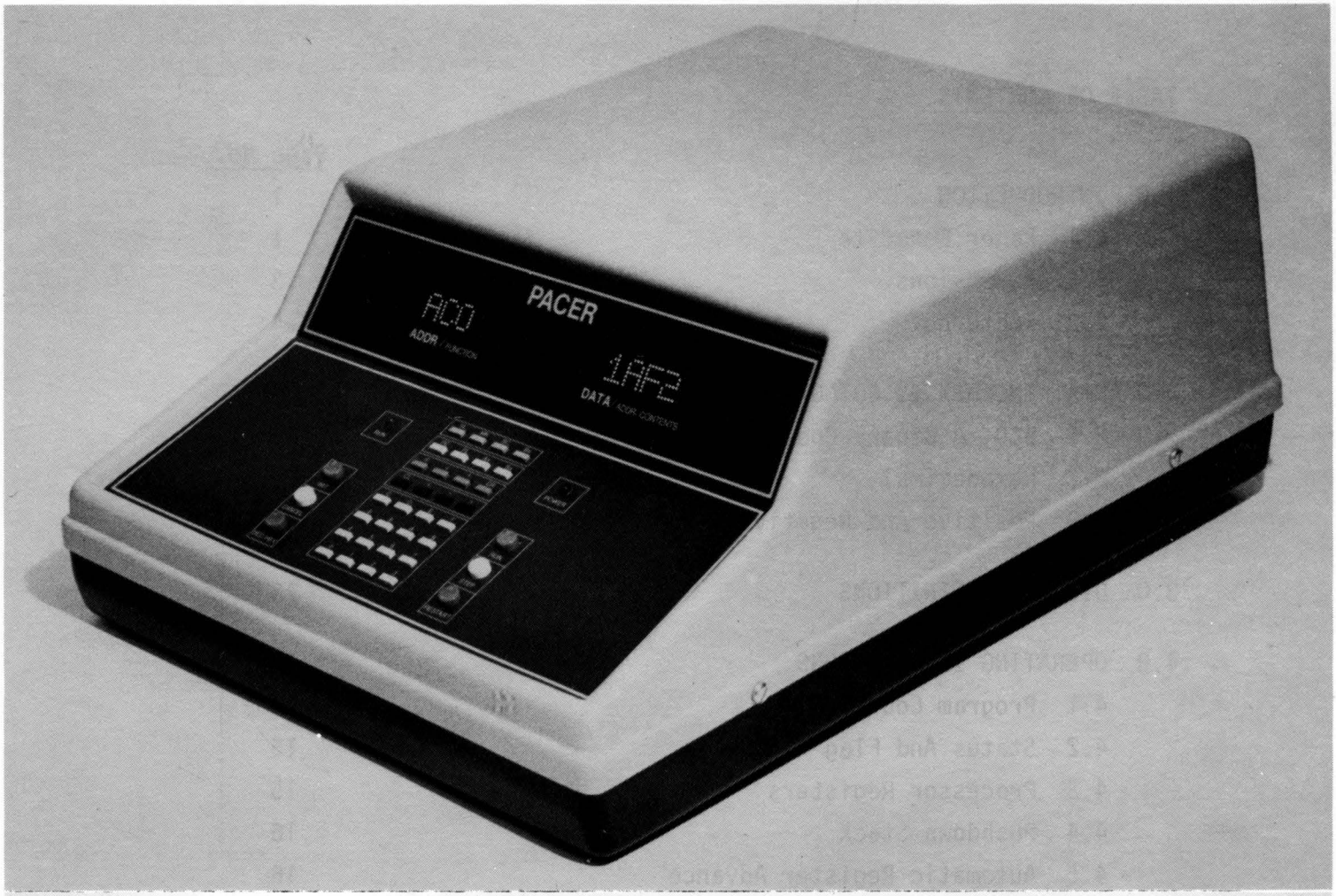
The pacer kit featuring the National 16 Bit "PACE" MPU.

Distributed Exclusively By Hamilton/Avnet

$5.00

pacer

The Hamilton/Avnet PACER system...
A complete Microcomputer system
you can understand and afford.


PACER
ADDR / FUNCTION
DATA

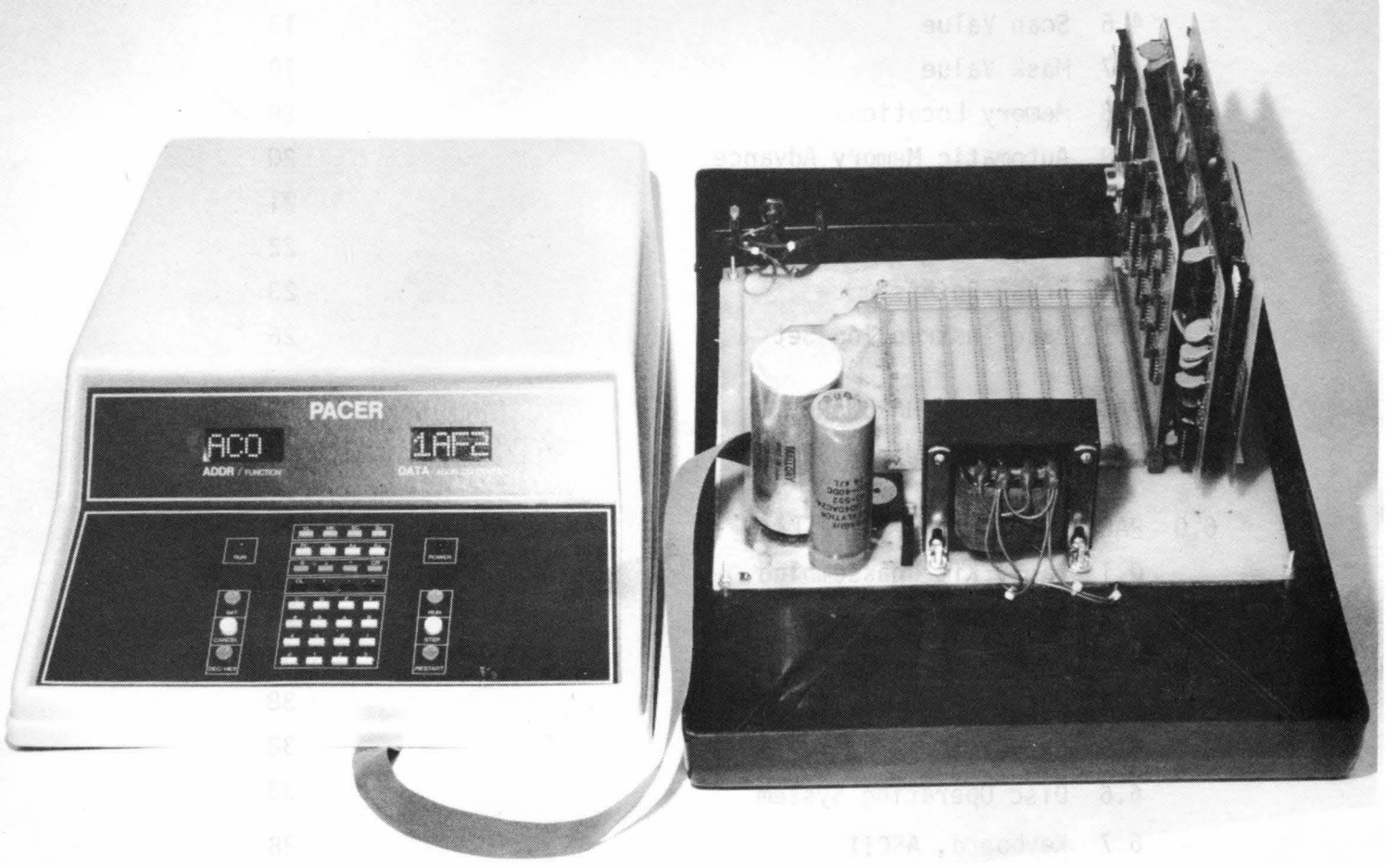

PACER
ADDR / FUNCTION
DATA

TABLE OF CONTENTS

1.0 The PACER is a complete desk top microcomputer development system that may be purchased in kit form, unassembled or completely assembled ready to plug in. In either case the kit is complete, containing everything from the power cord to the plastic enclosure. PACER is the easiest development system to use, its unique alphanumeric display actually talks to you in language you can understand at sight.

1.1 o WHAT ARE PACER'S BENEFITS?

o Complete set of control panel functions:

* Examine or modify
  - Contents of any computer register or memory location can be examined or modified.
  - Examine/modify a location with a single-key stroke (current location).
  - Examine/modify the current location and examine the next sequential location with a single-key stroke (current location +1).
  - Examine/modify the previous sequential location with a single-key stroke (current location -1).
* Run - Execute a program starting at a specified address
* Single Step - Execute a program one instruction at a time starting at a specified address.
* Word Scan - Scan through the MPU's registers or memory starting at a specified location until a location is found having a specified content.

o Complete set of DEBUG functions:

* HALT - Stop program execution at a specified address.
* Breakpoints - Set up to 10 breakpoints to "HALT" execution.
* Decimal to Hexadecimal Conversion - Single-key stroke.
* Hexidecimal Calculator - Plus (+), minus (-), and equal (=) keys
* Address Calculation in Hexadecimal - Current address, +, or - displacement.

o The CPU Board:

* National Semiconductor's PACE microprocessor has all inputs and outputs buffered for system expansion.

o Control Board

* Contains control program in 1K x 16 ROM and 256 x 16 of control RAM. All control memory is transparent to user.

o RAM/PROM Board:

* Contains 256 x 16 RAM, expandable to 1K x 16, and space for 1K x 16 PROM (in 512 x 16 modules).

o Mother Board:

* Printed wiring board which has room for eight user defined card locations besides the three listed. Cards are offset to insure proper position of board.

o Power Supply:

* The power supply provides +8 and -16 volts. These voltages are regulated on each board dependent upon its requirement.

o Full Alphanumeric Display:

* Two 4-digit alphanumeric displays are utilized to allow for easy to understand communication with the operator. For example, when looking at Accumulator 1, display reads: [A|C|1|.] [Ø|A|F|S]

o Keyboard:

* A 32-key keyboard plus 6 additional single keys are supplied to give the following inputs:

Data Entry - Decimal or hex entry 0-9, A-F.

Control - Examine or modify:

* Program counter
* Accumulators (0, 1, 2, 3)
* Stack
* Flag registers
* Current location
* Go to next (+1 in sequence)
* Go to last (-1 in sequence)
* Open/close register or memory

Hex Calculator

* + Hex addition
* - Hex substraction
* = Equal, display result
* Decimal to hex conversion

Debug

* Scan for value
* Scan for value with mask
* Scan memory
* Breakpoint (0-9)

Operate

* Single step program
* Run, begin execution of program
* Cancel, reset to prompt
* Executive restart, Halt CPU but do not reset
* Initialize, reset

o PACER is modular and may be expanded in many ways:

* TTY interface & line assembler
* Memory board - RAM
* Memory board - PROM/ROM
* Cassette interface
* CRT interface
* ASCII keyboard
* Bit interface
* Prototype card

1.2

CASE DIMENSIONS

13½ "	Wide
19"	Long
7"	Overall Height

1.3 PACE FEATURE LIST

I. PHYSICAL

A. Control console has a 37-key keyboard which provides complete "control panel" functions as well as a set of "debug" functions to facilitate program development.

B. Communication with the operator is via two 4-character displays that have full alphanumeric capability to reduce operator guesswork.

II. OPERATIONAL

A. The system provides the full set of normal "control panel" functions provided in larger computer systems. The individual functions are as follows:

1. Execute a program starting at a specified address (RUN).

2. Execute a program one instruction at a time starting at a specified address (STEP). After each program step, the address of the next instruction to be executed is displayed. The program is advanced one instruction with each entry of the STEP key.

3. Examine and/or modify the contents of any of the 16 computer registers (PC, FL, AØ → A3, SØ → S9). The registers are "opened" for examination or modification by either a single (PC and FL) or double (AX and SX) key entry. An open register may be optionally modified by simply keying in a new value. In either case, the register must be "closed" prior to opening another register. The close function is accomplished by a single-key entry. In addition to the computer registers, 12 pseudo-registers (described later) may also be examined and/or modified in this manner.

4. Examine and/or modify the contents of any memory location. Memory locations are "opened" by keying in the memory address. They may be modified and/or closed as described above for registers.

5. Stop a program that has "run away" or entered an "infinite loop", and return control to the operator. Two keys are provided: INIT which will reinitialize the system and clear all active registers, and RESTART "RS" which will halt the program and return control without altering the state of the system. In either case memory is left unaltered.

B. In addition to the normal "control panel" functions described in Section A, the following additional features are also provided:

1. Scan upwards in memory, starting at a specified address, unti a location is found having a specified content, and then automatically open that location for examination and/or modification as described in Section A-4. The location can be optionally modified and then closed with a special key "SC" that will automatically resume the scan until the next location is found. The scan is controlled by two pseudo registers VL and MK. VL contains the value to be searched for, while MK contains a mask that indicates which bits are to be compared. Both VL and MK may be examined and modified as described in Section A-3. The scan function can also be performed on the 16 in. computer registers and/or 12 pseudo registers (VL, MK and VØ thru V9).

2. In addition to the normal close function, which allows the user to manually specify the next register or memory location, and the SCAN close function described above; three additional automatic close functions are provided, all by single-key entries.

   a. Close the current location and automatically open the next sequential location (i.e., the current location +1).

   b. Close the current location and automatically open the previous sequential location (i.e., the current location -1).

   c. Close the current location and automatically open the memory location specified by the <u>contents</u> of the current location.

When the automatic close functions are used on the registers, the following order is assumed:

Relative Location	Ø	1	2 → 5	6 → 15	16	17	18 → 27
Register	PC	FL	AØ→ A3	SØ → S9	VL	MK	BØ → B9

NOTE: Pseudo registers VL, MK and BØ → B9 are described in Section C-2.

C. The set of special "debug" functions are as follows:

1. HALT instructions return control to the operator. As a result the operator may insert HALT instructions at points within his program where he wishes to manually examine the contents of registers or memory before continuing execution.

2. As an alternative to HALT instructions the operator may define up to 10 "breakpoints" where the system must automatically return control. The breakpoints function as HALTS but do not require the operator to modify his program. The breakpoints are specified by the 10 pseudo registers BØ → B9. A breakpoint is specified by setting the contents of a BX register to FFFF, thus FFFF may not be used as a break address.

3. Numeric values may be entered in either decimal or hexadecimal. The current numeric entry mode is indicated by the display and may be toggled between decimal or hexadecimal with a single-key entry. After entry, numeric values are always converted and displayed in hexadecimal, thereby allowing rapid decimal to hexadecimal conversion.

4. A two-function (+ and -) calculator capability is also provided, which may be used either for simple decimal or hexidecimal addition and subtraction (with the result displayed in hexadecimal) or for computing memory addresses and/or modifying register/memory locations. The user may include the current memory address in a computation if memory is being examined by entering a single key.

2.0 WHAT THE HEX IS GOING ON?

People type computers have been taught from childhood to recognize and manipulate freely a number system called decimal or base 10. This system uses 10 symbols to represent the values or numbers. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. We use combinations of these to form other numbers. Each number or digit position is assigned a value equal to its position in the number sequence. For example, the number 12,045.

Position No.	4	3	2	1	0				
	1	2	0	4	5				
						$= 5 \times 10^0$	=	5	
						$= 4 \times 10^1$	=	40	
						$= 0 \times 10^2$	=	000	
						$= 2 \times 10^3$	=	2,000	
						$= 1 \times 10^4$	=	10,000	
								12,045	

10 is the base value of the number system and 0, 1, 2, 3, 4 is the position or weighted value.

Computers being the simple machines that they are use a base 2 numbering system and use only zeros and ones to represent a value. By using groups of ones and zeros and assigning values to the bit positions, large numbers may be represented. The computer looks at the groups of ones and zeros and determines its value. The least significant bit would have a value of 2^0, the next bit would be 2^1, then 2^2, etc. Let's use a group of 5 bits and assign bit 0 as the least significant bit.

Bit No.							
0		1		1×2^0		1	
1		0		0×2^1		0	
2	=	1	=	1×2^2	=	4	
3		0		0×2^3		0	
4		1		1×2^4		16	
						21^{10}	

21 is the sum of the value of the bit positions.

It can also be seen that by using larger groups of bits, larger numbers may be represented. An eight bit computer which can handle eight bit positions in parallel can represent numbers from 0 to 255^{10}.

All Bits Equal 0						All Bits Equal 1					
Bit No.						Bit No.					
0	0	$0x2^0$	0			0	1	$1x2^0$	1		
1	0	$0x2^1$	0			1	1	$1x2^1$	2		
2	0	$0x2^2$	0			2	1	$1x2^2$	4		
3 =	0	= $0x2^3$	= 0			3 =	1	= $1x2^3$	= 8	=	255
4	0	$0x2^4$	0			4	1	$1x2^4$	16		
5	0	$0x2^5$	0			5	1	$1x2^5$	32		
6	0	$0x2^6$	0			6	1	$1x2^6$	64		
7	0	$0x2^7$	0			7	1	$1x2^7$	128		

A computer which has 16 bit positions may represent numbers with values from 0 to 65,535.

Another consideration in computers is the representation of not just numbers, but both positive and negative values. This may be accomplished by assigning one of the bits in a group as a plus/minus indicator. The normal method is to assign the most significant bit position to this task. If it is a logic 0, then the value is positive. If it is a logic 1, then the value is minus. Assuming a maximum group of eight bits and using the eighth position as the sign we may represent the following numbers:

Bit No.				
0	1	$1x2^0$		1
1	1	$1x2^1$		2
2	1	$1x2^2$		4
3	1	= $1x2^3$	=	8
4	1	$1x2^4$		16
5	1	$1x2^5$		32
6	1	$1x2^6$		64
sign bit 7	0	= +		+128

If bit 7 is equal to a 1, then the above number would be a negative or -127. It should be noted that by using the most significant bit for the sign, the maximum numbers that may be represented is only ±127. In a 16 bit computer this number would be ±32,767.

The human type being has difficulty in visually converting all those "1"'s and "0"'s to their represented value. Due to this, other methods of representing or reading these numbers have been implemented.

2.1 BCD or Binary Coded Decimal:

BCD uses groups of four binary bits or positions and only uses those combinations which add up to 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. For example:

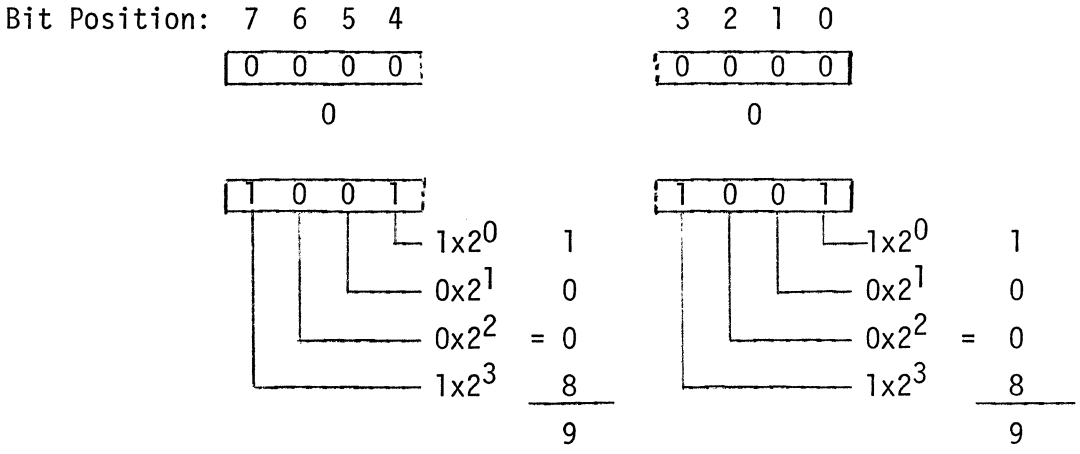
Bit	3	2	1	0		
	0	0	0	0	=	0
	0	0	0	1	=	1
	0	0	1	0	=	2
	0	0	1	1	=	3
	0	1	0	0	=	4
	0	1	0	1	=	5
	0	1	1	0	=	6
	0	1	1	1	=	7
	1	0	0	0	=	8
	1	0	0	1	=	9

The other binary combinations possible in the four bit positions are not allowed in the BCD method.

