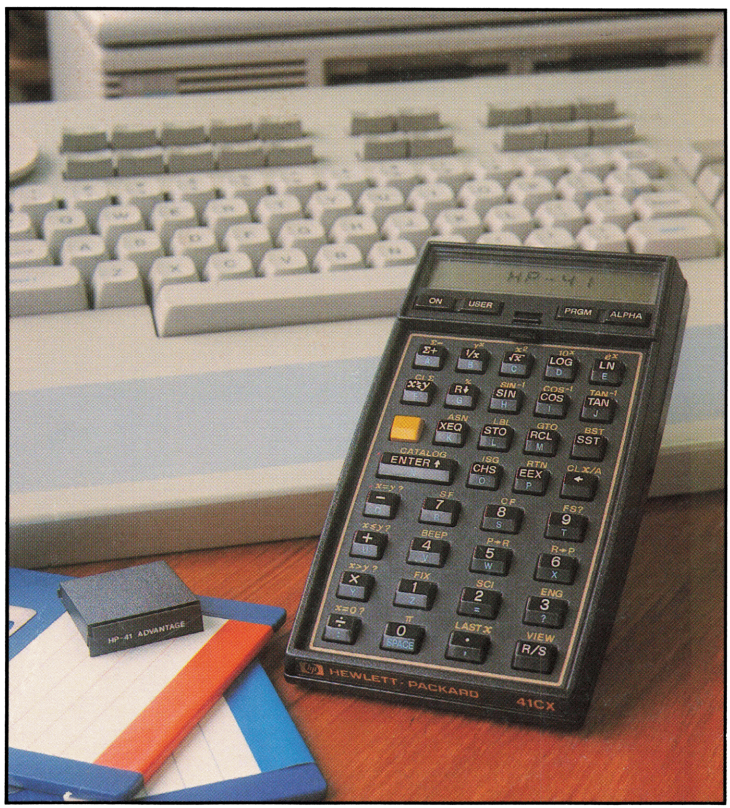
Computer Science On Your HP-41

Using the Advantage Module

By Ed Keefe



A GRAPEVINE PUBLICATION

Computer Science On Your HP-41

Using the Advantage Module

By Ed Keefe

Grapevine Publications, Inc.
P.O. Box 118
Corvallis, OR 97339-0118
U.S.A.

Acknowledgements...

Thanks and congratulations are extended to Hewlett-Packard Company's calculator division for continuing its tradition of producing top-quality products and documentation.

Cover Photo by Tom Brennan

Printed in the United States of America

ISBN 0-931011-10-8

Dedication and Appreciation

This book is dedicated to the patience of Helen, my wife, who trusted that I was doing something worthwhile without ever understanding what that might be.

I would also like to thank Brian Meeker who took the time and care to review this book and who pointed out those parts that might puzzle a beginning computer science student.

Table of Contents

CHAPTER 3.

CHAPTER 4.

CHAPTER 5.

CHAPTER 6.

CHAPTER 7.

CHAPTER 8.

CHAPTER 9.

Introduction

This book is all about a set of functions in your HP-41 Advantage Module and about a program that will turn your HP-41 into a real problem solver in computer science and engineering.

Whether you're a student of computer science or engineering, a professional already out in the "real world," or just working on the ultimate "hack" for your personal computer, you'll find the boolean functions and number base conversion operators of the Advantage Module to be indispensable.

And if you're wondering what else you can do with these Advantage Module functions, I'll show you a program that will make your 41CV behave like Hewlett Packard's "Computer Scientist" calculator, the HP-16C. I call this program the "16-Emulator" or the "16-E" for short.

A Brief Digression About the HP-16C

Hewlett Packard created the HP-16C calculator, "The Computer Scientist," in 1982, in response to the needs of computer scientists and engineers.

These professionals wanted a reliable tool that would give them fast number-base conversions and bit manipulating functions. That is exactly what they got in the HP-16C.

So Why Use An Emulator Program?

If you're an engineer or an engineering student, you probably already have an HP-41CV. This calculator, along with the Advantage Module and the 16-E program will save you the cost of the HP-16C. Surely, you can put that money to better use. You also won't have to remember to pack another piece of equipment every morning as you rush to catch the bus to work or dash off to your 8 A.M. class.

On the other hand, if you are not concerned about saving money and you enjoy keeping track of all your different calculators, the decision becomes more cloudy. You may decide that it's not a matter of "either one or the other" but "both." (HP will surely like that kind of decision.)

So, in case you are having trouble making this momentous decision, let me list the differences between the HP-16C calculator and the HP-41CV running the 16-E program.

ON THE ONE HAND...

1. The HP-16C is a "dedicated" calculator: the HP-41CV is more of a general purpose computer. This means, for instance, that the keys on the HP-16C are appropriately labeled for their functions. To use the 16-E program you will have to interpret the current labels on the HP-41CV's keyboard's or make up a keyboard overlay.

2. The HP-16C operates as a binary calculator. It is fast! On the other hand, the 16-E program uses the HP-41 as a decimal based computer. In order to emulate something as simple as a

shift of binary digits to the left, the 16-E program must go through two number-base conversions and a multiplication by two. The 16-E chugs along while the HP-16C zips along.

3. The HP-16C will handle binary numbers up to 64 bits wide while the 16-E will handle only a maximum of 32 bits at a time.

ON THE OTHER HAND....

4. The 16-E program will let you use alpha prompts in your own programs: the HP-16C doesn't have this capability. This means, for instance, that writing programs with the 16-E program is much easier. The HP-16C uses numeric key codes to show you the steps in a program.

5. When you have finished writing your program on the HP-41CV, you can save it on magnetic cards. This is something you can't do with the HP-16C.

6. Furthermore, if you have a printer, you can get a hardcopy of all your work with the 16-E. This is something else you can't do with the HP-16C.

7. Finally, if you are a professor of engineering or computer science and you have access to the HP-IL and video interfaces for the HP-41CV, then you can plug the HP-41CV into a computer projector. This makes a very dramatic visual aid for classroom presentations—far superior to writing on a chalk board.

8. Above all else, the 16-E program and this manual will let you take full advantage of the logic functions of the Advantage Module.

How To Use This Manual

As you read through this manual, realize that you are learning to use a new tool. The tool may look like your HP-41, but try to think of it as something new. Work through all the problems, or as many of them as you can. You may not understand all the concepts behind the problems. On the other hand, you may begin to uncover the concepts through the use of the 16-E program.

If the 16-E program helps you understand just one obscure idea in computer science, this book will have paid for itself. Truly you will "have the Advantage" in computer science.

Chapter 1.

LOADING THE 16-E PROGRAM

Your Calculator's Configuration

In order to use the 16-E program, you will need...

1. an HP-41CV or an HP-41CX,

2. the HP-41 Advantage module,

3. and this book.

Also, if you have an HP-41 Wand (barcode reader), this will come in *very* handy, so keep it close by. If you don't have a Wand, perhaps you can borrow one from the rich kid down the hall or the president of your company.

When your HP-41 is configured properly, you are ready to load the program.

But before you do that... *Do You Remember How to Load A Program?*

To use this book you will need to be able to follow HP-41 keystroke instructions, load, pack and execute a program.

I assume that you know how to do the following operations:

--how to read keystroke notation, (e.g. [XEQ] [ALPHA] SIZE [ALPHA] 11)

--how the Stack works, including
[+],[−],[X],[/],[X<>Y],[R↓], [STO] and [RCL]

--how to SIZE and PACK your HP-41CV's memory

--how to read and key in program steps such as:

SIGN
DSE X
ADV
ARCL X
ISG X
NOT

--how to move around in—and edit—a program (just in case you mis-key some step)?

If all of the above is familiar to you already, you should have no trouble in following the procedures for loading and running the 16-E program. On the other hand...if the above requirements are new to you or very old and long-forgotten by you, please take some time, now, to come "up to speed" on what you don't know or remember. Here are a couple of options:

--Look in your Owner's Handbook for whatever is on the above list that you don't remember.

--Read a good book: for some reason, the publishers recommend *An Easy Course in Programming the HP-41*, by Chris Coffin and Ted Wadman, available at many Hewlett-Packard dealers or directly from the publisher, by calling Grapevine Publications, Inc. (1-800-338-4331).

Preparing Your HP-41 CV's Memory

The 16-E program will need 184 registers of program memory in your calculator.

On top of that you will need 15 data registers to run the 16-E program and its "spin-off" programs.

So...make room for the program in your calculator.

Once you have made sufficient space for the 16-E program, you can begin.

If you have an HP-41 Wand, plug it in to your HP-41CV, turn to the part of this book that contains the barcode for the 16-E program and read it into your calculator.

If you don't have a Wand, limber up your fingers; you have a lot of keystroking to do. I count 781 separate program instructions. There are also about 37 key assignments to make.

Once you have keyed all the instructions into the HP-41CV, I highly recommend that you use an HP-41 Card Reader (yours, your room-mate's, or a friend's) to make a copy of the program. You will need six (6) magnetic cards to save the 16-E program.

Alternatively, if you don't have access to a Card Reader, but you do have some Extended Memory, you can store the 16-E program there when you are not using it.

Keystroke Time!

As you begin to key in the program, you may notice that many of the instructions in the 16-E program are duplicates of other instructions. For those of you who are adept at writing HP-41 programs, resist the temptation to combine these duplicate steps into subroutines. The duplication is needed to make the 16-E program respond more quickly to your bidding, even though it does make the program longer than is absolutely necessary.

KEYCODE	COMMENTS
01 >LBL "WS"	Initialize flags.
02 FIX 0	Integer format
03 CF 06	
04 SF 27	
05 CF 29	
06 CLA	Clear alpha register.
07 >LBL 13	Prompt for Word Size.
08 CF 05	
09 CF 22	
10 "|-WSIZ="	
11 ARCL 00	
12 PROMPT	
13 CLA	
14 FC?C 22	
15 GTO 13	If no input, prompt again.
16 STO 00	Word size
17 2	
18 X<>Y	
19 Y^X	
20 STO 01	2^word size
21 1	
22 -	
23 STO 02	2^word size -1
24 RCL 01	

```
25  2
26  /
27  STO 04          maximum positive number for this
                    word size
28  1
29  -
30  STO 03
31  RDN
32  GTO 13
33 >LBL "SB"        Set a bit if and only if it is clear.
34  STO 10          Save stack.
35  R^
36  STO 09
37  STO 08
38  R^
39  STO 07
40  R^
41  R^
42  RCL 00          Check to see if bit-to-be-set is in
                    bounds of current word size.
43  X>Y?
44  GTO 00
45  "ERR: "         If out of bounds: error message
46  RDN
47  GTO 13          Branch to WS routine.
48 >LBL 00          Set the bit by adding 2^bit to
                    contents of Y register.
49  RDN
50  BIT?
51  GTO 00
52  2
53  X<>Y
54  Y^X
55  +
56  R^
57 >LBL 00
58  RDN             Rearrange stack before exiting.
59  GTO 15          Exit.
```

```
60 >LBL "CB"
61  STO 10
62  R^
63  STO 09
64  STO 08
65  R^
66  STO 07
67  R^
68  R^
69  RCL 00
70  X>Y?
71  GTO 00
72  "ERR: "
73  RDN
74  GTO 13
75 >LBL 00
76  RDN
77  BIT?
78  GTO 00
79  RDN
80  GTO 15
81 >LBL 00
82  2
83  X<>Y
84  Y^X
85  -
86  GTO 15
87 >LBL "B?"
88  "NO"
89  BIT?
90  "YES"
91  +
92  LASTX
93  -
94  AVIEW
95  RTN
96 >LBL "RLN"
97  CF 07
```

Clear a bit if and only if it is set.

Save stack.

Check if bit-to-be cleared is in bounds.

If out of bounds, generate error message.

Clear bit by subtracting 2^bit from Y-reg.

Exit.

Test if a bit is set or clear.

Rotate Left by N bits.

```
98  X=0?          If N = 0, set flag 07.
99  SF 07
100 STO 05        Store N in R-05 as a counter;
101 STO 10        duplicate in R-10.
102 R^
103 STO 09        Save stack.
104 STO 08
105 R^
106 STO 07
107 R^
108 FS?C 07
109 GTO 15
110 >LBL 07
111 CF 00         Clear carry flag.
112 ST+ X         Multiply number by 2.
113 RCL 02        Is result less than maximum?
114 X>Y?
115 GTO 00        Yes
116 -             No: reduce number to keep it in
                  bounds.
117 SF 00         Set carry flag.
118 R^
119 >LBL 00
120 RDN
121 DSE 05        Decrement counter.
122 GTO 07
123 GTO 15        Exit.
124 >LBL "RRN"    Rotate Right by N.
125 CF 07
126 X=0?          If N = 0, then set flag 07.
127 SF 07
128 STO 05        Store N in R-05 as counter;
129 STO 10        duplicate in R-10
130 R^
131 STO 09        Save stack.
132 STO 08
133 R^
134 STO 07
```

```
135  R^
136  FS?C 07
137  GTO 15
138 >LBL 06          Rotation emulated:
139  RCL X
140  2
141  ST/ Z           Divide number by 2.
142  MOD             Use mod 2 to find if number is odd
143  CF 00           or even.
144  X#0?            If mod 2 is not 0, then set carry
145  SF 00           flag.
146  RDN
147  RND             Round the number.
148  FS? 00          If carry, then decrement number
149  DSE X            by 1.
150 >LBL 00
151  FS? 00          If carry, then add R-04 to number
152  RCL 04          to set MSB.
153  FC? 00
154  .
155  +
156  DSE 05          Decrement counter.
157  GTO 06          Repeat loop.
158  GTO 15
159 >LBL "RLCN"      Rotate Left through Carry Bit by N.
160  CF 07
161  X=0?
162  SF 07
163  STO 05          Set up counter.
164  STO 10
165  R^              Save stack.
166  STO 09
167  STO 08
168  R^
169  STO 07
170  R^
171  FS?C 07
172  GTO 15          Exit loop if N = 0,
```

```
173  CF 05          else set flag 05.
174 >LBL 11         Loop.
175  FS?C 00        If carry bit is 1, then
176  SF 05          set flag 05.
177  ST+ X          Multiply number by 2.
178  RCL 01         If result is less than maximum
                    allowed,
179  X<=Y?
180  SF 00          then set carry flag.
181  MOD            Used to keep number in bounds of
                    word size.
182  FS? 05         If flag 05 is set, add 1 (emulates a
183  1              transfer from the carry bit to the
                    number.)
184  FC?C 05
185  .
186  +
187  DSE 05         Decrement counter and
188  GTO 11         repeat the loop.
189  GTO 15         Exit when done.
190 >LBL "RRCN"     Rotate Right through Carry Bit by
191  CF 07          N.
192  X=0?           If N = 0, then
193  SF 07          set flag 07.
194  STO 05         Store counter.
195  STO 10         Save stack.
196  R^
197  STO 09
198  STO 08
199  R^
200  STO 07
201  R^
202  FS?C 07        If N = 0, then
203  GTO 15         exit routine.
204 >LBL 10         Loop.
205  FS? 00         If carry bit is already set, add R-01
206  RCL 01         to number. (Emulates the setting
207  FC?C 00        of the number's MSB.)
```

```
208  .
209  +
210  RCL X          Emulate rotation by division by 2.
211  2
212  ST/ Z
213  MOD            Used to determine if number is
214  CF 00          odd or even.
215  X#0?
216  SF 00
217  RDN
218  RND
219  FS? 00         If carry bit is set, reduce number
220  DSE X          by 1 to emulate a rotation into the
221 >LBL 00         carry bit.
222  DSE 05         Decrement counter and
223  GTO 10         repeat the loop.
224  GTO 15         Exit from routine.
225 >LBL "MSKR"     Mask on Right of Word Size.
226  STO 10         Save stack.
227  R^
228  STO 09
229  R^
230  STO 08
231  R^
232  STO 07
233  R^
234  2
235  X<>Y
236  Y^X
237  1
238  -              Mask = 2^N-1
239  RCL 02         Used to keep the mask in bounds
240  AND            of current word size.
241  GTO 15         Exit from routine.
242 >LBL "MSKL"     Mask on Left.
243  STO 10         Save stack.
244  R^
245  STO 09
```

```
246 R^
247 STO 08
248 R^
249 STO 07
250 R^
251 RCL 00      Used to find the complement of
252 X<>Y        the number in the current word
253 -           size.
254 2           Mask = 2^N-1
255 X<>Y
256 Y^X
257 1
258 -
259 NOT         Complement of number in word
                size of 32 bits.
260 RCL 02      Keep number in bounds
261 AND         of current word size.
262 GTO 15      Exit from routine.
263 >LBL "LJY"  Left Justify a number.
264 X=0?        If N = 0,
265 GTO15       exit,
266 STO 10      else
267 R^          save stack.
268 STO 09
269 R^
270 STO 08
271 R^
272 STO 07
273 R^
274 >LBL 02     Loop.
275 RCL 00      Current word size
276 DSE X       less 1.
277 BIT?        Is MSB of the number set?
278 GTO 01      If so, branch to...
279 RDN         else
280 ST+ X       shift left by 1.
281 GTO 02
282 >LBL 01
```

```
283  RDN          Rearrange stack and
284  GTO 15       Exit from routine.
285 >LBL "RJY"    Right Justify a Number.
286  X=0?         If N = 0, then
287  GTO15        exit from routine,
288  STO 10       else
289  R^           save stack.
290  STO 09
291  R^
292  STO 08
293  R^
294  STO 07
295  R^
296 >LBL 03       Loop.
297  0            Is LSB set?
298  BIT?
299  GTO 01       Yes: exit loop.
300  RDN          No:
301  1            Rotate right by 1.
302  ROTXY
303  GTO 03       Repeat loop.
304 >LBL 01
305  RDN          Rearrange stack and
306  GTO 15       exit from routine.
307 >LBL "SL"     Logical Shift Left.
308  STO 10       Save Stack.
309  R^
310  STO 09
311  R^
312  STO 08
313  R^
314  STO 07
315  R^
316  ST+ X        Shift emulation:  multiply by 2.
317  RCL 00
318  CF 00
319  SF 25        Set error ignore flag.
320  BIT?         Is MSB set?
```

```
321  SF 00          Yes: set carry bit.
322  CF 25          Clear error ignore flag.
323  RDN
324  GTO 15         Exit from routine.
325 >LBL "SR"       Logical Shift Right.
326  R^             Save stack.
327  STO 09
328  R^
329  STO 08
330  R^
331  STO 07
332  R^
333  RCL 02
334  AND
335  STO 10
336  CF 00          Clear carry flag.
337  0              Is LSB set?
338  BIT?
339  SF 00          Yes:  set carry bit.
340  RDN
341  1              Rotate number to right by 1.
342  ROTXY
343  2
344  ENTER^
345  31
346  Y^X
347  -              Keep number in bounds.
348  GTO 15         Exit from routine.
349 >LBL "ASR"      Arithmetic Shift Right.
350  R^             Save stack.
351  STO 09
352  R^
353  STO 08
354  R^
355  STO 07
356  R^
357  RCL 02
358  AND
```

```
359  STO 10          New lastx.
360  CF 00           Clear carry flag.
361  0
362  BIT?            If LSB is set, then
363  SF 00           set carry flag.
364  RDN
365  CF 05
366  RCL 00          Word size
367  DSE X           less one--
368  BIT?            Is MSB set?
369  SF 05           Yes:  set flag 05.
370  RDN
371  1               Rotate number right by 1.
372  ROTXY
373  2
374  ENTER^
375  31
376  Y^X
377  -               Reduce number to keep it in
                     bounds.
378  FC?C 05         If flag 05 is clear,
379  GTO 15          then exit from routine,
380  RCL 04          else add contents of R-04 to
381  +               keep the sign bit the same as
                     before.
382  GTO 15          Exit from routine.
383 >LBL "CHS"       Change sign of number.
384  STO 10          Save stack.
385  R^
386  STO 09
387  R^
388  STO 08
389  R^
390  STO 07
391  R^
392  CF 04           Clear out of range flag.
393 >LBL 17          Enter routine here if stack has
394  X=0?            been saved.
```

```
395  GTO 00
396  FS? 01        If 1's complement,
397  GTO 19        then branch to LBL 19.
398  FC? 02        If not 2's complement,
399  GTO 01        then...
400  RCL 04
401  X=Y?          If same as R-04, then
402  SF 04         set out of range flag.
403  RDN
404 >LBL 01
405  DSE X         Reduce number by 1.
406  STO X         NOP
407  FS? 03        Unsigned format?
408  SF 04         Set flag 04: CHS has no meaning in
                   UNS format.
409  GTO 19        Branch to LBL 19.
410 >LBL 00
411  FC? 01        Not 1's complement?
412  GTO 15        then exit from routine.
413 >LBL "NOT"
414  STO 10        Save stack.
415  R^
416  STO 09
417  R^
418  STO 08
419  R^
420  STO 07
421  R^
422 >LBL 19        Entry point for other routines.
423  NOT           Complement the number in word
                   size 32.
424  RCL 02        A mask that is used with
425  AND           AND to bring complement in
                   bounds.
426  GTO 15        Exit from routine.
427 >LBL "+"       Add two numbers.
428  STO 10        Save stack.
429  R^
```

```
430  STO 09
431  STO 08
432  R^
433  STO 07
434  R^
435  R^
436  RCL Y      Duplicate X and Y in Z and T.
437  RCL Y
438  +          Add numbers.
439  STO 05     Set aside in R-05.
440  RCL 01     e.g. 256 for wsiz=8.
441  CF 04      Clear out of range flag.
442  CF 00      Clear carry flag.
443  X<=Y?      If number is greater than 256,
444  SF 00      then set carry flag.
445  RDN
446  FC? 03     Not UNS format?
447  GTO 00     Yes
448  FS? 00     Carry flag set?
449  SF 04      Yes
450  GTO 15     Exit from routine.
451 >LBL 00     1's and 2's complement format...
452  RDN
453  RCL 03     Used to determine if second
454  X<Y?       add end is greater than or equal to
455  GTO 01     maximum positive value for
456  RCL Z      current word size.
457  X>Y?       If first addend is less than R-03,
458  GTO 00     then...
459  RDN        else
460  RCL 05     Get sum: is this greater than R-03?
461  X>Y?
462  SF 04      Yes: set out of range flag.
463  GTO 00
464 >LBL 01
465  RCL Z
466  X<=Y?
467  GTO 00
```

```
468 RDN
469 RCL 02       Max. number in current word size.
470 +
471 RCL 05
472 X<=Y?        If greater than or equal to sum of
                 numbers,
473 SF 04        then set flag 04.
474 >LBL 00
475 RCL 05
476 FC? 01       Not 1's complement?
477 GTO 00       Yes
478 FS? 00       Carry bit set?
479 ISG X        Yes:  increment number by 1.
480 >LBL 00
481 GT0 15       Exit from routine.
482 >LBL "-"     Difference of two numbers.
483 STO 10       Save stack.
484 R^
485 STO 09
486 STO 08
487 R^
488 STO 07
489 R^
490 R^
491 CF 04        Clear out of range flag.
492 CF 00        Clear carry flag.
493 X>Y?         If first number is less than second
                 number,
494 SF 00        then set carry flag.
495 RCL Y        Duplicate numbers in stack.
496 RCL Y
497 -            Take difference.
498 FS? 00       If carry is set, then
499 RCL 01
500 FC? 00
501 .
502 +            add R-01 to difference, else add 0.
503 FC? 03       Not UNS format?
```

```
504  GTO 00        Yes.
505  FS? 00        Carry bit set?
506  SF 04         Yes:  set out of range flag,
507  GTO 15        and exit from routine.
508 >LBL 00        Otherwise...
509  STO 05        Store difference in R-05.
510  RDN
511  RCL 03        If the second number is less than
512  X<Y?          or equal to contents of R-03, then
513  GTO 01
514  R^            else
515  X<=Y?         If difference is less than or equal
                   to R-03,
516  GTO 00
517  RDN
518  RCL 05
519  X<=Y?
520  SF 04         set out of range flag.
521  GTO 00
522 >LBL 01
523  R^            If the first number is greater than
                   R-03,
524  X>Y?
525  GTO 00        then...
526  RDN           else
527  1             increase difference by 1.
528  +
529  RCL 05
530  X>Y?          If difference is less than R-05,
531  SF 04         then set out of range flag.
532 >LBL 00
533  RCL 05
534  FC? 01        Not 1's complement?
535  GTO 00        Yes
536  FS? 00        Carry flag set?
537  DSE X         Decrease difference by 1.
538 >LBL 00
539  GTO 15        Exit from routine. (Whew!)
```

```
540 >LBL "*"      Multiply two numbers.
541  STO 10       Save stack.
542  R^
543  STO 09
544  STO 08
545  R^
546  STO 07
547  R^
548  R^
549  FC? 03       Not UNS format? (ie. 1's or 2's
                  complement)
550  XEQ 09       Yes...
551  *            Multiply.
552  FC? 03       Not UNS format?
553  RCL 03
554  FS? 03       UNS format?
555  RCL 02       R-02
556  CF 04        Clear out of range flag.
557  X<Y?         If R-02 is less than R-02 or R-03,
558  SF 04        then set out of range flag.
559  AND          Use mask with AND to keep
                  product in bounds.
560  FC?C 05      If flag 05 is clear,
561  GTO 15       then exit routine,
562  GTO 17       else
563 >LBL 09       This routine is used by both "*"
564  RCL 00       and "/" to determine if numbers
565  DSE X        are positive or negative and
566  CF 05        determines sign of result.
                  R-01 less one is used to find if
                  MSB is set.
567  BIT?         If MSB is set then...
568  SF 05        set flag 05.
569  FC? 05       If flag 05 clear, then
570  GTO 00       branch...
571  X<>Y         else swap numbers,
572  RCL 02       and
573  -            take the complement of the
```

```
574  ABS          number.
575  FS? 02       In 2's complement format,
576  ISG X        increase number by 1.
577  STO X        NOP
578  X<>Y         Swap numbers.
579 >LBL 00       Do similarly for other number...
580  RCL Z
581  X<>Y
582  CF 04
583  BIT?
584  SF 04
585  FC? 04
586  GTO 00
587  X<>Y
588  RCL 02
589  -
590  ABS
591  FS? 02
592  ISG X
593  STO X
594  X<>Y
595 >LBL 00
596  RDN
597  X<>Y
598  FS?C 04      Exclusive OR the two flags 04 and
                  05
599  FS?C 05      If either flag 04 or 05 is set BUT
600  FS?C 05      not both, then...
601  SF 05        set flag 05.
602  RTN          Return to calling program.
603 >LBL "/"      Divide two numbers...
604  STO 10       Save stack.
605  R^
606  STO 09
607  STO 08
608  R^
609  STO 07
610  R^
```

```
611 R^
612 FC? 02          Not 2's complement?
613 GTO 00          Yes:
614 X<>Y            No:
615 STO 05          Set aside dividend.
616 X<>Y
617 /               Divide.
618 RCL 04          R-04
619 RCL 02          R-02
620 /               Divide.
621 -               Subtract.
622 X=0?            This is a special case in 2's
623 SF 04           compliment where the maximum
624 X=0?            number in current word size is
625 GTO 01          divided by -1.  In this case, set out
                    of range flag.
626 RCL 05          Recall dividend.
627 RCL 10          Divisor.
628 >LBL 00
629 FC? 03          Either 1's or 2's complement.
630 XEQ 09          Check on signs of numbers.
631 /               Divide.
632 INT             Take only integer part.
633 LASTX
634 FRC
635 X#0?            If there is a remainder, then
636 SF 00           set carry flag.
637 RDN
638 FC? 03          If 1's or 2's complement format,
639 RCL 03          then use R-03.
640 FS? 03          If UNS format, then
641 RCL 02          use R-02
642 AND             as a mask to keep quotient in
643 >LBL 01         bounds.
644 FC?C 05         If flag 05 is clear, then
645 GTO 15          exit from routine,
646 GTO 17          else change sign of answer.
647 >LBL "RMD"      Find remainder two numbers.
```

```
648 STO 10
649 R^
650 STO 09
651 STO 08
652 R^
653 STO 07
654 R^
655 R^
656 FC? 02
657 GTO 00
658 X<>Y
659 STO 05
660 X<>Y
661 MOD
662 RCL 04
663 RCL 02
664 MOD
665 -
666 X=0?
667 SF 04
668 X=0?
669 GTO 01
670 RCL 05
671 RCL 10
672 >LBL 00
673 FC? 03
674 XEQ 09
675 MOD
676 INT
677 LASTX
678 FRC
679 X#0?
680 SF 00
681 RDN
682 FC? 03
683 RCL 03
684 FS? 03
685 RCL 02
```

