01448C PROGRAM DESCRIPTION I

Program Title PROGRAMMER PLUS

Contributor's Name James H. Boardman

Address 4321 North Bear Claw Way

City Tucson State/Country Arizona Zip Code 85715

Program Description, Equations, Variables This program is an aid to developing, debugging and testing computer routines involving Boolean functions and bit shifting operations. To provide maximum flexibility, signed or unsigned integers may be entered in one of 3 bases (hex, decimal or octal) and signed or unsigned results displayed in hex, decimal or octal (independent of input base). The program was specifically developed as a (superior) replacement for the TI PROGRAMMER calculator...RPN version, of course.

Rotation, complementation, justification and the common Boolean functions (AND, OR, EXCLUSIVE OR) are all provided. The program simulates a word size of anywhere between 2 and 32 bits (inclusive). Special provisions were made for allowing user written programs to call upon the various routines. Used in this way, the 41c program is transformed from a Boolean calculator to a powerful problem solving program aid. On page 23 you will find a detailed user written program example.

The next 29 pages give a complete description and more than 16 examples of using the PROGRAMMER PLUS program.

Necessary Accessories 4 memory modules OR quad memory OR HP 41CV

Operating Limits and Warnings Integers only for input (and output). Simulated word size must fall between 2 and 32 bits (inclusive). Hex integers with 9 or more digits may cause erroneous function results. Max hex integer digits: 12. Decimal integers greater than 4294967295 (2**32 - 1) may cause erroneous function results. Word size must always be set appropriately as explained on page 12.

Reference(s) Yaohan Chu, INTRODUCTION TO COMPUTER ORGANIZATION, Prentice-Hall, Inc. Englewood Cliffs, N.J., 1970

12-22-81

HP PROGRAMMER PLUS DOCUMENTATION

PROGRAM DESCRIPTION I
(continuation page)

1.0 TERMINOLOGY AND SYMBOLS

In the following discussion of functions and operations, the examples all assume you are starting from the "initialized state". Unless otherwise noted, the "initialized state" is reached by starting the program at step 1 followed by setting decimal output mode. Specific key presses to invoke USER mode functions will be designated by referring to the blue (ALPHA) letters. In the case of shifted functions, the word "GOLD" is used to denote that the shift key should be pressed first. Key sequences themselves are specified in a line separated by ";". For example, to achieve the initialized state, perform the following sequences:

GTO .001 ; R/S	DSPLY: "HEX: 41C"	FLGS: 1	MODE: U
D [DECO]	DSPLY: "DEC: 0"	FLGS: 1	MODE: U

The left portion of the above lines indicates which keys you should press. The label enclosed in square brackets is the name of the function being executed. DSPLY indicates what will appear in the display. FLGS indicates which flag annuciators will be on. MODE indicates which alpha annuciators will be on. (U = USER, A =ALPHA). Many of the examples in this section assume the display is in decimal mode output (as just achieved) and use decimal mode input which is the default mode after GTO .001 ; R/S. To illustrate the terminology further, let us proceed past the initialized state.

Example (1): HEX OUTPUT, OCTAL INPUT
From the initialized state:

E [HEXO]	DSPLY: "HEX: 0"	FLGS: 1	MODE: U
GOLD ; C [OCTI]	DSPLY: "HEX: 0"	FLGS: 1,3	MODE: U

Example (2): OCTAL OUTPUT, HEX INPUT
From the initialized state:

GOLD ; E [HEXI]	DSPLY: "DEC: 0"	FLGS: 1,4	MODE: U,A
ALPHA ; C [OCTO]	DSPLY: "OCT: 0"	FLGS: 1,4	MODE: U,A

Note that the ALPHA key had to be used to extinguish the ALPHA annunciator in order that the USER mode function "C" could be executed.

Return to initialized state:

ALPHA ; GOLD ; D [DECI]	DSPLY: "OCT: 0"	FLGS: 1	MODE: U
D [DECO]	DSPLY: "DEC: 0"	FLGS: 1	MODE: U

2.0 INPUT, OUTPUT MODES AND INTEGER ENTRIES

As you may have already deduced, the C, D and E functions specify the output mode (oCtal, Decimal and hEx) while the GOLD ; C, GOLD ; D, and GOLD ; E functions specify the corresponding input modes. In addition, flags 3 and 4 indicate the current input mode.

Flag 3	Flag 4	Input Mode
OFF	OFF	DECIMAL
OFF	ON	HEX
ON	OFF	OCTAL

Hex input mode is somewhat inconvenient for executing USER mode functions since the program leaves ALPHA mode on (for easy input of hex digits) when it stops with a new display. To defeat this feature, change step 191 from an AON instruction to AOFF. Now, however, it is inconvenient to enter hex digits (you must manually switch to ALPHA mode first). There is no overall convenient solution, but another alternative will be described later.

2.1 ENTERING INTEGERS

Before learning how to enter integers in the various modes, it may be helpful to understand that no matter what the display says ("HEX:", "DEC:" or "OCT:"), the stack registers ALWAYS contain numeric, decimal quantities (integers). The display always represents the base 8, base 10, or base 16 equivalent of the value contained in the X register.

2.1.1 POSITIVE INTEGERS

Decimal integers can be entered by either pressing the numeric digits 0 through 9 followed by R/S or followed by ENTER.

Octal integers can ONLY be entered by pressing the numeric digits 0 through 7 followed by R/S. Of course, the program must also be in the octal input mode (flag 3 "ON").

Hex integers can only be entered from the ALPHA keyboard (ALPHA mode on) followed by R/S. Since two key presses are required to enter each of the digits 0 through 9 (because the digits are shifted quantities in ALPHA mode) and since this can be rather inconvenient, the program accepts 2 representations for each of the 10 numeric digits. Either the digit itself OR the ALPHA letter on that same key can be used to enter a hex digit. For the other 6 hex digits, there is no problem and hence A through F are represented only by themselves. Thus, the hex quantity B75A can be entered either by:

B ; GOLD ; 7 ; GOLD ; 5 ; A ; R/S OR by B ; R ; W ; A ; R/S

The disadvantage of using the shorter key stroke sequence is that if the output mode is also hex, an internal decimal-to-hex conversion process will always be executed in order to display correct hex notation. If the longer input sequence is used and the output mode is hex, the internal conversion is not necessary since the output display process will use the same characters that were input.

2.1.2 NEGATIVE INTEGERS

For decimal and octal negative integers, use the same technique described above along with the usual CHS key press. In the case of hex input, you must actually enter a minus sign as the FIRST character of the ALPHA mode string. The minus sign is, of course, really the algebraic subtraction key. As with the numeric hex digits, the minus sign can be entered either by:

GOLD ; - OR by Q

2.2 OVER-RIDING THE PREVAILING INPUT MODE

Decimal input while in octal input mode: If the entered integer contains either the digit 8 or 9, the integer will be taken as a decimal quantity instead of octal.

Hex input while in either octal or decimal input mode: If ALPHA mode input is detected by the program (after pressing R/S), an attempt will be made to interpret the input as hex regardless of the prevailing input mode. If the attempt fails, the error message "IN ERR" will be momentarily displayed and the entry ignored. The stack contents will be unaffected and the previous display will return. If the attempt is successful, the equivalent decimal integer will be placed in the X register in a normal manner.

Numeric input while in hex input mode: If ALPHA mode is switched off and a numeric quantity entered, an attempt to interpret the input as octal will be made. If the attempt fails (integer contained either an 8 or 9) the entry will be taken as decimal.

3.0 USING THE PROGRAM FOR HEX, OCTAL, DECIMAL CONVERSIONS

It should now be evident that since the input mode is selected independently from the output mode, base conversion is automatic. Simply select the desired input and output modes. To illustrate, let's convert a series of positive integers from one base to another.

Example (3): OCTAL INPUT, HEX OUTPUT
From the initialized state, select the desired modes:

```
GOLD ; C [OCTI]          DSPLY: "DEC: 0"          FLGS: 1,3        MODE: U
E [HEXO]                 DSPLY: "HEX: 0"          FLGS: 1,3        MODE: U
```

Convert octal 3726347 to hex:

```
3726347 ; R/S            DSPLY: "HEX: FACE7"      FLGS: 1,3        MODE: U
```

Convert octal 7777 to hex:

```
7777 ; R/S               DSPLY: "HEX: FFF"        FLGS: 1,3        MODE: U
```

Example (4): HEX INPUT, DECIMAL OUTPUT
Continuing from the above example, select the desired modes:

```
D [DECO]                 DSPLY: "DEC: 4095"       FLGS: 1,3        MODE: U
GOLD ; E [HEXI]          DSPLY: "DEC: 4095"       FLGS: 1,4        MODE: U,A
```

Convert hex 13B05 to decimal:

```
GOLD ; 1 ; GOLD ; 3 ; B ; GOLD ; 0 ; GOLD ; 5 ; R/S
                         DSPLY: "DEC: 80645"      FLGS: 1,4        MODE: U,A
```

Convert hex 13B05 to decimal using short-hand notation:

```
Z?B W ; R/S              DSPLY: "DEC: 80645"      FLGS: 1,4        MODE: U,A
```

Once an integer has been entered, the prevailing OUTPUT mode can be changed to see that integer's value in any of the available base displays. Simply execute the desired output mode function (C, D or E). This action does not change the state of the stack registers or the LASTX register.

Example (5): DISPLAY DECIMAL 12345 in EACH of the AVAILABLE OUTPUT MODES
From the initialized state:

```
12345 ; R/S              DSPLY: "DEC: 12345"      FLGS: 1          MODE: U
C [OCTO]                 DSPLY: "OCT: 30071"      FLGS: 1          MODE: U
E [HEXO]                 DSPLY: "HEX: 3039"       FLGS: 1          MODE: U
```

Example (6): DISPLAY OCTAL 12345 in EACH of the AVAILABLE OUTPUT MODES
Continuing from the previous state:

GOLD ; C [OCTI]	DSPLY: "HEX: 3039"	FLGS: 1,3	MODE: U
12345 ; R/S	DSPLY: "HEX: 14E5"	FLGS: 1,3	MODE: U
D [DECO]	DSPLY: "DEC: 5349"	FLGS: 1,3	MODE: U
C [OCTO]	DSPLY: "OCT: 12345"	FLGS: 1,3	MODE: U

Example (7): DISPLAY HEX 12345 in EACH of the AVAILABLE OUTPUT MODES
Continuing from the previous state:

GOLD ; E [HEXI]	DSPLY: "OCT: 12345"	FLGS: 1,4	MODE: U,A
Z=?VW ; R/S	DSPLY: "OCT: 221505"	FLGS: 1,4	MODE: U,A
ALPHA ; D [DECO]	DSPLY: "DEC: 74565"	FLGS: 1,4	MODE: U,A
ALPHA ; E [HEXO]	DSPLY: "HEX: 12345"	FLGS: 1,4	MODE: U,A

4.0 INTRODUCTION TO BOOLEAN VARIABLES AND FUNCTIONS

If you are already familiar with Boolean algebra and functions, you will want to skip to the next section and proceed with how to use the program's built-in functions. Otherwise, this section is a brief introduction to the subject. It is not intended to make anyone an expert or even be especially thorough. The justification is simply to see why the particular built-in functions were chosen to be implemented.

