6502 USER NOTES

no. 16 ——————— $2.50

6502 FORTH is here !! (SEE INSIDE BACK COVER)

EDITORIAL

6502 FORTH is ready for distribution! I also have seven articles on 6502 FORTH enhancements for future issues. One of the articles even describes how to add machine language monitor-like functions to 6502 FORTH! That really drove home the point to me that FORTH is a complete programming system!

Issues 1-6 of the User Notes have been completely re-typed and are now going in to layout. The typing took longer than we expected so I can't announce back issues as being ready yet. If you would like immediate notification of back issue availability (issues 1-6) please send us a self addressed stamped envelope. As soon as we are ready to take orders, we will let you know the price. Your patience is appreciated.

I don't believe price alone should be the determining factor in purchasing computers software and equipment. This seems to be the case, though, with most of the hobbyists I have spoken with. Consumer education seems to be the way to turn this around.

In an upcoming issue, I plan to present comparison charts for all the 6502 assemblers which are available and another one for the 6502 disk systems.

Hopefully, this will give us all an overall picture of what we're really getting for our bucks.

—Whats Happening—

ROCKWELL has recently added several AIM-65 application notes to their already substantial array of system documentation.

No. R6500 N08   RS-232C INTERFACE FOR AIM-65
    R6500 N09   INTERFACING R6500 MICROPROCESSOR
                TO A FLOPPY DISK CONTROLLER.
    R6500 N11   INTERFACING KIM-4 TO AIM-65

    R6500 N12   A CRT MONITOR OR TV INTERFACE FOR
                AIM-65

(The last one is particularly interesting as it presents a complete hardware and software design for a 16 chip 40x16 video interface using the Motorola (and shortly Synertek's) 6845 CRT controller chip. This design looks to be useable on ANY 6502 based system with a 1 MHz clock. Parts costs should be around $100).

These application notes are available at no charge from:

> ROCKWELL MICROELECTRONIC DEVICES
> MARKETING SERVICES
> PO BOX 3669, RC55
> ANAHEIM, CA 92803

Rockwell also announced their AIM-65 Expansion Motherboard. This 5-slot backplane is bus compatible with the SYSTEM/65 as well as the Motorola Exorciser. Its priced at $195. For a complete product description which includes a complete schematic, ask for document number 29650 N57.

SYNERTEK has dropped the retail price of the SYM to $239!

They've also introduced several new chips which should help to bolster the 65XX family image. One of these chips, the 6551 ACIA, is mentioned in the 65XX FAMILY CHIP STUFF section elsewhere in this newsletter. The other new chips, the 6545 CRT controller and the floppy disc controller are not into production yet and will be mentioned more when they are real.

HDE has been shipping their mini-floppy systems and their cassette based assembler/editor package.

A number of firms, including HDE, have been caught in the industry wide LS (low power Schotky) component shortage which has been playing OEM's for most of the summer.

Before ordering anything that might contain LS components, it might be worth a phone call or a letter to the supplier to see what the situation is. This shouldn't be regarded as a blanket excuse for slow delivery times from suppliers but could be a possible explanation.

An outfit called Perry Peripherals is adapting the HDE mini floppy system to the S-100/KIMSI system. Should have more info by next issue.

We now have a European distributor! European stores can contact:

> ING. W. HOFACKER GMBH
> 8 MUNCHEN 75
> POSTFACH 437
> WEST GERMANY

An aquaintance of mine and a fellow ex-MOS Technology employee, Ray Bennett, President of RNB Enterprises, called me the other day with some disturbing news. Ray indicated that he noticed a very substantial decrease in his mail order business when the news of the World Power Systems fraud became known.

It's the old story of a few unsavory types messing things up for the rest.

LOCAL 6502 ORIENTED USER GROUPS

LICA (Long Island Computer Assoc.)
#6 Brookhaven Dr.
Rocky Point, NY 11778

SAN FERNANDO VALLEY 6502 USERS CLUB meet at 8:00 PM on the 2nd Tuesday of each month at Computer Components of Burbank, 3808 West Verdugo Ave, Burbank CA 91505. Contact Larry Goga (3816 Albright Ave, Los Angeles, CA 90066, 213-398-6086), for more info. This group also publishes a monthly newsletter which is available for $2.00 a year. Useful stuff!

WAKE--Washington Area KIM Enthusiasts--meet each month at the McGraw-Hill Continuing Education Center in Washington, D.C. to study operation, expansion, and applications of KIM 1 microcomputers. Meetings are at 7:30 on the third Wednesday of every month.

For a copy of the current WAKE newsletter, send a stamped, self-addressed envelope to WAKE, c/o Ted Beach, 5112 Williamsburg Blvd., Arlington, VA 22207, or phone (703) 538-2303.

FOUR PART HARMONY (cheap!!)

KIM-1 MUSIC PROGRAM

by Richard Martin

(A) ABSTRACT

The KIM-1 music program "plays" music in four part harmony on an unexpanded KIM-1 micro-computer.

(B) EQUIPMENT REQUIRED:

KIM-1 module with power supply, four (4) 150K ohm resistors, a 2.2 uF capacitor, an audio amplifier, and speaker. Refer to Figure 1 for hardware connection.

(C) OPERATION

The program causes a series of 75 microsecond pulses to be output on each of four PIA terminals (PA0-PA3). The frequency of the pulse waveform on each terminal is independently controlable by data stored in the KIM-1 memory. The pulses are mixed together by a simple resistance network and coupled through a capacitor to an ordinary audio amplifier.

(D) USAGE

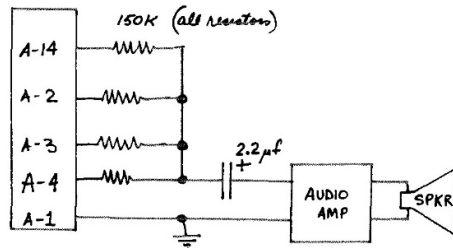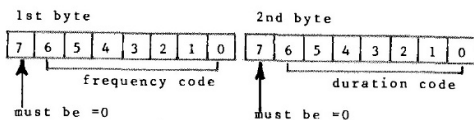The program accepts three different types of coded notes: normal, compressed, and branch.

FIGURE 1.

(1) Normal notes:

These notes require two bytes of storage.

must be =0   must be =0

Frequency code-Determines pitch of note. Refer to Figure 2 for a list of pitches and their respective codes. Notice that a frequency code of 00 (hex) specifies a rest (no tone generated).

Duration code-Determines length of note. The note will be held for the number of counts specified by the duration code. The codes in Figure 3 are recommended for most work.

Examples: 33 08 causes middle C to be played for eight counts. (eighth note)
2E 20 causes A above middle C to be played for 32 counts. (half note)

After the note is over, the playing program goes to the next consecutive byte in storage to fetch the next note.

79 (-1.19)	
	72 (+1.97)
6C (-4.42)	
66 (-5.47)	
	60 (-0.51)
5B (-7.91)	
	56 (-10.07)
51 (-6.38)	
4C (+3.92)	
	48 (-2.47)
44 (-3.51)	
	40 (+1.43)
3C (+13.17)	
	39 (+1.97)
36 (-4.42)	
MIDDLE C 33 (-5.47)	
	30 (-0.51)
2D (+11.21)	
	2B (-10.07)
28 (+15.12)	
26 (+9.92)	
	24 (-2.47)
22 (-3.51)	
	20 (+1.43)
1E (+13.16)	

Frequency codes are in hexadecimal. Numbers in () are the relative tuning in cents. This is to provide some idea of how out-of-tune the notes are (a pitch difference of 6 cents can be detected only by very sensitive ears).

figure 2.

(2) Compressed notes:

If several notes with the same duration are to be played in succession, they can be shortened to one byte each, conserving considerable amounts of memory space.

The pitch will change to that specified by the new frequency code. The duration code, however, will remain the same as it was for the last note.

frequency code

must be =1

1
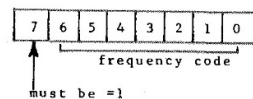
Example:

This is how the C scale would be coded:


```
66 20
58 10
D1
CC
C4
BC
B6
33 40
```

(3) Branch notes:

Branch notes are similar to normal notes, however, instead of going to the next consecutive address in memory for the next notes, the branch note causes the program to jump to a new address for the next note. Branch notes require four bytes in storage.
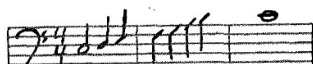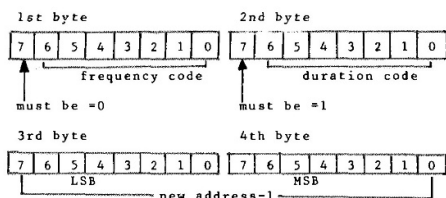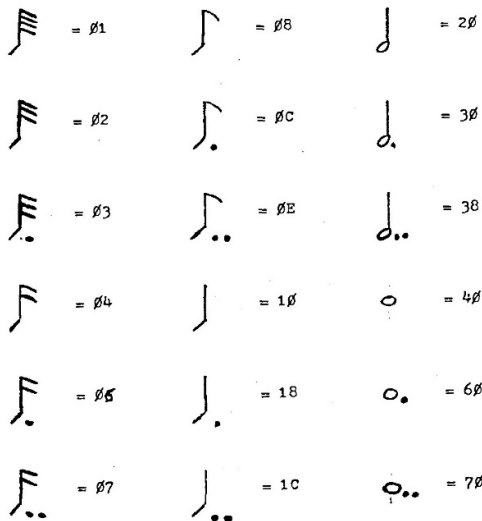
```
1st byte                2nd byte
 7 6 5 4 3 2 1 0        7 6 5 4 3 2 1 0
      frequency code         duration code

must be =0              must be =1

3rd byte                4th byte
 7 6 5 4 3 2 1 0        7 6 5 4 3 2 1 0
        LSB                     MSB
                new address-1
```

When the branch note is over, the next note played will be the one with its 1st byte at the new address. Branch notes are useful for making a song repeat, and for skipping unusable addresses in memory.


```
♫ = Ø1        ♩ = Ø8        𝅗𝅥 = 2Ø
♫ = Ø2        ♩. = ØC       𝅗𝅥. = 3Ø
♫ = Ø3        ♩.. = ØE      𝅗𝅥.. = 38
♪ = Ø4        ♩ = 1Ø        o = 4Ø
♪ = Ø6        ♩. = 18       o. = 6Ø
♪ = Ø7        ♩.. = 1C      o.. = 7Ø
```

(E) PROGRAMMING TECHNIQUES

(1) Locations 0000-00CB and 0100-010F are used by the playing program. All other locations may be used for music data. Music data normally resides from 0110-03FF.

(2) The stack pointer should always be set to 0A before loading music data. Otherwise, the stack may overwrite the music data.

(3) Keep in mind that the playing program is capable of playing four independent melodies simultaneously. The melodies or "parts" should be placed one after another in memory.

(4) Once all four parts have been loaded, the playing program's internal pointers and counters must be initialized as follows:

ADDR		
0008	LSB	
		Starting address-1 for the first part
0009	MSB	
000A	LSB	
		Starting address-1 for the second part
000B	MSB	
000C	LSB	
		Starting address-1 for the third part
000D	MSB	
000E	LSB	
		Starting address-1 for the fourth part
000F	MSB	

Locations 0010-0017 must be set to 01. The tempo is set at location 0007. Tempos range from 10-FF with FF being the slowest.

The playing program can now be started at location 0000.

(5) The playing program "slurs" the notes together (no seperation between the notes). Therefore:

```
cannot be coded as :  44   10    or    44   10
                      44   10           C4
```

To make the notes sound seperate, remove one count from the first note and insert one count of rest between the two notes. The example above would then be coded as:
```
                      44   ØF
                      00   01
                      44   10
```

Similar techniques may be employed to give staccato notes.

(6) An optional patch may be inserted into the program to make the fourth part sound an octave lower:

(refer to program listing)

Change loc. 009C from 40 to CC. Then Add:

```
00CC    A6 16       LDX    DURD
00CE    E4 17       CPX    XDURD
00D0    D0 04       BNE    Q
00D2    06 26       ASL    PD
00D4    06 27       ASL    XD
00D6    4C 40 00 Q  JMP    TONE
```

The patch occupies locations 00CC-00D8. Similar routines may be written for the other parts if necessary.

2   Figure 3

```
LINE#  ADDR   OBJECT    SOURCE        PAGE 0001

0010   2000             ;KIM-1 MUSIC PROGRAM
0020   2000             ;WRITTEN BY RICHARD MARTIN 5/76
0030   2000             ;
0040   2000             ;I/O AND TIMER ADDRESSES
0050   2000             PIA   =$1700        ;OUTPUT REGISTER
0060   2000             DDR   =$1701        ;DATA DIRECTION REGISTER
0070   2000             TIMER =$1707        ;INTERVAL TIMER
0080   2000             ;
0090   2000             ;INITIALIZE PIA AND START
0100   2000             *=$0
0110   0000             .OFF 2000
0120   0000   A9 0F     START LDA #$F
0130   0002   8D 01 17        STA DDR
0140   0005   D0 69          BNE NEW
0150   0007             ;
0160   0007             ;Z PAGE VARIABLES AND CONSTANTS
0170   0007             ;
0180   0007             TEMPO  *=*+1
0190   0008             PTAL   *=*+1         ;CURRENT NOTE POINTERS
0200   0009             PTAH   *=*+1
0210   000A             PTBL   *=*+1
0220   000B             PTBH   *=*+1
0230   000C             PTCL   *=*+1
0240   000D             PTCH   *=*+1
0250   000E             PTDL   *=*+1
0260   000F             PTDH   *=*+1
0270   0010             DURA   *=*+1         ;NOTE DURATION COUNTERS
0280   0011             XDURA  *=*+1         ;CURRENT NOTE DURATION
0290   0012             DURB   *=*+1
0300   0013             XDURB  *=*+1
0310   0014             DURC   *=*+1
0320   0015             XDURC  *=*+1
0330   0016             DURD   *=*+1
0340   0017             XDURD  *=*+1
0350   0018             RSTA   *=*+1         ;USED FOR OUTPUT SWITCHING
0360   0019   01        X01    .BYTE $1
0370   001A             RSTB   *=*+1
0380   001B   02        X02    .BYTE $2
0390   001C             RSTC   *=*+1
0400   001D   04        X04    .BYTE $4
0410   001E             RSTD   *=*+1
0420   001F   08        X08    .BYTE $8
0430   0020             PA     *=*+1         ;PITCH COUNTERS
0440   0021             XA     *=*+1         ;CURRENT PITCH
0450   0022             PB     *=*+1
0460   0023             XB     *=*+1
0470   0024             PC     *=*+1
0480   0025             XC     *=*+1
0490   0026             PD     *=*+1
0500   0027             XD     *=*+1
0510   0028   FE        R01    .BYTE $FE      ;USED FOR OUTPUT SWITCHING
0520   0029   FD        R02    .BYTE $FD
0530   002A   FB        R04    .BYTE $FB
0540   002B   F7        R08    .BYTE $F7
0550   002C             ;
0560   002C             ;RESET AND DELAY ROUTINES
0570   002C             ;FOR TONE GENERATION
0580   002C             ;
0590   002C   25 28     DELA   AND R01        ;TURN OFF OUTPUT A
0600   002E   EA               NOP            ;WASTE MORE TIME
0610   002F   D0 19            BNE RA         ;JUMP BACK TO MAIN ROUTINE
0620   0031   25 29     DELB   AND R02
0630   0033   EA               NOP
0640   0034   D0 1E            BNE RB
0650   0036   25 2A     DELC   AND R04
0660   0038   EA               NOP
0670   0039   D0 23            BNE RC
0680   003B   25 2B     DELD   AND R08
0690   003D   EA               NOP
0700   003E   D0 28            BNE RD
0710   0040             ;
0720   0040             ;MAIN TONE GENERATION ROUTINE
0730   0040             ;
0740   0040   C6 20     TONE   DEC PA         ;DECREMENT PITCH CNTR 'A'
0750   0042   D0 E8            BNE DELA
0760   0044   A6 21            LDX XA         ;RESTORE PITCH COUNTER
0770   0046   86 20            STX PA
0780   0048   05 18            ORA RSTA
0790   004A   C6 22     RA     DEC PB
0800   004C   D0 E3            BNE DELB
0810   004E   A6 23            LDX XB
0820   0050   86 22            STX PB
0830   0052   05 1A            ORA RSTB
0840   0054   C6 24     RB     DEC PC
0850   0056   D0 DE            BNE DELC
```

```
0860   0058   A6 25          LDX  XC
0870   005A   86 24          STX  PC
0880   005C   05 1C          ORA  RSTC
0890   005E   C6 26     RC    DEC  PD
0900   0060   D0 D9          BNE  DELD
0910   0062   A6 27          LDX  XD
0920   0064   86 26          STX  PD
0930   0066   05 1E          ORA  RSTD
0940   0068   8D 00 17  RD    STA  PIA         ;UPDATE ALL OUTPUTS
0950   006B   2C 07 17        BIT  TIMER       ;HAS AN INTERVAL PASSED?
0960   006E   10 D0          BPL  TONE         ;NO, KEEP GENERATING TONES
0970   0070                ;
0980   0070                ;THIS ROUTINE UPDATES THE NOTES
0990   0070                ;
1000   0070   A2 00     NEW   LDX  #$0         ;INITIALIZE X-INDEX REG
1010   0072   D6 10     NEW1  DEC  DURA,X      ;DECREMENT DURATION CNTR
1020   0074   D0 18          BNE  NXT          ;GO ON TO NEXT
1030   0076   20 C3 00        JSR  INPTLA      ;GET NEXT PITCH
1040   0079   30 23          BMI  SMDUR        ;IF MSB=1, USE SAME DURATION
1050   007B   95 21          STA  XA,X         ;STORE THE PITCH
1060   007D   95 20          STA  PA,X
1070   007F   F0 02          BEQ  NEW2         ;IF REST, STORE 00 IN RSTA
1080   0081   B5 19          LDA  XO1,X        ;OTHERWISE, USE XO1
1090   0083   95 18     NEW2  STA  RSTA,X
1100   0085   20 C3 00        JSR  INPTLA      ;GET NEXT DURATION
1110   0088   30 25          BMI  BRNCH        ;IF MSB=1, NOTE IS A BRANCH
1120   008A   95 11          STA  XDURA,X      ;STORE THE DURATION
1130   008C   95 10     NEW3  STA  DURA,X
1140   008E   E8        NXT   INX              ;SET 'X' TO UPDATE NEXT NOTE
1150   008F   E8             INX
1160   0090   E0 08          CPX  #$8          ;IF 'X'=8, THEN...
1170   0092   D0 DE          BNE  NEW1         ;WE ARE DONE
1180   0094   A5 07          LDA  TEMPO        ;INITIALIZE THE TIMER
1190   0096   8D 07 17        STA  TIMER
1200   0099   A9 10          LDA  #$10         ;INITIALIZE THE ACCUMULATOR
1210   009B   4C 40 00        JMP  TONE         ;RESUME TONE GENERATION
1220   009E                ;
1230   009E                ;ROUTINE FOR COMPRESSED NOTES
1240   009E                ;
1250   009E   29 7F     SMDUR AND  #$7F        ;SET MSB =0
1260   00A0   95 21          STA  XA,X         ;STORE THE PITCH
1270   00A2   95 20          STA  PA,X
1280   00A4   F0 02          BEQ  SMDU2        ;IF REST, STORE 0 IN RSTA
1290   00A6   B5 19          LDA  XO1,X        ;OTHERWISE, USE XO1
1300   00A8   95 18     SMDU2 STA  RSTA,X
1310   00AA   B5 11          LDA  XDURA,X      ;USE THE DURATION FROM
1320   00AC   4C 8C 00        JMP  NEW3         ;THE LAST NOTE
1330   00AF                ;
1340   00AF                ;BRANCH ROUTINE
1350   00AF                ;
1360   00AF   29 7F     BRNCH AND  #$7F        ;SET MSB =0
1370   00B1   95 11          STA  XDURA,X      ;STORE THE DURATION
1380   00B3   95 10          STA  DURA,X
1390   00B5   20 C3 00        JSR  INPTLA      ;GET LSB OF BRANCH ADDRESS
1400   00B8   A8             TAY              ;MOVE IT TO 'Y' TEMPORARILY
1410   00B9   20 C3 00        JSR  INPTLA      ;GET MSB OF BRANCH ADDRESS
1420   00BC   95 09          STA  PTAH,X       ;STORE IT IN HI ORDER PNTR
1430   00BE   94 08          STY  PTAL,X       ;STORE LSB OF BRANCH ADDRESS
1440   00C0   4C 8E 00        JMP  NXT
1450   00C3                ;
1460   00C3                ;SUBROUTINE TO INCREMENT POINTER
1470   00C3                ;AND LOAD ACCUMULATOR FROM THE ADDRESS
1480   00C3                ;HELD BY THE POINTER
1490   00C3                ;
1500   00C3   F6 08     INPTLA INC PTAL,X      ;INC LOW ORDER BYTE OF PNTR
1510   00C5   D0 02          BNE  INPT2        ;IF THERE IS A CARRY
1520   00C7   F6 09          INC  PTAH,X       ;THEN INC HI ORDER BYTE
1530   00C9   A1 08     INPT2 LDA  (PTAL,X)    ;LOAD ACC FROM INDIRECT
1540   00CB   60             RTS              ;POINTER AND RETURN
1550   00CC                ;
1560   00CC                ;
1570   00CC                FINISH .END
```

