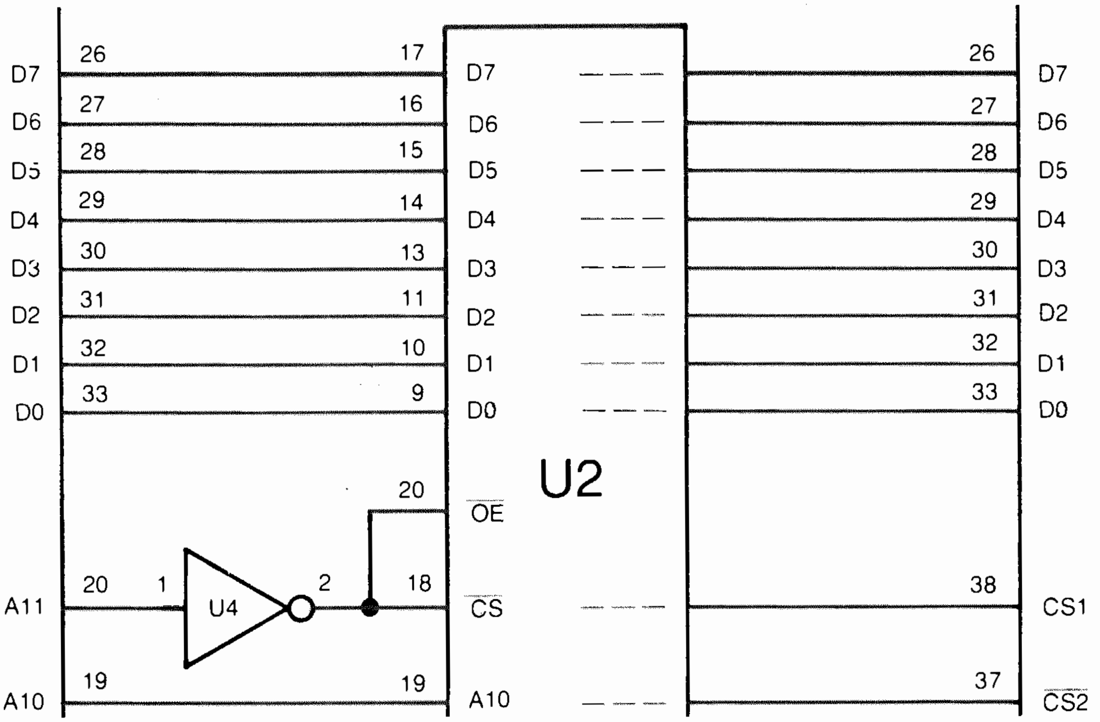
INTERACTIVE

OCTOBER 1981 ISSUE NO. 6

BREW UP A CONTROLLER



. . . see page 20

Rockwell International
where science gets down to business

EDITOR'S CORNER

FORTH AND PROM PROGRAMMER/ COED MANUALS READY

All you Forth and PROM Programmer/COED board users who received preliminary manuals with your purchase will be happy to know that the regular manuals are in!!! To get one, simply send the front cover of the preliminary manual together with your name and address (of course) and we'll rush one out to you. Send your request to SALES SUPPORT SERVICES, Rockwell Int'l, POB 3669, RC55, Anaheim, CA 92803.

Anyhow, the Forth manual (document #265) and the Prom Programmer/ COED manual (document #269) are also available for purchase. Contact your area sales office for price information.

HOME OFFICE

Electronic Devices Division
Rockwell International
3310 Miraloma Avenue
P.O. Box 3669
Anaheim, CA 92803
(714) 632-3729
TWX: 910 591-1698

EUROPE

Electronic Devices Division
Rockwell International GmbH
Fraunhoferstrasse 11
D-8033 Munchen-Martinsried
Germany
(089) 859-9575
Telex: 0521/2650

FAR EAST

Electronic Devices Division
Rockwell International Overseas Corp.
Itohpia Hirakawa-cho Bldg.
7-6, 2-chome, Hirakawa-cho
Chiyoda-ku, Tokyo 102, Japan
(03) 265-8806
Telex: J22198

CORRECTIONS TO ISSUE #5

Page 13—You may notice some problems if certain BASIC instructions are executed with the TTY drive located in page 2. Simply move the program to reside at location $00DC when using them with BASIC. The programs are completely relocatable with the only change required being to the .WOR address at the beginning.

Page 24—The GND connection on the AIM 65 is pin 1 (not L).

CORRECTIONS TO ISSUE # 4

Page 2—The new flat rate charges for out-of-warranty repairs on the AIM 65 is $59.80 (not $49.80).

Page 6—Line 2220 should read IFP=255THEN2210 (not IFP=225THEN2210).

All subscription correspondence and articles should be sent to:

EDITOR, INTERACTIVE
ROCKWELL INTERNATIONAL
POB 3669, RC 55
ANAHEIM, CA 92803

BASIC TRACE

Jeff Williams
Rockwell International

Ever wonder where you were in a BASIC program, or, how you got there from here when you can't get from here to there??? But, your program did it anyway???

When active, the following program prints out the line number of every BASIC statement just before it gets executed. Input/Output statements are left justified with a carriage return prior to execution (just to be pretty) and the line numbers are right justified in three columns.

To activate the routine, location 224 ($E0) must be poked with a non-zero value. Of course, to deactivate the trace, poke the same location with a zero. This trace function may be activated and deactivated within a BASIC program.

With a minor addition to the program, the contents of two memory locations may be monitored. Simply insert the following short "patch" between the instructions JSR SOUT and INC POS. (You'll end up with two lines containing the INC POS instruction)

```
LDA VALUE  ;

LDA BYTE1   ;ADDRESS OF THE FIRST BYTE
JSR NUMA
JSR BLANK   ;OUTPUT A BLANK
LDA BYTE2   ;ADDRESS OF THE SECOND BYTE
JSR NUMA
INC POS     ;ADD TO COLUMN COUNT
```

This technique can be expanded upon to monitor any BASIC parameter such as a variable etc.

Thanks to Steve West and Frank Nunnely for the neat idea on how to gain access to BASIC through the trap.

(Continued on page 22)

DRAMATIC PRICE CUTS!!!

In order to make Rockwell products an even bigger value, we have dropped prices on most of the RM65 board level products, the AIM 65/40, and all of the AIM 65 accessory ROMS (BASIC, Forth, PL-65, and the Assembler). Those ROM prices have been cut by more than 50%!!! Check with your local Rockwell dealer for details.

Rockwell does not assume any liability arising out of the application or use of any products, circuit, or software described herein, neither does it convey any license under its patent rights nor the patent rights of others. Rockwell further reserves the right to make changes in any products herein without notice.

AIM 65 BASIC "SCREEN EDITOR" PROGRAM

by Joe Hance
Rockwell International

One of the biggest shortcomings of the AIM 65 BASIC interpreter is the lack of any editing features, as it is, it is necessary to retype the entire line in order to correct a mistake in a BASIC line. By using this "Screen Editor" program, however, a line can be corrected by simply typing over any mistakes.

The editor is invoked by typing "LIST#X", where X is the line number of the line you wish to edit. The program "intercepts" the "LIST#" command in the page zero character fetch routine (thanks to Steve West and Frank Nunneley in INTERACTIVE #5) and sends the line to the editor buffer. The line can now be operated on by the "Screen Editor". When editing is finished, the line is forced into BASIC's line input routine (thanks to Mark Reardon of Rockwell for help with basic entry points).

The commands available are:

1) F1—Move cursor right. This key moves the cursor to the right one space.

2) F2—Move cursor left. This key moves the cursor position left one space.

3) F3—Insert at cursor. This key inserts one blank space at the cursor position. The rest of the line scrolls to the right.

4) DEL—Delete at cursor. This key deletes one character at the cursor. The rest of the line scrolls to the left.

5) CNTL F3—"∧". The "∧" symbol is now accessed with a CNTL F3 when in the editor (but not when in BASIC).

6) RETURN—Leave editor. Two returns will leave the editor and go back to BASIC after editing a line. Three returns are needed if an attempt is made to edit a nonexistent line.

All other keys, when typed, will replace the character under the cursor. The cursor is always in position number 11 on the AIM display. So the line actually moves by the cursor instead of the cursor moving past the line.

To assemble and load the program for a 4K AIM 65, type in the program without the comments to fit in less than 4K. Assemble and direct object to tape. Then initialize BASIC and limit memory size to 3695. Escape to the monitor and use the "L" command to load the editor. Reenter BASIC with the "6" command. Basic should now respond to the LIST#X command.

Example:

```
10 FOR I=1 TO 100
20 PRINT I;
30 NEXT K
```

We want to edit line 30 and change the "K" to an "I".

Type: LIST#30

and we see displayed:

```
30 NEXT K
         ∧ the cursor is here.
```

Type "F2" to move the cursor left:

```
30 NEXT K
        ∧ the cursor is now here.
```

Now type "I" to replace the "K":

```
30 NEXT I
        ∧ the cursor automatically scrolls.
```

Now press the RETURN key twice to send the line back to BASIC.