Boolean variables differ from algebraic variables in that they are quantities which may have only one of two possible values. Ordinary algebraic quantities may take on many different values. The two values for a Boolean variable are typically denoted by 1 and 0 (also "true" and "false").

As in algebra where the addition, subtraction, multiplication and division operators are defined, there is a set of Boolean operators which specify "logical" operations on two Boolean variables. Given variables X and Y, consider some arbitrary function, F(x,y), the result of which is a Boolean value. You can easily see that either F(x,y)=0 or F(x,y)=1. Since there are four combinations of x and y for which F(x,y) yields a result, we must therefore enumerate the resulting values for F(x,y) in order to completely define the function. Once each of the 4 results is listed, we can consider F(x,y) to be an operator. The following table shows that there are 16 possible Boolean operators denoted by F1 through F16.

Possible results of a Boolean Operator on Two variables X and Y

X	Y	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

To illustrate how to read this table, consider F7(1,0). In this case, X=1 and Y=0 so refer to the third row under the X Y heading. Look across to the column labeled F7 and read the value. F7(1,0) = 1.

Examining this table reveals that there are several trivial functions. F1 and F16, for example, yield a constant result and are therefore independent of the particular set of "input" values for X and Y. Also, it is evident that F4(x,y) = X and F6(x,y) = Y making these also "trivial" functions. F11 and F13 are similarly a function of only one of the "input" values. But they are very important (as we shall see) in that they have the effect of reversing or complementing the input value.

Now consider the other, non-trivial functions. These are: F2, F3, F5, F7, F8, F9, F10, F12, F14 and F15. It turns out that if we select F2, F7, F8 and F13 as a group and call them fundamental or "basic" operators, then all of the remaining functions (F3, F5, F9, F10, F12, F14, F15) can be expressed in terms of the basic ones. For example, F3(x,y) = F7(x,F2(x,y)). This is demonstrated in the following two tables.

X	Y	F2(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

X	F2(x,y)	F3 = F7(x,F2(x,y))
0	0	0
0	0	0
1	0	1
1	1	0

If you can keep the parenthesis straight, it will be found that:

F5(x,y) = F7(F2(x,y),y)
F9(x,y) = F13(F8(x,y),y)
F10(x,y)= F13(F7(x,y),y)
F12(x,y)= F13(F7(F2(x,y),y),y)
F14(x,y)= F13(F7(x,F2(x,y)),y)
F15(x,y)= F13(F2(x,y),y)

Notice that only F2, F7, F8 and F13 are really needed in order to generate all possible results of functions on two Boolean variables. It turns out that these are not the only ones that could have been selected, nor do they constitute a minimum set. If you choose the right 2 functions, all the others can be generated from them. However, the ones chosen happen to correspond to machine instructions commonly implemented on most computers. F2 is called the "AND" function, F7 is called the "EXCLUSIVE OR" function, F8 is the "INCLUSIVE OR" (or simply "OR") function and, as mentioned above, F13 is the "COMPLEMENT" function.

So far, we have considered only Boolean variables - that is, variables which take on a value of only 0 or 1. Since algebraic variables can be represented by a string of 1's and 0's (in base 2 form or as binary numbers), the fundamental functions can be applied to algebraic variables by performing the function on each pair of corresponding binary digits. For example, consider applying the AND function to the base 10 integers 9 and 5.

Base 10 integer:	9	5
Base 2 equivalent:	1001	0101

Now, number the "bits" in the binary equivalents from right to left:

BIT0(9)= 1	BIT0(5)= 1
BIT1(9)= 0	BIT1(5)= 0
BIT2(9)= 0	BIT2(5)= 1
BIT3(9)= 1	BIT3(5)= 0

Next, perform the AND function on each pair of corresponding bits

```
BIT0(9) BIT0(5)          BIT0(9) "AND" BIT0(5)
  1       1                        1
BIT1(9) BIT1(5)          BIT1(9) "AND" BIT1(5)
  0       0                        0
BIT2(9) BIT2(5)          BIT2(9) "AND" BIT2(5)
  0       1                        0
BIT3(9) BIT3(5)          BIT3(9) "AND" BIT3(5)
  1       0                        0
```

Finally, recombine the bits to form the result:

(1001) "AND" (0101) = (0001) or 9 "AND" 5 = 1

The three other fundamental functions (EXOR, OR, and COMPL) are likewise always applied to individual bits. AND, EXOR and OR operate on corresponding bit pairs of the two variables whereas COMPL operates on each bit of a single variable. It is probably obvious that in the PRGRMR+ program, AND, EXOR, OR and COMPL are the built-in functions which were referred to before. AND, XOR, and OR use arguments in the 41C's X and Y registers, leaving their result in X with the normal stack "drop" associated with all diadic functions. COMPL works only on the value in X (as a good monadic function should). Naturally, all four functions leave the original contents of X in the LASTX register.

4.1 USING The BOOLEAN FUNCTIONS

The Boolean functions are executed by using the local ALPHA functions F ("AND"), G ("OR"), H ("EXOR") and J ("COMPL"). The following examples will be performed in octal input and output mode because it is very easy to visulize the individual bits that comprise each octal digit. An octal digit always represents 3 binary bits. (hex digits always represent 4 bits which is also fairly easy, but decimal digits do not correspond to an integral number of bits). Octal 0 = (000), octal 1 = (001), octal 2 = (010) etc. through octal 6 = (110), octal 7 = (111). Let's see what the result of various Boolean operations is on a few pairs of integers.

Example (8): FIND the RESULT of 4377 "AND" 2155 (octal integers)
From the initialized state:

C [OCTO]	DSPLY: "OCT: 0"	FLGS: 1	MODE: U
GOLD ; C [OCTI]	DSPLY: "OCT: 0"	FLGS: 1,3	MODE: U
4377 ; R/S	DSPLY: "OCT: 4377"	FLGS: 1,3	MODE: U
2155 ; R/S	DSPLY: "OCT: 2155"	FLGS: 1,3	MODE: U
F [AND]	DSPLY: "OCT: 155"	FLGS: 1,3	MODE: U

The next example will illustrate use of each of the two argument Boolean functions ("AND", "OR" and "EXOR"). For arguments we will choose two special decimal numbers: 172 and 202. To see why these are special, write down the binary equivalents:

172 = (10101100)
202 = (11001010)

Comparing corresponding pairs of bits, you will find that every possible combination of "0" and "1" will be exercised twice by the functions. Predicting the results, we find that:

(10101100) "AND" (11001010) = (10001000) [136]
(10101100) "OR" (11001010) = (11101110) [238]
(10101100) "EXOR" (11001010) = (01100110) [102]

Partition the result bits into 2 groups of 4 bits each and observe that each group is identical. The results in hex should be three double digit numbers: 88, EE and 66. Now duplicate the results with the program.

Example (9): FIND 172 "AND" 202, 172 "OR" 202 and 172 "EXOR" 202. OUTPUT the RESULTS in BOTH DECIMAL and HEX FORMS.
From the intialized state:

172 ; R/S	DSPLY: "DEC: 172"	FLGS: 1	MODE: U
202 ; R/S	DSPLY: "DEC: 202"	FLGS: 1	MODE: U
F [AND]	DSPLY: "DEC: 136"	FLGS: 1	MODE: U
E [HEXO]	DSPLY: "HEX: 88"	FLGS: 1	MODE: U
172 ; R/S	DSPLY: "HEX: AC"	FLGS: 1	MODE: U
202 ; R/S	DSPLY: "HEX: CA"	FLGS: 1	MODE: U
G [OR]	DSPLY: "HEX: EE"	FLGS: 1	MODE: U
D [DECO]	DSPLY: "DEC: 238"	FLGS: 1	MODE: U
172 ; R/S	DSPLY: "DEC: 172"	FLGS: 1	MODE: U
202 ; R/S	DSPLY: "DEC: 202"	FLGS: 1	MODE: U
H [EXOR]	DSPLY: "DEC: 102"	FLGS: 1	MODE: U
E [HEXO]	DSPLY: "HEX: 66"	FLGS: 1	MODE: U

5.0 WORD SIZE AND THE COMPLEMENT FUNCTION

Almost all the functions in the PRGRMR+ program make use of a "word size" constant. "Word size" means the number of binary digits (bits) that are considered to be available for a binary representation of an integer. Consider for example how one would represent the integer 5 in binary form. There are many choices depending on the word size.

WORD SIZE	BINARY REPRESENTATION OF 5
0	Not possible
1	Not possible
2	Not possible
3	101
4	0101
5	00101
6	000101
etc., etc.	

Perhaps this seems trivial, but consider what happens if one wishes to perform the COMPLement function. Exactly how many bits get reversed? The answer depends on the currently selected word size.

WORD SIZE	NUMBER 5	COMPL of 5	BASE 10 equiv.
3	101	010	2
4	0101	1010	10
5	00101	11010	26
6	000101	111010	58

To demonstrate this using the PRGMR+ program in USER mode, let's select a word size of 7, enter the number 5 and complement it to see the result.

Example (10): COMPLEMENT 5 with WORD SIZE = 7
From the initialized state:

	DSPLY: "DEC: 0"	FLGS: 1	MODE: U
A [WSIZ]	DSPLY: "32, NEW SIZ:"	FLGS: 1	MODE: U
7 ; R/S	DSPLY: "DEC: 0"	FLGS: 1	MODE: U

The program has now been set to have a word size of 7. This size will remain in effect indefinitely until it is again changed with the "A" function.