DEMO TUNE TABLE

This song table occupies locations $0110-$0318. Before loading this song table in by hand or by cassette, BE SURE TO set the stack pointer to $0A by entering $0A into location $00F2 (SP). This is important!!! Remember that RESET will reset the stack to $FF. By the way, the name of the song is "Here's That Rainy Day" by Van Heusen.

Oh yes, so that the music program knows where the music table is, you must fill in the following data in the music program's pointer locations:

```
$0008  0F 01
$000A  CF 01
$000C  4F 02
$000E  CF 02
```

```
200   00 01 36 07 00 01 36 0F 00 01 36 08 56 20 5B 10
210   36 30 39 20 51 30 44 10 39 07 00 01 39 0F 00 01
220   40 07 00 01 44 07 00 01 44 0F 00 01 44 07 00 01
230   36 20 39 20 48 10 44 08 C8 4C 20 51 40 40 10 2B
240   08 A4 AB B6 C0 44 88 CF 01 3F 3D 3F BF 7E 3C B9
250   36 10 B9 C8 D1 DB AD AD B0 B6 C0 B9 51 08 F9 80
260   C8 C0 B9 B6 B0 AB AB 2D 17 00 01 2D 08 30 10 B9
270   3C 08 B0 B6 B9 3C 10 C0 C0 BC 39 08 B6 B3 B0 39
280   07 00 01 39 0F 00 01 39 07 00 01 40 07 00 01 40
290   0F 00 01 40 08 36 10 B9 BC C0 DB AD AD B0 B6 C0
2A0   B9 51 08 F9 48 07 00 01 48 0F 00 01 40 08 BC B6
2B0   B3 B0 2B 10 AD 30 28 36 08 B9 C0 36 20 E0 48 10
2C0   40 08 C4 C8 CB A4 AB B6 C0 39 90 4F 02 A9 3C 3D
2D0   00 30 24 10 00 40 80 80 80 00 38 51 08 48 07
2E0   00 01 48 0F 00 01 48 07 00 01 48 07 00 01 48 0F
2F0   00 01 48 08 00 30 48 10 00 40 80 56 07 00 01 56

300   0F 00 01 56 07 00 01 56 07 00 01 56 0F 00 01 56
310   08 00 40 80 80 00 C0 CF 02
```

```
110   48 07 00 01 48 0F 00 01 48 07 00 01 48 08 BC AD
120   24 07 00 01 24 10 2B 08 AB 28 20 48 07 00 01 48
130   0F 00 01 48 07 00 01 48 08 B9 B0 A8 2B 38 00 08
140   22 0F 00 01 22 0F 00 01 22 08 B6 B3 B0 24 10 3C
150   08 B9 36 10 39 08 B6 28 07 00 01 28 0F 00 01 28
160   07 00 01 28 08 C0 BC B9 2B 40 48 07 00 01 48 0F
170   00 01 48 07 00 01 48 08 BC AD 24 07 00 01 24 10
180   28 08 AB 28 20 48 07 00 01 48 0F 00 01 48 07 00
190   01 48 08 B9 B0 A8 2B 38 00 08 20 07 00 01 20 0F
1A0   00 01 20 07 00 01 20 08 B6 AB A8 24 10 39 08 B6
1B0   30 20 48 07 00 01 48 0F 00 01 48 07 00 01 48 08
1C0   B9 B0 AB 36 28 00 98 0F 01 40 40 40 C0 C0 40 00
1D0   56 20 5B 10 36 30 39 20 51 30 44 10 30 10 36 08
1E0   B9 40 10 C8 36 17 00 01 36 08 44 20 48 30 4C 10
1F0   51 37 00 01 51 08 30 07 00 01 30 0F 00 01 30 07
```

4

Jim Adams
17272 Dorset
Southfield, MI 48075

Did you ever wish you had a stack for data storage which wasn't messed up by interrupts and subroutine calls? How about a stack where you could easily get at the first thing put on instead of the last? These circular list (or stack) processing subroutines perform the pointer, data and counter manipulations to reduce the above functions to subroutine calls.

A circular list is a block of memory which wraps around at the ends (figure 1). The last slot and the first slot are next to each other. The subroutines use four pieces of information about each list. The number of available slots tells when to wrap around and when the list is full or empty. The current top and next bottom point to the data. This information is kept with each list in the order shown in figure 2.

To initialize a block of memory to be used as a list put the address of the list in $EC (low, high) and the number of slots (n) you want in $EA, then JSR INL. Use JSR INLO when n is in register A instead of in $EA. The number of slots will be set to n and the other three parameters will be set to 0.

To add data to the list, put the address of the list in $EB, $EC and the data in $EA, then JSR ATL or JSR ABL. Or you can put the data in register A and JSR ATLO or JSR ABLO. If the list is full the V bit will be set, the list will re-main unchanged and $EA will contain the data. If the list is not full the V bit will be clear, $EA and register A will contain the data, N and Z will reflect the value of the data, and the data will be added to the list.

To remove data from the list, put the address of the list in $EB, $EC then JSR RBL or JSR RTL. If the list is empty the V bit will be set. If the list is not empty V will be clear, $EA and register A will contain the data, N and Z will reflect the value of the data and the data will be removed from the list.

This version restricts the list and its parameters to a page so the maximum number of slots in any list is $FC. If you ask for more slots the parameters will be overwritten with data. If you ask for more slots than remain to the end of a page then the extra slots are at the beginning of the same page, not the next page. The V bit is set with BIT $1A09. This is a location in the KIM monitor containing a $40. To use the subroutines with SYM or AIM change $1A09 to any location where bit 6 is set (e.g., to an RTS location). Location $FC is used for temporary storage.
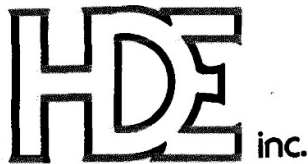
Circular lists can be used as first-in first-out buffers for asynchronous data transmission, queue storage, breadth-first search storage, failure sequence analysis, order preserving sorts, fixed sequence delays among other things. They can also be used like the 650X stack as first-in last-out buffers for depth-first search storage and reentrant subroutine storage.

```
0020  2000              ;ZERO PAGE REGISTERS TO FOLLOW
0025  2000                    *=$0
0030  0000       DATA   *=*+1          ;THIS LOCATION HOLDS THE NUMBER
0035  0001       ;OF SLOTS FOR SUBROUTINE 'INL' OR DATA TO
0040  0001       ;BE TRANSFERED TO LIST FOR 'ABL' OR 'ATL'
0045  0001       ;OR DATA TAKEN FROM LIST BY 'RBL' OR 'RTL'.
0050  0001
0055  0001       LPTR   *=*+2          ;ADDRESS LOW, HIGH OF LIST
0060  0003
0064  0003       TEMP   =$00FC         ;TEMPORARY SAVE LOCATION
0065  0003       ;THE LIST SETUP IN MEMORY IS:
0070  0003
0075  0003       ;      LIST    SLOTS AVAILABLE    1ST BYTE
0080  0003       ;              NUMBER USED        2ND BYTE
0085  0003       ;              CURRENT TOP        3RD BYTE
0090  0003       ;              NEXT BOTTOM        4TH BYTE
0095  0003       ;              1ST SLOT           5TH BYTE
0100  0003       ;                 .                  .
0105  0003       ;                 .                  .
0110  0003       ;              LAST SLOT          #AVAIL/4TH BYTE
0115  0003       ;
0120  0003       ;USE OF (IND),Y ADDRESSING AND FOUR BYTE OVERHEAD
0125  0003       ;MEANS MAX NUMBER OF SLOTS PER LIST IS $FC.
0130  0003
0135  0003       ;ENTRY POINTS;
0140  0003       ;      INLO    INITIALIZE LIST, # SLOTS IN 'A'
0145  0003       ;      INL     INITIALIZE LIST, # SLOTS IN 'DATA'
0150  0003       ;      ATLO    CONTENTS OF 'A' TO TOP OF LIST
0155  0003       ;      ATL     CONTENTS OF 'DATA' TO TOP OF LIST
0160  0003       ;      ABLO    CONTENTS OF 'A' TO BOTTOM OF LIST
0165  0003       ;      ABL     CONTENTS OF 'DATA' TO BOTTOM OF LIST
0170  0003       ;      RBL     CONTENTS OF BOTTOM OF LIST TO 'A' AND 'DATA'
0175  0003       ;      RTL     CONTENTS OF TOP OF LIST TO 'A' AND 'DATA'
0180  0003
0185  0003       ;STATUS:  V BIT IS SET IF ATTEMPT IS MADE TO ADD TO A
0190  0003       ;              FULL LIST OR REMOVE FROM AN EMPTY ONE.
0195  0003       ;         V BIT IS CLEAR IF SUCESSFUL TRANSFER  OCCURS.
0200  0003
0205  0003                    *=$2000
0210  2000
0215  2000  85 00   INLO   STA DATA       ;INITIALIZE LIST
0220  2002  A0 00   INL    LDY #0         ;C(DATA) TO
0225  2004  A5 00          LDA DATA       ;NUMBER OF SLOTS USED
0230  2006  91 01          STA (LPTR),Y
0235  2008  A9 00          LDA #0         ;ZERO TO...
0240  200A  C8            INY
0245  200B  91 01          STA (LPTR),Y   ;...NUMBER USED
0250  200D  C8            INY
0255  200E  91 01          STA (LPTR),Y   ;...CURRENT TOP
0260  2010  C8            INY
```

```
0265   2011   91 01            STA (LPTR),Y    ;...NEXT BOTTOM
0270   2013
0275   2013   85 00     ATLO   STA DATA        ;ADD TO TOP OF LIST
0280   2015   20 77 20  ATL    JSR FULL        ;SLOTS AVAILABLE?
0285   2018   70 4E            BVS END2        ;...EXIT IF NOT
0290   201A   20 BB 20         JSR POINT       ;YES, POINT TO
0295   201D   A5 00     PUT    LDA DATA        ;SLOT AND STUFF
0300   201F   4C 63 20         JMP END1        ;DATA
0305   2022
0310   2022   20 69 20  RBL    JSR EMPTY       ;REMOVE FROM BOTTOM
0315   2025   70 41            BVS END2        ;EXIT IF EMPTY
0320   2027   C8              INY
0325   2028   20 BB 20         JSR POINT       ;POINT TO SLOT
0330   202B   B1 01     GET    LDA (LPTR),Y    ;AND GET CONTENTS
0335   202D   85 00            STA DATA        ;TO 'DATA'
0340   202F   4C 65 20         JMP END3
0345   2032
0350   2032   20 69 20  RTL    JSR EMPTY       ;REMOVE FROM TOP
0355   2035   70 31            BVS END2        ;EXIT IF EMPTY
0360   2037   20 A1 20         JSR POINT1      ;POINT TO SLOT
0365   203A   20 2B 20         JSR GET         ;AND GET CONTENTS
0370   203D   A0 02            LDY #2          ;UPDATE CURRENT TOP
0375   203F   D0 10            BNE END         ;(UNCONDITIONAL)
0380   2041
0385   2041   85 00     ABLO   STA DATA        ;ADD TO BOTTOM
0390   2043   20 77 20  ABL    JSR FULL        ;SLOT AVAILABLE?
0395   2046   70 20            BVS END2        ;EXIT IF NOT
0400   2048   C8              INY              ;YES,
0405   2049   20 A1 20         JSR POINT1      ;POINT TO SLOT
0410   204C   20 1D 20         JSR PUT         ;AND STUFF DATA
0415   204F   A0 03            LDY #3          ;THERE
0420   2051
0425   2051   84 FC     END    STY TEMP
0430   2053   A0 00            LDY #0          ;NUMBER OF SLOTS SAME
0435   2055   B1 01            LDA (LPTR),Y    ;AS POINTER PLUS 1?
0440   2057   A4 FC            LDY TEMP
0445   2059   18              CLC
0450   205A   F1 01            SBC (LPTR),Y    ;IF SO,
0455   205C   F0 05            BEQ END1        ;RESET POINTER
0460   205E   B1 01            LDA (LPTR),Y    ;OTHERWISE
0465   2060   38              SEC              ;DECREMENT POINTER
0470   2061   69 00            ADC #0
0475   2063   85 01     END1   STA LPTR
0480   2065   A5 00     END3   LDA DATA
0485   2067   B8              CLV
0490   2068   60        END2   RTS
0495   2069
0500   2069                   ;SUBROUTINES TO CHECK LIST FULL OR EMPTY.
0505   2069                   ;V FLAG IS SET WHEN ANSWER IS "YES".
0510   2069                   ;IF ANSWER IS NO, SUBROUTINES INCREMENT
0515   2069                   ;OR DECREMENT NUMBER OF SLOTS USED IN
0520   2069                   ;ANTICIPATION OF A 'PUT' OR 'GET'.
0525   2069                   ;AT RTS, 'Y' POINTS TO CURRENT TOP POINTER.
0535   2069   A0 01     EMPTY  LDY #1          ;GET NUMBER OF
0540   206B   B1 01            LDA (LPTR),Y    ;SLOTS USED
0545   206D   F0 11            BEQ EMPTY1      ;BRANCH IS NONE
0550   206F   18              CLC              ;DECREMENT
0555   2070   E9 00            SBC #0          ;NUMBER USED
0560   2072   91 01     OUT    STA (LPTR),Y
0565   2074   C8              INY              ;TO CURRENT TOP POINTER
0570   2075   B8              CLV
0575   2076   60              RTS
0580   2077
0585   2077   A0 00     FULL   LDY #0          ;SLOTS AVAILABLE
0590   2079   B1 01            LDA (LPTR),Y    ;SAME AS
0595   207B   C8              INY              ;SLOTS USED?
0600   207C   51 01            EOR (LPTR),Y
0605   207E   D0 04            BNE INCR        ;BRANCH IF NOT
0610   2080   2C 09 1A  EMPTY1 BIT $1A09       ;SET V FLAG
0615   2083   60              RTS
0620   2084   A9 00     INCR   LDA #0          ;INCREMENT
0625   2086   38              SEC              ;NUMBER USED
0630   2087   71 01            ADC (LPTR),Y
0635   2089   D0 E7            BNE OUT         ;(UNCONDITIONAL)
0640   208B
0645   208B                   ;SUBROUTINES TO POINT TO LIST DATA
0650   208B
0655   208B   B1 01     POINT  LDA (LPTR),Y    ;POINTS TO SLOT 1
0660   208D   D0 0D            BNE NOSET       ;BRANCH IF NOT
0665   208F   84 FC            STY TEMP        ;YES, SAVE INDEX,
0670   2091   A0 00            LDY #0          ;GET NUMBER OF
0675   2093   B1 01            LDA (LPTR),Y    ;SLOTS MINUS 1
0680   2095   18              CLC              ;(LAST SLOT)
0685   2096   E9 00            SBC #0          ;THIS 'DECREMENTS'
0690   2098   A4 FC            LDY TEMP        ;THE POINTER
0695   209A   D0 03            BNE OVER
0700   209C   18        NOSET  CLC              ;DECREMENT
0705   209D   E9 00            SBC #0          ;POINTER
0710   209F   91 01     OVER   STA (LPTR),Y
0715   20A1   B1 01     POINT1 LDA (LPTR),Y    ;GET POINTER
0720   20A3   18              CLC              ;AND ADD
0725   20A4   69 04            ADC #4          ;OFFSET TO
0730   20A6   A8              TAY              ;FIRST SLOT
0735   20A7   60              RTS
```