1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

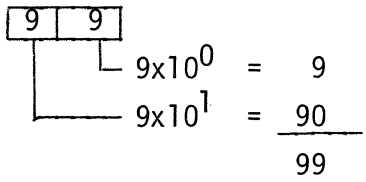
} Not Valid

In an eight bit computer, the decimal numbers of 00 thru 99 may be represented.



Note that the binary weighting system repeats for each four bit group.

This is then compensated for by applying the decimal or base 10 rules to the converted numbers.



By only having to weigh up to four binary bits a person can easily become efficient at converting binary to decimal and decimal to binary numbers.

The maximum numbers therefore that can be represented in an eight bit machine is then only 99_{10} in decimal versus 255_{10} in binary.

As can be seen, the efficiency of the computer is restricted due to the illegal combination in each four bit group. Another representation of binary numbers allows for all combinations of the four bit groups. This method is called hexidecimal.

2.2 Hexidecimal or HEX:

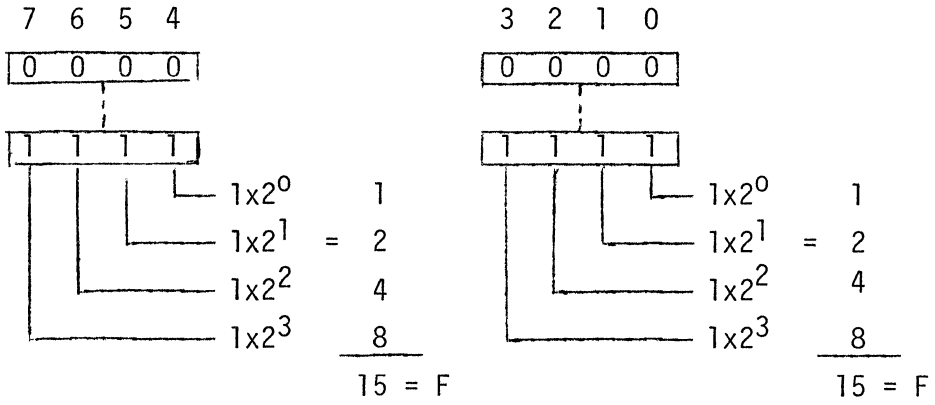
Hex uses a numbering system of base 16. Hex allows for all combinations of the four bit binary groups as follows:

Bit Position: 3 2 1 0	Binary	Hex Symbol
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	10	A
1 0 1 1	11	B
1 1 0 0	12	C
1 1 0 1	13	D
1 1 1 0	14	E
1 1 1 1	15	F

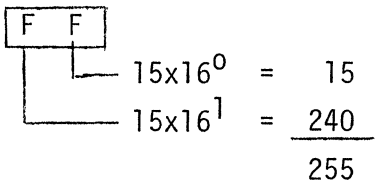
The ALPHA Notation A - F are used to allow for a single character representation of the four bit group without duplication.

With Hex we can now represent all 16 combinations of binary weights possible in a group of four bit positions.

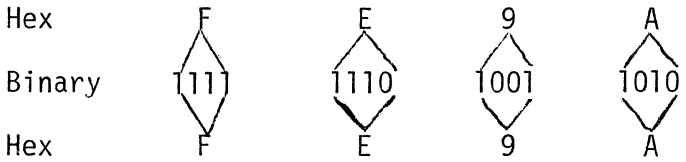
An eight bit computer can then represent the numbers 00 thru FF, which is equivalent to binary 0 thru 255.



Applying the same rules as for decimal, only using the base 16 instead of base 10.



As can be seen, binary numbers, no matter what the number of positions, can easily be converted by simply dividing them up into groups of 4 bits. For example, in a 16 bit computer:



Also, it becomes obvious that by using a hex symbol to represent an equivalent for 4 binary bits, it takes a lot less printed symbols. Most all computer documentation is now using the Hexidecimal representation of computer code.

2.3 Positive and Negative Numbers:

In Hex the method for representing positive and negative numbers is the same as for binary. The most significant bit of the most significant group is set to a zero for positive and a one for negative.

If we have four groups of 4 bits each as in a 16 bit computer, we could have:

1 Sign Bit

This number is equivalent to a positive (+) 32,767.

By making the most significant bit a logic 1, then the number becomes:

1 Sign Bit

This number is equivalent to a minus (-) 32,767.

3.0 DISPLAY DEFINITIONS

System has been initialized, either by turning on the power switch or by depressing the INIT key on the keyboard

System is in the "Prompt" mode - waiting to accept the next mode of operation

Decimal to Hexadecimal conversion mode - waiting for next mode of operation

An illegal function was attempted

Register range or memory range was exceeded

4.0 PACER OPERATOR INSTRUCTIONS

Upon initialization, the lefthand and righthand displays appear as shown below:

The operator can at this time perform any function he wishes, i.e., scan memory, look at processor status, enter data, modify data, read entire program or just bits of it, etc.

4.1 PROGRAM COUNTER (PC)

To look at the program counter (PC), depress "PC" on the keyboard, and read:

To modify the contents of the PC, the location desired is typed in. The display would read, for example:

Key in "2" 

Key in "3" 

Key in "4" 

Then read the display

Then the "CR" key is depressed on the keyboard, and the display reads:

The display must be in this position so that the operator is assured his data has been registered in the PC, and also in order to proceed to next function.

4.2 STATUS FLAG REGISTER (FL)

To view the status register, the operator depresses "FL":

To modify same, type in the desired data on the keyboard. The lefthand display will still remain the same. For example, "1, 2, F, D" are keyed in. The righthand display would then read:

Then the "CR" key is depressed and the data is at this time officially entered in the status flag register (FL). The display now will show:

4.3 PROCESSOR REGISTERS AØ, A1, A2 and A3

In order to read the registers, depress first "A", followed by the number of the register the operator desires (Ø, 1, 2, or 3). An example of what the display would show:

Then,

To modify data in the registers, the desired data is then typed in on the keyboard. The display will then read, for example:

Then, depress the "CR" key to complete the function. Display will then read:

After any given register is loaded, others may be loaded also by repeating the procedure using appropriate additional information.

4.4 PROCESSOR PUSHDOWN STACK (S)

Any level of the stack may be read by depressing "S" and the level of the stack the operator wishes to read (0 through 9). An example:

The "F's" show that no data has been entered thus far. Then, enter data desired, such as:

Then depress the "CR" key and read:

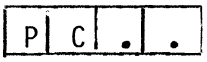
To re-read or verify the data in that location immediately following, again depress the "CR" key and read contents. Then, again, depress "CR" and return display to:

4.5 AUTOMATIC REGISTER ADVANCE

Another way all of the processor registers may be read or altered is to depress either the "↑" key or the "↓" key. The "↑" key will advance the display through the registers one after another for each depression. Data may be read or altered for a given register when that particular register is shown on the display.

The "CR" key need not be depressed following the entry, but instead, the "↑" key is depressed. This will also cause the display to advance to the next register. (See information on Page 17)

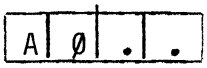
The "↓" key is like the "↑" key in that when depressed, it will display each of the registers so that data may be read or altered in the same way as the "↑", except that it is depressed when there is a need to REVERSE the order of registers displayed, i.e., A3, A2, A1, AØ, etc.



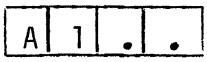
(The data in each register will appear on the right-hand display when that register is displayed.)

Depress "↑"

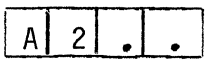
Depress "↑"



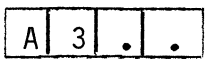
Depress "↑"



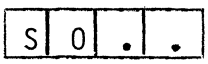
Depress "↑"



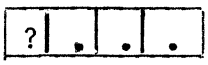
Depress "↑"

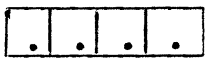


Depress "↑"



and so on through all the registers. Then when at "B9" the "↑" key is depressed, and the display will read:





4.6 SCAN VALUE

This feature enables the operator to read certain desired data already entered in the system memory by depressing the Value key "VL" on the keyboard. If it is desirable to know which addresses contain particular information, the Value key (VL) is depressed and then that desired information is entered.

The address that is the desired starting place for the scan is entered and then the Scan (SC) key is depressed. This will cause the machine to search for the next address that has that same value, and the display will then show that address and its data. The Scan key (SC) is depressed repeatedly and each address containing the scan value is read each time on the display until the entire

memory has been scanned. Example:

Depress "VL" and read display:

Enter the data desired,

Then depress "CR" and read:

Enter the starting address for the scan:

Now depress the Scan key (SC):

This is the first address, after the starting address, at which this data was found. At this time data can be modified by keying in the desired data. (It is automatically entered when the "SC" key is again depressed).

Depress Scan key (SC) again and read the display for the next address containing the specified value:

Depress Scan key (SC) repeatedly until the entire memory is scanned.

When the Pacer is scanning memory and there are no addresses with the data it is searching for, the display is blank until it searches all addresses. This takes approximately 25 seconds to scan 65,000 addresses.

Another feature on the PACER is: If the operator has data entered only in the first sections of memory and not all addresses are occupied, and the operator does not wish to wait until scanning is complete, he may depress the Cancel key (CANCEL) on the keyboard and automatically terminate the scan.

When scanning of the memory is complete, display reads:

4.7 MASK VALUE (MK)

This feature enables the operator to scan a certain digit or digits in the display, "masking out" the digit(s) in memory that are not desired to be read. Example: Depress the Value key (VL) and read display:

Then enter the desired data to be scanned:

Depress the "CR" key:

Now depress Mask key (MK) and read display:

At this point, enter either zeros or "F's" into each digit in order to designate which digit(s) are to be scanned and which are not. The digits desired to be masked out are entered as zeros. The digit(s) that are to be searched are entered as "F", meaning "enable". For example:

Depress the "CR " key on the keyboard:

Enter the desired starting address for the masked scan:

Now depress the Scan key (SC) to begin the scan and read:

The display will not only show the digit which had been enabled, but rather the entire contents of that address. Individual bits can be scanned for by setting the appropriate mask, i.e., if bits 1 and 2 are being scanned for, the mask would be "6".

4.8 MEMORY LOCATIONS

To examine any memory location, enter the address that is desired. Example:

and then depress the "CR" key

To modify the contents of the memory location, the desired new value should be typed in over the displayed data, such as:

Then depress "CR"

To re-read or verify the data at this address immediately following completion of this function, again depress the "CR" key and read display:

Then, again, depress "CR"

4.9 AUTOMATIC MEMORY ADVANCE

As with reading and/or altering of registers automatically, all that need be done to read or alter memory locations is to either depress the "↑" key or the "↓" key repeatedly.

The data at a certain address can be modified by keying in the new data and then depressing the "↑" key to: (a) register the new data, and (b) advance the display to the next address. Or, in the case of scanning in reverse, depress the "↓" key in lieu of the "↑" key.

4.10 SINGLE STEP

Enter the desired starting address:

Depress the single-step key. The instruction at the starting address is executed and the display shows the next address to be executed:

4.11 HALT

If a halt instruction is in the program when it is executed, the display will read:

4.12 BREAKPOINTS

The following is an example of a program to explain the usage of breakpoints.

Address	Mnemonic	Hex Code
0	JMP +1	1900
1	JMP +1	1900
2	JMP +1	1900
3	JMP +1	1900
4	JMP +1	1900
5	JMP -6	1800

To load memory, key in the starting address:

Depress the "CR" key:

Then key in the first instruction:

Depress the "↑" key:

Key in the rest of the program in the same way:

Depress the breakpoint key "B" on the keyboard:

Key in the first breakpoint to be set (0-9). Key in BØ and the display will read:

The display will read FFFF if no breakpoint had previously been set.

Key in the desired breakpoint address, such as (3):

Depress the "CR" key to enter the address for that particular breakpoint.

Any of the remaining breakpoints can be set in the same way. Key in, for example, (B6):

Then key in the breakpoint address (5) and the display would read:

Depress the "CR" key:

Now, key in the starting address for execution of the program:

Depress the single step key (STEP):

Depress 'STEP" key:

Depress "STEP" key:

Depress "STEP" key:

Depress "STEP" key:

Depress "STEP" key:

The breakpoint address will be displayed in "RUN" mode the same as it was in the "single" step mode.

Depress the "RUN" key.

Depress the "RUN" key again:

In either the "single step" or "run" mode it is possible to examine and/or modify any of the registers or memory locations before continuing program execution.

To examine registers AØ and A1, depress the register "A" key:

Key in register "Ø":

Depress the "CR" key:

Depress the "↑" key:

Depress the "RN" key:

To remove the breakpoints, depress the breakpoint key (B):

Key in the first breakpoint that was set (0):

Key in "FFFF" to remove the breakpoint:

Depress the "↑" key until the next breakpoint that needs to be removed is displayed. Then remove it in the same way.

Key in the starting address:

Depress the "RN" key:

The display is now blank because the program is being executed. Program execution can be terminated in one of two ways:

1. Depress the initialization "INIT" key. The "INIT" key will clear all of the registers and, therefore, the status of the system is not retained.
2. Depress the restart "RESTART" key. System status is not cleared, therefore, it is possible to then look at all of the registers, etc.

Depress the restart "RESTART" key:

PACE INSTRUCTION SET

ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then right.

Mnemonic Assembler Code		AC0	AC1	AC2	AC3	BASE PAGE (XX)	PC REL (XX+PC)	AC2 REL (XX+AC2)	AC3 REL (XX+AC3)								
HALT	0000																Halt
CFR r		0400	0500	0600	0700												Copy flags to register
CRF r		0800	0900	0A00	0B00												Copy register to flags
PUSHF	0C00																Push flags onto stack
PULLF	1000																Pull stack into flags
JSR disp(xr)						14XX	15XX	16XX	17XX								Jump to subroutine; XX = ±127; push PC onto stack
JMP disp(xr)						18XX	19XX	1AXX	1BXX								Jump; XX = ±127
XCHRS r		1C00	1D00	1E00	1F00												Exchange register and stack
ROL r,n,l		20XX	21XX	22XX	23XX												Rotate register left
ROR r,n,l		24XX	25XX	26XX	27XX												Rotate register right
SHL r,n,l		28XX	29XX	2AXX	2BXX												Shift left
SHR r,n,l		2CXX	2DXX	2EXX	2FXX												Shift right

Bit 1 = 1 include link bit
Bit 2 = 2 shift count
Bits 2-7 = N = shift count

fc		NOT USED	IE1	IE2	IE3	IE4	IE5	OVF	CRY	LINK	IEN	BYTE	F11	F12	F13	F14	NOT USED	
PFLG fc		3000	3100	3200	3300	3400	3500	3600	3700	3800	3900	3A00	3B00	3C00	3D00	3E00	3F00	Pulse or reset flag
SFLG fc		3080	3180	3280	3380	3480	3580	3680	3780	3880	3980	3A80	3B80	3C80	3D80	3E80	3F80	Set flag

cc		STACK Full	AC0 = 0	AC0 Bit15=0	AC0 Bit0=1	AC0 Bit1=1	AC0 ≠0	AC0 Bit2=1	CONT	LINK	IEN	CRY	AC0 Bit 15=0	OVF	JC13	JC14	JC15	
BOC cc,disp		40XX	41XX	42XX	43XX	44XX	45XX	46XX	47XX	48XX	49XX	4AXX	4BXX	4CXX	4DXX	4EXX	4FXX	Branch on condition (PC relative) XX = ±127

		AC0	AC1	AC2	AC3												
LI r, disp		50XX	51XX	52XX	53XX												Load immediate; load register with XX; XX = data Bit 7 of XX extends to Bits 8-15 of register

sr / dr		AC0 AC0	AC1 AC0	AC2 AC0	AC3 AC0	AC0 AC1	AC1 AC1	AC2 AC1	AC3 AC1	AC0 AC2	AC1 AC2	AC2 AC2	AC3 AC2	AC0 AC3	AC1 AC3	AC2 AC3	AC3 AC3	
RAND sr,dr		5400	5440	5480	54C0	5500	5540	5580	55C0	5600	5640	5680	56C0	5700	5740	5780	57C0	"AND" register to register; result to register (dr)
RXOR sr,dr		5800	5840	5880	58C0	5900	5940	5980	59C0	5A00	5A40	5A80	5AC0	5B00	5B40	5B80	5BC0	Exclusive "OR" register to register; result to register (dr)
RCPY sr,dr		5C00	5C40	5C80	5CC0	5D00	5D40	5D80	5DC0	5E00	5E40	5E80	5EC0	5F00	5F40	5F80	5FC0	Copy register to register

		AC0	AC1	AC2	AC3												
PUSH r		6000	6100	6200	6300												Push register onto stack
PULL r		6400	6500	6600	6700												Pull stack into stack

sr / dr		AC0 AC0	AC1 AC0	AC2 AC0	AC3 AC0	AC0 AC1	AC1 AC1	AC2 AC1	AC3 AC1	AC0 AC2	AC1 AC2	AC2 AC2	AC3 AC2	AC0 AC3	AC1 AC3	AC2 AC3	AC3 AC3	
RADD sr,dr		6800	6840	6880	68C0	6900	6940	6980	69C0	6A00	6A40	6A80	6AC0	6B00	6B40	6B80	6BC0	Add register to register; result to register (dr), overflow, and carry
RXCH sr,dr		6C00	6C40	6C80	6CC0	6D00	6D40	6D80	6DC0	6E00	6E40	6E80	6EC0	6F00	6F40	6F80	6FC0	Exchange register

		AC0	AC1	AC2	AC3												
CAI r, disp		70XX	71XX	72XX	73XX												Complement register and add XX; result to register Bit 7 of XX is extended to Bits 8-15

sr / dr		AC0 AC0	AC1 AC0	AC2 AC0	AC3 AC0	AC0 AC1	AC1 AC1	AC2 AC1	AC3 AC1	AC0 AC2	AC1 AC2	AC2 AC2	AC3 AC2	AC0 AC3	AC1 AC3	AC2 AC3	AC3 AC3	
RADC sr,dr		7400	7440	7480	74C0	7500	7540	7580	75C0	7600	7640	7680	76C0	7700	7740	7780	77C0	Add register to register plus carry; result to register (dr); overflow and carry

PACE INSTRUCTION SET

ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then right.