Let's check it. Type:

LIST 30

and we see: 30 NEXT I

INTERACTIVE GETS NEW PRINTER!

I've officially retired my DecWriter II printer from newsletter duty. A new Epson MX-80 is now assuming the role of generating program printouts. The MX-80 has turned out to be quite a versatile printer and quite deserving of all the praise it has received. There are a number of operating modes including compressed (132 char/line) and emphasized (it raises the paper slightly and makes another pass to fill in the dots) that make it ideal for newsletter duty. It's moderately fast (80 cps), relatively inexpensive (under $500) and seems to be very reliable. Anyhow, for those of you who would like to hook up the MX-80 to your AIM 65, stay tuned. In the next issue, we'll present the parallel interface driver software.

```
2000              ;
2000              ;
2000              ; BASIC "SCREEN" EDITOR
2000              ; FOR AIM-65 MICROCOMPUTER
2000              ;
2000              ; WRITTEN BY JOE HANCE
2000              ;
2000              ;
2000                     *=$010A
010A
010A  9B 0E              .WORD UOUT        ; SET UP USER OUTPUT VECTOR
010C                     *=$C8
00C8              ;
00C8              ; THIS IS THE "WEDGE" INTO
00C8              ; BASIC. IT INTERCEPTS
00C8              ; THE COMMANDS BEFORE
00C8              ; GOING TO BASIC
00C8              ;
00C8  4C 67 0E           JMP WEDGE
00CB  EA                 NOP
00CC                     *=$18
0018              BUFFR  *=*+70
005E                     *=$0E67
0E67              PHXY   =$EB9E
0E67              PLXY   =$EBAC
0E67              CLR    =$EB44
0E67              OUTPUT =$E97A
0E67              READ   =$E93C
0E67              OUTFLG =$A413
0E67  C9 99       WEDGE  CMP #$99          ; LOOK FOR "LIST" TOKEN
0E69  F0 08              BEQ LIST
0E6B  C9 3A              CMP #$3A
0E6D  B0 03              BCS NOTNUM
0E6F  4C CC 00           JMP $CC           ; RETURN TO BASIC
0E72  60          NOTNUM RTS
0E73  48          LIST   PHA
0E74  20 9E EB           JSR PHXY
0E77  A0 01              LDY #1            ; SET UP INDEX
0E79  B1 C6              LDA ($C6),Y       ; GET NEXT CHR
0E7B  C9 23              CMP #'#           ; IS IT A # ?
0E7D  F0 06              BEQ AOK
0E7F  20 AC EB    EXIT   JSR PLXY          ; NO # GO BACK
0E82  68                 PLA
0E83  38                 SEC               ; SET CARRY FOR BASIC
0E84  60                 RTS
0E85  E6 C6       AOK    INC $C6           ; PROCESS LIST#
0E87  D0 02              BNE AOK1
0E89  E6 C7              INC $C7
0E8B  A9 55       AOK1   LDA #'U           ; SET OUTPUT TO USER
0E8D  8D 13 A4           STA OUTFLG
0E90  A9 00              LDA #0
```

```
0E92  8D FD 0F          STA CRFLG        ; CLEAR FLAG
0E95  8D FE 0F          STA PNTR         ; CLEAR PNTR
0E98  4C 7F 0E          JMP EXIT         ; OK, DONE HERE


0E9B              ;
0E9B              ; USER OUTPUT HANDLER
0E9B              ; ALL OUTPUT FROM THE
0E9B              ; LIST COMMAND WILL
0E9B              ; COME HERE
0E9B              ;
0E9B  90 13       UOUT  BCC INIT
0E9D  68                PLA              ; GET THE CHR
0E9E  8E FF 0F          STX SAVX          ; SAVE X
0EA1  AE FE 0F          LDX PNTR         ; LOAD POINTER
0EA4  95 16             STA BUFFR-2,X    ; PUT CHR INTO BUFFER
0EA6  EE FE 0F          INC PNTR
0EA9  AE FF 0F          LDX SAVX
0EAC  C9 0A             CMP #$0A         ; END OF LINE?
0EAE  F0 01             BEQ CR
0EB0  60          INIT  RTS
0EB1              ;
0EB1              ; END OF LINE-CHANGE OUTFLG
0EB1              ; BACK TO NORMAL OUTPUT
0EB1              ;
0EB1  AD FD 0F    CR    LDA CRFLG        ; END OF LINE
0EB4  F0 08             BEQ FIRST
0EB6  A9 0D             LDA #$0D
0EB8  8D 13 A4          STA OUTFLG
0EBB  4C C4 0E          JMP EDIT         ; GO TO EDITOR
0EBE  A9 01       FIRST LDA #1           ; FIRST LF IGNORE
0EC0  8D FD 0F          STA CRFLG
0EC3  60                RTS


0EC4              ;
0EC4              ; ***** EDITOR *****
0EC4              ;
0EC4              ; ALL LINE EDITING IS DONE HERE
0EC4              ; THE VALID COMMANDS ARE:
0EC4              ; F1 - CURSOR RIGHT
0EC4              ; F2 - CURSOR LEFT
0EC4              ; F3 - INSERT AT CURSOR
0EC4              ; DEL - DELETE AT CURSOR
0EC4              ;
0EC4              ; NOTE: THE ^ CHARACTER IN BASIC
0EC4              ; CAN BE TYPED BY USING
0EC4              ; CNTL F3
0EC4              ;
0EC4              ; A RETURN ENDS THE EDITOR
0EC4              ;
0EC4  A9 00       EDIT  LDA #0
0EC6  8D FC 0F          STA COL1
```

```
0EC9  A0 00     HERE   LDY #0
0ECB  AE FC 0F         LDX COL1
0ECE  20 44 EB         JSR CLR          ; CLEAR DISPLAY
0ED1  B5 18     LOOP   LDA BUFFR,X      ; CHECK FOR END OF LINE
0ED3  C9 0D            CMP #$0D
0ED5  F0 4A            BEQ ENDLN
0ED7  20 7A E9         JSR OUTPUT       ; OUTPUT LINE
0EDA                   ; INCREMENT BOTH POINTERS
0EDA  E8               INX
0EDB  C8               INY
0EDC  C0 14     LP11   CPY #20          ; ONLY SEND 20
0EDE  D0 F1            BNE LOOP
0EE0  20 3C E9  KEY    JSR READ         ; GET A KEY
0EE3  C9 5D            CMP #']          ; IS IT AN F2 ?
0EE5  F0 61            BEQ LEFT         ; CURSOR LEFT
0EE7  C9 5B            CMP #'[          ; IS IT AN F1 ?
0EE9  F0 42            BEQ RIGHT        ; CURSOR RIGHT
0EEB  C9 5E            CMP #'^          ; IS IT AN F3 ?
0EED  F0 35            BEQ INSERT       ; INSERT CHAR
0EEF  C9 7F            CMP #$7F         ; IS IT A DELETE ?
0EF1  F0 34            BEQ DELETE
0EF3  C9 0D            CMP #$0D         ; IS IT A CR ?
0EF5  F0 33            BEQ FINIS        ; GO AWAY
0EF7  C9 1E            CMP #$1E         ; CNTL F3 ?
0EF9  D0 02            BNE F3
0EFB  A9 5E            LDA #$5E         ; CHANGE CNTL F3 TO "^"
0EFD                   ;
0EFD                   ; REPLACE CHARACTER
0EFD                   ; UNDER CURSOR WITH THE ONE
0EFD                   ; IN ACCUMULATOR
0EFD                   ; AND SCROLL
0EFD                   ;
0EFD  48        F3     PHA
0EFE                   ; CHECK FOR END OF LINE
0EFE  20 D6 0F         JSR ADD10
0F01  B5 18            LDA BUFFR,X