Save stack.

Rest of routine is similar to LBL"/" except that MOD is used in place of /.

```
686 AND
687 >LBL 01
688  FC?C 05
689  GTO 15
690  GTO 17
691 >LBL "UNS"      Set UNSigned display format.
692  SF 03
693  CF 01
694  CF 02
695  GTO 16
696 >LBL "2c"       Set 2's complement format.
697  SF 02
698  CF 01
699  CF 03
700  GTO 16
701 >LBL "1c"       Set 1's complement format.
702  SF 01
703  CF 02
704  CF 03
705  LBL 16
706 >LBL "DECV"     Display decimal number in
                    complement format within the
707  X<>L           bounds of current word size.
708  STO 10         Save stack.
709  R^
710  STO 09
711  R^
712  STO 08
713  R^
714  STO 07
715  LASTX
716 >LBL 15         Entry point for other routines
                    which have already saved the stack.
717  X=0?           If the number is 0 or positive, then
718  GTO 00
719  X>0?
720  GTO 00         Branch.
721  NOT            Complement the number...
```

```
722  RCL 02      Use R-02 as a mask
723  AND         with AND.
724  FS? 02      If 2's complement format, then
725  ISG X       increment number by 1.
726 >LBL 00
727  RCL 01      Use R-01 and
728  MOD         Mod function to keep number in
                 bounds.
729  RCL 00      R-01
730  DSE X       less one.
731  BIT?        Is the MSB set?
732  GTO 00      Yes:
733  RDN         No:
734  GTO 01
735 >LBL 00      Routine uses the alpha register to
736  RDN         display a positive number.
737  FC? 03
738  GTO 00
739 >LBL 01
740  " "
741  ARCL X
742  STO 06
743  RCL 10      Restore stack.
744  SIGN        Store in LastX.
745  RCL 09
746  RCL 08
747  RCL 07
748  RCL 06
749  FC? 06      If flag 06 is clear, show result.
750  AVIEW
751  ADV
752  RTN
753 >LBL 00      Routine is used to show negative
754  ENTER^      number in alpha register.
755  NOT
756  RCL 02
757  AND
758  FS? 02
```

```
759  ISG X
760  STO X
761  "-"
762  ARCL X
763  RDN
764  STO 06
765  RCL 10          Restore stack.
766  SIGN
767  RCL 09
768  RCL 08
769  RCL 07
770  RCL 06
771  FC? 06          If flag 06 is clear, then
772  AVIEW           show number.
773  ADV
774  RTN
775  >LBL "BVU"      Routine needed to correct bug in
                     Advantage ROM.
776  SF 25           This routine will allow for viewing
777  SF 15           the binary numbers on a TV using
778  BINVIEW         HP-IL interface.
779  CF 15           Flag 25 is used to get around the
780  CF 25           normal Out of Range error
781  ADV             message.  Numbers greater than
                     1023 are shown as unsigned
                     decimal numbers.
782  END             You made it!
```

Key Assignments

While you still have your fingers limbered up, you should key in the following assignments of functions to the various keys. Notice that even though these key assignments are optional, life with the 16-E will be much more cordial if you use the suggested key assignments.

1) [shift] [XEQ] [ALPHA] MSKR [ALPHA] [A]
(Assign "MSKR" to key 11.)

2) [shift] [XEQ] [ALPHA] MSKL [ALPHA] [shift] [A]
(Assign "MSKL" to key -11.)

3) [shift] [XEQ] BVU [ALPHA] [ALPHA] [B]
(Assign "BVU" to key 12.)

4) [shift] [XEQ] [ALPHA] BININ [ALPHA] [shift] [B]
(Assign "BININ" to key -12.)

5) [shift] [XEQ] [ALPHA] OCTVIEW [ALPHA] [C]
(Assign "OCTVIEW" to key 13.)

6) [shift] [XEQ] [ALPHA] OCTIN [ALPHA] [shift] [C]
(Assign "OCTIN" to key -13.)

7) [shift] [XEQ] [ALPHA] DECV [ALPHA] [D]
(Assign "DECV" to key 14.)

8) [shift] [XEQ] [ALPHA] WS [ALPHA] [shift] [D]
(Assign "WS" to key -14.)

9) [shift] [XEQ] [ALPHA] HEXVIEW [ALPHA] [E]
(Assign "HEXVIEW" to key 15.)

10) [shift] [XEQ] [ALPHA] HEXIN [ALPHA] [shift] [E]
(Assign "HEXIN" to key -15.)

11) [shift] [XEQ] [ALPHA] X [shift] [I] [shift] [J] Y [ALPHA] [F] (assign X<>Y to its own key, 21, for speedier response.)

12) [shift] [XEQ] [ALPHA] LJY [ALPHA] [shift] [F] (Assign LJY to key -21.)

13) [shift] [XEQ] [ALPHA] RDN [ALPHA] [G] (assign R↓ to its own key, 22, for speedier response.)

14) [shift] [XEQ] [ALPHA] RJY [ALPHA] [shift] [G] (Assign "RJY" to key -22.)

15) [shift] [XEQ] [ALPHA] SR [ALPHA] [H] (Assign "SR" to key 23.)

16) [shift] [XEQ] [ALPHA] SL [ALPHA] [shift] [H] (Assign "SL" to key -23.)

17) [shift] [XEQ] [ALPHA] RLN [ALPHA] [I] (Assign "RLN" to key 24.)

18) [shift] [XEQ] [ALPHA] RLCN [ALPHA] [shift] [I] (Assign "RLCN" to key -24.)

19) [shift] [XEQ] [ALPHA] RRN [ALPHA] [J] (Assign "RRN" to key 25.)

20) [shift] [XEQ] [ALPHA] RRCN [ALPHA] [shift] [J] (Assign "RRCN" to key -25.)

21) [shift] [XEQ] [ALPHA] ASR [ALPHA] [shift] [K] (Assign "ASR" to key -32.)

22) [shift] [XEQ] [ALPHA] CHS [ALPHA] [O] (Assign "CHS" to key 42.)

23) [shift] [XEQ] [ALPHA] [shift] [-] [ALPHA] [Q]
(Assign "-" to key 51.)

24) [shift] [XEQ] [ALPHA] N [shift] 0 T [ALPHA] [shift] [Q]
(Assign "N0T" to key -51.) NOTE: *THE MIDDLE CHARACTER IS ZERO RATHER THAN THE LETTER "O."*

25) [shift] [XEQ] [ALPHA] [shift] [+] [ALPHA] [U]
(Assign "+" to key 61.)

26) [shift] [XEQ] [ALPHA] OR [ALPHA] [shift] [U]
(Assign "OR" to key -61.)

27) [shift] [XEQ] [ALPHA] [shift] [*] [ALPHA] [Y]
(Assign "X" (MULT) to key 71.)

28) [shift] [XEQ] [ALPHA] AND [ALPHA] [shift] [Y]
(Assign "AND" to key -71.)

29) [shift] [XEQ] [ALPHA] [shift] [/] [ALPHA] [:]
(Assign "/" (DIVIDE) to key 81.)

30) [shift] [XEQ] [ALPHA] XOR [ALPHA] [shift] [:]
(Assign "XOR" to key -81.)

31) [shift] [XEQ] [ALPHA] SB [ALPHA] [shift] [V]
(Assign "SB" to key -62.)

32) [shift] [XEQ] [ALPHA] CB [ALPHA] [shift] [W]
(Assign "CB" to key -63.)

33) [shift] [XEQ] [ALPHA] B? [ALPHA] [shift] [X]
(Assign "B?" to key -64.)

34) [shift] [XEQ] [ALPHA] [shift] 1 [shift] C [ALPHA] [shift] [Z] (Assign "1c" to key -72.)

35) [shift] [XEQ] [ALPHA] [shift] 2 [shift] C [ALPHA] [shift] [=] (Assign "2c" to key –73.)

36) [shift] [XEQ] [ALPHA] UNS [ALPHA] [shift] [?] (Assign "UNS" to key –74.)

37) [shift] [XEQ] [ALPHA] RMD [ALPHA] [shift] [space] (Assign "RMD" to key –82.)

Now, that was a lot of keystroking, but it will pay off in easier use of the 16-E program. (How are your fingers?)

Chapter 2.

TESTING AND DEBUGGING THE 16-E PROGRAM

After you have loaded the program, press the [shift][GTO][.][.] keys to pack the program and make sure there is an END to it. The 16-E program is a lengthy program, so it will take a long time to pack—almost 30 seconds. I thought you should know that just in case you should wonder if your HP-41 was "stuck".

Here is a quick procedure that will let you discover if your keystroking is 100% accurate. Read the next TWO paragraphs *before* pressing any keys. (Otherwise, You will be watching the LCD display and won't have time to read the second paragraph.)

Press [shift][ENTER] 1 (CAT 1). The HP-41 will display 27 separate labels used in the 16-E program (preceded by the labels for any other programs that you have in your calculator.)

Stop the CATALOG run ([R/S]) when you see LBL DECV in the display and then press the [SST] key until you see:

END 1283

If the number in the display is 1283, you win! If it is not 1283, then you probably dropped a keystroke or two.

Now is the time to go to the beginning of the program and check it step by step against the listing in this book. The only differences that you should note are the use of (" ") the double quote marks in the printed listing and the super-script T in LBL's and alpha prompts.

Testing the 16-E

Once you have debugged the program, run the following example to convince yourself that it is indeed operating correctly.

If you PACK the program by pressing the [GTO][.][.] keys, expect that the program will respond slowly at first. The speed will pick up after you have performed the following test example.

Note that throughout this book I will represent the gold [shift] key with the "#" symbol, and the [ALPHA] switch by the double quote mark (").

Here is the extended test example. The left column contains the keystrokes. (I am assuming that you have made the suggested key assignments. If you have not done so, then use the long version of the keystrokes as given in the right hand column.)

The center column shows what you may expect to see in the display of your 41CV as a result of the keystrokes.

The last column will give a brief explanation of what has happened.

If you want to race ahead, you may skip this upcoming trial run of the 16-E program. However, I heartily recommend it as a way to get used to the various key assignments. The test run will also serve to "limber up" the program.

KEYSTROKE	DISPLAY	FUNCTION AND COMMENT
[#][D]	WSIZ = 32	XEQ"WS" to set appropriate number of bits for the word size. The number you see may be other than 32.
16 [R/S]	WSIZ = 16	
[R/S]	WSIZ = 16	R/S will not let you accidentally execute another part of the program.
[R/S]	WSIZ = 16	
[#][3]	65535	XEQ "UNS" to establish "unsigned" display format. Flag 3 should be visible in the display.
[#][E]	_ H	XEQ "HEXIN" to prepare for hexadecimal input.
ABCDE	ABCDE_ H	
[Enter]	703710	The decimal equivalent of ABCDE hex for a 32 bit word.

[D]	48350	XEQ "DECV" show the decimal representation of the number in its 16 bit, truncated version.
[E]	BCDE_ H	XEQ "HEXVIEW" which shows that the original number has been truncated to include only the least significant 16 bits of the original number.
[#][2]	-17186	XEQ "2c" to establish the "two's complement" display format. Flag 2 should be visible in the display. Flag 3 should be extinguished.
[Clx]	48350	Clear the display to show the internal representation of the number in the X-register.
[C]	136336 O	XEQ "OCTVIEW" to show the octal representation of the number.
[B]	48350	XEQ "BVU" Modified BINVIEW function. When the number is greater than 1023, the BVU operator defaults to the internal rep-resentation of the

		number rather than give an out of range error.
		To see the full binary representation of this 16 bit number, do the following:
[STO][.][L]		Store the number in the LastX register.
[#][E]	_ H	XEQ "HEXIN"
BC	BC_ H	Key in the leading 8 bits of the number. (Each hex character equals four binary digits.)
[Enter]	188	
[B]	10111100 B	
[#][E]	_ H	
DE	DE_ H	Key in the last 8 bits of the original number.
[Enter]	222	
[B]	11011110 B	Thus the total 16 bits would be: 1011 1100 1101 1110
[LastX]	48350	Restore the original 16 bit number.
[H]	24175	XEQ "SR" to shift the number one bit to the right. NOTE: this is equivalent to dividing the number by two. The binary equivalent would be 0101 1110 0110 1111 or 5E6F Hex.

[#][H]	-17186	XEQ "SL" to shift the number to the left. The number is automatically displayed in 2's complement format.
[CHS]	17186	XEQ "CHS" Note this is both the visible and internal representation of the number. You may press [Clx] once and apparently nothing will happen. Actually it has. You have cleared the display register and you are now looking at the X-register.
[CHS]	-17186	Change the sign again.
[#][K]	-8593	XEQ "ASR" (Arithmetic Shift Right). This shift preserves the sign of the original number.
2 [J]	-2149	XEQ "RRN" Rotate Right by N (in this case, by 2).
2 [I]	-8593	XEQ "RLN" Rotate Left by N. The reciprocal of the previous operation. Flag 0 will appear in the display, indicating that the "carry bit" is set.
CF 00	56943	Clear the carry flag and see the internal representation of -8593.

3 [#][J]	-9267	XEQ "RRCN" Rotate Right Through the Carry bit by N. Again, the carry flag (0) will be visible.
3 [#][I]	-8593	XEQ "RLCN" Rotate Left Through the Carry bit by N (in this case, by 3). The carry flag should be extinguished.
[#][-]	8592	XEQ "NOT" : take the complement of the number.
[E]	2190 H	HEXVIEW the number.
8 [A]	255	XEQ "MSKR" : Create an 8 bit "mask" on the right side of the 16 bit field.
[B]	11111111 B	This is what the right side of the mask looks like. The other half (on the left) is 8 "invisible" zeroes.
[#][*]	144	XEQ "AND" This will "and" all of the "bits" in the Y-register with those in the X-register.
[E]	90 H	Those bits that have been "anded" with 0 are eliminated.

8 [#][A]	-256	XEQ "MSKL" to create a mask of 1's on the left side of the 16 bit word.
[E]	FF00 H	There they are: each F is 1111, and each 0 Hex is 0000 Binary.
[#][+]	65424	XEQ "OR" to "or" this mask with the number in the Y-register. This is the internal representation. To see the 2's complement version...
[D]	-112	
[E]	FF90 H	The hex version of the same number.
[#][1]	-111	XEQ "1c" to see the one's complement format of this same number.
2 [CHS]	-2	A negative two
[*]	222	XEQ "*" to multiply the two negative numbers.
[#][G]	111	XEQ "RJY" to right justify the binary digits in a 16 bit field. This effectively strips off trailing zeroes.
[#][X<>Y]	-8703	XEQ "LJY" to left justify bits in a field of 16 bits.

[RCL] 04	32768	Just a number to use.
[–]	24064	XEQ "–" to subtract this number from -8703.
[LastX]	32768	Recall the value in Last-X
[+]	–8703	XEQ "+" to add the number to the value in the Y-register.
15 [#][6]	YES	XEQ "B?" to test if bit number 15 is set. It should be since bit #15 is the sign bit in one's complement.
[LastX]	15	Recall the bit number.
[#][5]	24064	XEQ "CB" to clear the bit and show the resulting value in one's complement format.
[LastX]	15	
[#][4]	–8703	XEQ "SB" to reset the sign bit and restore the number to its previous value.
20 [#][4]	ERR: WSIZ= 16	Trying to set bit #20 in a number with only 16 bits will generate this reminder.

[Clx] [RDN]	56832	Clear the display and roll the stack down to regain the number.
5 [/]	-1740	XEQ "/" to divide the number by 5. This is integer division, but, in this case the 0 flag is used to show that there is a remainder.
8703[CHS]	-8703	Reset the original number.
5 [#][0]	-3	XEQ"RMD", that is press [shift] [zero], to discover the remainder when the number is divided by 5.

If you have performed all of the above keystrokes and obtained results identical to those above, then your 16-E program is working "as advertised."

In the ensuing chapters I will explain more of the operation of the 16-E program. I will also describe the significance of all of the different display formats.

Chapter 3.

A TUTORIAL ON THE ADVANTAGE MODULE'S COMPUTER SCIENCE FUNCTIONS

I know you are eager to start using the 16-E program, but before showing you all the in's and out's of the program, let me show you some tips and techniques for the Advantage Module's computer science functions. These functions are the backbone of the 16-E program. They are also usable apart from the 16-E program. Discussing them now will save us time and prevent unnecessary confusion when we delve into the depths of the 16-E program.

If you are thoroughly familiar with the number-base conversion functions in the Advantage Module, you may jump to the next part of this book.

However, here is what you will miss:

1. How to do number-base conversions.
2. How to put commas in the output displays.
3. How to avoid the BININ bug.

Number Base Conversion Functions

The Advantage Module will let you convert numbers between base 10 (decimal), 2 (binary), 8 (octal) or 16 (hexadecimal) quickly. This should help you make some sense of those hex or octal dumps from your computer.

Here are some decimal numbers that you can convert to other display formats. You will be using the Advantage Module's "VIEW" functions which you have assigned to the keys in the top row of your HP-41.