6

PONG SOUND EFFECTS

Those of you who have purchased The First Book of KIM
may have noticed the Ping Pong program on page 95. Did
you also notice that something was missing? If you enter
the bytes 20 59 03 EA starting at address 02E9,   enter
the bytes 20 73 03 starting at address 031A,   rig up
PAO for audio output, and add the modification given below,
then the program will have BEEP, BOOP, and ZONK sound
effects much like the commercial versions.

```
****                  .LOC    00,89
00B9  XX        MEMY  XX,
008A  XX        CNT   XX,
008B  XX        PNT   XX,

****                  .LOC    03,59
0359  A5 84     MBEP  LDA Z   84,
035B  F0 13           BEQ R   RTN     NO SOUND IF GAME OVER
035D  08              PHP
035E  A9 02           LDA I   02,     MISSED THE BALL BEEP
0360  85 89           STA Z   MEMY    FREQUENCY COMPENSTTION
0362  A9 80           LDA I   80,     FREQUENCY
0364  85 8A           STA Z   CNT     DURATION
0366  20 BA 03        JSR A   ENTN    PRODUCE NOTE
0369  A9 FF           LDA I   FF,     NO BEEP, OR BOOP AFTER A ZONK
036B  85 8B           STA Z   PNT
036D  A5 84           LDA Z   84,
036F  28              PLP
0370  A2 04     RTN   LDX I   04,
0372  60              RTS

0373  E6 8B     HBEP  INC Z   PNT     CHECK FOR LAST MOVE A ZONK
0375  F0 21           BEQ R   BACK    YES
0377  A5 84           LDA Z   84,     HIT, OR SERVE BEEP, GET DIRECTION
0379  F0 1D           BEQ R   BACK    NO SOUND IF GAME OVER
037B  30 0E           BMI R   LBEP    LEFT, OR RIGHT?
037D  A9 08           LDA I   08,
037F  85 89           STA Z   MEMY    FREQUENCY COMPENSATION
0381  85 8A           STA Z   CNT     DURATION
0383  A9 20           LDA I   20,     FREQUENCY
0385  20 BA 03        JSR A   ENTN
0388  4C 98 03        JMP A   BACK

038B  A9 04     LBEP  LDA I   04,
038D  85 89           STA Z   MEMY
038F  A9 08           LDA I   08,
0391  85 8A           STA Z   CNT
0393  A9 40           LDA I   40,
0395  20 BA 03        JSR A   ENTN

0398  A5 84     BACK  LDA Z   84,     GET DURATION FOR ORIGINAL REVERSE
039A  18              CLC             AND CLEAR CARRY
039B  60              RTS

                                      TONE GENERATOR LOOP
                                      PARAMS:
                                      A=FREQUENCY
                                      MEMY=FREQUENCY DURATION COMPENSATION
                                      CNT=DURATION
039C  AA        CLK   TAX             MOVE FREQUENCY TO X
039D  EE 00 17        INC A   UPA     TOGGLE OUTPUT PIN 14
03A0  CA        DLX   DEX             DECREMENT X
03A1  F0 07           BEQ R   COMP    REPEAT TIMES COMPENSATION IF ZERO
03A3  EA              NOP             WAIT OTHERWISE
03A4  EA              NOP
03A5  EA              NOP
03A6  EA              NOP
03A7  EA              NOP
03A8  D0 F6           BNE R   DLX     DELAY FOR FREQUENCY
03AA  88        COMP  DEY             DECREMENT Y (FREQUENCY COMPENSATION)
03AB  F0 12           BEQ R   DUR     CHECK DURATION IF ZERO
03AD  EA              NOP             LOOP TIMING
03AE  EA              NOP
03AF  EA              NOP
03B0  EA              NOP
03B1  D0 00           BNE R   ON
03B3  D0 E7     ON    BNE R   CLK     KEEP TONE GOING
                                      ENTRY FOR REST
03B5  A2 00     RDUR  LDX I   00,     INITIALIZE PORT FOR SILENCE.   USE A14.
03B7  4C BC 03        JMP A   DUR1
                                      ENTRY FOR NOTE
03BA  A2 01     ENTN  LDX I   01,     INITIALIZE PORT FOR NOTE.   USE A14.
03BC  8E 01 17  DUR1  STX A   UPAD
03BF  C6 8A     DUR   DEC Z   CNT     DECREMENT DURATION
03C1  F0 04           BEQ R   DONE    RETURN IF ZERO
03C3  A4 89           LDY Z   MEMY    GET FREQUENCY COMPENSATION
03C5  D0 D5           BNE R   CLK     KEEP TONE GOING YET
03C7  60        DONE  RTS             RETURN
```

8

BASEBALL FOR THE KIM   by Robert Leedom

(Editors note-Bob Leedom, author of Hexpawn, pre-
sents another real cute diversion for the basic
KIM. Stuff like this still really excites me. I
don't usually like to publish hex dumps because
they are so frustrating to find your way through
them, but for those of you who want to see what
makes Baseball tick, you can get copies of the
listing-see the ad for User Notes cassettes.)

Copyright April '79 by Robert Leedom

     A video style action game for the KIM-1, which
uses the on board LED displays in three ways.

     1. You see the windup, the pitch (one of six)
and the swing.

KEY	PITCH
0	SLOW BALL
1	FAST BALL
2	UP CURVE
3	DOWN CURVE
4	RISER
5	SINKER

the bat

the windup
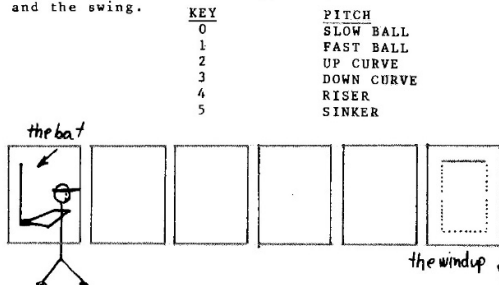
     2. You see the hit (if the pitcher was able
to get the ball in the strike zone, and if the
batters timing was good enough)-in this case, an
out-of-park home run:

the
swing

the ball
leaving
the park
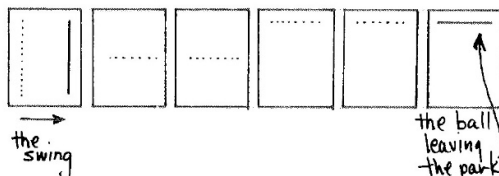
     ...but there are also six kinds of hits!

     3. You see everything you need to know about
the game's progress: naturally, you'll see the um-
pire's calls and score, but you'll also see, just
before each pitch (or at the touch of the 'PC' key
during windup), a compact status display. The
score may be seen during windup by pressing the
'+' key.

team at   seventh   # outs   men on   #balls   #strikes
bat   inning   first and
                              second
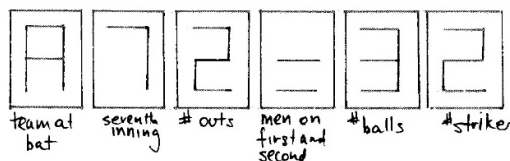
     The game can be played as nine-innings worth
of batting practice against KIM's pitching, or as
a two player game. In batting practice, the 'team
at bat' display will be blanked (of course, if the
two of you are tied at the end of nine innings, the
game will go into extra innings (E on the inning #
display) until there's a winner!). The KIM key-
board serves as the input for pitch selection and
for batting, but for about $3 worth of Radio Shack
switches, you can "remote" the pitchers' and/or
the batters controls! And only two data words are
necessary to support these changes.

Control Word	Value of Control Word	Batting and Pitching inputs
BATTER (loc. $002B)	Positive or Zero *	Batter uses Keyboard ("B"-key)
	Negative	Batter uses remote pushbutton switch.
PITCHER (loc. $002C)	Positive	Pitcher uses keyboard (keys 0-5)
	Zero *	Computer pitches
	Negative	Pitcher uses remote rotary and pushbutton switches

*VALUE SET ON PROGRAM TAPE

     KIM gives you a slight edge-if you're quick
enough to pass up a fourth wild pitch for a walk,
all your baserunners will advance. But the qual-
ity of pitching's pretty good-be on your toes!

REMOTE PITCH/BAT FOR KIM-1 BASEBALL

S1 SWITCH POSITION	SELECTED PITCH
1 , 7	SLOW BALL
2 , 8	FAST BALL
3 , 9	UP CURVE
4 , 10	DOWN CURVE
5 , 11	RISER
6 , 12	SINKER
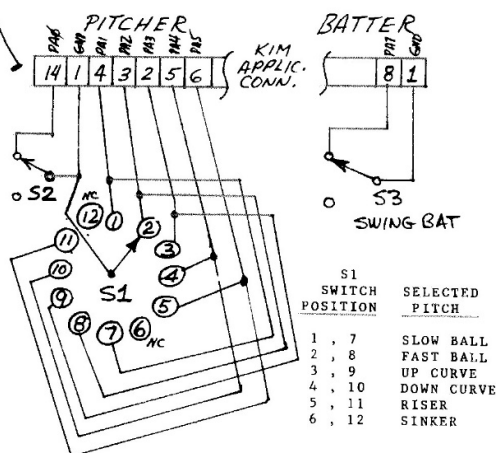
PARTS LIST

S1   single pole, 12 position switch (Radio Shack
      275-1385, $.99)

S2, S3   SPST switch, normally closed (Radio
      Shack 275-1548, 5 for $2.49)

     I mounted S1 and S2 in what I call the Pitch-
ers' Wand, a Head & Shoulders plastic shampoo bot-
tle. (The neck of the bottle is the handle, and
S1 is where the label was. S2 can be easily
flicked with the thumb while holding the handle.)
The reason for the 12-position PITCH SELECT switch
is to make it harder for the batter to memorize
switch positions and listen to clicks-for example,
one click from a slow ball can either be a fast-
ball or a sinker.

     S3 was mounted in a small plastic pill bottle
that fits easily into the hand.

10

Start Baseball at $0200. To restart the game,
hit GO during the windup or during the the endgame
display (six baseballs).

0000-03FF & 1780-17E6

```
000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
020   00 00 00 00 00 00 00 00 08 40 01 01 00 FD 00 00
030   00 FE 01 FF 02 03 A4 9F AF B5 BB C1 A9 8D 87 6F
040   75 00 01 40 41 08 09 48 49 3A 1A 2A 2A 25 2F 3A
050   1A 2A 2A 25 2F 20 20 24 24 29 2E 2F AA AA AF A5
060   A9 AB FA 5A FF 55 F9 5B 29 2D 2F 2A 2F 2B 2B 00
070   00 78 5C 00 00 00 00 50 1C 54 6D 7C 77 38 38 00
080   00 6D 78 50 00 00 00 39 77 3E 3D 76 78 37 06 6D
090   6D 79 5E 6E 79 50 5C 1C 78 00 00 00 5C 1C 78 73
0A0   3F 73 3E 73 00 71 3F 3E 38 00 76 3F 37 79 50 71
0B0   38 6E 5C 1C 78 6D 04 54 3D 38 79 5E 5C 1C 7C 38
0C0   79 7B 50 04 73 38 79 5E 7C 38 00 73 38 63 63 63
0D0   63 63 63 46 13 D0 04 A9 20 85 13 20 63 01 A5 13
0E0   85 24 A9 30 85 1F A0 07 20 17 01 88 D0 FA 60 73
0F0   18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

100   84 26 A0 05 B1 26 99 1F 00 88 10 F8 A0 C0 20 17
110   01 20 17 01 88 D0 F7 84 25 A0 00 A9 7F 8D 41 17
120   A2 09 84 FC B9 1F 00 20 4E 1F C8 C0 06 90 F3 20
130   3D 1F 20 6A 1F A4 25 60 EA 85 0B B5 49 48 48 29
140   FB EA 85 1D 4A 85 07 68 29 03 85 F4 68 4A 4A 29
150   03 85 F3 B5 5C A2 03 48 29 03 95 EF 68 4A 4A CA
160   10 F5 60 A9 00 A2 05 95 1F CA 10 FB A6 0B B4 EF
170   B9 27 00 95 1F A4 1D 60 A2 00 46 1A 90 1C A6 00
180   B5 11 69 00 95 11 E0 0B D0 10 A5 01 C9 09 90 0A
190   A5 1B C5 1C B0 04 86 1E D0 06 88 10 DD 8A F0 23
1A0   A5 1B 20 80 17 A6 2C F0 11 84 6F 85 70 A5 1C 20
1B0   80 17 84 73 85 74 A0 6F D0 06 84 75 85 76 A0 75
1C0   20 00 01 60 84 09 98 D8 C5 07 90 04 A5 1D E5 09
1D0   A2 08 86 13 C5 07 90 0A E5 07 48 A5 12 05 13 85
1E0   12 68 46 07 46 13 D0 EC A6 12 B5 2D 60 60 17 A4
1F0   AA 17 A4 AA 70 1F 70 1F 70 00 00 00 98 00 35 00
```

```
200   AD 06 17 85 15 F8 A9 0B 85 00 A9 00 85 1E 85 01
210   85 1B 85 1C A5 00 49 01 85 00 C9 0A D0 07 18 A5
220   01 69 01 85 01 A9 00 85 02 85 1A A9 00 85 04 85
230   05 A5 1A 09 08 85 1A A9 00 85 12 85 13 A9 05 85
240   0B 20 D3 03 A5 2C F0 24 A5 2C 10 10 20 C4 17 4A
250   90 12 29 1F A2 00 4A 90 32 E8 10 FA 20 2F 01 C9
260   06 AA 90 27 20 D3 00 20 91 17 B0 DC 20 AD 17 29
270   7F 09 20 85 08 20 D3 00 20 91 17 C6 0B D0 F6 20
280   AD 17 29 07 C9 06 90 02 E9 06 AA 20 AD 17 29 07
290   C9 03 B0 0D A8 8A D0 03 98 D0 06 E8 E8 E8 E8 E8
2A0   E8 20 3B 01 A9 FF 85 03 A9 30 85 0A 20 63 01 20
2B0   17 01 A6 03 10 18 C9 11 F0 09 A5 2B 10 10 20 C4
2C0   17 10 0B A9 06 85 0A A5 1F 0A 30 41 85 03 A5 1F
2D0   29 49 05 0A 85 1F 88 D0 D6 A5 03 10 2D C6 0B 10
2E0   CB A5 EF C9 02 F0 23 E6 04 A6 04 BD E7 1F 85 80
2F0   A0 7B 20 00 01 A5 04 C9 04 D0 6E A0 00 20 78 01

300   A5 1E F0 03 4C 9C 03 4C 2B 02 4C 6C 03 20 C4 01
310   85 06 18 69 0F AA F8 A9 01 20 39 01 20 63 01 A9
320   06 85 1F 20 17 01 88 D0 FA E6 0B A5 0B C9 06 D0
330   EB A6 06 B4 39 20 00 01 A9 00 85 09 20 AD 17 29
340   04 F0 02 E6 09 A4 06 10 B4 C8 D0 03 4C CD 17 A6
350   09 D0 03 C8 30 0D B4 3D 20 00 01 A6 09 86 06 F0
360   D2 D0 22 A5 05 C9 02 D0 03 4C 37 02 E6 05 A6 05
370   BD E7 1F 85 86 A0 81 20 00 01 A5 05 C9 03 D0 E9
380   A0 93 20 00 01 E6 02 A6 02 BD E7 1F 85 9A A0 99
390   20 00 01 A5 02 C9 03 F0 03 4C 2B 02 20 A0 01 A5
3A0   1E D0 26 A5 01 C9 09 B0 07 A5 2C D0 13 4C 1E 02
3B0   A5 2C F0 15 A5 00 C9 0B F0 09 A5 1B C5 1C 90 09
3C0   4C 14 02 A5 1B C5 1C F0 F7 20 91 17 A0 CD 20 00
3D0   01 F0 C9 A2 05 B4 00 B9 E7 1F 95 0C CA 10 F6 A5
3E0   01 C9 0A 90 04 A9 79 85 0D A5 1A 29 07 AA B5 41
3F0   85 0F A5 2C D0 04 A9 00 85 0C A0 0C 4C 00 01 B3
```

```
1780   48 4A 4A 4A 4A 4A AA BC E7 1F 68 29 0F AA BD E7 1F
1790   60 20 17 01 C9 20 D0 03 20 D3 03 C9 1B D0 03 20
17A0   A0 01 C9 19 D0 05 68 68 4C 05 02 38 60 38 D8 A5
17B0   15 65 16 65 19 85 14 A0 04 B9 14 00 99 15 00 88
17C0   10 F7 F8 60 A9 00 8D 01 17 AD 00 17 60 A5 1A 0A
17D0   05 02 C9 1B F0 04 C9 14 D0 09 A0 C7 20 00 01 84
17E0   1A E6 02 4C 85 03 FF 00 00 00 00 00 00 00 00 00
```

11

LANGUAGE LAB

basic

AN EDITOR FOR MICROSOFT BASIC

from Sean McKenna
64 Fairview Ave
Piedmont CA 94610

(Editors note: How many times have you had to reenter a whole line in a Basic program just to correct one small typo? No need for that aggravation any longer. Sean is generously sharing the editor portion of his Basic Enhancements Package with us).