Mnemonic	Assembler Code		AC0	AC1	AC2	AC3	BASE PAGE XX	PC REL (XX+PC)	AC2 REL (XX+AC2)	AC3 REL (XX+AC3)	
AISZ	r, disp		78XX	79XX	7AXX	7BXX					Add XX to register; skip next instruction if result = zero; XX = ±127
RTI	disp	7CXX									Return from interrupt; add XX to top of stack and place result in PC; XX = ±127; set IEN flag
RTS	disp	80XX									Return from subroutine; add XX to top of stack and place result in PC; XX = ±127
DECA	0, disp(xr)						88XX	89XX	8AXX	8BXX	Decimal add register AC0 to contents of effective address; result to AC0, overflow and carry; address = (XX + register shown); XX = ±127
ISZ	disp(xr)						8CXX	8DXX	8EXX	8FXX	Increment contents of effective address by 1; skip next instruction if result = 0; result is in EA; use address mode shown; XX = ±127
SUBB	0, disp(xr)						90XX	91XX	92XX	93XX	Subtract contents of effective address from AC0; result to AC0; use address mode shown; XX = ±127
JSR	@ disp(xr)						94XX	95XX	96XX	97XX	Jump to subroutine indirect; push PC onto stack; final address = to contents of location (XX + register shown); XX = ±127
JMP	@ disp(xr)						98XX	99XX	9AXX	9BXX	Jump indirect; final address = to contents of location (XX + register shown); XX = ±127
SKG	0, disp(xr)						9CXX	9DXX	9EXX	9FXX	Compare AC0 with contents of location (XX + register shown); XX = ±127; skip next instruction if AC0 > (EA)
LD	0, @ disp(xr)						A0XX	A1XX	A2XX	A3XX	Load indirect; load AC0 with contents of final address; address = contents of location (XX + register shown); XX = ±127
OR	0, disp(xr)						A4XX	A5XX	A6XX	A7XX	OR AC0 with contents of location (XX + register shown); XX = ±127; result to AC0
AND	0, disp(xr)						A8XX	A9XX	AAXX	ABXX	AND AC0 with contents of location (XX + register shown); XX = ±127; result to AC0
DSZ	disp(xr)						ACXX	ADXX	AEXX	AFXX	Decrement contents of effective address by 1; skip next instruction if result = 0; result is in EA; address = (XX + register shown); XX = ±127
ST	0, @ disp(xr)						B0XX	B1XX	B2XX	B3XX	Store indirect; store AC0 into final address; address = contents of location (XX + register shown); XX = ±127
SKAZ	0, disp(xr)						B8XX	B9XX	BAXX	BBXX	AND AC0 with contents of location (XX + register shown); skip next instruction if result = 0; XX = ±127
LSEX	0, disp(xr)						BCXX	BDXX	BEXX	BFXX	Load AC0 with sign extended; Bit 7 of location (XX + register shown) is extended to AC0 8-15; Bits 0-7 are loaded to AC0 Bits 0-7; XX = ±127
LD	r, disp(xr)						C0XX	C1XX	C2XX	C3XX	Load AC0 with contents of location (XX + register shown); XX = ±127
							C4XX	C5XX	C6XX	C7XX	Load AC1 with contents of location (XX + register shown); XX = ±127
							C8XX	C9XX	CAXX	CBXX	Load AC2 with contents of location (XX + register shown); XX = ±127
							CCXX	CDXX	CEXX	CFXX	Load AC3 with contents of location (XX + register shown); XX = ±127
ST	r, disp(xr)						D0XX	D1XX	D2XX	D3XX	Store AC0 to location (XX + register shown); XX = ±127
							D4XX	D5XX	D6XX	D7XX	Store AC1 to location (XX + register shown); XX = ±127
							D8XX	D9XX	DAXX	DBXX	Store AC2 to location (XX + register shown); XX = ±127
							DCXX	DDXX	DEXX	DFXX	Store AC3 to location (XX + register shown); XX = ±127
ADD	r, disp(xr)						E0XX	E1XX	E2XX	E3XX	Add AC0 to location (XX + register shown); XX = ±127; result to AC0
							E4XX	E5XX	E6XX	E7XX	Add AC1 to location (XX + register shown); XX = ±127; result to AC1
							E8XX	E9XX	EAXX	EBXX	Add AC2 to location (XX + register shown); XX = ±127; result to AC2
							ECXX	EDXX	EEXX	EFXX	Add AC3 to location (XX + register shown); XX = ±127; result to AC3
SKNE	r, disp(xr)						F0XX	F1XX	F2XX	F3XX	Compare AC0 to location (XX + register shown); XX = ±127; if not equal skip next instruction
							F4XX	F5XX	F6XX	F7XX	Compare AC1 to location (XX + register shown); XX = ±127; if not equal skip next instruction
							F8XX	F9XX	FAXX	FBXX	Compare AC2 to location (XX + register shown); XX = ±127; if not equal skip next instruction
							FCXX	FDXX	FEXX	FFXX	Compare AC3 to location (XX + register shown); XX = ±127; if not equal skip next instruction

5.0 PACER AND YOUR HARDWARE

Pacer provides the user with many features intended to make it easy for the user to;

1) learn to operate Pacer
2) to be able to write programs
3) enter the programs into Pacer memory
4) debug the program using Pacer debug features
5) execute the debugged program
6) design and breadboard various interfaces
7) checkout the interfaces using the available card slots on the mother board
8) use components and/or sub-assembly, memory board, CPU board, etc., of Pacer in production systems

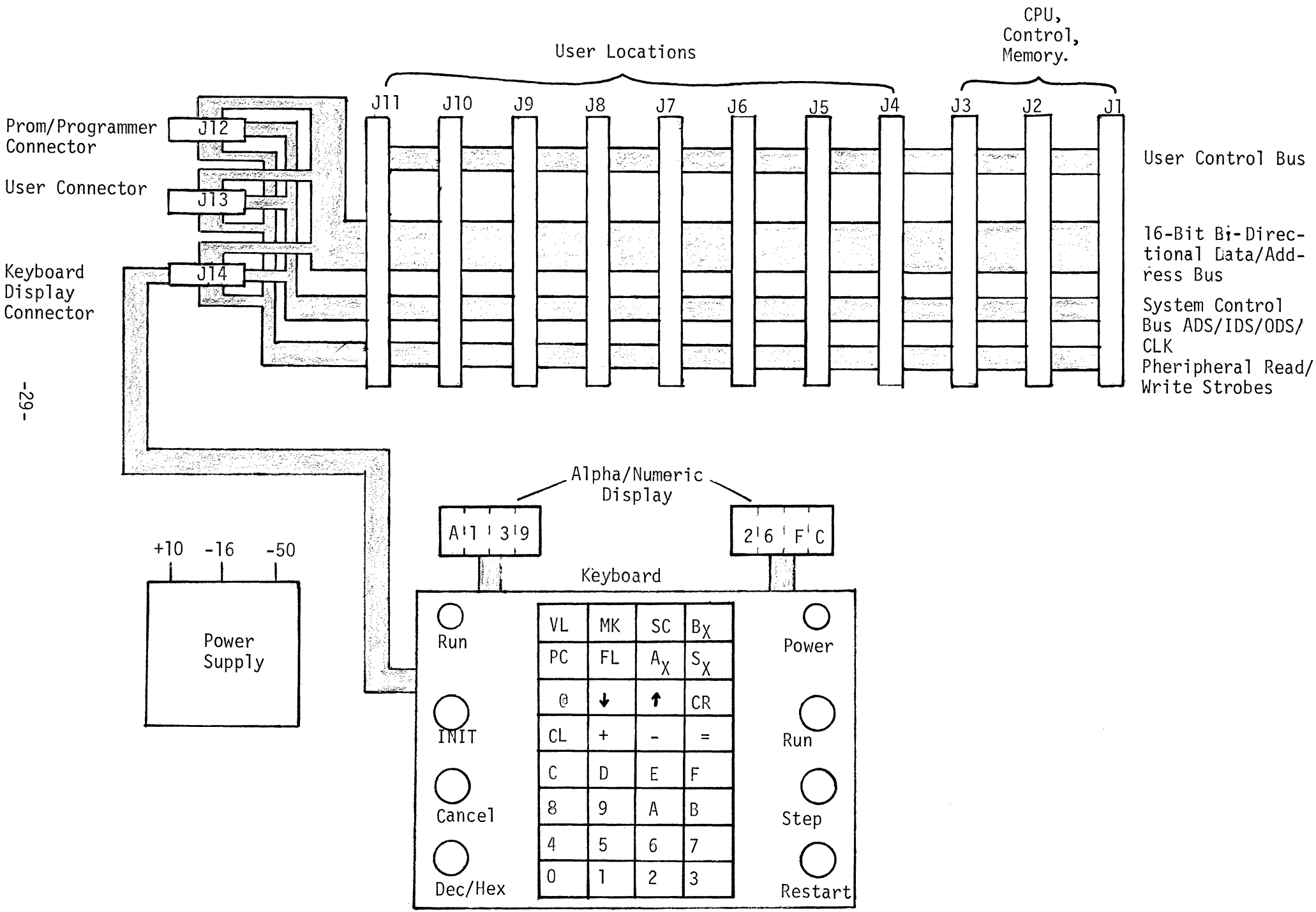
The Pacer is organized as indicated by the block diagram below. Three connector positions are used on the mother board for the CPU, Control and memory cards. The remaining eight locations are for user expansion and application development.

There are essentially four groups of signals on the back plane. These are:

- 16-Bit Bi-directional Data Bus
- System Control Bus (BADS, BIDS, BODS)
- User Control Bus (NBADS, BODS, BIDS, BCLK, Interrupt inputs, Jump conditions and Flag lines)
- Decoded Peripheral Read/Write Strobes

Besides these, there are other signals on the back plane used to interconnect the three Pacer system cards. These signals may be utilized, however, caution should be exercised to prevent system manfunctions due to overloading, etc.

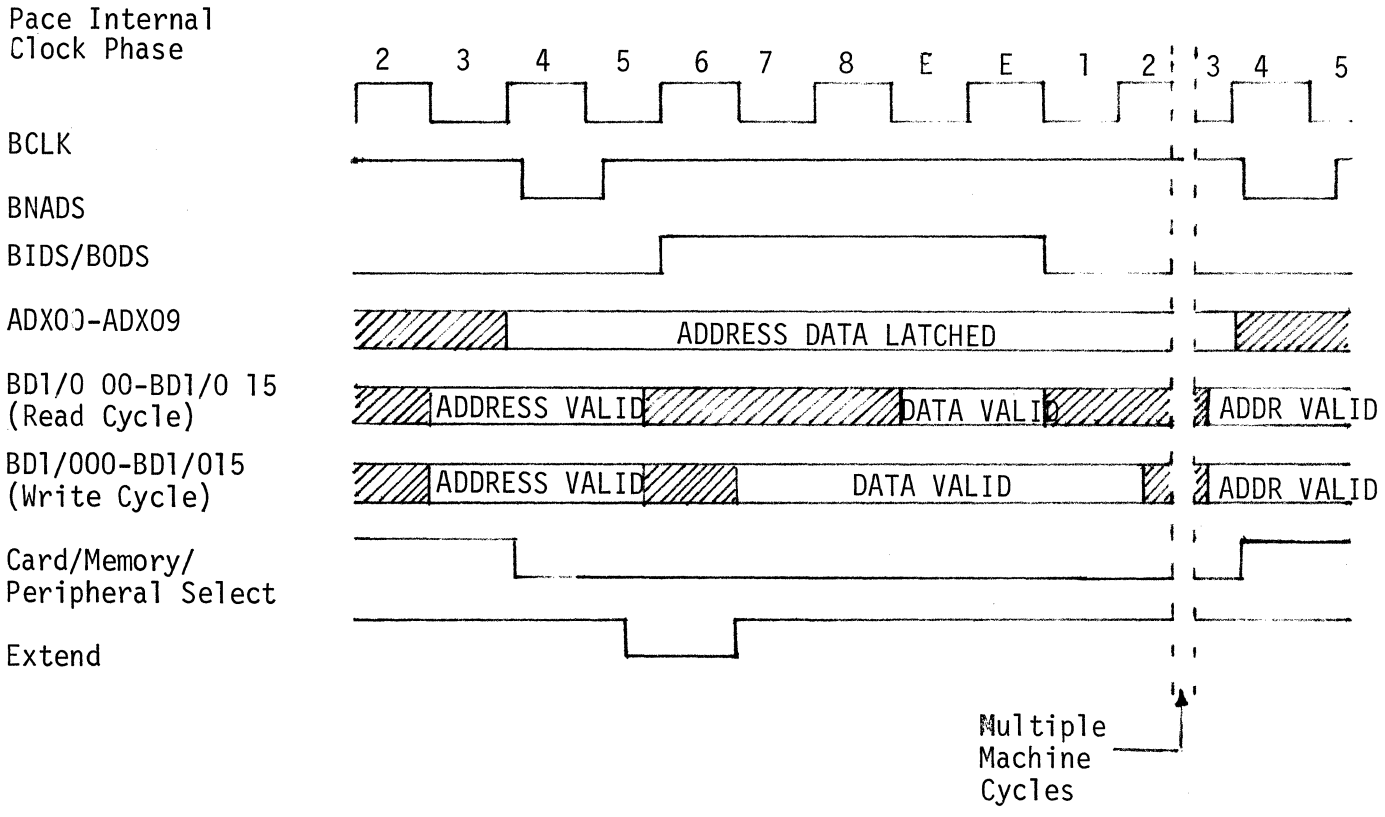
PACER BLOCK DIAGRAM



The following requirements need to be considered when attaching peripherals to the Pacer system.

1. The Pacer supplies unregulated supplies to the edge card connector. The user is required to have regulators on his application card. The prototyping cards which are available to Pacer users have the necessary area and mounting for a positive and negative regulator.

2. The Pace system communicates with peripherals and memory by first puting a 16-bit address onto the data bus. A signal, NBADS, is then present to signify that the address is valid. A comparison or decode of part or all of these 16 bits would then select the user application. Either the address bits, the decode,or comparison true must be latched at NBADS time in order to supply a static select signal until the trailing edge of BODS or BIDS.



I/O TIMING

3. Data being transfered from a user onto the data bus must be enabled only when the user logic is selected and the BIDS strobe is present. Tri-state devices should be utilized for this purpose. Facilities are provided using resister packs on the mother board if user wishes to use open collector logic or extend the back plane signals.

All of the Pace user signals are provided on the back plane. The following is a list, with description, of each of these. For further information regarding these and other aspects of Pace, the reader is requested to use the Pace Technical Description, National Semiconductor Publications No. 4200078A and other documents contained in the Pacer Document Package.

5.1 DESCRIPTIONS OF PACE HARDWIRED SIGNALS

Signal Mnemonic/Name	Description
BD00BD15/Data Bits 00-15	Input/output Data Bus Lines.
BIDS/Input Data Strobe	Pace output signal used to enable external devices so data can be placed on-line to Pace. IDS operation is as follows: 1. Following output of peripheral or memory address information from Pace, D00-D15 data line drivers assuming high-impedance state and Pace Control Logic drives IDS Signal high. 2. IDS remains high for approximately 1.5 CLK periods. (plus extended) 3. Valid input data to Pace must be present on D00-D15 Input/Output Data Bus Lines when IDS is driven low again by Control Logic after approximately 1.5-CLK-period duration.
BODS/Output Data Strobe	Pace output signal used to enable external devices to accept data output from Pace, ODS operation is as follows: 1. Following output of peripheral or memory address information from Pace, data are placed on D00-D15 Input/Output Data Bus Lines by Pace.

2. At approximately the same time that data are placed on Input/Output Data Bus, ODS Signal is driven high by Pace Control Logic to signify that output data from Pace are available to memory or peripherals.
3. ODS remains high for approximately 1.5 CLK periods.
4. Output data remains on Input/Output Data Bus after ODS is driven low again by Control Logic after approximately 1.5-CLK-period duration. Thus, ODS trailing edge can be used to clock Pace output data into External Data Latch. ODS can also be used as read/write control signal for external RAM memory elements.

NBADS/Address Data Strobe

Pace output signal used to clock address information from Pace into Address Latches. After address information is placed on Input/Output Data Bus by Pace, NADS Signal is driven low for approximately 0.5 CLK period by Pace Control Logic. NADS is active in middle of approximately 1.5 CLK periods that address information is valid. Thus, either edge of NADS can be used to clock address information.

EXTEND/Extended Data Transfer

Pace input signal used to temporarily increase time duration of data input/output transfers to accommodate accessing of slow memories or peripherals without altering CLK frequency. EXTEND Signal must be driven high at beginning of ODS or IDS. If EXTEND is held high, data-transer operation is extended by 1 CLK period. Holding EXTEND high for additional n clock periods increases data-transfer timing by n + 1 clock periods.

NINIT/Initialize

Pace input signal that initializes microprocessor functions. When NINIT is low, Pace operation is suspended and all Pace strobe signals (IDS, ODS, NADS, and so forth) are set to inactive state. After NINIT completes low-to-high transition, the following conditions are effected:

1. Pace Program Counter contents are set to zero.

2. Internal Stack Pointer (indicates last Stack level accessed) is cleared.
3. All flags and interrupt enables are set low except Level-0 Interrupt Enable which is set high. All other registers contain an arbitrary value.

NHALT/Control Panel Halt

Pace Control Logic input/output signal used for nonmaskable Level-0 Interrupt, microprocessor stall, and programmed HALT indicator output. When NHALT is applied as low input, microprocessor operation halts after completing execution of current instruction. When Halt Instruction is executed, NHALT Line is driven low by Pace Control Logic for a 7/8 duty cycle. Microprocessor can be stalled by using external open-collector driver to hold NHALT Line low for desired time duration, thereby overriding NHALT output buffer on Pace chip.

CONTIN/Continue Jump Condition

Pace Jump Condition Multiplexer input/output signal used to sense external signal through BOC Instruction. Also used to restore microprocessor operation from suspended state or cause subroutine branch to Level-0 Interrupt Service Routine (generally used to implement Control Panel functions). Driving CONTIN Input high for 4 CLK periods, minimum, causes halted microprocessor to resume operation. As output, CONTIN is driven low for approximately 3 clock periods by Pace Jump Condition Multiplexer to acknowledge that microprocessor operation is stalled. CONTIN Line must be pulsed to terminate Halt Instruction.

