6502 USER NOTES

NO. 15 $2.50

EDITORIAL

BACK ISSUES TO BE AVAILABLE

We're now in the process of reprinting issues 1-6 of the 'NOTES' All pertinent information from Volume 1 (including the complimentary issue) will be combined into one giant issue and organized according to subject matter. You'll be able to find information alot faster and It'l be easier to read!

Am not sure of the price yet. That info will be available in the next issue. Volume 2 (issues 7-12) will be next.

6502 SYSTEM CENTERFOLD?

Well, not quite, but I would like to start featuring one 6502 system in each issue. If you'd like to have your system featured in the 'NOTES', send in a black & white glossy photograph of your machine (you can be in the picture also) and a few paragraphs describing what you have, what kind of software you use , and what you do with your machine. The picture should be well focused etc.

I always enjoy hearing about your system and I'm sure other readers would also. Let's hear from you!!!

SUBMITTING ARTICLES

Since all articles will be retyped, they need only be readable. Typing it would, of course, guarantee readability. Program listings, on the other hand, may not be reyped so, if at all possible, use white paper and a fresh ribbon on your printer. If there's no way you can generate an original source listing, then a handwritten source listing with MOS mnemonics, and labels of up to six characters, (don't forget to use labels when referencing zero page locations) will be satisfactory. Comments should be preceded by a semicolon.

This will make it easy for me to assemble your program for publication. Disassembler output is not very satisfactory except when heavily commented, labeled and all zero page registers identified by name.

Perhaps the best way to submit program source listings would be to send a cassette of the assembler source file and I can then assemble it and run a listing on my Decwriter. I can assemble source files from either the Micro-ade assembler (Peter Jennings) or the MOS/ARESCO/HDE assemblers. If you send a S.A.S.E., I'll return your cassettes. It would be wise to dump two copies of the file to cassette just in case.

I can read most of the Hypertape-recorded cassettes I receive once I adjust the azimuth of the cassette head for the higher audio level while reading the program. I think this head adjustment problem has probably accounted for most of the tape interchange problems I've been aware of. The machines I use to make the newsletter cassettes have been adjusted as close as possible and 30 seconds of synch characters precede the program for setting up your equipment. So far, we have not had any cassettes returned, so we must be doing something right.

BUGS IN ISSUE #14

INSIDE COVER: The correct price for 1-6 or 7-12 from Mark Kantrowitz is $7.00 add $3.00 for airmail overseas.

PAGE 4: (top of the page) the rest of the BANNNER listing should run from $2600-$28FF and not $3400-$36FF.

PAGE 15 : the short program in the NEW COMMAND FOR BASIC should read:

```
1000 PRINT: PRINT "Enter space when ready
";: GET A$: IF A$<>" " THEN 1000.
```

FROM SYNERTEK

Apparently, SYNERTEK has rewritten the SYM monitor to clean up the problems they had reading cassettes and a few other minor glitches. No word yet on the price for retrofit but I should have that info by next issue.

The following notice is being reprinted from Vol IV No 5 of the CACHE REGISTER. I don't know how true it is - but it pays to look before you leap.

WORLD POWER SYSTEMS: FRAUD!!!

It appears that World Power Systems is a carefully instigated fraud for ripping off computer hobbyists and small businesses. One Chicago business is out of luck for $4500. The scenario reads like a TV police show, complete with prison escape, aliases, etc.

I guess it's appropriate to repeat the advice that so many have given before: if you don't know the integrety of the company, do all your business in person or via C.O.D. Better yet, deal with local, reputable dealers, like the computer stores; or those who advertise in the CACHE Register, like Lloyd Smith of Smith Computer Systems. (With his good prices, it's a shame he hasn't been getting more business from his ads, and has had to cut them back.)

Ward Christensen

WE'RE GOING TO SUPPORT OSI!

From all indications, there are alot of frustrated OSI users out in the field.

I've looked over the C-1P and C2-4P and they seem like reasonable machines for the money. The 'USER NOTES' will try to fill in where the documentation leaves off so we really have our work cut out for us.

Pass the word along to any OSI users you know of.

We've already got a few goodies to pass along. For the first installment, see the comments section in this issue. KIM, of course, will still get the bulk of our support

CHEAP RAM!!

Joe was kind enough to lend me one of his dynamic RAM cards for a firsthand opportunity to see how well it worked. I cycled the board for several hours with a couple of the dynamic memory tests contained in the HDE Comprehensive Memory Test (CMT) package. The board performed flawlessly!

As far as I can tell, this RAM card should be useable with other 6502 machines uncluding OSI, and PET besides the KIM, SYM and AIM.

ERIC

A 32K DYNAMIC RAM BOARD FOR THE KIM-4 BUS

by J. C. Williams
55 Holcomb St.
Simsbury, Ct 06070

Two years ago, 16K x 1 dynamic memory chips such as Mostek's 4116 sold for about $40 each; they're now less than $10 each and available from many semiconductor manufacturers. These prices mean that a 32K board can be built for about $200. In addition, the board will draw less than 200 mA from the +8 Volt power supply, 200 mA from the +15 supply and 5 mA from the -15 supply. Memories for the APPLE II and TRS-80 microcomputers are based on these devices, as are many mimicomputer memories; in spite of old rumors, dynamic memories work reliably.

The circuit of figures 1A and 1B is a 32K byte (16K if only 8 memory chips are installed) memory for the KIM-4 bus which easily fits on a 4½"x6½" circuit board. Figure 2 shows the layout used for one of the prototypes built on a Vector 3662 plugboard. In eight months of constant use with a KIM 1 and KIM-4, no problems of any kind have been encountered with this unit. A second unit, built at the end of 1978, also works well.

It would take a long write-up to explain how dynamic memories work and this note is about a specific circuit. Readers who want to learn more details could start with Lane Hauck's article in the July, 1978, issue of BYTE and progress to manufactuerers' data sheets and application notes. Mostek's 1978 Memory Data Book and Designer's Guide is especially useful and has excellent applications information.

In the circuit of figure 1, memory refreshes are "hidden" during Ø1 ($\overline{Ø2}$) of the 65XX processor cycle. This can be done because although the processor puts out address and R/W information during Ø1, read or write operations are done during Ø2. The circuit described "gives" the memory to the processor during Ø2 and to the refresh circuit during $\overline{Ø2}$. Memory chips used in this way must be fast enough to function at approximately twice the processor clock frequency. Devices with a 200 ns access time and a 450 ns cycle time are required for this circuit if the processor has a 1 MHz clock.

Figure 3 is a timing diagram which shows what must be done to interface the 4116's (or pin-compatable equivilent) to a 1MHz 65XX bus. The bus provides Ø2, R/W, $\overline{\text{RESET}}$, and address information. During write cycles it provides data and during read cycles it takes data. The specific bus times marked on figure 3 were taken from the MOS Technonology Hardware Manual. The 4116's require a Row Address Strobe ($\overline{\text{RAS}}$), a Column Address Strobe ($\overline{\text{CAS}}$), $\overline{\text{WRITE}}$, and multiplexed address information at specified times. Figure 3 times were selected for the most reliable operation using the full time available during Ø2. Four types of memory cycles can occur: 1) Read 2)Write 3)Refresh and 4)Null. Read or Write cycles occur during Ø2 if the processor has addressed a location on the board. Refresh cycles occur during $\overline{Ø2}$ once every 32 clock cycles or during every $\overline{Ø2}$ if $\overline{\text{RESET}}$ is low. During Null cycles no 4116 activity occurs.

The circuit of figure 1 implements the timing using one CMOS and eleven TTL integrated circuits. The two 16K X 8 banks of 4116's have address lines A0-A6 driven by multiplexers U8-U11. To eliminate undershoot on A0-A6, 1.5 k pull-up resistors are required. Nand gates U6 and U7 drive the 4116 $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WRITE}}$ lines. Nand gate U4 and twelve bit ripple counter U5 produce a $\overline{\text{REFRESH}}$ signal once every 32 clock cycles as well as provide the seven bit refresh address to be used. A $\overline{\text{REFRESH}}$ signal is also produced when $\overline{\text{RESET}}$ is low in order to insure proper start-up of some manufacturer's memory chips. Since Refresh cycles are dependent only on the existence of a 1MHz Ø2 signal on the KIM-4 bus, any hardware controlling the bus must provide such a signal. One-shot multivibrators U1 and U2 provide row and column address strobe timing signals when triggered by other signals. U12 generates the $\overline{\text{BOARD SELECT}}$, UPPER BLOCK SELECT, and ROW ADDRESS 5 and 6 signals by comparing the four most significant bits of the KIM-4 address bus with the settings of the "starting address" switches. Sections of U3 are used as buffers, delay elements and inverters.

The construction of this circuit is not difficult, but requires care, planning and some experience. Layout is important to minimize the length of lines carrying high speed signals and undesirable coupling between lines. A low impedance ground and power supply distribution are essential because of the high peak currents drawn by the memory chips during clocking. Don't skimp on bypass capacitors and use #20 or larger tinned copper buswire for grounding. The grounding and bypass layout of figure 2 works well. Wire-wrapped connections are best made with a Vector Electronics Co. Model 180 "slit and Wrap" tool which enables one to solder to the leads of resistors, capacitors and edge connector pads as well as make "daisy chained" wraps. Once this tool is used, you'll never want to measure, cut and strip regular wire again.

All parts which attach to the board should be on hand before any construction is started. The following sequence may be of help in buidling one of these boards:

1. Attach wire wrap IC sockets and voltage regulators to board with "five minute" epoxy glue. Heat sinks are not required.
2. Run the ground bus on the bottom of the board using #20 or larger tinned copper bus wire. Start at edge connector pads 1 and A and go around the outside of the board to pads 22 and Z. Stick in and solder bypass capacitors between the bus and the proper IC socket pins as you go to hold the bus wire in place. Complete the ground network with additional lengths of bus wire to the "inner" IC's and install the remaining bypass capacitors.
3. Install the resistors (mounted vertically in some cases) and remaining capacitors by sticking their leads through the board and soldering them to the appropriate pins. Cut any uncommitted leads to ¼ for later "slit and wrap" connection. --The remaining connections can be made with "slit and wrap" techniques- don't forget to solder after wrapping round leads.
4. Run the +5, +12 and -5 Volt power supply lines from the outputs of the respective regulators to the correct IC pins, bypass capacitor leads and pull up resistor (+5 Volt only) leads. Also run the +8, +16 and -16 Volt lines from the proper edge connector pads to the correct regulator input pins and bypass capacitors. These lines may be conveniently run on the board top. "Plug in" the board, power it up and check for correct power and ground connections at every IC location.
5. Wire the remainder of the circuit in stages checking between data sheet pinouts, schematics and drawings to eliminate errors. The stages could be a)address lines b)control and timing logic c)data lines and d)row and column address strobes and $\overline{\text{WRITE}}$ lines. It is helpful to use wire with a different color insulation for each stage.

Install all IC's except 4116's and test the board on a KIM-4 bus. Set the starting address of the board as desired (for example $2000) and turn on the power-the system should operate normally. Load test programs in operatonal memory which will "exercise" the board.

Read Test

```
0200 AD 00 20   READ LDA $2000
0203 4C 00 02        JMP READ
```

Write Test

```
0200 8D 00 20   WRITE STA $2000
0203 4C 00 02         JMP WRITE
```

While one of these programs runs, an oscilloscope can be used to check the $\overline{RAS}$, $\overline{CAS}$ and other signals produced at a memory chip socket in the selected block. Ø2 should be used as the 'scope trigger and displayed on one channel so that the signal being tested can be compared with it. Adjust timing if necessary by changing one-shot timing resistors. Signals to the other memory block can be checked by changing the address used in the test program (for example to $6000). $\overline{RAS}$ signals produced during $\overline{Ø2}$ by $\overline{REFRESH}$ may be observed by holding RESET low.

Install 4116's and test using the monitor and a program such as Memory Test by Jim Butterfield in the First Book of KIM. If trouble-shooting is needed, the type of problem is an indication of what's wrong. For example, if one bit in one block is always wrong suspect a bad 4116 or data line wiring. If the errors seem to be random, the 4116's may be too slow or there may be excessive noise on the power supply lines. Based on experience with the prototypes, once the clock timing has been adjusted, there will be no problems at all.

The author hopes that the availability of large, low-cost memories will stimulate the development of software for 65XX systems. Any correspondence on the memory circuit should be sent to the above address.