Character oriented editor for KIM BASIC:

1) Set the BASIC I-O calls and the I-O calls to your system as indicated in the listing.

2) The delimiter is set by the contents of $02CB (it is a backslash although appears as a ¢ in the program due to printer strangeness)

3) The command is set by the contents of $0214 to be ")" in the first position of an input line.

4) Before making any use of the program, FIXFLG and INDEX ($00ED and $00E3) must be initialized to $00.

5) To edit a program type the command ")" followed by any valid line number, backslash, material to be deleted, backslash, material to be added, backslash, CR. If the edit goes well you will get the usual "OK" prompt. If the string you specified for editing was not found you will get a "MATCH ERROR". If the addition asked for caused a line overflow you will get a "TOO LONG ERROR". If you neglect to include all three delimiters you will get a "SYNTAX ERROR". In all error situations the original line remains intact. You may delete material by not putting anything between the 2nd and 3rd delimiters. You may move a line be doing an edit on the line number itself and then deleting the original line. Line and character delete (@ and _) will operate as usual during an edit line input.

```
0060: 0200          FIXIN  ORG   $0200
0070: 0200          FIXFLG *     $00ED
0080: 0200          INDEX  *     $00E3
0090: 0200          PLACE  *     $00E8
0100: 0200          POINT  *     $00E1
0110: 0200          LENGTH *     $00E9
0120: 0200          LIMIT  *     $00E5
0130: 0200          TERMNL *     $0017
0140: 0200          CRCONT *     $00E4
0150: 0200          YTEMP  *     $00E0
0160: 0200          BASBUF *     $001B
0170: 0200          MATCH  *     $0110
0180: 0200          BASERX *     $2321
0190:
0200:                 * * * * * * * * * * * * *
0210:               BASIC Input call from $2456 to here
0220:                 * * * * * * * * * * * * *
0230: 0200 24 ED          BIT   FIXFLG  IF fix flag set
0240: 0202 10 07          BPL   GETIN
0250: 0204 E6 ED          INCZ  FIXFLG  THEN clear it,
0260: 0206 A6 E8          LDXZ  PLACE   set X to end of line,
0270: 0208 A9 0D   CRRET  LDAIM $0D     and return to BASIC with a CR
0280: 020A 60     RETURN  RTS
0290:                 * * * * * * * * * * * * *
0300: 020B 20 C0 17 GETIN JSR   $17C0   Call to your own input routine
0310:                 * * * * * * * * * * * * *
0320: 020E C9 0D          CMPIM $0D     IF user input is not a CR
0330: 0210 D0 F8          BNE   RETURN  THEN return to BASIC
0340: 0212 A5 1B          LDAZ  BASBUF  ELSE IF first character in
0350: 0214 C9 29          CMPIM ')      BASIC input buffer is not ")"
0360: 0216 D0 F0          BNE   CRRET   THEN return to BASIC
```

12

OK
LIST

10 abcdefghijklmnop
20 qrstuvwxyz
30 All the king's horses and all the king's men
40 Couldn't put Humpty together again
OK

(1)) 10¢defg¢DEFG¢

OK

(2)) 30¢king's ¢¢

OK

LIST

10 abcDEFGhijklmnop
20 qrstuvwxyz
30 All the horses and all the king's men
40 Couldn't put Humpty together again
OK

(3)) 40¢king's¢queen's¢

?MATCH ERROR
OK

(4)) 30¢king's¢queen's

?SYNTAX ERROR
OK

(5)) 30¢king's¢king's and duke's and bishop's and page's¢

?TOO LONG ERROR
OK

(6))30¢king's¢queen's¢

OK

(7))30¢30¢50¢

OK
LIST

10 abcDEFGhijklmnop
20 qrstuvwxyz
30 All the horses and all the queen's men
40 Couldn't put Humpty together again
50 All the horses and all the queen's men
OK

```
0370:  0218  C6 ED      DECZ   FIXFLG    ELSE set flags for input and
0380:  021A  C6 E3      DECZ   INDEX     output routines,
0390:  021C  86 E5      STXZ   LIMIT     save X for buffer check
0400:  021E  A9 00      LDAIM  $00       clear all counters
0410:  0220  AA         TAX
0420:  0221  A8         TAY
0430:  0222  85 E8      STAZ   PLACE
0440:  0224  85 E0      STAZ   YTEMP
0450:  0226  20 C3 02   FNDPOS JSR    DELIMI    find start of edit string
0460:  0229  D0 FB      BNE    FNDPOS
0470:  022B  86 E1      STXZ   POINT     and save X for end of routine
0480:  022D  20 C3 02   ESTRNG JSR    DELIMI    save edit string in match
0490:  0230  F0 06      BEQ    NEWSTG
0500:  0232  99 10 01   STAAY  MATCH
0510:  0235  C8         INY
0520:  0236  D0 F5      BNE    ESTRNG
0530:  0238  88         NEWSTG DEY       save length of edit string
0540:  0239  84 E9      STYZ   LENGTH
0550:  023B  C8         INY
0560:  023C  20 C3 02   NEXNEW JSR    DELIMI    save new string in match
0570:  023F  F0 06      BEQ    OVER
0580:  0241  99 10 01   STAAY  MATCH
0590:  0244  C8         INY
0600:  0245  D0 F5      BNE    NEXNEW
0610:  0247  A9 0D      OVER   LDAIM  $0D       set counter for output
0620:  0249  99 10 01   STAAY  MATCH
0630:  024C  A9 04      LDAIM  $04
0640:  024E  85 E4      STAZ   CRCONT
0650:  0250  A9 99      LDAIM  $99       put LIST token at start of buffer
0660:  0252  85 1B      STAZ   BASBUF
0670:  0254  A6 E1      LDXZ   POINT     put pointer in X
0680:  0256  D0 B0      BNE    CRRET     and return to BASIC with a CR
0690:
0700:
0710:  0258  A5 E4      COUNTR LDAZ   CRCONT    IF counter=0
0720:  025A  D0 0F      BNE    NOTHRU
0730:  025C  E6 E3      INCZ   INDEX
0740:  025E  A9 0D      LDAIM  $0D       THEN clear flag
0750:  0260  24 E9      BIT    LENGTH
0760:  0262  30 05      BMI    FOUND     check that string was found
0770:  0264  A2 AB      LDXIM  SAB
0780:  0266  4C 21 23   JMP    BASERX    "mismatch" error exit
0790:  0269  D0 07      FOUND  BNE    GOOUT
0800:  026B  C6 E4      NOTHRU DECZ   CRCONT    ELSE decrement counter
0810:  026D  60         RTS              and return
0820:             * * * * * * * * * *
0830:             * BASIC output call from $2A51 to here *
0840:             * * * * * * * * * *
0850:
0860:  026E  24 E3      FIXOUT BIT    INDEX     IF flag clear
0870:  0270  30 03      BMI    DOFIX
0880:             * * * * * *
0890:  0272  4C 17 10   GOOUT  JMP    $1017     Call your own output routine
0900:             * * * * * *
0910:  0275  C9 0D      DOFIX  CMPIM  $0D       ELSE IF CR
0920:  0277  F0 DF      BEQ    COUNTR    THEN go check counter
0930:  0279  C9 0A      CMPIM  $0A       ELSE IF LF
0940:  027B  F0 DB      BEQ    COUNTR    THEN ditto
0950:  027D  86 E1      STXZ   POINT     ELSE save X
0960:  027F  A4 E0      LDYZ   YTEMP     load indexes
0970:  0281  A6 E8      LDXZ   PLACE
0980:  0283  24 E9      BIT    LENGTH
0990:  0285  30 0C      BMI    NOMATC
1000:  0287  D9 10 01   CMPAY  MATCH     IF character match
1010:  028A  D0 07      BNE    NOMATC    THEN if length match
1020:  028C  C4 E9      CPYZ   LENGTH
1030:  028E  F0 0F      BEQ    EDIT
1040:  0290  C8         INY
1050:  0291  D0 02      BNE    CONTIN
1060:  0293  A0 00      NOMATC LDYIM  $00
1070:  0295  20 B2 02   CONTIN JSR    BUMP
1080:  0298  86 E8      CLOSE  STXZ   PLACE     save and restore registers
1090:  029A  84 E0      STYZ   YTEMP
1100:  029C  A6 E1      EROUT  LDXZ   POINT
1110:  029E  60         RTS
1120:  029F  E8         EDIT   INX
1130:  02A0  CA         DELETE DEX       do delete
1140:  02A1  C6 E9      DECZ   LENGTH
1150:  02A3  10 FB      BPL    DELETE
1160:  02A5  C8         INSERT INY       and insert
1170:  02A6  B9 10 01   LDAAY  MATCH
1180:  02A9  C9 0D      CMPIM  $0D       till done
1190:  02AB  F0 EB      BEQ    CLOSE
1200:  02AD  20 B2 02   JSR    BUMP
1210:  02B0  D0 F3      BNE    INSERT
1220:
1230:  02B2  95 1B      BUMP   STAZX  BASBUF    Inserts character in buffer
1240:  02B4  E8         INX              checking for overflow
1250:  02B5  E4 17      CPXZ   TERMNL    IF line too long
1260:  02B7  90 08      BCC    CLEAR
1270:  02B9  A9 00      LDAIM  $00       clear pending input line
1280:  02BB  85 1B      STAZ   BASBUF
1290:  02BD  A2 B7      LDXIM  $B7       "too long" error exit
1300:  02BF  4C CF 02   JMP    ERROR
1310:  02C2  60         CLEAR  RTS
1320:
1330:  02C3  E8         DELIMI INX       looks for delimiter and checks
1340:  02C4  E4 E5      CPXZ   LIMIT     for complete command
1350:  02C6  10 05      BPL    SYNERR
1360:  02C8  B5 1B      LDA2X  BASBUF
1370:  02CA  C9 5C      CMPIM  '¢
1380:  02CC  60         RTS
1400:  02CD  A2 10      SYNERR LDXIM  $10       "syntax" error exit
1410:  02CF  E6 E3      ERROR  INC2   INDEX     resets flags
1420:  02D1  E6 ED      INC2   FIXFLG
1430:  02D3  4C 21 23   JMP    BASERX    and returns to BASIC via error entry
```

It boggles the mind to think about what Sean
could do if he had a source listing to
work from..... Eric

13

SOME BASIC HINTS FROM

Bob Kurtz
Micro-Z Electronic Systems
Box 2426
Rolling Hills CA 90274

A. Several articles have been written about programs in BASIC that provide a word processor or text editor capability. Unfortunately, BASIC uses the comma (,) and the colon (:) as commands, and if they appear in the text that you are writing – BASIC will immediately reply with an error statement. The following POKE instructions, placed early in your program, will de-activate these commands:

    XXXX POKE 11031,34:POKE 11035,34

At the end of the program, insert the following instructions to put BASIC back the way it was:

    ZZZZ POKE 11031,58:POKE 11035,44

B. Some versions of KIM BASIC will not execute the SPC command properly–but will execute it the same as the TAB command. The reason is that there was a CLC instruction improperly located. Make the following changes:

298B	18	298E	C9
298C	F0	298F	2C
298D	67		

This will permit the SPC(X) instruction to space over X units from the last location on the terminal–not form the left margin.

BASIC NOTE

from Sean McKenna
64 Fairview Ave
Piedmont CA 04610

In issue #31 of Dr. Dobb's Journal there was a machine language renumber program for PET BASIC. In the current issue (#36) there are notes indicating changes which allow the program to work for KIM BASIC. Relative to the other renumbering programs which have appeared in the notes this is a vast improvement: it changes decimal numbers in a line only *immediately* following a GOTO, GOSUB, THEN, or ON; it will accomodate a change from any size number to any other size without any special consideration such as leaving spaces, etc.; it is very FAST, naturally. The program also revealed some interesting aspects of BASIC's mysterious inner workings which may be useful in other contexts.

PRODUCT REVIEW   by the Editor

MICROSOFT BASIC ENHANCEMENTS

If you're bothered by the fact that Microsoft Basic doesn't have an automatic line number feature, a line editor, or a renumbering command, then you're in luck. Sean McKenna who shared his Basic auto-line number with us in issue #14 has come up with a dynamite mod package for the 9 digit Basic (will work with the package from Johnson Computer or Micro-Z).

This 1.25K package is written in machine code and includes the auto line number command, a re-number feature, a line editor, an append program capability and a controlled output (outputs 16 line and waits for user input). All in all, a really nice package that worked right the first time I loaded it in. By the way, getting this package interfaced to Basic was no sweat at all–since this package interfaces *itself* to Basic!

All that's necessary, once both programs are loaded,to start running at an address contained in the mod package. The mod package hooks itself into Basic and then jumps to Basics initialization routine. Really clean.

This mod package consists of a cassette, and a 20 page manual which includes a complete source listing (!). It sells for $20 (plus $1.50 S&H) and is available from Sean McKenna, 64 Fairview Ave., Piedmont CA 94610.

A fair price for some powerful Basic enhancements.

                                                Eric

focal

First of all, I want to thank Dave and Don Marsh from the 6502 Program Exchange (2920 Moana Ln., Reno NV 89509) for providing me with the source listing for their version of FCL-65E. The listing has been invaluable in getting all the mods set up for both versions of FOCAL (one from Aresco and the other from the Program Exchange).

By the way, both versions must be suitably modified as per issue #14 in order to use the modifications that will be presented. Program Exchange FCL-65E users need to move the start up message at line 00.00 to the top of the 8K block by moving the data at $35D4 through $35F3 to start at $3FE0.

In trying to cooordinate these mods across two versions of FOCAL, I've run across a zero-page usage problem in the Program Exchange versions. This version uses about 50 bytes of zero-page for terminal I/O. (According to the exchange, this was done to make FOCAL more portable between different machines). Anyhow, the long and short of it is – these I/O routines will have to be moved back into FOCAL to allow freer use of zero page.

Once the line 0.0 has been moved up to the top of the 8K block, the I/O routines from $00A9-$00DC can be moved to start at location $35D4. Of course, the internal references to OUT and IN will have to be changed to reflect the changes.

FOCAL ENHANCEMENT PACKAGE

The 'NOTES' is now distributing a very useful FOCAL enhancement package that will let you save and load complete FOCAL programs on cassette as well as lines or groups of lines and/or program variables. Commands may also be executed directly from cassette. The package was written by Joe Woodard. For ordering info, see the cassette software ad in this issue.

ADDING A CASSETTE INTERFACE AND A USER FUNCTION TO 6502 PROGRAM EXCHANGE'S FOCAL 65-E

by William C. Clements, Jr.
1489 51ST Ave East
Tuscaloosa Al 35404

The FOCAL language is really a good alternative to BASIC, at least for KIM users. Of course, it doesn't have everything. The features I missed the most were a cassette interface and the ability to execute a machine-language routine within a FOCAL program. This article shows how to add tape Load and Keep commands and how to implement a "user" function similar to that of TINY BASIC. The modifications apply to FOCAL-65 (V3D) for the KIM-1 as supplied by the 6502 Program Exchange.

/4

Listing 1 gives the code needed to add the cassette interface commands. I began it directly after the FOCAL interpreter, because I had moved the RAM allocation for program text and variables to another area. It can go anywhere in memory that you wish, with simple relocation and listing the addresses of routines KEEP and LOAD in FOCAL's command dispatch tables. The cassette Load command enters the regular KIM monitor at $1873, and the Keep command uses a Hypertape routine in my system; it's almost a necessity to use a cassette dump routine faster than KIM's, since the memory required to store the FOCAL program statements in ASCII form can be large.

The tape operations could have been done using the existing I-O handlers provided in FOCAL, but I preferred to use conventional commands. The form of the commands is L xx to load a file having hexadecimal i.d. "xx" and K xx to record a file with i.d. "xx" onto tape.