0F03  C9 0D            CMP #$0D
0F05  D0 0C            BNE NOCR
0F07  E8               INX
0F08  E0 45            CPX #69          ; CHECK FOR LINE TOO BIG
0F0A  D0 04            BNE STORE
0F0C  68               PLA
0F0D  4C C9 0E         JMP HERE
0F10  95 18     STORE  STA BUFFR,X
0F12  CA               DEX
0F13  68        NOCR   PLA
0F14  95 18            STA BUFFR,X
0F16  8A               TXA
0F17  38               SEC
0F18  E9 0A            SBC #10
0F1A  AA               TAX
0F1B                   ; SCROLL
0F1B  EE FC 0F  OK1    INC COL1
```

```
0F1E   4C 40 0F    OK     JMP NEGTST
0F21   4C 6B 0F    ENDLN  JMP ENDL1
0F24               ;
0F24               ; JUMP TABLE FOR OUT
0F24               ; OF RANGE RELATIVE BRANCHES
0F24               ;
0F24   4C 78 0F    INSERT JMP INSR1
0F27   4C A8 0F    DELETE JMP DEL2
0F2A   4C E1 0F    FINIS  JMP FINIS1
0F2D               ;
0F2D               ; SCROLL CURSOR RIGHT
0F2D               ;
0F2D   EE FC 0F    RIGHT  INC COL1
0F30   20 D3 0F           JSR ADD9
0F33   B5 18              LDA BUFFR,X
0F35   48                 PHA
0F36   8A                 TXA
0F37   38                 SEC
0F38   E9 09              SBC #9
0F3A   AA                 TAX
0F3B   68                 PLA
0F3C   C9 0D              CMP #$0D
0F3E   F0 08              BEQ LEFT
0F40               ; TEST FOR COLUMN ONE NEGATIVE
0F40   2C FC 0F    NEGTST BIT COL1
0F43   30 12              BMI OK2
0F45   4C C9 0E           JMP HERE
0F48               ;
0F48               ; SCROLL CURSOR LEFT
0F48               ;
0F48   CE FC 0F    LEFT   DEC COL1
0F4B   10 D1              BPL OK
0F4D   A9 F5              LDA #$F5
0F4F   CD FC 0F           CMP COL1
0F52   D0 03              BNE OK2
0F54   EE FC 0F           INC COL1
0F57   20 44 EB    OK2    JSR CLR
0F5A   A0 00              LDY #0
0F5C   AE FC 0F           LDX COL1
0F5F               ; OUTPUT BLANKS ON LINE
0F5F   A9 20       LP10   LDA #$20
0F61   20 7A E9           JSR OUTPUT
0F64   C8                 INY
0F65   E8                 INX
0F66   30 F7              BMI LP10
0F68   4C D1 0E           JMP LOOP
0F6B               ; END OF LINE
0F6B               ; OUTPUT BLANKS
0F6B   A9 20       ENDL1  LDA #$20
0F6D   20 7A E9    LP1    JSR OUTPUT
0F70   C8                 INY
0F71   C0 14              CPY #20            ; ONLY 20 BLANKS
0F73   D0 F8              BNE LP1
```

```
0F75  4C E0 0E                JMP KEY
0F78                   ;
0F78                   ;  INSERT A SPACE UNDER CURSOR
0F78                   ;
0F78  A0 00            INSR1  LDY #0
0F7A  B9 18 00         LP7    LDA BUFFR,Y
0F7D  C9 0D                   CMP #$0D
0F7F  F0 08                   BEQ MOVE
0F81  C8                      INY
0F82  C0 44                   CPY #68           ; DON'T ALLOW MORE
0F84  D0 F4                   BNE LP7             THAN 70 CHARS
0F86  4C C9 0E                JMP HERE
0F89                   ; MOVE REST OF LINE OVER
0F89  20 D3 0F         MOVE   JSR ADD9
0F8C  8A                      TXA
0F8D  8D FB 0F                STA CURSOR
0F90  B9 18 00         LP9    LDA BUFFR,Y
0F93  C8                      INY
0F94  99 18 00                STA BUFFR,Y
0F97  88                      DEY
0F98  88                      DEY
0F99  CC FB 0F                CPY CURSOR
0F9C  D0 F2                   BNE LP9
0F9E  A9 20                   LDA #$20
0FA0  C8                      INY
0FA1  99 18 00                STA BUFFR,Y
0FA4  88                      DEY
0FA5  4C 40 0F                JMP NEGTST
0FA8                   ;
0FA8                   ;  DELETE CHARACTER UNDER CURSOR
0FA8                   ;
0FA8  20 D6 0F         DEL2   JSR ADD10
0FAB                   ; CHECK FOR CR
0FAB                   ; DON'T DELETE A CR IF HERE
0FAB  B5 18                   LDA BUFFR,X
0FAD  C9 0D                   CMP #$0D
0FAF  D0 03                   BNE DEL3
0FB1  4C 40 0F                JMP NEGTST
0FB4                   ; MOVE REST OF LINE OVER
0FB4  AE FC 0F         DEL3   LDX COL1
0FB7  8A               DEL1   TXA
0FB8  18                      CLC
0FB9  69 0B                   ADC #11
0FBB  AA                      TAX
0FBC  B5 18                   LDA BUFFR,X
0FBE  CA                      DEX
0FBF  95 18                   STA BUFFR,X
0FC1  48                      PHA
0FC2  8A                      TXA
0FC3  38                      SEC
0FC4  E9 0A                   SBC #10
0FC6  AA                      TAX
0FC7  E8                      INX
```

```
0FC8  68                  PLA
0FC9  C9 0D               CMP #$0D
0FCB  F0 03               BEQ STOP
0FCD  4C B7 0F            JMP DEL1
0FD0  4C 40 0F   STOP     JMP NEGTST
0FD3             ;
0FD3             ; ADDS 9,10,OR 11 TO COLUMN
0FD3             ; TO LOCATE PROPER CURSOR
0FD3             ;
0FD3  A9 09      ADD9     LDA #9
0FD5  2C                  .BYTE $2C
0FD6  A9 0A      ADD10    LDA #10
0FD8  2C                  .BYTE $2C
0FD9  A9 0B      ADD11    LDA #11
0FDB  18                  CLC
0FDC  6D FC 0F            ADC COL1
0FDF  AA                  TAX
0FE0  60                  RTS


0FE1             ;
0FE1             ; SEND EDITED LINE
0FE1             ; BACK TO THE BASIC
0FE1             ; INPUT BUFFER
0FE1             ;
0FE1             ; MOVE LINE INTO
0FE1             ; BASIC INPUT BUFFER
0FE1  A2 00      FINIS1 LDX #0
0FE3  B5 18      LPA      LDA BUFFR,X
0FE5  C9 0D               CMP #$0D
0FE7  F0 05               BEQ QUIT
0FE9  95 16               STA $16,X
0FEB  E8                  INX
0FEC  D0 F5               BNE LPA
0FEE             ; STORE A NULL AT THE END
0FEE  A9 00      QUIT     LDA #0
0FF0  95 16               STA $16,X
0FF2             ; FIX THE STACK TO RETURN
0FF2  68                  PLA
0FF3  68                  PLA
0FF4             ; X AND Y HAVE BUFFER ADDRESS
0FF4  A2 15               LDX #$15
0FF6  A0 00               LDY #$0
0FF8             ; BASIC LINE INPUT ROUTINE
0FF8  4C 87 B2            JMP $B287
0FFB             ; RAM STORAGE LOCATIONS
0FFB             CURSOR   *=*+1
0FFC             COL1     *=*+1
0FFD             CRFLG    *=*+1
0FFE             PNTR     *=*+1
0FFF             SAVX     *=*+1
1000                      .END
```