5 ; R/S	DSPLY: "DEC: 5"	FLGS: 1	MODE: U
J [COMPL]	DSPLY: "DEC: 122"	FLGS: 1	MODE: U

Evidently, the COMPL of 5 using a word size of 7 has a decimal value of 122. Obviously, if we reverse all the bits again by taking another COMPL, the original value should return:

```
J [COMPL]                DSPLY: "DEC: 5"          FLGS: 1        MODE: U
```

Starting with 5, find the COMPL for various word sizes:

```
A [WSIZ]                 DSPLY: "7, NEW SIZ:"     FLGS: 1        MODE: U
9 ; R/S                  DSPLY: "DEC: 5"          FLGS: 1        MODE: U
J [COMPL]                DSPLY: "DEC: 506"        FLGS: 1        MODE: U
A [WSIZ]                 DSPLY: "9, NEW SIZ:"     FLGS: 1        MODE: U
23 ; R/S                 DSPLY: "DEC: 506"        FLGS: 1        MODE: U
5 ; R/S                  DSPLY: "DEC: 5"          FLGS: 1        MODE: U
J [COMPL]                DSPLY: "DEC: 8388602"    FLGS: 1        MODE: U
```

The default word size for the program is 32. If you will be mostly dealing with machines whose word size is other than this, the default can be reset by changing step 96 from 32 to any INTEGER from 2 through 32. No program checks are made to ensure that you have selected a number in this range, but larger or smaller selections will cause incorrect results for most operations.

5.1 NEGATIVE INTEGERS AND TWO'S COMPLEMENT

So far we have considered only positive integers in our discussion of Boolean functions and word size. What about negative integers? How can they be represented in terms of a string of 1's and 0's? Almost all modern day computers use the left-most bit in the binary representation of a number to indicate whether it is dealing with a positive or negative integer. The left-most bit is commonly referred to as the "most significant bit" or MSB. Naturally, then, the right-most bit is the "least significant bit" or LSB. In most computers the MSB is used as a "sign" bit when the machine is dealing with signed integers. MSB=0 indicates a positive integer, MSB=1 indicates a negative integer.

Unlike written notation, however, where we negate a number simply by putting a "-" in front, the negative representation in binary form does NOT involve simply setting the MSB to 1. There are several reasons for this, probably the most important one being that when an integer and it's negative are added, the result should be 0 (zero). If all we did to represent a negative number was turn the MSB bit "on", this additive property could not be easily implemented in the machine's hardware. As an illustration, consider 5 (000101) and -5 (100101). If you perform a binary addition, the result is (101010) which would represent -10 instead of the desired 0.

Instead, -5 is represented by (111011). Now, when 5 and -5 are added the result is correct: (000101) + (111011) = ([1]000000). EXCEPT, there was a "carry" (represented by [1]) into the next higher, non-existent bit. This is no problem. Since there is no place to put

the overflow bit, it is simply discarded (or kept track of by the machine in some other form). This leaves all zeroes in the result word. How do you generate (111011) from -5? The process involves only two steps: start with the positive representation of the number (000101), COMPL it (111010), and add 1 (111011). This process is called "two's complementation". When carried out on any positive integer (MSB=0), the result will be the negative of that integer and the MSB will always be set to 1.

The COMPL process discussed in the previous section is called "one's complementation" and is most useful when we are not particularly interested in dealing with signed numbers. After all, there is nothing very special about the MSB. If we agree that all numbers in a program will be positive, then the MSB might as well just be used as part of the number itself. For cases where we DO care about the sign of an integer, it would be nice to be able to take a "two's complement" as well as a "one's complement". In the PRGRMR+ program, the "J" function is used for both kinds of complementing. The selection is governed by FLAG 1. If flag 1 is ON, then the J function performs a one's COMPL. If flag 1 is OFF, the J function performs a two's COMPL (i.e., it adds 1 to the result of a one's COMPL operation).

There is no function in the program to turn flag 1 ON or OFF. This is mainly because there are no more available local ALPHA functions available! All 15 are allocated for other functions. It is very easy, however, to manually SF 01 or CF 01 as appropriate. The program sets "one's COMPL" as the default (flag 1 ON) in step 84. Steps 1 through 102 are all initialization steps, performed only once when you GTO .001 ; R/S. Many program defaults are setup in these steps including flag 1 and can be changed to suit individual tastes. To set "two's COMPL" as the default for the J function, change step 84 from SF 01 to CF 01.

6.0 SIGNED MAGNITUDE DISPLAY

Using the sign bit (MSB) to distinguish between negative and positive numbers and then actually displaying a minus sign for negative numbers is called the "signed magnitude" display mode. Signed magnitude display mode is controlled by FLAG 2. If "ON", flag 2 signifies that integers with their MSB set to 1 will be displayed with a minus sign followed by the magnitude of the number. Flag 2, in turn, is controlled by the local ALPHA function "a". (GOLD ; A). Whenever GOLD ; A is executed, the program reverses the state of flag 2. If it was ON, it is turned OFF and vice versa. Notice that the default is for flag 2 to be OFF. To change the default, change step 85 from CF 02 to SF 02.

Continuing from the last example, display the current number in signed magnitude form. Recall that the program currently has the word size set to 23 and the display is showing the one's COMPL of 5.

Example (11): DISPLAY SIGNED MAGNITUDE VALUE for -5, WORD SIZE = 23

	DSPLY: "DEC: 8388602"	FLGS: 1	MODE: U
GOLD ; A [SGNM]	DSPLY: "DEC: -6"	FLGS: 1,2	MODE: U

What went wrong? Instead of "-5", we have "-6". The problem is that in the previous example, we took the "one's COMPL" of 5 instead of the "two's COMPL". Try again by taking the "one's COMPL" to get back a 5:

J [COMPL]	DSPLY: "DEC: 5"	FLGS: 1,2	MODE: U

Now, manually clear flag 1 so that the COMPL function will use the "two's COMPL" instead:

GOLD ; CF 01 ; J	DSPLY: "DEC: -5"	FLGS: 2	MODE: U

With the correct result at hand, you will note that subsequent presses of the J [COMPL] function will simply turn the minus sign on or off. We could have predicted the "erroneous" result of -6 above by thinking about the binary representation. Going back to 6 bits for the word size instead of 23, we had: 5 (000101). Then it was one's COMPLemented: 58 (111010). If we now examine this binary number with the idea in mind that it represents a negative number (MSB=1), what would that number be? To find out, take the two's COMPL: (000101) + (1) = (000110) or 6. Since the MSB is ON, (111010) must be the representation for -6 and, indeed, this is what was initially displayed above. The word size doesn't matter here. The results are the same since the one's COMPL of 5 in 23 bit form has 20 leading 1's instead of just 3 as in this example.

Signed magnitude display is most useful to programmers when they are examining binary "dumps" of memory from a computer. The dump programs almost always print out words in unsigned form (MSB is just another "bit"). However, the programmer may know that certain of the numbers are really signed integers. This display mode easily gives a translation of

the dumps in the same form as the programmer originally thought about the numbers.

To illustrate this procedure, let's translate a series of dumps from a hex form to the real signed decimal integers that they represent. Assume we have a 16 bit machine (word size = 16), and that a hex dump of the following numbers has occurred: F2A0, A35B, 7F6C, 90C2, D197.

Example (12): TRANSLATE HEX INTEGERS into DECIMAL SIGNED MAGNITUDE FORM
From the initialized state:

A [WSIZ]	DSPLY: "32, NEW SIZ:"	FLGS: 1	MODE: U
16 ; R/S	DSPLY: "DEC: 0"	FLGS: 1	MODE: U
GOLD ; A [SGNM]	DSPLY: "DEC: 0"	FLGS: 1,2	MODE: U
GOLD ; E [HEXI]	DSPLY: "DEC: 0"	FLGS: 1,2,4	MODE: U,A

Now the program is ready to accept the dumps. Remember that the hex digits can be entered either in long form using GOLD or short form form using just the letters on the corresponding keys. To make this description more concise, the individual key presses for long form won't be shown.

F2A0 ; R/S	DSPLY: "DEC: -3424"	FLGS: 1,2,4	MODE: U,A
A35B ; R/S	DSPLY: "DEC: -23717"	FLGS: 1,2,4	MODE: U,A
7F6C ; R/S	DSPLY: "DEC: 32620"	FLGS: 1,2,4	MODE: U,A
90C2 ; R/S	DSPLY: "DEC: -28478"	FLGS: 1,2,4	MODE: U,A
D197 ; R/S	DSPLY: "DEC: -11881"	FLGS: 1,2,4	MODE: U,A

7.0 ROTATION AND JUSTIFICATION

In many computer applications the programmer is not interested in dealing with integer values. Instead, he is more concerned with bit patterns. In other words, it is not a binary number that is being dealt with but the individual bits themselves. Perhaps the bits are considered to be flags as in the 41C or perhaps a group of bits taken together have some particular meaning. This is particularly true when programs deal with devices like terminals or line printers. To make a terminal print a particular letter, for example, it must transmit a certain pattern of bits to activate the mechanism for printing.

Computers usually have several instructions designed to handle bit patterns. This program has four functions which treat the X register in a similar way. All of them involve moving a pattern to the right or left.

7.1 RIGHT AND LEFT JUSTIFICATION

The justification functions cause continuous movement until a bit with the value 1 fills either the MSB or LSB position. The following table illustrates the effect of the left and right justify functions.

WORD SIZE	START PATTERN	RIGHT-JUSTIFY	LEFT-JUSTIFY
6	001000 [8]	000001 [1]	100000 [32]
7	0110010 [50]	0011001 [25]	1100100 [100]
8	10001000 [136]	00010001 [17]	10001000 [136]

To demonstrate this using the program, let's duplicate the results of the above table. Since binary modes for input and output are not available, we will use the numbers in square brackets which indicate the decimal value of the preceeding binary number.