Readers who have program control of their tape recorders might want to use these commands inside a FOCAL program to manipulate tape files. I can only use them in the immediate execution mode, since I have to push buttons on the recorder. The KIM tape routines exit to loc. zero, which my code sets up with a jump instructin. Hitting the G key on the TTY after either tape operation is through will get you back into FOCAL. All starting and ending address for the tape files are automatically set by the routines, including the final address after loading a file.

The Keep routine uses Hypertape stored in my system at loc. $C400; the jump at location TAPOUT will need to be fixed by the user to suit his own system. The jump in loc. zero restarts FOCAL at its cold start, as that's the only way I can use it. If you want to get back into the middle of an executing FOCAL program, the jump at location JMPFOC and the data bytes at locations ADLOW and ADHIGH will have to be changed.

The "user" function works like the one in TINY BASIC; it allows user-supplied machine code to be executed as a FOCAL function. The FOCAL code to invoke it is S FUSR (a_1, a_2, a_3, a_4), where the a's are the four arguments. The first, a_1, must always be present because it is the address to which the program will jump to begin the user's code.

a_2, a_3, and a_4 are optional. a_2, if present, will be evaluated and the least significant eight bits stored in the accumulator before executing the user's code. a_3 and a_4, if present, are similarly evaluated and placed in the X and Y registers, respectively. Thus up to three bytes may be transmitted directly from the FOCAL program to the machine code (more of course can be transmitted in either direction by using FOCAL's version of PEEK and POKE, the FMEM function). The arguments can be constants, simple variables, or any other legal FOCAL expressions, and as such have decimal values.

As examples, the statement S FUSR(8192,0,16, 10) will cause a jump to location $2000 with zero in A, $10 in X, and $0A in Y. The statements

$$1.1 \ S \ A=100$$
$$1.2 \ S \ B=13$$
$$1.3 \ S \ FUSR(625-(A+B),,B,)$$

would jump to $200 with $0D in X. Note that there are always three commas, as FOCAL uses them to tell which argument is which. If you want the variable FUSR itself to have a numerical value after its execution such as FRAN or FABS do, you can have your machine code put that value into the floating accumulator FAC1 (locs. $80-83 - see p. 7 of the 6502 Program Exchange's listing of FOCAL 65-E). Your machine code must transfer control to loc. FPOPJ (in FOCAL) when it is ready to re-enter FOCAL, and it will return to the point in your FOCAL program where FUSR was invoked. Listing 2 gives the machine code needed for adding FUSR to FOCAL.

The changes required within the tables of the FOCAL interpreter to make it recognize K. L, and FUSR and to execute the codes in Listings 1 and 2 are given now. The format follows that of the original listing of FOCAL.

ARESCO	PROGR. EXCH		
$350B	$34F4	18	BYTE HFUSR
3527	3510	36	HBYTE FUSR
3543	352C	41	BYTE FUSR
3557	3540	4B	ASCII 'K'
3558	3541	4C	ASCII 'L'
356B	3554	36	HBYTE KEEP
356C	3555	36	HBYTE LOAD
357D	3566	04	BYTE KEEP
357E	3567	33	BYTE LOAD

```
0010  2000            ;CASSETTE INTERFACE AND USER FUNCTION MODS
0020  2000            ;FOR FOCAL FROM W. CLEMENTS 1979
0030  2000
0040  2000            ;KIM LOCATIONS
0050  2000            FACK   =$1FAC
0060  2000            ID     =$17F9
0070  2000            FREG   =$F1
0080  2000            INL    =$F8
0090  2000            SAL    =$17F5
0100  2000            SAH    =$17F6
0110  2000            EAL    =$17F7
0120  2000            EAH    =$17F8
0130  2000            LOADT  =$1873
0140  2000            VEB    =$17EC
0150  2000            HYPER  =$0200        ;ADDRESS OF HYPERTAPE ROUTINE
0160  2000
0170  2000            ;FOCAL LOCATIONS
0180  2000            FOCAL  =$2000
0190  2000            GSPNOR =$29A3        ;($29B1 IN ARESCO VERSION)
0200  2000            PBADR  =$31
0210  2000            VARBEG =$3E
0220  2000            INTGER =$2F85        ;($2F93 IN ARESCO VERSION)
0230  2000            HI     =$81
0240  2000            CHAR   =$2B
0250  2000            NXIARG =$2F7B        ;($2F89 IN ARESCO VERSION)
0260  2000
0270  2000            ;MOD LOCATIONS
0280  2000                   *=$0
0290  0000            JMPCOM *=*+2         ;JUMP VECTOR IN ZERO PAGE
0300  0002            NARGS  *=*+1         ;NUMBER OF ARGS IN USR
0310  0003            SAVA   *=*+1
0320  0004            SAVX   *=*+1
0330  0005
```

```
0340  0005
0350  0005                          ;FOCAL MODS START HERE
0360  0005                          *=$35EB
0370  35EB                          .OFF 2000
0380  35EB  20 A3 29   SUB    JSR GSPNOR    ;GET NEXT BLANK CHAR
0390  35EE  20 AC 1F          JSR PACK      ;CONVERT TO HEX AND STORE
0400  35F1  20 A3 29          JSR GSPNOR    ;REPEAT FOR
0410  35F4  20 AC 1F          JSR PACK      ;NEXT DIGIT
0420  35F7  A5 F8            LDA INL
0430  35F9  8D F9 17         STA ID         ;SET TAPE ID
0440  35FC  A9 4C            LDA #$4C       ;SETUP JUMP LOCATION
0450  35FE  85 00            STA JMPCOM      ;IN ZERO PAGE
0460  3600  A9 00            LDA #0         ;CLEAR STATUS REG
0470  3602  85 F1            STA PREG RTS   ;AND RETURN
0480  3604
0490  3604  20 EB 35   KEEP   JSR SUB       ;SET ID ETC
0500  3607  A5 31            LDA PBADR      ;SET KIM
0510  3609  8D F5 17         STA SAL        ;TAPE REGISTERS
0520  360C  A5 32            LDA PBADR+1
0530  360E  8D F6 17         STA SAH
0540  3611  A5 3E            LDA VARBEG
0550  3613  8D F7 17         STA EAL
0560  3616  A5 3F            LDA VARBEG+1
0570  3618  8D F8 17         STA EAH
0580  361B  A9 00    ADRLOW LDA #<FOCAL
0590  361D  85 01            STA JMPCOM+1   ;MAKE JUMP INSTR. A
0600  361F  A9 20    ADRHI  LDA #>FOCAL    ;RETURN TO COLDSTART
0610  3621  85 02            STA JMPCOM+2
0620  3623  4C 00 02 TAPOUT JMP HYPER
0630  3626
0640  3626  AD ED 17  ENLOAD LDA VEB+1     ;SET ADDRESS AT END OF
0650  3629  85 3E            STA VARBEG     ;PROGRAM TEXT
0660  362B  AD EE 17         LDA VEB+2
0670  362E  85 3F            STA VARBEG+1
0680  3630  4C 00 20 JMPFOC JMP FOCAL      ;RETURN TO FOCAL
0690  3633
0700  3633  20 EB 35   LOAD  JSR SUB       ;SET ID ETC
0710  3636  A9 26            LDA #<ENLOAD   ;MAKE JUMP POINT TO
0720  3638  85 01            STA JMPCOM+1   ;THE REST OF THE TAPE
0730  363A  A9 36            LDA #>ENLOAD   ;LOAD ROUTINE
0740  363C  85 02            STA JMPCOM+2
0750  363E  4C 73 18         JMP LOADT      ;READ THE CASSETTE
0760  3641
0770  3641                        ;NOW FOR THE 'USR' FUNCTION
0780  3641
0790  3641  A9 4C     FUSR   LDA #$4C       ;SET UP JUMP LOC.
0800  3643  85 00            STA JMPCOM
0810  3645  20 85 2F         JSR INTGER     ;GET FIRST ARG. IN FAC1
0820  3648  85 01            STA JMPCOM+1   ;REARRANGE LOW AND HIGH ORDER
0830  364A  A5 82            LDA M1+1       ;BYTES INTO JUMP LOCATION
0840  364C  85 02            STA JMPCOM+2   ;THAT WILL EXECUTE USER CODE
0850  364E  A9 00            LDA #0         ;ZERO THE ARG. COUNTER
0860  3650  85 02            STA NARGS
0870  3652  20 7A 36         JSR USRARG     ;EVALUATE AND SAVE HOWEVER MANY
0880  3655  84 03            STY SAVA       ;ARGUMENTS ARE LEFT
0890  3657  20 7A 36         JSR USRARG
0900  365A  84 04            STY SAVX
0910  365C  20 7A 36         JSR USRARG
0920  365F  A5 02            LDA NARGS
0930  3661  F0 0E            BEQ JMPUSR     ;JUMP TO USER'S CODE IF NO MORE ARGS
0940  3663  C9 01            CMP #$1
0950  3665  F0 0B            BEQ STAC       ;SET 'A'=ARG, IF ONE ARG LEFT
0960  3667  C9 02            CMP #$2
0970  3669  F0 09            BEQ STACX      ;SET 'A'=ARG1, 'X'=ARG2 IF TWO LEFT
0980  366B  C9 03            CMP #$3
0990  366D  F0 05            BEQ STACX      ;SET 'A'=ARG1, 'X'=ARG2, 'Y'=ARG3
1000  366F  A5 03     STAC   LDA SAVA       ;ARG1 IN 'A'
1010  3671  4C 00 00  JMPUSR JMP JMPCOM     ;GO DO USER'S CODE
1020  3674  A5 04     STACX  LDA SAVX
1030  3676  AA               TAX
1040  3677  4C 6F 36         JMP STAC
1050  367A
1060  367A  A5 2B     USRARG LDA CHAR       ;GET CURRENT CHARACTER
1070  367C  C9 2C            CMP #','        ;ANOTHER ARGUMENT?
1080  367E  F0 04            BEQ GETARG     ;YES, GO GET IT
1090  3680  C9 29            CMP #')         ;END OF STATEMENT?
1100  3682  F0 06            BEQ RET        ;YES, RETURN NOW
1110  3684
1120  3684  20 7B 2F  GETARG JSR NXIARG     ;EVALUATE NEXT ARG.
1130  3687  A8               TAY
1140  3688  E6 02            INC NARGS      ;COUNT ARGS PAST FIRST
1150  368A  60        RET    RTS            ;RETURN
1160  368B
1170  368B                        .END
```

16

FOCAL LED OUTPUT

```
2000                    ;THIS ROUTINE MAKES KIMS LEDS ANOTHER
2000                    ;OUTPUT DEVICE FOR FOCAL
2000                    ;WRITTEN BY BERNHARD MULDER
2000                    ;         MOZARTSTR 1
2000                    ;         6744 KANDEL
2000                    ;         WEST GERMANY
2000
2000                    ;THE STARTING ADDRESS FOR THIS ROUTINE
2000                    ;MUST BE PLACED INTO FOCALS OUTPUT
2000                    ;DEVICE TABLE SO FOCAL KNOWS WHERE IT IS.
2000
2000                    ;IN THE ARESCO VERSION, PUT THE HIGH ORDER
2000                    ;BYTE OF THE STARTING ADDRESS FOR THE LED
2000                    ;OUTPUT ROUTINE IN LOCATION $35B9 AND THE
2000                    ;LOW ORDER ADDRESS IN $35C0. IN THE 6502
2000                    ;PROGRAM EXCHANGE VERSION OF FCL-65E, THIS
2000                    ;WOULD CORRESPOND TO LOCATIONS $35A4 AND
2000                    ;$35A9 RESPECTIVELY.
2000
2000                    ;TO USE THE LEDS AS AN OUTPUT DEVICE,
2000                    ;SIMPLY OUTPUT TO DEVICE #2 BY
2000                    ;EXECUTING 'S FODV (2)'. ALL SUBSEQUENT
2000                    ;OUTPUT WILL BE SENT TO THE DISPLAYS.
2000                    ;
2000            PADD    =$1741
2000            SBD     =$1742
2000            SAD     =$1740
2000                    ;ZERO PAGE LOCATIONS
2000                    *=$A9
00A9            SPE1    *=*+1         ;THESE TWO CELLS CONTROL THE DISPLAY
00AA            SPE2    *=*+1         ;SPEED OF KIMS 7 SEGMENT DISPLAYS
00AB            SPE3    *=*+1
00AC            SPE4    *=*+1
00AD                    ;START THIS ROUTINE RIGHT AFTER THE CASSETTE
00AD                    ;AND USER FUNCTION MODS
00AD
36B0                    *=$36B0
36B0                    .OFF 2000
36B0  A8        SEVS    TAY
36B1  A2 01     SEV1    LDX #$1
36B3  BD 0A 37          LDA DISP,X
36B6  9D 09 37          STA DIS,X
36B9  E8                INX
36BA  E0 06             CPX #$6
36BC  D0 F5             BNE SEV1
36BE  B9 10 37          LDA TABB,Y
36C1  9D 0F 37          STA DISP+5
36C4  A2 01     ENRY    LDX #$1
36C6  B5 A9     ENR1    LDA SPE1,X
36C8  95 AB             STA SPE3,X
36CA  CA                DEX
36CB  10 F9             BPL ENR1
36CD  20 E3 36          JSR ZEI1       ;DISPLAY
36D0  C6 AB     ENR3    DEC SPE3
36D2  D0 F9             BNE ENR3
36D4  C6 AC             DEC SPE4
36D6  D0 F5             BNE ENR3
36D8  18                CLC            ;FINISH WITH NO ERRORS
36D9  60                RTS
36DA  A0 00     ZEI2    LDY #$0
36DC  8C 41 17          STY PADD
36DF  8C 42 17          STY SBD
36E2  60                RTS
36E3  20 DA 36  ZEI1    JSR ZEI2
36E6  A9 7F             LDA #$7F
36E8  8D 41 17          STA PADD
36EB  A2 09             LDX #$9
36ED  8E 42 17  ZEI3    STX SBD
36F0  B9 0A 37          LDA DISP,Y
36F3  8D 40 17          STA SAD
36F6  98                TYA
36F7  48                PHA
36F8  A0 7F             LDY #$7F
36FA  88        ZEI4    DEY            ;DELAY ABOUT 500 CYCLES
36FB  D0 FD             BNE ZEI4
36FD  68                PLA
36FE  A8                TAY
36FF  C8                INY
3700  E8                INX
3701  E8                INX
3702  C0 06             CPY #$6        ;ALL DIGITS FINISHED ?
3704  D0 E7             BNE ZEI3
3706  20 DA 36          JSR ZEI2       ;ALL OUT
3709  60        DIS     RTS            ;DIS=DISP-1
370A  00        DISP    .BYTE 0,0,0,0,0,0
370B  00
370C  00
370D  00
370E  00
370F  00
                        ;HERES THE TABLE WITH THE SEVEN SEGMENT
                        ;CODES. ASCII CODE=TABLE INDEX
3710  00        TABB    .BYTE 0,0,0,0,0,0,0,0
3711  00
3712  00
3713  00
3714  00
3715  00
3716  00
3717  00
```

;ALL DISPLAYS OFF

;****BEWARE***NON-ROMABLE CODE HERE!!!

HEX DUMP STARTS HERE →

TO SAVE SPACE... HERE'S A HEX DUMP OF THE SEVEN SEGMENT CODE TABLE.

```
3710  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3720  00 00 00 00 00 00 00 00 00 00 8C C0 98 D2
3730  8B 8A A2 00 B6 C6 82 B9 BF EF C1 C5 00 D3
3740  BF 86 DB CF E6 ED FD 87 FF EF C1 C5 00 D3
3750  00 F7 FC D8 DE F9 F1 B0 F6 84 9E F0 BB B7 D4 DC
3760  F3 E7 D0 ED F8 9C EA BE EE C9 00 00 00 00 00 00
3770  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3780  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

17

tiny basic

TINY BASIC   Editors note

Several of you were apparently confused as to
how to add the Tiny Basic mods from #15 to your
systems. I wholeheartedly recommend you pick up
the Tiny Basic Experimenters Kit mentioned in one
of the articles. (It's available for $15 from
6502 Program Exchange, 2920 Moana Ln, Reno NV
859C9.

MICHAEL DAY

TINY BASIC PAGE 0 MEMORY MAP
for TOM PITTMAN's TINY BASIC TB651K V.1K

```
0000 - 000F   UNUSED
0010 - 001F   USED IN PROTO VERSIONS ONLY
0020 - 0021   USER SPACE LOW ADDRESS
0022 - 0023   USER SPACE HIGH ADDRESS
0024 - 0025   PROGRAM END + STACK RESERVE
0026 - 0027   TOP OF GOSUB STACK
0028 - 0029   CURRENT BASIC LINE #
002A - 002B   IL PROGRAM COUNTER
```

TVT-6/TINY BASIC INTERFACE

by Michael Allen
6025 Kimbark
Chicago IL 60637

I had a lot of trouble getting Tom Pittman's
Tiny Basic to work with the KIM-1/TVT-6 combina-
tion. Now, looking back, the input and output
routines included below seem fairly simple and
straight-forward. So I thought I should share
these with you to help those who may be making the
same mistakes I was.

The T. B. version I have resides in memory
locations 0200 to 0AC6. You must change six bytes
within T.B. as follows;

1. Set 0207 to C7 and 0208 to 0A. This is a jump
to a subroutine to input a character. The input
routine saves the return address to T.B. then jumps
to the SCAN program and stays there until inter-
rupted by a strobe signal from a key being pressed
on the keyboard. If the IRQ vector has been pro-
perly set to 0AD3, a character is sent to the cur-
sor subroutine. Then a return is made to T.B.
Note that a CLI (clear interrupt status) instruc-
tion was inserted in SCAN (underlined in the hex
dump).

2. Set 020A to F3 and 020B to 0A. This is a jump
to the output subroutine where the miscellaneous
characters T.B. sends for the benefit of a tele-
type are trapped before falling through to the
cursor subroutine.