1. Input 123 dec and convert this number to hex, octal and binary format.

KEYSTROKES	DISPLAY	COMMENTS
[USER]		Turn on the User Mode.
123	123	Decimal input
[E]	7B H	
[C]	173 O	
[B]	1111011 B	

2. Now convert the decimal number, 4567, to the other three display formats.

KEYSTROKES	DISPLAY	COMMENTS
4567	4567	
[E]	11D7 H	
[C]	10727 O	
[B]	4567	Say what? (Read on...)

Notice that there are some built-in limitations to these "VIEW" functions. First of all, the BINVIEW function, will translate only those numbers less than 1024. Executing BINVIEW with a number greater than 1023 in the X-Register will result in an OUT OF RANGE error message. I have modified the BINVIEW function so that you will not get this message. Instead, when you press the [B] key, you are activating this modified BINVIEW function. The new function, labelled "BVU" will check to see if the decimal number is greater than 1023. If it is, the BVU function won't even try to convert it to binary. It will just leave it as it is. That's why, apparently, nothing happened when you pressed [B] in the above example.

Similarly, the largest octal number that the display can show is ten 7's (7777777777 O) or 1073741823 decimal.

The only format that can display all 32 bits of the internal word is the hexadecimal format.

Eight F's (FFFFFFFF H) are equivalent to 32 1's (binary) or 4294967295 (decimal). This is the largest integer number that any of the Advantage Module's functions can handle. (See Note 1)

I trust that these functions operate fast enough to whet your appetite for more. If you have ever had to do such number conversions by hand, you can really appreciate the speed of the Advantage Module's number base conversions.

Breaking Up Output With Commas

The "VIEW" functions of the Advantage Module will even let you intersperse commas in the binary, hexadecimal or octal display.

To see how this feature works, set flags 28 and 29 from the keyboard. Then switch out of USER mode; set FIX 2 display format, key in 16777215, and press the [USER] key to turn on the USER annunciator in the display.

Now press the [E] key, and you will see FF,FF,FF H. The 2 in the FIX 2 command determines the spacing of the commas!

Try the same operation using FIX 4 and you will observe the commas interspersed every four digits. (See Note 2)

Flag 21 Not Accounted For

Normally if you set flag 21 on your HP-41 and you have a functioning printer attached, the calculator will print out a number or string of characters each time you execute a VIEW or AVIEW function in a running program. On the other hand, if you do not have a printer, or if the attached printer is turned off, then the HP-41 will halt whatever it is doing and display the contents of the alpha register or X-register any time that an AVIEW or VIEW function occurs.

You might, rightfully, suppose that the BINVIEW, OCTVIEW, and HEXVIEW functions would conform to this flag 21 protocol. Surprise! These functions totally ignore flag 21.

So, if you plan to use any of these VIEW functions in a program, you will have to place a STOP command after each VIEW function to get the program to halt. Furthermore, the only way to print the contents of the display is to operate the printer in NORM mode. (See Note 3)

Input Functions

The BININ, OCTIN, and HEXIN functions in the Advantage Module present some interesting features. And, yes, even an undocumented feature or two of which you should be aware.

A First Approximation to Keyboard Lockout

Before reading any further, take a second to prepare the 41C to do some potentially irritating operations.

Switch off USER mode and reset the calculator to FIX 0 display format. Then press [GTO][.][.] and let the calculator PACK itself. The calculator will now be outside the boundaries of the 16-E program (and any other programs that you may have in memory.)

Now turn the USER mode on and press [#][B]. Try keying in the following number (even though it is not a binary number.)

11230045.11

Actually all you will be able to key in is 110011_ B . How about that? Keyboard Lockout! Surprise! This isn't mentioned in the HP Manual. So, let's try some other things.

Again, press [#][B]. This time press and HOLD the [D] key in the upper row of keys. You should see 01 LOG and then NULL. If this were truly keyboard lockout, then this key should exhibit no response. Try this same strategy but press the other top row keys and notice the odd behavior of the display. When you are doing this, be sure you hold down the key until you see NULL in the display. Only then should you release the key.

Now [#][B] again. This time, press the [D] key and release it before you see the NULL prompt. Press the [PRGM] switch to go into program mode and there you will see:

01 BININ

I honestly don't understand how this happens (and it happens with the OCTIN function as well). Here is a way to insert the BININ function in a program, while NOT being in PRGM mode.

It's an undesirable feature, especially if you unwarily insert the instruction into the 16-E program. If this happens, you'll have to delete the unwanted function from the program and then pack it. (What a nuisance!)

Note that the HEXIN function has more of the capabilities of "keyboard lockout," but it is not perfect either.

In short, where the Advantage Module Manual says "The binary input for BININ must be 0's and 1's..." (p.127), highlight the word "must" as a warning to yourself that you should not trust undocumented features no matter how alluring they might be. Observe the same precaution with the OCTIN and HEXIN functions as well.

No Excessive Input

There is one working instance of keyboard lockout in the Advantage Module. The Advantage Module's input functions will not let you key in more than ten binary or octal digits. It will also not allow for more than eight hexadecimal digits even though, theoretically, the display and the calculator should be able to handle such a number as 123 456 789 Hex = 4,886,718,345 Dec.

Problems With Solutions...

Here are several number base conversion problems. Try solving these problems on the 16-E as a way to become familiar with the assigned keys.

1. Input ABCDE Hex and display the number in octal and decimal format.

Here are the keystrokes that will let you solve this problem:

KEYSTROKES	DISPLAY	COMMENTS
[USER]		Turn on USER Mode.
[#][E]	_ H	
ABCDE	ABCDE_ H	
[Enter]	70310	Decimal number
[C]	2536336 O	Octal number

You may be wondering why you need to press the [ENTER] key at the end of the string of hex digits,

ABCDE. The reason for pressing the [ENTER] key is to terminate the digit entry before you press another key. In the above problem, if you did NOT press the [ENTER] key, and then pressed the [C] key, you would see ABCDEC_ H in the LCD.

There are other ways to terminate hex number entry. For example, you could press the [ALPHA] key twice. Or you could press and hold the [TAN] key, for instance, until you saw the NULL message in the LCD. When you release the key, the hex number entry will terminate and you will see the decimal equivalent in the X-register. I like to use the [ENTER] key: it makes life simpler.

2. Input 4771 Octal and convert this number to hex and decimal format.

KEYSTROKES	DISPLAY	COMMENTS
[#][C]	_ O	Octin
4771	4771_ O	
[E]	9F9 H	You do not need to press [ENTER] here, since pressing the [E] key will terminate octal number entry.
[CLX]	2553	CLx really only clears the display register and shows you the naked decimal number in the X-register.

3. Input 11011001 binary and convert this to decimal, octal and hexadecimal format.

KEYSTROKES	DISPLAY	COMMENTS
[#][B]	_ B	
11011001	11011001_B	
[Enter]	217	This [ENTER] terminates the binary input and shows the equivalent base-10 number in the X-register.
[C]	331 O	
[E]	D9 H	

Boolean Functions

Having said enough, for now, about the number conversion functions, let's move on to the "Boolean Functions" in the Advantage Module. These functions include AND, inclusive OR, XOR (eXclusive OR), NOT, BIT? (a bit-tester), and ROTXY (a fast bit rotating function).

Again, if you are a past master of the AND, OR, and XOR functions in the Advantage Module, you may omit this part of the chapter. There are only two sections that you should look at. The first section will answer the perplexing question: "How does NOT differ from NOT?" See page 60 for the answer. The second section will show you how to overcome the BIT?-bug in the Advantage Module. This section is on page 62.

AND, OR, XOR

Some desktop computers will give you an 8-bit AND, OR and XOR function. The Advantage Module goes several steps beyond this. In this case, "smaller means bigger!" The AND, OR, and XOR functions in the Advantage Module are "true" 32-bit functions. And they are fast: each takes about 0.25 seconds to operate on any two numbers. They always operate on all 32 bits.

What this means is that if you key in the following two 32 bit hexadecimal numbers and press the [#][*] key, you will AND these two numbers together and the result will be a 32 bit number.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
FEDCBA98	FEDCBA98 H	This is a full 32 bit number since each hex digit is the same as 4 binary digits (bits).
[ENTER]	4275878552	This is the decimal equivalent of this hex number.
[#][E]	_ H	
89ABCDEF	89ABCDEF H	Another 32 bit hex number.
[#][*]	2290649224	The decimal result of ANDing the two hex numbers.

[E]	88888888	H	This is the hex representation of the result.

To get a better idea of what is going on in this example, here is a table of the first 16 hexadecimal numbers and their binary equivalents.

HEX	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Each time you key in a separate hex digit you are, in effect, keying in 4 binary digits. So, one way of looking at the result of your entering the two hex numbers above would be to imagine that there are really two binary numbers in the X and Y registers of the stack.

In the Y-register: 1111 1110 1101 1100 1011 1010 1001 1000

In the X-register: 1000 1001 1010 1011 1100 1101 1110 1111

I have inserted blank spaces between the groups of bits just to make it easier to read and interpret.

Now, pressing the AND keys ([#][*]) amounts to 32 separate applications of the standard truth table definition of AND, namely:

In the X-register: 1000 1000 1000 1000 1000 1000 1000 1000

which is equivalent to 88888888 hex.

In case you have forgotten, here is what the standard truth table definition of AND looks like.

p	q	AND
0	0	0
0	1	0
1	0	0
1	1	1

AND is sometimes called "logical multiplication" because, if you multiply the binary digits in the "p" column by those in the "q" column, you get the results shown in the AND column. (That is a good enough reason for assigning the AND operator to the [#][*] key on the HP-41.)

Logicians like to call AND by its fancy name "conjunction." The English equivalent of the truth table is usually given as: "The only time that a statement is TRUE (1) is when all the separate conjuncts are true."

Computer scientists and engineers are more likely to work with the AND operator in terms of 0 and 1 and forget about the English meaning of the AND operator.

For those of you who may have forgotten the truth table definitions of the OR operators, here they are:

p	q	OR
0	0	0
0	1	1
1	0	1
1	1	1

p	q	XOR
0	0	0
0	1	1
1	0	1
1	1	0

The OR on the left is sometimes called the "inclusive OR" to keep its meaning separate from the Exclusive OR on the right.

Logicians refer to Inclusive OR by its formal title, "disjunction."

Sometimes Inclusive OR is called "logical addition" and that almost works. The sum of the bits in the "p" column with those in the "q" column gives the results in the OR column—almost. (The last row is "off" by 1.) But three out of four isn't bad.... Anyway the idea of "logical addition" makes it seem logical to assign this function to the [#][+] key.

Inclusive OR is like the English phrase "One or the other OR both," while Exclusive OR is more like the phrase "either one or the other but NOT both." The XOR function is assigned to the [#][/] key. There is no logical reason for this key assignment: it's just convenient.

An Example:

Let's use the same hexadecimal numbers as before, and see what happens when we OR them together.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
FEDCBA98	FEDCBA98 H	This is a full 32 bit number, since each hex digit is the same as 4 binary digits (bits).
[ENTER]	4275878552	This is the decimal equivalent of this hex number.
[#][E]	_ H	
89ABCDEF	89ABCDEF H	Another 32 bit hex number.
[#][+]	4294967295	The decimal result of ORing the two hex numbers.
[E]	FFFFFFFF H	This is the hex representation of the result.

What happens if we use XOR on these same two numbers?

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	

FEDCBA98	FEDCBA98 H	This is a full 32 bit number, since each hex digit is the same as 4 binary digits (bits).
[ENTER]	4275878552	This is the decimal equivalent of this hex number.
[#][E]	_ H	
89ABCDEF	89ABCDEF H	Another 32 bit hex number.
[#][/]	2004318071	The decimal result of XORing the two hex numbers.
[E]	77777777 H	This is the hex representation of the result.

The Complement Operator

The NOT function is based on the truth table definition

p	NOT-p
0	1
1	0

That is an easy definition to work with. It is also a very easy procedure for a computer to do. All computers have built into them an "inverter circuit"

which will change the state of a signal from OFF to ON or ON to OFF.

The Advantage Module has the advantage of performing 32 of these inversions every time you use the NOT operator.

For example, try this..

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
000000DE	000000DE H	
[ENTER]	222	The decimal equivalent of the number.
[XEQ][ALPHA]		
NOT [ALPHA]	4294967073	
[E]	FFFFFF21 H	The hex equivalent of the complemented (NOTed) number.

To get a clearer picture of what is going on here, imagine the binary version of the same keystrokes.

222 = 0000 0000 0000 0000 0000 0000 1101 1110
NOT = 1111 1111 1111 1111 1111 1111 0010 0001
=4294967073(dec)

All the 1's have been changed to 0's and all 0's to 1's.

Please note that I had you key in the NOT function letter by letter, [XEQ] "NOT." I did not have you use the NOT function contained in the 16-E program. The two functions are slightly different. The Advantage Module's NOT function is spelled N-Oh-T, while the

16-E's NOT function is spelled N-zero-T. This may be confusing, but it will become clear later on. The Advantage Module's NOT function operates on all 32 bits of a number. The 16-E's NOT function operates on any number of bits from 1 to 32. It's slower, but more flexible.

Here are some problems that will let you check out your understanding of the Advantage Module's logic functions.

Problems With Solutions...

1. What is the result of 712 dec AND 444 dec ?

KEYSTROKES	DISPLAY	COMMENTS
712	712	
[Enter]	712	
444	444	
[#][*]	136	

2. What is the result of 712 Octal OR 444 Octal?

KEYSTROKES	DISPLAY	COMMENTS
[#][C] 712	712_ O	
[#][C] 444	444_ O	
[#][+]	494	
[C]	756 O	

3. What is the result of 712 Hex XOR 444 Hex ?

KEYSTROKES	DISPLAY	COMMENTS
[#][E] 712	712_ H	
[#][E] 444	444_ H	
[#][/]	854	
[E]	356 H	

Bit Manipulations

The BIT? function in the Advantage Module allows you to test the status of any of the bits in a 32 bit word.

The BIT? operator is something of a hybrid. It requires two numbers. The first number is the decimal equivalent of a binary number and the second number is the position of the bit you want to test. The first number is in the Y-register. The bit-position number is in the X-register. When you execute the BIT? function (XEQ "BIT?"), the display will show only the response YES or NO to let you know if the bit is a 1 or a 0, respectively. At that point, when you press the [Clx] key to clear the display, you will find that nothing has changed, of course. Clearing the display ([CLx]) shows that the stack has not dropped.

This operation of the BIT? function is different from the implementation of the "B?" function on the HP-16C. In the HP-16C the "B?" function operates in a typical RPN fashion, dropping the stack and preserving LastX. In the 16-E program, the BIT? function has been imbedded in some other code to make it behave more like a true RPN two-number function. (See Note 4.)

Finally, when you use the BIT? function in a program, you will not see any YES or NO response in the display. This is normal. In a program the BIT? function operates just like one of the other HP-41 conditional functions (X=Y?, X=0?, etc.). In this case, if the bit in question is "1", then the program executes the next statement in the sequence. If the bit in question is "0", the program will skip the next program step.

As an example of how you would test the bits in a number, try the following example...

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
F0F0F0F01	F0F0F0F1 H	This is the number whose bits we want to test.
[ENTER]	4042322161	
31	31_	
[XEQ]"BIT?"	YES	The 32nd bit is set.
[CLx]	31	
[RDN]	4042322161	
15	15_	
[XEQ]"BIT?"	YES	The 16th bit is also set.
[RDN]	4042322161	
1	1_	
[XEQ]"BIT?"	NO	The 2nd bit is not set.

Rotating Bits

The ROTXY function is a very fast bit rotating function. It operates on all 32 bits in a word.

To see how this function operates, do the following exercise.

Press [#][E] and key in ABCDEF , a six-digit, hexadecimal number. Realize, however, that, internally, the calculator is working with an 8-digit hexadecimal number (= 32 bits). So, close your eyes and imagine the internal representation as 00ABCDEF.

Now, to rotate the 32 bit number 8 bits to the right, imagine that you have printed the 8 hex digits on a strip of paper and that you have wrapped the strip of paper around a coffee mug. The two ends of the strip of paper touch each other. Finally, imagine that you rotate the coffee cup counter-clockwise to the right by 2 hex digits. What you would see, as a result, would be the sequence EF00ABCD.

To get the HP-41 to do something similar, press the [Enter] key (and see 11259375) . Next, key in 8 and then XEQ "ROTXY" and see 4009798605. Finally, XEQ "HEXVIEW" and you will get EF00ABCD H in the display.

Suppose you want to rotate a number to the left instead of the right. Will using a negative number do the trick? The answer is "no" (well, sometimes "yes" and sometimes "no"). What will work for left rotations is to use a right rotation of (32-number of left rotations). Thus, if you want to rotate a number left by 8 bits, use (32-8 =24) ROTXY to get the job done.

This ability to rotate numbers will allow us to emulate such features as shifting or justifying numbers in the 16-E program.

Summary

You have covered quite a bit of material in this chapter. So far, you have learned about the number base conversion functions in the Advantage Module. You had a chance to learn how to use these functions to covert numbers in base 10 (decimal) to and from bases 2 (binary), 8 (octal), and 16 (hexadecimal). You also learned about some of the peculiar features of these functions in the Advantage Module. For example, the BINVIEW, OCTVIEW, and HEXVIEW functions all ignore the printer existence flag (flag 21) in the HP-41. AND, OR, XOR, and NOT.

Then you took a look at the Boolean (logic) functions: AND, OR, XOR, NOT. We reviewed the truth table meanings of these functions and saw a couple of instances of their use.

Finally we looked at the functions that would let you test the ON or OFF status of any bit in a binary number (BIT?) and rotate all 32 bits of a number to the right.

That's a lot of material to cover, especially if it's all new to you. Much of the material will be covered again in the next chapter when we look at similar functions in the context of the 16-E program.

For now, try to do the following problems using your HP-41 with the Advantage Module. Convince yourself that you are beginning to learn at least some of this material.

POP QUIZ
(Number Base Conversions)

1. Input EC98 Hex and convert to Octal.

2. Input 1011000000 Bin; convert to Hex and decimal.

3. Input 17776 Octal and convert to hex and decimal.

4. Input 763 Dec and convert to hex, octal and decimal.

(Some Problems Using Logic Functions)

5. What is the result of NOT(FFFFECDB hex) AND NOT(1101 bin)?

6. What is the result of NOT(317 Oct XOR 45 Oct) AND 3072 dec?

7. What is the result of FFFF Hex AND 1111 Bin?

8. What is the result of EA72 Hex XOR EA73 Hex?

9. What is the result of 7613 Oct AND 313 Dec XOR 547 Hex?

ANSWERS TO THE POP QUIZ

1. [#][E] EC98 H [ENTER] (see 60568) and the press [C] to see 166230 O.

2. Press [#][B] 1011000000 [ENTER] (see 704)—the decimal equivalent. Press [#][E] to see 2C0 H (the hex equivalent.)

3. Press [#][C] 17776 [ENTER] (see 8190)—the decimal equivalent. Press [#][E] to see 1FFE H (the hex equivalent).

4. Key in 763 and press either press [#][B] or [XEQ]"BINVIEW" to see 1011111011 B. Press [C] to see 1373 O and then press [E] to see 2FB H.

5. Press [#][E] and key in FFFFECDB. The [XEQ]"NOT" and see 4900 (the complement in decimal form). Press [#][B] 1101 [XEQ]"NOT" to see 4294967282. Press [#][*] to AND the two numbers and see 4896. Finally press [E] to see 1320 H.

6. Press [#][C] 317 [#][C] 45 [#][/] and see 234 (the decimal representation of the XOR of the two octal numbers). Then press [XEQ]"NOT" and see 4294967061. Press 3072 [#][*] and see 3072. To view the hex equivalent (C00 H), just press [E].

7. Press [#][E] FFFF [#][B] 1111 [#][*] to see 15 (F H if you then press [E]).

8. Press [#][E] EA72 [#][E] EA73 [#][/] to see 1 which is 1 in any other base as well.

9. Press [#][C] 7613 [ENTER] (see 3979) and then 313 [#][*] (see 265). Press [#][E] 547 [#][/] and see 1102 (dec).

NOTES...

NOTE 1: Actually the DISPLAY could handle nine or ten F's, but the 41CV can't handle this large a number in integer format. To see what would happen if you tried to convert FFFFFFFFF hex to decimal, just use the 41C to compute 2^36 - 1.

To perform this computation, turn off the USER switch. Then press 2 [Enter] 36 [y^x] 1 [-] and you will see 6.8719476 10 in FIX 9 display format. The 41C has exceeded the limits of its FIX display format and has defaulted to SCI format with the loss of some accuracy.

NOTE 2: You may find this helpful. Personally, I find it confusing. The commas are interspersed every three digits in decimal format and then every two spaces in binary, octal, and hex format. Because of this potential confusion, I have arbitrarily set the display format to FIX 0 and cleared flag 29 at the beginning of the 16-E program. Later, you may wish to change this in the 16-E program to suit your own taste.

NOTE 3: As a matter of fact, no amount of coaxing or swearing will get the BINVIEW function to print. The OCTVIEW and HEXVIEW functions will trigger the printer, but not BINVIEW. In the 16-E program, the BINVIEW function is surrounded with flag settings and clearings. All this flag-waving is necessary just to get the BINVIEW function to display something on a video monitor.

This anomalous behavior of the Advantage Module's VIEW functions may or may not be a bug. Nevertheless, this total ignoring of the status of flag 21 is definitely inconsistent with the normal operation of the 41C.

Does all this mean that you should send the Module back to HP? I wouldn't. What the programmers at HP have managed to accomplish is so good, in my estimation, that I am willing to overlook a few minor flaws. This is a human tool, made by real humans. It's a work of art. And so far, I have not found any perfect work of art.

NOTE 4: There are certain anomalies with the BIT? function. On occasion, there is no response shown in the display. I leave it to the interested reader to discover the pattern of responses and null-responses. I have documented this anomaly and sent the "bug" report to Hewlett-Packard Technical Support.

Chapter 4.

A STEP BY STEP GUIDE TO USING THE 16-E PROGRAM

It's time to take a close look at the 16-E program to see how it can expedite your work in computer science and/or engineering.

In this chapter of the book we will be learning about...

1. Word sizes and how to set them using the 16-E program.

2. Three new base-10 display formats: unsigned, one's complements and two's complements.

3. How the CHS function operates in the 16-E program.

Setting the Word Size

"Word size" means the number of bits (binary digits) that the 16-E will use to represent a binary number.

Here is the procedure to set the word size in the 16-E program:

Press the [#][D] key (shifted D). The computer will show you the current word size with the prompt:

WSIZ = 16

The 16-E program uses the contents of register 00 to save the current word size. So, if you have been using

the computer for other purposes, the current word size may be whatever remains in register 00.

Let's set the word size to 8 and work in that environment for the time being. Simply press 8 and [R/S]. The computer will display:

WSIZ = 8

If you press [R/S] again, the program will only re-display the above prompt. The 16-E will prevent you from leaving this part of the program and accidentally jumping into another part of the program.