NUMBER CONVERSION PROGRAM

Jens Grysbjerg
UNESCO, Box 3311
Dakar, SENEGAL

When working in BASIC, it's useful to have a number conversion program which goes from HEX to DECIMAL and vice versa. Here are two routines which do just that.

The first program accepts a decimal number of up to five digits and converts it to a hex number from $0000 to $FFFF. An error message is displayed if the number exceeds this range. Start this program running at $0ECE and enter the decimal number you wish to convert. If it's less than five digits long press the RETURN key to terminate it. The hex equivalent will be displayed. The DEL key may be used to correct any typing errors on input. If you'd like to do another number conversion, press the RETURN key, otherwise press ESC to go back to the monitor. The printer may be enabled to print the results if you wish.

The second program converts hex numbers ($0000 to $FFFF) to decimal and starts running at $0F62. Otherwise, it works just like the previous routine but with the number of digits you can input limited to four.

The programs use 3 zero-page locations ($F0, $F1 and $F2) which are normally used for the Editor 'F' command. These locations are outside the zero-page area used by BASIC so when you need to convert numbers, you can exit and reenter BASIC without damaging your program. Be sure to limit the memory size to 3789 ($0ECD) when BASIC is first entered.

```
2000             ;THIS ROUTINE CON-
2000             ;VERTS DECIMAL NUM-
2000             ;BERS UP TO 65535
2000             ;TO HEXADECIMAL
2000             INT    =$00F0
2000             LO     =$00F1
2000             HI     =$00F2
2000             ERROR  =$E391
2000             CURPO2 =$A415
2000             RDRUB  =$E95F
2000             RB2    =$E95C
2000             BLANK  =$E83E
2000             EQUAL  =$E7DB
2000             OUTPUT =$E97A
2000             NUMA   =$EA46
2000             READ   =$E93C
2000             CRLOW  =$EA13
2000             DIBUFF =$A438
```

```
2000                         *=$0ECE
0ECE
0ECE                  START

0ECE                  ;CLEAR HI AND LO
0ECE  A9 00                  LDA #0
0ED0  85 F2                  STA HI
0ED2  85 F1                  STA LO

0ED4                  ;OUTPUT 3 BLANKS
0ED4  20 3E E8               JSR BLANK
0ED7  20 3E E8               JSR BLANK
0EDA  20 3E E8               JSR BLANK

0EDD                  ;GET A CHR, ECHO D/P
0EDD  20 5F E9        NXTCHR JSR RDRUB


0EE0                  ;RETURN?
0EE0  C9 0D           TEST   CMP #$0D
0EE2  F0 15                  BEQ FIVE

0EE4                  ;DECIMAL CIFFER?
0EE4  C9 30                  CMP #$30
0EE6  90 04                  BCC INVALI
0EE8  C9 3A                  CMP #$3A
0EEA  90 06                  BCC VALID

0EEC                  ;INVALID, BACKSPACE
0EEC  20 5C E9        INVALI JSR RB2
0EEF  4C E0 0E               JMP TEST

0EF2                  ;5 DIGITS ?
0EF2  A0 07           VALID  LDY #7
0EF4  CC 15 A4               CPY CURPO2

0EF7  B0 E4                  BCS NXTCHR

0EF9                  ;OUTPUT SP
0EF9  20 3E E8        FIVE   JSR BLANK

0EFC                  ;ADJUST TO MSD
0EFC  A2 03                  LDX #3

0EFE                  ;GET A DIGIT
0EFE  BD 38 A4        NEXT   LDA DIBUFF,X

0F01                  ;ALL DIGITS DONE?
0F01  C9 20                  CMP #'
0F03  F0 08                  BEQ DONE

0F05                  ;CONVERT TO DECIMAL
0F05  20 33 0F               JSR CONV
```

```
0F08             ;NUMBER > 65535?
0F08  B0 23             BCS OVERFL

0F0A             ;SET UP NEXT DIGIT
0F0A  E8                INX
0F0B  90 F1             BCC NEXT

0F0D             ;OUTPUT  =  SP  $
0F0D  20 D8 E7   DONE   JSR EQUAL
0F10  20 3E E8          JSR BLANK
0F13  A9 24             LDA #'$'
0F15  20 7A E9          JSR OUTPUT

0F18             ;RESULT TO D/P
0F18  A5 F2             LDA HI
0F1A  F0 03             BEQ SUPRES
0F1C  20 46 EA          JSR NUMA
0F1F  A5 F1      SUPRES LDA LO
0F21  20 46 EA          JSR NUMA

0F24             ;WAIT FOR ANY KEY
0F24  20 3C E9   WAIT   JSR READ

0F27             ;CR AND LF TO D/P
0F27  20 13 EA          JSR CRLOW

0F2A  4C CE 0E          JMP START

0F2D             ;NUMBER > $FFFF,
0F2D             ;PRINT 'ERROR'
0F2D  20 91 E3   OVERFL JSR ERROR

0F30  4C 24 0F          JMP WAIT
0F33                    *=*
0F33             ;WITH THANKS TO

0F33             ;LEO SCANLON

0F33             ;ASCII,SO CLEAR MSD
0F33  29 0F      CONV   AND #$0F
0F35  85 F0             STA INT

0F37             ;SAVE OLD VAL ON STK
0F37  A5 F2             LDA HI
0F39  48                PHA
0F3A  A5 F1             LDA LO
0F3C  48                PHA

0F3D             ;MULTIPLY BY 4
0F3D  06 F1             ASL LO
0F3F  26 F2             ROL HI
0F41  06 F1             ASL LO
0F43  26 F2             ROL HI
```

```
0F45             ;ADD OLD VALUE
0F45  68                PLA
0F46  65 F1             ADC LO
0F48  85 F1             STA LO
0F4A  68                PLA
0F4B  65 F2             ADC HI
0F4D  85 F2             STA HI

0F4F             ;MULTIPLY BY 2
0F4F  06 F1             ASL LO
0F51  26 F2             ROL HI

0F53             ;OVERFLOW?
0F53  B0 0C             BCS END

0F55             ;ADD NEW VALUE
0F55  A5 F0             LDA INT
0F57  65 F1             ADC LO
0F59  85 F1             STA LO
0F5B  A5 F2             LDA HI
0F5D  69 00             ADC #0
0F5F  85 F2             STA HI

0F61  60         END    RTS
0F62                    .END
```

```
2000             ;THIS ROUTINE CON-
2000             ;VERTS HEXADECIMAL
2000             ;NUMBERS UP TO FFFF
2000             ;TO DECIMAL
2000             FLAG   =$00F0
2000             LO     =$00F1
2000             HI     =$00F2
2000             NOUT   =$EA51
2000             BLANK  =$E83E
2000             OUTPUT =$E97A
2000             DIBUFF =$A438
2000             RDRUB  =$E95F
2000             CURPO2 =$A415
2000             EQUAL  =$E7D8
2000             READ   =$E93C
2000             RB2    =$E95C
2000             CRLOW  =$EA13
2000             PACK   =$EA84
2000             HEX    =$EA7D
2000                    *=$0F62
0F62
```

```
0F62                START

0F62                ;OUTPUT 3 SP AND 1 $
0F62  20 3E E8             JSR BLANK
0F65  20 3E E8             JSR BLANK
0F68  20 3E E8             JSR BLANK
0F6B  A9 24                LDA #'$'
0F6D  20 7A E9             JSR OUTPUT

0F70                ;CLEAR DIBUFF+3
0F70  A9 00                LDA #0
0F72  8D 3B A4             STA DIBUFF+3

0F75                ;GET A CHR, ECHO D/P
0F75  20 5F E9      NXTCHR JSR RDRUB

0F78                ;RETURN?
0F78  C9 0D         TEST   CMP #$0D
0F7A  F0 12                BEQ FOUR

0F7C                ;HEXADECIMAL CHR?
0F7C  20 84 EA             JSR PACK
0F7F  90 06                BCC VALID

0F81                ;NOT HEX, SO BACKSP
0F81  20 5C E9             JSR RB2
0F84  4C 78 0F             JMP TEST

0F87                ;4 DIGITS?
0F87  A0 07         VALID  LDY #7
0F89  CC 15 A4             CPY CURPO2

0F8C  B0 E7                BCS NXTCHR

0F8E                ;ADJUST X TO CURPO2
0F8E  AE 15 A4      FOUR   LDX CURPO2
0F91  CA                   DEX

0F92                ;Y = BYTE NO.
0F92  A0 00                LDY #0

0F94                ;HI-NIBBLE ASCII/HEX
0F94  BD 37 A4      PAKNXT LDA DIBUFF-1,X
0F97  20 7D EA             JSR HEX

0F9A                ;LO NIBBLE ASCII/HEX
0F9A  BD 38 A4             LDA DIBUFF,X
0F9D  20 84 EA             JSR PACK

0FA0  99 F1 00             STA LO,Y
```

```
0FA3                ;NXT ASCII DBYTE
0FA3  CA                   DEX
0FA4  CA                   DEX
0FA5  C8                   INY

0FA6                ;ALL CHR PACKED?
0FA6  E0 04                CPX #4
0FA8  B0 EA                BCS PAKNXT

0FAA                ;'SP = SP' TO D/P
0FAA  20 3E E8             JSR BLANK
0FAD  20 D8 E7             JSR EQUAL
0FB0  20 3E E8             JSR BLANK

0FB3                ;CLEAR FLAG
0FB3  A0 00                LDY #0
0FB5  84 F0                STY FLAG

0FB7                ;COUNT = 0
0FB7  A2 00         NXTDIG LDX #0
0FB9  38                   SEC

0FBA                ;SUBTRACT LOW
0FBA  A5 F1         SUBT   LDA LO
0FBC  F9 F7 0F             SBC TABL,Y
0FBF  85 F1                STA LO

0FC1                ;SUBTRACT HIGH
0FC1  C8                   INY
0FC2  A5 F2                LDA HI
0FC4  F9 F7 0F             SBC TABL,Y

0FC7                ;BACK TO LOW
0FC7  88                   DEY
0FC8                ;NEGATIVE?
0FC8  90 05                BCC ADDBCK

0FCA                ;STORE HI & CONTINUE
0FCA  85 F2                STA HI
0FCC  E8                   INX
0FCD  B0 EB                BCS SUBT

0FCF                ;TOO FAR, SO ADDBACK
0FCF  A5 F1         ADDBCK LDA LO
0FD1  79 F7 0F             ADC TABL,Y
0FD4  85 F1                STA LO

0FD6                ;DIGIT ZERO?
0FD6  8A                   TXA
0FD7  D0 04                BNE NOZERO
0FD9  24 F0                BIT FLAG
0FDB  10 06                BPL SUPRS
```

TIDBITS

Users of AIM 65 systems who would like to expand their keyboards will find a dip cable that has piggyback sockets on both ends of interest. This allows another 16 pin dip to be plugged in on top of the cables dip plug at either end of the cable.

It's available from:
ARIES ELECTRONICS
BOX 130
FRENCHTOWN, N.J. 08825

Order part #16-XXX-208, where XXX is the length in inches, i.e. 12" = 012.