Example (13): LEFT and RIGHT JUSTIFY USING VARIOUS WORD SIZES
From the initialized state:

A [WSIZ]	DSPLY: "32, NEW SIZ:"	FLGS: 1	MODE: U
6 ; R/S	DSPLY: "DEC: 0"	FLGS: 1	MODE: U
8 ; R/S	DSPLY: "DEC: 8"	FLGS: 1	MODE: U
B [RJY]	DSPLY: "DEC: 1"	FLGS: 1	MODE: U
GOLD ; B [LJY]	DSPLY: "DEC: 32"	FLGS: 1	MODE: U
A [WSIZ]	DSPLY: "6, NEW SIZ:"	FLGS: 1	MODE: U
7 ; R/S	DSPLY: "DEC: 32"	FLGS: 1	MODE: U
50 ; R/S	DSPLY: "DEC: 50"	FLGS: 1	MODE: U
B [RJY]	DSPLY: "DEC: 25"	FLGS: 1	MODE: U
GOLD ; B [LJY]	DSPLY: "DEC: 100"	FLGS: 1	MODE: U
A [WSIZ]	DSPLY: "7, NEW SIZ:"	FLGS: 1	MODE: U
8 ; R/S	DSPLY: "DEC: 100"	FLGS: 1	MODE: U
136 ; R/S	DSPLY: "DEC: 136"	FLGS: 1	MODE: U
B [RJY]	DSPLY: "DEC: 17"	FLGS: 1	MODE: U
GOLD ; B [LJY]	DSPLY: "DEC: 136"	FLGS: 1	MODE: U

7.2 RIGHT AND LEFT ROTATION

The left and right rotation functions are similar to the justify functions in that bit patterns are moved right or left. Now, however, you are asked how many shifts to the right or left are wanted. If you respond with a positive integer, the shifts will take place to the right. For left shifts, input a negative number. There is no maximum number of shifts. However, if your request is such that a 1 bit would shift off the end of the binary string, the bit is not lost. Bits shifted off the end cause flag 0 to turn on. Before the next shift movement is performed, flag 0 is checked. If it is ON, a 1 will shift into the opposite end of the binary string. Otherwise, zeroes shift into the opposite end. This whole operation is commonly known as rotation with end-around carry. It is NOT true shifting (in which bits moving off one end are lost). Actually, flag 0 is effectively an extra bit in the binary string.

The rotation operations are the only routines in the entire program that do anything with flag 0. You can always see whether a bit is being held before a rotation function is executed. Flag 0 is NOT cleared prior to the operation so that anything "left-over" from a previous rotate will have an effect on the next rotate. If this is undesirable for you, flag 0 can be always cleared at the beginning of an operation by inserting a CF 00 instruction just AFTER step 339. Step 339 contains an SF 08 instruction.

The following table illustrates the rotation function as it proceeds step by step. The contents of flag 0 is indicated within parenthesis and the equivalent decimal value is indicated in square brackets.

DIR.	SIZ	START PATTERN	AFTER 1 ROT	AFTER 2 ROTS	AFTER 5 ROTS
right	8	(0)00100010 [34]	(0)00010001	(1)00001000	(0)00100001 [33]
left	5	(1)00010 [2]	(0)00101	(0)01010	(0)10001 [17]

Again, let's duplicate the results in the table by running the same process on the 41C.

Example (14): ROTATE DECIMAL 34 to the RIGHT 5 PLACES USING WORD SIZE 8
From the initialized state:

A [WSIZ]	DSPLY: "32, NEW SIZ:"	FLGS: 1	MODE: U
8 ; R/S	DSPLY: "DEC: 0"	FLGS: 1	MODE: U
34 ; R/S	DSPLY: "DEC: 34"	FLGS: 1	MODE: U
I [ROT]	DSPLY: "PLACES?"	FLGS: 1	MODE: U
5 ; R/S	DSPLY: "DEC: 33"	FLGS: 1	MODE: U

To duplicate the left rotation table example, note that flag 0 must be set before doing the rotation. This can, of course, be easily done with a manual SF 00. However, to be slightly tricky, we can set flag 0 by doing one more, single place right rotation. This will serve to demonstrate that the state of flag 0 carries over from one rotation execution to the next.

Example (15): ROTATE DECIMAL 2 to the LEFT 5 PLACES USING WORD SIZE 5
Continuing from the previous state:

I [ROT]	DSPLY: "PLACES?"	FLGS: 1	MODE: U
1 ; R/S	DSPLY: "DEC: 16"	FLGS: 0,1	MODE: U
A [WSIZ]	DSPLY: "8, NEW SIZ:"	FLGS: 0,1	MODE: U
5 ; R/S	DSPLY: "DEC: 16"	FLGS: 0,1	MODE: U
2 ; R/S	DSPLY: "DEC: 2"	FLGS: 0,1	MODE: U
I [ROT]	DSPLY: "PLACES?"	FLGS: 0,1	MODE: U
5 ; CHS ; R/S	DSPLY: "DEC: 17"	FLGS: 1	MODE: U

7.3 ROTATIONS IN COMBINATION WITH BOOLEAN FUNCTIONS

The ability to use bit pattern functions in combination with the Boolean functions and the normal 41C operations (+, -, *, /) will now be demonstrated. In using combination functions, very complex operations executed at high speeds on a computer can be duplicated and minutely "examined" by a programmer. The example here will be a rather simple one.

Suppose we wish to split a 16 bit word into two 8 bit halves (commonly called "bytes"). After the split, the left half must be one's complemented and added to the right half. Splitting a word is most easily accomplished with the "AND" function. The procedure will be to load a "mask" value to be used with the "AND" function into the Y register. The 16 bit word to be split will then be loaded into the X register and the "AND" performed. This will isolate the right half in X. Since the original 16 bit word is still in LASTX, we will recall it, load another "mask" value (this time into X), left justify the mask (since we now want the left half of the word) and perform another "AND". Finally, the isolated left half-word (now in X) will be complemented, rotated 8 places to the right and added to Y (which still contains the right half-word). The final result is to be displayed in hex mode. Let's pick the octal number 125252 as the value to be split. The mask value for isolating bytes is octal 377. Again, when dealing with bit patterns, it is easier to work with octal or hex modes in order to visulize the individual bits.

Example (16): COMBINATION OPERATIONS to ISOLATE and ADD HALF-WORDS
From the initialized state:

C [OCTO]	DSPLY: "OCT: 0"	FLGS: 1	MODE: U
GOLD ; C [OCTI]	DSPLY: "OCT: 0"	FLGS: 1,3	MODE: U
A [WSIZ]	DSPLY: "32, NEW SIZ:"	FLGS: 1,3	MODE: U
16 ; R/S	DSPLY: "OCT: 0"	FLGS: 1,3	MODE: U
377 ; R/S	DSPLY: "OCT: 377"	FLGS: 1,3	MODE: U
125252 ; R/S	DSPLY: "OCT: 125252"	FLGS: 1,3	MODE: U
F [AND]	DSPLY: "OCT: 252"	FLGS: 1,3	MODE: U
GOLD ; LASTX ; C [OCTO]	DSPLY: "OCT: 125252"	FLGS: 1,3	MODE: U
377 ; R/S	DSPLY: "OCT: 377"	FLGS: 1,3	MODE: U
GOLD ; B [LJY]	DSPLY: "OCT: 177400"	FLGS: 1,3	MODE: U
F [AND]	DSPLY: "OCT: 125000"	FLGS: 1,3	MODE: U
J [COMPL]	DSPLY: "OCT: 52777"	FLGS: 1,3	MODE: U
I [ROT]	DSPLY: "PLACES?"	FLGS: 1,3	MODE: U
8 ; R/S	DSPLY: "OCT: 177125"	FLGS: 0,1,3	MODE: U
377 ; R/S	DSPLY: "OCT: 377"	FLGS: 0,1,3	MODE: U
F [AND]	DSPLY: "OCT: 125"	FLGS: 0,1,3	MODE: U
+ ; E [HEXO]	DSPLY: "HEX: FF"	FLGS: 0,1,3	MODE: U

Note particularly in the above example that after issuing a standard 41C function (LASTX and +) the "standard" display for this program is obliterated and the decimal result of the operation appears instead. That is the reason for executing a program output mode function after LASTX. After the "+", the hex output mode function was executed because the example called for a hex display of the result.

8.0 USING THE FUNCTIONS AS SUBROUTINES FROM YOUR OWN PROGRAM

Special care was taken in the design of this program to make all the functions usable as subroutines from your own program. For this reason, the 41C instructions for each function begin with two labels. One of the labels is referenced when you execute the local ALPHA function. The other label, which is a global ALPHA, is to be referenced from user written programs.

8.1 LABEL CONVENTIONS

The global ALPHA labels all consist of double letters. The "AND" function, for example, has the local ALPHA label: *LBL F and the global ALPHA label *LBL "FF". Similarly, the global label for "OR", "EXOR", "ROT", "COMPL", "LJY" and "RJY" respectively are: *LBL "GG", *LBL "HH", *LBL "II", *LBL "JJ", *LBL "bb" and *LBL "BB".

8.2 THE ROTATE FUNCTION

Among these functions, only the rotate ("ROT") requires a keyboard prompt response before function execution proceeds. Recall that [ROT] asks: "PLACES?". To avoid the prompt when using "ROT" from a user program, the program must supply in register 22 a value indicating how many shifts are desired and in which direction. In other words, prior to a call to the "ROT" function, register 22 must be pre-loaded with whatever PROMPT response would have been keyed in. Negative integers cause left rotation, positive integers rotate X to the right. If you WANT the "ROT" function to PROMPT, just make sure register 22 is clear before calling the subroutine.

Note that during execution of ANY function, register 22 is always cleared just before returning to your program. So, it must be loaded everytime you call "ROT" (except when you WANT the prompt).

Don't forget about flag 0! The "ROT" function includes flag 0 as a bit in the simulated word. Your program must set or clear flag 0 appropriately to achieve correct results.

8.3 THE COMPLEMENT FUNCTION

Recall that flag 1 governs the [COMPL] function (global ALPHA label "JJ"). If "ON", the COMPL function performs a one's complement on X. If "OFF", a two's complement is performed. Your program must set or clear flag 1 accordingly.

8.4 CHANGING THE WORD SIZE

Register 25 and 26 always contain values pertinent to the current simulated word size. Register 25 contains the integer number of bits in the simulated word. Register 26 contains 2 raised to the power of the integral number of bits. Thus, to change the word size to 23 bits, say, your program would execute:

2 ; ENTER ; 23 ; STO 25 ; Y**X (Y up-arrow X) ; STO 26

Remember that erroneous results will occur if the word size is not in the range 2 to 32 (inclusive).

8.5 DISPLAYING RESULTS

As with the Boolean functions, the 3 output mode functions are also equipped with a double set of labels. Global label "CC" displays the current X value in octal form, "DD" displays in decimal form and "EE" displays in hex form. Calling any of these labels does not change the state of either the stack or the LASTX register, so your program can easily display intermediate or final results in whatever form you wish.