FIGURE 1A 32 K RAM BOARD FOR KIM-4 BUS

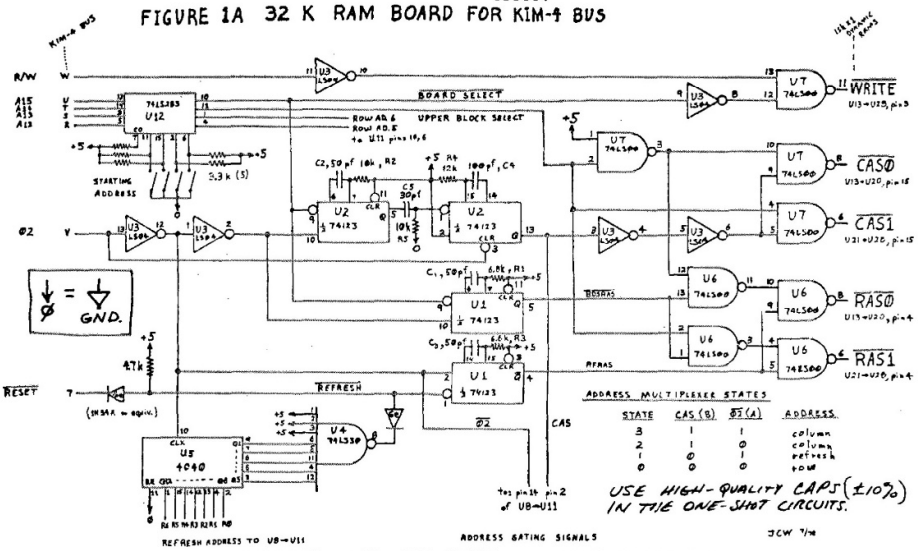


FIGURE 1B 32K RAM BOARD - ADDRESS MULTIPLEXER

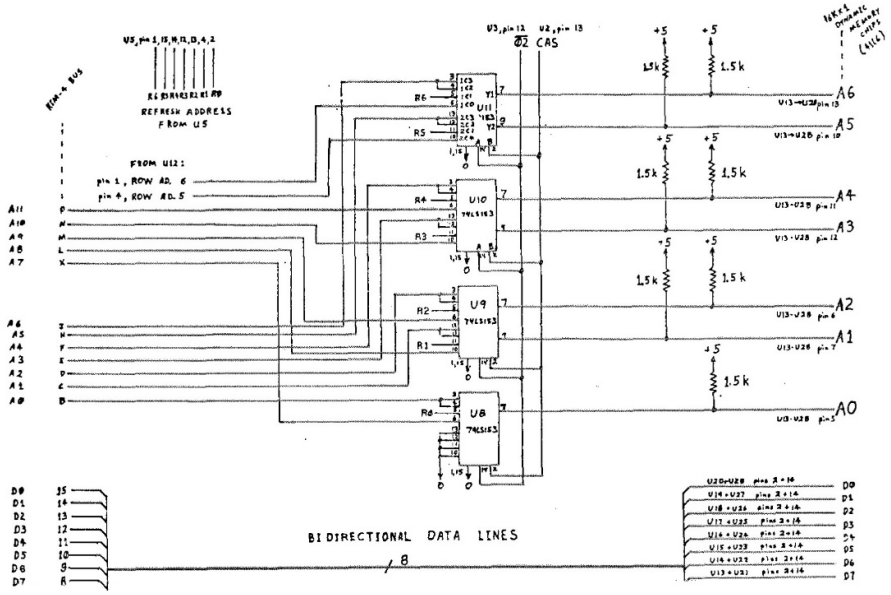


FIGURE 2-PARTS LAYOUT, GROUND BUS

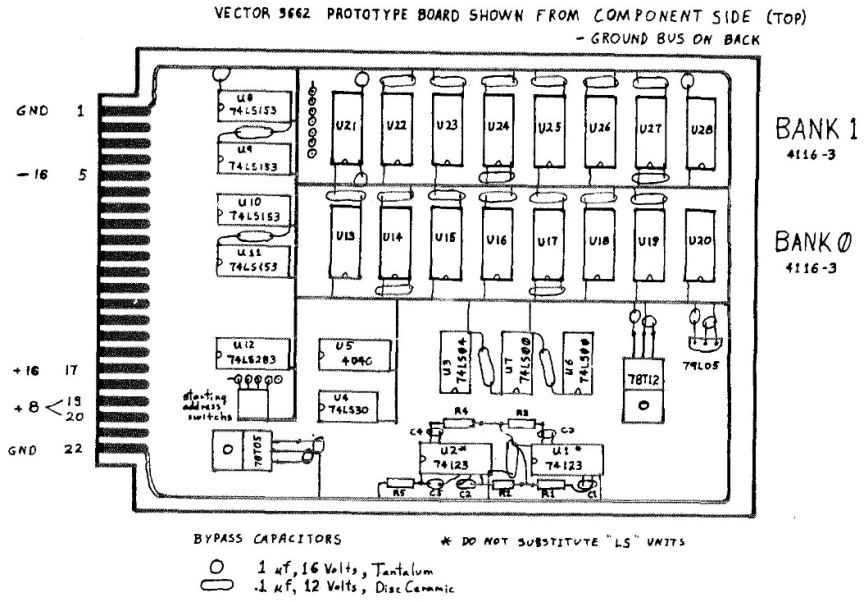
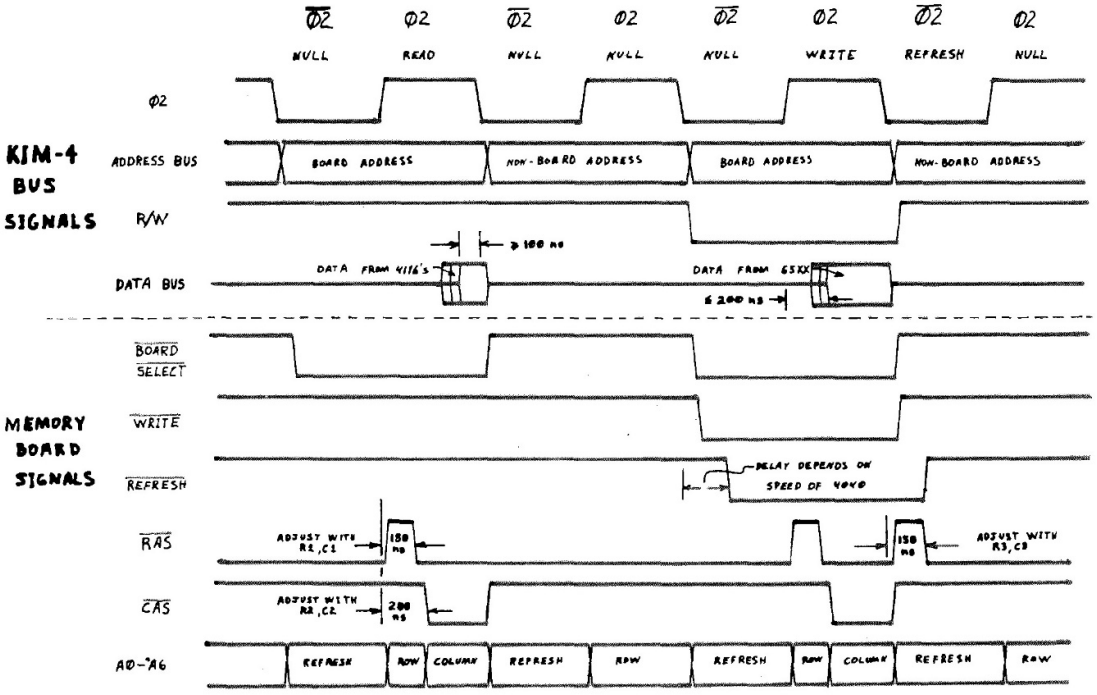


FIGURE 3 - TIMING DIAGRAMS - 1 MHz CLOCK

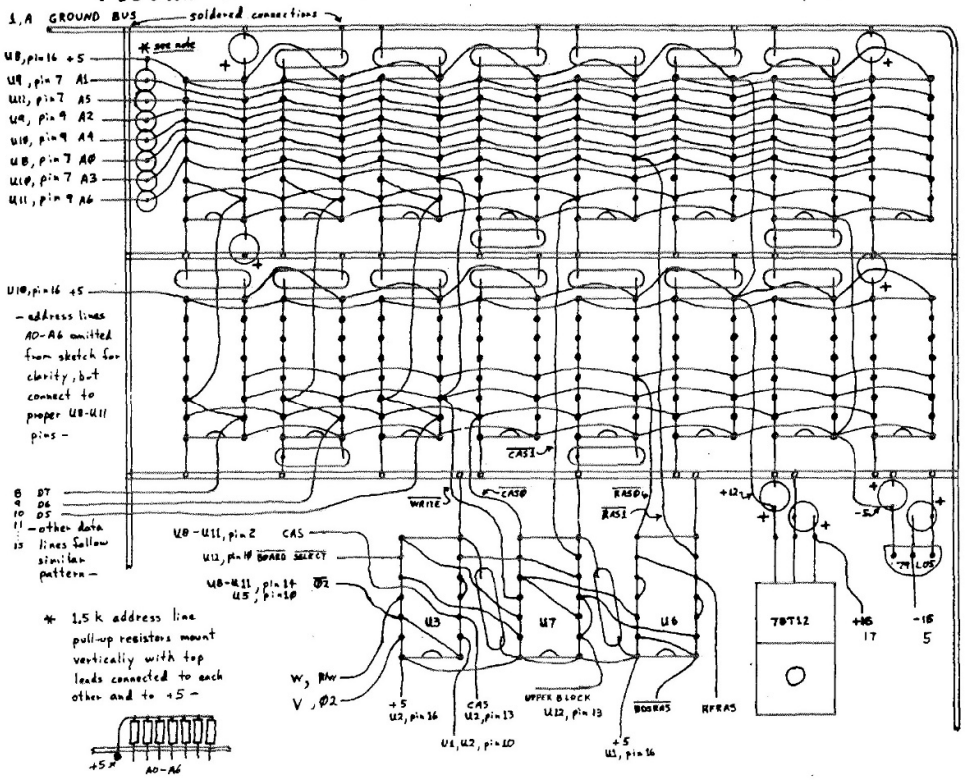
- ALL TIMES MEASURED RELATIVE TO KIM-4 Ø2 EDGES



Some Errors! See #16 - pg 25

FIGURE 4 - WIRING SKETCH - MEMORY MATRIX

top view



650X SAVE AND RESTOR ROUTINES

by Jim Green
807 Bridge Street
Bethlehem, PA 18018

These routines save and recover A, Y and X register values. The ability to protect these values is particularly useful when they might be lost due to modification within, say, a device routine. In the programs below no additional data memory (other than the stack) is required. This makes it possible to save register contents to any level of nesting permitted by the size of the available stack.

At the outset one should note that the two subroutines illustrated below, SAVE and RESTOR, can be replaced with just twenty bytes of code (see ASAVE and ARESTR below) which will execute in less than a sixth of the time.

Having said that, why would anyone want to know about, much less use, these routines? First, the exercise in writing or understanding the routines is interesting, I think. Second, and more important, a pair of subroutine calls is easier on the overburdened mond of the progrmmer than remembering the sequence of the ten lines of code. (Did I save the Y before the X or vice versa?!) A third possible reason, that of saved program space, would only exist if in excess of 6 call pairs (ie. a SAVE and a RESTOR) are made to these routines.

The alternative code sequences are:

```
ASAVE:  PHA             ;push A value to stack
        TYA
        PHA             ;push Y value to stack via A
        TXA
        PHA             ;push X value to stack via A
        TSX             ;use stack pointer
        LDAX    $0103   ; to get A copy
        PHA             ;save it also
        LDAX    $0101   ;retrieve X value
        TAX             ; and restore X
        PLA             ;now restore A
        ...
ARESTR: PLA             ;pull X value from stack to A
        TAX             ;restore X
        PLA             ;pull Y value from stack via A
        TAY             ;restore Y
        PLA             ;restore A
```

The interesting aspect of the subroutine code presented below is that subroutines are used to perform stack operations. Since the subroutines themselves use the stack as the place where their return addresses are saved, it is necessary to move some stack bytes around and to do this regardless of the current value of the stack pointer.

At the beginning of any subroutine, after it has been called, the state of the stack may be represented as shown in Figure 1:

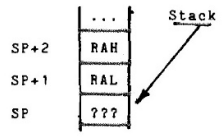


Figure 1.

Where RAH and RAL are the high and low bytes of the return address, "???" is the next available stack byte, and SP is the address pointed to by the stack pointer.

If the operations starting with ASAVE (above) are now invoked, the stack would appear as shown in Figure 2a. Notice that the return address is now hidden and no longer directly available. An attempt to return from the subroutine at this point would get lost by returning to the address equal to Y (as the high byte) and X (as the low). Clearly, some swaps must be made.

To accomplish this, the stack pointer is moved "up" two bytes (remember that stacks work upside down). Then RAH and RAL are copied onto the two new locations (Figure 2b.). The entire block of 5 bytes is then shifted "down" (Figure 2c.), and finally the stack pointer is re-established just "above" the return address. The return address is now accessable so that after an RTS (Figure 2d.) only the A, Y and X values remain on the stack.