```
002C - 002D   BASIC POINTER
002E - 002F   SAVED POINTER
0030 - 007F   INPUT BUFFER AND COMPUTATION STACK
0080 - 0081   RANDOM NUMBER SEED
0082 - 0083   VARIABLE 'A'
0084 - 0085   VARIABLE 'B'
....          ....
00B4 - 00B5   VARIABLE 'Z'
00B6 - 00B7   TRANSFER WORK POINTER
00B8 - 00B9   MISC WORK REGISTER
00BA - 00BB   MISC WORK REGISTER
00BC - 00BD   TEMPORARY STORAGE REGISTER
00BE          RUN MODE FLAG
00BF          PRINT CONTROL
00C0          INPUT BUFFER POINTER
00C1          COMPUTATION STACK POINTER
00C2          2nd ½ OF STACK POINTER (ALWAYS 00)
00C3          COUNTER (USED IN PN ONLY)
00C4 - 00C5   IL XQ POINTER
00C6 - 00C7   GOSUB STACK WORK POINTER
00C8 - 00D7   USED IN SPHERE VERSIONS ONLY
00D8 - 00FF   UNUSED
```

There are the major use of these registers
only they may be used for other purposes on an
availability basis.

3. Set 020F to 08. This allows T.B. to recognize
the ASCII backspace.

4. Set 028C to 0E. When starting T.B. at 0200
(cold start), this byte determines how T.B. de-
fines the lowest address of program space.

5. Also be sure to set 17FE to D3 and 17FF to 0A.

I relocated SCAN to be able to reload T.B.
from tape in one load. The version of SCAN shown
is from Don Lancaster's Popular Electronics ar-
ticle except for bytes 0BA4 and 0BCC which were
changed in order to display pages 0C00 and 0D00.

The Cursor program is adapted from Don's but
is much shorter as it only supports backspace and
carriage return controls--all you really need with
T.B. (also INPUT sets lowercase to uppercase so
you don't have to shift back and forth.)

KIM's Memory map now appears thus:

```
0020-00B9   Used by tiny BASIC
00E8-00EE   Used by I/O routines
0200-0AC6   Tiny BASIC
0AC7-0B79   INPUT & OUTPUT Subroutines
0B7A-0BDC   SCAN
0BDD-0BFF   34 bytes for USR subroutines (I put
            Don Box's subscripted variable SBR's
            here; see KUN #5.)
0C00-0DFF   TVT-6 display area
0E00-13FF   1.5K program area
```

18

TVT6/TINY BASIC INTERFACE LISTING

```
SET 17FE = D3, 17FF = 0A

                INPUT    PLA LOW        SAVE ... RETURN ... ADDRESS.
0AC7 68
0AC8 85 E8               STA TEMP
0ACA 68                  PLA HI
0ACB 85 E9               STA TEMP+1
0ACD BA                  TSX
0ACE 86 EA               STX TEMP+2     AND STACK ... POINTER.
0AD0 4C A7 0B            JMP SCAN

0AD3 AD 00 17   BREAK    LDA CHAR       GET CHARACTER.
0AD6 29 7F               AND #$7F       REMOVE PARITY.
0AD8 C9 61               CMP #$61       LOWER CASE LETTER?
0ADA 90 02               BCC SKIP       NO; SKIP AHEAD.
0ADC E9 20               SBC #$20       YES; MAKE UPPER CASE.
0ADE 85 EB      SKIP     STA TEMP+3     SAVE CHARACTER.
0AE0 C9 0D               CMP #$0D       CARRIAGE RETURN?
0AE2 F0 03               BEQ RTN1       YES; RETURN.
0AE4 20 FB 0A            JSR CURSOR     NO; ENTER CHARACTER.
0AE7 A6 EA      RTN1     LDX TEMP=3     RESTORE ... STACK POINTER.
0AE9 9A                  TXS
0AEA A5 E9               LDA TEMP+1     RESTORE ... RETURN ... ADDRESS.
0AEC 48                  PHA
0AED A5 E8               LDA TEMP
0AEF 48                  PHA
0AF0 A5 EB               LDA TEMP+3     GET CHARACTER.
0AF2 60         RTN2     RTS            RETURN TO TINY.

0AF3 C9 0B      OUTPUT   CMP #$0B       TRAP ... CONTROL ... CHARACTERS.
0AF5 30 FB               BMI RTN2
0AF7 C9 7F               CMP #$7F
0AF9 B0 F7               BCS RTN2
0AFB 48         CURSOR   PHA            SAVE CHARACTER.
0AFC A0 00               LDY #0         RESET INDEX.
0AFE A5 EE               LDA EE         GET CURSOR HI ADDR.
0B00 C9 0D               CMP #$0D       IS CURSOR ON PAGE 0D?
0B02 F0 04               BEQ CONT       YES; CONTINUE.
0B04 C9 0C               CMP #$0C       NO; OR ON PAGE 0C?
0B06 D0 2F               BNE SCROLL     NO; INITIALIZE CURSOR.
0B08 B1 ED      CONT     LDA (ED),Y     GET OLD CHARACTER.
0B0A 29 7F               AND #$7F       REMOVE CURSOR.
0B0C 91 ED               STA (ED),Y     REPLACE.
0B0E 68                  PLA            RECALL NEW CHARACTER.
0B0F C9 20               CMP #$20       IS IT A CHARACTER?
0B11 B0 59               BCS ENTER      YES; ENTER IT.
0B13 C9 0D               CMP #$0D       CARRIAGE RETURN?
0B15 D0 0C               BNE SKIP1      NO; SKIP
0B17 A5 ED               LDA ED         YES; MOVE CURSOR ... TO RIGHT SIDE.
0B19 09 1F               ORA #$1F
0B1B 85 ED               STA ED
0B1D 20 61 0B            JSR INCR       AND REPLACE.
0B20 4C 6F 0B            JMP END        INCREMENT CURSOR.

0B23 C9 08      SKIP1    CMP #$08       BACKSPACE?
0B25 D0 4C               BNE RESTORE    NO; CONTINUE.
0B27 C6 ED               DEC ED         YES; DECREMENT CURSOR.
0B29 A9 FF               LDA #$FF       TEST FOR PAGE ...
0B2B C5 ED               CMP ED              UNDERFLOW.
0B2D D0 44               BNE RESTORE    O.K. TO CONTINUE.
0B2F C6 EE               DEC EE         DECREMENT PAGE.
0B31 A9 0B               LDA #$0B       TEST FOR SCREEN ...
0B33 C5 EE               CMP EE              UNDERFLOW.
0B35 D0 3C               BNE RESTORE    O.K.
0B37 A9 00               LDA #0         NOT O.K.; HOME CURSOR.
0B39 85 ED      SCROLL   STA ED         TO 0C00
0B3B A9 0C               LDA #$0C       (UPPER LEFT OF SCREEN)
0B3D 85 EE               STA EE
0B3F A0 20               LDY #$20       ADD OFFSET TO INDEX.
0B41 B1 ED      LOOP     LDA (ED),Y     MOVE ...
0B43 A0 00               LDY #0               CHARACTER ... UP.
0B45 20 5F 0B            JSR STORE
0B48 D0 F5               BNE LOOP       LOOP UNTIL END OF SCREEN.
0B4A 18                  CLC            CLEAR FLAG.
0B4B A9 E0               LDA #$E0       HOME CURSOR
0B4D 85 ED      HOME     STA ED         TO 0DE0
0B4F A9 0D               LDA #$0D       (LOWER LEFT OF SCREEN).
0B51 85 EE               STA EE
0B53 B0 1E               BCS RESTORE    FINISH IF FLAG SET.
0B55 A9 20               LDA #$20       ELSE; CLEAR LAST LINE.
0B57 20 5F 0B   SPACE    JSR STORE      ENTER SPACT TO ...
0B5A D0 F9               BNE SPACE      END OF LINE.
0B5C 38                  SEC            SET FLAG.
0B5D B0 EC               BCS HOME       TRY AGAIN.
0B5F 91 ED      STORE    STA (ED),Y     ENTER CHARACTER.
0B61 E6 ED      INCR     INC ED         INCREMENT CURSOR.
0B63 D0 06               BNE RTN        OVERFLOW?
0B65 E6 EE               INC EE         YES; INCR CURSOR TO NEXT PAGE.
0B67 A9 0E               LDA #$0E       TEST FOR SCREEN OVERFLOW.
0B69 C5 EE               CMP EE
0B6B 60         RTN      RTS
0B6C 20 5F 0B   ENTER    JSR STORE      ENTER CHARACTER.
0B6F D0 02      END      BNE RESTORE    END OF SCREEN?
0B71 F0 C4               BEQ SCROLL     YES; SCROLL UP.
0B73 B1 ED      RESTORE  LDA (ED),Y     GET CHARACTER.
0B75 09 80               ORA #$80       ADD CURSOR.
0B77 91 ED               STA (ED),Y     REPLACE.
0B79 60                  RTS            RETURN TO I/O ROUTINES.
```

HEX DUMP OF RELOCATED "SCAN" PROGRAM:

```
0B7A             EA 8D 84 0B 48 68
0B80 D0 00 20 00 80 69 08 C9 C0 90 F0 20 DA 0B 20 00
0B90 80 AA AD 83 0B 69 1F 8D 83 0B 8A D0 AA EA 69 C0
0BA0 20 00 80 C9 86 90 D3 AD D9 0B 49 80 30 05 8D D9
0BB0 4B A2 66 20 DA 0B 20 DA 0B 10 05 8D D9 4B A2 67
0BC0 20 1E 80 8D 83 0B A9 00 8D 83 0B A9 84 8D 84 0B
0BD0 20 00 80 18 CA 30 A4 10 ED 80 B0 00 60
```

KIMSI, S-100

16K RAM MOD

from Vince Coppola
12 Charles St.
Plantsville, CT 06479

I have installed the Digital Research S-100, 16K static RAM board sucessfully in my KIM-KIMSAI system, with only one minor modification. Remember, the board uses 2114 1Kx4 bit RAMS, so for 4K of memory all you need is 8 RAM chips. Also, the board can be depopulated to one 4K block or any multiple of 4K right on up to the full 16K. The manufacturer also states that a full blown board will draw less that 2 amps.

The modification I came up with, consists of grounding pin 75 of the board. This is done because pin 75 on the KIMSAI is not used and is therefore left floating. By grounding it and inserting the associated jumper, pin 4 of U-43 will be brought low therefore presetting the flip-flop and siabling the bank-select circuitry. Also, the enable LED should now stay always lit.

[EDITORS' NOTE-Vince also told me a little about the CGRS disk system that he has running on his KIMSI. I'll try to get more details on this system from CGRS for the next issue].

SYM

SYM BASIC   CASSETTE NOTES

from Eugene Garapic
14231 Thompson
Brookpark, OH 44142

I've found a bug in SYM BASIC.

The problem is that you can't save or load Basic programs from WARM START. You can see the problem if you leave Basic and then re-enter Basic through the WARM START location (so as not to clear the BASIC program). You'll find it impossible to save the program on tape in the Basic format.

Fortunately, the fix isn't difficult. Simply key in the following machine language program when you want to re-enter Basic. Install this routine in high memory as not to interfere with your Basic program.

```
20 86 8B JSR ACCESS; UNPROTECT SYSTEM RAM
4C 00 00 JMP WARMSTART; AND RE-ENTER BASIC
```

WARM START is at location $0000 and jumps to location $C27E. COLD START is at $C000 and jumps to $DE6D.

As for the SYM cassette interface:

Put a capacitor in series with the SYM audio output for 1000% better results. Why? Because this isolates the D.C. component in SYM's audio output from the recorder input. (a .01 uf works fine).

The cassette recorder must have enough power to drive SYM's LED recording indicator or else you are working in the critical mode and will get unpredictable results. Use a recorder with at least 1 watt output. The SANKYO ST-50 (available locally for about $40.00) cassette recorder has a tape counter, automatic shutoff, and works great with SYM.

SYM ON-BOARD SPEAKER TOGGLE ROUTINE

from Bruce McKenzie

This is a subroutine to invert the state of the SYM's on-board speaker. If it is called regularly, it will produce an audible tone. Use it as you would 'INC 1700' on the KIM, or 'INC A000' on the SYM. This routine saves all registers, and is totally relocatable. As it stands now, memory locations 90D and 91C must be modified to point to an unused zero page location to hold the flag.

One final word- the speaker, being electrostatic works best at higher frequencies.