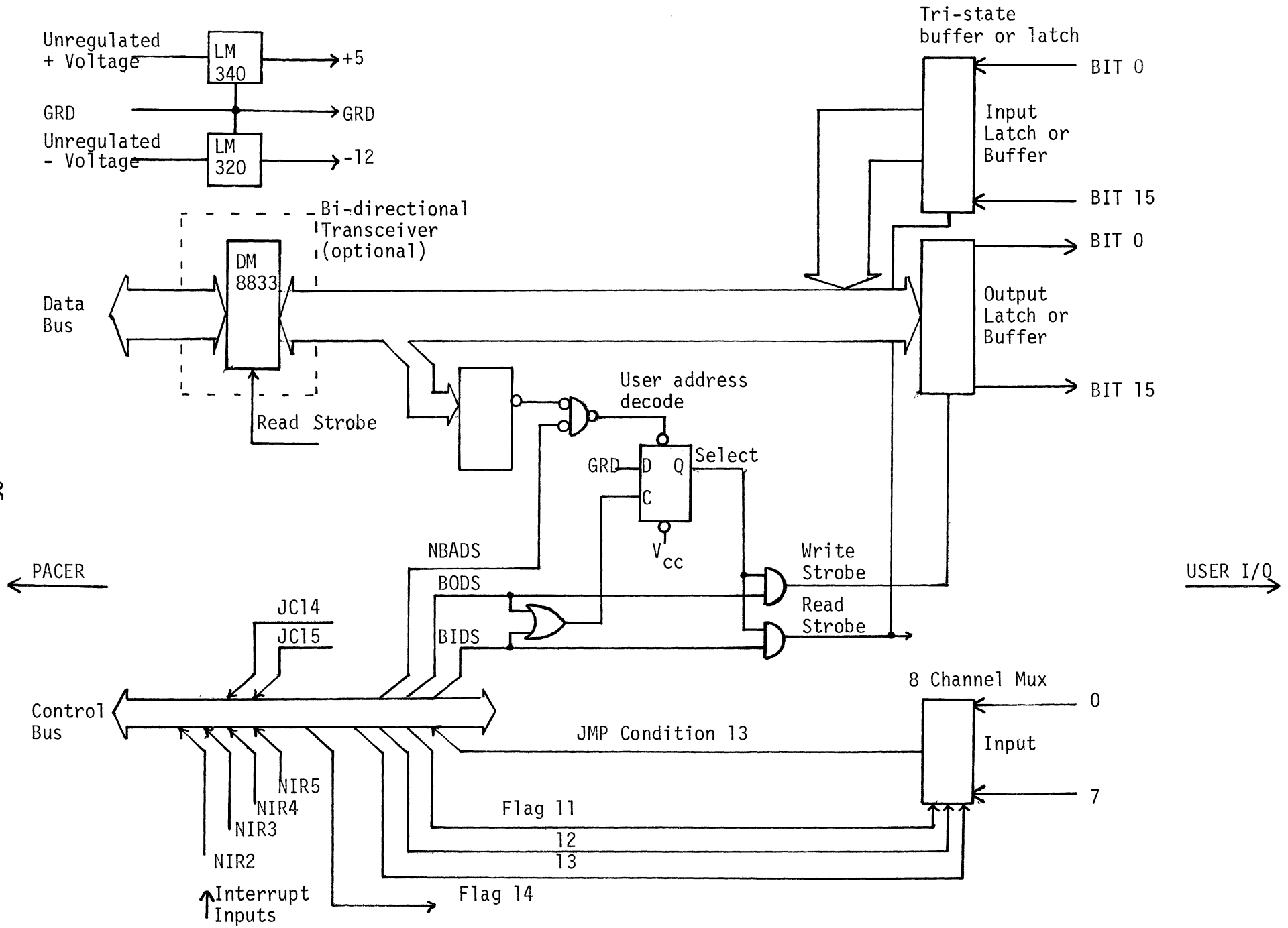
BPS/Base Page Select

Input signal to Pace Control Logic that enables one of two base-page addressing schemes to be selected. When BPS is low, first 256 words of memory constitute base page (page zero). When BPS is high, first 128 memory words and last 128 memory words constitute base page.

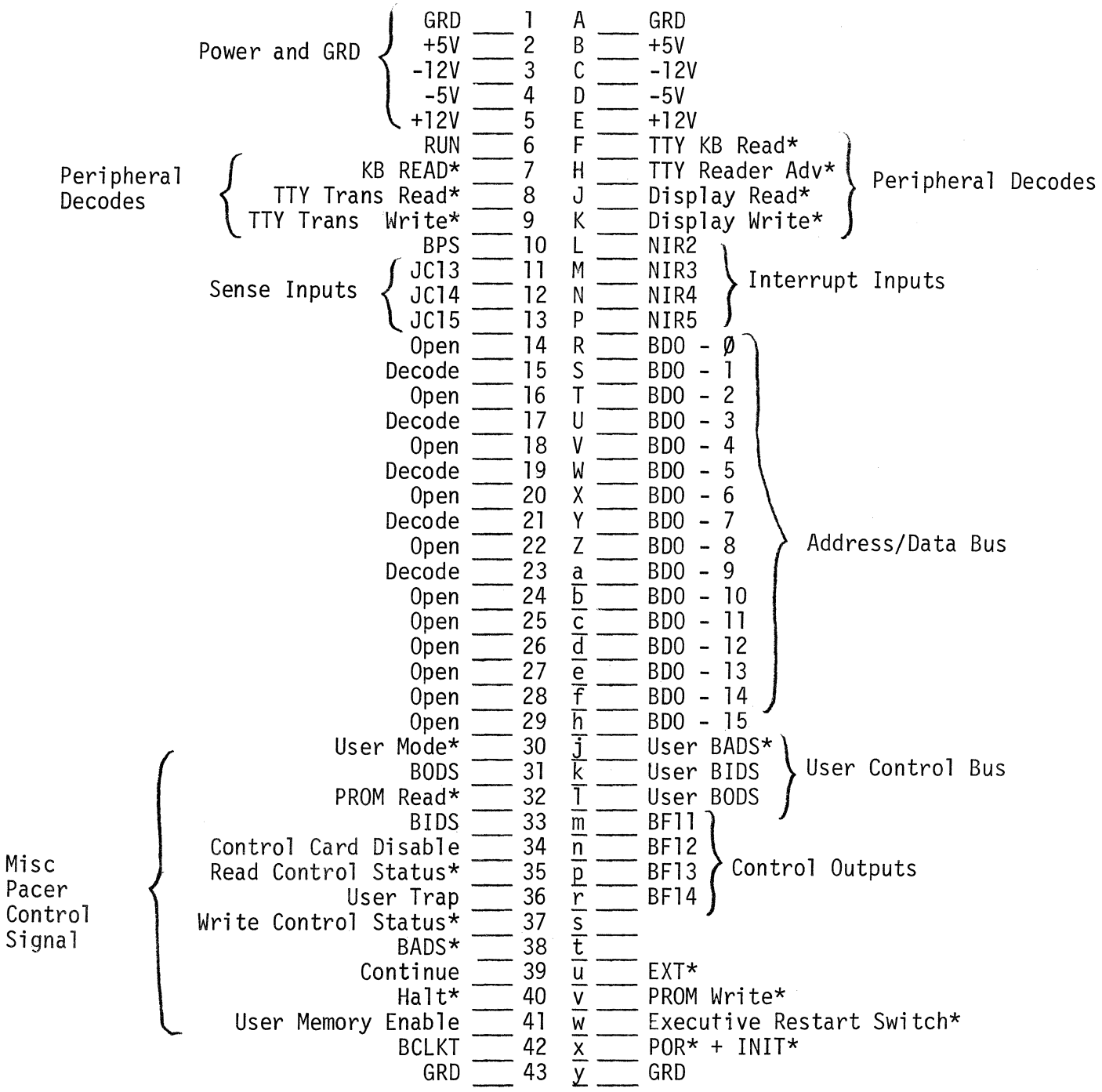
JC13, 14, 15/Jump Conditions 13, 14, and 15

User-specified branch-condition inputs to Pace Jump Condition Multiplexer. Some possible uses are testing system status and receiving serial data. When JC13, 14

	or 15 is high, Pace Branch-On Condition Instruction effects program branch if Jump Condition Input is true.
F11, 12, 13, 14/Flags 11, 12 13 and 14	Pace Status and Control Flags Register general-purpose control flag outputs. F11-14 may be used for direct control of system functions or serial data output. Individual flags may be set by Pace Set Flag Instruction and pulsed or reset by Pulse Flag Instruction. Push Flag and Pull Flag Instructions permit contents of Status and Control Flags Register to be saved on Stack during Interrupt Service Routine or subroutine execution, and then restored.
NIR2, 3, 4, 5/Interrupt Requests 2, 3, 4 and 5	Inputs to Pace Interrupt Control Logic. When NIR2, 3, 4 or 5 Input is low for 1 CLK period, minimum, corresponding internal Interrupt Request Latch is set.



USER INTERFACE TO PACER



MOTHERBOARD BUS PIN NUMBER ASSIGNMENTS

The following connectors may be inserted into the motherboard for expansion.

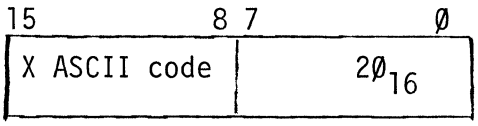
Connector Company	Part No.
Viking Industries 21001 Nordhoff St. Chatsworth, CA 91311 (213)341-4330	2V8 43/1AV5
S.A.E. 340 Martin Ave. Santa Clara, CA 95050 (408)243-9200	SA-M-43D/2-3

These connectors may be purchased through franchised Reps or Distributors in your local area. For specific Reps or Distributors in your area, please call the phone numbers listed above.

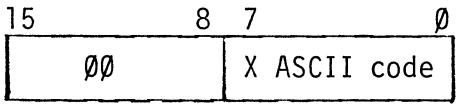
PACER PRODUCT PRICE LIST

6.1	Pacer Kit	$695
6.2	Pacer Assembled	$855
6.3	PAC-I	$175
6.4	PAC II	$199
6.5	Cassette	Future
6.6	Disc Operation System	Future
6.7	Keyboard	Future

The form 'X' is assembled as:



The form "X is assembled as:



All codes are 7-bit ASCII.

5.3 EXPRESSIONS

Simple Expression

Consist of self-defining terms combined as follows:

$$\{-\}\, t_1 \left[\begin{matrix}+\\-\\*\\/\end{matrix}\right] t_2 \left[\begin{matrix}+\\-\\*\\/\end{matrix}\right] t_3 \text{ ---- } \left[\begin{matrix}+\\-\\*\\/\end{matrix}\right] t_n$$

\+ = addition

\- = subtraction

* = unsigned multiplication (overflow is ignored)

/ = unsigned division (without rounding, divide by Ø produces an undefined result)

Examples:

-123 = -123_{10}

25/4 = 6_{10}

ØABC*5 = $2748_{10}*5 = 1374Ø_{10}$

"A+2 = $65_{10}+2 = 67_{10}$

'A'/256 = "A = 65_{10}

Complex Expressions

Same as simple expressions except user defined terms may be used.

Examples:

.+25

ABCD+22/X24

5.4 USER DEFINED TERMS

Period . is equal to the current location being examined in alter mode, or the location being listed by the list (LS)

command. Not allowed in other commands or in Alter mode while examining registers.

Symbols — as defined by the user in Alter mode. May have any 16-bit value. (See the symbol table description which follows.)

NOTE

The remaining portion of this manual is pertinent only if user has Pacer option PAC-1. This is the teletype and line assembler board. This option may be purchased and plugged into the Pacer mother board. A teletype printer with punch and reader is required to utilize all of the features of PAC-1. However, if no use of paper tape is desired, a teletype without the punch and reader is sufficient.

TABLE OF CONTENTS

1.0 • WHAT IS IT?

Pace 2 is an optional PC card to Pacer. The PC card contains a parallel TTY interface, an RS232C interface and Pace 2 ROM's. Pace 2 (like Pacer) is implemented as a special type of "Control Panel" program. Pace 2 communicates with the user via a teletype, or teletype-like CRT terminal, on a "line-at-a-time" basis.

1.1 • WHAT ARE PACE 2'S BENEFITS?

- Complete set of Control Panel functions

  * Examine or modify
    - Contents of any computer register or memory location can be examined or modified
    - Examine/modify a location with a single-key stroke (current location)
    - Examine/modify the current location and examine the next sequential location with a single-key stroke (current location +1)
    - Examine/modify the previous sequential location with a single-key stroke (current location -1)
    - Examine/modify the current location and examine the memory location specified by the contents of the current location
    - Examine/modify the contents of the calculated effective address
  * Display
    - Contents of a specified number of registers or memory location (a block of memory)

The registers/memory contents can be displayed in hexadecimal or any one of four optional formats.

- ASCII
- ASSEMBLY LANGUAGE FORMAT
- UNSIGNED DECIMAL
- SIGNED DECIMAL

* Run - Execute a program starting at a specified address

* Single Step - Execute a program one instruction at a time starting at a specified address. Following each "step", the contents of any register(s) or memory location(s) can automatically be displayed in hexadecimal or in one of the optional display formats

* Scan - Scan through the contents of the MPU's register(s) or memory location(s) starting at a specified location until a location is found having a specified content

* Scan with Mask - The scan can be specified to scan for a bit(s) digit(s) or word

* Set - Set the contents of any register(s) or memory location(s) to a specified value

* Memory Load - Load a binary formatted paper tape into the specified memory locations

* Memory Save - Saves the specified memory locations on paper tape in a binary format

1.2 • Complete set of DEBUG functions

* HALT - Stop program execution at a specified address

* Break Points - Set up to five breakpoints to "HALT" execution and display the contents of the specified register(s) and memory location(s) in hexadecimal or one of the optional formats

* Snap Points - Set up to five snap points to display the contents of the specified register(s) and memory location(s) in any format. Program execution is not halted

* List Snap/Break Points - Type out all of the specified snap/break points and their memory locations

* Reset Snap/Break Points - Reset individual snap/break points or all of them

* Decimal to Hexadecimal Conversion	-	Single-key stroke
* Hexadecimal Calculator	-	Plus(+), minus(-), multiply(x), divide(÷) and equal(=) functions
* Address Calculator in Hexadecimal	-	Current address + or - displacement

1.3 • Complete line-by-line Assembler

* Assembly Input	-	Program entry - Assembly language format - Assembler converts it to hexadecimal
* Symbols	-	Assembler allows the use of symbols and does all address assignment and referencing
* Assembly Language Output	-	Assembler converts the 16-bit memory word to assembly language format
* List Symbol Table	-	Type out the symbol table and the value associated with it (memory address or constant)
* Delete Symbol Table	-	Delete the specified symbol or clear the table

1.4 • System functions

* Test Keyboard	-	Test the keyboard to see if it is being used
* Input Data from Keyboard	-	Test the keyboard until it is used and load the character
* Output Data to the Printer	-	Output a specified character to the printer

1.5 DISPLAY DEFINITIONS

Display	Meaning
>	System has been initialized either by turning on the power switch or by depressing the INIT key on the front panel.
>	System is in the "prompt" mode waiting to accept the next input.
??	An illegal function was attempted.
?	Register range or memory range was exceeded.
RST (Memory Address)	The restart switch on Pacer was depressed and stopped execution of a user program and returned control to Pace 2.
STP (Memory Address)	Pace 2 in single-step mode.
HALT (Memory Address)	A "halt instruction" was executed.
BRK (Memory Address)	PC equals a specified break point address.
SNP (Memory Address)	PC equals a specified snap point address.
!R	A symbol definition has resulted in the re-definition of the same symbol currently in the table.
!F	Indicates a symbol definition was ignored due to a table full condition.
!L	Indicates that a five or six character symbol was encountered during a load command.

1.6 CONTROL CHARACTER DEFINITIONS

Character		Meaning
←	Rub Out Key	- Deletes last character typed
↑D	Control-D	- Used to terminate command execution
↑C	Control-C	- Used to terminate a line entry and to terminate a command
TAB	Control-I	- Spaces the carriage to the beginning of next input field
	Control-N	- Continue scan in alter mode

1.7 COMMAND SUMMARY

Character	Meaning
DS	Display memory/register in hexadecimal, ASCII or decimal
LS	List memory in assembly language format
SC	Scans memory/register and displays those addresses equal to the scan value
ST	Set memory/register to specified value
LD	Load a binary paper tape into memory
SV	Punch a paper tape to save the specified memory location
BP	Define a break point at a specified memory location
SP	Define a snap point at a specified memory location
LB	List all break/snap points
RB	Reset break/snap points
SS	Single step
RN	Run
SY	List the symbol table
DL	Delete symbols
AL	Alter mode

```
% (memory address)                  - run
% (memory address, display mode)   - single step
!SY                                - delete symbols
'                                  - scan memory
?                                  - display value
```

2.0 COMMUNICATING WITH PACE 2

Pace 2 communicates with the user via a teletype, or teletype-like CRT terminal, on a "line-at-a-time" basis. The user inputs commands, data, etc. via the teletype (TTY) keyboard. The system output or display is the TTY printout.

Pace 2 will prompt the user and then wait until a complete line of information has been entered before examining and acting upon the entry. The user tells Pace 2 that the line is complete by entering a line termination character (normally carriage RETURN). If the user detects an entry error prior to entering the termination character, he may correct it by use of the following "line editing" characters.

Character	Function
RUBOUT	Deletes one character to the left. Successive RUBOUT's will delete successive characters. RUBOUT is echoed by the system as a back arrow (←).
Control-C	Deletes the entire line and causes Pace 2 to re-prompt. Control-C is echoed as ↑C. (Note: Control-C is also used for other functions as described later).
Control-D	Similar to Control-C. Control-D is echoed as ↑D. (Note: Control-D is also used to terminate command execution as described later).

Any line containing unknown information will cause Pace 2 to output the syntax error message (??) and re-prompt the entry.

To aid in assembly language input, the operator may request automatic spacing to the next preset tabulation point by entering a Control-I (TAB). Pace 2 will output spaces and enter them into the buffer until the next tab point is reached. The tab points are six characters apart starting at Position 7 (ie., character positions 7, 13, 19, etc. within the buffer).

The line buffer within Pace 2 is 40 characters long. Any attempt to go beyond the buffer limits (either forward by character entry or backward by character delete) will cause the bell to be sounded and the entry ignored. Characters

other than the 64 printing characters and the special characters mentioned in this and later sections are ignored by Pace 2.

2.1 PACE 2 - HARDWARE FEATURES

Pace 2 (like Pacer) is implemented as a special type of "control panel" program. The ROM containing the Pace 2 operating program and the I/O addresses assigned to the operator's console are located in the 8K block from $C000_{16}$ to $DFFF_{16}$. This address space is completely locked out as far as user access is concerned. Any attempt to write in this area will be ignored while any attempt to read will return the value $FFFF_{16}$. The base page read/write memory (RWM) used by Pace 2, however, is separate from that of the user and is totally inaccessible to the user. As a result Pace 2 will remain operable even after a massive failure of the user's program.

The control panel features incorporated into the Pace 2 hardware are described as follows:

INIT	The INIT button is used to reinitialize and restart Pace 2. The program will display the message: PACE 2 and then prompt the user for the first command. All registers and the user symbol table are cleared. An INIT is automatically performed on power up.
RESTART	The RESTART button will stop execution of a user program and return control to Pace 2. Pace 2 will type the message: RST Memory Address where Memory Address is the address of the next instruction to be executed (ie., the value of PC) and program execution will cease. All registers and the user symbol table are left unaltered.
Single Step	Pace 2 is able to execute a single instruction of a user program and then regain control. This feature is used to implement Step Mode and break/snap points. In Step Mode the message: STP Memory Address

is displayed after each instruction execution where Memory Address is the value of PC. Pace 2 then waits for a keyboard entry from the operator before continuing with the next instruction. The commands which initiate Step Mode allow the operator to specify other information to be displayed after each execution step.

2.2 PACE 2 OPERATING MODES

Pace 2 operates in three basic modes: Command Mode, Alter Mode and User Mode. In Command Mode and Alter Mode the system is controlled by the operator, via Pace 2, while in ser ode the system is controlled by the user's running program. Control may be returned to Pace 2 either automatically, from the user's program, or manually by use of the RESTART or INIT buttons.