The RESTOR routine does essentually the same thing in reverse. One additional wrinkle occurs at RESTI, in which the current value of A replaces the saved value of A on the stack before program control drops into the RESTOR routine. This feature is useful in single byte input routines where we wish to protect the Y and X values but to replace the old A value with the new input value.

SAVE and RESTOR (or RESTI) may be invoked anywhere in a program subject to the restriction that each SAVE call be ultimately followed by a RESTOR (or RESTI) call at the corresponding stack level. The partial code below illustrates the application of the routines. Notice that the pair of calls within SUBROUT are nested within the pair outside SUBROUT:

```
...
JSR     SAVE          SUBROUT JSR     SAVE
...                             ...
JSR     SUBROUT                 JSR     RESTOR
...                             RTS
JSR     RESTOR
...
```

As stated at the outset, these routines will save neither program time nor program space but they may, in the long run, save a programmer from undue wear and tear. Besides, they were fun to write.

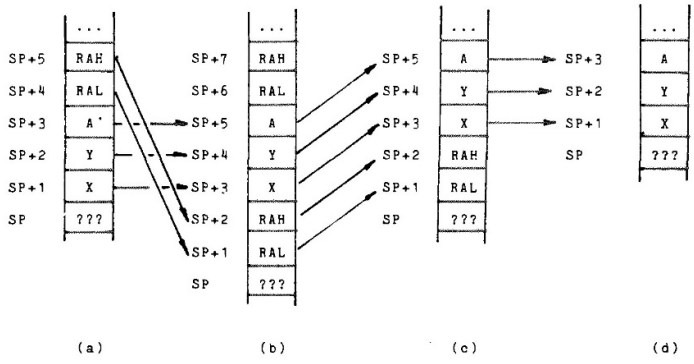


Figure 2. Stack values due to SAVE.

```
                ;       650X REGISTER SAVE AND RESTOR ROUTINES
                ;
                ;       VERSION 0.1, 14 FEB 79
                ;
                ;       COPYRIGHT, 1979
                ;       COMMERCIAL RIGHTS RESERVED BY
                ;
                ;       J. S. GREEN, COMPUTER SYSTEMS
                ;       807 BRIDGE STREET
                ;       BETHLEHEM, PA 18018
                ;       (215) 867-0924
                ;
                        .DEF  PGONE=$0100       ;START OF PAGE ONE
                ;
                        .LOC  $0200
                ;
                ;       SAVE A, Y, & X REGISTER VALUES ON STACK
                ;
0200 48         SAVE:   PHA             ;SAVE A
0201 98                 TYA
0202 48                 PHA             ;SAVE Y
0203 8A                 TXA
0204 48                 PHA             ;SAVE X
0205 48                 PHA             ;ADD TWO BYTES TO STACK
0206 48                 PHA
0207 BA                 TSX             ;USE STACK POINTER TO
0208 BD 07 01           LDAX  PGONE+7   ;  MOVE RETURN ADDRESS
020B 9D 02 01           STAX  PGONE+2   ;  TO TOP OF STACK
020E BD 06 01           LDAX  PGONE+6
0211 9D 01 01           STAX  PGONE+1
0214 A0 04              LDY#  4
```

```
0216 BD 05 01  SAVE1:  LDAX  PGONE+5     ;SHIFT LAST 5 BYTES OF
0219 9D 07 01          STAX  PGONE+7     ;  STACK DOWN TWO CELLS
021C CA                DEX               ;  TO COVER OLD ADDRESS
021D 88                DEY
021E 10 F6             BPL   SAVE1       ;LOOP TIL 5 DONE
0220 68                PLA               ;ADJUST POINTER
0221 68                PLA
0222 BD 0C 01          LDAX  PGONE+$0C   ;NOW RESTORE REGISTERS
0225 48                PHA               ;ACC
0226 BD 0B 01          LDAX  PGONE+$0B
0229 A8                TAY               ;Y REGISTER
022A BD 0A 01          LDAX  PGONE+$0A
022D AA                TAX               ;X REGISTER
022E 68                PLA               ;ACC
022F 60                RTS
                ;
                ;      RESTORE X & Y ONLY
                ;
0230 BA        RESTI:  TSX               ;USE STACK POINTER
0231 9D 05 01          STAX  PGONE+5     ;  TO OVER-RITE OLD A
                ;
                ;      RESTORE A, Y, & X
                ;
0234 48        RESTOR: PHA               ;ADD TO STACK
0235 48                PHA
0236 BA                TSX               ;USE STACK POINTER
0237 A0 04             LDY# 4
0239 BD 03 01  RESTR1: LDAX  PGONE+3     ;  TO SHIFT LAST 5 BYTES
023C 9D 01 01          STAX  PGONE+1     ;  OF STACK UP 2 CELLS
023F E8                INX               ;TO MAKE ROOM FOR RETURN
0240 88                DEY               ;  ADDRESS
0241 10 F6             BPL   RESTR1      ;BR TIL 5 DONE
0243 BA                TSX
0244 BD 02 01          LDAX  PGONE+2     ;MOVE RETURN ADDRESS
0247 9D 07 01          STAX  PGONE+7
024A BD 01 01          LDAX  PGONE+1
024D 9D 06 01          STAX  PGONE+6
0250 68                PLA               ;ADJUST STACK POINTER
0251 68                PLA
0252 68                PLA               ;X VALUE
0253 AA                TAX
0254 68                PLA               ;Y VALUE
0255 A8                TAY
0256 68                PLA               ;ACCUMULATOR VALUE
0257 60                RTS
                ;
                       .END
```

TELEPHONE DIALER

by Mark Kantrowitz
15 Midway Court
Rockaway NJ 07866

This telephone dialer program will dial a telephone number (of any length) by pressing a single key on KIM's keypad. The hardware consists of a 7406 inverter, NPN transistor, a couple of resistors and a 12 volt relay (see figure). The switching end of the relay is connected between the green wire and the logic box where the green wire was connected.

Up to 16 different telephone numbers can be dialed. You must first store the numbers in memory. Preceeding every telephone number, you must store an I.D. number. The first I.D. number is A0 and goes up to AF. Each telephone number consists of one or more bytes. Each byte of the telephone number consists of two digits of the telephone number. Except when there is an odd number of digits, in which case a "F" is placed in the last nybble of the last byte of that particular telephone number. A "FF" in the phone table indicates the end of the table.

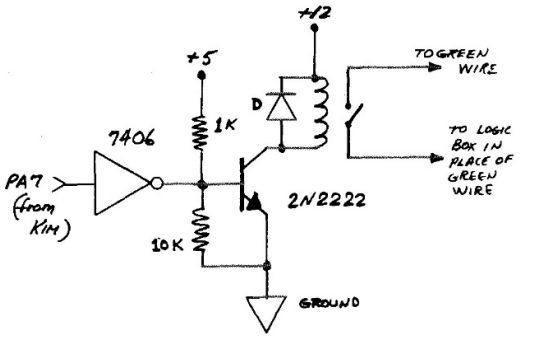
As an example, a typical phone table would look like this starting in location 0018:

A0 20 16 25 17 19 A1 35 91 81 9F A2 80 02 65 48 45 FF

This phone table has three numbers in it. Pressing 0 on the keypad will cause, 201-6251719 to be dialed.

When you start the program at 0200, the display will flash "PHONE". When you press a key on the keypad, the program changes that key to an I.D. number and searches for it in the phone table. If it is there it dials the number associated with it. If the I.D. number is not in the table, the display will flash "ERROR". As the program dials a number , it displays it in a banner fashion. After dialing, it returns to displaying "PHONE".

NOTE: The Telephone Co. takes a dim view of equipment attached to their lines without their approval.