By the way, one of the users of the 16-E program asked me to alert other other users to a potential problem. He had been running another program in the HP-41C. This program used registers 01 through 04 to store data. When he started to use the 16-E program, he executed the WS function and discovered that the word size was 16 bits, just as he had left it. So he started to work some digital circuit problems. He got some very bizarre results, and was ready to call me with a "bug report". Then he remembered that he had inadvertently changed the contents of registers 01 through 04. He re-ran the WS function again and, this time, he keyed in 16 for the desired word size and then pressed the [R/S] key. The 16-E program worked correctly once again. The moral of the story is this: if you seem to be getting some obviously incorrect results to your computations, try resetting the word size and see if that doesn't clear up the problem.

This is really all you need to know about the Word Size function in the 16-E program. If you want to, you may skip ahead to the next topic.

If, however, you want to get on more intimate terms with the word size operation, read on.

The Advantage Module's logic functions work only on 32 bit numbers. On the other hand, most computer scientists work with either a four, eight or sixteen bit environment. So one of the first capabilities of a computer science calculator should be the ability to set the word size.

The 16-E has this capability, and then some. It won't limit you to just the standard word sizes. You can specify any word size from 0 to 32 bits. (Granted, you may never need a word size of 11, but it's reassuring to know that you can work problems in an 11 bit environment.)

For example, suppose you wanted to represent the decimal number 6 in binary.

The decimal number 6 is 110 binary (you should read this as one-one-zero and not one hundred ten). 110 is three binary digits long. Another way of saying this is "110 has a word size of 3 bits." So, if you wanted to, you could set the word size to 3 (don't do this, however.) In that 3-bit word size, you could work with the range of number 0 to 7. Rather limiting, wouldn't you say?!

Now, with a word size of 8, the binary representation of 6 would be 00000110. There is still the "110" on the right of the number which gives the value of 6 in base 10.

The added zeroes in the binary representation of a this 8 bit number are shown to give you a picture of how wide the number is. The 16-E does not display

these leading zeros. You will have to supply the leading zeros in your imagination. (I have tried in vain to find a simple way to get the 16-E program to display these leading zeros. There does not appear to be a simple way to get the job done other than redesigning the Advantage Module itself. If you can find a good way of doing this, be sure and let me know.)

Using the [D] Key With A Chosen Word Size

You can use the [D] key to make the computer trim a number down so that it will fit into the current word size.

Now that you are working in an 8 bit environment, let's try a simple experiment.

You undoubtedly know that the maximum positive number of 8 bits is 11111111 Binary or 255 Dec (FF Hex). But suppose you didn't know this. The 16-E program can show you the maximum number in any given word size. Here's how...

For now, we want to work with positive (unsigned) numbers only. (Later we will look at how a computer can show negative numbers.) So to make sure that we are working with positive numbers only, press the [#][3] keys. This will set the UNSIGNED display format. The goose will fly across the LCD, and the 3 annunciator will appear in the display. Now we're ready.

So, press the [#][B] key to activate the BININ function and key in ten 1's. Granted this is more than the computer can handle with a word size of 8 bits, but let's see if the 16-E can deal with that.

Press [Enter] and you will see 1023. This is, indeed, the decimal equivalent of 11111111(bin), but it is too large a number for one having only 8 bits.

Now press the [D] key. The goose will fly and the number 255 will appear in the display. Press the [B] key and you will see that the 16-E has truncated the number to use only the least significant 8 bits of your input: 1111 1111 B.

Try the above sequence of instructions again. This time key in 1110000000 B and see what you get. You should, of course, see 128 and then 10000000 B. Do you see what happened? It seems that the 16-E program has kept the 8 bits on the right of the number and has thrown away any other binary digits (bits) on the left of the number.

You can try this procedure with any decimal (or hex or octal B number) that is greater than 255(d). The 16-E will trim the number so that only the first 8 bits of the number (those numbered 0 to 7 from the right) will be used in computations.

Other Display Formats

So far you have been working in UNSIGNED format. In the real world of computing, numbers are both positive and negative. Internally, computers do not use a minus sign to save a negative number. The typical way to do this is to have the "most significant bit" (the one furthest to the left) represent the sign of the number. This strategy leads to two different display formats on the 16-E calculator: "two's complement" and "one's complement" formats.

TO ESTABLISH THE 1'S COMPLEMENT FORMAT, PRESS THE [#][1] KEYS.

USE [#][2] TO SET THE HP-41 IN 2'S COMPLEMENT FORMAT.

You can check which display format you are in by looking at the numeric flag-annunciator in the display. The 1 and 2 flag-annunciators stand for 1's and 2's complement, respectively and 3 stands for unsigned format.

If you are already a whiz at working with base 10 numbers in these two different formats, you may jump over the next discussion and move on to page 86.

Let's take a good look at these different, and sometimes confusing, display formats.

To do so, let's get the calculator to work with only a limited range of numbers: 0 to 7. You will remember, from above, that we can do this with a word size of 3. So press [#][D] and key in 3 followed by [R/S] when you see the WSIZ=8 prompt. This will give you the message WSIZ=3 in the LCD.

Next, press the [#][3] keys to make sure that you are in UNSigned display format. You should see the decimal number 7 in the LCD. By the way, 7 is the largest positive number for a word size of 3 bits. To see the binary equivalent of the number 7, you may press the [B] key and see 111 B in the LCD.

Suppose we wanted to represent both negative and positive numbers in this limited word size. We could do this by setting aside the Most Significant Bit (MSB)

to stand for the sign of the number. For this reason the MSB is often called the SIGN BIT.

According to the convention, in computing circles, if the MSB of a binary number is 1, then the equivalent base-10 number is said to be negative. If the Sign Bit is 0 then the base-10 equivalent number is understood to be positive.

Now, if the most significant bit represents the sign of the number, then only two bits remain to express the quantity of the number. So now it would appear that the largest number would be +3 and the smallest number would be –3. That sounds reasonable, but, in this case, reason is not the sole arbiter.

Lest you think that putting a 1 or 0 in the sign bit is all there is to creating a negative or positive number, think again.

Here is a table of values that we will refer to in the discussion that follows.

BINARY	DECIMAL		
NUMBER	UNS	1c	2c
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	–3	–4
101	5	–2	–3
110	6	–1	–2
111	7	–0	–1

On the left is the binary version of the numbers from 0 to 7, and in the second column is the Unsigned Decimal equivalent of these binary numbers.

Then comes the "1c" column. The positive numbers in this column correspond to those in the UNS column. Next, look at the bottom half of the chart. To generate a "negative decimal number," start with the binary equivalent of the positive value of the number and change all the 0's to 1's and all the 1's to 0's. This process is called "complementing" or inverting. The 1 in the sign bit says that the number is negative. The other two bits gives the magnitude of the number.

Thus to generate -3(dec), in 1's complement format, you would take the binary equivalent of 3, 011(b), and invert all the 1's and 0's to get 100(b). Similarly, to generate -1(d), you would take 1(d) = 001(b) and complement the binary number to get 110(b). What happens if you complement 0(d) = 000(b)? Well, you get 111(b), which is, indeed, a negative number. It is a most unusual negative number: -0(d).

Perhaps because of trying to explain to people the meaning of "negative zero," computer scientists have devised another way to represent negative numbers: the "two's complement" format. It is simply the "one's complement" format with one (1) added to the negative numbers. This eliminates having to deal with -0, but now you have to deal with the smallest integer value for a three bit word being "-4". Take your pick of which display format you like. Two's complement is more widely used in computer science.

You can get the 16-E to show you which numbers are which in this scheme of things.

For example, to see the one's complement representation of 7, just key in 7 and press the [#][1] keys. You should see -0 in the display. You can then

press the [#][2] key to see the two's complement representation, –1.

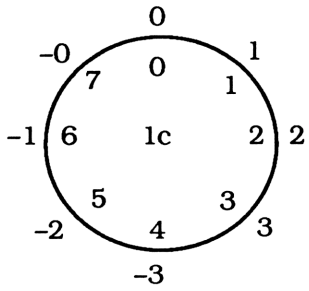
Here is an alternative keystroke sequence to do the same thing:

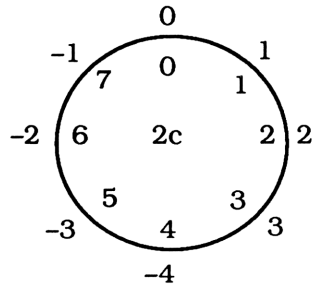
TO EXAMINE A NUMBER IN ANY GIVEN DISPLAY FORMAT, SET THE DISPLAY FORMAT FROM THE KEYBOARD OF THE CALCULATOR. NEXT, KEY IN THE POSITIVE DECIMAL NUMBER. FINALLY, PRESS THE [D] KEY (XEQ "DECV").

Thus, for example, in two's complement, if you key in 5 and press [D], you will see the value –3 in the display.

If you are clear on all this, you may hop over the next section and try your hand at some problems. On the other hand, if you are still a bit confused, perhaps this next section will help you understand the use of signed numbers in computers.

Another way to view the relationship between unsigned and signed numbers is to view them as numbers on the face of a clock. Below is a diagram of this for a word size of 3 bits.





In case you are wondering: "why do you need to complement a number in order to get a negative number?" The answer lies in the way in which computers do arithmetic.

When computers add, they do so in binary. And, when the computer adds –45 and +45 in binary, it had better get a result of 0, right?

Now, if all that happened in a change of sign was to change the MSB, then the ADDITION of

```
+45 =  00101101
–45 =  10101101     (wrong way to create a negative
                     number)
  0 ?  11111011 = 251 in 8 bits.
```

This is obviously not 0 as it should be.

On the other hand, when the binary number is complemented along with the change of the Most Significant Bit, then the two numbers do, indeed, add together to become 0.

To see this, take the 2's complement of 45:

+45(d) = 00101101(b) and the 1's complement is

–45(d) = 11010010(b) and the 2's comp. is this no.

```
      +             1(b)
  11010101(b) (= –45(d) in 2's complement).
```

Now, adding 00101101(b)

+11010011(b)

gives 100000000(b)

If you have never done binary addition before, it's just

like decimal addition, except, when you add 1(b) + 1(b) you get 10(b). You would keep the 0 and carry the 1 into the next column and add that to the other numbers in the column.

In this case, we get a number that is too large for an 8-bit word size, so the computer ignores any bit beyond the MSB which leaves us with 00000000(b) = 0(d). This is the right answer.

All of this inverting and working in binary numbers is necessary because our stone age computers only work with two numbers: 0 and 1. It's truly amazing what you can coax out of a machine that only can reckon with 0's and 1's. Now, suppose on a different planet, in a galaxy far, far away, the electronic wizards had invented a transistor that would show more than just two electrical states. Suppose the transistors could register 10 discrete electrical states. This would make the design of their computers far simpler, smaller, cheaper and faster. Of course, computer science students on this planet would have to adapt to using truth tables that contained 10 truth values rather than just the two (True and False) that we have to deal with. Isn't that something to boggle your mind? I'll bet you didn't know that some logicians on earth have already been working logic problems in 10-valued logic. We all can do arithmetic in a 10-valued number system. All we're waiting for is for you to invent the 10-state transistor. Now there's a challenge for you computer engineers!

Coming back to the real world, here are some problems that you can try on the HP-41 to help you learn the keystrokes to use.

Problems With Solutions...

1. What is the "two's complement" representation of the following hexadecimal number? Use a 32 bit word size.

   D014 = ?

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 32 [R/S]	WSIZ = 32	Set word size to 32 bits.
[#][E] 0000D014	0000D014_ H	XEQ "HEXIN" and key in hex number.
[#][2]	53268	XEQ "2c"and see the answer.

2. What is the octal representation of 53268(d) ?

KEYSTROKES	DISPLAY	COMMENTS
[C]	150024 O	XEQ "OCTVIEW"

3. What is the binary representation of this number ?

KEYSTROKES	DISPLAY	COMMENTS
[B]	53268	XEQ "BVU" (The 16-E is programmed to default to unfor-

matted decimal output when the binary representation is out of range.)

4. 2655F314(h) =? in decimal format

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	XEQ "HEXIN"
2655F314	2655F314_ H	Key in hex number
[Enter]	643166996	Equivalent decimal number.

5. What is the octal representation of the above number, 643166996(d)?

KEYSTROKES	DISPLAY	COMMENTS
[C]	4625371424 O	XEQ "OCTVIEW"

6. What is D9AA0CEC(h) in decimal 2's complement format?

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	H	XEQ "HEXIN"
D9AA0CEC	D9AA0CEC H	

[Enter]	3651800300	Unsigned number
[D]	-643166996	XEQ "DECV" (2's complement)
[Clx]	3651800300	Clx to see the unsigned number in the X-register.

OK! Are you ready? Here it comes, *another...*

POP QUIZ!

Here are some hex numbers. Perform the indicated conversions.

1. AB643106(h): convert to decimal both 1's and 2's complement format.

2. 106E0FAB(h): convert to unsigned, 1's complement, 2's complement and octal formats.

3. FEDCBA98(h): convert to unsigned, 1's complement and 2's complement decimal format.

4. FFFFFFFF(h): convert to unsigned, 1's complement and 2's complement decimal format.

5. What is the largest positive number that can be accomodated with a word size of 32 and 2's compl. format?

6. Similarly, what is the largest positive number in 2's compl. format with a word size of 16 bits?

ANSWERS TO THE POP QUIZ

Isn't this book neat? You get the answers to ALL the problems and you don't even have to look in the back of the book. Try not to peek before you get some answers, however.

1.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 32 [R/S]	WSIZ = 32	
[#][E]	_ H	
AB643106	AB643016 H	
[#][2]	-1419497210	2's complement dec. number.
[#][1]	-1419497209	1's complement dec. number.
[CLx]	2875470086	Unsigned dec. equivalent of original hex number.
[C]	OUT OF RANGE	Number is greater than 1073741823(d)

2.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
106E0FAB	106E0FAB H	
[ENTER]	275648427	Unsigned format
[#][2]	275648427	2's complement
[#][1]	275648427	1's complement
[C]	2033407653 O	

3.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
FEDCBA98	FEDCBA98 H	
[ENTER]	4275878552	Unsigned format
[#][2]	-19088744	2's complement
[#][1]	-19088743	1's complement

4.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]	_ H	
FFFFFFFF	FFFFFFFF H	
[ENTER]	4294967295	
[#][2]	-1	2's complement
[#][1]	-0	1's complement

Remember that eight F's hex = thirty two 1's and, in 1's compl., when the MSB is 1, then the number is considered negative and all 31 remaining bits are complemented to become 0's. This gives the representation of -0.

5. One way to let the calculator answer this question is to reason that if you use the MSB as a sign bit you are left with only 31 bits. So press [#][E] FFFFFFFF and then press [#][D] 31 [R/S] and see WSIZ = 31. Then press [E] to see 7FFFFFFF H and press [CLx] to see 2147483647 (d).

6. For this question you can use the [#][D] 15 [R/S] and [E] keys to see 7FFF H and then [CLx] to see 32767 dec.

The Effect of CHS

Changing the sign of number only has meaning in those display formats that accomodate negative signs, namely 1's and 2's complement formats.

To see the effect of the CHS (Change Sign) operator in the 16-E calculator, first set the word size to 8 bits and establish 2's complement format. ([#][D] 8 [R/S] and [#][2]).

Press [#][B] (XEQ "BININ") and key in 11010011. Now press the [D] key to view the decimal equivalent in 2's complement. You should see –45. The MSB is 1, so the number is negative. Now press the [CHS] key (XEQ "CHS") and see 45 in the display. Press the [B] key (XEQ "BVU") and you will see 101101 B. Realize that the leading zero (0) has been suppressed and 1 has been added.

Note that the CHS operator in the 16-E does not simply change the sign of a decimal number as it does on a standard 41C. It does a lot more: it complements all the bits in the number.

Later, we will take a look at the logical NOT operator. This is the true complementing operator. Nevertheless, when the display format is 1's complement, the CHS and NOT operators are identical. This correspondence between the CHS and NOT operators does not hold in 2's complement format, however.

If this much about the CHS function is enough for you, feel free to move ahead to page 88. Otherwise, read on...

To take another look at the CHS operator, set WSIZ=16, and establish 1's complement. ([#][D] 16 [R/S] and [#][1]) Press [#][E] and key in FEDC Hex and press the [Enter] key to see 65244. Now press the [D] key (XEQ "DECV") to see the 1's complement representation, -291 dec.

Now press [CHS] and see 291. It appears as if the only thing that has changed is the sign. Then press the [E] key (XEQ "HEXVIEW") to see 123 H. This should convince you that something definitely has happened to change FEDC Hex to 123 Hex.

You can reverse the sign by pressing either the [CHS] key or the [#][-] keys (XEQ "NOT") BUT ONLY IF YOU ARE IN 1's complement format. Try pressing the [#][-] keys this time to see -291.

Here is another example for those who are amused with the lore of numbers. To do this example, set the word size to 32 bits and 1's complement format. Then press [#][E] and key in FEDBCA98 Hex and press the [Enter] and [D] keys to see -190088743. Now change the sign ([CHS]) and finally press [E] to see 1234567 Hex. This is the rest of the hexadecimal digits but in ascending order. (Those of you who think in binary will see immediately how this happens.) Similarly, you can start with 89ABCDEF H and wind up with 76543210 Hex. (I am confident that this has got to be the ultimate answer to one of the most profound questions in the universe.) (See Note 1.)

CHS and UNSigned Format

While you are working with a word size of 32 bits, switch the display format to UNSigned by pressing the [#][3] keys.

Then key in 1 and press the [CHS] key. The display will show 4294967295. Notice that the flag 4 annunciator appears in the display.

The appearance of the flag 4 annunciator is the 16-E's way of signalling that [CHS] has no meaning in this Unsigned format. The program will give you the 2's complement of the number, but it is also telling you that the actual number is negative and is outside the range of the current word size.

Problems With Solutions...

1. What is the largest positive number, in 1's complement format, for a word size of 12 bits? (There are some computers that thrive on 12 bit words.)

One way to find out is...

KESTROKES	DISPLAY	COMMENTS
[#][D] 12 [R/S]	WSIZ = 12	Set word size to 12 bits.
[#][1]		Set 1's complement.
0 [CHS]	-0	
[Clx]	4095	The largest unsigned number in 12 bits.

[USER]		Switch out of USER mode.
2 [/]	2048	Divide by 2.
[XEQ]"INT"	2047	2047 is the largest positive number in 1's compl. format.
[USER]		Switch back into USER mode.

2. What is the unsigned value for -2047 (1's compl.) using a 12 bit word size? What is the hex representation for 2047 and -2047?

KESTROKES	DISPLAY	COMMENTS
2047 [CHS]	-2047	
[Clx]	2048	
[E]	800 H	
[CHS]	2047	
[E]	7FF H	

POP QUIZ (A Shorty!)

1. What is the result of changing the sign of 3456 dec (2's compl. format) with a word size of 8?

ANSWER TO THE POP QUIZ

1. Press the keys [#][2] and [#][D] 8 [R/S] to see WSIZ=8. Key in 3456 and press the [D] key. This will show the result to be -128 (d).

   The number 3456 has been truncated to fit in an 8 bit word. Now press the [CHS] and notice that the sign does not change, rather the 4 flag annunciator is showing. The 16-E is telling you that you are at a point in the 2's complement numbering scheme where [CHS] does nothing except possibly create an overflow condition. Can you find another spot in 2's compl. where the [CHS] operator has no apparent effect?)

REVIEW

In this section of the course you have begun to learn how to use the 16-E program on your HP-41.

You learned what is the meaning of "word size" and how to set different word sizes with the [#][D] keys.

You then learned how to use the [D] key to trim numbers down to size so that they would be within the bounds of the current word size.

Next, you were shown how the computer represents negative numbers. We talked at some length about

the two decimal formats for representing signed numbers.

Finally, we discussed the CHS operator in the 16-E program.

That's quite a bit to learn, especially if you have never encountered any of these ideas before. Why not take a break and come back to this point later.

If, however, you are mentally alert and eager to forge ahead, then, by all means: "avantage, avantage!"

Chapter 5.

THE USE OF THE 0 AND 4 FLAGS BY THE 16-E PROGRAM

The 16-E program uses the 0 and 4 flags to alert you to certain conditions that are taking place in the 16-E.

To get an idea of how these two flags function in the 16-E program, try the following examples.

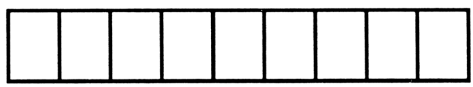
An Example for Flag 0

For the first example, establish a word size of 8 bits and set the display format to 2's complement. ([#][D] 8 [R/S] and [#][2] will do the trick.)

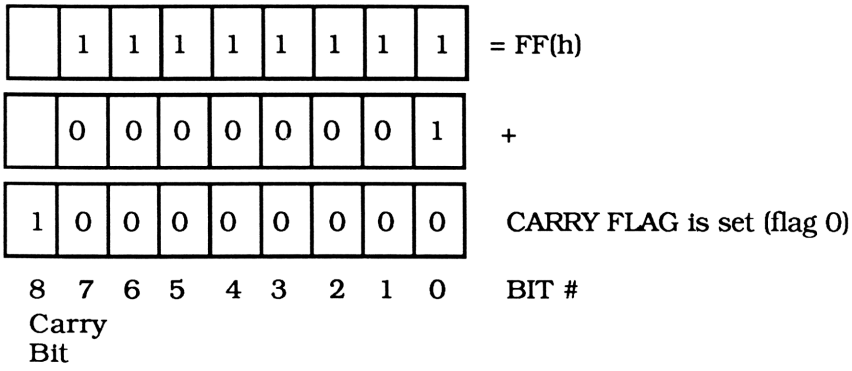
Press [#][E] and key in FF(h). Press the [Enter] key and then the [D] key. You will see the decimal representation of this number in the display, -1.

Now key in 1 and press the [+] key to add -1 and 1. The goose will fly and the display will show the result 0. The 0 flag annunciator will also appear in the display indicating that the "carry bit" is equal to 1.

To visualize the role of a "carry bit," imagine that the 16-E is actually working with a 9 bit word rather than the 8 bit word that you specified.



When you entered FF Hex, you, effectively, set bits 0 through 7 equal to 1. Now, when you added 1 to this number, you would have changed all 8 bits to zeros and the ninth bit would have become 1.



The imagined ninth bit is the carry bit.

Next, key in 1 and press [+] again. The 0 annunciator will disappear. This is as it should be. One added to zero is one and the value of the carry bit is zero.