```
0090 XX          Flag location-will be either '06'
                   or '08'
0900 20 88 81     JSR SAVER
0903 D8           CLD
0904 A9 0D        LDA #0D
0906 20 86 8B     JSR ACCESS
0909 20 A5 89     JSR CONFIG
090C A5 90        LDA FLAG
090E C9 06        CMP #06
0910 F0 06        BEQ *+6
0912 A9 06        LDA #06
0914 D0 02        BNE *+2
0916 A9 08        LDA #08
0918 8D 02 A4     STA OUTREG
091B 85 90        STA FLAG
091D 4C C4 81     JMP RESALL
0920
```

Courtesy of the San Fernando Valley 6502 Users Group.

AIM info

Some Useful
AIM NOTES

from Al Davidson
5746 Ballenmoor
Memphis TN 38118

Rockwell has come up with the KIM owner's finest fantasy. Easy-on-the-eye display, easy-to-operate cassette interface, easy-to-peck-on-keyboard, easy-to-use printer; It's all there, 6502 fans!

The bare AIM 65 is a prize alone, not to mention basic, which is at last her (got mine 6/20).

This is certainly the machine for the bells and whistles addict, but you want more? Here are a few tried and true suggestions-----

2MHZ OPERATION-Cut the run on top of the board connecting pin 12,Z10 to pin 5,Z10. Connect pin 12 to the common on a spdt switch. Also connect pin 5,Z10 and pin 8,Z10 to the remaining two terminals on the switch. This allows you to select 1 or 2MHZ operation. The printer operates a great deal less than perfect at 2M, and of course, the cassette info will be twice as fast. This may or may not agree with your recorder, but it performs 200% with the superscope mod# C-108. A more elaborate means for switching may incorporate the CS line for Z32, which is used for both printer and cassette. This could slow the clock down to 1MHZ when the cassette or printer are being "spoken" to.

By the way, doubling the proc. speed has no detremental effects on the machine when running basic along with an extra 4K of 2116 memory added on-board piggy-back style. Fast-Basic fans, take note!! (This is however, recommended for on-board systems only.)

2716's- The ROM sockets on-board can easily be used for 2716 (5V) eproms. Cut the run under the socket connecting pin 18 to all, and jumper 18 to gnd. Since the 2716 is only a 2K eprom, it appears twice in memory; at the first and second 2K boundarys of the socket. If you don't mind that, plug 'em in!

KIMSI-The AIM is billed as being hardware compatible with KIM-close! But no cigar! The difference (other than obvious address decoding brought out to the connectors) is in the generation of RAM R/W on pin E-Z. The KIM circuit nands buffered phase 2 with inverted R/W. The AIM nands buffered phase 2 with buffered R/W. El Wrongo! But easy to remedy, if you're surgically inclined. First, disable the sys R/W to pin 4, Z13. Cut the trace at pin 4 and also the trace that runs under the cip and towards Z32. Now, jumper pin 4,Z13 to pin 5,Z16, which is the correct phase of the R/W line. Now, to re-establish R/W to Z32, jumper pin 22,Z32 to pin 6,Z16. This takes care of the RAM R/W difference.
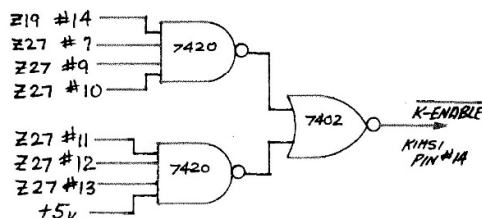

FIG 1. GENERATES NEW K-ENABLE SUITABLE FOR AIM MEMORY MAP. SEE ALSO KIMSI MANUAL PAGE A4 (3) FIGURE 'C'.

Next, to accomodate AIM's memory map, we have to create a new K-enable. This is explained on page A4(2) of the KIMSI manual, but the general idea is to tell the KIMSI to turn itself off when the proc. is "talking" to a device on it's own board. I used the circuit in fig. 1 on a small piece of breadboard mounted on the AIM. This, I thought, was better than running 8 lines to the KIMSI prototype area. As explained in the KIMSI manual. The KIMSI decoding circuitry has to be disabled by cutting the "V" shaped run under the board at Z11, pin 11.

Now, connect the new K-enable line to KIMSI pin 4 and enjoy!

These two mods render a trouble-free machine, as far as AIM-KIMSI interface goes, but the extra circuitry along with the added lead length to the KIMSI prohibits the use of a 2 MHZ clock.

The 9-digit Microsoft Basic-in-ROM is certainly worth the 100 bucks. Room is provided on board for the ROMS. The peek and poke statements make

access to ports and memory convenient, & the 'wait' statement provides high-level port-watching that's easy. Access to user machine language routines could have been easier, but at least it's provided for. Cassette save & load commands are certainly worth honorable mention, too.

This is a real 8K basic, with an impressive list of string handling and arithmetic functions not to mention the usefulness of the error codes and good documentation.

Although the basic is well incorporated into the AIM system using the keyboard, printer, and display, user I/O may easily be used instead, as with all the other AIM I/O functions. (I'm using a KENT-MOORE video board in the KIMSI for output).

AIM TAPE PROBLEMS

from Steve Bresson
1666 Independence Ct
Severn, MD 21144

I have found two problems with the AIM65, both in the tape I/O area.

I. The first is in using the LOAD, (L), command to input object code from tape. When the last record of a file occurs within 6 or 7 bytes of the end of a BLOCK, the loader will either hang up while looking for another block of input, or give an ERROR when it finds another block, which does not have the correct block number.

The error is in the LOAD ($E2E6) program. It first reads the record length, and then the ADDRESS/# of Records, depending on the type of record. If the record length in 00, then the record is the last in the file, and it attempts to read (5+1=)6 bytes from the input buffer. But there are only 3 bytes left from the valid input data, so that if there is enough garbage at the end of the buffer, nothing will happen. When the last byte of the record occurs in the last 2 bytes of the buffer, the input routine attempts to read another block to get the additional data. This causes the hang-up/error.

	1		5		1		
BEGIN FILE	BLK#	FILENAME	∅D	RECORD	/ RECORD		
NORMAL RECORD	";"	#BYTES	STARTING ADDR.	DATA	∅D		
LAST RECORD	";"	∅∅	# RECORDS	CHECKSUM	∅D		

The error can be easily shown by the following procedure:

1) place known data in $200-220
2) do a DUMP, (D), to tape, (T) and dump the following blocks of data in one file:
   $200-217,
   $200-207,
   $200-209.
3) rewind the tape and check it using the verify, (3), command. This should show no problems
4) rewind again, and attempt to load, (L). When it hangs, do a RESET, and look in the tape buffer ($116-165). Your data should be there along with the record data.

NOTE: While this problem may not occur too often, it can be a pain when it does, if you do not know what is happening.

II. The second problem is an incompatability between the AIM BASIC tape output and that used by the TEXT EDITOR, and expected by the verify, (3) command. The text editor puts out, and expects to receive a text file ended with a double (CR). Basic can read a text editor file as long as a (ctl-Z) is the last character in the file before the (CR) (CR). But, when Basic writes a file, it puts out a (CR) (LF) (CR) (LF) (CTL-Z) at the end. Both the editor and verify hang-up when reading this.

SOME PERSONAL VIEWS ON HOW AIM COMPARES...

by Jim Butterfield
Toronto

AIM vs KIM

1. AIM cost more.
2. The AIM power supply is a bit tougher than KIM's: 24 volts at 0.5 amperes is harder to procure than KIM's 12 volts at about 0.1 amp, even allowing for the fact that the 24 volts doesn't need regulation.
3. AIM's display is larger, but tougher to work, since it's mostly planned for serial ASCII input; you can't get to individual segments as you can with KIM.
4. KIM has a more flexible system for re-arranging memory; AIM is directed more towards a completely fixed memory map.
5. AIM's built-in printer is a great bonus, even if you have to live with 20 columns.
6. The basic AIM system has spendid monitor features, including Single-step and Breakpoint, with options such as register display and next-instruction disassembly.
7. AIM also has a disassembler and mini-assembler in the basic monitor; they are very handy.
8. AIM's plug-in assembler and plug-in Basic chips make these enhancements simple to add. Extra memory (up to 4K) also goes on-board.
9. There are cassette recorder control lines for AIM, although I haven't been able to make mine work. AIM will write KIM-compatible tape; but its own format is quite nice, and the display gives you a running commentary on what's happening during saves and loads.
10. The text editor is a nice built-in feature; not an earth-shaker, but handy enough.
11. KIMs outnumber AIMs by a very large margin. There are more opportunities to find other KIM-users, programs, etc.

AIM vs PET

1. PET costs more.
2. PET's power supply is built in; you have to go hunting for an AIM power supply, and it's extra.
3. The full CRT display of the PET is of course much more useful than AIM's little 20-character LED strip. PET also has graphics and/or lower case. PET's CRT, however, increases weight and size; AIM (when suitably packaged) is far more portable.
4. AIM may be expanded on-board to 4K; depending on the model, PET may have up to 32K of RAM built in.
5. PET does not come with a printer; AIM does. It costs a fair piece of change to add a printer to PET, although such a printer would of course have more than 20 columns. AIM has a built-in teletype interface if needed; PET needs an adapter to do this.
6. AIM's monitor is excellent for machine-language work. PET's machine language monitor is much less powerful. Enhanced monitors are being passed through the PET user community, but even these don't have all the features that are built into AIM.
7. AIM Basic (an optional extra) is very similar to that of PET. Pet's is somewhat better. I particularly miss the SYS command which isn't available on AIM. Basic file handling is somewhat better on the PET.
8. PET can be expanded with disk and printer to a quite powerful standard system. AIM is capable of this, but there are no standard Rockwell products for this kind of expansion, and each user tends to be on his own.
9. PET' outnumber AIMs by a large margin. There are more opportunities to find other PET-users, programs, etc.

65XX chip family stuff

6522 USER NOTES

from John T. O'Donell
Aydin Monitor Systems
401 Commerce Drive
Fort Washington PA 19034

1. Recently attended a marketing presentation by Synertek in Philadelphia. Conrad Boisvert of Synertek introduced their 6500 family UP's and support chips including the 6522. Enclosed you will find a copy of the SY6522 spec document given out at the presentation. Haven't gone thru all of it in detail but it appears to answer a lot of questions and correct a lot of errors found in the MOS Technology document.

2. Conrad says that Synertek has corrected the problem with the shift register shifting in at system clock rate and generating 9 clock pulses per shift operation instead of 8. The corrected devices are supposed to be available now, but, according to Conrad, there is no change to the part number. Thus you have to go by date code (buy the most recent).

3. The uncorrected device can still be used for shifting in at system clock rate since it does stop shifting and generates an interrupt after 8 clock pulsrs. The extra shift clock pulse presents a problem only for the device providing the serial data. If that device readies more than 8 bits at a time in its serial output shift register then there will be a problem. However, if after each serial byte transfer the controlling UP causes the remote device to load its next byte into its serial output register then the extra clock pulse will be ignored. Naturally there is no problem if the remote device is another 6522.

4. Having said all that we come to a subtlety in the timing of the shift-in operation that will cause a problem if the remote device is other than another 6522. The subtlety involves the timing relationship of the data on CB2 to the rising edge of the shift clock on CB1: the data should be held stable for one full 02 clock cycle after the rising edge of CB1 shift pulse. In our application we wanted to load a byte of data into a 74LS299 universal shift register and then shift it into a 6522. So we connected the serial output of the LS299 to CB2 and connected the shift clock from CB1 to the SL299 clock input. We were using shift-in at system clock rate and each byte acquired was shifted left by 1 bit. When I observed that there were 9 shift clocks I thought I had the answer to the problem and called MOS TECHNOLOGY to find out what to do about it.

I spoke to Rich Gapin there who told me that although there are 9 shift clocks the 6522 interrupts and stops shifting after 8 so that wasn't the cause of the problem. Subsequent discussion revealed the timing constraints and made apparent that there had to be an extra stage of storage between the output of the LS 299 and CB2. Consequently the serial input data goes to the D input of a 74LS74 clocked by the CB1 shift clock. Q output of the LS74 connects to CB2. The LS299 and LS74 both shift data on the rising edge of the clock.

Therefore each bit shifted out of the LS299 will be stable at CB2 input immediately after CB1 clock rising edge.

23

A COUPLE OF 6522 APPLICATIONS NOTES

from Conrad Boisvert
Synertek Inc

SY6522 Generating Long Timed Intervals

The SY6522 Versatile Interface Adapter contains two 16-bit counter/timers for a variety of purposes, among them the generation of timed interrupts. Each counter is 16 bits long, so the maximum count-down is 2^{16} or 65,536 counts. With a 1MHz processor clock rate, this translates to a maximum time of about 65.5 msec.

In some cases, this may not be long enough. To achieve longer timed intervals, several schemes may be used. Among them are:

1. Increment or decrement a memory location each time the timer interrupt occurs. In this way, an additional factor of up to 256x can be achieved, resulting in a maximum of about 16.8 seconds. However, extra program steps are needed.

2. The two SY6522 timers may be connected externally (Figure 1), resulting in an effective 32-bit counter/timer. In this way, intervals longer than one hour may be achieved.
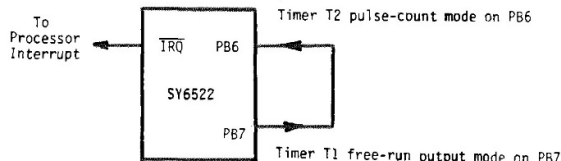

Figure 1-Connection to Use T1 and T2 as 32-bit Counter

PROGRAMMING CONSIDERATIONS

To cascade the two counters together it is necessary to do the following:

1. Connect PB6 and PB7 together. These pins will not be useable as general I/O functions in this case.
2. Program T1 mode to free-run with output on PB7.
3. Program T2 mode to count pulses on PB6 input.

In this way, the waveform on PB7 is,
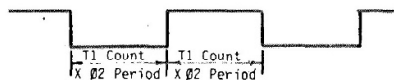
Since timer T2 pulse-counting mode counts negative-edge transistions, it is clear that T2 will decrement as follows:
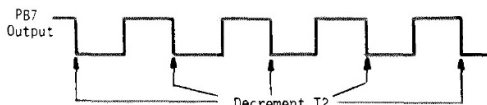
Thus, T2 decrements will occur at the following intervals,

T2 RATE=2x (T1 COUNT) x (02 PERIOD)

And, hence, the total time will be,

$T = 2 \times$ (T1 count) x (T2 count) x ($\emptyset2$ period)

Thus, the maximum is $2 \times 65,536 \times 65,536 \times 1$ us = 8590 seconds = 142 minutes = about 2½ hours.

24

SY6522 - GENERATING A 1Hz SQUARE-WAVE SIGNAL

The SY6522 (Versatile Interface Adapter) has two integral 16-bit timers intended to perform a variety of programmable functions. One capability is to use timer T1 to generate a continuous square-wave output on peripheral pin PB7.

The timer is clocked by the system clock, $\emptyset2$, which normally operates at 1MHz. The waveform generated is illustrated in Figure 1.
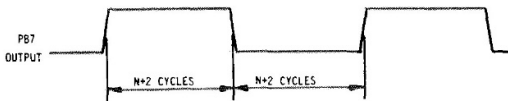

Figure 1 - PB7 output Waveform

Note that the period of the waveform is 2N+4 cycles, with a 16-bit counter, the maximum number of cycles is where N is the number set into the timer.

$N_{MAX}=2^{16}-1 = 65,535$

Hence, the maximum programmable period is

$P_{MAX}=2N_{MAX}+4= 131,074$ cycles.

This is about 131 msec for a 1MHz system clock, considerably less than 1000 msec, the period of a 1Hz signal.

One way to extend the period is to use the PB7 output signal as a clock input to the shift register on the SY6522. If a pattern of 11110000 is set into the shift register, then the output of the shift register will appear as in Figure 2.
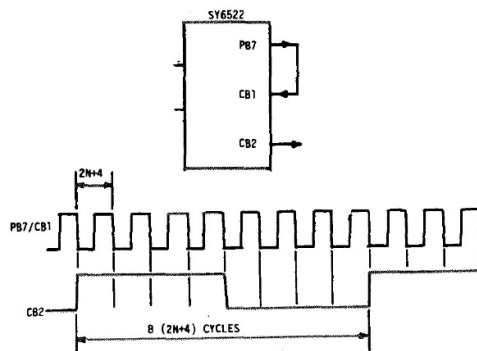

Figure 2 - Shift Register Output Waveform

Note that the period is extended by a factor of 8 by this method.

$P_{MAX}=8 (2N+4)$

Hence, for 1 Hz, $P_{MAX} = 1,000,000$ and N = 62,498. Thus, it is necessary to store the number, 62,498, into the timer T1 in order to generate the 1 Hz waveform. When translated into hexadecimal format, the result is F422, and F4 is loaded into the high byte and 22 into the low. The step-by-step sequence for programming this is shown in Figure 3.

Note the expecially the following points:

* Loading the T1 high-order counter (Register 5) initiates the timer in its free-running mode.
* PB7 data direction must be set to an output for the pulses to occur.