Cost 12" @ 11.72 ea., 24" @ 14.00 ea., 36" @ 14.00 ea.—other lengths available

R. Riley
Box 4310
Flint, MI 48504

```
0FDD            ;SET FLAG
0FDD  38        NOZERO SEC
0FDE  66 F0            ROR FLAG

0FE0            ;OUTPUT DIGIT
0FE0  20 51 EA         JSR NOUT

0FE3            ;NEXT EXP OF 10
0FE3  C8        SUPRS  INY
0FE4  C8               INY

0FE5            ;DONE 4 DIGITS?
0FE5  C0 08            CPY #8
0FE7  90 CE            BCC NXTDIG

0FE9            ;YES, OUTPUT REMAIND
0FE9  A5 F1            LDA LO
0FEB  20 51 EA         JSR NOUT

0FEE            ;WAIT FOR ANY KEY
0FEE  20 3C E9         JSR READ

0FF1            ;CLEAR & GOTO START
0FF1  20 13 EA         JSR CRLOW
0FF4  4C 62 0F         JMP START
0FF7  10 27     TABL   .WOR 10000
0FF9  E8 03            .WOR 1000
0FFB  64 00            .WOR 100
0FFD  0A 00            .WOR 10
0FFF                   *=*
0FFF                   .END
```

EASIER USR FUNCTION USE

George Meldrum
Rockwell International

When using Basic, it is often necessary to "drop" into machine language for certain operations. With AIM 65 BASIC, this is accomplished with the USR function. The starting address of the machine language routine needs to be "poked" into memory locations $0004 and $0005 and the routine called with a statement something like I=USR(Y) where 'I' is a variable which can be returned to BASIC from the machine code and 'Y' is a variable which can be passed to the machine language routine from BASIC. We'll discuss how to use these variables in a moment.

Normally, if multiple machine language subroutines are to be used, each one of their addresses must be converted to decimal and "poked" into the appropriate locations before they can be used. This can easily lead to errors and takes up some room in the program.

What I have written is a sort of a subroutine "distributor". That is, all subroutine calls get routed through a special machine language routine that determines exactly which of the subroutines gets called. It uses a variable passed from Basic (like the 'Y' variable) to figure this out.

Now, about those variables. When we execute the statement I=USR(Y), the 'Y' variable gets stuffed into a special Floating Point Accumulator in memory. Since a typical machine language program cannot readily use this number in its floating point format, it must usually be converted to an integer. Fortunately, BASIC contains such a subroutine to do that. It's located at $BEFE and converts this floating point format number to a two-byte signed integer in locations $00AC (MSB) and $00AD (LSB). Simply perform a JSR $BEFE instruction to accomplish this. Of course, this variable 'Y' must be an integer within the range of +32,767 to −32,768 or an FC error will occur.

A two-byte signed integer can also be returned to BASIC through the variable 'I' (see above) by placing the MSB of the integer in the 6502 Accumulator and the LSB in the Y register and using the instruction JSR $C0D1 to convert that number to a floating point format and placing it in the Floating Point Accumulator. Upon returning to BASIC via an RTS instruction, that value will be found in the 'I' variable.

As we said before, it's the variable that gets passed FROM BASIC that determines which of the machine language subroutines will get called. The subroutine distributor takes this variable and indexes its way into a list of subroutine addresses (see MATRIX in the listing). The order that the subroutine addresses are placed in this list determines what value the variable will have to be to call it. For example, if you wish to call SUB0 (in the listing) the variable would have to equal zero. To call SUB1, the variable would have to equal 1, and so on.

```
2000                    ;********************************
2000                    ;**                            **
2000                    ;**   PROGRAM TO IMPLEMENT THE **
2000                    ;**    USR FUNCTION OF BASIC   **
2000                    ;**      BY GEORGE MELDRUM     **
2000                    ;**        JUNE 29, 1981       **
2000                    ;**                            **
2000                    ;********************************


2000                    ;ZERO PAGE EQUATES
2000                    VECTOR =$D7             ;JUMP VECTOR FOR SUBROUTINES
2000                    LSB    =$AD             ;LOW BYTE FROM FPHEX ROUTINE

2000                    FPHEX  =$BEFE           ;CHANGE FLOATING POINT TO HEX

2000                           *=$F00           ;STARTING ADDRESS
0F00

0F00  20 FE BE                 JSR FPHEX        ;CONVERT ARGUMENT TO HEX
0F03  A5 AD                    LDA LSB          ;GET ARGUMENT
0F05  0A                       ASL A            ;MAKE IT TWICE AS LARGE
0F06  AA                       TAX              ;PUT IT IN INDEX REGISTER
0F07  BD 15 0F                 LDA MATRIX,X     ;GET LOW BYTE OF ADDRESS
0F0A  85 D7                    STA VECTOR       ;PUT IT IN JUMP VECTOR
0F0C  E8                       INX
0F0D  BD 15 0F                 LDA MATRIX,X     ;GET HIGH BYTE
0F10  85 D8                    STA VECTOR+1     ;PUT IT INTO JUMP VECTOR
0F12  6C D7 00                 JMP (VECTOR)     ;JUMP TO SUBROUTINE


0F15  1B 0F             MATRIX .WORD SUB0       ;STARTING ADDRESSES OF
0F17  1F 0F                    .WORD SUB1       ;THE SUBROUTINES
0F19  23 0F                    .WORD SUB2


0F1B                    ;EXAMPLES OF SUBROUTINES

0F1B  20 A3 E7          SUB0   JSR $E7A3
0F1E  60                       RTS

0F1F  20 A7 E7          SUB1   JSR $E7A7
0F22  60                       RTS

0F23  20 F0 E9          SUB2   JSR $E9F0
0F26  60                       RTS

0F27                           .END
```

CPU CLOCK CIRCUITS

Rockwell is now recommending an alternative clock circuit to the ones that were presented on page 2-16 of the 6502 Hardware Manual. Evidently, the RC Network and the Parallel Mode Crystal Controlled Oscillator just haven't proved reliable enough in operation. (Something to do with the internal design of the 6502). This problem affects 6502's from ALL three manufacturers.

Here is the recommended clock oscillator circuit and some additions to it which will allow the use of low-cost crystals and/or be able to operate with slow memory or peripheral devices.

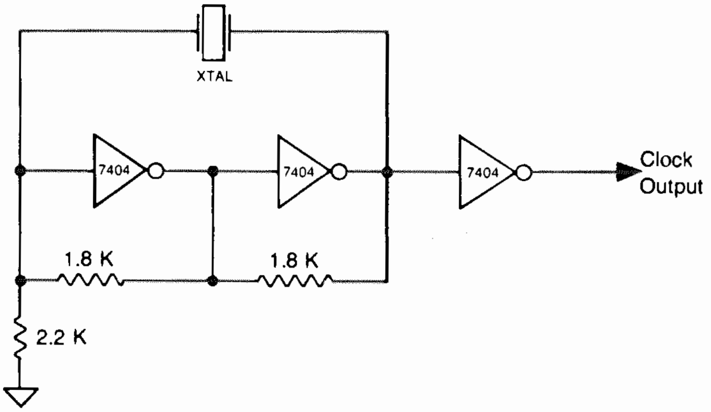


Figure 1 BASIC CRYSTAL OSCILLATOR CIRCUIT

A 1 or 2 MHz crystal can be used in the circuit in figure 1 to directly drive the single phase clock input of an R6500 family CPU. In this case, you'll need to connect the output to the phase ϕ (IN) pin on the CPU (pin #37 on the R6502).

Perhaps you'd like to use a low-cost crystal or, maybe you need a two-phase clock for driving an R6512, for example. You can do both with just one TTL package shown in figure 2.

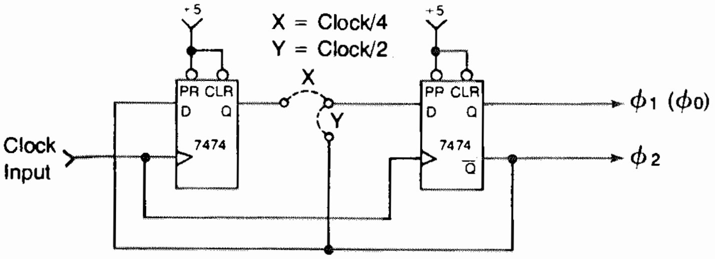


Figure 2 DIVIDER/TWO PHASE CIRCUIT

To use this circuit, you need a crystal either two or four times faster than the desired system clock rate. The position of the jumper ('X' or 'Y') determines whether the circuit will divide the incoming clock frequency by two or four. For a really cost effective clock design, you can use a 3.5795 color tv crystal and divide it down by four to get system clock freq. of around 900 KHz. (close enough to 1 MHz for most applications.) Or, if you plan on using an R6551 ACIA in your design, you can avoid having to use two crystals by using the 1.8432 MHz baud rate crystal in the system clock and divide it by two to provide about a 920 KHz clock for your CPU. The signal from the last inverter gate in the clock circuit will go directly to your ACIA chip. By the way, this same divider circuit is used on the AIM 65 to divide a 4 MHz clock down to 1 MHz.

The outputs from the second section of the 7474 flip-flop can be used as a two phase clock circuit. We've verified this by installing an R6512 in our AIM 65. Two very minor mods were required but it works great. (Since any mods to your AIM 65 will invalidate your warranty, I don't recommend that you try this. But, if you HAVE to know what we did to get an R6512 running in an AIM 65, here it is: install a jumper from pin 8 of Z10 to pin 3 of Z9 and another jumper from pin 36 of Z9 to pin 37 of Z9).

There are circumstances, such as when you have a slow block of memory or a slow peripheral device, when you would like to have your system run at full speed at all times except when you are accessing that slow section of memory or peripheral device. Well, the circuit in figure 3 will help you do just that.