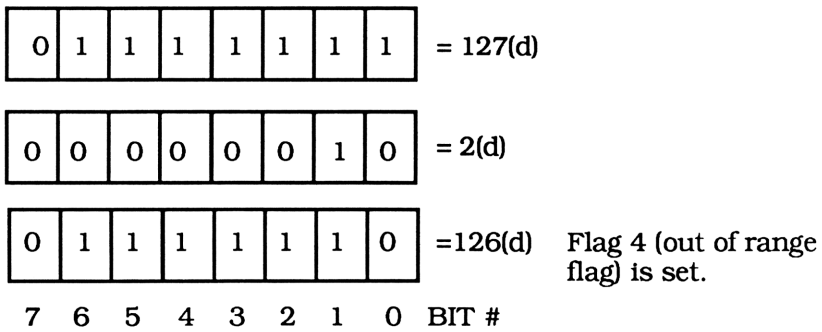
The Role of Flag 4

Flag 4 turns on whenever an arithmetic operation goes out of bounds for a given word size. When flag 4 is on, you should suspect that the results of your computations are inaccurate. Check your work and then clear flag 4, if it is set, before continuing. (Press [shift][CF][D] to clear the flag. See Note #1.)

As an example of "out of range flag," flag 4, try the following. Set the word size equal to 8 and use the 2's complement display format. This is the same configuration as above.

Key in 127 and press the [Enter] key. Next, key in 2 and press the [*] key to multiply the numbers. The result will be 126 and the 4 annunciator will appear in the display. This indicates that the result is greater than the maximum positive number in this display format. The maximum positive number is, of course, 127. Pressing [#][CF][D] will clear the flag.

Note that, when a result is out of range, the lower bits in the result (those that fit in the specified word size) still appear in the display.



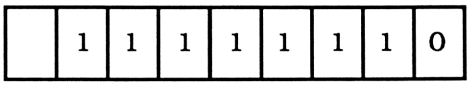
Furthermore, if the operation was multiplication or division, then the sign of the result will be the sign called for by the operation. That is, if one of the numbers was negative then the result will be negative. The display will indicate this even though the magnitude of the answer is incorrectly shown.

Notice in the diagram above, that bit #7, in the product, is 0 even though the unsigned product of 127 * 2 = 254(d) = 11111110(b). In the case of signed numbers the sign bit is generated from the sign bits of the two numbers being multiplied. The product actually has only 7 bits in which to stay in bounds with signed numbers.

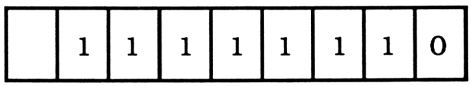
A Difference In 1's and 2's Format

There is a difference in the way that the 16-E handles the carry (or borrow) bit in 1's and 2's complement format.

For instance, in 1's complement format, with word size equal to 8, the following arithmetic operation would be true.

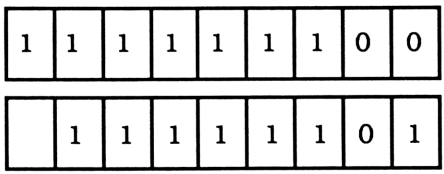


= –1 (dec: 1's complement)



= –1(d)

Adding the two together gives

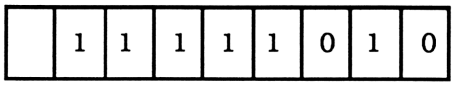


The 1 in the carry bit turns on flag 0, but this is not the correct answer. In order to come up with the right answer, the carry bit must be added to the number to give the figure on the left. This is –2(d), the right answer to the problem.

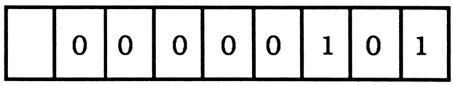
Note that, when you perform this addition on the 16-E calculator, the 0 annunciator turns on. This means that the 16-E had to add 1 to the normal result of adding two binary numbers to come up with the correct result. Computer people call this method of addition an "end around carry."

On the other hand, if the carry bit is 0, the computer will add the binary numbers without any such end around carry.

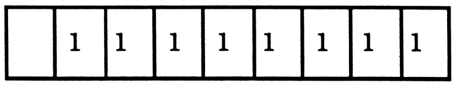
For instance, consider the following example.



= -5 (decimal: 1's complement)



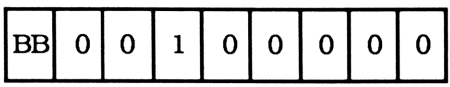
= +5 (decimal: 1's complement)



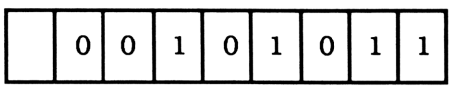
= -0 (decimal: 1's complement)
Carry flag is OFF.

In 1's complement format the procedure for subtraction is very similar to that for addition. In this case, if the carry bit (now referred to as the "borrow bit") is equal to 1 after the subtraction operation, then the computer will subtract 1 from the result of binary subtraction.

For example,

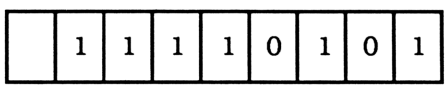


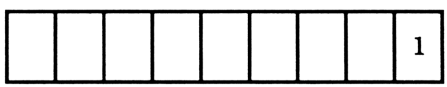
= 32 (decimal: 1's complement)
BB = Borrow Bit



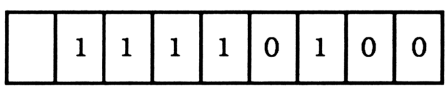
= 43 (decimal: 1's complement)

Subtracting the second from the first (32d–43d)





Which is taken from the borrow bit and subtracted from the result to give...

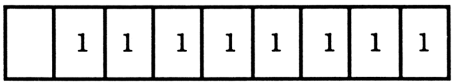


= -11 (decimal: 1's complement)
Carry/Borrow bit is set.

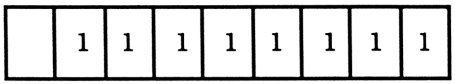
8 7 6 5 4 3 2 1 0 BIT #

On the other hand, in 2's complement (and unsigned) formats, the results of addition and subtraction are simply the sum or difference of the binary numbers.

Thus, in 2's complement format...

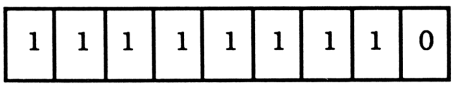


= -1 (decimal: 2's complement)



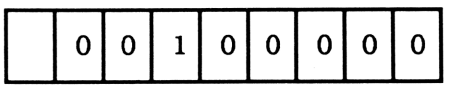
= -1 (decimal: 2's complement)

Added together... gives

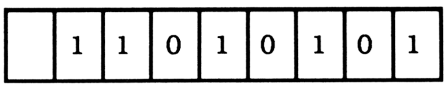


= -2 (decimal: 2's complement)
The carry bit is set, but the answer is correct simply by ignoring the carry bit.

AND NOW...

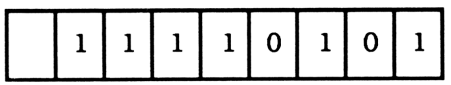


= 32 (decimal: 2's complement)



= -43 (decimal: 2's complement)

When added together give...



= -11 (decimal: 2's complement)

These results are independent of the status of the carry or borrow bits. Perhaps this independence from the carry bit is one reason why computer scientists prefer the 2's complement format to the 1's complement format.

Mixed Mode Arithmetic

You will become more familiar with the significance of the carry and out of range flags in the 16-E calculator as you progress through this manual. For now, let's spend some more time examining how the 16-E calculator handles arithmetic.

You will find that arithmetic on the 16-E is basically the same as it is on the 41C. There is, however, the added feature of saving a few keystrokes over ordinary RPN.

Let's begin with the following example:

Suppose we want to find the result of

$$\frac{\text{C hex} * \text{40 oct} + \text{110 bin}}{\text{13 dec}}$$

all in 2's complement and with word size of 16 bits.

Here is one way to obtain the result of this problem...

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		Set 2's complement format.
[#][E]	_ H	
C	C H	
[#][C]	_ O	
40	40_ O	

[*]	384	Multiply to find decimal equivalent.

NOTE: You should NOT press [ENTER] before pressing the [*] key. When you key in 40(o), the number is in the display and the X-register as well. Pressing [ENTER] at this point would push 40(o) into the Y-register. That is not what we want.

[#][B]		_ B	
110		110 B	
[+]	390		
13	13		
[/]	30		Integer divide, the 0 flag is not set which means that there is no remainder.

As an alternative way of doing the same problem, consider the following....

KEYSTROKE	DISPLAY		COMMENTS
13	13		
[#][B]		_ B	
110		110_ B	
[#][C]		_ O	
40		40_ O	
[#][E]		_ H	
C		C_ H	
[*]	384		
[+]	390		
[X<>Y]	13		
[/]	30		

Notice that this approach requires no use of the

[Enter] key. The Advantage Module input operators evidently have a "built-in-Enter" with automatic stack lift!

Missing Arithmetic Operators

The 16-E calculator lacks all of the other mathematical operators of the 41C. Personally, I deem this to be no great loss. The operators are still there. Just shift out of USER mode and use them. Just realize that the 16-E operates only in integer mode. Thus, such an operation as "1/x" would have no meaning to the 16-E calculator.

If you have need of the SQRT function, just shift out of USER mode, press the [C] key to compute the square root of a number in the X-Register. Then XEQ "INT" from the keyboard. Finally shift back to USER mode and press the [D] key to view the result in the current display format.

Problems With Solutions...

1. The HP-16C Users Manual gives the following problem on page 41:

"Find (5A0 H) / (177764 O)"

The answer given in the manual is FF88 H. The problem assumes a word size of 16 and 2's complement format.

Is this answer correct?

What would be the answer in UNSigned format?

KEYSTROKE	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		Set 2's complement format
[#][E]	_H	
5A0	5A0_H	
[#][C]	_O	
177764	177764_O	
[/]	-120	Dec. number in 2's complement.
[E]	FF88 H	Answer is correct. Note that, since the 0 flag annunciator is off, there is no remainder.

Now, let's do the same integer division in unsigned format....

[#][3] Shift to UNS format

Repeat the same keystrokes as shown above. In this case the answer will be 0, and the 0 annunciator will appear in the display.

REMEMBER:

IN THE CASE OF DIVISION, IF THE 0 ANNUNCIATOR IS ON, THAT MEANS THERE IS A NON-ZERO REMAINDER TO THE DIVISION.

To find the remainder, you may repeat the same keystrokes as shown above. In place of the division

operation, perform the RMD operation on the 16-E. That is press the [#][0] keys. Remember that you are still in UNS format.

[#] CF 00		Clear flag 0
[#][E]	_H	
5A0	5A0_H	
[#][C]	_O	
177764	177764_O	
[#][0]	1440	Dec. number in 2's complement.
[E]	5A0 H	Answer in hex format.

WHOA! Are you a little bit confused at this point? Are you saying, "I see how you get the answer you do in UNSigned format--that's just what I'd get if I did the division on paper, using the decimal equivalents of the numbers...

Thus $$\frac{1440\text{ (d)} = 5A0\text{ (h)}}{65524\text{(d)} = 177764\text{(o)}}$$

I get a quotient of 0 with a remainder of 1440 (d).... *But*, how can it be that, in 2's complement format, the program gives –120 (d) or FF88 (h) with no remainder. That doesn't LOOK right."

It sure *doesn't* LOOK right, at first, but, once you get used to working in different word sized numbers and different complements, it *will* look right. Realize that 177764 (o) is the same as –12 (d) in 2's complement and word size 16. Sure enough, looked at in that light,

$$\frac{14440\text{ (d)}}{-12\text{ (d)}} = -120\text{ (d)} = \text{FF88 (h)}.$$

2. What is (45 Hex) + (25 dec) in Hex format: WSIZ = 16 and 2's complement format?

KEYSTROKE	DISPLAY	COMMENTS
[#][2]		Reset 2's complement format.
[#][E] 45	45_ H	
[Enter]	69	[ENTER] is needed here to terminate hex number input.
25 [+]	94	
[E]	5E H	

3. What is the quotient and remainder of (7 oct) / (5 oct)?

KEYSTROKE	DISPLAY	COMMENTS
[#][C] 7	7_ O	
[#][C] 5	5_ O	
[/]	1	0 annunciator shows non-zero remainder
[#][C] 7	7_ O	
[#][LastX]	5	Get 5 from the LastX register.
[#][0]	2	XEQ "RMD"
[C]	2_ O	

4. Suppose we have an assembler program in a computer's memory. We have called for a dump of

the program and we have located, on our printout, the zeroth line of the program. It is located at the "absolute address" in computer memory of 3F0(h).

Realizing that both program lines and computer memory addresses are numbered sequentially, we wish to find the absolute addresses of lines 15, 25, and 700. We also want to find the address in memory of line -20, since that location contains some useful "header" information that may be causing the program to "crash and burn."

KEYSTROKE	DISPLAY		COMMENTS
[#][E] 3F0		3F0_ H	
[Enter]			
[Enter]			
[Enter]	1008		Load the stack.
15 [+]	1023		
[E]		3FF H	Address of instruction #15
[Clx][Clx]	0		[CLx] once to clear the display. [CLx] twice to clear the X-Register and disable stack lift, so we can put a new number in the X-Register.
25 [+]	1033		
[E]		409 H	Address of instruction #25
[Clx][Clx]	0		
700[+]	1708		
[E]		6AC H	Address of instruction #700

[Clx][Clx]	0		
20 [–]	988		
[E]		3DC H	Address of instruction #–20

REVIEW

In this section of the course, you have had the golden opportunity to see how the 16-E program handles 1's and 2's complement arithmetic (dealing with signed numbers).

You also probably learned more than you wanted to know about the little carry and borrow bits and how they interact with the 0 and 4 flags in the LCD. The 0 annunciator may show that a result used the carry bit. Or, in the case of division, the 0 annunciator will tell you if there is a remainder to your division.

Then you took a look at how the 16-E can deal with chained arithmetic operations and even add hexadecimal numbers to binary numbers at the flick of a finger. That alone is worth the price of admission, wouldn't you agree?

Your knowledge of the workings of the 16-E program is continuing to grow. And, if you have worked most of the sample problems and tried to understand what you were doing, then your expertise in computer math just increased. Do you feel your head getting any larger?

Now, while all this information is fresh in your mind, try the following problems...

POP QUIZ

1. How would you set word size to 16 bits and establish 2's complement, decimal display format with the 16-E program?

2. Once you have done problem #1, then what is 234(o) + 450(d) + ABC(h) in decimal and in octal?

3. What is

   $$\frac{\text{AEF(h)} * \text{256(o)}}{\text{11011(b)} - \text{12(d)}}$$ in hexadecimal? (use a word size of 32)

   (If you get the out of range error, flag 4, retry the problem with WSIZ = 32.)

4. What is

   $$\frac{\text{BBBB(h)} - \text{2222(o)}}{\text{1111(b)} * \text{99(d)}}$$ in binary?

   (Again, do this problem with word size = 32.)

5. What is the remainder in problem 4?

ANSWERS TO THE POP QUIZ

1.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		2 flag appears in LCD

2. [#][C] 234 — 234_ O
   [ENTER] — 156
   450 — 450
   [+] — 606
   [#][E] ABC — ABC_ H
   [+] — 3354
   [C] — 6432 O

3. [#][D] 32 [R/S] — WSIZ = 32
   [#][E] AEF — AEF_ H
   [#][C] 256 — 256_ O
   [*] — 487026
   [#][B] 11011 — 11011_ B
   [ENTER] — 27 — Why? Because next key would be another 1 unless you terminate entry.
   12 — 12
   [-] — 15
   [/] — 32468
   [E] — 7ED4 H — The 0 flag is set.

4. [#][E] BBBB — BBBB_ H
   [#][C] 2222 — 2222_ O
   [-] — 46889
   [#][B] 1111 — 1111_ B
   [ENTER] — 15
   99 — 99
   [*] — 1485
   [/] — 31
   [B] — 11111 B — With 0 flag on.

5. [#][E] BBBB — BBBB_ H
   [#][C] 2222 — 2222_ O
   [-] — 46889
   [#][B] 1111 — 1111_ B

[ENTER]	15
99	99
[*]	1485
[#][0]	854
[B]	1101010110 B

NOTES...

NOTE 1: Admittedly this method of resetting a flag is not standard programming procedure on the HP-41C. It does correspond to the method of clearing the out of range flag on the HP-16C, however.

Space for your own, personal NOTES:

Chapter 6.

LOGICAL OPERATORS IN THE 16-E PROGRAM

Earlier, you had a chance to learn something about the logical operators in the Advantage Module. Now you will see these same functions in the context of the 16-E program.

There is only one rule to learn when using the AND, OR, and XOR functions with the 16-E program.

RULE: Whenever you execute AND, OR, or XOR, you will also need to press the [D] key as the last key in the sequence.

Why this rule? Well, the AND, OR, and XOR functions in the 16-E program are the very same ones in the Advantage Module.

Because these functions come directly from the Advantage Module, they work on all 32 bits of the values in the X and Y registers. In order to limit their effect to the number of bits in the current word size, it is necessary to use the [D] key to trim the results of these functions so that the results will be kept in bounds. (See NOTE 1.)

On the other hand, the NOT operator does not require this additional keystroke. When you use NOT to find the complement of a number, the 16-E program will automatically trim the result to fit into the current word size.

You may remember, when we talked about the NOT function in the Advantage Module, I mentioned that the N0T function in the 16-E program was slightly different from the full 32 bit NOT operator in the Advantage Module.

For one thing, the two functions are spelled differently. The NOT function in the Advantage Module is spelled "N-Oh-T", while the N0T function in the 16-E program is spelled "N-zero-T". The reason for the difference in spellings is so you will be able to assign the 16-E's N0T to a key, and still be able to assign the Advantage Module's NOT function to a separate key, if you so desire.

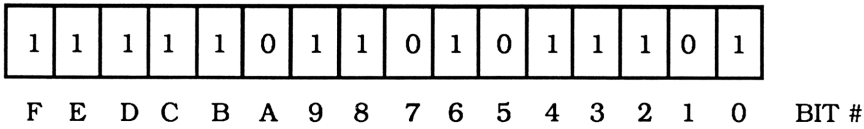
The major difference between the two functions is that the Advantage Module's NOT will invert ALL 32 bits in a number in the X-Register. The 16-E's N0T function, on the other hand will, in effect, invert only those bits allowed by the current word size.

If you look at lines 422 through 425 of the 16-E program, you will see that the N0T routine uses the Advantage Module's NOT function. It also uses the AND function to "trim off" the excess bits that go beyond the bounds of the current word size.

Both the NOT and the N0T functions do the same thing: they change 1's to 0's and 0's to 1's. This operation is called "complementing" or "inverting."

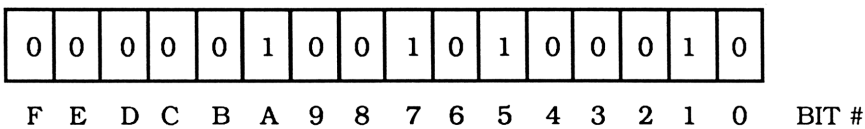
For example, if you are working with a word size of 16 bits and 2's complement format, and you keyed in the decimal number 64349 and pressed the [#][-] keys, the display would show the result 1186. Here is what happens from a binary point of view...

Here is what 64349(d) looks like in binary:



In 2's complement format, this same unsigned number would be -1187(d).

Now, when you press the [#][-] keys, the 16-E program will invert the values of all the bits in the above number to give a decimal value of 1186(d) or



On the other hand, if you were to press the [USER] button to switch off USER mode, and then keyed in 64349(d) and then [XEQ]"NOT", you would see the number 4294902946(d), which is the full 32 bit complement of 64349(d) in unsigned format.

Here are some problems that will let you gain experience in using the 16-E's various logic functions.

Problems With Solutions...

For this first example, set WSIZ = 16 and 2's complements format.

Suppose we want to verify DeMorgan's theorem for a couple of arbitrary decimal numbers. (DeMorgan's theorem, in symbolic logic, states that NOT(p AND q) is equivalent to (NOT p) OR (NOT q), where p and q

are any two "atomic statements".)

A computer engineer would describe one of DeMorgan's Theorems by saying something like: " The output from a two input NAND-GATE would be the same as if you inverted the same inputs and fed the two signals into an OR-GATE."

This is a very handy theorem to know about when you are rummaging through your collection of parts, trying to find an integrated circuit chip with OR gates in it. All you can find is a bunch of NAND-GATE circuits. DeMorgan might just save you a trip to Radio Shack.

Since we are working with just two variables (p and q) in this example, we can set up a truth table like the one below.

p q	p AND q	NOT(p AND q)	NOT p	NOT q	(NOT p) OR (NOT q)
0 0	0	1	1	1	1
0 1	0	1	1	0	1
1 0	0	1	0	1	1
1 1	1	0	0	0	0

This gives us the very general proof of one of DeMorgan's theorems.

We begin with the two columns on the right, labelled "p" and "q". These contain all the possible combinations of 0 and 1. Then we take a look at the result of ANDing these binary digits. The result is shown in the next column. We then NOT the results to get the binary digits in the next column.

We back up and NOT (take the complement of) all the p's and q's. The results are shown in the fourth and fifth columns. Finally, we OR the results of these two

columns to get the results in the last column. Then we can just "eyeball" the last column and the third column and, taa-daa, the results are the same! Indeed, DeMorgan is right!

Now, let's see how easy it is to show this on the 41C. The only trick to doing this is to take the columns of values of p and q and lay them on their side....like so:

p 0011
q 0101

We can imagine that we are working with two binary numbers with a word size of 4. So, here is the procedure to follow to prove DeMorgan's theorem using the 16-E program.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 4 [R/S]	WSIZ = 4	Set word size to 4 bits.
[#][3]		Set unsigned format.
[#][B]	_B	
0011	0011 B	Key in the value for p.
[#][B]	_B	
0101	0101 B	Key in the value for q.
[#][*]	1	The decimal result of ANDing 0011 (b) and 0101 (b)...and, of course, you could imagine that this is equivalent to 0001 (b).

[#][–]	14		NOT(p AND q) in decimal.
[#][B]		_B	
0011		0011 B	The value for p again.
[#][–]	12		NOT p in decimal
[#][B]		_B	
0101		0101 B	The value for q.
[#][–]	10		NOT q in decimal
[#][+]	14		OR of the two values. Now you COULD press [X<>Y] and see that the contents of X and Y are the same. However you may be more logical and reason that, if the numbers are identical, then XOR will be 0 (zero).
[#][/]	0		XEQ "XOR" Therefore equivalent. Q.E.D.