Command Mode is the primary Pace 2 operating mode and enables the user to perform such functions as examining or altering blocks of memory (or registers), loading or saving memory via papertape load modules, setting break or snap points, listing the symbol table and so forth.

Alter Mode is a secondary operating mode of Pace 2 and is entered from Command Mode. Alter mode provides the user with an interactive means of entering, modifying and debugging his programs. Individual registers or memory locations can be conveniently opened, examined or modified and closed. Instructions may be entered in assembly language format and the resulting program executed in either Single Step or User Mode (RUN). In general, Alter Mode is a highly expanded Pacer.

In User Mode Pace 2 provides the running program with a set of "system functions" that enables it to communicate with the Pace 2 operators console and return control to Pace 2 after execution is complete. The system functions are executed by Pace 2 as if they were part of the Pace instruction set.

2.3 COMMAND MODE

The following sections describe in detail the functions that may be performed in Pace 2 Command Mode. The first sections describe the terms used in the individual command descriptions which follow. The term descriptions define the parameters used in the commands and how they are specified. The command

descriptions define the function of each command in detail and are sub-categorized into command groups as follows:

1) Memory/register block display/modify commands
2) Memory load/save commands
3) Break/snap point commands
4) Program execution commands
5) Symbol table commands

Alter mode is described separately following the command descriptions.

In the term and command descriptions, the term "memory" refers to the user's RW or RO memory while the term "registers" refers to the 16 Pace operating registers, namely;

PC	Program counter
FL	Flags
AØ-A3	Accumulators Ø-3
SØ-S9	Stack locations Ø-9

All command lines are prompted by Pace 2 with the character (>) and are terminated by the operator with a RETURN (indicated by a ↵ in the definitions).

2.4 DISPLAY MODE

While hexadecimal is the normal form in which numeric values are displayed, certain commands allow the operator to specify other (optional) display modes. The optional display modes are specified by single character codes as follows:

Character	Display Mode
A	ASCII. The 16 bit value is treated as two ASCII coded bytes (bits 15 and 7 ignored) and is displayed as "X_1X_2" where X_1 is the upper byte (bits 14-8) and X_2 is the lower byte (bits 6-Ø). Any codes other than the 64 character printing set (21_{16}-$5F_{16}$) are displayed as blanks.
U	Unsigned decimal. The value is treated as a 16 bit positive integer and displayed as such in decimal (e.g. $FFFF_{16}$ is displayed as 65535).

S	Signed decimal. The value is treated as a 15 bit signed integer and displayed as such in decimal (e.g. $FFFF_{16}$ is displayed as -1).

2.5 MEMORY/REGISTER BLOCK DISPLAY/MODIFY COMMANDS

The commands in this set are used to display or modify blocks of registers or memory locations. The functions performed by these commands are also available in Alter Mode, but on a single register, one-memory-location-at-a time basis. Any command in this set may be prematurely terminated by entering either a Control-C or Control-D.

ABBREVIATIONS USED IN THE COMMAND DESCRIPTIONS

ra	Register Address
rr	Register Address Range
ri	Implied Register Address Range
ma	Memory Address
mr	Memory Address Range
ar	Any type of register address (ra, rr, ri)
am	Any type of memory address (ma or mr)*
aa	Any type of address (ar or am)*
sy	Any user symbol
se	Simple expression
ce	Complex expression
ae	Any expression
dm	Display mode
ss	Snap specification
al	Assembly language
ti	A 1-6 character title symbol
n	A numeric value
b	One or more blanks (spaces) or Control-I (TAB)
↵	A RETURN (CR)

*if aa or am is specified as optional, the implied memory address range is selected (0-$FFFF_{16}$) whenever the address is omitted.

Upper case letters or special characters ($, #, etc.) indicate actual operator entries while lower case letters indicate entries to be determined by the operator (ie., $RØ implies $RØ while $Rn implies $RØ, $R1, $R2 or $R3, etc.). Terms of the form description are the same as lower case letters with the meaning specified by the description. "Curly" brackets, { and }, are used to denote optional entries while "straight" brackets, [and], are used to denote a set of entries of which one must be selected. For example:

X{Y}Z	could be entered as either XZ or XYZ
A[X/Y]B	must be entered as either AXB or AYB

2.5.1 DISPLAY MEMORY/REGISTER COMMAND

The display command will print out the specified memory or register address and display their contents in the specified format.

1) Hexadecimal
2) ASCII
3) Signed decimal
4) Unsigned decimal

The following format defines the display command:

DS{b{aa}{/dm}}

Command Code	Address	/	Display Mode	
			NONE	HEXADECIMAL
DS	Any type of memory or register address		A	ASII
			U	Unsigned decimal
			S	Signed decimal

Register addresses can be specified by the following forms:

Single Register:	$PC	Program counter
	$FL	Status register flags
	$AØ	Accumulator Ø
	⋮	⋮
	$A3	Accumulator 3

	$SØ	Stack level Ø (top of stack)
	$S9	Stack level 9
Register Groups:	$	All registers (ie., PC thru S9)
	$P	PC and FL
	$A	AØ thru A3
	$S	SØ thru S9
Register Ranges:	An_1$ -n_2 or	register n_1 to and including
	Sn_1$ -n_2	n_2 where $n_1 < n_2$
Example:	$AØ-2	Accumulator 0 thru 2
	$S4-7	Stack level 4 thru 7
	An_1$ > or	Register An_1$ thru 3 or Sn_2$ thru
	Sn_2$ >	9 where $n_1 < 3$ and $n_2 < 9$
Example:	$A2 >	Accumulator 2 and 3
	$S7 >	Stack level 7 thru 9

Memory addresses are specified in the following way:

Decimal: Any number not preceded by an Ø

Example:
123 = $123_{1Ø}$
6 = $6_{1Ø}$
28 = $28_{1Ø}$
etc.

Hexadecimal: Any number preceded by an Ø

Example:
Ø123 = 123_{16}
Ø6 = 6_{16}
Ø28 = 28_{16}
ØlFA = $1FA_{16}$
etc.

Single Address:	n	Memory location
Example:	ØlØ	Memory location $1Ø_{16}$

Memory Ranges:	n_1 - n_2	Memory locations n_1 through and including n_2 where $n_1 < n_2$.
Example:	Ø 1Ø - Ø1ØØ	Memory address $1Ø_{16}$ thru $1ØØ_{16}$.
	Ø - 255	Memory address Ø thru FF_{16}.

In certain commands if the memory address is omitted, the range Ø - ØFFFF is implied.

DISPLAY COMMAND EXAMPLES

Note: In all examples operator-typed information is underlined.
Pace 2 typed information is not underlined.

Display the contents of the PC:

```
> DS $PC (CR)
 $PC   ØØØØØ
>
```

Display the contents of the sixth level of the stack:

```
> DS $S6 (CR)
 $S6   ØFFFF
>
```

Display the contents of all the CPU registers:

```
> DS $ (CR)
 $PC   ØØØØØ
 $FL   ØØØØØ
 $AØ   ØØØØØ ØØØØØ ØØØØØ ØØØØØ
 $SØ   ØFFFF ØFFFF ØFFFF ØFFFF
 $S4   ØFFFF ØFFFF ØFFFF ØFFFF
 $S8   ØFFFF ØFFFF
>
```

Display the contents of stack levels 4 thru 7:

```
> DS $S4-7 (CR)
 $S4   ØFFFF ØFFFF ØFFFF ØFFFF
>
```

Display the contents of memory locations Ø thru E_{16}:

```
> DS Ø-ØE (CR)
 ØØØØØ   ØØØØØ ØØØØØ ØØØØØ ØØØØØ                    HEXADECIMAL
>
```

```
ØØØØ4   Ø4146 Ø2ØØØ ØØØØØ Ø4434
ØØØØ8   ØØØØØ ØØ1ØØ ØØØØØ Ø1ØØØ
ØØØØC   Ø8ØØØ Ø4646 Ø2ØØØ
>
>DS Ø-ØE/A (CR)
ØØØØØ  '  ' '  ' '  ' '  '                    ASCII
ØØØØ4  'AF' '  ' '  ' 'D4'
ØØØØ8  '  ' '  ' '  ' '  '
ØØØØC  '  ' 'FF' '  '
>
>DS Ø-ØE/S (CR)
ØØØØØ       Ø      Ø        Ø       Ø          SIGNED DECIMAL
ØØØØ4   16710   8192        Ø   17460
ØØØØ8       Ø    256        Ø    4096
ØØØØC  -32768  17990     8192
>
>DS Ø-ØE/U (CR)
ØØØØØ      Ø     Ø       Ø      Ø              UNSIGNED DECIMAL
ØØØØ4  16710  8192       Ø  17460
ØØØØ8      Ø   256       Ø   4Ø96
ØØØØC  32768 17990    8192
>
```

Display the contents of memory location $AFØ_{16}$:

```
>DS ØAFØ (CR)
ØØAFØ   ØØØAE                                  HEXADECIMAL
>
```

Display the contents of memory location AF_{16} thru $B3_{16}$:

```
>DS ØAF-ØB3 (CR)
ØØØAF   ØØØØØ Ø2ØA8 ØØØ4C Ø4Ø78               HEXADECIMAL
ØØØB3   Ø242Ø
>
```

2.5.2 LIST MEMORY COMMAND

Assembly language output is produced by the list (LS) command and may also be requested in Alter Mode or as part of a snap specification. When requested, Pace 2 will interpret the contents of memory as Pace instructions and list them as they might appear in an assembly listing.

The address and contents are displayed in hexadecimal. The label field will be listed if a symbol is found in the User Symbol Table (in Pace 2 read/write memory) having a value equal to the address. If more than one symbol has this value, the first symbol encountered will be listed.

The following format defines the list command:

LS {b {am}}

Command Code	Address
LS	Any type of memory address

List Command Examples:

List the contents of memory locations Ø thru 8_{16}:

```
>LS Ø-Ø8 (CR)
ØØØØØ  Ø38ØØ          PFLG LK
ØØØØ1  ØCØØC  GO:     LD AO,VAL
ØØØØ2  Ø51E7          LI A1,-25
ØØØØ3  ØC821          LD A2,ADDR
ØØØØ4  ØD2Ø3  LOOP:   ST AØ,3(A2)
ØØØØ5  Ø7AØ4          AISZ A2,4
ØØØØ6  Ø79Ø1          AISZ A1,1
ØØØØ7  Ø18Ø4          JMP LOOP
ØØØØ8  ØØØØØ         HALT
>
```

2.5.3 SCAN MEMORY/REGISTER COMMAND

The Scan Command will scan the specified register or memory address(s) and print out the address(s) with contents equal to the specified value (or not equal to the specified value, if # is entered) after being masked by the specified mask (or $FFFF_{16}$ if no mask is specified). The algorithm is as follows:

1) Take the contents of the register or memory address under test, "exclusive OR" with the specified value and then AND with the specified mask.
((contents) ⊕ value) • (mask)

2) Display the address if the result is =Ø and "#" not specified or the result is ≠ Ø and "#" is specified.

One of the optional parts of the Scan Command allows the operator to modify the meaning of the value. It can search for a given value or it can search for all values other than a named value by depressing the "not" key (#) immediately preceding the value not desired. The Pace 2 will then scan all other values and display them.

The following format defines the Scan Command:

SC {aa} {/dm} , {#} se_1 {,se_2}

Command Code	Address	/	Display Mode,	#	Value,	Mark
SC	any register or memory address(s)		NONE A- S- u-	= or ≠	Contents being searched for	digit(s) or bit(s) being masked out

Scan Examples:

Example 1:

Scan the contents of all of the registers for Ø:

```
>SC $,Ø (CR)
$PC   ØØØØØ
$AØ   ØØØØØ ØØØØØ
>
```

The contents of the PC,AØ and A1 were Ø.

Example 2:

Scan the contents of all of the registers for Ø1FFE:

```
>SC $,Ø1FFE (CR)
$SØ   Ø1FFE
>
```

The contents of only level Ø of the stack were Ø1FFE.

Example 3:

Scan the contents of memory address Ø thru $ØFF_{16}$ (256_{10}) for Ø38ØØ:

```
>SC Ø-ØFF,Ø38ØØ (CR)
ØØØØØ   Ø38ØØ
>
```

Example 4:

Scan the contents of memory addresses Ø thru ØFF for ØCØØC:

```
>SC Ø-ØFF,CØØC (CR)  -  Re-entered
ØØØ1C   ØCØØC
ØØØ3A   ØCØØC
ØØØ3E   ØCØØC
>
```

Example 5:

Scan the contents of memory address Ø thru ØFF for Ø:

```
>SC 0-0FF,0 (CR)
ØØØØØ   ØØØØØ ØØØØØ ØØØØØ ØØØØØ
ØØØØ4   ØØØØØ
ØØØ6    ØØØØØ ØØØØØ ØØØØØ
ØØØØA   ØØØØØ ↑D                Scan terminated by Control-D
>
```

Then to scan using the "not" value:

Example 6:

```
>SC Ø-ØFF,#Ø (CR)
ØØØØ5   Ø2ØØØ
ØØØØ9   ØØ1ØØ
ØØØØB   Ø1ØØØ Ø8ØØØ
ØØØØE   Ø2ØØØ
ØØØ1Ø   Ø6Ø31 ØØØ12 ØØ2Ø3 ØØ1ØØ
ØØØ14   ØØ18Ø ØØØ1Ø ØØØØ3
>
```

The second optional feature of the Scan Command allows the operator to scan the registers or memory for a certain digit, digits, bit or bits "masking out" the digit(s) or bit(s) not desired to be read. In the mask, enter hexadecimal characters into each digit in order to designate which bit(s)

are to be "masked out" and which are not. The bit(s) desired to be masked out are entered as zeros. The bit(s) that are to be scanned for are entered as "1's" meaning "enable".

Example 7:

Scan the contents of memory locations Ø thru ØFF for the value 9 in the fourth digit (Bits 15, 14, 13, 12).

```
>SC Ø-ØFF,Ø99ØØ,ØFØØØ (CR)
ØØØØØ   Ø99ØØ
ØØØØ8   Ø99ØØ
ØØØ1C   Ø99ØØ
ØØØF1   Ø9EB6
>
```

Example 8:

Scan the contents of memory locations Ø thru ØFF for the value 9 in the third digit (Bits 11, 10, 9,8).

```
>SC Ø-ØFF,Ø99ØØ,ØØFØØ (CR)
ØØØØØ   Ø99ØØ
ØØØØ8   Ø99ØØ
ØØØ1C   Ø99ØØ
ØØØ33   Ø19F3
>
```

Example 9:

Scan the contents of memory locations Ø thru ØFF for a "1" in Bit 15.

```
>SC Ø-OFF,Ø99ØØ,Ø8ØØØ (CR)
ØØØØØ   Ø99ØØ ØC38Ø ØC732 ØC733
ØØØØ4   ØC733 ØC733 ØC733cØC733
ØØØØ8   Ø99ØØ ØC6D5
ØØØ1Ø   ØFFFF ØFFFF ØFFFF ØFFFF
ØØØ14   ØFFFF ØFFFF ØFFFF ØFFFF
ØØØ18   ØFFFF ØFFFF
ØØØ1B   ØFFFF Ø99ØØ ↑D          Scan terminated by Control-D
>
```

Example 10:

Scan the contents of memory locations Ø thru OFF for an "ASCII T" in the upper byte (Bits 15-8).

```
>SC Ø-ØFF/A,'T',ØFFØØ (CR)
ØØØ1C   'T4'
ØØØC2   'TØ'
>
```

Note: All ASCII input must be enclosed in single quotes.

Example 11:

Scan the contents of Ø thru ØE for -14541_{10}.

```
>SC Ø-ØE/S,-14541 (CR)
ØØØØ3   -14541 -14541 -14541 -14541
>
```

2.5.4 SET MEMORY/REGISTER COMMAND

The Set Command sets the contents of the specified register(s) or memory locations to the specified value. The following format defines the set command:

STb {aa} ,se↵

Command Code	Address	,	Value
ST	Any register or memory address		Any decimal or hexadecimal value

Example 1:

Set the contents of all of the registers to Ø.

```
>ST $Ø (CR)?? - syntax error occurred because a comma is missing
                between the $ and the Ø.
```

Example 2:

Set the contents of Accumulators 2 and 3 to $123_{1Ø}$.

```
ST $A2-3,123 (CR)
```

Example 3:

Set the contents of memory location Ø-FF to Ø.

```
>ST Ø-ØFF,Ø (CR)
```

2.6 MEMORY LOAD/SAVE COMMANDS

The commands in this group are designed to load or save blocks of memory, and/or the User Symbol Table, on paper tape. The tapes loaded or saved are in Pace binary load module format as created by the Pace assembler.

SNEAK PREVIEW

IF YOU THINK THIS IS NEAT...
THE SCAMP VERSION OF PACER
IS ON ITS WAY!

A SCAMP Conversion Kit for your PACER
will be available June 1976!

These commands allow the user to save a manually entered program (and its defined symbols) for later reloading and execution.

2.6.1 LOAD COMMAND

The following format defines the Load Command:

LD{b[S/L]}↵

Command Code	Options
LD	S- L-

The Load Command loads a binary load module from the operator's console paper tape reader (if available). The"S"parameter specifies that symbol records, if encountered, are to be loaded into the User Symbol Table. The L parameter also specifies symbol loading, but in addition requests listing of the symbols being loaded. Symbols longer than four characters will be ignored.

As loading proceeds, the following status messages are listed:

Message	
TI = ti	Indicates the title of the load module as read from the title record.
SY = sy	Indicates the symbol"sy"was read and loaded into the symbol table. This message appears only if the "L" option was specified.
LO = ma	Indicates the lowest memory address loaded.
HI = ma	Indicates the highest memory address loaded.
ST = ma	Indicates the starting address specified in the end record.

The LO = and HI = messages will appear only if memory is actually loaded (ie., if a data record is encountered) and will be listed at the end of loading after the end record is processed. If any address or value within the load module is other than absolute (ie., external or relocatable) the message:

REL BIN

will be outputed at the end of loading. The address or value will be used if it were absolute.

Each record read from the load module is checked for errors. If a checksum error is encountered, one of the following messages will be output:

Message	Meaning
CKS TTL	Checksum error in a title record.
CKS SYM	Checksum error in a symbol record.
CKS ma,n	Checksum error in a data record. The record contains "n" words to be loaded starting at location ma.
CKS END	Checksum error in an end record.

After the message is printed, Pace 2 will wait for one of the following operator responses:

Response	Meaning
Control-C	Re-read the record. The operator must manually reposition the tape to the beginning of the record prior to entering the response.
Control-I	Ignore the error and process the block. This is echoed as "↑I".
Control-D	Terminate the load command.

All "control" characters are entered by depressing the CTRL key on the TTY and holding it down while depressing the appropriate letter key.

If the first word of a record is in error, Pace 2 will print the message:

FMT ERR

and wait for either the Control-C or Control-D response as specified above.

While reading the tape, Pace 2 checks the interbyte timing. If the reader does not respond within 500 msec (approximate), due to a reader jam or end-of-tape condition, the message:

TMO ERR

will be printed and the load command terminated. The operator may use the feature to prematurely terminate the load by manually stopping the reader.

Example 1:

Load data records from paper tape but do not load symbols.

LD (CR)

Example 2:

Load data records and symbol records from paper tape.