D = 1N914

```
MDE ASSEMBLER REV 2.0

LINE#  ADDR   OBJECT     SOURCE        PAGE 0001

 0010  2000              ;TELEPHONE DIALER PROGRAM BY MARK KANTROWITZ
 0020  2000                    *=$0
 0030  0000              MSG   *=*+12
 0040  000C              DISPLY *=*+7
 0050  0013              FLASH  *=*+5
 0060  0018              TABLE  *=*+1            ;START OF NUMBER TABLE
 0070  0019
 0080  0019              PAD    =$1700
 0090  0019              PADD   =$1701
 0100  0019              TIMER  =$1707
 0110  0019
 0120  0019              CHANGE =$1F40
 0130  0019              GETKEY =$1F6A
 0140  0019
 0150  0019              ABLE   =$1FE7
 0160  0019              SBD    =$1742
 0170  0019              SAD    =$1740
 0180  0019              SADD   =$1741
 0190  0019
 0200  0019
 0210  0019                    *=$0200
 0220  0200  A2 00        INITS  LDX #0          ;INITS MESSG PNTR
 0230  0202  BD 5E 03     INITS1 LDA MESSG,X     ;GET A MESSAGE BYTE AND
 0240  0205  95 00              STA MSG,X        ;STORE IN Z-PAGE LOCATIONS
 0250  0207  E8                 INX
 0260  0208  E0 0C              CPX #$0C         ;DONE YET?
 0270  020A  D0 F6              BNE INITS1
 0280  020C  D8           START  CLD
 0290  020D  A2 00              LDX #$0          ;INITS INDEX FOR DISPLAYING
 0300  020F  86 13              STX FLASH        ;"PHONE" MESSAGE.
 0310  0211  A9 FF        SET    LDA #$FF        ;SET TIMER TO .25 SEC
 0320  0213  8D 07 17           STA TIMER
 0330  0216  A6 13        FIVE   LDX FLASH
 0340  0218  86 14              STX FLASH+1
 0350  021A  A0 09              LDY #$9          ;SELECT FIRST DIGIT
 0360  021C  A9 7F        FOUR   LDA #$7F         ;SET DIRECTIONAL REGISTER
 0370  021E  8D 41 17           STA SADD
 0380  0221  84 17              STY FLASH+4
 0390  0223  A6 14              LDX FLASH+1
 0400  0225  8C 42 17           STY SBD
 0410  0228  B5 00              LDA MSG,X        ;LOAD SEGMENT CONTROL BYTE
 0420  022A  8D 40 17           STA SAD
 0430  022D  E6 14              INC FLASH+1      ;INCREMENT FOR NEXT DIGIT
 0440  022F  A9 10              LDA #$10
 0450  0231  85 15              STA FLASH+2
 0460  0233  85 16        TWO   STA FLASH+3      ;DELAY FOR A FEW
 0470  0235  C6 16        ONE   DEC FLASH+3      ;MILLISECONDS
 0480  0237  D0 FC              BNE ONE
 0490  0239  C6 15              DEC FLASH+2
 0500  023B  D0 F6              BNE TWO
 0510  023D  20 40 1F           JSR CHANGE
 0520  0240  20 6A 1F           JSR GETKEY       ;GET A KEY
 0530  0243  C9 10              CMP #$10         ;VALID KEY? (0-F)
 0540  0245  30 26              BMI THREE        ;IF YES, FIND TELEPHONE NUMBER
 0550  0247
 0560  0247  18                 CLC
 0570  0248  E6 17              INC FLASH+4      ;INCREMENT TO SELECT
 0580  024A  E6 17              INC FLASH+4      ;NEXT DIGIT
 0590  024C  A4 17              LDY FLASH+4
 0600  024E  C0 15              CPY #$15         ;PAST 6TH DIGIT?
 0610  0250  D0 CA              BNE FOUR         ;IF NOT, LIGHT NEXT DIGIT
 0620  0252  2C 07 17           BIT TIMER        ;.25 SECONDS UP?
 0630  0255  10 BF              BPL FIVE         ;IF NOT, LIGHT DISPLAY AGAIN
 0640  0257
 0650  0257  A9 FF              LDA #$FF         ;SET TIMER FOR ANOTHER .25 SEC
 0660  0259  8D 07 17           STA TIMER        ;FOR DISPLAY BLANKING
 0670  025C  20 40 1F     SIX   JSR CHANGE
 0680  025F  20 6A 1F           JSR GETKEY       ;GET A KEY
 0690  0262  C9 10              CMP #$10         ;VALID KEY? (0-F)?
 0700  0264  30 07              BMI THREE        ;IF YES, FIND TELEPHONE NUMBER
 0710  0266  2C 07 17           BIT TIMER        ;.25 SECONDS PASS?
 0720  0269  10 F1              BPL SIX          ;IF NOT, GET A KEY
 0730  026B  30 A4              BMI SET          ;IF SO, FLASH DISPLAY AGAIN
 0740  026D
 0750  026D              ;FIND TELELPHONE NUMBER
 0760  026D  18           THREE  CLC              ;'A' CONTAINS I.D.
 0770  026E  69 A0              ADC #$A0         ;MAKE INPUT LOOK LIKE TABLE I.D.
 0780  0270  A2 00              LDX #$00
 0790  0272  D5 18        TEN   CMP TABLE,X      ;I.D. MATCH?
 0800  0274  EA                 NOP              ;FOR TABLE RELOCATION
 0810  0275  F0 12              BEQ EIGHT        ;IF SO, DIAL NUMBER
 0820  0277  E8           NINE  INX              ;IF NOT, PASS OVER NUMBER
 0830  0278  B4 18              LDY TABLE,X
 0840  027A  EA                 NOP              ;FOR TABLE RELOCATION
 0850  027B  C0 A0              CPY #$A0
 0860  027D  90 F8              BCC NINE
```

```
0870  027F  C0 FF           CPY ##FF        #END OF TABLE?
0880  0281  D0 EF           BNE TEN         #IF NOT, COMPARE WITH I.D.
0890  0283  A9 06           LDA ##6         #IF SO, FLASH 'ERROR' MESSAGE
0900  0285  85 13           STA FLASH
0910  0287  D0 88           BNE SET
0920  0289
0930  0289                 #DIAL NUMBER
0940  0289  A0 05     EIGHT  LDY ##5
0950  028B  A9 00           LDA ##0         #ZERO OUT DISPLAY AREA
0960  028D  99 0C 00  ELEVEN STA DISPLY,Y
0970  0290  88               DEY
0980  0291  10 FA           BPL ELEVEN
0990  0293  E8       NEXT   INX
1000  0294  B5 18           LDA TABLE,X     #LOAD TWO NUMBERS
1010  0296  EA               NOP             #FOR TABLE RELOCATION
1020  0297  C9 A0           CMP ##A0        #NUMBER COMPLETED?
1030  0299  B0 26           BCS TWELVE      #IF SO, GO TO START
1040  029B  4A               LSR A           #ISOLATE LEFT DIGIT
1050  029C  4A               LSR A           #BY SHIFTING IT RIGHT
1060  029D  4A               LSR A
1070  029E  4A               LSR A
1080  029F  29 0F           AND ##0F        #MASK LEFT NIBBLE
1090  02A1  85 12           STA DISPLY+6    #STORE FOR DISPLAY
1100  02A3  C9 00           CMP ##0         #IF ZERO, MAKE IT #0A
1110  02A5  D0 02           BNE SKIP
1120  02A7  A9 0A           LDA ##0A
1130  02A9  85 14     SKIP  STA FLASH+1     #STORE FOR DIALING
1140  02AB  20 1B 03        JSR MOVE        #SHIFT DISPLAY LEFT
1150  02AE  20 D9 02  LAP   JSR PULSE       #LIGHT DISPLAY AND PULSE PHONE
1160  02B1  D0 FB           BNE LAP         #FINISH PULSING DIGIT?
1170  02B3
1180  02B3                 #IF NOT, CONTINUE PULSING.....
1190  02B3  20 05 03        JSR DELAY       #DELAY FOR .5 SEC
1200  02B6  B5 18           LDA TABLE,X     #LOAD RIGHT DIGIT
1210  02B8  EA               NOP             #FOR TABLE RELOCATION
1220  02B9  29 0F           AND ##0F        #MASK LEFT NIBBLE
1230  02BB  85 12           STA DISPLY+6    #STORE FOR DISPLAY
1240  02BD  C9 0F           CMP ##0F        #END OF NUMBER?
1250  02BF  D0 03           BNE ZERO        #IF NOT, CONTINUE
1260  02C1  4C 0C 02  TWELVE JMP START      #IF SO, GO TO BEGINNING
1270  02C4
1280  02C4  C9 00     ZERO  CMP ##0         #NUMBER ZERO?
1290  02C6  D0 02           BNE PASS        #IF SO, MAKE IT #0A
1300  02C8  A9 0A           LDA ##0A
1310  02CA  85 14     PASS  STA FLASH+1     #STORE FOR DIALING
1320  02CC  20 1B 03        JSR MOVE        #SHIFT DISPLAY OVER ONE
1330  02CF  20 D9 02  LITE  JSR PULSE       #LIGHT DISPLAY AND PULSE PHONE
1340  02D2  D0 FB           BNE LITE        #IF NOT DONE PULSING, CONTINUE
1350  02D4  20 05 03        JSR DELAY       #DELAY FOR .5 SEC
1360  02D7  30 BA           BMI NEXT        #GET NEXT DIGIT
1370  02D9
1380  02D9                 #SUBROUTINE TO PULSE PHONE...
1390  02D9
1400  02D9  A9 80     PULSE  LDA ##80
1410  02DB  8D 01 17        STA PADD        #SET PA7 TO OUTPUT
1420  02DE  A9 00           LDA ##00        #TURN ON A7
1430  02E0  8D 00 17        STA PAD
1440  02E3  A9 31           LDA ##31        #SET TIMER FOR .1 SEC
1450  02E5  8D 07 17        STA TIMER
1460  02E8
1470  02E8  20 30 03  PLAY  JSR ETIL        #LIGHT DISPLAY
1480  02EB  2C 07 17        BIT TIMER       #TIMER UP?
1490  02EE  10 F8           BPL PLAY        #IF NOT, GO LIGHT DISPLAY
1500  02F0  A9 80           LDA ##80        #TURN PA7 OFF
1510  02F2  8D 00 17        STA PAD
1520  02F5  A9 31           LDA ##31        #SET TIMER FOR .1 SEC
1530  02F7  8D 07 17        STA TIMER
1540  02FA  20 30 03  UP    JSR ETIL        #LIGHT DISPLAY
1550  02FD  2C 07 17        BIT TIMER       #TIMER UP?
1560  0300
1570  0300  10 F8           BPL UP          #IF NOT, GO LIGHT DISPLAY
1580  0302  C6 14           DEC FLASH+1     #DECREMENT DIGIT
1590  0304  60               RTS

1600  0305
1610  0305                 #SUBROUTINE TO DELAY .5 SECONDS
1620  0305  A0 01     DELAY  LDY ##1
1630  0307  84 13           STY FLASH
1640  0309  A9 FF     TIME   LDA ##FF
1650  030B  8D 07 17        STA TIMER       #SET TIMER FOR .25 SEC
1660  030E  20 30 03  YALP   JSR ETIL        #LIGHT DISPLAY
1670  0311  2C 07 17        BIT TIMER       #TIME UP?
1680  0314  10 F8           BPL YALP        #IF NOT, LIGHT DISPLAY
1690  0316  C6 13           DEC FLASH       #SET TIMER FOR .25 SEC DELAY
1700  0318  10 EF           BPL TIME
1710  031A  60               RTS
1720  031B
```

```
1730  031B                  ;SUBROUTINE TO MOVE DISPLAY
1740  031B  A0 00           MOVE   LDY #$0
1750  031D  B9 0D 00        CONT   LDA DISPLY+1,Y   ;LOAD DIGIT
1760  0320  99 0C 00               STA DISPLY,Y     ;STORE IN PLACE TO LEFT
1770  0323  C8                     INY
1780  0324  C0 05                  CPY #$5          ;FINISHED?
1790  0326  D0 F5                  BNE CONT         ;IF NOT, CONTINUE
1800  0328  A4 12                  LDY DISPLY+6     ;LOAD INCOMING DIGIT
1810  032A  B9 E7 1F               LDA ABLE,Y       ;GET BIT REPRESENTATION
1820  032D  85 11                  STA DISPLY+5     ;STORE IN 6TH DSPLY POSITION
1830  032F  60                     RTS
1840  0330
1850  0330                  ;SUBROUTINE TO LIGHT DISPLAY
1860  0330  A0 09           ETIL   LDY #$9
1870  0332  84 16                  STY FLASH+3      ;SET DIRECTIONAL REGS
1880  0334  A9 7F                  LDA #$7F
1890  0336  8D 41 17               STA SADD
1900  0339  A0 00                  LDY #$0
1910  033B  A5 16           INUE   LDA FLASH+3      ;SELECT DIGIT
1920  033D  8D 42 17               STA SBD
1930  0340  B9 0C 00               LDA DISPLY,Y     ;LOAD CONTROL BYTE
1940  0343  8D 40 17               STA SAD
1950  0346  A9 10                  LDA #$10         ;DELAY FOR .5 MILLISECONDS
1960  0348  85 15                  STA FLASH+2
1970  034A  85 17           ATS    STA FLASH+4
1980  034C  C6 17           CED    DEC FLASH+4
1990  034E  D0 FC                  BNE CED
2000  0350  C6 15                  DEC FLASH+2
2010  0352  D0 F6                  BNE ATS
2020  0354  E6 16                  INC FLASH+3      ;GET NEXT DIGIT SELECT
2030  0356  E6 16                  INC FLASH+3
2040  0358  C8                     INY
2050  0359  C0 07                  CPY #$7          ;DONE?
2060  035B  D0 DE                  BNE INUE         ;IF NOT, CONTINUE
2070  035D  60                     RTS
2080  035E
2090  035E  F3              MESSG  .BYTE $F3,$F6,$BF,$D4,$F9,$00  ;"PHONE" MESSAGE
2090  035F  F6
2090  0360  BF
2090  0361  D4
2090  0362  F9
2090  0363  00
2100  0364  79                     .BYTE $79,$50,$50,$5C,$50,$00  ;"ERROR" MESSAGE
2100  0365  50
2100  0366  50
2100  0367  5C
2100  0368  50
2100  0369  00
2110  036A                        .END

ERRORS = 0000
```

LANGUAGE LAB

basic

SOME IMPORTANT BASIC MODS from Christopher Flynn, 2601 Claxton Dr. Herndon, VA 22070

Enclosed are listings of two machine language programs which should be of interest to users of Johnson Computer's Microsoft BASIC. The first subroutine MLDSPT is a dispatch which BASIC can use to activate user-written machine language subroutines. The second subroutine ARRSAV/ARRLOD provides an easy way to save and load data on cassette tape from BASIC arrays (either floating point or integer).

Before describing the subroutines, I would like to mention the features of my eclectic system-perhaps I can share experiences with someone. First of all, I am using a KIMSI motherboard that is populated with 16K of RAM and an Ithaca Audio EPROM board. For a console device, I use an SSM VDB-1B board. My software TTY emulator is home-brew, approximately 500 bytes long, and completely romable and position independent. Hypertape is great, but for tape I/O, a Tarbell board has really proved its worth. Finally, in the hardware area, hard copy is produced by a faithful SWTPC PR-40 printer. So much for my system...

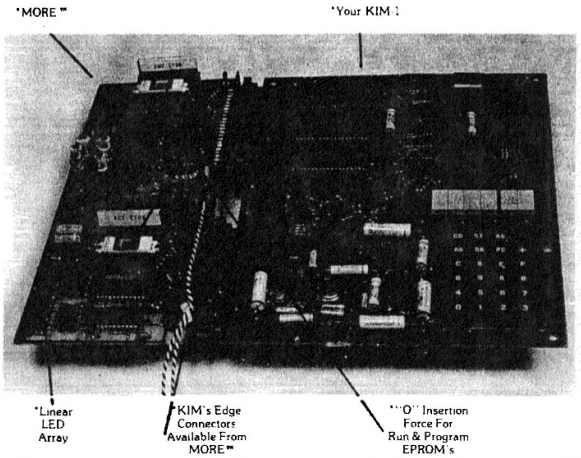
MLDSPT

The USR function in BASIC is used to invoke machine language subroutines. One of the drawbacks of Microsoft's USR is that there is no way to directly specify the address of the user subroutine. Instead, the address of the subroutine must be POKEd into BASIC.

Not having access to the source code for BASIC, I could not attack the problem head-on. MLDSPT is an alternate way of calling machine language subroutines and works as follows.

MLDSPT itself is invoked by the USR function. (This implies that USRLOC must be patched with the address of MLDSPT.) MLDSPT queries locations $FE or 254 into which the programmer has POKEd a subroutine number. MLDSPT then activates the proper subroutine and the subroutine can access the argument of the USR function if desired.

Subroutine numbers can be in the range of 0 - 127. MLDSPT multiplies the subroutine number by two and uses the result to index a table of subroutine addresses. The proper address is fetched



T. T. I.

TENNESSEE TECHNICAL INTRODUCES
THE NEW
MORE™ BOARD
for KIM series Micro-computers

LOADED WITH FEATURES
More EPROM---up to 2K
More RAM---up to 3K
More Output---16 latched bits
More Use---run, copy, program to or from any eprom type mix.

*Program, copy, verify run any of the industry standard EPROM's (2708, 2758, either 2716). Copies to or from any mix of EPROM types!

*Sockets for 3K of 2114 type RAM with decoding to give KIM 4K continuous RAM. Decoding may also be set to 1K boundries within an 8K block to anywhere in memory.

*Zero insertion force sockets for both run and program EPROM sockets.

*No extra power supplies needed--runs from KIM's +5 and +12 supply. Regulated -5V and programming voltages generated on board.

*Documentation and software listings furnished.

*Optional 2708 EPROM containing program and verify routines available. Optional KIM format tape available.

*MORE™ does not monopolize the KIM edge connectors--they pass through so you simply unplug your existing connector, plug on the MORE board, and reconnect your connectors.

*Special LED lets you see what the KIM audio tape interface is doing. SEE your programs load!