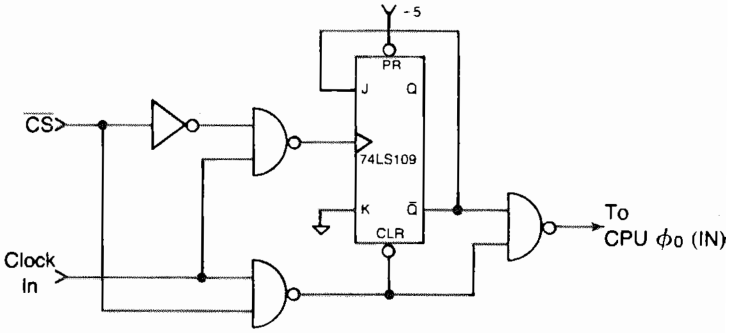


Figure 3 CLOCK STRETCHING CIRCUIT

The CS input gets connected to the low true chip select that enables the slow memory or peripheral. Whenever that signal is low (indicating that the peripheral or memory is being selected) the clock input signal gets divided in half to slow the CPU down. When the CS line is high, everything works normally (the clock signal goes through the circuit unaltered).

TEXT BUFFER DATA RECOVERY TECHNIQUES

by Dr. Lawrence A. Ezard
2149 Kentwood Dr.
Lancaster, PA 17601

This section suggests ways to ''recover'' the information in the Text Buffer if you have inadvertently re-initialized the Editor with an E command before permanently storing the old Test Buffer contents onto a cassette tape.

The effect of an inadvertent E command depends entirely on how far you have progressed since typing E. Consider the following situations:

1. If you merely typed E, and have not yet responded to the FROM= prompt, the original Text Buffer contents are still intact, and you can escape to the Monitor by pressing ESC. The contents of 00DF to 00E6 are also intact.

2. If you typed in an address in response to the FROM= prompt, and have pressed RETURN, but then pressed ESC the Editor will have stored the specified starting address in two parameters in memory—BOTLN (addresses $00EI and $00E2) and TEXT (addressed $00E3 and $00E4). However, the end-of-text character, $00 will not yet be stored in the starting address location.

3. If you typed an address and RETURN in response to both the FROM= and TO= prompt and then press ESC, the Editor will have stored the specified starting address in TEXT (addresses 00E3 and 00E4) and the specified ending address in END (addresses 00E5 and 00E6). The value contained at NOWLN (addresses 00DF and 00E0) and the value contained at BOTLN (addresses 00E1 and 00E2) will be the specified starting address. The end-of-text character, $00, will be stored in the specified starting address location.

As you can see, an inadvertent E command may do as little damage as affecting no Text Buffer locations (1 above) or only one Text Buffer location and some parameters in memory or it may affect some—or most, or all—of the information in the Text Buffer (4 above). Clearly, your recovery procedure depends on how much damage was done, but here are the corrective steps you need to take to reconstruct the original Text Buffer:

1. If you responded to the FROM= with ESC all addresses associated with NOWLN, BOTLN, TEXT and END should be unchanged and the text buffer memory should be unchanged. Use the M command to assure that this is true.

2. If you responded to the FROM= prompt with the address then realized that a mistake had occurred and you pressed ESC:

   A. The addresses associated with TEXT and BOTLN must be restored using the M and / command.

   B. Address information at NOWLN and END as well as the text buffer memory should be checked to be sure that it is unchanged and satisfactory using the M command.

3. If you responded to the FROM= and TO= prompt with address information and then pressed ESC:

   A. The addresses associated with NOWLN, BOTLN, TEXT, and END must be restored using the M and / commands.

   B. Since the address specified in the response to the FROM= prompt contains the end-of-text character, 00, this data must be restored to its original ASCII code value using the M and / command.

4. If you responded to the FROM= and TO= prompt with address information and also entered some text the restoration procedure is as follows:

   A. Use the M command to display the current address associated with BOTLN (contents of address 00EI and 00E2). Display the contents of this address and use the / command to change the contents of this location from hexadecimal 00 to hexadecimal 40 corresponding to ASCII code character@. For example, if the current data at 00EI is 0B (low order byte address) and the current data at 00E2 is 02 (high order byte address) then the M command would be used to display the contents of address 020B. The value of this address is the end-of-text character 00 which should be changed to an easily recognized, valid ASCII code (such as 40 for the symbol @) which occurs nowhere else in text memory space. This means that it will be possible to easily find this character later using the F command and change it to its correct ASCII code using the C command.

   B. Using the M and space commands search memory from the correct original starting address using the M and SPACE commands until the entry 0D followed by the end-of-text character 00 is found. The address associated with the 00 is the end of text for the original text buffer. This address should be stored in BOTLN (addresses 00El and 00E2).

   C. The addresses associated with NOWLN, TEXT and END must be restored. Use the M and / commands to restore TEXT and END to their original values. Set the value of NOWLN equal to the original value of TEXT. This sets NOWLN to the beginning of the text.

D. Finally, the undesired lines of text can be deleted using the K command. The original desired lines of text can be entered into the text buffer using the I or R command.

After all the recovery procedures above have been completed the actual recovery should be verified. Use the T command to re-enter the text editor and display the top line. The D command can then be used to move down a few lines to assure proper operation. The B command should be used to verify that the last line is fetched and printed. The U command could be used to print a few lines above the last line of text to assure proper operation. If desired the L command can be used to list all the lines of text.

TEXT BUFFER DATA RECOVERY USING CASSETTE TAPE

A cassette tape recording should always be made of the information in the text buffer memory. Then if vital information is inadvertently destroyed the cassette tape can be used to restore the information using the E command.

OTHER TEXT BUFFER DATA RECOVERY TECHNIQUES

An analysis of the operation of the text editor reveals that proper operation of the text editor commands requires two sets of conditions.

1. The addresses associated with NOWLN, BOTLN, TEXT, and END must be correct.

2. The only occurrence of 00 in the entire text buffer memory must be at the address specifed by BOTLN. Furthermore, the 00 data must follow the ASCII code 0D for carriage return. If there are any 00 entries prior to the actual end of the text it will not be possible for commands such as D, F, and C to go beyond the first occurrence of the 00.

ADDRESS	PARAMETER	PARAMETER NAME
00DF	Line pointer address low byte	NOWLN
00ED	Line pointer address high byte	
00E1	Actual text ending address low byte	BOTLN
00E2	Actual text ending address high byte This is the address of the end-of-text character 00.	
00E3	Text Buffer starting address low byte	TEXT
00E4	Text Buffer starting address high byte	
00E5	Text Buffer ending address low byte	END
00E6	Text Buffer ending address high byte	

With the above information a recovery technique can be formulated.

1. Use the M and / command to set TEXT to the first address in the text buffer memory. Address 00E3 should be set to the low order byte starting address. Address 00E4 should be set to the high order byte starting address.

2. Use the M and / command to set NOWLN to the first address in the text buffer memory. Address 00DF should be set to the low order byte starting address. Address 00E0 should be set to the high order byte starting address.

3. Use the M and / commands to set END to the last available address in the text buffer memory. Address 00E5 should be set to the low order byte ending address. Address 00E6 should be set to the high order byte ending address.

4. The most difficult task now left is to restore the proper address associated with BOTLN. Address 00E1 must contain the low order byte address of BOTLN and address 00E2 must contain the high order byte address of BOTLN.

   A. If the address associated with BOTLN was recorded before information in the text buffer memory was destroyed this original address should be entered for BOTLN using the M and / commands. If the BOTLN address is not known it must be found by the method outlined below.

   B. In either of the cases the presence of any 00 entry prior to the correct BOTLN address must be found and restored to its original value. This can be done in the following manner:

      (1) Re-enter the text editor with the T command.

      (2) Use the F command to search for a character that you are sure does not exist in the memory space (an example is!)

      (3) Since the character is not found the END message will be displayed or the display will be blank. Now exit the text editor with the Q command.

      (4) The M command followed by the address 00DF is now entered to find the value of the current active line specified by the line pointer, NOWLN. The contents of address 00DF is the low order byte address of NOWLN. The contents of address 00E0 is the high order byte address of NOWLN.

      (5) The NOWLN address is the address of the first byte of data on the line *above* the line containing the data 00.

      (6) Use the M command to access the data on the line specified by NOWLN by typing M followed by the NOWLN address.

(7) Use the SPACE command to search successive memory locations for the occurrence of 00.

(8) If this occurrence is undesirable use the / command to change the 00 to an easily recognized character that is used nowhere else in memory. The hexadecimal value 40 corresponding to the ASCII character @ is probably a good choice.

(9) Repeat steps B(1) through B(8) until all undesirable 00 entries are deleted from the text memory.

C. The desirable end-of-text character 00 entry can be recognized because it will satisfy two requirements.

(1) The desirable 00 must follow the carriage-return ASCII code 0D.

(2) When the *address* of the desirable end-of-text character 00 is placed in BOTLN correct operation of the text editor commands will be restored. This can be checked with commands such as T, B, U, D, and F.

D. There is just one final step required to restore the text editor data. In step B(8) above any undesirable 00 entries were changed to 40 corresponding to the ASCII code character @. All these @ characters must be restored to their original correct ASCII code. This is most easily done using the text editor.