LD S (CR)

Example 3:

Load data records and symbol records from tape and print out the symbols as they are loaded.

```
>LD L (CR)
TMO ERR
```
The reader was not in the start position, therefore, we got a time out error.

```
>LD L (CR)
CKS TTL ↑D
```
The title block had a checksum error. Command was terminated with a Control-D.

```
>LD L (CR)
CKS TTL ↑I
TI=MECS
SY=STA !R
SY=LOOP !R
LO=Ø3Ø
HI=Ø54
ST=Ø3Ø
```
The title block again read a checksum error-- the title should be MESS. The Control-I was used to finish reading the block.

The !R is a warning message that the symbols had been previously defined.

2.6.2 SAVE COMMAND

The following format defines the save command:

SVbti $\left\{,S\right\}\left\{,am,\left\{---Am_4\right\}\right\}\left\{;ma\right\}$

Command Code	Title	, Symbol	, Address	; Address
SV	Program title	S - if a symbol record is to be punched	Any type of memory address (4 blocks max separated by commas)	Starting of the program

The Save command saves as a binary load module the currently defined user symbols and/or specified blocks of user memory. The load module is output to the operator's console paper tape punch (if available). Symbols are output if the "S" parameter is specified. Up to four blocks of memory may be specified. The starting address, if required, is specified by the ;ma parameter.

After the command is entered, Pace 2 waits for the operator to turn on the punch and depress the SPACE key. When punching ceases, Pace 2 waits for the operator to turn off the punch and depress the SPACE key prior to the next command prompt. The operator may prematurely terminate the command at any time by entering either Control-C or Control-D.

Example 1:
Create a load module with title A containing only the user symbols:

```
>SV A, S (CR)
```

Example 2:
Create a load module with title PROG X, stored at memory locations Ø1O-ØFF, with a starting address of Ø1Ø.

```
> SV PROGX,Ø1Ø-ØFF;Ø1Ø (CR)
```

Example 3:
Create a load module with title MESS, stored at memory locations Ø3Ø-Ø54, with a starting address of Ø3Ø.

```
>SV MESS,S,Ø3Ø-Ø54;Ø3Ø (CR)
 MESS
ØØ4STALOOP1
ØS@C$$ 11
(1< @D   I WILL KEEP
ZHTHE MESSAGE SHORT ----HI
! HT!
@&ØV
```

The TTY will print out all printable information as the tape is punched.

2.7 BREAK/SNAP POINT COMMANDS

Pace 2 maintains a break/snap point table within its own read/write memory. The table can hold up to five break/snap point definitions along with their

associated snap specifications. Each break/snap point defines a memory location at which Pace 2 will terminate execution and/or execute the associated snap specification. Whenever one or more break/snap points are set, the effective instruction execution time will be increased to approximately 1 millisecond/instruction.

2.7.1 SNAP SPECIFICATIONS

These are used to specify which active registers and/or memory addresses are to be displayed during a break point, snap point or single step execution. A snap specification consists of one or more of the following terms, separated by commas.

1) register address

2) register address range

3) implied register address range

4) memory address

5) memory address range

6) implied memory address range

Register addresses (1, 2 and 3 above) are "ORed" together to eliminate multiple listing of the same register (e.g., $RØ-2, $R1-3 will be listed as if $R was specified). A maximum of two memory addresses (4 and 5 above) may be included in any single snap specification (ie., 1ØØ-1Ø9,ØlØØ). Any memory address range greater than 4095 locations (FFF_{16}) will be limited to 4095.

When a snap specification is executed, the information requested is listed in three groups as follows:

1) Header, consisting of a three letter code (indicating the reason for the listing), the current value of PC (indicating the next instruction to be executed) and optional header information described below.

2) Registers, consisting of the specified registers (if any) in the order FL, AØ-A3 and SØ-S9 (PC is always listed in the header).

3) Memory, consisting of the specified memory locations (if any) in the order specified.

The three-letter header codes are as follows:

Code	Meaning
BRK	Break point (PC equals a specified break point address).
SNP	Snap point (PC equals a specified address).
STP	Single step.

(If a break or snap point is encountered in single step mode, the break or snap point snap specification will supersede the single step snap specification).

The optional header information consists of the contents of the location specified by PC listed in either hexadecimal or both hexadecimal and assembly language formats. The optional information is requested by specifying one of the following characters as the first term of the snap specification.

Character	Option
H	Hexadecimal only
L	Hexadecimal and assembly language

All register and memory contents are listed in hexadecimal. Execution of a snap specification may be terminated by entering Control-C.

The following format defines the snap specification:

Output Format	,	Address
H - Hexadecimal only L - Hexadecimal and assembly language		Any register or memory address

The following format defines the break point command:

BP,bma {,SS} {;n} ↵

Command Code	Address	, Snap Spec	; Executions
BP	Any memory address	H L address	The number of times the breakpoint must be executed before the break point is performed.

Example 1:

Set a break point at memory address Ø3 that will be executed prior to the fourth access of memory location Ø3.

```
>BP Ø3;4 (CR)
```

After the break point has been set, if the Run command is executed the following print out would occur:

```
>RN Ø1 (CR)      (run command)
BKP ØØØØ3
```

Example 2:

Set a break point at memory address Ø5. Execute the break point prior to each access of address Ø5. When the break point is executed, Pace 2 will printout:

1) all of the accumulators

2) the hexadecimal value and assembly language at Ø5

3) the contents of memory location Ø thru Ø5.

```
>BP Ø5,L,$A,Ø-Ø5 (CR)
```

After the break point has been set, if the Run command is executed the following printout would occur:

```
>RN Ø1 (CR)
BKP  ØØØØ5  Ø18ØØ        JMP  .-5
$AØ   Ø3F8D  ØØØØ2  ØCF92  ØC3A6
ØØØØØ   Ø18Ø1  Ø18Ø2  Ø18Ø3  Ø18Ø4
ØØØØ4   Ø18Ø5  Ø18ØØ,
```

2.7.2 SNAP POINT COMMAND

The snap point command is identical to the break point command except the snap specification is displayed without a program break (ie., program execution continues after the snap printout).

The following format defines the snap point command:

SPbma {SS} {;m} ↵

Command Code	Address	,	Snap Spec	;	Number of Executions
SP	Any memory address		H L Any register and memory address		Number of executions between snap point executions

Example 1:

Print out the contents:

1) of the snap address in hexadecimal and assembly language format

2) of address Ø thru Ø3 in hexadecimal

prior to the execution of the instruction at the snap address.

```
>SP Ø3,L,Ø-Ø3 (CR)

>RN Ø1 (CR)
SNP  ØØØØ3  Ø18Ø4        JMP  .+1
ØØØØØ   Ø18Ø1  Ø18Ø2  Ø18Ø3  Ø18Ø4
SNP  ØØØØ3  Ø18Ø4        JMP  .+1
ØØØØØ   Ø18Ø1  Ø18Ø2  Ø18Ø3  Ø18Ø4
SNP  ØØØØ3  Ø18Ø4        JMP  .+1 'D
```

2.7.3 LIST BREAK/SNAP POINT COMMAND

The list break/snap point command lists all of the specified break/snap points.

The following format will define the printout.

Address	Nv	*
Break/snap address	Number of executions (+1) prior to the next break or snap	* will be printed next to Nv if the address is a break point

```
>LB (CR)
ØØØØ1     1Ø
ØØØØ3      1
ØØØØ4      4 *
```

2.7.4 RESET BREAK/SNAP POINT COMMAND

The break/snap point command resets the break/snap points at the location that is defined, or all of the break/snap points if an @ is entered.

RBb[m@a]

Example 1:

Reset the break point at location Ø4.

```
>RB Ø4 (CR)
>
```

Example 2:

Reset the snap point at location Ø1.

```
>RB Ø1 (CR)
>
```

Example 3:

Reset all break/snap points.

```
>RB @ (CR)
>
```

2.8 PROGRAM EXECUTION COMMANDS

This group includes the commands used to run or single step through a program. The functions performed by these commands are also available in alter mode.

2.8.1 SINGLE STEP COMMAND

SS {b{ ma} {,ss_1}}↵

Enter Single Step user mode at location ma or at the location currently specified by PC if ma is omitted. The snap specification, if entered, will be executed after each step. The program is advanced to the next instruction by depressing the SPACE key. The system will return to Pace 2 Command mode in any of the following cases.

1) A system function Ø (or illegal system function) is executed (as described later).

2) By entering Control-D.

Example 1:

Enter Single Step mode at location Ø12.

```
>SS Ø12 (CR)
STP   ØØØ12     Ø5314 (SP)
STP   ØØØ13     ØC8Ø1 (SP)
STP   ØØØ14     Ø5ØFF CONTROL-C ↑C
```

Example 2:

Enter Single Step mode at location Ø12 and output the memory content in hexadecimal and assembly language.

```
>SS Ø12 (CR)
STP  ØØØ12   Ø5314   GO:   LI A3,2Ø (SP)
STP  ØØØ13   ØC8Ø1         LD A2,ALPH (SP)
STP  ØØØ14   Ø5ØFF         LI AØ,-1 (SP)
```

Example 3:

Enter Single Step mode at location Ø12 and display the next instruction in hexadecimal and assembly language, to be executed after each step.

```
>SS Ø12 (CR)
STP   ØØØ13   ØC8Ø1     LD A2,ACPH (SP)
STP   00014   Ø5ØFF     LI AØ,-1
```

2.8.2 RUN COMMAND

The Run command starts user program execution at the specified memory location or at the location specified by the PC if a memory address is not specified. The system will automatically return to Pace 2 Command mode in any of the following cases:

1) A break point location is encountered.

2) A halt instruction is executed.

3) A system function Ø (or illegal system function) is executed (as described later).

If an infinite loop occurs, the operator may force the system to return to Command mode as follows:

1) By entering Control-D during any snap point execution.

2) By pressing RESET.

3) By pressing INIT.

Using the following program, to illustrate the Run modes:

```
>LS Ø1Ø-Ø15 (CR)
ØØØ1Ø  Ø5ØØØ  STA:  LI AØ,Ø        Load AØ with Ø
ØØØ11  Ø51Ø4        LI A1,4        Load A1 with 4(10)
ØØØ12  Ø78Ø1  GO:   AISZ AØ,1      Increment AØ, test for Ø and
                                   skip the next instruction if Ø
ØØØ13  Ø79FF        AISZ A1,-1     Decrement A1, test for Ø and skip
                                   the next instruction if Ø
ØØØ14  Ø1812        JMP GO         Jump to address Ø12
ØØØ15  ØØØØØ       HALT
```

Example 1:

Execute the program starting at address Ø11.

```
>RN Ø11 (CR)
HLT  ØØØ16
```

Example 2:

Execute the program starting at address Ø1Ø after a snap was set at Ø13 to look at A1 and a snap was set at Ø15 to look at A1.

```
 RNØ1ØCR ??          Syntax error, no space between RN and address

>RN Ø1Ø CR
SNP  ØØØ13
$A1   ØØØØ4
SNP  ØØØ13
$A1   ØØØØ3
SNP  ØØØ13
$A1   ØØØØ2
SNP  ØØØ13
$A1   ØØØØ1
SNP  ØØØ15
$A1   ØØØØØ
HLT  ØØØ16
>
```

Example 3:

Execute the program starting at address Ø1Ø after a break point was set at address Ø14.

```
>RN Ø1Ø CR
SNP  ØØØ13
$A1   ØØØØ4
BKP  ØØØ14
>
```

Example 4:

Execute the program starting at address Ø1Ø and depress the restart button to get back to the Command mode. (Break point at Ø14 was reset).

```
>RN Ø1Ø CR
SNP  ØØØ13
$A1   ØØØØ4
SNP  ØØØ13
$A1   ØØØØ3
RST  ØØØ12
>
```

2.9 SYMBOL TABLE COMMANDS

The commands in this group allow the operator to examine and delete the contents of the User Symbol Table. The functions performed by these commands are also available in Alter mode.

2.9.1 LIST SYMBOL COMMAND

SY (CR)

Lists the entire contents of the User Symbol Table in the form sy n (where n is the value associated with the symbol). The listing is in alphabetic order and may be terminated by entering either Control-C or Control-D.

Example:

```
>SY (CR)
ADDR 00021
GO   00001
LOOP 00004
VAL  0000C
>
```

2.9.2 DELETE SYMBOLS COMMAND

DL b[@/sy](↵)

Deletes symbol (sy) as all user symbols if "*" is entered.

Example 1:

```
>DL  ADDR (CR)
```

Example 2:

```
>DL @ (CR)
```

2.10 ALTER MODE

Alter mode is entered from Command mode with the command:

AL (CR)

Once in Alter mode, the system will remain there until the operator either enters Control-D or presses INIT. If User mode is entered from Alter mode to run a program, the system will return to Alter mode when execution is terminated.

Within Alter mode, the user may examine or modify register or memory contents in an interactive manner, define user symbols, and execute any of the special

Alter mode commands. To examine a register or memory location, the user merely enters the desired address. Pace 2 then "opens" the specified address by displaying its contents in hexadecimal and, optionally, a previously specified Display mode. The user may then enter a new value and "close" the address, thus modifying it, or simply close it without modification. Once the address is closed a new address may be opened or any other Alter mode function performed. As with Pacer, Alter mode also provides automatic close functions which close any open location and automatically open a new location as specified by the selected function.

The Alter mode functions are described in detail in the following sections. All Alter mode functions are prompted by a RETURN/LINE FEED combination. Lines entered in Alter mode are terminated with either a RETURN or other specified control characters. In the examples which are shown, underlining is used to indicate operator input, encircling is used to indicate a control character input (ie., Ⓧ would indicate Control-X).

2.10.1 EXAMINING/MODIFYING MEMORY/REGISTERS

To open a memory location or register, the user enters the desired address (ma or ra) followed by a RETURN. Pace 2 will display the contents in hexadecimal and, if previously selected, a secondary Display mode and then wait for a second operator entry on the same line. If the contents are to be modified, the operator should enter the new contents and then terminate the line with either RETURN or one of the automatic close characters.

The secondary display mode is selected by entering one of the following control characters:

Character	Secondary Display
Control-H	None. The secondary display is deleted.
Control-A	ASCII
Control-U	Unsigned decimal
Control-S	Signed decimal
Control-L	Assembly language

When assembly language mode is selected, additional spaces will be output after the operands so that the following input lines will start at the same position. Also, since the assembly language display applies only to memory locations, it will be omitted while examining registers.

The display mode select characters are not echoed and may be entered at any time that Pace 2 is waiting for an input line. A given Display mode remains in effect until a new mode is selected.

The manner in which the new contents of an open address are specified is determined by the type of address. For register addresses, the new contents may be entered as any expression while for memory addresses either any expression or assembly language input is permitted. The line termination characters used to close an open address are described below. The term "current address" refers to the currently open or last opened address contained in an internal register maintained by Pace 2. The current address is initially set to the value of PC.

Character	Function
RETURN	Close the current address and ouput a new prompt. This allows the operator to manually enter a new address or execute some other Alter mode function.
LINE FEED	Close the current address and automatically open the current address + 1.
ESC, ACK, or ALT MODE	Close the current address and automatically open the current address - 1.
Control-V	Close the current address and automatically open the memory location specified by the contents of the current location.
Control-E	Close the current address and automatically open the memory location specified by the effective address as explained in the following paragraphs.

Control-N	Close the current address and automatically open the next highest address which satisfies the scan criteria (explained below).

The automatic close functions which index to a new address (LINE FEED, ESC and Control-N) treat the registers and memory as idependent linear arrays. The register array has the following relative address sequence.

Address	Register
Ø	PC
1	FL
2-5	AØ-A3
6-15	SØ-S9

Whenever one of these functions attempts to exceed the current array bounds (Ø-15 for register, Ø-$FFFF_{16}$ for memory), the function is aborted and the error message ? is output. The operator may switch to the memory array with the Control-V and Control-E functions or to either array by manually specifying an address in that array.

The "effective address" used by the Control-E function is contained in an auxilliary address register maintained internally by Pace 2. The effective address is set by either the "?" command (described in a later section) or by the display, or entry in assembly language, of a memory reference instruction that uses base page or PC relative addressing. In the latter case, the effective address is set to the address of the memory location referenced. As a result, the user can modify a location by entering:

JMP ABC+5 (E)

and then automatically open the location specified by ABC+5. The effective address is initially set to the value of PC.

The Control-N function is controlled by scan value and scan mask registers maintained internally by Pace 2. The value and mask registers are identical to the VL and MK registers of Pacer as is the scan function initiated by Control-N. The scan registers are set by the command described in a later section.

The line termination characters described above may also be used to automatically open a new register when no register is currently open. In this case, the operator enters the desired character in place of a register or memory address. If RETURN is entered, the current address is opened again.

To enter Alter mode type

AL (CR)

the following examples show memory/register examination/modification.

Example 1:
Examine and modify the PC.

$PC (CR) ØØØØØ ØFAB3 (CR)

Example 2:
Modify address Ø1Ø and automatically open address Ø11.

Ø1Ø (CR) Ø1ØC3 ØCA43 (LF)
Ø11 ØØØØØ

Example 3:
Modify address Ø1Ø and automatically open address ØØF.

Ø1Ø (CR) Ø1ØC3 ØCA43 (AL)
ØØF ØØØØØ

Example 4:
Modify address Ø1Ø and automatically open address determined by the contents of address Ø1Ø.

Ø1Ø (CR) Ø1ØC3 ØCA43 (V)
ØCA43 ØØØØØ

3.0 PROGRAMMING FUNDAMENTALS

This section describes the two general types of computer instructions and the way in which they are used in computer programs. The first type of instruction is distinguished by the fact that it operates upon data that is stored in some memory location and must tell the computer where the data is located in memory so that the computer can find it. This type of instruction is said to reference a location in memory; therefore, these instructions are often called memory reference instructions (MRI).

When speaking of memory locations, it is very important that a clear distinction is made between the address of a location and the contents of that location: A memory reference instruction refers to a location by a 16-bit address; however, the instruction causes the computer to take some specified action with the content of the location. Thus, although the address of a specific location in memory remains the same, the content of the location is subject to change. In summary, a memory reference instruction uses a 16-bit address value to refer to a memory location, and it operates on the 16-bit binary number stored in the referenced memory location.

The second type of instructions are the operate instructions, which perform a variety of program operations without any need for reference to a memory location. Instructions of this type are used to perform the following operations: clear the accumulator, test for negative accumulator, halt program execution, etc.

3.1 PROGRAM CODING

Binary numbers are the only language which the computer is able to understand. It stores numbers in binary and does all its arithmetic operations in binary. What is more important to the programmer, however, is that in order for the computer to understand an instruction it must be represented in binary. The computer can not understand instructions which use English language words. All instructions must be in the form of binary numbers (binary code).

3.2 Binary Coding

The computer has a set of instructions in binary code which it "understands". In other words, the circuitry of the machine is wired to react to these binary numbers in a certain manner. These instructions have the same appearance as any other binary number; the computer can interpret the same binary configuration of Ø's and 1's as data or as an instruction. The programmer tells the computer whether to interpret the binary configuration as an instruction or as data by the way in which the configuration is encountered in the program.

Suppose the computer has the following binary instruction set.