2. You are showing your friend, a student of logic, your newly acquired 16-E. Your friend notes that the calculator can perform AND, OR, NOT and XOR operations. He or she wonders if it has a "material implication" operator. Material implication is the operator in logic that performs

"IF...THEN..." statements. You respond that it should not be too difficult to design such an operator. You reason that, since "If p, then q" is equivalent to "NOT(p) OR q", you should be able to emulate material implication with the existing logic functions of the 16-E. To to test your design, you have the standard truth-table definition of the "If...then..." operator

p	q	If p then q
0	0	1
0	1	1
1	0	0
1	1	1

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 4 [R/S] [#][3]	WSIZ =4	Set WSIZ = 4 and UNS format.
[#][B] 0101	0101_ B	This is the "q" column of the truth table but in row format
[#][B] 0011	0011_ B	and the "p" column of the truth table but in row format.
[#][–]	12	NOT(p)
[#][+]	13	OR q
[B]	1101 B	Which is the expected result. (Stand this result

upright and compare it with the right column in the truth table on page 115.)

Question: How large a "truth table" could you work with on the 16-E. That is, how many variables could the calculator handle?

Answer: up to 5 variables. How's that? Well, here's the reasoning behind this conclusion.

You may have noticed that, when we worked problems involving just two variables, we needed to use a truth table for two values. The truth table turned out to be four rows deep. Then we laid the truth table on its side and we had two numbers that were 4 bits wide. Four rows of two values became two rows of four values. For example, the truth table for AND with two values started out as

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

and wound up becoming

0101
0011

Similarly, if we wanted to design a truth table for three values, we would have to have one that was 8 rows deep...

X	Y	Z	AND
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

We could flip this truth table on its side and have

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

That is we would have three numbers, each of which is 8 bits wide.

Now, for four values, we would wind up with 4 numbers that would be 16 bits wide. For 5 values we would need 5 numbers that would be 32 bits wide. That's as wide a number as the HP-41 can handle at one time.

(Actually, you could use the 16-E program to work with truth tables of any length, just as long as you were willing to rework the same logic problem FOR EACH GROUP OF 32 BITS in the problem. Thus, for example, if you had a logic problem involving 6 variables, you would have 6 numbers of 64 bits each. You would have to solve the same logic problem twice: once for the first 32 rows of the truth table and again for the second 32 rows.)

As an example of a logic problem involving five variables, consider the following problem from symbolic logic.

Premise 1: p AND q
Premise 2: r implies s implies t

Conclusion: q AND s implies t

The question is "is the conjunction of the premises equivalent to the conclusion?" That is "is the argument form Valid?"

Ooops, that's the logician in me speaking. What the problem means in electronic terms is this:

If I had two digital circuits that emulated the behavior of the two premises above, and I were to feed the outputs from each of these circuits into an AND-GATE, would the ouput from this AND-GATE be identical to the output from a digital circuit that emulated the conclusion above?

For me the easiest way to set up the problem is to write a logical equation.

The first term in the equation is easy: p AND q.

The second term is trickier. I have to make use of the equivalence that we looked at before: "If X then Y is equivalent to NOT(X) OR Y."

So I can write the second terms as :

" If r then if s then t " as "If r then (NOT(s) OR t)" which is the same as NOT(r) OR (NOT(s) OR t). This is the same as NOT(r AND s) OR t, by DeMorgan's Theorem.

The conclusion can be written as NOT(q AND s) OR t .

So the logical equation could be written as...

((p AND q) AND (NOT(r AND s) OR t)) XOR (NOT(q AND s) OR t).

Now, any good RPNer knows that there are no such things as parentheses, right? So let's rewrite this logical equation in true RPN style:

p q AND r s AND NOT t OR AND q s AND NOT t OR XOR

If the results of the XOR operation is 0, then you would know that the "argument is valid" or the two circuits are identical in behavior.

You continue to solve the problem by generating the following truth table:

p	q	r	s	t	
					Now convert the patterns of 1's and 0's to hexadecimal numbers so that the patterns
0	0	0	0	0	of 32 bits may be entered into
0	0	0	0	1	the16-E program.
0	0	0	1	0	
0	0	0	1	1	t = 55555555 Hex
					s = 33333333 Hex
					r = 0F0F0F0F Hex
					q = 00FF00FF Hex
0	0	1	0	0	p = 0000FFFF Hex
0	0	1	0	1	
0	0	1	1	0	
0	0	1	1	1	Then the argument may be analyzed on the16-E calculator

with the following keystrokes.

0	1	0	0	0			
0	1	0	0	1			
0	1	0	1	0	Set WSIZ =32 and 2's		
0	1	0	1	1	complement format.		
					KEYSTROKES	DISPLAY	COMMENT
0	1	1	0	0			
0	1	1	0	1	[#][D] 32 [R/S]	WSIZ=32	
0	1	1	1	0	[#][2]		
0	1	1	1	1			
					[#][E] FFFF	FFFF H	p
1	0	0	0	0	[#][E] FF00FF	FF00FF H	q
1	0	0	0	1	[#][*]	255 AND	
1	0	0	1	0			
1	0	0	1	1	[#][E] F0F0F0F	F0F0F0F H	r
					[#][E] 33333333	33333333 H	s
1	0	1	0	0	[#][*]	50529027	AND
1	0	1	0	1	[#][-]	-50529028	NOT
1	0	1	1	0	[#][E] 55555555	55555555 H	t
1	0	0	1	1	[#][+]	4261281277	OR
					[#][*]	253 AND	
1	1	0	0	0			
1	1	0	0	1	[#][E] FF00FF	FF00FF H	q
1	1	0	1	0	[#][E] 33333333	33333333 H	s
1	1	0	1	1	[#][*]	3342387 AND	
					[#][-]	-3342388	NOT
1	1	1	0	0	[#][E] 55555555	55555555 H	t
1	1	1	0	1	[#][+]	4292739037	OR
1	1	1	1	0			
1	1	1	1	1	[#][/]	4292738848	XOR

Since the final result is not zero, the "argument is invalid," or the two circuits are not behaviorally identical.

3. Harriet "The Pro" Grammer is working on a code for her IBM-PC. She wants to input 4 octal numbers (with a maximum of 16 bits each) and get out a hexadecimal number. Her scheme is

   (p AND q) XOR (r OR s).

   She uses the numbers p=67271(o), q=73333(o), r=44505(o), s=106120(o). The answer that comes up on the video dislay is BAD1. Is the computer behaving properly?

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	Set WSIZ = 16 and 2's compl.
[#][2]		
[#][C] 67271	67271 O	
[#][C] 73333	73333 O	
[#][*]	26265	AND
[#][C] 44505	44505 O	
[#][C] 106120	106120 O	
[#][+]	52565	OR
[#][/]	43980	XOR
[E]	ABCC H	HEXVIEW. The PC has not been programmed correctly.

4. For this problem, set WSIZ = 32

   What is D574D09F XOR B4966427 in binary ?

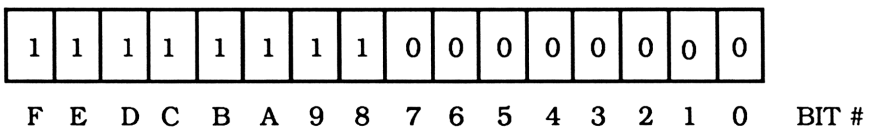
KEYSTROKES	DISPLAY	COMMENTS
[#][D] 32 [R/S]	WSIZ = 32	
[#][E] D574D09F	D574D09F H	
[#][E] B4966427	B4966427 H	
[#][/]	1642247352	XOR
[E]	61E2B4B8 H	Convert to Hex.
[#][E] 61	61 H	Key in the total hex number 2 digits at a time and convert them to binary.
[Enter]	97	Needed to terminate input.
[B]	1100001 B	
[#][E] E2	E2 H	This is one way to view the binary equivalents of the full 32 bit hexa-decimal number.
[Enter]	226	
[B]	11100010 B	
[#][E] B4	B4 H	
[Enter]	180	It's not very elegant.
[B]	10110100 B	We will develop a program that does the job better.
[#][E] B8	B8 H	
[Enter]	184	
[B]	10111000 B	

Creating Masks

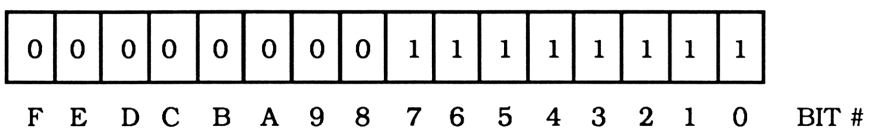
Masking is a technique for isolating portions of a binary number. The MASK operators in the 16-E create a string of ones on the right or left side of a binary word. You need to place a number in the X-register to determine the size of this string of ones.

The MSKR operator will justify the ones on the right of the word while the MSKL function will justify the pattern of ones on the left of the word.

For example, with a word size of 16 bits, if you execute the function 8 "MSKL" (8 [#][A]) the resulting pattern of 0s and 1s would look like this:



On the other hand, the keystrokes 8 [A] (8 "MSKR") would generate the following bit-pattern:



Once you have created a mask, you may use this string of ones to "filter through" selected portions of other binary numbers.

For example, suppose you are using a word size of 16 bits and unsigned format. You have entered the hexadecimal number EC96 and you want to separate the hexadecimal number into two 8 bit numbers.

Here is the technique that will start the process.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ =16	
[#][3]		Unsigned format
[#][E] EC96	EC96_ H	
[Enter]	60566	1110 1100 1001 0110 Bin
STO 12		Set this aside for future use.
8 [A]	255	0000 0000 1111 1111 Mask on right 8 bits.
[#][*]	150	AND
[B]	10010110 B	0000 0000 1001 0110 The right half is "filtered" through the mask.
RCL 12	60566	1110 1100 1001 0110
8 [#][A]	65280	1111 1111 0000 0000 XEQ "MSKL"
[#][*]	60416	AND
[E]	EC00 H	1110 1100 0000 0000

The only thing left to do is to shift or rotate the last binary number generated above 8 bits to the right to create the binary number 1110 1100. We will learn how to do this in the next section of the book.

You can, of course create masks of any size you wish as long as they fit within the bounds of the word size in effect.

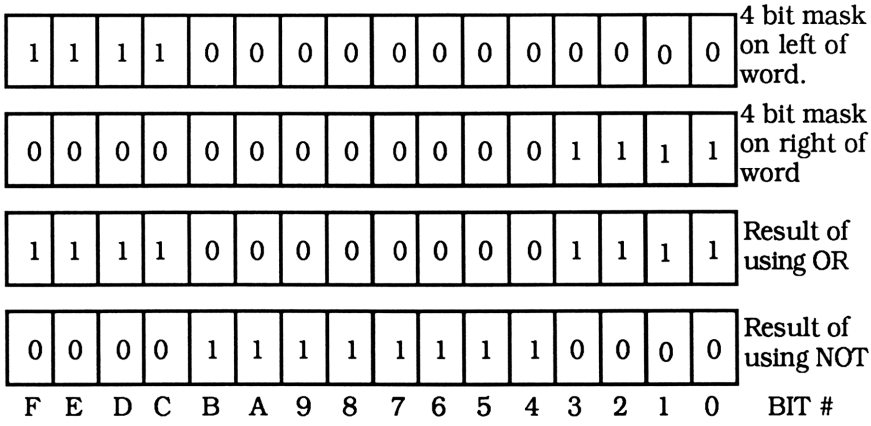
If you specify a mask that is larger than the current word size, the 16-E will accept this without question. The effect will be that the computer will create a mask of all one's. This mask will be of the same size as the current word size.

Thus for a word size of 16 bits, you could generate masks of 13 bits on the left, or 5 bits on the right, or 1 bit on the left.

If you want to create a mask that will "let through" the middle 8 bits of a 16 bit number, there are a couple of ways that you can do this on the 16-E.

The preferred method for isolating portions of such a number is to create a mask of 4 bits on the left (F000 Hex). Then create a mask of 4 bits on the right (000F Hex). Use the OR operator to combine these two masks into one (F00F Hex). Complement this mask with the NOT operator to create the mask 0FF0 Hex. Use this mask with your 16 bit number and the AND function to filter out the middle 8 bits.

Here is a sketch of what would happen:



This final result could then be ANDed with any other number to "filter through" bits B through 4 of the other number.

One Extended Example...

Use the masking functions along with the logical operators to break down a full 32 bit number into a series of 8 bit numbers.

This will involve a four step process since 32 bits divided by 8 bits equals 4.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 32 [R/S]	WSIZ = 32	
[#][3]		Set Unsigned format.
[#][E] FEDCBA98	FEDCBA98 H	
STO 12	4275878552	Store the number for later use.
8 [#][A]	4278190080	Create an 8 bit

		mask on left.
[E]	FF000000 H	
[#][*]	4261412864	AND
[E]	FE000000 H	The left 8 bits are filtered through.
RCL 12	4275878552	
8 [#][A]	4278190080	Create an 8 bit mask on left.
[E]	FF000000 H	
16 [A]	65535	Create a 16 bit mask on right.
[#][+]	4278255615	OR
[E]	FF00FFFF H	HEXVIEW
[#][–]	16711680	NOT
[E]	FF0000 H	Leading zeroes suppressed.
[#][*]	14417920	AND
[E]	DC0000 H	Next 8 bits are filtered out.
RCL 12	4275878552	
8 [A]	255	Create an 8 bit mask on right.
[E]	FF H	Leading zeroes suppressed.
16 [#][A]	4294901760	Create a 16 bit mask on left.
[#][+]	4294902015	OR
[E]	FFFF00FF H	HEXVIEW
[#][–]	65280	NOT
[E]	FF00 H	Leading zeroes suppressed.
[#][*]	47616	AND
[E]	BA00 H	Next 8 bits are filtered out
RCL 12	4275878552	

8 [A]	255		Create an 8 bit mask on right.
[E]		FF H	
[#][*]	152		AND
[E]		98 H	The right 8 bits are filtered through.

REVIEW

In this section of the course, you learned how to use the the logic functions in the 16-E program. You saw several examples that incorporated the AND, OR, XOR and NOT functions.

Then you learned one practical application of the use of these logic functions: masking. Masking is a strategy you will use again and again in computer science and engineering.

For now, try the following set of problems to see if you have mastered the techniques of ANDing, ORing, NOTing and masking with the 16-E program.

POP QUIZ

Use WSIZ = 16 and 2's complement format.

1. What is 19(h) AND 1A(h) in hexadecimal format?

2. What is 233(o) OR 362(o) in octal format?

3. What is (111(b) XOR 37(h)) AND NOT(10(d)) in binary?

4. What is (343(o) OR 371(o)) / NOT(1011(b)) in hexadecimal?

5. Does NOT(249(d) OR 546(d)) = NOT(249(d)) AND NOT(546(d)) ? This is an alternative version of DeMorgan's theorem.

6. Using WSIZ = 32 and UNS format, and the hexadecimal number FC998E22, separate out the Most Significant Bit and then the 15 bits on the right.

7. With the same WSIZ, and the hexadecimal number, ABCDEF12, filter out the bits numbered 29 through 21.

   NOTE: the computer numbers bits starting with the least significant bit on the right as bit #0. With a 32 bit word size, the MSB is bit #31.

ANSWERS TO THE POP QUIZ

1.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		2's compl.
[#][E] 19	19 H	
[#][E] 1A	1A H	
[#][*]	24	
[E]	18 H	

2.

KEYSTROKES	DISPLAY	COMMENTS
[#][C] 233	233 O	
[#][C] 362	362 O	
[#][+]	251	
[C]	373 O	

3.

KEYSTROKES	DISPLAY	COMMENTS
[#][B] 111	111 B	
[#][E] 37	37 H	
[#][/] 48		
10 10		
[#][–] -11		
[#][*] 48		
[B]	110000 B	

4.

KEYSTROKES	DISPLAY	COMMENTS
[#][C] 343	343 O	
[#][C] 371	371 O	
[#][+]	251	
[#][B] 1011	1011 B	
[#][–]	–12	
[/]	–20	
[E]	FFEC H	with a remainder.

5.

KEYSTROKES	DISPLAY	COMMENTS
249 [ENTER]	249	
546	546	
[#][+]	763	
[#][–]	–764	
249	249	
[#][–]	–250	
546	546	
[#][–]	–547	
[#][*]	64772	
[#][/]	0	Equivalence holds

6.

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 32 [R/S]	WSIZ = 32	
[#][3]	4294967295	Unsigned format
[#][E] FC998E22	FC998E22 H	
[ENTER][ENTER]	4237921826	Push onto stack
1 [#][A]	2147483648	Create mask of 1 on left.
[#][*]	2147483648	
[E]	80000000 H	
[RDN]	4237921826	
15 [A]	32767	Mask on right
[#][*]	3618	
[E]	E22 H	

7.

KEYSTROKES	DISPLAY	COMMENTS
[#][E]ABCDEF12	ABCDEF12 H	
[ENTER]	2882400018	
2 [#][A]	3221225472	Mask of 2 bits on left.
20 [A]	1048575	Mask of 20 bits on right.
[#][+]	3222274047	Combine two masks.
[#][–]	1072693248	Invert
[#][*]	734003200	AND
[E]	2BC00000 H	Hex equivalent.

NOTE 1: The real reason for this rule is to save additional bytes of code in the 16-E program. Nevertheless, if you are not too concerned with adding code to the already code-heavy 16-E program, you might try adding the following.

Begin at line 413 and insert these lines of code:

```
413 LBL"ANd"
414 AND
415 GTO 16
416 LBL"0R"
417 OR
418 GTO 16
419 LBL"X0R"
420 XOR
421 GTO 16
```

Then assign "ANd" (note the small "d") to key -71; assign "0R" to key -61 and assign "X0R" to key -81. (note the use of zero, 0 rather than capital O in the labels. This is a clever little ruse to help you keep your functions straight. (I have tried this modification to the program and found that it has very little use, other than slowing down the operation of the logic functions. Most often, when using the calculator, you will key in numbers that are within bounds to begin with. And, the logic functions will not cause the numbers to go out of bounds as might the arithmetic functions.)

Chapter 7.

SHIFTING, ROTATING, AND JUSTIFYING NUMBERS

The 16-E calculator contains several functions that will let you move binary digits to the right or left. There are still other functions that will let you rotate bits clockwise or counterclockwise within the bounds of the current word size.

The Advantage ROM has a similar function: ROTXY. The ROTXY operator is super fast. It will rotate the binary digits in a 32 bit word in less time than it takes to press the ENTER key. There is only one problem with this ROTXY operator: It only works with 32 bit words.

The 16-E calculator implements the shifting and rotating functions in standard RPN code. These bit-moving functions will work with any word size. They do sacrifice speed for this flexibility, however.

Shifting Left or Right

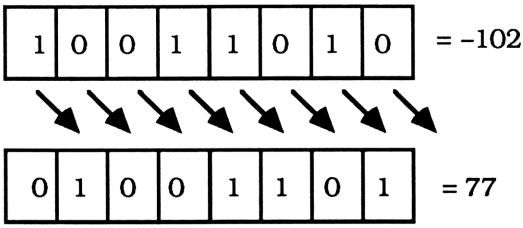
The first set of moving functions that we will look at are those that shift numbers to the right or left.

In the 16-E the functions SR (Shift Right), SL (Shift Left), and ASR (Arithmetic Shift Right) all serve to

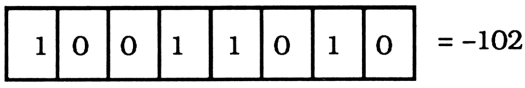
shift a binary number ONE bit to the left or right. The key assignments for these functions are the [A], [shift][A] and [shift][F] keys respectively.

As an example of a shift to the right, set up the 16-E calculator with a word size of 8 bits (2's compl. format). (Press the keys [#][D] 8 [R/S] and [#][2].) Then enter the binary number 10011010 (-102 dec). Use the [#][B] keys to initiate binary input.

Now press the [H] key ("SR") and see 77 in the display. Then press [B] and see 1001101 B. There are only seven bits shown. You will have to imagine that the Most Significant Bit is a zero. The 16-E has shifted all the binary digits one bit to the left. The empty MSB takes the default value of 0.

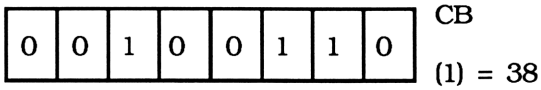


The reverse operation is XEQ "SL" ([#][H]). This operation will shift the binary number to the left by one bit and the calculator will set the vacated Least Significant Bit to zero.

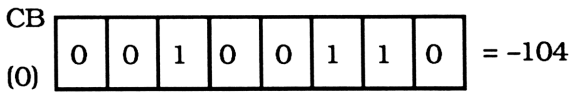


Now try two SR's and notice that, on the second SR, the 0 flag annunciator turns on. When the 16-E shifts a number, and the LSB goes "out of bounds," the 16-E saves the value, temporarily, in the "carry bit."

The decimal number should now be 38 or 00100110 Bin.



Press [#][H] twice. Notice that the zero annunciator disappears from the display. In this case the calculator shifts a zero into the carry bit. You should visualize the carry bit as positioned on the left of the binary number. When the carry bit contains a zero, the annunciator shuts off.

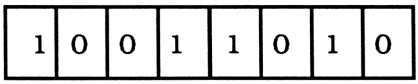


The decimal number should now be -104 or 10011000 Bin.

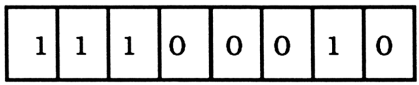
Press [#][H] once more and the decimal number will be 48 (=0011000 B) and the carry bit annunciator will be ON.

Most texts refer to the SL and SR operations as logical shifts. Opposed to this is the ASR (Arithmetic Shift Right) operator. The difference between a logical shift right and an arithmetic shift right is that the latter PRESERVES the sign of the original number as the number shifts to the right.

To see this sign preservation, key in the decimal number 102. Then press [CHS] and view the result in binary, 10011010 B.



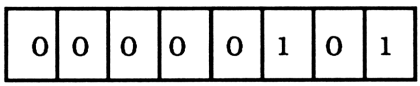
Now press [#][K] (XEQ "ASR") and the result will be -51(d) or 11001101(b). Press [#][K] once more and the result will be -26(d) or 11100110(b) and the carry bit flag will be ON.

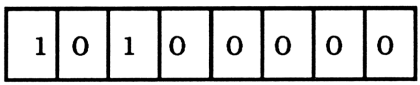


Justifying

Right justifying a number means that the binary equivalent of a number moves to the right until a 1 winds up in the LSB of the given word. Thus, if you begin with the binary number, 00101000 and press [#][G], you will wind up with 00000101 B.

Similarly, if you left justify 0000 0101 B, you will wind up with 1010 0000 Bin. The keys to press are [#][F].





Both of these operations preserve the original number in LastX.

These justifying functions may tax your patience a little. The worst case (right justifying -2147483648(d) with a word size of 32 bits) takes an horrendous 17 seconds. Gratefully, I have not had to use these functions too often. I hope you don't have to either.