*Two 8-bit latchable output ports with 16 LED's in a linear array. Two DIP headers for easy port access. NOTE: the LED's are great for status indicators, educational or game use. Make lights dance! Make a bar graph indicator. Make a disco-KIM!

*MORE™ includes a prototyping area for your individual projects, i.e. add additional RUN EPROM sockets, etc.

*All IC's in sockets. No alterations to KIM!

*G-10 epoxy board with plated through holes. 4" x 10¾".

*Four sets of programming and/or run personality keys that allow you to change EPROM types in seconds with no board alteration. Keys for 2708, 2716 (3 voltage) 2758, 2716 (5 V only) all included!

*MORE™ will not become obsolete for many years and is a powerful and useful tool: Ideal for education, development, dedicated applications. And just plain fun!

MORE™ board complete with four personality key sets (8 keys), documentation, software listings, assembled and tested. Less EPROM's and RAM's. $169.95 prepaid in U.S.

Check or M.O. to:
T.T.I.
P.O. Box 2328
Cookeville, TN. 38501

Please allow 4 to 6 weeks for delivery.
Software on tape--$10.00 extra.
Software in 2708 EPROM--$30.00 extra.

from the table and pushed on the stack. Next, MLDSPT issues an RTS instruction which pulls the address from the stack into the program counter. Thus, the subroutine is invoked. Please heed the notes on the listings. The addresses in the address table are the actual machine language subroutine addresses minus one.

Examples in the next section will illustrate the use of MLDSPT.

ARRSAV/ARRLOD

One of the curious omissions from Microsoft BASIC is a feature to save and load data using cassette tape. Simple machine language routines have been written to overcome this deficiency. Currently, data stored in either floating point or integer arrays can be stored on or read from tape. To save character string arrays, the information must first be moved to a numeric array. To read character string arrays, the information must be read into a numeric array and then moved to a character string array.

SAVING DATA

Perform the following steps to save data on tape.

1. In my system, using MLDSPT, the save routine is machine language routine number 3. This may vary in other systems. POKE 3 into location 254 (decimal).

2. Inform ARRSAV of the name of the array to be saved. This is accomplished by POKing the numeric value of the first character of the array name into location 6027 (decimal). Similarly, POKE the numeric value of the second character of the array name into location 6028 (decimal). If the array name is only one character long, set location 6028 to 0. If the array is an integer array, set the high order bits of 6027 and 6028 to 1. Do this even if the integer array name is only one character long. A DIM statement for the array must appear before data can be written form the array.

3. Prepare the tape recorder and invoke the USR function. USR will activate MLDSPT which will, in turn, execute ARRSAV. Control will return to BASIC.

LOADING DATA

Reading data back in is very similar, but there are a few cautions to be observed.

1. POKE the number of ARRLOD into location 254. In my system ARRLOD is routine number 4.

2. As described above, POKE the array name into decimal locations 6027 and 6028. Data will only load into an array having the same name as the array from which the data was written to tape. Furthermore, the data must be loaded into an array having at least as many bytes as the original array. Finally, a DIM statement must appear for an array before data can be loaded into the array.

3. Prepare the tape recorder and invoke the USR function.

EXAMPLES

The following BASIC program segments show how MLDSPT and ARRSAV/ARRLOD are used.

Saving Data

```
10 DIM A(100)
20 POKE 254,3 : REM SET UP MLDSPT
30 POKE 6027,ASC("A"): REM ARRAY NAME
40 POKE 6028,0
50 Z=USR(0): REM SAVE DATA
```

Loading Data

```
10 DIM A(100): REM DEFINE ARRAY BEFORE LOAD
20 POKE 254,4: REM SET UP MLDSPT
30 POKE 6027,ASC("A"): REM ARRAY NAME
40 POKE 6028,0
50 Z=USR(0):  REM LOAD DATA
```

HOW IT WORKS

The commented listings expalin fairly well the operation of ARRSAV/ARRLOD. The idea is to search BASIC's array symbol table for the desired array name. Once located, data is either written from the proper symbol table entry or loaded into the entry.

Each entry in the array symbol table is organized as follows:

Byte 0 - first character of array name
Byte 1 - second character of array name
Byte 2 - low order byte of the length of the entry
Byte 3 - high order byte of the length of the entry

Information on the number of dimensions in the array and the actual contents of the array follow the first four bytes.

Users should be aware that ARRSAV/ARRLOD represents a minimal approach to tape data handling. Completely absent from these routines is any kind of error-checking facility. For example, no indication is given if the name POKEd into locations 6027 and 6028 cannot be found in the array symbol table. No reporting of read errors is performed. Lastly, no checking is done to prevent the destruction of the array symbol table.

Error checking and a file-naming mechanism are areas where an experimenter can customize and improve on the ideas presented here.

NOTES ON THE LISTINGS

The listings for MLDSPT and ARRSAV/ARRLOD were made usng a home-brew editor/text formatter and mnemonic assembler. All addresses are assumed to be in hex in order to save typing of the leading $.

The routines are stored in 2708 EPROMs in my system - hence the awkward addresses. The routines may be relocated as long as the table of addresses in MLDSPT is properly updated.

Lastly, note that my tape save and load subroutines are also located in EPROM at $E800 and $E886. The JSRs to these routines can be replaced with JMPs to KIM routines at $1800 and $1873 in that order. However, the KIM tape routines return to the KIM monitor. BASIC will have to be restarted in the command mode from the terminal or keypad.

```
                ;
                ; MICROSOFT BASIC MACHINE
                ; LANGUAGE DISPATCHER
                ;
                ; INVOKED BY Z=USR(0).
                ; SUBROUTINE NUMBER IS POKED INTO
                ; LOCATION $FE (254).
                ; SUBROUTINE NUMBER MUST BE IN THE
                ; RANGE 0-127.
                ;
                ;
                ; FETCH SUBROUTINE NUMBER.
                ; MULTIPLY BY 2 TO OBTAIN OFFSET
                ; INTO ADDRESS TABLE.
                ; INVOKE MACHINE LANGUAGE ROUTINE.
                ;
E02D A5FE   MLDSPT LDA   FE
E02F 0A            ASL   A
E030 AA            TAX
E031 BD3AE0        LDA   E03A,X
E034 48            PHA
E035 BD3BE0        LDA   E03B,X
E038 48            PHA
E039 60            RTS
```

```
                ;
                ;
                ; TABLE OF MACHINE LANGUAGE
                ; ROUTINE ADDRESSES.
                ; NOTE: ENTRY IN THE ADDRESS TABLE
                ; IS THE ACTUAL M.L. ROUTINE
                ; ADDRESS MINUS ONE.
                ; ADDRESSES ARE STORED HI, LO.
                ;
                ; INKEY
                ;
E03A E3                 .BYTE $E3
E03B FF                 .BYTE $FF
                ;
                ; BITON - TURN ON GRAPHICS CELL.
                ;
E03C E6                 .BYTE $E6
E03D B2                 .BYTE $B2
                ;
                ; BITOFF-TURN OFF GRAPHICS CELL.
                ;
E03E E6                 .BYTE $E6
E03F C7                 .BYTE $C7
                ;
                ; ARRSAV-SAVE ARRAY DATA.
                ;
E040 E0                 .BYTE $E0
E041 67                 .BYTE $67
                ;
                ; ARRLOD-LOAD ARRAY DATA.
                ;
E042 E0                 .BYTE $E0
E043 6E                 .BYTE $6E

. LOCATIONS $E044 through $E067 are reserved for
  future subroutine calls.
                ; MICROSOFT BASIC ARRAY
                ; SAVE AND LOAD.
                ;
                ; ROUTINE TO SAVE AND LOAD DATA
                ; FROM BASIC ARRAYS.
                ; SAVE IS ROUTINE NO. 3.
                ; LOAD IS ROUTINE NO. 4.
                ; POKE THE NAME OF THE ARRAY INTO
                ; LOCATIONS $178A AND $178B.
                ; IF THE ARRAY NAME IS ONE
                ; CHARACTER, PUT $00 IN $178B.
                ; IF THE ARRAY IS AN INTEGER
                ; ARRAY, SET THE HIGH ORDER BITS
                ; OF $178A AND $178B TO 1.
                ;
                ; (USES $FC AND $FD AS POINTERS.)
                ;
                ;
                ;
                ; ENTRY POINT FOR ARRAY SAVE.
                ; DATA IS SAVED WITH ID = 1.
                ;
E068 A901   ARRSAV LDA  #$01
E06A 8DF917        STA  17F9
E06D D005          BNE  *+5
                ;
                ; ENTRY POINT FOR ARRAY LOAD.
                ; LOAD WITH ID = $FF.
                ;
E06F A9FF          LDA  #$FF
E071 8DF917        STA  17F9
                ;
                ; PICK UP START OF SYMBOL TABLE
                ; FROM $7C,$7D AND STORE IN
                ; POINTER REGISTER $FC,$FD.
                ;
E074 A57C   TAP0   LDA  7C
E076 85FC          STA  FC
E078 A57D          LDA  7D
E07A 85FD          STA  FD
                ;
                ; TEST IF POINTER HAS REACHED END
                ; OF SYMBOL TABLE $7E,$7F.
                ; RETURN TO CALLER IF END OF TABLE
                ; IS REACHED.
                ;
E07C A57E   TAP1   LDA  7E
E07E C5FC          CMP  FC
E080 D007          BNE  *+7
E082 A57F          LDA  7F
E084 C5FD          CMP  FD
E086 D001          BNE  *+1
E088 60            RTS
                ;
                ; MOVE SYMBOL TABLE ENTRY LENGTH
                ; TO KIM CASSETTE ENDING ADDRESS.
                ; LENGTH FIELDS ARE OFFSET FROM
                ; START OF ENTRY BY 2&3 BYTES.
                ; $17F7 & $17F8 ARE KIM CASSETTE
                ; ADDRESSES.
                ;
E089 A003   TAP2   LDY  #$03
E08B B1FC          LDA  (FC),Y
E08D 99F517        STA  17F5,Y
E090 88            DEY
E091 C001          CPY  #$01
E093 D0F6          BNE  *-10
                ;
                ; TEST FOR MATCH BETWEEN DESIRED
                ; ARRAY NAME ($178A & $178B) AND
                ; CURRENT SYMBOL TABLE ENTRY NAME.
                ; ARRAY NAME IS 2 CHARACTERS LONG
                ; AND IS OFFSET 0 & 1 FROM START
                ; OF SYMBOL TABLE ENTRY. USE THE
                ; X REGISTER TO COUNT THE NUMBER
                ; OF MATCHED CHARACTERS.
                ;
                ; CARRY SET IF NAMES MATCH.
                ; CARRY CLEARED IF MISMATCH.
                ;
E095 A200   TAP3   LDX  #$00
E097 B1FC          LDA  (FC),Y
E099 D98A17        CMP  178A,Y
E09C D007          BNE  *+7
E09E E8            INX
E09F 88            DEY
E0A0 10F5          BPL  *-11
E0A2 38            SEC
E0A3 B001          BCS  *+1
E0A5 18            CLC
                ;
                ; ALWAYS SET POINTER TO NEXT
                ; SYMBOL TABLE ENTRY AND SET UP
                ; KIM CASSETTE TAPE ADDRESSES.
                ; $17F5,$17F6 IS KIM S.A.
                ; $17F7,$17F8 IS KIM E.A.
                ; $FC,$FD IS POINTER.
                ; NOTE: LENGTH HAS BEEN LOADED
                ; INTO E.A.
                ;
E0A6 A5FC   TAP5   LDA  FC
E0A8 8DF517        STA  17F5
E0AB 6DF717        ADC  17F7
E0AE 8DF717        STA  17F7
E0B1 85FC          STA  FC
E0B3 A5FD          LDA  FD
E0B5 8DF617        STA  17F6
E0B8 6DF817        ADC  17F8
E0BB 8DF817        STA  17F8
E0BE 85FD          STA  FD
                ;
                ; X REGISTER WILL CONTAIN $02
                ; IF PROPER SYMBOL TABLE ENTRY
                ; WAS LOCATED. OTHERWISE, GO
                ; LOCATE AND TEST NEXT ENTRY IN
                ; THE SYMBOL TABLE.
                ;
E0C0 E002          CPX  #$02
E0C2 D0B8          BNE  *-72
                ;
                ; EXAMINE KIM TAPE ID ($17F9)
                ; TO DETERMINE WHETHER TO SAVE
                ; OR LOAD.
                ; $E886 IS TAPE INPUT.
                ; $E890 IS TAPE OUTPUT.
                ; RETURN TO BASIC.
                ;
E0C4 ADF917        LDA  17F9
E0C7 C9FF          CMP  #$FF
E0C9 D004          BNE  *+4
E0CB 2086E8        JSR  E886
E0CE 60            RTS
E0CF 2000E8        JSR  E800
E0D2 60            RTS
```

BASIC SPEED REPORT

by Harry D. Bolch
Lone Star Elec.
PO Box 488
Manchaca Tx 78652

Glad to see the 'Notes become your full-time job. When last we talked you were working at MOS Technology. You may recall that we were discussing the possibility of increasing the clock speed on a KIM. I did replace the 6530's with new ones, the 6502 with a 6502A, and the 1MHz Xtal with a 2.01 MHz Xtal. To my knowledge, the only eight-bit system that is faster is the OSI system described by Curt Priest of Cambridge, MASS; the 2.01 MHz KIM with the Microsoft BASIC executes benchmark programs more than 25% faster than the fastest 4MHz Z-80 system.

forth

KIMFORTH is moving right along. The source code has been typed in and it assembles correctly and runs!!! FORTH documentation is fairly complete and is now being typed in to the system. Some cassette support software still needs to be added and verified to operate correctly so KIMFORTH isn't quite ready for distribution yet.