Setting flag 2 "ON" will force the display to be in signed magnitude form, flag 2 "OFF" results in the unsigned display.

8.6 ENTERING HEX INTEGERS FROM YOUR PROGRAM

There is one more global ALPHA label which can be called to convert a hex integer generated by your program into decimal. Global label "HX" provides this function. The procedure is to load the ALPHA register with the hex digits and XEQ "HX". Your ALPHA string will be converted to a decimal equivalent in the X register. The normal stack lift process will also occur just as though your program had directly entered the decimal value instead.

8.7 USER PROGRAM EXAMPLE

To illustrate the process, we will write a program to duplicate the results of Example (16). First, ensure you are in your own program space by executing GTO .. Switch to PRGM mode and key in the following program:

STEP	INST	ARGUMENT	COMMENTS
01	*LBL	"SPLIT"	
02	XEQ	"CC"	Force octal display
03	2		Change word size
04	ENTER		to 16 bits
06	16		
07	STO	25	
08	Y**X		Also store 2**16
09	STO	26	
10	377		Generate "mask"
11	DEC		in octal
12	125252		Load number to be split
13	DEC		also in octal
14	XEQ	"FF"	Perform "AND" function
15	LASTX		Recover number
16	377		Generate another mask
17	DEC		
18	XEQ	"bb"	Left justify the mask
19	XEQ	"FF"	and extract left byte
20	SF	01	Take one's complement
21	XEQ	"JJ"	(call the COMPL routine)
22	8		Setup for right rotate
23	STO	22	(this does a stack lift)
24	RDN		Undo the lift
25	CF	00	Ensure flag 0 has a "0"
26	XEQ	"II"	Finally, "ROT"
27	377		One more mask
28	DEC		
29	XEQ	"FF"	Extract just the left byte
30	+		Right byte was in Y
31	XEQ	"EE"	Display in hex form
32	STOP		and stop with display
33	GTO	"SPLIT"	Repeat if desired
34	END		That's it

Position the 41C to step 01 and press R/S. The final display, if you loaded the program correctly, will be "HEX: FF" -- just as in Example (16) where all of this was done manually. On my 41CV with fairly fresh batteries, the above program executes in 58 seconds.

8.8 AVAILABLE STORAGE REGISTERS

The PRGRMR+ program makes use of all registers from 00 up to and including register 75. So, to have your own registers, you must execute a SIZE 077 or higher. However, registers 21, 22, 23 and 24 are temporaries. That is, they are used only when functions are actually being executed and their contents do not have to be maintained between function calls. Therefore, if you only need a few registers BETWEEN calls to the Boolean routines, go ahead and use these. But remember that their contents will be destroyed after a return to your program.

9.0 CONVENIENCE SUGGESTIONS

When using the functions in "calculator" mode (not from a user program), I find it inconvenient to execute a "-", "+", "*", "/" etc. and have the display obliterated. Also, of course, when a "+" or whatever is executed you wind up looking at the X register in its raw decimal form. As a result, one winds up always executing [OCTO] or [HEXO]. To avoid this, several tiny routines can be added to the PRGRMR+ program which perform the desired function AND end up re-creating the desired display. Pick your own global ALPHA labels (avoiding, however, the ones already used!) and ASN your routine(s) to the corresponding key(s). The routine should always begin with the global label you want (of course), perform the function you want, and return by XEQ 93 ; GTO 21.

For example, to implement a handy routine for displaying the results of a "+", you could install the following routine:

```
*LBL "+"
 +
 XEQ 93
 GTO 21
```

The best place to insert routines like this is just after step 256 (your routine would start at step 257). When placed there, you can use an already defined sequence of "XEQ 93 ; GTO 21" which exists at local label 14. (Label "14d" in the listing). Placement there also does not have the effect of slowing down hex input dispatch calls which do global ALPHA calls for each hex digit. (Notice that all the hex input labels are defined at the end of the program so that the 41C will find them as quickly as possible.) Using this technique, your "+" routine would become:

```
257     *LBL "+"
258      +
259      GTO 14
```

I typically install a number of these routines for convenience. In addition to the other algebraic operators, ENTER, RDN, LASTX, etc. are other good candidates. Such routines are not included in the PRGRMR+

program as received from the library because they would require the assignment of a global ALPHA label to the corresponding key. This is not appropriate according to library standards.

Of course, to make use of the above "+" routine, you must finish the job by:

```
ASN ; ALPHA ; GOLD ; + ; ALPHA ; +
```

10.0 PROGRAM SIZE STATISTICS AND REDUCTION TECHNIQUES

The program as supplied occupies 269 registers and references all registers between 00 and 75 (inclusive). Therefore, a SIZE 076 must be performed before the program will run. The program fits on 7 cards, barely overflowing onto the 13th track. As a result of all this, it needs either a 41CV or a 41C with 4 memory modules.

The program can be reduced in size significantly. One of the first things one could do is to create a data card which would hold all the constants created in steps 4 through 81. All these steps could then be deleted and replaced by an RDTA instruction. Simply load the initialization routine by itself (replacing step 82 with a STOP), execute it, and then write your data cards. 5 tracks will be required. Since initialization is a once only process, the entire section could be deleted after one execution. Another thing which will reduce the size is to change or delete all the global ALPHA labels. EXCEPT, the hex input dispatch labels which all begin with *LBL "?". Even these could be deleted if you change the hex input conversion routine. (In that case, you would not need any of the steps from 543 to the end). I tried several different hex conversion techniques and chose this one for speed reasons. A simple alternative method would be to split off each hex digit (as is done already, except you would need to get rid of the leading "?") and fall into an ISG loop comparing the digit with the ALPHA constants stored in registers 00 through 15.

Eliminating the shifting displays would save some space also. To do this, eliminate subroutine 96, change everything referencing 96 to reference 93 instead, and delete the ALPHA display constants appearing near the start of each function. The "ROT" function sets up its own shifting display, so you would want to eliminate steps 333 through 336 as well.

If you do all of the things mentioned, the program should fit into a 41C with only 2 extra memory modules. However, this is an estimate. I have not actually tried it.

11.0 PROGRAM ORGANIZATION AND LOGIC

The PRGRMR+ program can be grouped into 9 major sections. They are: initialization, output display processing, integer input conversion processing, mode setting routines, Boolean function routines, rotation routines, justification routines, utility routines and hex input dispatch routines.

11.1 INITIALIZATION (steps 1 Through 102)

The bulk of this section is involved in setting 64 registers to hold the ALPHA constants for hex output and the Boolean function tables. The rest of the initialization sets up default operation modes and parameters. Setting defaults differently is perfectly possible. The main things to be careful about are:

1.) Register 25 (word size) must contain a value between 2 and 32. Register 26 must be set to 2 raised to the power of this value.

2.) Flags 3 and 4 should never both be "ON". All other combinations are valid and govern the input mode:

```
3 "OFF", 4 "OFF"  implies decimal input
3 "OFF", 4 "ON"   implies hex input
3 "ON",  4 "OFF   implies octal input
```

3.) Similarly, flags 5 and 6 should never both be "ON".

```
5 "OFF", 6 "OFF"  implies decimal output
5 "OFF", 6 "ON"   implies octal output
5 "ON",  6 "OFF"  implies hex output
```

4.) Since I don't have a printer, I did not attempt to take printer displays into account except to simply clear flag 21 at initialization.

5.) Decimal point and grouping is turned off (flag 27) since it would be difficult to provide consistency between modes and since the program is only designed to deal with integers.

11.2 OUTPUT DISPLAY (steps 103 Through 195)

A significant complexity factor in this section is the logic to correctly handle signed magnitude output. Steps 103 through 111 provide various entry points for program routines to finish up by creating the next display. Actual display generation is fairly simple except for the hex output section (steps 139 through 184). Processing always terminates at step 195 with an RTN instead of a PROMPT so that user written programs can call the various functions as subroutines.

11.3 INTEGER INPUT CONVERSION (steps 196 Through 256)

Once again, fairly simple routines except for hex input. The main problem with hex input is in splitting off each ALPHA character. Once separated, each digit is used as part of a dispatch address which is called at step 241 to load the corresponding hex value into X. The hex input routine loops until all of the ALPHA input is exhausted (up to 12 characters) or until a non hex character is encountered.

11.4 MODE SETTING (steps 257 Through 285)

A set of simple routines to set flags appropriately for the various input/output modes. User added routines should go in this area of the program.

11.5 BOOLEAN FUNCTIONS

11.5.1 (steps 286 Through 318)

This section "sets things up" for the "AND", "OR" and "EXOR" functions. The major work is really done by the routines at labels 90 and 94. Note the pointers to the Boolean tables. They point to the first register used for each function.

11.5.2 (steps 397 Through 409)

These few steps perform the COMPLement functions. If flag 1 is "ON", a one's complement is done. If "OFF", a two's complement is performed. Either operation affects only the X and LASTX register.

11.6 ROTATION (steps 319 Through 396)

This section includes the PROMPT for the number of places to shift. It also checks register 22 to see if a user program has already set up a shift parameter value. Recall that negative values cause left rotates, positive values cause right rotates and that flag 0 is involved as a bit in the end-around carry process.

11.7 JUSTIFICATION (steps 414 Through 445)

Fairly straightforward routines.

11.8 UTILITY SUBROUTINES

11.8.1 LABELS 90, 94

These routines do the bulk of the work for the "AND", "OR" and "EXOR" functions. Routine 94 loops until all bits from the smallest of the two arguments are used up. Bits are split off each argument two at a time and used to generate pointers to the Boolean tables for looking up the result of the operation.

11.8.2 LABEL 91

Used in conjunction with setting the simulated word size.

11.8.3 LABELS 92, 93

These routines maintain stack integrity. 93 saves the stack, 92 restores it.

11.8.4 LABEL 96

A stack save routine (falls into 93) which starts the shifting display going. The idea for this came from Patrick Shibli in HP Key Notes V5N1p12c. It does slow execution somewhat and can be eliminated with no loss except for the info contained in the display.

11.8.5 LABEL 95

Changes a negative number to a positive equivalent for proper operation of various functions.

11.9 HEX INPUT DISPATCHES (steps 543 Through 616)

Placed at the end of the program for shortest possible global ALPHA label search time.

12.0 SUMMARY

This completes the description of the functions and capabilities of the PRGRMR+ program. It may be helpful to review the name of each function and give a short description of what they do.