(1) Re-enter the text editor using the T command.

(2) Use the F command to find each @ character.

(3) When this line is found use the C command to change the @ character to its original correct value. The operator must be able to recognize the correct value to insert by reading the line.

MULTIPLE TEXT BUFFERS

It is possible to have several Text Buffers reside in memory at the same time. The operating rules are quite simple.

1. Each Text Buffer memory block to be set up must be initialized by using the E command.

2. Before initializing the next Text Buffer the address parameters associated with NOWLN, BOTLN, TEXT and END in memory locations 00DF to 00E6 must be recorded for future use.

3. To access a particular Text Buffer the operator must load the particular Text Buffer address parameters associated with NOWLN, BOTLN, TEXT, and END in their respective memory locations.

4. The actual re-entry to the Text Buffer is then achieved from the AIM 65 monitor using the T command.

TEXT LINE LENGTH LIMITATIONS

When using the text editor in the *read* mode there is a maximum limit of 60 characters allowed on a single line. If an attempt is made to enter more than 60 characters from the keyboard the result is that the characters are not entered and there is no response. The RETURN key should be pressed to terminate this line.

The change command, C, can be used to add characters, delete characters, or change characters on a line. If using the C command results in more than 60 characters being placed on a line it is possible that the text editor will not respond to key commands from the keyboard and that the response, if any, will be unpredictable. To regain control the operator can use the reset switch to re-enter the AIM 65 monitor. The text editor can now be re-entered with the T command. The F and K commands can be used to find and delete text lines which exceed 60 characters. The desired text information can then be added using the I command.

Before the C command is used to add characters to a line it is recommended that the operator examine the line length to be sure that the new line length will not exceed 60 characters when the change has been completed. -⊖-

SUPER-SIMPLE SINGLE-LINE DISASSEMBLER

You want to hear the simplest method of disassembling a single instruction line to the display?

Turn the printer off and enter the 'K' command as usual followed by the starting address. When you get the '/' prompt press the '.' (period) key BUT DON'T RELEASE IT YET. The first instruction should now be dissassembled on the display. Now, hold down any other key (the comma key is convenient) and then release the period key. At this point the second instruction will be displayed. Hold down the period ('.') key again and release the comma (',') key. Another line will be displayed. If you want to skip ahead a number of instructions, release both keys and watch the display. When you wish to stop it, simply hold down a key.

Get it? I'll leave it up to you to figure out exactly why it works.

But we should all thank Kurt Peter (Kolner Str. 6, 6053 OBERTSTRAUSEN 2, West Germany) for the tip. What a great new feature he discovered. Thanks Kurt! -⊖-

LETTERS TO THE EDITOR

Dear Editor,

In the back of the AIM 65 BASIC USER MANUAL (Appendix F), you present a program which converts a hex number to a decimal one. The only problem with it is that the range of hex numbers is limited to from $0000 to $7FFF. I modified the Basic portion slightly to handle hex numbers up to $FFFF. Here's the new program:

```
1  PRINT "HEX/DEC CONVERTER"
2  PRINT "TYPE-IN 4 FIGURE HEX NUMBER"
5  POKE 4,161: POKE 5,15
10 DIM H (4)
15 INPUT H$
20 FOR I=1 TO 4
25 H (I)=ASC (MID$ (H,I,1))
30 POKE 4048+I,H (I)
35 NEXT
40 X=USR (I)
45 IF X<0 THEN X=65536-ABS (X)
50 PRINT X
55 GOTO 15
```

Hope you find it useful.

Sincerely,
M.I. Forsyth-Grant
Catworth Court, Rhydspence,
Whitney, Hereford
ENGLAND HR3 6EY

Dear Editor,

I have read with interest Mark Reardon's article "TTY Output Utility Programs" in Issue 5 of "Interactive". I have had the same problem when I wanted to switch between keyboard and TTY under software control in order to enter data from the keyboard and use the TTY to print the processed and formatted data.

After using a poor approach with a USR routine that was very slow I found a much simpler way which permits you to switch from TTY to keyboard control and back completely under software control.

This method manipulates the status of bit 3, port B (PB3) of the Z 32 VIA. Normally this bit is programmed as an input and its state is determined by the position of S3, the TTY-KBD switch. By executing the instruction:

POKE 43010,63 in BASIC, or
LDA#$3F
STA$A802 in assembler language this bit is re-progammed as an output. After this has been done the state of the bit can be set high=Keyboard by executing:

POKE 43008,252 in BASIC, or
LDA#$FC
STA$A800 in assembler language.

It is set low=TTY by executing:

POKE 43008,244 in BASIC, or
LDA#$FA
STA $A800 in assembler language.

The switch should be set in position "KBD". The method also works when it is set to "TTY" but the software and the hardware try to pull the level at the pin in different directions and the VIA might get somewhat hot. The Baud rate setting also has to be initialized, either by entering the baud rate manually or, if the TTY has a keyboard by doing the normal TTY startup once.

Erich A. Pfeiffer, Ph.D., P.E.
265 Viejo Street
Laguna Beach, CA 92651

Dear Mr. Rehnke:

I find that the MCT-2 for the safety isolation circuit on page 4 of Interactive No. 4 is difficult to obtain.

But the 4N33 in the Application Note 230, RS-232C Interface For AIM 65 is easy to obtain.

Now, in Interactive No. 5, Easy RS 232C, I see you are using the MCT-2 instead of something like a 4N33.

When people write constructive articles I wish they would give a number of devices that would work equally as well. You may want to list some of these in your next issue.

Cordially,
R. D. Overby
805 North 11th Avenue
Fargo, North Dakota 58102

HEAR YOUR AIM 65

Robert P. Barrett
Messiah College
Grantham PA 17027

A small addition to the AIM that has helped much in saving/loading cassettes is a crystal earphone. It is soldered to the ground and the AUDIO IN line from the recorder. Both lines are on top of the board & the AUDIO IN can be located as it goes from C-11 to a hole thru the circuit board and finally on to pin L of edge connector J1.

A crystal earphone has a high impedance and does *not* draw significant power. Most cassette player/recorders send the signal being recorded back out the monitor jack so that the earphone "listens in" during both the loading and saving (dumping) operations.

Hearing what is being recorded or played provides the following help:

1.) It is easier to search a cassette for the start of a program.

2.) There is an audible reminder of the tap gap setting and if it is still at the default value.

3.) One can sometimes hear tape drop out and other recording problems.

4.) The operator is afforded the general pleasure of hearing a tape going into the AIM and seeing the tape blocks being counted.

The proper crystal earphone is available for $1.99 from Heathkit (part no. 401-36)

(EDITOR'S NOTE: Mr. Barret was kind enough to send me the proper crystal earphone so I could try it out. Works great!!!)

AIM 65 COURSE TO BE OFFERED

The Foundation for Computer Education Inc has announced plans for holding a number of microcomputer seminars around the country. These three day seminars are based on the AIM 65 and are intended to introduce the student to microcomputer hardware, software and interfacing. The fee for the course is $850.00 and includes the AIM 65 as well as some additional documentation and class notes. For more information on the schedule and the cities involved contact the company at Box 668, Ogden, Iowa 50212. Their phone number is 515-275-4524 or 712-843-2000.

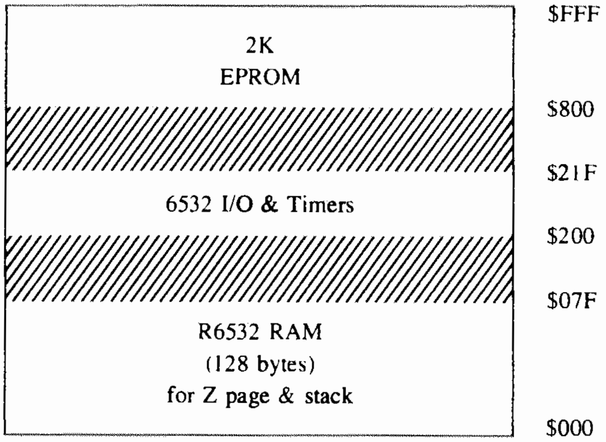
LOW COST CONTROLLER RECIPE

There are certain applications where it makes sense to build your own dedicated controller system. If you feel the need, here is a design that could start your grey matter working.

It uses an R6502 processor and an R6532 RIOT (RAM, I/O and Timer) chip, along with a low-cost 2716 EPROM, a color TV crystal and a few other parts.