Getting a software package of this size to market is no easy thing and usually takes more time than one would like.

You APPLE owners will be happy to hear that, according to the FORTH INTEREST GROUP, Captain Software of Berkeley California is offering a disc-based APPLE-FORTH system that conforms to the "international FORTH standards."

It was further stated in correspondence with the FORTH INTEREST GROUP that the programs being offered by Programma Consultants and Seawell Marketing are not true FORTH implementations because, at least in the Programma software, the "inner-interpreter" concept, essential to FORTH, is not implemented.

focal

FOCAL MODS

....speed it up
a little...

from Bernhard Mulder
Mozart Str 1
6744 Kandel
West Germany

We change the procedure EATCR (and EATCR1) which is called by the findline, which in turn is called from the GOTO, IF, ON, DO command routines.

We assume that the carriage return char is in memory and avoid the call of the routine GETC, where switches are tested which will never be set, when we caome from EATCR (start the following routine at $26D0 in the Aresco version 3D and $26DD in the "6502 Program Exchange" FCL-65E (V3D).

```
C6  2A    ECR1  DEC  TXTP     ;EATCP1
A4  2A    EACR  LDY  TXTP     ;EATCR
A9  0D          LDA  #0D      ;load CR which we are
                               looking for
D0  01          BNE  TST1
C8        LABL  INY           ;next character in line
D1  28    TST1  CMP  (TXTA),Y;C.R. found already?
D0  F8          BNE  LABL     ;branch if noy
B1  28          LDA  (TXTA),Y;store away for others
85  2B          STA  CHAR
C8              INY
98              TYA           ;calculate address
                               part CR.
18              CLC
65  28          ADC  TXTA
85  28          STA  TXTA
85  33          STA  TXA2
A5  29          LDA  TXT1
69  00          ADC  #00
85  29          STA  TXT1
85  34          STA  TA21
A9  00          LDA  #00
85  2A          STA  TXTP
85  35          STA  TXP2
60        ENCR  RTS
```

Make the following changes to Aresco V3D

```
208D 20 D0 26 (was 20 D7 26)
21FF 20 D0 26 (was 20 D7 26)
22E1 20 D0 26 (was 20 D7 26)
2752 20 D2 26 (was 20 D0 26)
```

or make the following changes to the Program Exchange FCL-65E

```
208D 20 DD 26 (was 20 E4 26)
21FF 20 DD 26 (was 20 E4 26)
22E3 20 DD 26 (was 20 E4 26)
275F 20 DF 26 (was 20 DD 26)
```

Those of you with ROR instructions in your CPU can eliminate the ROR simulator in FOCAL with the following code.

Start at $3291 for the Aresco version 3D
Start at $3293 for the Program Exchange FCL-65E

```
7E 89 00  ROR1  ROR  EP4,X ;need not simulate ROR
E8              INX
D0 FA           BNE  ROR1
60              RTS
```

Plenty more mods in store for FOCAL. Until next issue.

tiny basic

TINY BASIC CASSETTE SAVE & LOAD

by William C. Clements, Jr.
Univ. of Alabama
Chem & Metal Eng.
Box 2662
University, Al 35486

I recently bought TINY BASIC and the accompanying experimenter's kit, and have enjoyed finding out how the BASIC statements are broken down and implemented. With a little study one can easily pick up the pseudolanguage used to program the inner interpreter, and then all sorts of possibilities exist for custom modifications to suit one's whim. I noticed the comments about transferring BASIC statements to and from cassette tape in Issue 13 (Lew Edwards, p. 14), and thought perhaps your readers might be interested in how I added the SAVE and LOAD commands to my version of TINY BASIC for the KIM-1. With my implementation, TINY can use the existing KIM monitor routines (or any others if one wishes) to save and load programs, and transfer of starting and ending addresses, etc. is handled by a machine language routine. The cassette file number is specified in the added BASIC commands: SAVE X or LOAD X, where X is any integer 0 _ X _ 255 corresponding to KIM file I.D.'s 00 through FF. My version of TINY is the one having the cold start at 2000 hex; corresponding address offsets can be added for other versions.

The patch to the Intermediate Interpreter is made at relative location 00B7, as shown on p.38 of the Experimenter's Manual. This is address 2827 absolute. The patch is as follows:

```
                ;test for keyword SAVE
00B7 8B534156C5 TAPE BC LOAD "SAVE"
                ;push start address of
00BC 09 29          LB 29
                ;save routine onto stack
00BE 09 0E          LB 0E
                ;do it again
00C0 0B             Q DS
                ;error stop if file id not number
00C1 C0             BN
                ;go to save routine at 290E(H)
00C2 2E             US
                ;test for keyword LOAD
00C3 8A4C4F41C4 LOAD BC DFLT "LOAD"
                ;push start address of
00C8 09 29          LB 29
                ;load routine onto stack
00CA 09 28          LB 28
                ;go to load routine at
00CC 38 C0          J Q
                ;2928(H) via above instructions
00CE A0          DFLT BV *
                '      '  (continue with
                '      '  remaining IL code)
```

The constants after the LB commands specify the hex addresses of the machine language routines which handle the SAVE X and LOAD X functions. The line labeled DFLT is thus moved from relative location 00B7 to 00CE, resulting in an offset of 17_H or 23_D for remaining lines. This must be accommodated in the jump and jump subroutine commands in the I.L. The changes in destination for those instructions which jump beyond the patch are listed. All error messages originating beyond the patch will also be increased by 23_D.

My version jumps to a pair of machine language routines which initialize the file i.d., SAL, SAH, and the TINY BASIC registers. BASIC files are saved using a Hypertape routine stored in EPROM at location $C400_H$; if the user wishes to use the KIM tape dump routine, he should change the contents of location 2927_H to 18_H. Appropriate routines can of course be relocated anywhere the user wishes, so long as the correct entry points are provided for in the I.L. patch. After execution of a SAVE or LOAD, TINY must be manually reentered at the warm start (the limits of memory for the BASIC statements are set for my system when BASIC is first entered). A jump to warm start could of course be placed at the end of the tape dump and load routines if ones stored in RAM instead of ROM were being used.

These alterations were worth their trouble in added convenience: SAVE 01 is a lot easier than exiting TINY, storing 01 in 17F9, and looking up the memory bounds for the BASIC statements to set SAL and SAH manually. I hope this modification will be of interest to other users of TINY BASIC.

MACHINE LANGUAGE ROUTINES USED BY THE PATCH

```
2906  8D F9 17  00    STA 17F9(H)  STEPS COMMON TO
      A9 00           LDA $00      BOTH
      85 F1           STA 00F1(H)  ROUTINES
      60              RTS

290E  20 06 29  SAVE  JSR 00       FILE SAVE ROUTINE
      A5 20           LDA 0020(H)
      8D F5 17        STA 17F5(H)
      A5 21           LDA 0021(H)
      8D F6 17        STA 17F6(H)  INITIALIZATION
      A5 24           LDA 0024(H)
      8D F7 17        STA 17F7(H)
      A5 25           LDA 0025(H)
      8D F8 17        STA 17F8(H)
      4C 00 C4        JMP HYPERTAPE
2928  20 06 29  LOAD   JSR QQ set 17F9(H), 00F1(H)
      4C 73 18        JMP TPLOAD read tape
292E  AD ED 17  ENTER  LDA EAL set address
      85 24           STA 0024(H) at end
      AD EE 17        LDA EA(H)  of BASIC
      85 25           STA 0075(H) program file
      4C 03 20        JMP BASIC go to warm start
```

Restart BASIC at ENTER (loc. $292E_H$) after loading. Restart at warm start (2003_H in my version) after saving.

Summary of additional modifications to I.L. Code (new transfer statement destination caused by insertion of patch)

Relative Location (See pp. 36-40 TINY BASIC Experimenter's Manual)	New Instruction
0014	30 D3
001F	30 D3
0029	30 D3
004B	30 D3
0052	30 D3
0054	31 4B
0056	30 D3
0073	30 D3
009E	30 D3
00BE	30 EA
00C4	30 EA
00C8	30 EA
00CE	30 EA
00D3	30 F9
00D7	30 F9
00F7	31 47
0114	30 D3
0116	31 41
0118	31 41
0125	30 D3
012C	38 D3

TINY BASIC STRINGS

by Michael E Day
2590 DeBok Rd
West Linn, Or 97068

Here is the string mod I've been using which I access thru the USR verb. This requires 512 bytes of memory, and is relocatable and will run out of ROM or protected memory except for the storage area which operates out of RAM, however it can be located in any 256 byte block of free memory.

PEEK $ USR(2816,ADDRESS)
PEEK at string at the string relative address ADDRESS. Returns decimal value of addressed byte.

POKE $ USR(2822,ADDRESS,DATA)
POKE data byte DATA into the string relative address ADDRESS. Returns string relative address plus one.

INPUT SP$ USR(2832,BEGIN,END)
INPUT a string of characters beginning with string relative address BEGIN, echoing back a space with each input character, until a carriage return is encountered, or the ending address END is reached. Returns the string relative ending address plus one.

INPUT $ USR(2839,BEGIN,END)
INPUT a string of characters as in INPUT SP$, but without the space echo. Returns the string relative ending address plus one.

PRINT SP$ USR(2905,BEGIN,END)
PRINT the character string beginning with the string relative address BEGIN, and print a space after each character, until a carriage return is encountered, or the ending address END is reached. Returns the string relative ending address plus one.

PRINT $ USR(2912,BEGIN,END)
PRINT the character string as in PRINT SP$, but without the space echo. Returns the string relative ending address plus one.

SEARCH $ USR(2946,BEGIN,DATA)
SEARCHes for the BCD equivalent of decimal value DATA, beginning at string relative address BEGIN, until a match is found, or the ending address of variable "L" is reached. Returns the string relative ending address plus one.
If a match is not found the return address will be 0 (zero). Variable "L" is decremented once per test until match is found, or it is 0.

```
MOVE $     USR(2966,FROM,TO)  (Length in variable
                               "L")
```

MOVEs a group of characters of the length in variable "L" beginning at the relative string address FROM, and moving them to relative string address TO, for the length of variable "L". Returns the FROM ending address plus one. Variable "L" is zeroed. (Lower 8 bits only, see notes on addressing of strings).

SET POINTERS

These are memory formating routines that are addressed by the other routines, and are listed with USR statements only for reference. They do not need to be accessed by TINY.

OPERATIONAL NOTES

Addressing is limited to 0-256 (8 bit addressing) and the upper bits are ignored (I.E. 512 will appear as a 0, and 513 will appear as a 1).

The string array table is perminently fixed to 256 bytes in length, and dedicated for this purpose. This table may be located anyplace in RAM so long as intrusion from other sources is not allowed. Relocation is done by changing the page location address at 0BAA (0BAA A0 0C LDY #0C). The routines that access the table are clean. (They are relocatable, and will operate out of ROM or protected memory.)

All data passed through the USR statements both to and from is in decimal. The data inside the routines however, remain in BCD.

In the PRINT and INPUT routines, if the BEGIN address is less than the END address, an error exit will occur which causes the exit address to be 0, and the funciton asked for is not performed.

If only one address is given, the second address will be assumed to be equal to the first address given (I.E. USR(2912 0) will print out a single character at location 0 and return an address value of 1 to TINY.

As with any USR statement in TINY, the address and data information passed through the USR statement can be calculated from any expression.

(Such as USR(2912,B,E-2) can be used to print a string starting at the address in variable "B", and using the E-2 to suppress the ending carriage return, and another variable can be used to pick-up the returning ending address.)

The routines given have been located at the end of TINY, as this allows for easy isolation from TINY by revising the user memory starting address located at 028B.

```
028B   A9 0B     LDA  #0B  Old starting address
028B   A9 0D     LDA  #0D  New starting address
```

This is the only place that TINY references this, so it is the only thing that needs to be changed. NOTE: A cold start MUST be done after this change to set the pointers, or else they will have to be set by hand.

The entire string mod requires less than 512 bytes of memory (256 bytes for the array, and 187 bytes for the routines.)