Instruction A	111Ø ØØØØ ØØØ1 ØØ1Ø	This binary number instructs the computer to add the contents of location ØØØØ ØØØØ ØØØ1 ØØ1Ø to accumulator Ø.
Instruction B	111Ø ØØØØ ØØØ1 Ø111	This binary number instructs the computer to add the contents of location ØØØØ ØØØØ ØØØ1 Ø111 to accumulator Ø.

If instruction B is contained in a memory location with an address of ØØØØ ØØØØ ØØØ1 ØØ1Ø and the binary number ØØØØ ØØØ1 1111 1111 is stored in a location with an address of ØØØØ ØØØØ ØØØ1 Ø111, the following program could be written:

Location	Content
ØØØØ ØØØØ ØØØ1 ØØ1Ø	111Ø ØØØØ ØØØ1 ØØ1Ø
ØØØØ ØØØØ ØØØ1 Ø111	ØØØØ ØØØ1 1111 1111

If this program were to be executed, the number ØØØØ ØØØ1 1111 1111 would be added to the accumulator.

3.3 Hexadecimal Coding

If binary configurations appear cumbersome and confusing, the reader will now understand why most programmers seldom use the binary number system in actual practice. Instead, they substitute the hexadecimal number system. The reader should not proceed until he understands these two number systems and the conversions between them.

Henceforth, hexadecimal numbers will be used to represent the binary numbers which the computer uses. Although the programmer may use hexadecimal numbers to describe the binary numbers within the computer, it should be remembered that the hexadecimal representation itself does not exist within the computer.

When the conversion to hex is performed, Instruction B becomes $DØ17_{16}$ and the previous program becomes.

Location	Content
$ØØ12_{16}$	$DØ17_{16}$
$ØØ17_{16}$	$Ø1FF_{16}$

To demonstrate that a computer can not distinguish between a number and an instruction, consider the following program.

Location	Content	
ØØ11	DØ12	(Instruction A)
ØØ12	DØ17	(Instruction B)
.		
.		
.		
ØØ17	Ø1FF	(The number $Ø1FF_{16}$)

Instruction A, which adds the contents of location 0012 to the accumulator, has been combined with the previous program. Upon execution of the program

(assuming the initial accumulator value=Ø), the computer will execute instruction A and add $D017_{16}$ as a number to the accumulator obtaining a result of $D017_{16}$. The computer will then execute the next instruction, which is DØ17, causing the computer to add the contents of 0017 to the accumulator. After the execution of the two instructions the number D216 is in the accumulator. Thus, the above program caused the number $D017_{16}$ to be used as as instruction and as a number by the computer.

3.4 Mnemonic Coding

Coding a program in hex numbers, although an improvement upon binary coding, is nevertheless very inconvenient. The programmer must learn a complete set of hex numbers which have no logical connection with the operations they represent. The coding is difficult for the programmer when he is writing the program, and this difficulty is compounded when he is trying to debug or correct a program. There is no easy way to remember the correspondence between a hex number and a computer operation.

To simplify the process of writing or reading a program, each instruction is often represented by a simple 3- or 4-letter mnemonic symbol. These mnemonic symbols are considerably easier to relate to a computer operation because the letters often suggest the definition of the instruction. The programmer is now able to write a program in a language of letters and numbers which suggests the meaning of each instruction.

The computer still does not understand any language except binary numbers. Now, however, a program can be written in a symbolic language and translated into the binary code of the computer because of the one-to-one correspondence between the binary instructions and the mnemonics. This translation could be done by hand, defeating the purpose of mnemonic instructions, or the computer could be used to do the translating for the programmer. Using a binary code to represent alphabetic characters, the programmer is able to store alphabetic information in the computer memory. By instructing the computer to perform a translation, substituting binary numbers for the alphabetic characters, a program is generated in the binary code of the computer. This process of translation is called "assembling" a program. The program that performs the translation is called an assembler.

It is well to make some observations about the assembler at this point.

1) The assembler itself must be written in binary code, not mnemonics.

2) It performs a one-to-one translation of mnemonic codes into binary numbers.

3) It allows programs to be written in a symbolic language which is easier for the programmer to understand and remember.

3.5 ELEMENTARY PROGRAMMING TECHNIQUES

Mastery of the instruction set is the first step in learning to program Pacer. The next step is to learn to use the instruction set to obtain correct results and to obtain them efficiently. This is best done by studying the following programming techniques. Examples, which should further familiarize the reader with the instructions and their uses, are given to illustrate each technique.

The modern digital computer is capable of storing information, performing calculations, making decisions based on the results and arriving at a final solution to a given problem. The computer can not, however, perform these tasks without direction. Each step which the computer is to perform must first be worked out by the programmer.

The programmer must write a program, which is a list of instructions for the computer to follow to arrive at a solution for a given problem. This list of instructions is based on a computational method, sometimes called algorithm, to solve the problem. The list of instructions is placed in the computer memory to activate the applicable circuitry so that the computer can process the problem. This section describes the procedure to be followed when writing a program to be used on the Pacer.

3.6 PROGRAMMING PHASES

In order to successfully solve a problem with a computer, the programmer proceeds through the five programming phases listed below:

1) Definition of the problem to be solved,

2) Determination of the most feasible solution method,

3) Design and analysis of the solution--flowcharting,

4) Coding the solution in the programming language, and

5) Program checkout.

The definition of the problem is not always obvious. A great amount of time and energy can be wasted if the problem is not adequately defined. When the problem is to sum four numbers, the defining phase is obvious. However, when the problem is to monitor and control a performance test for semiconductors, a precise definition of the problem is necessary. The question that must be answered in this phase is: "What precisely is the program to accomplish?"

Determining the method to be followed is the second important phase in solving a problem with a computer. There are perhaps an infinite number of methods to solve a problem, and the selection of one method over another is often influenced by the computer system to be used. Having decided upon a method based on the definition of the problem and the capabilities of the computer system, the programmer must develop the method into a workable solution.

The programmer must design and analyze the solution by identifying the necessary steps to solve the problems and arranging them in a logical order, thus implementing the method. Flowcharting is a graphical means of representing the logical steps of the solution. The flowcharting technique is effective in providing an overview of the logical flow of a solution, thereby enabling further analysis and evaluation of alternative approaches.

Having designed the problem solution, the programmer begins coding the solution in the programming language. This phase is commonly called programming but is actually coding and is only one part of the programming process. When the program has been coded and the program instructions have been stored in the computer memory, the problem can be solved. At this point, however, the programming process is rarely complete. There are very few programs written which initially function as expected. Whenever the program does not work properly, the programmer is forced to begin the fifth step of programming, that of checking out or "debugging" the program.

The program checkout phase requires the programmer to methodically retrace the flow of the instructions step-by-step to find any program errors that may exist. The programmer can not tell a computer: "You know what I mean!", as he might say in daily life. The computer does not know what is meant until it is told, and once given a set of instructions, the computer follows them precisely. If needed instructions are left out or coding is done incorrectly, the results may be surprising. These flaws, or "bugs" as they are often called, must be found and corrected. There are many different approaches to finding bugs in a program; however, the chosen approach must be organized and painstakingly methodical if it is to be successful.

3.7 FLOWCHARTING

A simple problem to add three numbers together is solved in a few, easily determined steps. A programmer could sit at his desk and write out three or four instructions for the computer to solve the problem. However, he probably could have added the same three numbers with paper and pencil in much less time than it took him to write the program. Thus, the problems which the programmer is usually asked to solve are much more complex than the addition of three numbers, because the value of the computer is in the solution of problems which are inconvenient or time consuming by human standards.

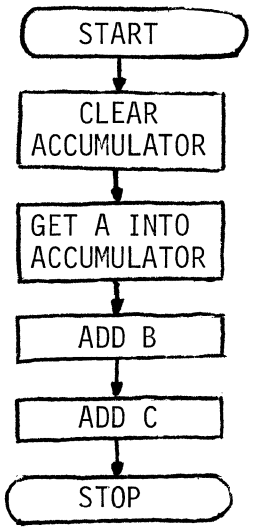
When a more complex problem is to be solved by a computer, the program involves many steps, and writing it often becomes long and confusing. A method for solving a problem which is written in words and mathematical equations is extremely hard to follow, and coding computer instructions from such a document would be equally difficult. A technique called flowcharting is used to simplify the writing of programs. A flowchart is a graphical representation of a given problem, indicating the logical sequence of operations that the computer is to perform. Having a diagram of the logical flow of a program is a tremendous advantage to the programmer when he is determining the method to be used for solving a problem, as well as when he writes the coded program instructions. In addition, the flowchart is often a valuable aid when the programmer checks the written program for errors.

The flowchart is basically a collection of boxes and lines. The boxes indicate what is to be done and the lines indicate the sequence of the boxes. The boxes are of various shapes which represent the action to be performed in the program.

The following are examples of flowcharts for specific problems, illustrating methods of attacking problems with a computer program as well as illustrating flowcharting techniques. Example 1 adds three numbers together. Example 2 puts three numbers in increasing order.

Example 1: Straight-Line Programming

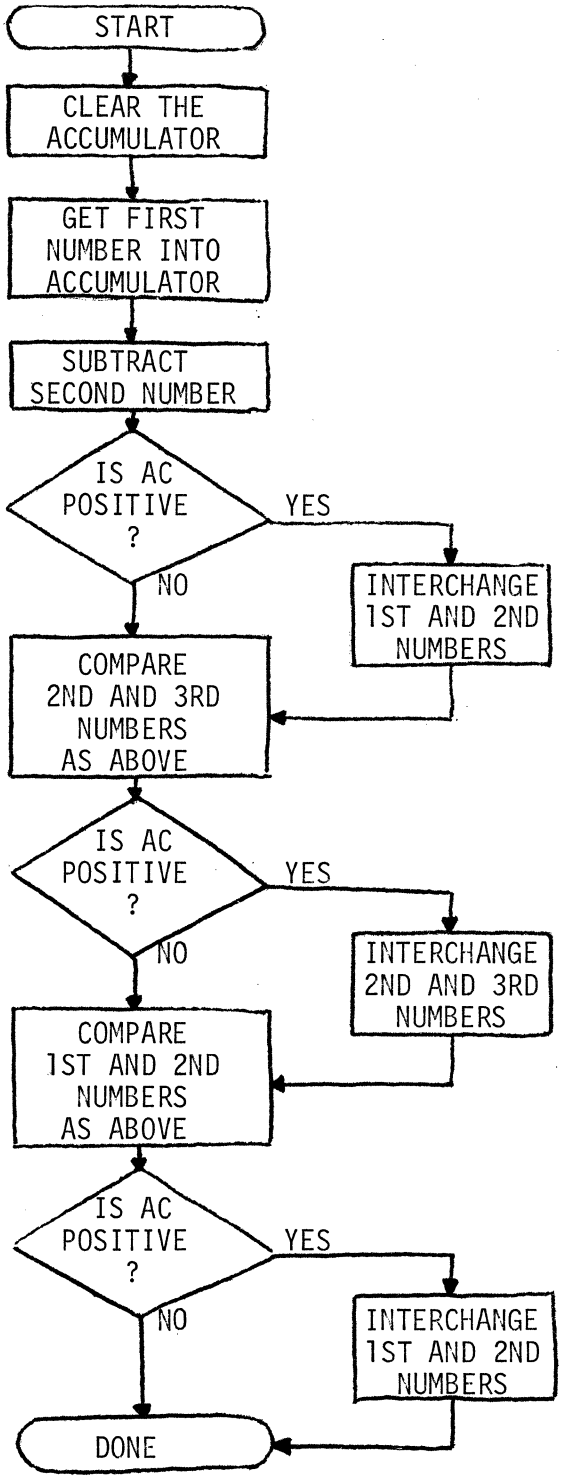
Example 1 is an illustration of straight-line programming. As the flowchart shows, there is a straight-line progression through the processing steps with no change in course. The value of X, which is equal to A+B+C is in the accumulator when the program stops.



Example 1 - Add Three Numbers

Example 2: Program Branching

Example 2 is designed to arrange three numbers in increasing order. The program must branch to interchange numbers that are out of order. (Branching, a common feature of programming, is described in the user's manual). Note that the arithmetic operations of subtraction are done in the accumulator, which must be cleared initially.



Example 2 - Arrange Three Numbers in Increasing Order

4.0 CODING A PROGRAM

The introduction of an assembler enabled the programmer to write a symbolic program using meaningful mnemonic codes rather than the octal representation of the instructions. The programmer could now write mnemonic programs such as the following example, which multiplies 18_{10} by 36_{10} using successive addition.

```
Ø20     LIØ,Ø       (Initialize)
Ø21     LIØ-18      (Set up a CNTR
Ø22     STØ, 212    count the additions of 36)
Ø23     ADDØ, 211   (Add 36)
Ø24     ISZ 212     (Skip if CNTR is 0)
Ø25     JMP 204     (Add another 36 if not done)
Ø26     HALT        (Stop after 18 times)
Ø27     0012        (Equal to 18₁₆)
Ø28     0024        (Equal to 36₁₆)
Ø29     0000        (Holds the tally)
```

Writing the above program was greatly simplified because mnemonic codes were used for the hex instructions. However, writing down the absolute address of each instruction is clearly an inconvenience. If the programmer later adds or deletes instructions, thus altering the location assignments of his program, he has to rewrite those instructions whose operands refer to the altered assignments. If the programmer wishes to move the program to a different section of memory, he must rewrite the program. Since such changes must be made often, especially in large programs, a better means of assigning locations is needed. The assembler provides this better means.

Location Assignment

As in the previous program example, most programs are written in successive memory locations. If the programmer assigned an absolute location to the first instruction, the assembler could be told to assign the next instructions to the following locations in order. The assembler maintains a current location counter by which it assigns successive locations to instructions.

Symbolic Addresses

The programmer does not at the outset know which locations he will use to store constants or the tally. Therefore, he must leave blanks after each MRI and

come back to fill these in after he has assigned locations to these numbers. In the previous program, he must count the number of locations after the assigned initial address in order to assign the correct values to the MRI operands. Actually this is not necessary, because he may assign symbolic names (a symbol followed by a ":" is a symbolic address) to the locations to which he must refer, and the assembler will assign address values for him. The assembler maintains a symbol table in which it records the hex values of all symbolic addresses. With symbolic address name tags, the program is as shown below.

```
Ø20          LIØ,Ø
Ø21  GO:     LIØ,-18
Ø22          STØ,CNTR
Ø23  MULT:   ADDØ,B
Ø24          ISZ CNTR
Ø25          JMP MULT
Ø26          HALT
Ø27  A:
Ø28  B:
Ø29  CNTR:
```

Note: The ":" after a symbol (e.g., GO:) indicates to the assembler that the symbol is a symbolic address.

4.1 Symbolic Programming Conventions

Any sequence of letters (A,B,C...,Z) and digits (0,1,...,9) beginning with a letter and terminated by a delimiting character (: or =) is a symbol.

User-defined symbols (stored in the external symbol table) must be four characters in length.

The colon after a symbol in a line of coding (e.g., MULT:LDØ) indicates to the assembler that the value of MULT is the address of the location in which the instruction is stored. When an instruction that reference MULT (now a symbolic address) is encountered, the assembler supplies the correct address value for MULT. (Care must be taken that a symbolic address is never used twice in the same program and that all locations referenced by an MRI are identified somewhere in the program.)

The assembler will recognize the arithmetic symbols + and - in conjunction with numbers or symbols, thereby enabling "address arithmetic". For example, the instruction JMP START+1 will cause the computer to execute the instruction in the next location after START.

The decimal point, or period, is a character which is interpreted by the assembler as the value of the current location counter. This special symbol can be used as the operand of an instruction; for example, the instruction JMP.-1 causes the computer to execute the preceding instruction.

The equal sign is used to define symbols. This character is used to replace an undefined symbol with the value of a known quantity.

4.2 ASSEMBLY LANGUAGE INPUT

Assembly language input from the operator's console is allowed only in alter mode. The form of the input is very similar to that required by the Pace assembler, namely:

$$\{\langle \text{label} \rangle :\}\ \text{b}\ \langle \text{opcode} \rangle\ \{\text{b}\langle \text{operands} \rangle\}$$

Example:

```
LOOP:   LD Ø, Ø1E4
```

The opcode may be any of the 45 Pace instruction mnemonics or one of the special opcodes DATA and SYSF described below. The label field may contain any user defined symbol. If specified, the label symbol will be entered into the symbol table with a value equal to the current memory address (ie., the value of .). The operand field must be specified as required by the opcode. The special Pace assembler directives (=, and the set of opcodes starting with .) are not allowed opcodes. When a label is not specified the TAB key may be used to line up the opcode fields. One or more spaces, however, may be used if desired.

The operand syntax required for each opcode is specified in a table below. The special symbols used in the definitions are defined as follows:

Symbol	Meaning
a	Any expression which evaluates to a 2-bit accumulator address (Ø-3) or one of the

following special symbols:

Symbol	Value
AØ	Ø
A1	1
A2	2
A3	3

$a_Ø$ Any expression which evaluates to Ø or the special symbol AØ.

c Any expression which evaluates to a 4-bit branch condition code (Ø-15) or one of the following special symbols:

Symbol	Value	Condition
SF	Ø	Stackfull
ZR	1	AØ = Ø
PS	2	AØ $\geq$ Ø
BØ	3	$AØ_Ø$ = 1
B1	4	$AØ_1$ = 1
NZ	5	AØ $\neq$ Ø
B2	6	$AØ_2$ = 1
CN	7	Contin = 1
LK	8	LNK = 1
IE	9	IEN = 1
CY	1Ø	CRY = 1
NG	11	AØ < Ø
OV	12	OVF = 1
J3	13	JC13 = 1
J4	14	JC14 = 1
J5	15	JC15 = 1

d Any expression which evaluates to an 8-bit signed value (-128 to +127).

f Any expression which evaluates to a 4-bit flag code (Ø-15) or one of the following special symbols:

Symbol	Value	Flag
I1	1	IE1
I2	2	IE2

(cont.)

Symbol	Value	Flag
I3	3	IE3
I4	4	IE4
I5	5	IE5
OV	6	OVF
CY	7	CRY
LK	8	LNK
IE	9	IEN
BY	A	BYTE
F1	B	F11
F2	C	F12
F3	D	F13
F4	E	F14

m	Any expression representing a memory address or the form d(a) representing a displacement and base accumulator (or addressing mode). d and a are defined above.
$m_{\emptyset}$	Any expression representing a memory address.
V	Any expression.
V_7	Any expression which evaluates to a 7-bit value (Ø-127).
V_1	Any expression which evaluates to a 1-bit value (Ø or 1).

The special symbols described above are separate from those in the symbol table and may be used only where specified. They may not be combined in expressions.

The operand syntax definitions, alphabetically according to opcode, are as follows:

Opcode	Operand(s)
ADD	a,m
AISZ	a,d
AND	$a_{\emptyset}$,m
BOC	c,$m_{\emptyset}$
CAI	a,d
CFR	a
CRF	a

(cont.)