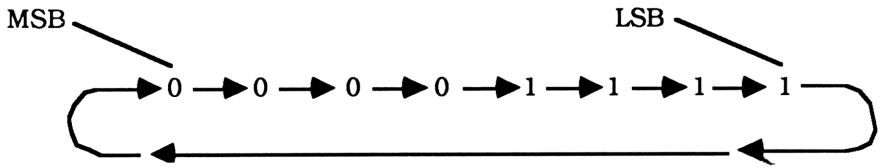
Rotating Bits

The 16-E can perform four different bit-rotating functions. These functions are, perhaps, more of interest to computer engineers and digital circuit designers than they are to programmers.

These four different functions are...

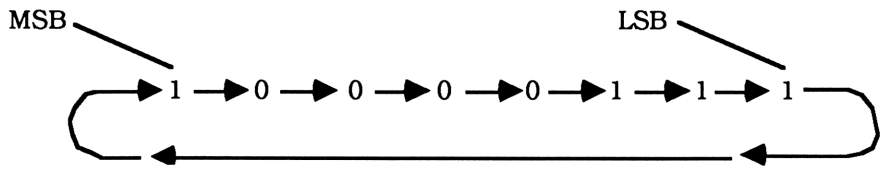
LABEL	ASSND KEY	NAME
RRN	[J]	Rotate Right by N bits.
RLN	[I]	Rotate Left by N bits.
RRCN	[#][J]	Rotate Right through the Carry bit by N.
RLCN	[#][I]	Rotate Left through the Carry Bit by N.

To visualize what Rotating Right or Left looks like, imagine that you are using an 8 bit word with the bits arranged along a line that wraps around. The MSB and the LSB would be next to each other.



To emulate this in your calculator, press the [#][D] 8 [R/S] keys, along with [f][3] to set Unsigned format. Then press [#][B] and 1111. Then press [ENTER] to see 15 in the LCD.

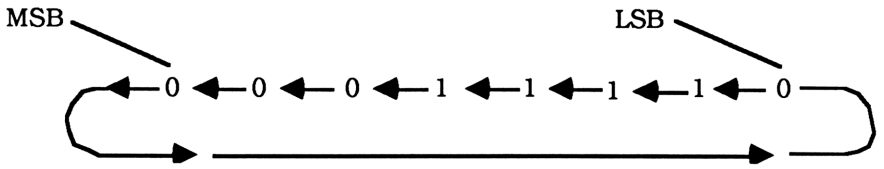
When you press the 1 [J] keys, the pattern will become



In the picture above, Rotating Right amounts to a right-hand rotation of the bits along the line, with the last bit on the line being shifted around and into the first position. In the LCD you will see 135, and when you press [B] you will see 10000111 B with the 0 annunciator on.

The LSB and MSB in the diagrams stand for "Least Significant Bit" and "Most Significant Bit," with the lower line representing the boundary between the most significant bit and least significant bit. If in the process of rotating, a 1 crosses this boundary, the 0 annunciator will turn on. If a 0 passes the boundary on the way from the LSB to the MSB then the 0 annunciator is turned off.

Rotating Left is just the reverse process. It amounts to a counter-clockwise rotation. Key in [#][B] 1111 and press the [ENTER] key. Then key in 1 (the number of bits to rotate) and press the [I] key. The number in the LCD will be 30. If you press [B] you will see 11110 B . And in the diagram the pattern would be:

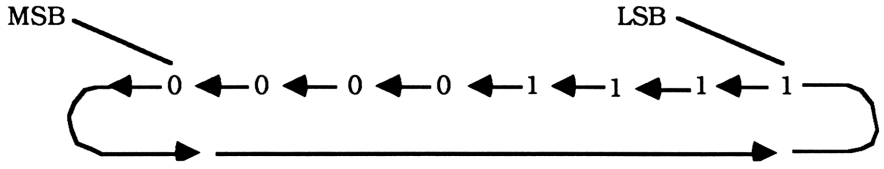


Notice that, in this case, a zero crossed the boundary between the LSB and MSB. Thus the 0 annunciator would be turned off.

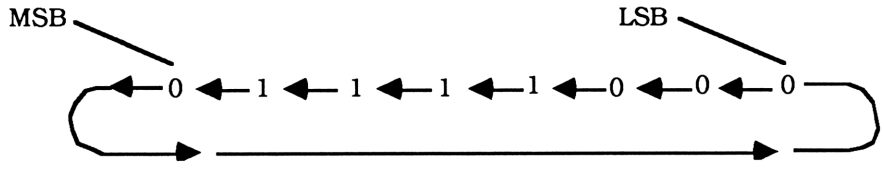
Of course the number of bits by which you want to rotate a number can be more than 1. Let's take a look at what would happen if we rotated 1111(b) to the left by 3 bits.

Begin by keying in [#][B] 1111 [ENTER] and then key in 3 and press the [I] key.

The original arrangement would look like...



After you pressed [I] the arrangement would look like...



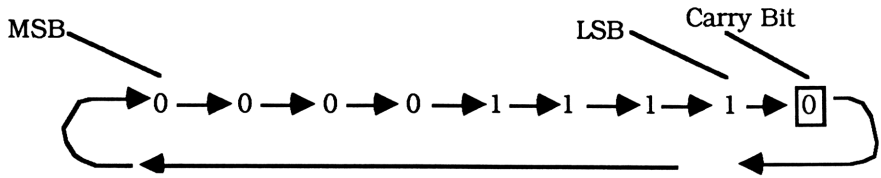
and the LCD would show 120. The 0 annunciator should be off, since the last bit to pass the boundary is

a 0. (If the last bit to pass the boundary is a 1, then the 0 annunciator will be turned on.)

Rotating Through The Carry Bit

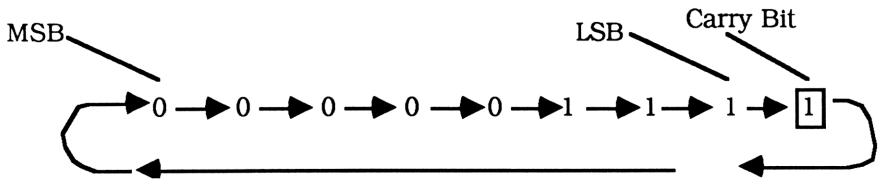
There is another kind of rotation. This kind is called "Rotation Through the Carry Bit."

We can use the same diagram as above, BUT WITH THIS DIFFERENCE: the boundary between the LSB and MSB becomes a bit-bucket (the carry bit). The bit bucket actually becomes an extra bit in the rotation process. Thus, even though we have set the word size to 8 for this series of examples, there are actually nine bits as far as rotation goes.



To see how this type of rotation works, press the keys 0 [+] to insure that the 0 annunciator is off. Key in [#][B] 1111. Then press [ENTER] [1] [#][J] (Rotate Right through the Carry Bit by 1). The LCD will show the result 7 and the 0 annunciator will be turned on. Press the [B] key and you will see 111 B. One of the "1" digits is missing (we had 4 of them to begin with) and, although it's not shown in the HP-41 display, there is an extra 0 in the binary number. What happened?

To answer this question take a look at the diagram after you pressed the [#][J] keys.



In the process of rotating THROUGH THE CARRY BIT, the 0 that was in the carry bit is moved into the MSB and the original LSB is moved into the carry bit. The 1 in the carry bit turns on the 0 annunciator. The LCD shows only the 8 bits in the byte. The binary digit in the carry bit bucket is shown only by the status of the 0 annunciator.

Of course Rotation to the Left through the Carry Bit by N. ([#][I]) is just the reverse of RRCN. The MSB is moved into the carry bit bucket, setting or clearing the 0 annunciator and the contents of the carry bit are moved into the LSB.

As before, you can rotate numbers through the carry bit by any number of bits from 0 up to the number of bits in the current word size. (If you use a number larger than this, the HP-41 will just keep on rotating--going around in circles.)

Please be patient with these functions on the 16-E: they are slow. For example, the worst case of rotating a 32 bit word thirty-two bits to the right takes approximately 23 seconds on the 16-E. Admittedly this is slow! About the only thing in favor of these procedures in the 16-E program is that they work and they do accomodate various word sizes.

If your work demands that you perform dozens of bit manipulations every day and you need a faster response time, then you are a prime candidate for the HP-16C calculator.

Problems With Solutions...

1. What is the octal result of shifting 96 dec to the left by 6 bits? (Use WSIZ = 16: 2's compl.)

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		
96 [#][H]	192	XEQ "SL"
[#][H]	384	
[#][H]	768	
[#][H]	1536	
[#][H]	3072	
[#][H]	6144	
[C]	14000 O	OCTVIEW

2. What is the decimal result of rotating 219 dec by 15 places to the right, both normally and then through the carry bit? Use a WSIZ of 16: 2's compl.

KEYSTROKES	DISPLAY	COMMENTS
219 [ENTER]	219_	
15 [J]	438	XEQ "RRN" (~10sec.)
219 [ENTER]	219_	
15 [#][J]	876	XEQ "RRCN

3. What is the result of left and right justifying 876 dec in a 16 bit word? Show the results in hexadecimal.

KEYSTROKES	DISPLAY	COMMENTS
876 [#][G]	219	XEQ "RJY"
[E]	DB H	
[#][F]	-9472	XEQ "LJY"
[E]	DB00 H	

4. What is the binary representation of 11110000 rotated left by 5 ? (Use WSIZ = 16: 2's compl.)

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 16 [R/S]	WSIZ = 16	
[#][2]		2's complement format
[#][B] 11110000	11110000 B	
[ENTER]	240	
5 [I]	7680	
[B]	7680	Out of range for BINVIEW.
[E]	1E00 H	Use HEXVIEW.
[#][E] 1E	1E H	HEXIN
[ALPHA][ALPHA]	30	Another way to get out of HEXIN, without lifting the stack.
[B]	11110 B	So 0001 1110 0000 0000 is the full 16 bit binary number.

5. What is the hex result of rotating 0001 1010 1110 1001 Bin to the left by 1 bit? ANS: 35D2 H

   NOTE: You can't key a 16 bit binary number into the HP-41. It won't handle more than 10 binary digits as input. What shall we do? What shall we do?

   No problem! Just convert the binary digits to hex digits, of which there would be 4 in the above number, and use them as the equivalent of binary digits.

KEYSTROKES	DISPLAY	COMMENTS
[#][B] 11010	11010 B	Convert 8 bits at a time.
[E]	1A H	
[#][B]	11101001 B	
[E]	E9 H	
[#][E]	1AE9 H	Key in full 16 bits
[ENTER]	6889	Unsigned dec. equiv.
1	1	Rotation factor
[I]	13778	
[E]	35D2 H	

6. Suppose you only had a calculator with a fixed word size of 8 bits. How would you go about using the calculator to rotate the above number to the left?

ANS:

a. Key in left half of the binary number and XEQ "SL" [#][D] 8 [R/S] and [#][2] to set word size to 8 bits. Then press [#][E] 1A [#][H] (see 52(d)).

b. LastX to recover the original number. (See 26)

c. Key in right half of number and then rotate it 1 bit to the left through the carry bit.
Thus...[#][E] E9 [ENTER] 1 [#][I] (see -46).

d. Swap, X<>Y and rotate this half of the number to the left by 1 through the carry bit.
OK...[x<>y] 1 [#][I] (see 53).

e. The left half of the rotated number is in the X-register and the right half is in the Y register. To confirm this, press [E] (see 35 H); press [x<>y] and [E] (see D2 H) AND THERE YOU ARE!

This same technique can be used to rotate a 64 bit number on a calculator with a maximum of 32 bit words.

Testing, Setting and Clearing Bits

On the 16-E calculator you can test any of the 32 bits available for a given word. The way to do this is to have the number whose bits you want to test in the X-register. Then key in the number of the bit you want to test. This number can be any number from 0 to 31. Then XEQ "B?" or simply press [#][6]. If the bit is set the display will say YES, otherwise it will say NO.

If you use a bit number greater than 31, you will get the message DATA ERROR.

To clear or set an individual bit, just have a number in the X-register and then key in the number of the bit you want to set or clear and either XEQ "SB" or XEQ "CB" respectively. The assigned keys for SB and CB

are [#][4] and [#][5].

If you try to set or clear a bit that is greater than the number of bits in the current word, you will get the message "ERR: WSIZ = xx" and the program will halt. (xx will be the current word size.)

An error of this type will destroy the contents of the stack. If there is a need to recover from this type of error, then [XEQ] 15 from the keyboard before continuing with any computations. (That is, just press the keys [XEQ][1][5].) This will restore the stack to its pre-error arrangement.

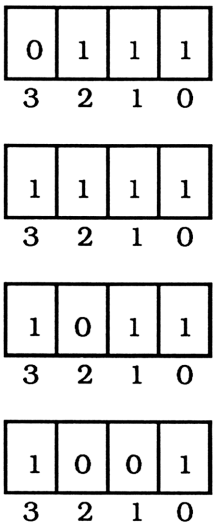
A Problem With Solution...

For example, set WSIZ = 4 and 2's complement format and key in the number 111 Bin. Then use the set and clear bit function to change the sign of this number. That is, convert 7 to –7 in 2's complement format "the hard way".

KEYSTROKES	DISPLAY	COMMENTS
[#][D] 4 [R/S] [#][2]	WSIZ = 4	
7 [B]	111 B	
3 [#][4]	–1	SB #3
2 [#][5]	–5	CB #2
1 [#][5]	–7	CB #1
[B]	1001 B	Remember that 2's compl. is defined as complementing all bits in the number and then adding one bit. In

this case we could have cleared bit #0 and then added 1, but this is not necessary.

Here is a bit diagram of the above problem.



REVIEW

In this section of the course you discovered even more of the ins and outs of the 16-E program. In particular you discovered how to shift and rotate the bits in any number.

We noted that there is a difference between a logical shift and an arithmetic shift. The Arithmetic Shift preserves the sign of the number during the shifting process.

You also saw that there are two kinds of rotation possible in the 16-E program: normal rotation and rotation through the carry bit.

Finally you found out how easy it is to test, set and clear the individual bits in a number.

Admittedly, much of this material on shifting and rotating will be confusing, especially if this is your first encounter with such strange goings-on. This confusion will abate after you gain some practical experience in using these ideas. Pushing buttons on the calculator can help only so far. You'll begin to appreciate these ideas after you have to shift bits or rotate them to change the way your printer works or to convert ASCII code to "binary coded decimal", but that's a story for another book.

For now, let's tackle another wonderful...

POP QUIZ!

1. With WSIZ = 8 and 2's complement format, what is the effect of setting the sign bit in the number 127(d)?

2. With the same configuration, what is the effect of pressing [#][K] ([XEQ] "ASR") when you start with -1(d) in the display?

3. How does clearing the sign bit (MSB) in the above problem effect subsequent executions of ASR?

ANSWERS TO THE POP QUIZ

1. KEYSTROKES	DISPLAY	COMMENTS
[#][D] 8 [R/S]	WSIZ = 8	
[#][2]		
127 [ENTER]	127	
7 [#][4]	-1	Setting the sign bit 0111 1111(b) = 127(d) to 1111 1111(b) or -1(d) in 2's compl.
2. [#][K]	-1	An arithmetic shift right does, indeed, shift all the bits to the right. In a logical shift the MSB would be 0 but the ASR function overrides this by resetting the MSB to 1. Note that the 0 flag is ON since there was a 1 shifted into the carry bit.
3. 7 [#][5]	127	Equivalent to 01111111(b).
[#][K]	63	
[#][K]	31	
[#][K]	15	
[#][K]	7	
[#][K]	3	
[#][K]	1	
[#][K]	0	

Chapter 8.

PROGRAMMING WITH THE 16-E

The 16-E program is useful enough on its own, but it really shines when you use it as the basis for your own programs.

You may use almost all of the routines in the 16-E program in your own programs with the following two exceptions:

1. Since the WS (Word Size) routine does not contain a RTN statement, you will not be able to use this routine in your own programs. If your program should call the WS routine, then you will find yourself stuck in the 16-E program with no easy way to get back to your program. However, for those who really need this capability, help is just a few pages away. Turn to the end of this section and read NOTE 1.

2. The other function that will not work in your program is the "B?" routine from the 16-E program. Instead of this "B?" function, use the BIT? function from the Advantage ROM. BIT? will do what you want it to in a program: act as a conditional. Bit will not flash "YES" or "NO" in the LCD: "B?" will.

Suppressing Intermediate Output With Flag 06

You may have noticed that all of the routines in the 16-E produce visible output. This means that, when you use these routines in your programs, they will show or print any and all intermediate results.

If you want to suppress these intermediate results, start YOUR program with the command SF 06 and end YOUR program with the command CF 06. Flag 06 tells the 16-E program to suppress the viewing of results.

HERE ARE THREE SAMPLE PROGRAMS THAT WILL SHOW YOU WHAT YOU CAN DO WITH THE FUNCTIONS FROM THE 16-E PROGRAM:

PROGRAM 1: LBL "SV"

On most CP/M computers there is a command, "SAVE,"that will let you save a program on disk. The command has the form SAVE n FileName, where n is the number of 256 byte blocks of data you want to save. The data or program begins at location 0100 Hex in the computer. To find n, you must also know the ending location of the program in the computer. Then, to compute the number of 256 byte blocks, all you need do is subtract the beginning location from the ending location of the program or data. Divide this difference by 256 and add 1 to the number of blocks of data if there is a remainder from this division process.

Here is the program that will do this very nicely.

```
01  LBL "SV"
02  SF 06          Suppress intermediate results.
03  XEQ "UNS"      Set the display to unsigned format.
04  256            The beginning location of the
                   program or data ( 100 Hex).
05  X>Y?
06  ENTER^
07  XEQ "-"        Subtract beginning from ending
                   location.
08  LASTX
09  XEQ "/"        Divide by 256.
10  1
11  X<>Y
12  FS? 00         Check if there is a remainder in the
                   division.
13  XEQ "+"        Add 1 to the quotient if there is a
                   remainder.
14  CF 00          Clear flag 00 in any case.
15  CF 06          Reset flag 06.
16  END
```

Running the program:

1. [XEQ]"HEXIN" and key in the ending location of the program

2. [XEQ]"SV" and view the final output, the number of blocks needed for "n" in the CP/M command SAVE n FileName.

For example, set WSIZ=16; key in 950(h) and [XEQ] "SV" to see 9. (See Note 2 for a faster version of this program.)

PROGRAM 2: BIT SUMMATION

On the HP-16C calculator there is a function which finds the number of ones in a given binary number.

Here is a simple program that will do the same thing on the 41C. The program uses the BIT? function from the Advantage ROM. The program assumes that REG 00 contains the current value of word size.

```
01  LBL "BSUM"
02  .O32        Set up an accumulator in Register
                12.
03  STO 12
04  RDN
05  RCL 00      Get the current word size and
                reduce it by 1.
06  DSE X
07  1 E3        Set up a pointer in the X-reg that
08  /           will go from 0 to one less than the
                current word size.

09  LBL 00      LOOP
10  BIT?        Check if the bit is set.
11  ISG 12      If it is then increment the
                accumulator by 1.
12  ISG X       Increment the bit pointer.
13  GTO 00      Repeat the loop.
14  VIEW 12     View the result.
15  END
```

To run the above program, simply key a number into the X-register and [XEQ] "BSUM". The result will be the sum of the 1's in the binary representation of the number. Note that the above program does not preserve the stack.

For example, with a current word size of 16 bits, key

in the number ABCD Hex and then [XEQ]"BSUM" and see the result, 10, in the display. (The X-register will contain the word size and the Y-register will contain the original number in its unsigned, decimal format.)

PROGRAM 3: DOING WINDOWS ON THE 16-E

The HP-16C calculator "does windows." This has to be one of the most interesting features of the "Computer Scientist" calculator. The HP-16C has a display that accomodates only 8 digits at a time. Yet, with a WINDOW key, you can view up to 8 "windows" or displays of 8 digits. This is an extremely clever way to let you see up to 64 binary digits of a given number.

Here is a program for the 41CV that will let you see all 32 bits of a number. The program uses many of the functions from the 16-E program and will show you both the hexadecimal and binary representation of a decimal number.

When keying this program into the HP-41, be careful of lines 17 and 31. The function shown is the Advantage Module's NOT operator: not the 16-E's "NOT". To key these lines into the program, press [XEQ] [ALPHA] NOT [ALPHA].

01	LBL "WNDO"	
02	HEXIN	Prompt for input of hex number.
03	STO 12	Store this in register 12.
04	SF 06	Turn off display of intermediate results.
05	8	
06	XEQ "MSKR"	Set up a mask of 8 bits on the far right of the number and "filter"
07	AND	through the 8 low order bits.

```
08  HEXVIEW     Show the hexadecimal
09  STOP
10  XEQ "BVU"   and binary representations of these
                bits.
11  STOP
12  8           Create an 8 bit mask
13  XEQ "MSKR"  on the far right.
14  16          Create a 16 bit mask on the far left
15  XEQ "MSKL"  of the 32 bit word.
16  OR          Combine these into one mask.
17  NOT         Then change all 1's to 0's and vice
                versa.
18  RCL 12      Get the original number.
19  AND         Filter through the bits 08-15.
20  8           Rotate these bits to the right by 8
                bits
21  XEQ "RRN"
22  HEXVIEW     and view the hex
23  STOP
24  XEQ "BVU"   and binary representations of these
                bits.
25  STOP
26  8           Create a mask of 8 bits on the far
27  XEQ "MSKL"  left
28  16          and another mask of 16 bits on the
29  XEQ "MSKR"  far right of the 32 bit word.
30  OR          Combine these masks and
31  NOT         take the complement of the
32  RCL 12      combination.
33  AND         Filter through bits 16-23,
34  16          and rotate these 8 bits to the right
35  XEQ "RRN"   by 16 bits.
36  HEXVIEW     Display the hex
37  STOP
38  XEQ "BVU"   and decimal versions of the bits.
39  STOP
40  RCL 12
41  8           Create a mask of 8 bits on the
```

```
42  XEQ "MSKL"   far left.
43  AND          Filter through the 8 most
                 significant bits
44  8            and
45  XEQ "RLN"    rotate these 8 bits to the left
46  HEXVIEW      and display the hex
47  STOP         and
48  XEQ "BVU"    decimal versions of these bits.
49  CF 06        Clear flag 06.
```

To test the program, BE SURE THAT WSIZ = 32. Then [XEQ] "WNDO". At the " _H" prompt, key in the number ABCDEF98 Hex and press the [R/S] key. You should see the successive outputs of

```
      98 H        Press [R/S] after each output
                  to proceed
10011000 B
      EF H
11101111 B
      CD H
11001101 B
      AB H
10101011 B
```

This program is slow and it does not preserve the stack. It also demands that you previously set the WSIZ to 32 bits. The program shows the high order byte first. We want it to show the low order byte first.

Let's see if we can't correct these shortcomings.

Begin by assuming that we will always want to view the contents of a 32 bit number. This will simplify the solution to the problem.