NAME	KEY PRESS	DESCRIPTION
[SGNM]	GOLD ; A	Sets/clrs flag 2 to govern signed magnitude mode.
[WSIZ]	A	Sets internal simulated word size (2 to 32).
[LJY]	GOLD ; B	Left justifies X. MSB set to 1 in simulated word.
[RJY]	B	Right justify X. LSB set to 1 in simulated word.
[OCTI]	GOLD ; C	Sets flag 3, clrs flag 4. Specifies octal input.
[OCTO]	C	Specifies octal output display.
[DECI]	GOLD ; D	Clrs flag 3 and flag 4. Specifies decimal input.
[DECO]	D	Specifies decimal output display.
[HEXI]	GOLD ; E	Clrs flag 3, sets flag 4. Specifies hex input.
[HEXO]	E	Specifies hexadecimal output display.
[AND]	F	Boolean "AND" between X and Y. Stack drops.
[OR]	G	Boolean "OR" between X and Y. Stack drops.
[EXOR]	H	Boolean "EX OR" between X and Y. Stack drops.
[ROT]	I	Rotate X right (+ve response to prompt).
		Rotate X left (-ve response to prompt).
[COMPL]	J	One's complement of X (flag 1 "on")
		Two's complement of X (flag 2 "off")

Recall that, like the standard 41C monadic and diadic functions, anytime a function modifies the X register, a copy of the original is placed in LASTX.

ADDENDUM

In section 2.0 page 4, it was mentioned that an alternative method for hex digit entry and function convenience would be described later. The idea is to spell out the desired function name, preceeding it with a ":". This is slightly more convenient (I think) than switching out of ALPHA mode to execute the function. If nothing else, the technique is interesting. To execute the "AND" function, for example, while in hex input mode, you would enter ":FF" ; R/S.

All that is needed to implement this feature is a function dispatch routine which executes when a failure is encountered during hex digit input processing. (The ":" will cause this failure to occur). Here are the needed steps:

GTO .243 to position the 41C at the appropriate step for insertion

STEP	Key Entry		Comment
244	XEQ	92	Restore stack
245	SF	25	Protect against error
246	GTO IND	24	Dispatch to function

Recall that all functions are labeled withdouble letter global ALPHA labels (section 8.1, page 21). These are the names that you would use with this technique.

PROGRAM DESCRIPTION II

Sample Problem (Sketch if Desired)

Henry Programmer is developing a subroutine for his home computer to encode 4 octal integers into a non-unique hex integer. To see if it is working OK, he needs an independent way to find results from the function that his subroutine computes. His encoding function is: (n1 "AND" n2) "EXOR" (n3 "OR" n4). Given 4 octal integers: n1=67271, n2=73333, n3=44504 and n4=106120, Henry's subroutine comes up with the HEX result: "BAD1". Is Henry's subroutine functioning correctly?

As shown below, (67271 "AND" 73333) "EXOR" (44504 "OR" 106120) = "ABCD" (hex)
Therefore, Henry's subroutine must have a bug.

Note that if the program has already been initialized in a previous problem, the first step (initialization) is not necessary. In this case, however, it may be necessary to press function (E) as the first step to achieve HEX output mode.

SOLUTION:

Input	Function	Display	Comments
GTO .001	R/S	HEX: 41C	Initialize
	GOLD (C)	HEX: 0	Set for octal input
67271	R/S	HEX: 6EB9	Enter n1
73333	R/S	HEX: 76DB	Enter n2
	(F)	HEX: 6699	Display (n1 "AND" n2)
44504	R/S	HEX: 4944	Enter n3
106120	R/S	HEX: 8C50	Enter n4
	(G)	HEX: CD54	Display (n3 "OR" n4)
	(H)	HEX: ABCD	(n1 "AND" n2) "EXOR" (n3 "OR" n4)

USER INSTRUCTIONS

SIZE: (HP-41C) 076

STEP	INSTRUCTIONS	INPUT	FUNCTION	DISPLAY
1.	Enter the program			
2.	Initialize	GTO .001	R/S	HEX: 41C
3.	Set modes as desired			
	-decimal input (default)		GOLD D	HEX: 0
	-hex output (default)		E	HEX: 0
	-signed magnitude OFF (default)		GOLD A	toggles flag 2
	-COMPL = 1's complement (default)	SF 01		
	-octal input		GOLD C	HEX: 0
	-hex input		GOLD E	HEX: 0
	-octal output		C	OCT: 0
	-deciaml output		D	DEC: 0
	-signed magnitude ON (flag 2 ON)		GOLD A	toggles flag 2
	-COMPL = 2's complement	CF 01		
	-WSIZ (see USER DES. I, sect 5.0, pg 12)		A	"n, NEW SIZ:"
4.	Enter numbers, perform functions as you would with any HP calculator using X,Y,Z,T and LASTX.			
	- terminate integer entry with R/S - for hex entries, enter either numeric digits or ALPHA letter on corresponding key. If negative, always enter minus sign first. - default WSIZ is 32 bits			
5.	Diadic functions (stack drops)			
	-AND		F	X "AND" Y
	-OR		G	X "OR" Y
	-EXOR (exclusive OR)		H	X "EXOR" Y
6.	Monadic functions (operates on X)			
	-RJY (right justify)		B	X, right justified
	-LJY (left justify)		GOLD B	X, left justified
	-ROT (rotate left or right) enter nmbr of places to shift enter -ve number to ror left, enter +ve number to rot right.	n	I R/S	"PLACES?" X, rotated
	-COMPL		J	X, complemented

Step	Key Entry		Comments
1	*LBL "HP+"		Start of prog
2	"HP PRGRMR+"		
3	XEQ	96	INITIALIZATION
4	16.075		Init. Boolean
5	ENTER		table
6	0		constants
7	*LBL	09	(L09a)*
8	STO IND	Y	
9	ISG	Y	
10	GTO	09	(G09a)
11	1		
12	STO	33	
13	STO	35	
14	STO	41	
15	STO	45	
16	STO	48	
17	STO	49	
18	STO	61	
19	STO	64	
20	STO	71	
21	STO	74	
22	2		
23	STO	38	
24	STO	39	
25	STO	42	
26	STO	46	
27	STO	52	
28	STO	54	
29	STO	62	
30	STO	67	
31	STO	68	
32	STO	73	
33	3		
34	STO	43	
35	STO	47	
36	STO	50	
37	STO	51	
38	STO	53	
39	STO	55	
40	STO	56	
41	STO	57	
42	STO	58	
43	STO	59	
44	STO	63	
45	STO	66	
46	STO	69	
47	STO	72	
48	"0"		Init. hex
49	ASTO	00	ALPHA
50	"1"		constants
51	ASTO	01	
52	"2"		
53	ASTO	02	
54	"3"		
55	ASTO	03	
56	"4"		
57	ASTO	04	
58	"5"		
59	ASTO	05	
60	"6"		
61	ASTO	06	
62	"7"		
63	ASTO	07	
64	"8"		
65	ASTO	08	
66	"9"		
67	ASTO	09	
68	"A"		
69	ASTO	10	
70	"B"		
71	ASTO	11	
72	"C"		
73	ASTO	12	
74	"D"		
75	ASTO	13	
76	"E"		
77	ASTO	14	
78	"F"		
79	ASTO	15	
80	"10"		
81	ASTO	16	
82	FIX	0	
83	CF	00	
84	SF	01	
85	CF	02	
86	CF	03	
87	CF	04	
88	SF	05	
89	CF	06	
90	CF	09	
91	CF	10	
92	CF	21	
93	SF	27	
94	2		
95	CF	29	
96	32		
97	XEQ	99	(X99a)
98	"41C"		
99	ASTO	22	
100	CLA		
101	ASTO	23	
102	GTO	20	
103	*LBL	23	OUTPUT
104	"X>WRD SIZ"		PROC.

Note: Refer to "HP-41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

Step	Key Entry		Comments
105	AVIEW		
106	*LBL	22	
107	RCL	17	
108	*LBL	21	
109	SF	10	
110	STO	17	
111	*LBL	20	
112	X<0?		
113	GTO	13	(G13a)
114	X#0?		
115	FC?	02	
116	GTO	14	(G14a)
117	ST+	X	
118	RCL	26	
119	X<=Y?		
120	ST-	17	
121	RCL	17	
122	*LBL	13	(L13a)*
123	FC?	02	
124	XEQ	95	
125	STO	17	
126	*LBL	14	(L14a)*
127	CF	25	
128	FS?	05	
129	GTO	13	(G13b)
130	"OCT: "		Octal output
131	FC?	06	
132	SF	25	
133	FC?	06	
134	OCT		
135	FC?C	25	
136	"DEC: "		Decimal output
137	ARCL	X	
138	GTO	14	(G14b)
139	*LBL	13	(L13b)*
140	"HEX: "		Hex output
141	FS?	10	
142	GTO	12	(G12b)
143	X<0?		
144	GTO	11	(G11a)
145	FS?	09	
146	GTO	12	(G12b)
147	*LBL	10	(L10a)*
148	ARCL	22	
149	23		
150	GTO	13	(G13c)
151	*LBL	11	(L11a)*
152	FS?	09	
153	GTO	10	(G10a)
154	*LBL	12	(L12b)*
155	X<0?		
156	|- "-"		
157	ABS		
158	STO	21	
159	SF	25	
160	LOG		Setup hex
161	16		digit
162	LOG		loop
163	/		cntr
164	INT		
165	X=0?		
166	CF	25	
167	X<>	21	
168	FC?C	25	
169	GTO	13	(G13c)
170	*LBL	01	(L01a)*
171	STO	Y	Loop to
172	16		generate
173	RCL	21	hex
174	Y**X		digits
175	STO	Z	
176	/		
177	INT		
178	ARCL IND X		
179	*		
180	-		
181	DSE	21	
182	GTO	01	(G01a)
183	*LBL	13	(L13c)*
184	ARCL IND X		
185	*LBL	14	(L14b)*
186	CLX		Clr "ROT"
187	STO	22	parm. val.
188	XEQ	92	Restore stk
189	AOFF		
190	FS?	04	
191	AON		(hex input)
192	CF	22	
193	CF	23	
194	AVIEW		View
195	RTN		results
196	FS?	23	ALPHA inp?
197	GTO	14	(G14c)
198	FC?	22	Num. inp?
199	RCL	X	(no,dup. X)
200	XEQ	93	Save stk
201	FS?	22	Num. inp?
202	FC?	03	Oct. mode?
203	GTO	21	(no, done)
204	SF	25	Cnvrt oct
205	DEC		to dec
206	STO	17	
207	GTO	21	Done
208	*LBL "HX"		Hex inp