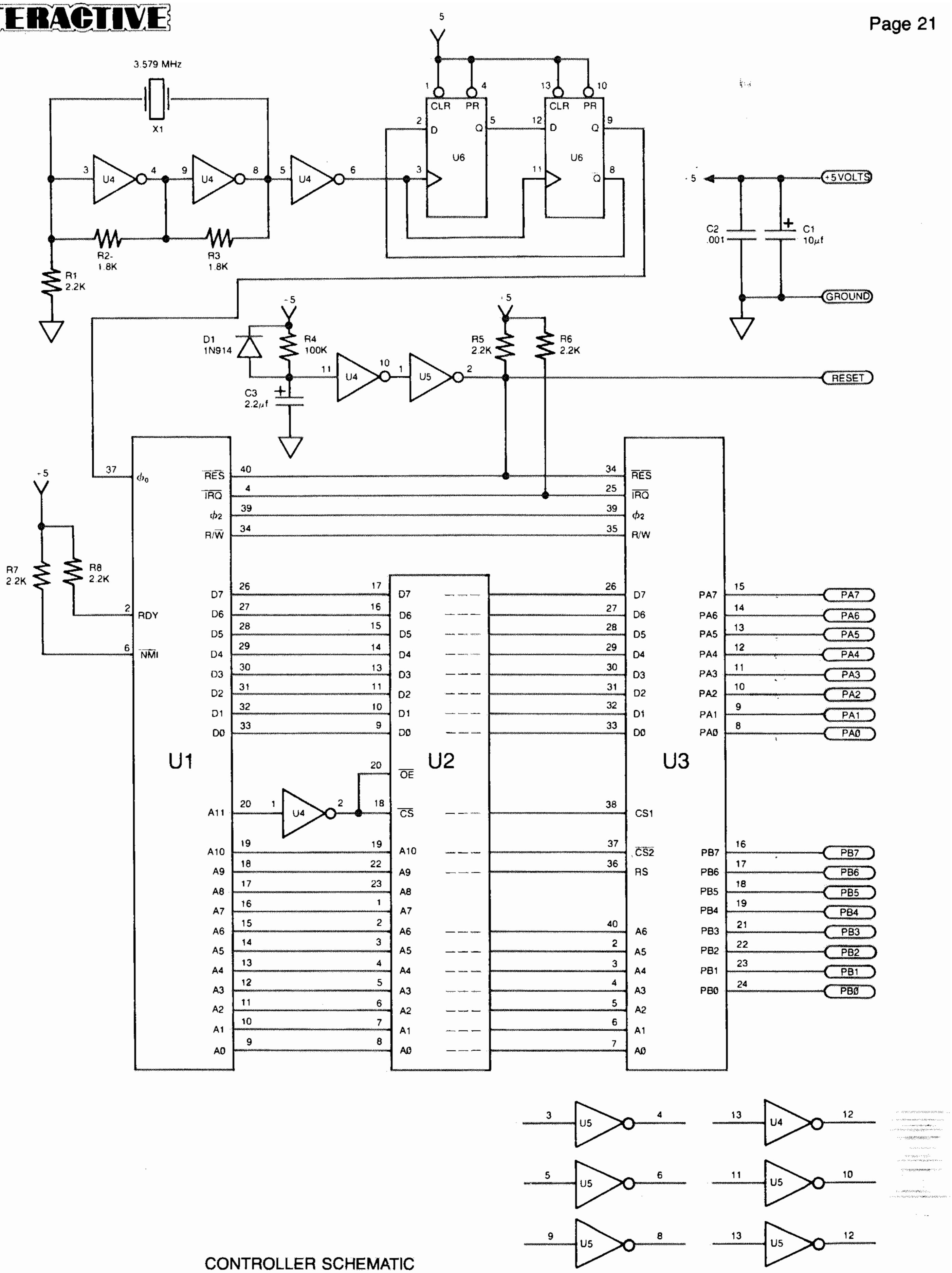
There are even a few spare inverter gates that can be used for I/O interfacing functions. The clock and divider circuit is from one of our application notes (Low-Cost Crystal Oscillator for Clock Input. Document #208) The 7474 is used to divide the 3.579 Mhz clock by four, which produces a system clock frequency of about 900 Khz. A very simple Power-On-Reset circuit, consisting of D1, C3, R4 and two inverter gates is used. (This circuit has worked quite well in other systems.)

Here is a system memory map:



And a parts list:

PART	PART NUMBER	POWER CONNECTIONS		
		+5	GROUND	# of pins
U1	R6502	8	1,21	40
U2	2716	24	12	24
U3	R6532	20	1	40
U4	74LS04	14	7	14
U5	7407	14	7	14
U6	7474	14	7	14



CONTROLLER SCHEMATIC

SPARE GATES

(Continued from page 2)

```
2000              ;TRACE PROGRAM
2000              ;
2000              ;EQUATES
2000              ;
2000              SOUT   =$CB08
2000              OUT    =$E9BC
2000              NUMA   =$EA46
2000              CRLOW  =$EA13
2000              BLANK  =$E83E
2000              PHXY   =$EB9E
2000              PLXY   =$EBAC
2000              ;
2000              ;ZERO PAGE
2000              ;
2000              TXT    =$00C6
2000              OTXT   =$0085
2000              CURLIN =$0081
2000                     *=$00E0
00E0
00E0              FLG    *=*+1
00E1              LTXT   *=*+2
00E3              POS    *=*+1
00E4              SAVX   *=*+1
00E5  0F 27       BUF    .WORD 9999,999,99,9
00E7  E7 03
00E9  63 00
00EB  09 00
00ED              ;
00ED              ;BASIC TRAP
00ED              ;
00ED                     *=$00C8
00C8  4C 9C 0F           JMP TRACE
00CB  EA                 NOP
00CC              BASC   =*
00CC                     *=$0F9C
0F9C  20 9E EB    TRACE  JSR PHXY
0F9F  48                 PHA
0FA0              ;
0FA0              ;IF $F0=0 TRACE OFF
0FA0              ;IF $F0#0 TRACE ON
0FA0              ;
0FA0  A5 E0              LDA FLG
0FA2  F0 40              BEQ SAMLIN
0FA4              ;
0FA4              ;DIRECT CMMD?
0FA4              ;YES==>SAMLIN
0FA4              ;
0FA4  A6 82              LDX CURLIN+1
0FA6  E8                 INX
0FA7  F0 3B              BEQ SAMLIN
0FA9              ;
0FA9              ;COMPARE OLD
0FA9              ;TO LAST
```

```
0FA9              ;
0FA9  A5 81              LDA CURLIN
0FAB  C5 E1              CMP LTXT
0FAD  D0 06              BNE NEWLIN
0FAF  A5 82              LDA CURLIN+1
0FB1  C5 E2              CMP LTXT+1
0FB3  F0 2F              BEQ SAMLIN
0FB5              ;
0FB5              ;UPDATE LAST TEXT
0FB5              ;
0FB5  A5 81       NEWLIN LDA CURLIN
0FB7  85 E1              STA LTXT
0FB9  A5 82              LDA CURLIN+1
0FBB  85 E2              STA LTXT+1
0FBD              ;
0FBD              ;P/O CURLIN
0FBD              ; RIGHT JUSTIFY
0FBD              ;EACH COLUMN
0FBD              ;
0FBD  A2 06              LDX #6
0FBF  20 F0 0F    P01    JSR RJ
0FC2  A6 E4              LDX SAVX
0FC4  CA                 DEX
0FC5  CA                 DEX
0FC6  10 F7              BPL P01
0FC8  20 08 CB           JSR SOUT
0FCB  E6 E3              INC POS
0FCD              ;
0FCD              ;FORMAT FOR A PRINT
0FCD              ;OR INPUT TOKEN
0FCD              ;
0FCD  68                 PLA
0FCE  48                 PHA
0FCF  C9 97              CMP #$97
0FD1  F0 0A              BEQ PRNT
0FD3  C9 84              CMP #$84
0FD5  F0 06              BEQ PRNT
0FD7              ;
0FD7              ;3 LINES /CR
0FD7              ;CK HEAD POSITION
0FD7              ;
0FD7  A5 E3              LDA POS
0FD9  C9 03              CMP #$3
0FDB  90 07              BCC SAMLIN
0FDD  A9 00       PRNT   LDA #0
0FDF  85 E3              STA POS
0FE1  20 13 EA           JSR CRLOW
0FE4  68          SAMLIN PLA
0FE5  20 AC EB           JSR PLXY
0FE8  C9 3A              CMP #$3A
0FEA  90 01              BCC SAM1
0FEC  60                 RTS
0FED  4C CC 00    SAM1   JMP BASC
```

(Continued on next page)

```
0FF0                  ;
0FF0                  ;RIGHT JUSTIFY RTN
0FF0                  ;
0FF0  A5 81           RJ      LDA CURLIN
0FF2  86 E4                   STX SAVX
0FF4  D5 E5                   CMP BUF,X
0FF6  A5 82                   LDA CURLIN+1
0FF8  F5 E6                   SBC BUF+1,X
0FFA  B0 03                   BCS RJ1
0FFC  4C 3E E8                JMP BLANK
0FFF  60              RJ1     RTS
1000                          .END
```

COMING UP!

Have received several good articles on the use of AIM 65 in Computer Aided Design (CAD) applications. Look for a handy Fourier Series program in the next issue. Forth seems to be getting quite popular according to the feedback I'm getting. I'm going all out to get a number of Forth "goodies" for issue #7. Some good information on this new and exciting computer language in the next issue. Is your system idle during the lunch hour. What a shame, especially when you could be playing a mini-adventure game (assuming you have BASIC w/4K of RAM). Watch for it in the next issue!

NEWSLETTER EDITOR
ROCKWELL INTERNATIONAL
P.O. Box 3669, RC55
Anaheim, CA 92803 U.S.A.