Opcode	Operand(s)
DATA	V
DECA	$a_{\emptyset}$,m
DSZ	m
HALT	
ISZ	m
JMP	{e} m
JSR	{e} m
LD	a,m or $a_{\emptyset}$,@m
LI	a,d
LSEX	$a_{\emptyset}$,m
OR	$a_{\emptyset}$,m
PFLG	f
PULL	a
PULLF	
PUSH	a
PUSHF	
RADC	a,a
RADD	a,a
RAND	a,a
RCPY	a,a
ROL	a,v_7,v_1
ROR	a,v_7,v_1
RTI	{d}
RTS	{d}
RXCH	a,a
RXOR	a,a
SFLG	f
SHL	a,v_7,v_1
SHR	a,v_7,v_1
SKAZ	$a_{\emptyset}$,m
SKG	$a_{\emptyset}$,m
SKNE	a,m
ST	a,m or $a_{\emptyset}$,@m
SUBB	$a_{\emptyset}$,m

The order and function of the operands are identical to those of the Pace assembler (refer to the Pace User's Manual for further information). The addressing mode for memory reference instructions is automatically determined by Pace 2 unless explicitly specified by accumulator relative addressing of the form d(a). Where d is any expression which evaluates to an 8-bit signed value (-128 to + 127) and a is an accumulator (Ø thru 3).

The DATA opcode is assembled as the value of the operand expression. The SYSF opcode is assembled as a system function which is described later.

Note: Assembly language input is an immediate one pass function. As a result, any symbol referenced in an operand (other than the special symbols) must be defined at the time of reference. If a referenced symbol is redefined at a later time, it will NOT alter the previously assembled referencing instructions.

Example:

```
VAL = Ø10
ADDR = Ø20

        PFLG    $
        LD      AØ,VAL
        LI      1,-25
        LD      2,ADDR
LOOP:   ST      Ø,3(A2)
        AISZ    2,4
        AISZ    A1,1
        JMP     LOOP
        HALT
```

Note: VAL and ADDR must have been defined prior to their reference. If VAL or ADDR are redefined later it will not affect the assembled code (ie., the two LD instructions will still refer to the locations specified by the original values of VAL and ADDR).

4.3 THE SYMBOL TABLE

Pace 2 maintains a user symbol table within its own read/write memory. Symbols are entered into the table either manually, in Alter mode, by explicitly defining them or automatically, with the load (LD) command, from

the symbol blocks of a load module. Symbols stored in the table contain from one to four alphanumeric characters starting with an alphabetic character. Symbols may be redefined at any time. Any redefinition, however, will result in a warning message. Individual symbols or the entire contents of the table may be deleted at any time by use of the "!" function in alter mode or the delete (DL) command. The symbol table is automatically cleared by a power up or INIT operation. The basic symbol table holds up to 26 symbols with an option available to enlarge it to 121 symbols. The value assigned to a symbol may be any 16-bit quantity.

Warning messages that are printed during symbol table manipulation are as follows:

!R	Indicates that a symbol definition has resulted in the redefinition of the same symbol currently in the table. This is printed even if the new and old values are the same.
!F	Indicates that a symbol definition was ignored due to a table full condition.
!L	During a load command, which includes a symbol load and list, this message indicates that a five or six character symbol was encountered and ignored. If the list was not specified, the symbol will be ignored and the message omitted.

Other symbol errors, such as manually defining a five character (or longer) symbol or referencing an undefined symbol, will result in a syntax error message (??).

4.4 SYMBOL DEFINITION

Symbols are defined in alter mode by entering the sequence:

symbol = expression

where any expression may be used. Symbol definition will not change the current address on any of the other internal registers used by Pace 2. A special case of symbol definition, however, may be used to set the current address to a memory address defined by an expression. This special case is entered as:

. = <expression>↵

After the definition is entered, the specified address is automatically opened and displayed.

```
> ABC=5 (CR)                    Defines the symbol ABC as 5

>.Ø1ØØ*4 (CR)                   Opens the memory location specified by
  ØØ4ØØ     Ø5678              1ØØ16*4
>
```

4.5 ASSEMBLY LANGUAGE OUTPUT

Assembly language output is produced by the list (LS) command and may also be requested in alter mode or as part of a snap specification. When requested, Pace 2 will interpret the contents of memory as Pace instructions and list them as they might appear in an assembly listing.

The address and contents are displayed in hexadecimal. The label field will be listed if a symbol is found in the user symbol table having a value equal to the address. If more than one symbol has this value, the first symbol encountered will be listed. The opcode and operands are output as described for assembly language input with the following exceptions:

1) a, $a_Ø$, c, and f operands will always be listed as the special symbol corresponding their value. f operands having values of Ø and 15 will be listed as decimal numbers.

2) d, v_7, and $v_Ø$ operands will be listed as decimal numbers. For RTS and RTI d will be omitted if Ø.

3) Memory reference operands (m and $m_Ø$) will be listed according to the referenced address as follows:

   a) ⟨symbol⟩{[±] n} if a symbol is found having a value within ± 9 of the referenced address. The symbol closest in value to the address will be selected.

   b) .{[±n]} if the address of the instruction is within ± 9 locations of the referenced address and no symbol is found having a closer value.

   c) reference address$_{16}$⟩ if neither of the above criteria are satisfied.

Instructions having unused fields that are non-zero (with exception of SYSF) will be listed as data using the DATA opcode. Sufficient spaces are automatically inserted to line up the opcode operand fields.

5.0 SYSTEM FUNCTIONS

The Pace 2 system functions are implemented by using a sub-set of the redundant Pace HALT instructions. The Pace 2 hardware traps all HALT instructions and examines the actual instruction to see if it was a system function or not. If it is not a system function, Pace 2 retains control and outputs a halt message. If it is a system function, Pace 2 attempts to perform the requested function. System functions have the format $01XX_{16}$ where XX if the function code as described below. The assembly language format is:

```
SYSF   ce
```

where the value of ce (XX) must be one of those described below.

XX	Function
Ø	Return to Pace 2. If AØ=Ø no message is printed, otherwise the message: ERR ma is printed where ma is the address of the word following the SYSF.
1	Test for input from the operator's console. If no input is ready AØ will be set to -1. Otherwise, AØ will be set to $ØØCC_{16}$ where CC is the ASCII code entered from the keyboard (8 bits).
2	Get a character or byte from the operator's console or its attached paper tape reader. If LNK=1 the reader will be pulsed prior to waiting for character entry. In any case, Pace 2 will wait until a character is entered (on byte read). The character (byte) is returned in AØ as for the test function.
3	Output a character or byte to the operator's console and/or its attached paper tape punch. Bits 7-Ø of AØ are output. The byte will be punched if the punched has been manually activated.

If execution of the system function is successful, the instruction following the SYSF is skipped (the normal case). If an error is encountered, however, AØ is set to an appropriate error code and the instruction following the

SYSF is executed.

Note: The set of system functions currently implemented always skip the following instructions; ie., no errors are generated.

Only those registers/bits explicitly specified above are altered by the system functions. All other registers/bits are left unaltered. If an undefined system function is encountered, it will be executed as SYSF Ø with AØ set to $5Ø_{16}$.

Note: The RESTART button will have no effect while SYSE 2 is waiting for console input. To effect a restart, hold down the RESTART button and enter any key from the keyboard.

The following example programs will show how to use the alter mode features and show how they can save program coding and debugging time.

Program No. 1

Address	Hex Code	Mnemonic		Comment
Ø1Ø		LI	AØ,Ø	Load "Ø's" into ACØ
Ø11		LI	A1,4	Load ØØØ4 into AC4
Ø12		AISZ	AØ,1	Add on to ACØ and then test the result for zero. If zero, skip the next instruction.
Ø13		AISZ	A1,-1	Add a -1 (subtract 1) to AC4 and then test the result for zero. If zero, skip the next instruction.
Ø14		JMP	Ø12	Jump to address Ø12
Ø15		HALT		Halt

The mnemonic must now be converted to hexadecimal. The hexadecimal codes are obtained from the conversion chart and the address values are calculated where necessary (JMP).

Type in:

>AL (CR)

Alter mode is now active.

```
Ø1Ø (CR)   ØØØØØ   Ø5ØØØ (LF)
Ø11        ØØ2ØØ   Ø51Ø4 (LF)
Ø12        Ø1Ø8Ø   Ø78Ø1 (LF)
Ø13        ØFFFØ   Ø79FF (LF)
Ø14        ØØDEC   Ø1812 (LF)
Ø15        ØØA23   ØØØØØ (CR)
```

We now have the program loaded into memory.

An easier way to enter the program into the system memory is to use the line assembler feature of Pace 2.

```
>AL (CR)
Ø1Ø   ØØØØØ   STA:   LI Ø,Ø
Ø11   ØØ2ØØ          LI 1,4
Ø12   Ø1ØØØ   GO:    AISZ Ø,1
Ø13   ØFFFØ   AISZ   1,-1 ??      (syntax error)
Ø13   ØØDEC          AISZ 1,-1
Ø14   ØØA23          JMP GO       By using a symbol, we have the
Ø15   Ø45AB                       assembler calculate the address
↑D                                that it needs to jump to.
```

The program is loaded in memory and can now be executed.

```
% Ø1Ø (CR)          Run command
HLT ØØØ16          It ran and halted at address Ø15
                   when A,1 became zero.
```

Two snap points can now be set and A1 will "count down". Set a snap at address Ø13 and Ø15 as defined earlier.

```
% Ø1Ø (CR)
SNP  ØØØ13
$A1   ØØØØ4          Count is at 4
SNP  ØØØ13
$A1   ØØØØ3
SNP  ØØØ13
$A1   ØØØØ2
```

```
SNP  ØØØ13
$A1   ØØØØ1
SNP  ØØØ15
$A1   ØØØØØ                    Count is at Ø and the AISZ
HLT  ØØØ16                     instruction has skipped the
                               JMP instruction
```

The following example shows the print out when both break points and snap points are set and unit is in the single step mode.

```
! %1Ø,L CR

 STP  ØØØ11  Ø51Ø4          LI A1,4
 STP  ØØØ12  Ø78Ø1  GO:     AISZ AØ,1
 SNP  ØØØ13
 $A1   ØØØ4
 BKP  ØØØ14
 STP  ØØØ12  Ø78Ø1  GO:     AISZ AØ,1
 SNP  ØØØ13
 $A1   ØØØ3
 BKP  ØØØ14
 STP  ØØØ12  Ø78Ø1  GO:     AISZ AØ,1 'C
```

Example 2:

The following program uses the system function to allow the TTY to be used in the users program. The program will input a character from the TTY and then printout the character.

```
>AL (CR)
Ø2Ø  ØØØØØ LOOP:   SYSF 2 (LF)
Ø21  ØØØØØ (LF)
Ø22  ØØØØØ SYSF  3 (CR) ??      Syntax error (need to tab or space)
Ø23  ØØØØØ
Ø22  ØØØØØ (CONTR I) SYSF 3 (LF)
Ø23  ØØØØØ (LF)
Ø24  ØØØØØ (CONTR I) JMP LOOP (LF)
```

The program is now loaded to and ready to execute.

```
! Ø2Ø (CR) FROM HERE ON I HAVE CONTROL FROM THE KEY BOARD
I CAN TYPE ANY THING AND IT WILL BE PRINTED OUT FOR ME
0123456789
RST ØØØ23                          Depress the restart key and then
                                   type any character to transfer
                                   control back to alter mode.
```

Example 3:

```
>THE FOLLOWING PROGRAM (SUBROUTINE) WILL ALLOW AN ASCII MESSAGE
TO BE PRINTED OUT.

>AL (CR)
      LI ??
Ø3Ø Ø3DØØ TAB   LI 3,Ø4Ø (LF)      Load A3 with the #40
Ø31 Ø48CØ LOOP: LD Ø,(3) (LF):R    Load AØ with content of addr 40
Ø32 Ø8ØØ8       ROR AØ,8,Ø (LF)    Rotate AØ right eight places
Ø33 ØØØE4       SYSF 3 (LF)        Output to TTY lower byte
Ø34 ØØØ88 (LF)
Ø35 Ø2Ø16       ROR AØ,8,Ø (LF)    Rotate AØ right eight places
Ø36 ØØØ4C       SYSF 3 (LF)        Output to TTY lower byte
Ø37 ØØØØ4 (LF)
Ø38 Ø1444       AISZ 3,1 (LF)      Add one to value in A3
Ø39 ØØØEC       JMP LOOP (LF)      Jump to add Ø31
Ø4Ø ØØØ8Ø '  ' (LF)
Ø41 ØØØØØ '  ' (LF)
Ø42 ØØØØØ 'I ' (LF)
Ø43 ØØØØØ 'WI' (LF)
Ø44 ØØØØØ 'LL' (LF)
Ø45 Ø4ØØØ 'KE' (LF)
Ø46 ØØØØØ 'EP' (LF)                See write up for ASCII constants
Ø47 ØØØØØ      (LF)
Ø45 Ø4B45 ' K' (LF)
Ø46 Ø455Ø 'EE' (LF)
Ø47 ØØØØØ 'P ' (LF)
Ø48 ØØØØØ 'TH' (LF)
Ø49 ØØØØ2 'E ' (LF)
Ø4A ØØØØØ 'ME' (LF)
```

To examine the program and the message, type a Control-D to enter the command mode.

```
>LS Ø3Ø-Ø3A CR
ØØØ3Ø Ø534Ø       LI A3,64
ØØØ31 ØC3ØØ LOOP: LD AØ,(A3)   List the program
ØØØ32 Ø241Ø       ROR AØ,8,Ø
ØØØ33 ØØ1Ø3       SYSF 3
ØØØ34 ØØØ88       DATA Ø88
ØØØ35 Ø241Ø       ROR AØ,8,Ø
ØØØ36 ØØ1Ø3       SYSF 3
ØØØ37 ØØØØ4       DATA Ø4
ØØØ38 Ø7BØ1       AISZ A3,1
ØØØ39 Ø42C7       BOC PS,Ø1
ØØØ3A Ø1831       JMP LOOP
```

Display the message in ASCII

```
>DS Ø4Ø-Ø55/A CR
ØØØ4Ø  '  ' '  ' 'I ' 'WI'
ØØØ44  'LL' ' K' 'EE' 'P '
ØØØ48  'TH' 'E ' 'ME' 'SS'
ØØØ4C  'AG' 'E ' 'SH' 'OR'
ØØØ5Ø  'T ' '--' '--' 'HI'
ØØØ54  '! ' '  '
```

To execute the program:

```
>AL CR
%Ø3Ø CR  I WILL KEEP THE MESSAGE SHORT ----HI!
```

Depress the restart switch and get RST ØØ3Ø.

5.1 ALTER MODE COMMANDS

All Alter Mode commands are specified with a single character and are terminated with a RETURN. The entry formats and command functions are described below. Only those registers specified are altered during execution.

Delete Symbol

: [sy / @] ↵	Deletes the symbol sy or the entire symbol table if @ is entered. Identical to the DL command.

!ABC (CR) Deltes symbol ABC

Display Value

?ae↵

Displays the value of the expression ae in hexadecimal and sets the effective address to this value. The value is printed on the same line preceded by =.

?32*4 (CR) = Ø8Ø

Enter User Mode

%{ma}{,ss}↵

Starts executing a program in user mode at location ma or at the PC location if ma is omitted. If ss is specified, the program will be run in single step mode. When execution is terminated, the current and effective addresses are set to the value of PC. The command functions identical to the ., GO and SS commands.

%0100,L (CR) Starts execution in single step mode at location $1ØØ_{16}$.

Scan Memory

↑{[ra/ma]},{#}se_1{,se_2}↵

Sets the scan value to se and the scan mask to se and proceeds to scan memory/registers upward starting at the specified address (ma/ra) or the current address if ma/ra is omitted. The scan algorithm is identical to that of the SCAN command. The first address satisfying the scan criteria is automatically opened. Control-N may then be used to advance to the next scan location.

↑0100,0F000 (CR) Scan memory starting at $1ØØ_{16}$ for the first location containing the value $FØØØ_{16}$.

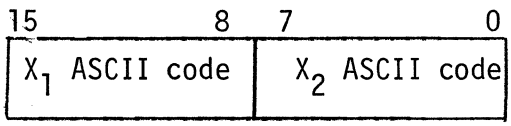
5.2 SELF-DEFINING TERMS

Numbers

Entered in decimal or hexadecimal in the form decimal number or Ø hexadecimal number , respectively.

ASCII Constants

Entered as either 'X_1'X_2','X', or "X. The form 'X_1X_2' is assembled as:



Hamilton/Avnet-Pacer a complete Microprocessor System!

The Pacer features the Pace MPU and all National Semiconductor I/O Memory and support components.

Quick to assemble:
All the accessory components and PC Boards are furnished. Also, a power supply, case, mother board and assembly book are supplied so you can be in operation within hours.

Easy to use:
A system monitor gives you a complete set of:

Control Functions
- Examine and modify
- Single Step
- Executive restart
- Run

Debug Functions
- Breakpoints
- Halt
- Hex calculations

Also, included is the Pacer User's Manual that gives instructions and sample programs on these functions in a simple self-teaching format.

Expandable:
Assembly language programming can be accomplished with the "PAC-I" TTY Interface/Program Assembler Card for $175.00. Extra memory is available with the "PAC-II" 2K by 16 RAM static RAM card for $199.00.

Low Cost – $695.00:
The "PACER" is a complete desk top "PACE" microcomputer development system that contains everything.

Special Offer:

A completely assembled ready-to-plug-in "PACER" can be purchased for an extra cost of $160.00. Just specify "PACER" Kit #2 on your order.

"PACER" Kit features these National Semiconductor Components.

Memory Board Components
Four MM2112N (256 x 4) RAM's for 256 Words. Space for both 12 more MM2112N and four MM5204Q (512 x 8) PROMS to expand memory.

Other Components
Mother Board to reduce interconnect wiring and make expansion easy. Power supply, +10 and – 16 volts. LM320 and LM323 series of on-card regulators. Assorted standard National Tri-State* Logic.

CPU Board Components
One National IPC-16A/500D "PACE" 16-Bit MPU with necessary input and output buffer components.

Control and I/O Board Components
Two DM8531D ROM's for 1K Words of system monitor. Four MM2112N (256 x 4) static RAM's. One MM5740N Keyboard Encoder. Two DS 8859N Hex Latch and LED Driver Circuits. All required support components to interface with the two 4-digit displays and a 32-key pad.

NATIONAL FROM Hamilton/Avnet ELECTRONICS

SOUTHERN CALIFORNIA
Avnet, (213) 558-2345
Hamilton Electro, (213) 558-2121
San Diego, (714) 279-2421

WEST
Albuquerque, (505) 765-1500
Denver, (303) 534-1212
Mountain View, (415) 961-7000
Phoenix, (602) 275-7851
Salt Lake City, (801) 262-8451
Seattle, (206) 746-8750

NORTH CENTRAL
Chicago, (312) 678-6310
Cleveland, (216) 461-1400
Dayton, (513) 433-0610
Detroit, (313) 522-4700
Minneapolis, (612) 941-3801
St. Louis, (314) 731-1144

MID-ATLANTIC
Baltimore, (301) 796-5000
Connecticut, (203) 762-0361
Long Island, (516) 333-5800
Cedar Grove, (201) 239-0800
Cherry Hill, (609) 234-2133

SOUTHERN
Atlanta, (404) 448-0800
Dallas, (214) 661-8661
Houston, (713) 526-4661
Huntsville, (205) 533-1170
Kansas City, (913) 888-8900
Miami, (305) 925-5401

NORTHEAST
Boston, (617) 273-2120
Rochester, (716) 442-7820
Syracuse, (315) 437-2642

CANADA
Montreal, (514) 331-6443
Ottawa, (613) 226-1700
Toronto, (416) 677-7432

INTERNATIONAL
Telex 67-3692

Call HAMILTON/AVNET