A possible mod would be to place the array page address in zero page memory, and modify it with TINY before going into the routines. This would allow for greater than 256 bytes, but program management must be closely followed, or strange things might happen!!!

The cancel code used in TINY will terminate an INPUT $ without putting the character into the array, therefore this code can not be used directly. All previous characters will have been inserted however.

```
PEEK $     USR(2816,ADDRESS)
0B00  20 A8 0B  JSR  0BA8    Set pointers A
0B03  B1 18     LDA  (18),Y  Pick up data
0B05  60        RTS          Return to TINY


POKE $     USR(2822,ADDRESS,DATA)
0B06  20 A8 0B  JSR  0BA8    Set pointers A
0B09  91 18     STA  (18),Y  Store data
0B0B  E6 18     INC  18      Increment pointer
0B0D  A5 18     LDA  18      Return address to TINY
0B0F  60        RTS          Return to TINY


INPUT SP$   USR(2832,BEGIN,END)
0B10  20 B5 0B  JSR  0BB5    Set pointers B
0B13  84 1B     STY  1B      Clear 1B
0B15  B0 03     BCS  0B1A    Goto Input routine


INPUT $     USR(2839,BEGIN,END)
0B17  20 B5 0B  JSR  0BB5    Set pointers B
0B1A  A9 3F     LDA  #3F
0B1C  20 09 02  JSR  0209    Print a "?"
0B1F  A9 20     LDA  #20
0B21  20 09 02  JSR  0209    Print a "SP"
0B24  20 06 02  JSR  0206    Get a character
0B27  CD 10 02  CMP  0210    Is it "ESC"?
0B2A  F0 28     BEQ  0B54    If so return to TINY
0B2C  CD 0F 02  CMP  020F    Is it "BS"?
0B2F  D0 11     BNE  0B42    If so back up
0B31  A5 1A     LDA  1A
0B33  C5 18     CMP  18      Is it begin of array?
0B35  F0 E8     BEQ  0B1F    If so restart
0B37  C6 18     DEC  18      Decrement pointer
0B39  A5 1B     LDA  1B      Input SP$ ?
0B3B  D0 E7     BNE  0B24    If not get next
                               character
0B3D  AD 0F 02  LDA  020F    Get "BS"
0B40  90 DF     BCC  0B21    Print it
0B42  91 18     STA  (18),Y  Store data


INPUT $     USR(2839,BEGIN,END) Con't.
0B44  E4 18     CPX  18      Is it end of array?
0B46  F0 0C     BEQ  0B54    If so return to TINY
0B48  E6 18     INC  18      Increment pointer
0B4A  C9 0D     CMP  #0D     Is it a "CR"
0B4C  F0 08     BEQ  0B56    If so return to TINY
0B4E  A5 1B     LDA  1B      Print a "SP"?
0B50  D0 D2     BNE  0B24    If not get next byte
0B52  F0 CB     BEQ  0B1F    Print a "SP"
0B54  E6 18     INC  18      Increment pointer
0B56  A5 18     LDA  18      Return exit address
                               to TINY
0B58  60        RTS          Return to TINY

PRINT SP$  USR(2905,BEGIN,END)
0B59  20 B5 0B  JSR  0BB5    Set pointers B
0B5C  84 1B     STY  1B      Clear 1B
0B5E  B0 03     BCS  0B63    Goto print routine


PRINT $     USR(2912,BEGIN,END)
0B60  20 B5 0B  JSR  0B5B    Set pointers B
0B63  B1 18     LDA  (18),Y  Pick up data
0B65  20 09 02  JSR  0209    Print character
0B68  E4 18     CPX  18      Is it end of array?
0B6A  F0 11     BEQ  0B7D    If end return to TINY
0B6C  E6 18     INC  18      Increment pointer
0B6E  C9 0D     CMP  #0D     Is it a "CR"?
0B70  F0 0D     BEQ  0B7F    If so return to TINY
0B72  A5 1B     LDA  1B      Print a "SP"?
0B74  D0 ED     BNE  0B63    If not get next byte
0B76  A9 20     LDA  #20
0B78  20 09 02  JSR  0209    Print a "SP"
0B7B  D0 E6     BNE  0B63    Go get next byte
0B7D  E6 18     INC  18      Increment pointer
0B7F  A5 18     LDA  18      Get exit address
0B81  60        RTS          Return to TINY

SEARCH $    USR(2946,BEGIN,DATA) (Length in var-
                                  iable "L")
0B82  02 A8 0B  JSR  0BA8    Set pointers A
0B85  B1 18     LDA  (18),Y  Pick up test byte
0B87  E6 18     INC  18      Increment pointer
0B89  C5 1A     CMP  1A      Found match?
0B8B  F0 06     BEQ  0B93    If so return to TINY
0B8D  C6 98     DEC  98      Decrement variable'L'
0B8F  D0 F4     BNE  0B85    If not get next byte
0B91  84 18     STY  18      Clear 18 (pointer)
0B93  A5 18     LDA  18      Return exit address
                               to TINY
0B95  60        RTS          Return to TINY


MOVE $      USR(2966,FROM,TO)  (Length in variable
                                "L")
0B96  20 A8 0B  JSR  0BA8    Set pointers A
0B99  B1 18     LDA  (18),Y  Pick up byte
0B9B  91 1A     STA  (1A),Y  Store it
0B9D  E6 18     INC  18
0B9F  E6 1A     INC  1A      Increment pointers
0BA1  C6 98     DEC  98      Decrement variable'L'
```

```
0BA3  D0 F4     BNE  0B99     If end return to TINY
0BA5  A5 18     LDA  18       Return exit address
                                to TINY
0BA7  60        RTS           Return to TINY


SET POINTERS A  USR(2984,Y,A)
0BA8  84 18     STY  18       Save begin
0BAA  A0 0C     LDY  #0C      Set array page
0BAC  84 19     STY  19       Store array page
0BAE  84 1B     STY  1B       Store array page
0BB0  A0 00     LDY  #00      Clear Y
0BB2  85 1A     STA  1A       Save A
0BB4  60        RTS           Exit
```

```
SET POINTERS B  USR(2997,Y,A)
0BB5  20 A8 0B  JSR  0BA8     Set pointers A
0BB8  AA        TAX           Save end
0BB9  A5 18     LDA  18       Recapture begin
0BBB  85 1A     STA  1A       Save it
0BBD  E4 18     CPX  18       Bad address?
0BBF  B0 03     BCS  0BC4     If so go error
0BC1  68        PLA
0BC2  68        PLA           Discard string link
0BC3  98        TYA           Clear A
0BC4  60        RTS           Exit


READ KEY    USR(3064)
0BF8  AD 00 C0  LDA  0C00     Pick up data
0BFB  29 7F     AND  #7F      Clear bit 8 (Strobe)
0BFD  A0 00     LDY  #00      Clear Y
0BFF  60        RTS           Return to TINY
```

assembler

```
HDE ASSEMBLER REV 1.1

LINE#  ADDR  OBJECT     SOURCE        PAGE 0001

 0010  2000             ;THIS IS A SYMBOL TABLE SORT ROUTINE FOR
 0020  2000             ;THE MOS/ARESCO ASSEMBLER. IT GETS PATCHED
 0030  2000             ;IN TO THE ASSEMBLER AND INSTALLED IMMEDIATELY
 0040  2000             ;FOLLOWING IT STARTING AT $F067.
 0050  2000             ;WRITTEN BY J. FATOVIC
 0060  2000
 0070  2000
 0080  2000             ;FOR THIS ROUTINE TO OPERATE, CHANGE
 0090  2000             ;$EB9D AND $EB9E TO $67 $F0 RESPECTIVELY.
 0100  2000             ;
 0110  2000                     *=$10
 0120  0010             FLAG    *=*+1
 0130  0011             CADL    *=*+2
 0140  0013             NADL    *=*+2
 0150  0015
 0160  0015             MON     =$1C14
 0170  0015                     *=$DF
 0180  00DF             STSAVE  *=*+2
 0190  00E1             NSTAT   =$EAEE
 0200  00E1             SYMLEN  =$06
 0210  00E1                     *=$69
 0220  0069             SYMPTR  *=*+2
 0230  006B                     *=$4E
 0240  004E             NOSYM   *=*+2
 0250  0050
 0260  0050
 0270  0050                     *=$F067
 0280  F067                     .OFF 2000
 0290  F067
 0300  F067  A9 01      SORT1   LDA #1          ; SET FLAG
 0310  F069  85 10              STA FLAG
 0320  F06B
 0330  F06B  A5 DF              LDA STSAVE      ; INIT. CURRENT ADR
 0340  F06D  85 11              STA CADL
 0350  F06F  A5 E0              LDA STSAVE+1
 0360  F071  85 12              STA CADL+1
 0370  F073
 0380  F073  A9 01              LDA #1          ; INITIATE POINTER
 0390  F075  85 6A              STA SYMPTR+1
 0400  F077  A9 00              LDA #0
 0410  F079  85 69              STA SYMPTR
 0420  F07B
 0430  F07B  20 D9 F0           JSR ADRNS       ; INIT. ADR OF NEXT SYMBOL
 0440  F07E
 0450  F07E  A0 00              LDY #0
 0460  F080  B1 11      SORT2   LDA (CADL),Y
 0470  F082  D1 13              CMP (NADL),Y
 0480  F084
 0490  F084  F0 2D              BEQ SRT1        ; IF EQUAL COMP NEXT CHAR
 0500  F086
 0510  F086  B0 33              BCS REX         ; NEXT SYMBOL PRECEDES -
 0520  F088             ;SO EXCHANGE REGS
 0530  F088
 0540  F088  A0 00      SORT3   LDY #0
 0550  F08A
 0560  F08A  A5 13              LDA NADL        ; MAKE ADR OF NEXT SYMBOL -
 0570  F08C  85 11              STA CADL        ;-CURRENT ADDRESS
 0580  F08E  A5 14              LDA NADL+1
 0590  F090  85 12              STA CADL+1
 0600  F092
 0610  F092  20 D9 F0           JSR ADRNS
```

```
0620  F095
0630  F095
0640  F095  E6 6A           INC SYMPTR+1     ; INCREMENT POINTER
0650  F097  D0 04           BNE COMP
0660  F099  E6 69           INC SYMPTR
0670  F09B  F0 13           BEQ FINE
0680  F09D
0690  F09D
0700  F09D  B8        COMP   CLV             ; FIND IF THIS IS THE
0710  F09E  A5 69           LDA SYMPTR       ;LAST LINE
0720  F0A0  C5 4E           CMP NOSYM
0730  F0A2  90 DC           BCC SORT2
0740  F0A4
0750  F0A4  A5 6A     COMP2  LDA SYMPTR+1
0760  F0A6  C5 4F           CMP NOSYM+1
0770  F0A8  90 D6           BCC SORT2
0780  F0AA
0790  F0AA  A5 10           LDA FLAG         ; CHECK IF FLAG SET, IF SET EXIT
0800  F0AC  29 01           AND #1
0810  F0AE  F0 B7           BEQ SORT1
0820  F0B0
0830  F0B0  4C EE EA  FINE   JMP NSTAT        ;  SORT COMPLETED
0840  F0B3
0850  F0B3  C8        SRT1   INY             ; POINT TO NEXT CHAR
0860  F0B4  C0 08           CPY #SYMLEN+2
0870  F0B6  D0 C8           BNE SORT2
0880  F0B8
0890  F0B8  4C 14 1C        JMP MON          ; ERROR, RETURN TO MONITOR
0900  F0BB
0910  F0BB
0920  F0BB  A0 00     REX    LDY #0           ; EXCHANGE REGISTERS
0930  F0BD  B1 11     RX1    LDA (CADL),Y
0940  F0BF  48               PHA
0950  F0C0  B1 13            LDA (NADL),Y
0960  F0C2  91 11            STA (CADL),Y
0970  F0C4  C8                INY
0980  F0C5  C0 08             CPY #SYMLEN+2
0990  F0C7  D0 F4             BNE RX1
1000  F0C9
1010  F0C9  88                DEY
1020  F0CA  68        RX2    PLA
1030  F0CB  91 13            STA (NADL),Y
1040  F0CD  88                DEY
1050  F0CE  10 FA            BPL RX2
1060  F0D0
1070  F0D0  A9 FE            LDA #$FE         ; RESET FLAG
1080  F0D2  25 10            AND FLAG
1090  F0D4  85 10            STA FLAG
1100  F0D6  4C BB F0          JMP SORT3
1110  F0D9
1120  F0D9
1130  F0D9  18        ADRNS  CLC
1140  F0DA  D8               CLD
1150  F0DB  A5 11            LDA CADL
1160  F0DD  69 08            ADC #SYMLEN+2
1170  F0DF  85 13            STA NADL
1180  F0E1  A5 12            LDA CADL+1
1190  F0E3  69 00            ADC #0
1200  F0E5  85 14            STA NADL+1
1210  F0E7  60                RTS
1220  F0E8
1230  F0E8              FINISH .END

ERRORS = 0000
```

AIM info

WARNING WARNING WARNING

by Leo Scanlon
Documentation Manager
Rockwell Microelectronic Devices
P.O. Box 3669, RC55
Anaheim, CA 92803

As Documentation Manager at Rockwell, I read with interest the article on AIM 65 Manual corrections published on page 20 of 6502 User Notes, No. 14. In this article, reader Jody Nelis recommends using Texas Instruments #TP-27225 thermal paper with the AIM 65. I urge you to warn all readers NOT to use this particular paper type in their AIM 65's. We have found this particular paper to be so highly abrasive that it can ruin the printer head in a matter of hours! In fact, because of experiences with this paper, we mailed a bright red warning to all AIM 65 owners, giving them a list of "approved" paper types.