Note: Refer to "HP-41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

Step	Key Entry		Comments
209	*LBL	14	(L14c)*
210	XEQ	93	
211	CLX		
212	ASTO	22	
213	ASTO	24	
214	ASHF		
215	ASTO	23	
216	SF	07	
217	CF	09	
218	CF	10	
219	"?????"		Setup stk
220	ASTO	Z	
221	*LBL	02	(L02a)*
222	16		
223	ST*	21	Build dec
224	RDN		equiv.
225	ST+	21	
226	RDN		
227	CLA		
228	ARCL	Y	
229	ARCL	24	
230	*LBL	03	(L03a)*
231	ASTO	X	
232	X=Y?		
233	GTO	13	(G13d)
234	ASHF		
235	ASTO	24	
236	"??"		Split off
237	ARCL	X	ldg hex
238	ASHF		digit
239	ASTO	X	
240	SF	25	
241	XEQ IND	X	Dispatch
242	FS?	25	to get val.
243	GTO	02	(G02a)
244	"IN ERR"		
245	AVIEW		Error
246	GTO	22	return
247	*LBL	13	(L13d)*
248	ARCL	23	Process 2nd
249	FS?C	07	set of chars
250	GTO	03	(G03a)
251	XEQ	92	Done
252	RCL	21	
253	FS?	09	
254	CHS		
255	XEQ	93	
256	GTO	20	
257	*LBL	e	MODE SETTING
258	SF	04	routines
259	CF	03	
260	GTO	14	(G14d)
261	*LBL	d	
262	CF	04	
263	CF	03	
264	GTO	14	(G14d)
265	*LBL	c	
266	CF	04	
267	SF	03	
268	GTO	14	(G14d)
269	*LBL	"EE"	
270	*LBL	E	
271	SF	05	
272	CF	06	
273	GTO	14	(G14d)
274	*LBL	"DD"	
275	*LBL	D	
276	CF	05	
277	SF	06	
278	GTO	14	(G14d)
279	*LBL	"CC"	
280	*LBL	C	
281	CF	05	
282	CF	06	
283	*LBL	14	(L14d)*
284	XEQ	93	
285	GTO	21	
286	*LBL	"FF"	BOOLEAN
287	*LBL	F	PROCESSORS
288	"AND"		
289	XEQ	90	
290	28		AND tbl ptr
291	XEQ	94	
292	GTO	14	(G14e)
293	*LBL	"GG"	
294	*LBL	G	
295	"OR"		
296	XEQ	90	
297	44		OR tbl ptr
298	GTO	04	(G04a)
299	*LBL	"HH"	
300	*LBL	H	
301	"EXOR"		
302	XEQ	90	
303	60		EXOR tbl ptr
304	*LBL	04	(L04a)*
305	XEQ	94	
306	RCL	17	
307	X=0?		
308	RCL	18	
309	INT		
310	RCL	22	
311	*		
312	ST+	21	

Note: Refer to "HP-41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

Step	Key Entry		Comments
313	*LBL	14	(L14e)*
314	RCL	20	Fix stk
315	X<>	19	to display
316	X<>	18	results
317	RCL	21	
318	GTO	21	
319	*LBL "II"		ROTATE PROC.
320	*LBL	I	
321	X<0?		
322	XEQ	95	Make X +ve
323	XEQ	93	
324	RCL	22	Need to
325	X#0?		PROMPT?
326	GTO	14	(G14f)
327	"PLACES?"		
328	CF	22	
329	PROMPT		
330	FC?	22	
331	GTO	21	
332	*LBL	14	(L14f)*
333	"ROT"		
334	SF	25	
335	AVIEW		
336	GTO	"?"	
337	CF	08	
338	X<0?		
339	SF	08	
340	ABS		
341	RCL	25	
342	1		
343	+		
344	MOD		
345	X=0?		
346	GTO	22	
347	1 E3		
348	/		
349	ISG	X	
350	STO	24	Setup rot
351	RCL	17	counter
352	RCL	26	Chk wrd siz
353	X<=Y?		Too small?
354	GTO	23	(yes)
355	STO	T	
356	RDN		Setup stk
357	FS?C	08	ROT left?
358	GTO	14	(G14g)
359	*LBL	05	(L05a)*
360	STO	Y	
361	2		
362	MOD		
363	X#0?		
364	SF	08	
365	CLX		
366	+		
367	FS?C	00	
368	+		
369	2		
370	/		Rot. right
371	INT		
372	FS?C	08	
373	SF	00	
374	ISG	24	
375	GTO	05	(G05a)
376	GTO	21	
377	*LBL	14	(L14g)*
378	X<>Y		
379	CLX		
380	+		
381	*LBL	06	(L06a)*
382	ST+	X	Rot. left
383	X<Y?		
384	GTO	14	(G14h)
385	X<>Y		
386	MOD		
387	SF	08	
388	*LBL	14	(L14h)*
389	FS?C	00	
390	ISG	X	
391	*LBL	00	(null)
392	FS?C	08	
393	SF	00	
394	ISG	24	
395	GTO	06	(G06a)
396	GTO	21	
397	*LBL "JJ"		COMPLEMENT
398	*LBL	J	PROC
399	"COMPL"		
400	XEQ	96	
401	CHS		
402	1		
403	-		
404	X<0?		
405	XEQ	95	
406	FC?	01	
407	ISG	X	(2's comp.)
408	*LBL	00	(filler)
409	GTO	21	
410	*LBL	A	WSIZ
411	XEQ	93	
412	XEQ	91	
413	GTO	22	
414	*LBL "bb"		LEFT
415	*LBL	b	JUSTIFY
416	"LJY"		

Note: Refer to "HP 41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

Step	Key Entry		Comments
417	XEQ	96	
418	RCL	26	
419	X<>Y		
420	X<=0?		
421	GTO	21	
422	*LBL	07	(L07a)*
423	STO	17	
424	ST+	X	
425	X<Y?		
426	GTO	07	(G07a)
427	GTO	22	
428	*LBL "BB"		RIGHT
429	*LBL	B	JUSTIFY
430	"RJY"		
431	XEQ	96	
432	X<0?		
433	XEQ	95	
434	STO	L	
435	X=0?		
436	GTO	21	
437	*LBL	08	(L08a)*
438	LASTX		
439	STO	17	
440	2		
441	/		
442	FRC		
443	X=0?		
444	GTO	08	(G08a)
445	GTO	22	
446	*LBL	a	SIGNED MAG
447	FC?C	02	MODE
448	SF	02	
449	GTO	21	
450	*LBL	90	BOOLEAN
451	XEQ	96	UTILITY
452	X<0?		
453	XEQ	95	Ensures
454	STO	17	args are
455	RCL	18	+ve
456	X<0?		
457	XEQ	95	
458	STO	18	
459	1		
460	STO	22	
461	CF	08	
462	RDN		
463	X<=Y?		Ensures
464	RTN		smallest
465	STO	17	arg is in X
466	X<>Y		
467	STO	18	
468	RTN		
469	*LBL	91	WSIZ utility
470	CLA		
471	ARCL	25	
472	2		
473	┤- ", NEW SIZ:"		
474	CF	22	
475	PROMPT		
476	FC?	22	
477	RCL	25	
478	*LBL	99	(L99a)*
479	X<=0?		
480	GTO	91	
481	STO	25	
482	Y**X		
483	STO	26	
484	RTN		
485	*LBL	92	STK restore
486	RCL	27	
487	STO	L	
488	RCL	20	
489	RCL	19	
490	RCL	18	
491	RCL	17	
492	RTN		
493	*LBL	96	
494	SF	25	
495	AVIEW		
496	GTO	"?"	
497	*LBL	93	STK save
498	STO	27	
499	STO	17	
500	RDN		
501	STO	18	
502	RDN		
503	STO	19	
504	RDN		
505	STO	20	
506	CLX		
507	STO	21	
508	RCL	17	
509	RTN		
510	*LBL	94	BOOLEAN
511	RCL	17	utility
512	4		
513	MOD		
514	ST-	17	
515	RCL	18	
516	X=0?		
517	SF	08	
518	4		
519	ST*	Z	
520	MOD		

Note: Refer to "HP 41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

Step	Key Entry	Comments
521	ST- 18	
522	+	
523	+	
524	RCL IND X	Get 2 bit
525	4	result
526	ST/ 17	
527	ST/ 18	
528	X<> 22	
529	ST* 22	
530	*	
531	ST+ 21	
532	X<> T	
533	FC? 08	
534	GTO 94	
535	RTN	
536	*LBL 95	+VE X
537	X<> 26	utility
538	STO 21	
539	X<> 26	
540	ST+ 21	
541	X<> 21	
542	RTN	
543	*LBL "?Q"	HEX INPUT
544	SF 10	DISPATCHES
545	*LBL "?-"	
546	SF 09	
547	0	
548	RTN	
549	*LBL "? "	
550	SF 10	
551	*LBL "?0"	
552	0	
553	RTN	
554	*LBL "?Z"	
555	SF 10	
556	*LBL "?1"	
557	1	
558	RTN	
559	*LBL "?="	
560	SF 10	
561	*LBL "?2"	
562	2	
563	RTN	
564	*LBL "??"	
565	SF 10	
566	*LBL "?3"	
567	3	
568	RTN	
569	*LBL "?V"	
570	SF 10	
571	*LBL "?4"	
572	4	
573	RTN	
574	*LBL "?W"	
575	SF 10	
576	*LBL "?5"	
577	5	
578	RTN	
579	*LBL "?X"	
580	SF 10	
581	*LBL "?6"	
582	6	
583	RTN	
584	*LBL "?R"	
585	SF 10	
586	*LBL "?7"	
587	7	
588	RTN	
589	*LBL "?S"	
590	SF 10	
591	*LBL "?8"	
592	8	
593	RTN	
594	*LBL "?T"	
595	SF 10	
596	*LBL "?9"	
597	9	
598	RTN	
599	*LBL "?A"	
600	10	
601	RTN	
602	*LBL "?B"	
603	11	
604	RTN	
605	*LBL "?C"	
606	12	
607	RTN	
608	*LBL "?D"	
609	13	
610	RTN	
611	*LBL "?E"	
612	14	
613	RTN	
614	*LBL "?F"	
615	15	
616	RTN	

Note: Refer to "HP 41C OWNER'S HANDBOOK AND PROGRAMMING GUIDE" for specific information on keystrokes. The function index is found at the very back of the handbook.