```
0005  2000              ;PROGRAM TO GENERATE 1HZ SQUARE-WAVE OUTPUT
0010  2000              ;ON 6522 PB7 OUTPUT PIN USING T1 TIMER
0015  2000              ;AND SHIFT REGISTER
0020  2000              ;
0025  2000              ;----------SY6522 ADDRESSES----------
0030  2000      DDRB    =$A802          ;DATA DIRECTION REG
0035  2000      T1CH    =$A805          ;T1 HIGH BYTE
0040  2000      T1LL    =$A806          ;T1 LOW BYTE
0045  2000      SR      =$A80A          ;SHIFT REGISTER
0050  2000      ACR     =$A80B          ;AUXILIARY CONTROL REG
0055  2000              ;
0060  2000              *  =$0200       ;START ADDRESS
0065  0200
0070  0200  A9 F0       LDA #%11110000
0075  0202  8D 0A A8    STA SR          ;STORE SHIFT PATTERN
0080  0205  A9 DC       LDA #$DC
0085  0207  8D 0B A8    STA ACR         ;SET UP T1 AND SHIFT REG
0090  020A  A9 22       LDA #$22
0095  020C  8D 06 A8    STA T1LL        ;LOW BYTE
0100  020F  A9 F4       LDA #$F4
0105  0211  8D 05 A8    STA T1CH        ;HIGH BYTE + START
0110  0214  A9 80       LDA #$80
0115  0216  8D 02 A8    STA DDRB        ;SET PB7 AS OUTPUT
0120  0219  4C 19 02  LOOP  JMP LOOP    ;PROGRAM HALT
0125  021C
0130  021C              .END
```

6551 ACIA MINI-SPECS

SYNERTEK has recently released a very inter-
esting new addition to the 65XX family. It's
called the 6551 Asynchronous Communication Inter-
face Adapter (ACIA) and is a considerable upgrade
to the present Motorola 6850 ACIA. The best fea-
ture of the 6551 is the on-chip baud rate genera-
tor. The baud rate is software programmable and
generates 15 baud rates (50 to 19.2 K baud) from
a standard 1.8432 MHZ external Xtal. There's al-
so an external 16X clock input for non-standard
baud rates (up to 125 Kbaud). I just finished
wire wrapping up an RDE prototyping card, with
two 6551's which are driven by a single 1.8432
MHZ Xtal in a TTL oscillator configuration. (By
the way, I'm using the lowest cost battery oper-
ated wire-wrap gun from OK Machine & Tool and it's
been working like a champ. I whole heartedly
recommend wirewrap for getting prototypes up and
debugged quickly.)

This board will serve as the system I/O mod-
ule for use by my homebrew 6512 CPU board in my
"dream machine".

One of the 6551s will be used by my Hazel-
tine CRT while the other will be driving my print-
er. Both will be RS-232. Later, I may build up
another board for modem control.

The 6551 is, of course, fully programm-
able as far as: word lengths; number of stop bits;
parity bit generation and detection; interrupt
operation etc. Also, modem control signals are
provided. A very versatile chip indeed!

Get more info from SYNERTEK or ROCKWELL.

Eric

BUGS

ouch!!!

THE CASE OF THE SWITCHED SOFTWARE

Bruce Nazarian

Sharp-eyed readers of 6502 Notes will already
have noticed my inadvertent error in issue #15,
re: music mods.

Author Armand Camus is, in fact, quite cor-
rect in his references to the page zero locations
used in his software. Not having the original
software (as published in BYTE) handy, I had no
way to know his references weren't incorrect: so,
be aware, you KIM music lovers-the BYTE (also COM-
PUTERIST) software is not the same as the Advanced
Music Software being marketed by MTU.

Also, in recent conversations with Dave Cox,
Marketing manager for MTU, I was advised of the
existence of a NEW KIM DAC board, which Dave as-
sures me will outperform the old DAC..It must be
quite a board, as the old DAC was nothing to sneeze
at! Several changes have been made, but all for
the better. The new board only requires a single
supply voltage (+5), and should have an improved
signal-to-noise ratio over the old one. I have
ordered a new one from them and will have a user's
report as soon as I have it up and running.

My sincere apologies to any readers I may
have caught off-guard with my software mistake,
and sincere apologies to author Camus for doubting
his eyesight!!!

BUGS IN #15, PAGE 4

There are some errors in the wiring sketch
(fig. 4) for the dynamic RAM board. The schematic
on the pre ious page is believed correct however.
The problem in the wiring sketch involves place-
ment of the address lines so the sketch is still
useable once the address lines are referenced to
the correct pins on the RAM chips.

BUGS IN #15 PAGE 22,23

The article by Cass Lewart, 1:32640 should be
1:32896 the ocrrect formula is:

$$(FF \ hex + 2) \quad \frac{(FF \ hex + 1)}{2} \quad K = 0$$

and the last row of numbers in the table should be:
32.9 ms, 263 ms, 2.11 ms, 33.7 sec.

PAGE 26

Reverse the polarity of the diode placed a-
cross the 5 volt relay.

25

OSI

READ KIM TAPES ON YOUR OSI SYSTEM

By Robert Solomon
5868 Joanne Ct.
North Ridgeville OH 44039

In my previous letter, I stated that I was developing a high speed PLL tape interface for the OSI. After looking at the benefits of that approach, I have abandoned that idea. It would be of much greater help to the OSI experimenter to increase his access to cassette-based experimenter software.

A greater amount of 6502 experimenter software is available to the KIM owner than will probably ever be available to the OSI owner. So all that needs to be done is to make the OSI system compatable with the KIM tape format. I have an OSI 404V CPU and the owners manual says you can install a KIM chip on the board. It is not quite that easy, as I found out when I studied the operaiton of the KIM 6530 chip. The solution to that problem is explained in this article.

As I described in my previous letter, I have a 16K system consisting of two 4K boards and one 8K board. I also stated I was in the process of implementing KIM Focal 65. Focal for KIM resides at 2000 to 360A with user program above 360A. If my 16K of memory were continuous from 0000, I would only have about 2½K of user workspace. But I need to have the memory continuous for my Tiny Basic which starts at 0200. So I set up my memory as follows:

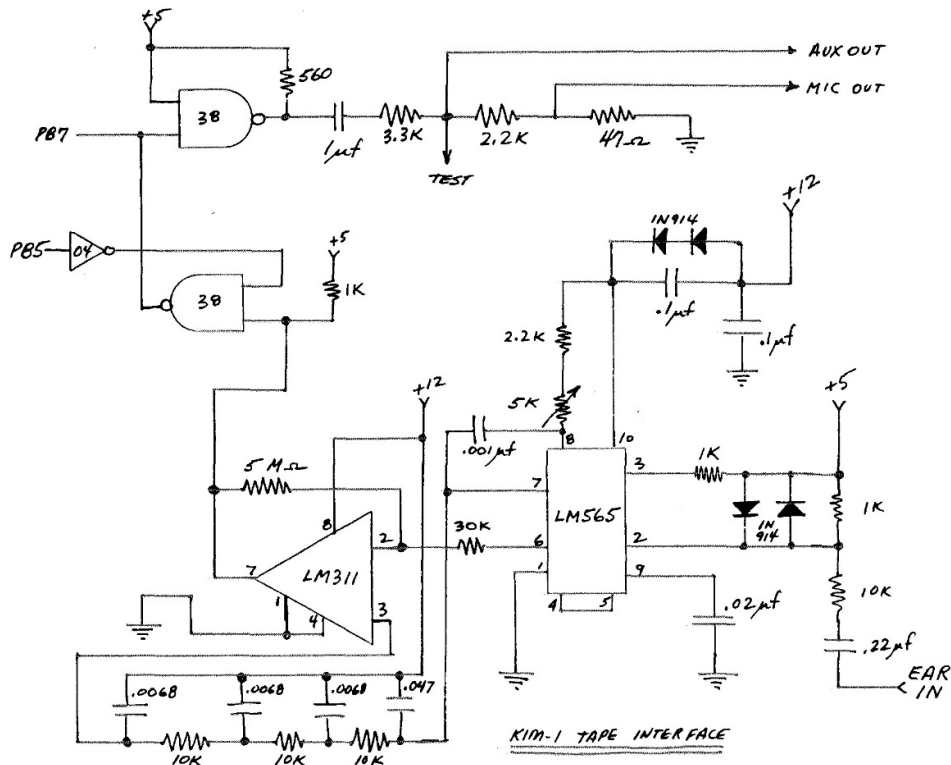4K from 0000 to 0FFF for page zero and I/O to teletype, CRT, and KC format tape;

8K from 2000 to 3FFF for general use (including Focal); and 4K which can be placed at either 1000 ro 1FFF ro 4000 to 4FFF controlled by a front panel DPDT switch.

Now I can run either Tiny or Focal and have workspace located as needed based on the front panel switch position. I also plan on using 9-digit Microsoft KIM Basic located from 2000 up.

All I need is a method to load KIM tapes into the OSI. There is also a high speed tape format for KIM called hypertape; and Lou Edwards has available a 4800 baud Zip-tape system for KIM. It would be nice to be able to use these on the OSI.

Implementing the 6530 chip was not quite as simple as one would expect. In order to understand the problems, you need to understand the KIM program. (Anyone contemplating using the KIM simulator should get a copy of the KIM-1 Users Manual.)

Each 6530 includes 1K of ROM, 64 bytes of RAM, i/o ports, and a timer. The 6530-002 chip ROM contains the KIM keyboard, display, and TTY operating programs. The 6530-003 chip ROM contains the tape interface program. The problem is that the -003 ROM program uses the RAM, I/O, and timer on the -002 chip! The tape programs also exit to a location in the -002 ROM.

KIM-1 TAPE INTERFACE

26

The solution to this problem is shown in the second schematic. I managed to fit it all on the CPU board, but I recommend putting it on a separate board. Then you could install the tpae interface circuit on the board also. (A prototype board is available from D&N Micro Products, 3932 Oakhurst Dr., Fort Wayne, IN, 46815, for $27.00 including postage, handling, and connectors.) My solution involved circuitry to make the 6530-003 use its own RAM, I/O, and Timer for the tape functions. I also implemented a 1K RAM at 1C00 to 1FFF and the tape program will exit into this RAM.

The 7404, 7420, and 74145 decode the six high order address lines to provide the selects for the 6530 I/O, Timer, and RAM. (K5 to CS1 for 1400-17FF); 6530 ROM (K6 to RS0 for 1800-1BFF); and the 1K RAM (K7 to 2102 CE for 1C00-1FFF) Address lines A0 thru A9 go directly to the 6530 except for A6. The proper handling of A6 is really the solution to the whole problem.

The selection between the 6530-002 and the 6530-003 is made by RS0 and A6. When RS0 is low, A0 thru A9 must directly access the 6530. RS0 for the 003 is connected to K6 and the RS0 for the 002

is normally connected to K7. I use K7 to select the 1K RAM instead of the 002 chip. When RS0 is high, the 6530 is selected by CS1 being low. Both the 002 and 003 respond to K5 connected to CS1. The 002 and 003 are distinguished by the status of A6. A high on A6 selects the 002 and a low on A6 selects the 003.

So all we have to do is invert A6 to select the 003 I/O, RAM, and Timer for the tape interface. I used the 7486 Exclusive-OR gate to perform this function. When K6 is low (ROM Select) A6 is not inverted and we have access to the tape program in ROM. When K6 is high (at all other times) A6 is inverted and we have access to the 6530-003 RAM, I/O, and Timer to support the KIM tape interface.

The 1K RAM is necessary because the tape ROM program exits to an address in the 6530-002 ROM program. Unless we have something at that destinaiton, the machine could run wild and do who knows what to the program. An advantage of having this 1K RAM is that you can run KIM programs as is, with appropriate vectors or subroutines in the 1K RAM. This will make it unnecessary to make patches within the main program in many cases.

You will notice that there is a disable switch in the 6530 circuit. Since my system has that movable 4K block of memory, I cannot have the 6530 or 1K RAM functional when the block is located at 1000-1FFF. I have done this by disabling the 7420 decoder IC and by forcing the R/W low to both the OR-wired 7403 on the 1K RAM and the 6530. This prevents both systems from putting data onto the data bus when they are disabled.

Now that we have the hardware, we need the software to make it work. At this point, I will leave this up to the reader. In the future, I plan to submit software information. (Boy! what a sneaky way to make you subscribe to User Notes). Besides, I have to leave room in this issue for other stuff.

145 DECODER						
D	C	B	A	K5	K6	K7
0	1	0	1	0	1	1
0	1	1	0	1	0	1
0	1	1	1	1	1	0
1	X	X	X	1	1	1

K5 for 1400-17FF - 6530 I/o, Timer, RAM
K6 for 1800-1BFF - 6530 ROM
K7 for 1C00-1FFF - 1K RAM

RS0	A6	IC-3 OUT
0	0	0
0	1	1
1	0	1
1	1	0

IC	MEM	6530
A9	K-8	14	5
A8	K-2	15	6
A7	K-4	16	7
A5	B-6	2	10
A4	B-4	7	11
A3	B-2	6	12
A2	B-8	5	13
A1	B-10	4	14
A0	B-12	8	15

27

WHAT IS KIMATH?

Some of you have asked for more details on the KIMATH program that we are making available.

KIMATH is a group of floating-point math subroutines capable of performing operations up to 16 decimal digits of precision. The functions supported by KIMATH include ADD, SUBTRACT, MULTIPLY, DIVIDE, LOG, ANTILOG, TANGENT, ARTANGENT, and SQUARE ROOT: Special subroutines are included to evaluate polynomial expressions, which can be used to approximate most math functions.

It should be stressed that KIMATH is not a complete math package, only a group of subroutines. (I/O routines are left up to the user). This means KIMATH is totally independent of any operating system dependencies and makes this package useful for most any 6502 based machines, (such as SYM & AIM) not just KIM.

Basically, you would load one or two special register areas (each register can be up to about 20 bytes in length) with the number (s) to be worked on and then call the proper subroutines in KIMATH to do the operation. KIMATH only operates on numbers in an unpacked BCD format, but routines are provided to convert to and from other data formats such as packed BCD, and unpacked ASCII for easier storage and output data formatting. (That's right, KIMATH does its calculating in BCD, not binary).

A useful addition to KIMATH, called MATHPAC, was published in Doctor Dobbs Journal! (volume 2 Issue #10). MATHPAC provides the needed I/O routines and a host of other features, such as variable storage and extended computational ranges not possible with KIMATH alone. MATHPAC needs 2K of additional memory.

Additional info on KIMATH should be gotten from the KIMATH manual - available for around $15.00 from several sources including:

Johnson Computer   A B Computers
P.O. Box 523   Box 104
Medina Oh 44256   Perkasie Pa 18944
216-725-4560   215-257-8195

        Falk-Baker
        382 Franklin Avenue
        Nutley NJ 07110
        201-661-2430

The KIMATH manual contains a complete source listing of KIMATH.

KIMATH ON CASSETTE OR EPROM FOR AIM, SYM & KIM & APPLE

KIMATH is now available assembled to any location, and comes with a sorted symbol table for easy routine lookup.

ON KIM COMPATIBLE CASSETTE FOR AIM,SYM,KIM $20.00

ON 2Kx8 5v. EPROM (TI 2516 or INTEL 2716) $80.00
        (APPLE VERSION IS ONLY AVAILABLE
ORDERING INFORMATION: ON EPROM)

You must include the following information with your order for cassette and EPROM versions of KIMATH:

Hex starting address for main program (normally
    $F800)
Hex starting address for 23 bytes of zero-page
    storage (normally $0000)
Hex starting address for 154 bytes of RAM for the
    argument registers (normally $0200)

    ORDER FROM:

    6502 USER NOTES, POB 33093, N. Royalton, Ohio
                                        44133

28

REVIEWS ETC.

THE MORE BOARD FROM T.T.I.*

Frankly, I wasn't exactly overjoyed when I received this product for review. After all, why should 3K of RAM expansion turn me on when I had over 10 times that much on my regular system.

My big change in attitude concerning the MORE board came after I realized that my other EPROM burner just would not program the Texas Instruments 2716 because of incompatabilities.

I even started to design an EPROM burner that would program the TI2716 when I suddenly realized that the solution to the problem was already at hand.

Upon a closer look at the MORE board, I discovered a very nicely engineered EPROM programming system which works with all of the popular EPROMS (2708, INTEL 2716, TI 2716, and 2516) eliminates the need for any voltages besides the usual KIM +5 and +12, and includes enough RAM on-board to solve the problem of how to burn a 2K EPROM when you have only 512 bytes of useable RAM.

The MORE board has turned my spare KIM into an EPROM programming system which has twice the capability of some of the commercial EPROM programming units at a fraction of the cost.

I haven't gotten around to burning the TI2716 yet, but I have put my disk system bootstrap into a 2708. The board performed flawlessly. Since MORE has two on-board EPROM sockets (one for programming and the other mapped into normal memory space), EPROMS can easily be copied. Of course, the EPROM burning software (which is included with the MORE in the form of complete source listings) can be relocated and burned into a 2708 for semi-permanent storage in the memory-mapped EPROM socket if you really don't need this copying capability.

The only negative things I can say about the MORE board are that I had to do a little trimming on the connector to get it hooked up to KIM and the preliminary documentation has a few soft spots.

A small file took care of the first problem and the second problem, according to T.T.I., is in the process of being corrected.

I forgot to mention that the board appears to be of industrial quality with all IC's socketed and two quick load zero insertion Textool sockets for the EPROMS. No EPROMS or RAMS are included. MORE comes fully assembled and, according to T.T.I. can also be used on the Rockwell AIM system.

I'm happy with the MORE board, plan on using more EPROMS in the near future and will probably cause a real scene when T.T.I. asks for their board back!

MORE is available for $170 from
    T.T.I., POB 2328, Cookeville, Tn 38501.

PRODUCT ANNOUNCEMENT

FLOPPY DISK FOR THE AIM-65

COMPAS Microsystems (224 S.E. 16th St, Ames, Iowa 50010) announces availability of a mini-floppy interface for the AIM-65.

"The base price of $850 includes DAIM controller board with all operating system in EPROM, power supply and one packaged disk drive. Price for an additional drive is $350. The components of the DAIM system may also be purchased on an individual basis if the user desires. Delivery is stock to sixty days. The system is currently in production. A complete operating manual will be supplied for evaluation purposes for the nominal charge of $5."

6502 FORTH

* 6502 FORTH IS A COMPLETE PROGRAMMING SYSTEM WHICH CONTAINS
  AN INTERPRETER/COMPILER AS WELL AS AN ASSEMBLER
  AND EDITOR.
* 6502 FORTH RUNS ON A   KIM-1 WITH A SERIAL TERMINAL.
  (TERMINAL SHOULD BE AT LEAST 64 CHR. WIDE)
* ALL TERMINAL I/O IS FUNNELLED THROUGH A JUMP TABLE NEAR
  THE BEGINNING OF THE SOFTWARE AND CAN EASILY BE
  CHANGED TO JUMP TO USER WRITTEN I/O DRIVERS.
* 6502 FORTH USES CASSETTE FOR THE SYSTEM MASS STORAGE DEVICE
* CASSETTE READ/WRITE ROUTINES ARE BUILT IN (INCLUDES HYPER-
  TAPE).
* 92 OP-WORDS ARE BUILT INTO THE STANDARD VOCABULARY.
* EXCELLENT MACHINE LANGUAGE INTERFACE.
* 6502 FORTH IS USER EXTENSIBLE.
* 6502 FORTH IS A TRUE IMPLEMENTATION OF FORTH ACCORDING
  TO THE CRITERIA SET DOWN BY THE FORTH INTEREST
  GROUP.
* SPECIALIZED VOCABULARIES CAN BE DEVELOPED FOR SPECIFIC
  APPLICATIONS.
* 6502 FORTH RESIDES IN 8K OF RAM STARTING AT $2000 AND
  CAN OPERATE WITH AS LITTLE AS 4K OF ADDITIONAL
  CONTIGUOUS RAM.

6502 FORTH PRICE LIST

KIM CASSETTE, USER MANUAL, AND
   COMPLETE ANNOTATED SOURCE LISTING $90.00
   ($2000 VERSION)   PLUS S&H   4.00

> OUR USER MANUAL ASSUMES SOME
> PREVIOUS KNOWLEDGE OF FORTH.
>
> IF YOU HAVE NO IDEA WHAT FORTH
> IS ALL ABOUT — SEND AN S.A.S.E
> (BUSINESS SIZE) AND
> ASK FOR A "FORTH BIBLIOGRAPHY"

USER MANUAL (CREDITABLE TOWARDS SOFTWARE
   PURCHASE) $15.00
                        PLUS S&H   1.50

AIM & SYM VERSIONS TO BE AVAILABLE
   6502 USER NOTES
   POB 33093   also for the HDE floppy
   N. ROYALTON, OHIO
                    44133

KIM SOFTWARE ON CASSETTE

FOCAL CASSETTE OPERATING SYSTEM
   ($4000-$4920) includes instructions,
   cassette and complete source list-
   ing. Price includes shipping &
   handling (see FOCAL section in this
   issue for more info)(works with ei-
   ther verison of FOCAL). $37.50

BASEBALL (from this issue)   6.00

BASEBALL source listing (16 pages)   5.00

KIMATH (specify $2000 or $F800 version)
   (includes manual erata sheet)   12.00

HEXPAWN (from issue #13)   5.00

DISASSEMBLER (from issue #14)   5.00

BANNER (from issue #14)   5.00

   These cassettes are original dumps, not cop-
ies, made with top quality 5-screw housing cas-
settes in the HYPERTAPE X 3 tape speed. Thirty
seconds of sync characters precede the program to
enable you to tune up your recorder or PLL.

   Payment must be in U.S. Funds. Overseas
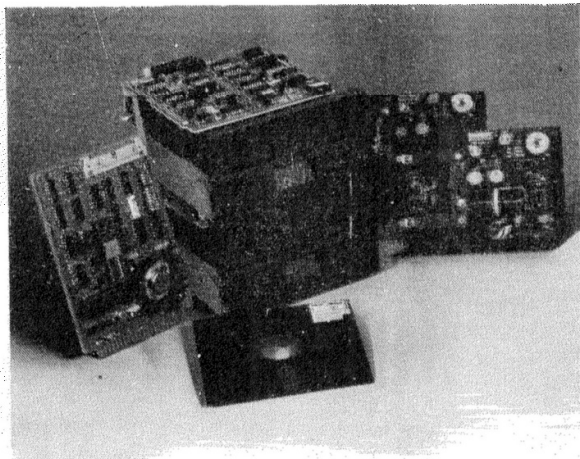customers please include $1.00 extra per cassette
for extra postage.

   6502 USER NOTES, POB 33093, N. Royalton
                                       44133