Here are some short cuts that we can incorporate. For instance, rather than use the MSKR and MSKL

routines in our program, let's just use the results of these masking routines. Instead of using the 16-E's "RRN" and "RLN" routines, substitute the ROTXY function from the Advantage ROM.

Thus, in place of

```
05   8
06   XEQ "MSKR"
```

we will use the number 255.

Similarly in place of the following sections of code we will use the final results.

CODE		RESULTS
12	8	
13	XEQ "MSKR"	255
14	16	
15	XEQ "MSKL"	4294901760
16	OR	4294902015
17	NOT	65280
.		
.		
.		
26	8	
27	XEQ "MSKL"	4278190080
28	16	
29	XEQ "MSKR"	65535
30	OR	4278255615
31	NOT	16711680
.		
.		
.		
41	8	
42	XEQ "MSKL"	4278190080

With this much being said, here is the program that will do the job:

```
01  LBL "W"
02  STO 06        Save the current stack.
03  R^
04  STO 09
05  R^
06  STO 08
07  R^
08  STO 07
09  R^
10  LASTX
11  STO 10
12  65280         Store the constants for masking
13  STO 11
14  16711680
15  STO 12
16  4278190080
17  STO 13
18  RCL 06        Filter out the left-most bits
19  AND
20  24
21  ROTXY         Rotate the bits 24 places to the
                  right and
22  HEXVIEW       view hex
23  STOP          and
24  BINVIEW       binary equivalents.
25  STOP          The use of STOP is necessary since
                  BINVIEW does not stop program
                  execution.
26  RCL 12        Get new mask.
27  RCL 06        Get original number.
28  AND           Filter.
29  16
30  ROTXY         Rotate.
31  HEXVIEW       View.
```

```
32  STOP        HEXVIEW does not stop program
                execution.
33  BINVIEW
34  STOP
35  RCL 06      Get original number.
36  RCL 11      Get mask.
37  AND         Filter.
38  8
39  ROTXY       Rotate.
40  HEXVIEW     View.
41  STOP
42  BINVIEW
43  STOP
44  RCL 06      Get original number.
45  255         Mask.
46  AND         Filter.
47  HEXVIEW     View.
48  STOP
49  BINVIEW
50  RCL 10      Restore the original stack.
51  SIGN
52  RCL 09
53  RCL 08
54  RCL 07
55  RCL 06
56  RTN
```

To run this program, you need to have a number in the X-Register whose hex and binary digits you wish to see. BE SURE AND SET THE SIZE OF THE 41CV's DATA REGISTERS TO 14. Then simply [XEQ] "W" and note the results.

This program is certainly much faster than the previous version. Note that it also is independent of the current word size and displays the binary digits from left to right. The program does not have all the flexibility of the WINDOW operator on the HP-16C,

but it does share some of the speed and utility of that function.

You may wish to assign this new function to a key. I like the [#][O] (the ISG key) for this. Don't forget to press the [USER] key to get out of USER mode before attempting to use the ASN function. [#][K] means "ASR" in USER mode, doesn't it?

You may also want to incorporate the new routine into the 16-E program. It sounds like a good idea, but, read the next paragraph completely before deciding to take this route.

You can bring the "W" program INTO the 16-E program most easily. Just place the "W" program immediately after the 16-E program in the HP-41's memory. Then get into the 16-E program and move the program pointer to the END. Then press [#][RTN]. This will enter a RTN into the program. [SST] and [CLx] will wipe out the END and then [SST] will show you LBL "W". Now for the punch line: even after you "recompile" the 16-E program, you will find that the "W" program runs much more slowly than it does when it is a stand-alone program. For this reason I always keep the "W" program separate from the 16-E program.

I have attempted to show you how easy it is to program the 16-E calculator. I have also showed you a couple of techniques that you may use to refine and speed up your programs. I have not mentioned anything about using loops, subroutines, conditional branching, and on and on. If you want to know more about the nifty things you can do with the 16-E program, pick up the excellent book, *An Easy Course In Programming the HP-41* and enjoy yourself and your calculator even more than you have to date.

NOTES

NOTE 1: If it is essential that your program have the ability to set the word size, then you may modify the 16-E routine by inserting a RTN command between lines 31 and 32. This will let you use the routine as a subprogram in your own programs. You may also wish to make the routine automatic by keying lines 13 through 28 from the 16-E program into your own program. In your program you should precede these lines with the number of bits that you want to set for a word size for that particular program. (Why did I choose to have a "GTO 13" at the end of the "WS" routine rather than a "RTN"? Well, I somehow got the nasty habit of pressing the [R/S] key after I had just changed the word size. This would push me ahead into the "SB" routine which would always give me an error message: very messy. GTO 13 solved my problem. I hope I didn't leave you, the more careful user of the HP-41, with a bigger problem.)

NOTE 2: In line 07, you may substitute the HP-41C's minus operator for the minus (LBL "−") routine in the 16-E.

You may also replace lines 09-13 with the faster commands:

09	/	Normal division
10	INT	
11	RCL X	
12	LASTX	
13	MOD	Check to see if there is a remainder.
14	X≠0?	If there is a remainder, increase
15	ISG Y	the quotient by 1.

```
16  LBL 00      --a null operator (NOP)
17  RDN         Move the quotient into the
                X- register.
18  CF 06
19  END
```

Space for more of your own NOTES:

Chapter 9.

TECHNICAL DETAILS OF THE 16-E

Summary of User Instructions for the 16-E Program

STEP 1: *SET SIZE*

Set the size of the HP-41 to 11 or greater.

STEP 2: *LOADING*

Load the 16-E program:

a. To capture the global key assignments, turn ON the USER mode before loading the program from magnetic media. To avoid key assignments, turn USER mode OFF prior to loading.

STEP 3: *INITIALIZATION*

INPUT	FUNCTION	DISPLAY
	XEQ "WS"	WSIZ=ww
	[#][D]	

a. If ww is the correct word size..

INPUT	FUNCTION	DISPLAY
	[R/S]	WSIZ=ww

b. ...otherwise to change word size...

INPUT	FUNCTION	DISPLAY
WW	[R/S]	WSIZ=ww

This routine will set FIX 0 and clear flags 6 and 29 and set flag 27, the USER mode. If you wish to avoid the use of the assigned keys, switch the USER mode OFF.

STEP 4: *DISPLAY FORMATS*

Note that all computations are performed in floating point decimal. The results are shown via the alpha register. The X-register always contains the unsigned decimal equivalent of the number in the display and alpha registers.

To switch to....

a. One's complement format...

INPUT	FUNCTION	DISPLAY
	XEQ "1c"	NNN
	[#][1]	1 annun on

b. Two's complement format...

INPUT	FUNCTION	DISPLAY
	XEQ "2c"	NNN
	[#][2]	2 annun on

c. Unsigned decimal format...

INPUT	FUNCTION	DISPLAY
	XEQ "UNS"	NNN
	[#][3]	3 annun on

d. To obtain the result of an operation in the current decimal format...

INPUT	FUNCTION	DISPLAY
	XEQ"DECV"	NNN
	[D]	

e. To view the binary equivalent...

INPUT	FUNCTION	DISPLAY
	XEQ"BVU"	01010 B
	[B]	

Note that numbers greater than 1023 cause the routine to default to current decimal format.

f. To view octal or hexadecimal equivalent...

INPUT	FUNCTION	DISPLAY
	XEQ"OCTVIEW"	
	[C]	1234567 O
	XEQ"HEXVIEW"	
	[E]	129AEF0 H

STEP 5: *BYTE MANIPULATIONS*

a. Shift Right...

INPUT	FUNCTION	DISPLAY
	XEQ "SR"	SRSR
	[H]	

b. Shift Left...

INPUT	FUNCTION	DISPLAY
	XEQ "SL"	SLSL
	[#][H]	

c. Arithmetic Shift Right...

INPUT	FUNCTION	DISPLAY
	XEQ "ASR"	ASAS
	[#][K]	

d. Rotate Right by n

INPUT	FUNCTION	DISPLAY
nnn	XEQ "RRN"	rrnrrn
	[J]	

e. Rotate Left by n

INPUT	FUNCTION	DISPLAY
nnn	XEQ "RLN"	rlnrln
	[I]	

f. Rotate Right through the carry bit by n.

INPUT	FUNCTION	DISPLAY
nnn	XEQ "RRCN"	rcrcr
	[#][J]	

g. Rotate Left through the carry bit by n.

INPUT	FUNCTION	DISPLAY
nnn	XEQ "RLCN"	rlcrlc
	[#][I]	

(For a rotation of 1, use 1 for nnn)
(nnn is the number of bits by which to rotate the number.)

h. Right Justify a number...

INPUT	FUNCTION	DISPLAY
	XEQ "RJY"	rjyrjy
	[#][G]	

i. Left Justify a number...

INPUT	FUNCTION	DISPLAY
	XEQ "LJY"	ljyljy
	[#][F]	

j. Mask Right (justified)...

INPUT	FUNCTION	DISPLAY
nnn	XEQ "MSKR"	mmmm
	[A]	

(Key in the number of bits to use as a mask.)

k. Mask Left (justified)...

INPUT	FUNCTION	DISPLAY
nnn	XEQ "MSKL"	nnnn
	[#][A]	

(Key in the number of bits to use as a mask.)

l. Changing the sign of a number

INPUT	FUNCTION	DISPLAY
	XEQ "CHS"	–NNN
	[CHS]	

(Note that, in unsigned format, this operation has no meaning. Flag 4 is set as a reminder that the number is negative.)

STEP 6: *BIT MANIPULATIONS*

a. Test Bit...Key in the number of the bit to be tested (0-31) Note that there is no error trap for out of range values.

INPUT	FUNCTION	DISPLAY
bb	XEQ "B?" [#][6]	YES NO

b. Setting and Clearing Bits: key in the number of the bit to be set or cleared...

INPUT	FUNCTION	DISPLAY
bb	XEQ "SB" [#][4]	nnnn
bb	XEQ "CB" [#][5]	nnnn

Note that out of range numbers will cause the program to branch to the Word Size routine as a reminder that there is an error condition.

STEP 7: *LOGICAL FUNCTIONS*

a. AND, OR, EXOR are all functions taken directly from the Advantage ROM. Their execution from the keyboard of the 16-E should be followed by the command XEQ"DECV" in order to obtain the correct decimal equivalent in the current word size.

INPUT	FUNCTION	DISPLAY
	XEQ "AND" [#][*]	

XEQ "OR"
[#][+]
XEQ "XOR"
[#][/]

b. To find the complement of a number.

INPUT	FUNCTION	DISPLAY
	XEQ "NOT"	

(Note there is a difference between the NOT operator inherent to the Advantage ROM and the NOT operator in the 16-E. The former works only for 32 bit words. The latter will operate with any size word up to 32 bits.)

INPUT	FUNCTION	DISPLAY
	XEQ "NOT"	
	[#][-]	

STEP 8: *ARITHMETIC FUNCTIONS:* +, -, *, /

These functions are similar to the two number functions on the 41C. However, out of range errors (for the current word size) will turn on the annunciator for flag 04 without terminating the program's operation. (Much testing was done for out of range errors on the + and - routines but only for WSIZ = 4 and 8.

INPUT	FUNCTION	DISPLAY
	XEQ "+"	
	[+]	
	XEQ "-"	
	[-]	
	XEQ "*"	
	[*] XEQ "/"	

[/]
XEQ "RMD"
[#][0]

STEP 9: *INPUT OTHER THAN DECIMAL*

a. To input binary numbers (use only the numbers 0 and 1.)

INPUT	FUNCTION	DISPLAY
	XEQ "BININ"	
	[#][B]	

b. To input octal numbers (use only numbers 0,1,2,3,4,5,6,7).

INPUT	FUNCTION	DISPLAY
	XEQ "OCTIN"	
	[#][C]	

c. To input hexadecimal numbers (use only hex numbers 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Any other keys will terminate entry.

INPUT	FUNCTION	DISPLAY
	XEQ "HEXIN"	
	[#][E]	

Data Registers Used

REG 00: Word size
REG 01: 2^WSIZ
REG 02: 2^WSIZ-1
REG 03: Max. Positive Num.
REG 04: Max. Positive Num.+1
REG 05: counter/scratch
REG 06: X-Register
REG 07: Y-Register
REG 08: Z-Register
REG 09: T-Register
REG 10: LastX

Program Errors and Probable Causes

Program Error:	Reasons for this error:
"NONEXISTENT"	You did not set the SIZE of the calculator to 11 or greater. Or you mis-keyed the name of a 16-E function. e.g. You keyed in [XEQ] "MASKR" instead of [XEQ] "MSKR".
"DATA ERROR"	You tried to find out if bit 32 in a number was set or not? (Any number larger than 31 will cause this error.)
"ERR: WSIZ=8"	You tried to set or clear a nonexistent binary digit (one that is out of bounds for the current word size.)

NOTE: Sometimes these error messages will get stuck in the display. If that happens, wait three seconds and press the [Clx] key to procede.

The 16-E program is very "rugged." You should not be able to "crash" the HP-41 by pressing any of the keys in the 16-E. If you encounter any "bugs" or glitches in the program, please check your work and see if you can get the bug to show its ugly head again. Let me know. I'll fix the bug or explain why it's not really a bug at all.

Barcode "WS," *16-E Program*, 184 Registers Required

WS

ROW 1: LINES 1-5

ROW 2: LINES 5-10

ROW 3: LINES 10-19

ROW 4: LINES 20-32

ROW 5: LINES 32-39

ROW 6: LINES 40-46

ROW 7: LINES 47-56

ROW 8: LINES 57-62

ROW 9: LINES 63-72

ROW 10: LINES 72-79

ROW 11: LINES 80-87

ROW 12: LINES 87-90

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 13: LINES 91-97

ROW 14: LINES 97-108

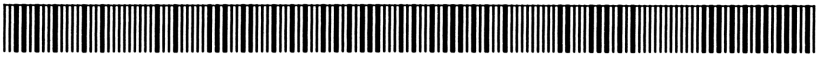
ROW 15: LINES 108-115

ROW 16: LINES 116-123

ROW 17: LINES 124-128

ROW 18: LINES 129-138

ROW 19: LINES 139-147



ROW 20: LINES 148-156

ROW 21: LINES 156-159

ROW 22: LINES 159-169

ROW 23: LINES 170-176

ROW 24: LINES 177-185

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 25: LINES 186-190

ROW 26: LINES 190-198

ROW 27: LINES 199-207

ROW 28: LINES 207-216

ROW 29: LINES 216-224

ROW 30: LINES 224-228

ROW 31: LINES 229-240

ROW 32: LINES 241-244

ROW 33: LINES 245-257

ROW 34: LINES 258-263

ROW 35: LINES 263-271

ROW 36: LINES 272-280

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 37: LINES 281-285

ROW 38: LINES 285-295

ROW 39: LINES 296-304

ROW 40: LINES 305-310

ROW 41: LINES 311-320

ROW 42: LINES 320-325

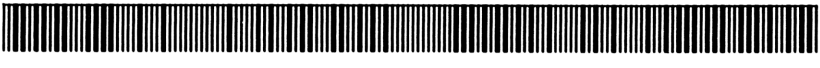
ROW 43: LINES 325-335

ROW 44: LINES 336-344

ROW 45: LINES 345-349

ROW 46: LINES 349-360

ROW 47: LINES 360-368



ROW 48: LINES 368-377

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

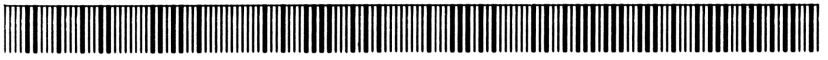
WS

ROW 49: LINES 378-383

ROW 50: LINES 383-392

ROW 51: LINES 392-398

ROW 52: LINES 399-407



ROW 53: LINES 407-413

ROW 54: LINES 413-420

ROW 55: LINES 421-427

ROW 56: LINES 427-436

ROW 57: LINES 437-445

ROW 58: LINES 446-452

ROW 59: LINES 453-462

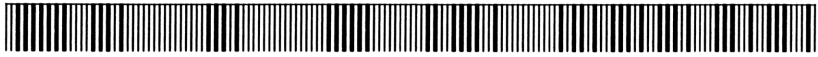
ROW 60: LINES 462-471

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

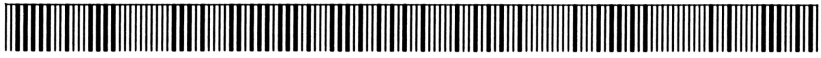
WS

ROW 61: LINES 472-479

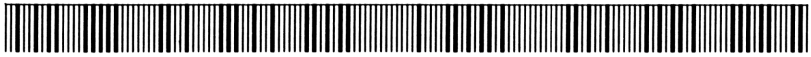
ROW 62: LINES 480-486



ROW 63: LINES 487-495



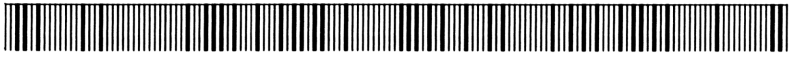
ROW 64: LINES 497-504



ROW 65: LINES 505-512

ROW 66: LINES 514-521

ROW 67: LINES 523-532



ROW 68: LINES 534-539

ROW 69: LINES 541-548

ROW 70: LINES 550-556

ROW 71: LINES 557-562

ROW 72: LINES 563-569

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 73: LINES 571-578

ROW 74: LINES 580-586

ROW 75: LINES 587-595

ROW 76: LINES 597-603

ROW 77: LINES 604-612

ROW 78: LINES 614-623

ROW 79: LINES 625-632

ROW 80: LINES 634-642

ROW 81: LINES 643-647

ROW 82: LINES 648-656

ROW 83: LINES 657-667

ROW 84: LINES 668-675

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 85: LINES 677-685

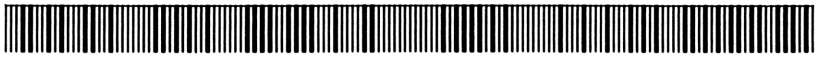
ROW 86: LINES 687-691

ROW 87: LINES 692-695

ROW 88: LINES 696-699

ROW 89: LINES 701-703

ROW 90: LINES 705-707



ROW 91: LINES 708-718

ROW 92: LINES 719-725

ROW 93: LINES 727-734

ROW 94: LINES 736-743

ROW 95: LINES 745-755

ROW 96: LINES 756-762

Barcode "WS," *16-E Program*, 184 Registers Required (Continued)

WS

ROW 97: LINES 763-773

ROW 98: LINES 775-778

ROW 99: LINES 779-782

Barcode "W," *Window Program*, 15 Registers Req'd.

W

ROW 1: LINES 1-9

ROW 2: LINES 10-14

ROW 3: LINES 14-16

ROW 4: LINES 16-24

ROW 5: LINES 24-32

ROW 6: LINES 33-41

ROW 7: LINES 42-49

ROW 8: LINES 49-57

REFERENCES

Boardman, James H., "The Programmer Plus," program 01448C, HP Users Library, Corvallis, OR.

Carlstrom, Randy, "Number Systems for Microcomputers," *COMPUTERS AND ELECTRONICS*, Dec., 1983, pp. 47-55.

Hewlett-Packard, *HP-16C USERS MANUAL*, 1982, Hewlett-Packard Co., Corvallis, OR.

Hewlett-Packard, *HP-41 ADVANTAGE: ADVANCED SOLUTIONS PAC*, July, 1985, Hewlett-Packard Co., Corvallis, OR.

Keefe, Edward M. *THE HEWLETT-PACKARD 16C: A WORKSHOP MANUAL*, private publication, 1983.

Pearce, Craig A., "The Micro Scene," *PPC JOURNAL*, v9n4p71, v9n5p20, v9n7p57, 1982.

Impress A Friend!

With a book from the press at

Grapevine Publications, Inc.

P.O. Box 118, Corvallis, OR 97339, Tel. 1-800-338-4331

To save time and shipping costs, first check for our books at your local Hewlett-Packard Dealer. Or simply use this handy Order Form. Check the book(s) you would like to order on the list below. Then fill out the form on the back of this page, and send it to us along with your check, money order, or VISA or MasterCard number. *OR* , call our Toll-Free Order Line at 1-800-338-4331.
Thank you!

Please send me the following books:

__*An Easy Course In Programming the HP-41*	$19.95
__*Using Your HP-41 Advantage ROM: Statics*	$11.95
__*Using Your HP-41 Advantage ROM: Electrical Circuits*	$11.95
__*Computer Science On Your HP-41*	$14.95
__*The HP-16C Training Guide*	$24.95
__*An Easy Course In Programming the HP-11C & HP-15C*	$19.95
__*An Easy Course In Using the HP-12C*	$19.95
__*The HP-12C Pocket Guide: Just In Case*	$4.95
__*The HP Business Consultant Training Guide*	$21.95
__*The HP Business Consultant Pocket Companion*	$7.95
__ Shipping & Handling: Special 4th Class (2-3 weeks)	$2.00
UPS (for faster service)	$3.50

Don't forget to fill out your name & address on the other side!

YES! Please send me the book(s) I've marked on the other side of this page.

Name _____

Address _____

City _____

State & Zip _____

Telephone _____

__ My check or money order is enclosed.

__ I'll use my MasterCard or VISA.

Bank Card No. _____

Exp. Date _____

Signature _____

SHIPPING & HANDLING (Don't forget to include this!):

Special 4th Class (allow 2-3 weeks)	$2.00
For faster UPS service	$3.50

TOTAL ENCLOSED (Cost of Books + Shipping): $_____

SEND TO: GRAPEVINE PUBLICATIONS, INC.
P.O. BOX 118
CORVALLIS, OR 97339-0118, U.S.A.

OR CALL US: TOLL-FREE 1-800-338-4331 *Thanks!*

Computer Science On Your HP-41

Using the Advantage Module

By Ed Keefe

Computer scientists! Computer students! Here is a thoroughly useful and enjoyable tool, written by Ed Keefe, who is both a teacher of computer programming and an experienced HP-41 user and programmer.

Keefe has written a program for your HP-41 that, with the help of the powerful HP-41 Advantage Module, simulates the HP-16C (Hewlett-Packard's "Computer Scientist" calculator) on your HP-41.

You get all those functions you've always needed, including number base conversions in binary, octal, hexadecimal (and decimal, of course), Boolean functions, AND, OR, XOR, NOT, bit rotation, summation, justifying, masking, 1's and 2's complement and unsigned word format, and variable word sizes (up to 32 bits).

You'll soon be doing binary arithmetic and problem-solving with ease and speed—thanks to this powerful program and its common-sense key assignments. Then you'll learn how to use parts of this program to build your own solutions.

You get all this, including the program listings, barcode, instructions, examples and documentation. And it's all written in a friendly, conversational style. All that's missing is your HP-41, your Advantage Module and you!

0 12841 00010 1

ISBN 0-931011-10-8

Grapevine Publications, Inc. • P.O. Box 118 • Corvallis, OR • 97339 • U.S.A.