REGISTERS, STATUS, FLAGS, ASSIGNMENTS

DATA REGISTERS

00	"0"	47	3
01	"1"	48	1
02	"2"	49	1
03	"3"	50	3
04	"4"	51	3
05	"5"	52	2
06	"6"	53	3
07	"7"	54	2
08	"8"	55	3
09	"9"	56	3
10	"A"	57	3
11	"B"	58	3
12	"C"	59	3
13	"D"	60	0
14	"E"	61	1
15	"F"	62	2
16	"10"	63	3
17	Stack save: X	64	1
18	Stack save: Y	65	0
19	Stack save: Z	66	3
20	Stack save: T	67	2
21	Scratch	68	2
22	WSIZ prompt arg.	69	3
23	Scratch	70	0
24	Scratch	71	1
25	# of bits in WSIZ	72	3
26	2**WSIZ	73	2
27	LASTX save	74	1
28	0	75	0
29	0		
30	0		
31	0		
32	0		
33	1		
34	0		
35	1		
36	0		
37	0		
38	2		
39	2		
40	0		
41	1		
42	2		
43	3		
44	0		
45	1		
46	2		
47	3		

STATUS

SIZE 076		TOT. REG. 269		USER MODE	
ENG		FIX 0	SCI	ON X	OFF
DEG	X	RAD	GRAD		

FLAGS

#	INIT S/C	SET INDICATES	CLEAR INDICATES
0	C	ROT carry bit=1	ROT carry bit=0
1	S	COMPL=1's comp.	COMPL=2's comp
2	C	signed mag. ON	signed mag. OFF
3	C	octal input	decimal or hex inp
4	C	hex input	decimal or oct inp
5	S	hex output	decimal or oct out
6	C	decimal output	hex or oct output
7	C	internal scratch	internal scratch
8	C	internal scratch	internal scratch
9	C	internal scratch	internal scratch
10	C	internal scratch	internal scratch
21	C	(never set)	no printer output
22	C	numeric key press	no numeric key p.
23	C	ALPHA key press	no ALPHA key press
27	S	USER mode	normal mode
29	C	Digit grouping ON	Digit grouping OFF

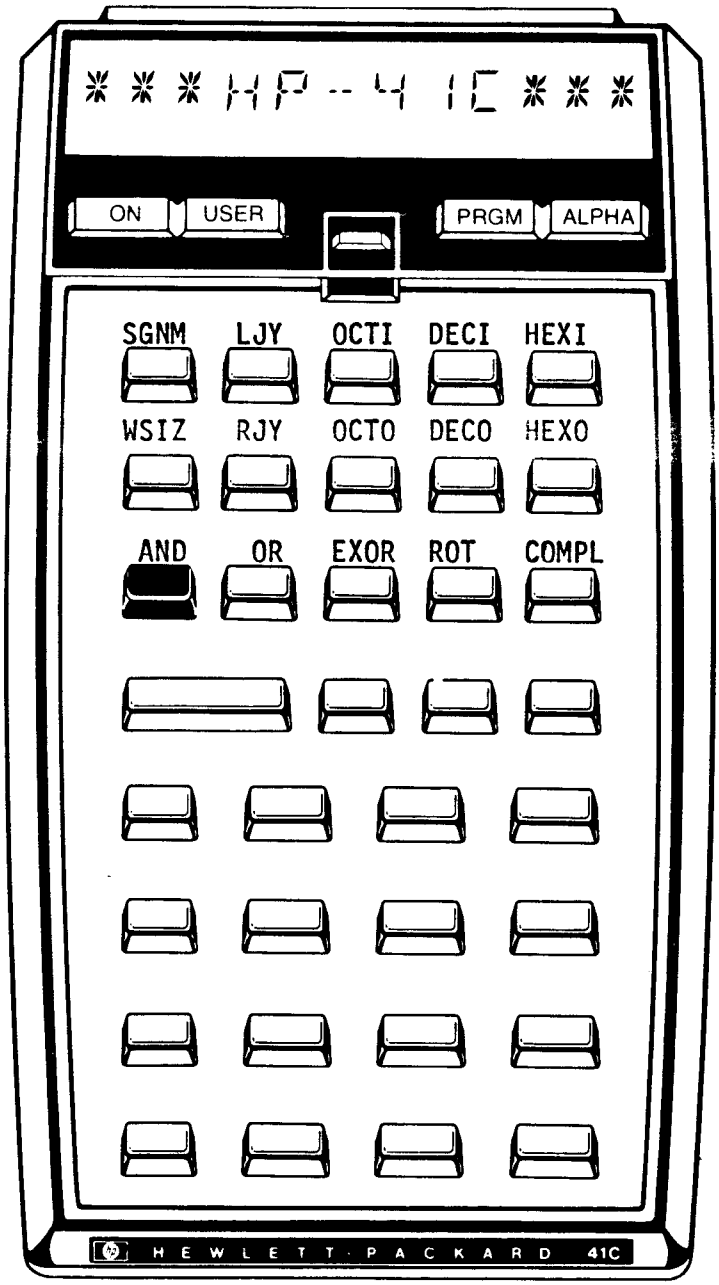
ASSIGNMENTS (LOCAL ALPHA)

FUNCTION	KEY	FUNCTION	KEY
SGNM	GOLD A	HEXO	E
LJY	GOLD B	AND	F
OCTI	GOLD C	OR	G
DECI	GOLD D	EXOR	H
HEXI	GOLD E	ROT	I
WSIZ	A	COMPL	J
RJY	B		
OCTO	C		
DECO	D		

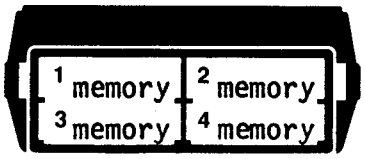
01448C

KEYBOARD CARD LABELING

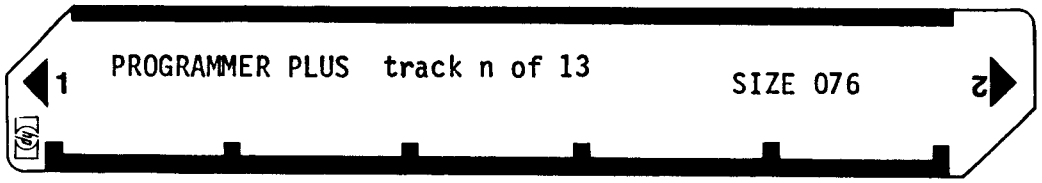
KEYBOARD



SYSTEM CONFIGURATION



CARD



ROW 1 (1 : 2)

ROW 2 (2 : 4)

ROW 3 (4 : 12)

ROW 4 (13 : 19)

ROW 5 (19 : 26)

ROW 6 (26 : 32)

ROW 7 (33 : 39)

ROW 8 (40 : 46)

ROW 9 (46 : 52)

ROW 10 (53 : 59)

ROW 11 (59 : 65)

ROW 12 (66 : 72)

ROW 13 (72 : 78)

ROW 14 (79 : 84)

ROW 15 (85 : 91)

ROW 16 (91 : 97)

ROW 17 (98 : 103)

ROW 18 (103 : 106)

ROW 19 (106 : 113)

ROW 20 (113 : 120)

ROW 21 (121 : 127)

ROW 22 (128 : 132)

ROW 23 (132 : 137)

ROW 24 (137 : 142)

ROW 25 (142 : 149)

ROW 26 (150 : 157)

ROW 27 (158 : 166)

ROW 28 (167 : 173)

ROW 29 (174 : 182)

ROW 30 (183 : 190)

ROW 31 (191 : 198)

ROW 32 (199 : 204)

ROW 33 (204 : 208)

ROW 34 (209 : 216)

ROW 35 (216 : 220)

ROW 36 (221 : 229)

ROW 37 (229 : 236)

ROW 38 (237 : 243)

ROW 39 (244 : 248)

ROW 40 (248 : 254)

ROW 41 (255 : 260)

ROW 42 (260 : 266)

ROW 43 (267 : 271)

ROW 44 (271 : 275)

ROW 45 (276 : 280)

ROW 46 (280 : 286)

ROW 47 (286 : 289)

ROW 48 (289 : 293)

ROW 49 (293 : 298)

ROW 50 (299 : 301)

ROW 51 (302 : 308)

ROW 52 (308 : 316)

ROW 53 (316 : 320)

ROW 54 (320 : 326)

ROW 55 (327 : 330)

ROW 56 (331 : 336)

ROW 57 (336 : 345)

ROW 58 (346 : 351)

ROW 59 (352 : 358)

ROW 60 (359 : 368)

ROW 61 (369 : 376)

ROW 62 (376 : 386)

ROW 63 (387 : 394)

ROW 64 (394 : 398)

ROW 65 (398 : 403)

ROW 66 (404 : 410)

ROW 67 (410 : 414)

ROW 68 (414 : 418)

ROW 69 (418 : 426)

ROW 70 (426 : 430)

ROW 71 (430 : 435)

ROW 72 (436 : 444)

ROW 73 (445 : 450)

ROW 74 (450 : 456)

ROW 75 (457 : 464)

ROW 76 (465 : 473)

ROW 77 (473 : 474)

ROW 78 (475 : 481)

ROW 79 (482 : 489)

ROW 80 (489 : 496)

ROW 81 (496 : 503)

ROW 82 (504 : 511)

ROW 83 (512 : 520)

ROW 84 (521 : 528)

ROW 85 (529 : 535)

ROW 86 (536 : 542)

ROW 87 (543 : 545)

ROW 88 (545 : 550)

ROW 89 (551 : 554)

ROW 90 (554 : 559)

ROW 91 (559 : 562)

ROW 92 (563 : 566)

ROW 93 (566 : 571)

ROW 94 (571 : 574)

ROW 95 (575 : 579)

ROW 96 (579 : 583)

ROW 97 (584 : 586)

ROW 98 (586 : 591)

ROW 99 (591 : 595)

ROW 100 (595 : 599)

ROW 101 (599 : 603)

ROW 102 (604 : 608)

ROW 103 (608 : 612)

ROW 104 (612 : 617)

ROW 105 (617 : 617)