The approved paper types are:

1. Rockwell #TT270
   Source: Rockwell Service Center
   6004 Threadgill Avenue
   El Paso, Texas 79924
   Phone (800) 351-6018

2. Sears #3974 or #3986

3. Olivetti #74707 or #74708

4. NCR #T1102
   Note: This paper produces black print, the others produce blue print.

All other corrections noted in the article have been picked up in a set of change pages that we mailed to AIM 65 owners that returned the Document Registration Form.

Incidentally, I always appreciate comments, corrections, gripes, etc. from readers of our manuals, and invite them to write to me directly at the address at the top of this article.

READING KIM CASSETTES from D.R.

Something not mentioned in the AIM65 owners manual makes reading KIM tapes impossible. The ID number is the last two digits of the file name. To read file #2B, enter 'xxx2B' in response to the 'F=' prompt. It took me quite awhile to figure it out,.and I thought I'd pass it along.

See $E3A4 in monitor listing for code.

EPROMS FOR AIM from D.R.

I have a modification for AIM 65 to allow use of on-board ROM sockets with 2758* EPROMS. Use a low profile 24-pin socket and bend legs 18, 19, 20 away from the body. Solder a bridge across 19,20 and then attach 1" wire wrap wire to junction scrap away soder-mask at botton of chip with an exacto knife and solder loose end of wire to exposed spot. Next attach 2½ - 3" piece os WWW to pin 18 and pin 10 of Z27. (For use in address range of D000 - DFFF.) See page 7-10 of AIM 65 User's Manual for pin # of different CS 11ines, insert socket and prom. Until the assembler and basic show up the sockets may be used for user programs with single key entry. Jade Computer Products have 2758 EPROMS (Intel +5V only) in stock, and have good service.

*You can do this with TMS 2516, 2716, 2732 etc.

NOTES ON AIM USER I/O

by Larry Goga
3816 Albright Ave.
Los Angeles, Ca 90066

According to the AIM-65 User's Guide, there is only one user character input subroutine which will display a cursor, echo a character, and allow the delete key to function. (see Section 7.7.1 in the User's guide.) This subroutine is identified variously as RUBOUT or RDRUB and resides at address E95F in the AIM Monitor. If you have experienced difficulty in getting this subroutines to support the DELETE function do not be alarmed. After consulting Rockwell about this problem it seems that there is more to using this subroutine than meets the eye.

The AIM documentation says that RDRUB uses the accumulator and the Y Index Register. Although this is true, what is not explained is that the Y-Index Register must be incorporated into the user's program.

If the Y-Index Register is zero when you call RDRUB then the DELETE function will not work. If the Y-Index Register is negative (MBS set) when you call RDRUB then strange things will happen when the DELETE key is pressed. You may also have found that when the DELETE function is working and you attempt to delete beyond the first character display position the program hangs-up and a question mark is shown in the center of the display. The only way out of this problem is "RESET".

The solution to these problems is to use the Y-index Register as an input counter. The Y-Index Register should be cleared to zero before calling RDRUB. Then, call RDRUB, and upon returning increment the Y-index Register. In this manner the Y-Index Register will contain a count of the number of characters which have been inputted from the keyboard to the display. This positive count in the Y-Index Register is the number of times the DELETE key will work (ie. Y=0, no deletes; Y=5, 5 deletes; etc.). The reason for using the Y-Index Register in this manner is that the RDRUB subroutines automatically decrements the Y-Index Register every time the DELETE key is pressed, but does not return from the subroutine until some other displayable character key is pressed.

An example of this use of RDRUB and the Y-Index Register will be found on page 35 of the AIM MONITOR LISTINGS. In a subroutine called ADDIN at address EAAE we find the Y-Index Register being cleared to zero in line 1668; and, after checking for a carriage return or space, we find the Y-Index Register being incremented at line 1673. After checking for not more than 10 characters inputted, the program loops back to input the next character. By implementing these steps in your program you should find that the DELETE function will work correctly.

(Courtesy of the San Fernando Valley 6502 Users Group)

MEMORY TEST PROGRAM

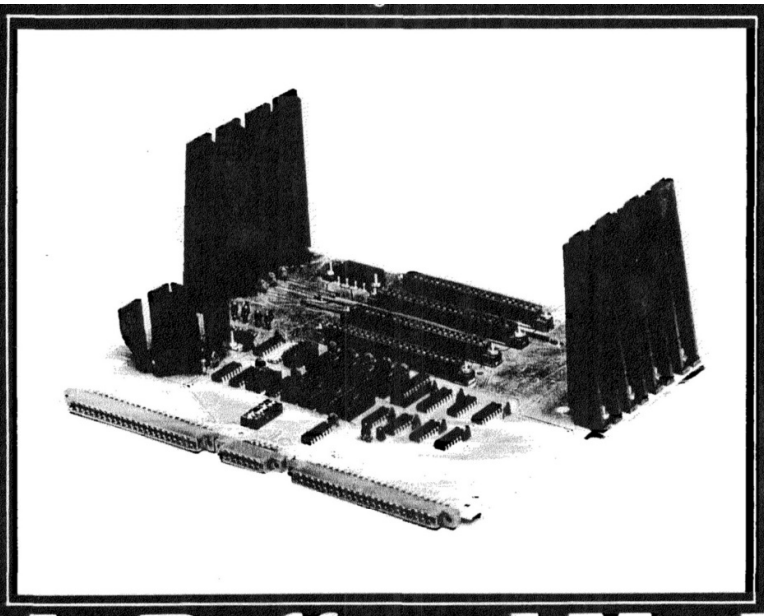
ADAPTED FROM "MEMORY TEST" BY JIM BUTTERFIELD
FROM "THE FIRST BOOK OF KIM"

MODIFIED TO RUN ON ROCKWELL AIM-65
BY LARRY GOGA

ENTERED: 5 JANUARY 1979
REVISED: 6 Jan 79

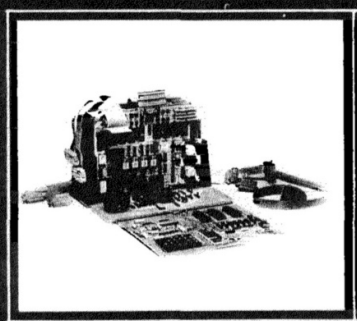
NOTE:
FOR "FROM" AND "TO" PROMPTS ENTER ONLY THE LOW AND HIGH PAGE LIMITS (HIGH ORDER ADDRESS BYTE) THEN TYPE <CR>.

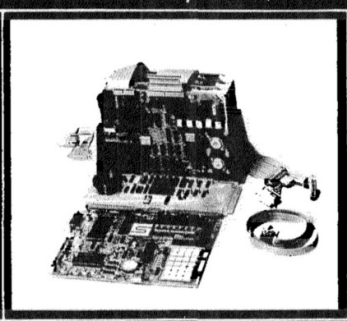
cont. on page 20

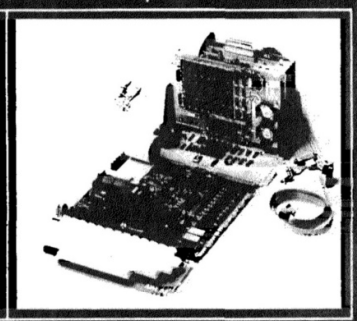


Little Buffered Mother

The ultimate Motherboard for any KIM-1, SYM-1, or AIM-65 system







Features:

- 4K Static RAM on board
- +5V, +12V, and −12V regulators on board
- 4 + 1 buffered expansion slots
- Accepts KIM-4 compatible boards
- Full access to application & expansion connector
- LED indicators for IRQ, NMI, and power-on
- Also compatible with SEA-1, SEA-16, the PROMMER, SEA-PROTO, SEA-ISDC, and more
- Onboard hardware for optional use of a 17 (128K addressing limit)
- Mounts like KIM-4 or with CPU board standing up
- 10 slot Motherboard expansion available - SEAWELL's Maxi Mother

Introduction price $99.
Regular price $119.
with 4K Static RAM installed $169.
Assembled — 3 month warranty

For further information contact:

SEAWELL Marketing Inc.
P.O. Box 17006
Seattle, WA 98107

SEAWELL Marketing Inc.
315 N.W. 85th
Seattle, WA 98117
(206) 782-9480

```
0000            MTEST  ORG   $0000       FID: MTEST

                ***** MEMORY LOCATIONS  *****

0000            BEGIN  *     $0085
0000            END    *     $0086
0000            POINTL *     $0087
0000            POINTH *     $0088
0000            FLAG   *     $0089
0000            MOD    *     $008A
0000            FLIP   *     $008B
0000            ADDRL  *     $A41C
0000            ADDRH  *     $A41D

                ***** SUB-ROUTINE EQUATES  *****

0000            FROM   *     $E7A3
0000            TO     *     $E7A7
0000            BLANKT *     $E83B
0000            CRLOW  *     $EA13
0000            MFAIL  *     $E839

                ***** BEGIN PROGRAM  *****

0000 20 A3 E7   INPUT  JSR   FROM   ;GET LOW PAGE LIMIT
0003 AD 1C A4          LDA   ADDRL
0006 85 85             STAZ  BEGIN
0008 20 3B E8          JSR   BLANKT
000B 20 A7 E7          JSR   TO     ;GET HIGH PAGE LIMIT
000E AD 1C A4          LDA   ADDRL
0011 85 86             STAZ  END

0013 A9 00      START  LDAIM $00    ;ZERO POINTERS FOR
0015 A8                TAY          ;LOW-ORDER ADDRESSES
0016 85 87             STAZ  POINTL
0018 85 89      BIGLP  STAZ  FLAG   ;=00 FIRST PASS, =FF SECOND PASS
001A A2 02             LDXIM $02
001C 86 8A             STXZ  MOD    ;SET 3 TEST EACH PASS
001E A5 85      PASS   LDAZ  BEGIN  ;SET POINTER TO START OF
0020 85 88             STAZ  POINTH ;TEST AREA
0022 A6 86             LDXZ  END
0024 A5 89             LDAZ  FLAG
0026 49 FF             EORIM $FF    ;REVERSE FLAG
0028 85 8B             STAZ  FLIP   ;=FF FIRST PASS, =00 SECOND PASS
002A 91 87      CLEAR  STAIY POINTL ;WRITE ABOVE FLIP VALUE
002C C8                INY          ;INTO ALL LOCATIONS
002D D0 FB             BNE   CLEAR
002F E6 88             INCZ  POINTH
0031 E4 88             CPXZ  POINTH
0033 B0 F5             BCS   CLEAR

                ;FLIP VALUE IN ALL LOCATIONS - NOW CHANGE 1 IN 3
0035 A6 8A             LDXZ  MOD
0037 A5 85             LDAZ  BEGIN
0039 85 88             STAZ  POINTH ;SET POINTER BACK TO START
003B A5 89      FILL   LDAZ  FLAG   ;CHANGE VALUE
003D CA         TOP    DEX
003E 10 04             BPL   SKIP   ;SKIP 2 OUT OF 3
0040 A2 02             LDXIM $02    ;RESTORE 3-COUNTER
0042 91 87             STAIY POINTL ;CHANGE 1 OUT OF 3
0044 C8         SKIP   INY
0045 D0 F6             BNE   TOP
0047 E6 88             INCZ  POINTH ;NEW PAGE
0049 A5 86             LDAZ  END    ;HAVE WE PASSED END OF
004B C5 88             CMPZ  POINTH ;TEST AREA?
004D B0 EC             BCS   FILL   ;NOPE, KEEP GOING
                ;MEMORY SET UP, NOW TEST IT
004F A5 85             LDAZ  BEGIN  ;SET POINTER BACK TO START
0051 85 88             STAZ  POINTH
0053 A6 8A             LDXZ  MOD    ;SET UP 3-COUNTER
0055 A5 8B      POP    LDAZ  FLIP   ;TEST FOR FLIP VALUE
0057 CA                DEX          ;2 OUT OF 3 TIMES
0058 10 04             BPL   SLIP   ;OR
005A A2 02             LDXIM $02    ;1 OUT OF 3
005C A5 89             LDAZ  FLAG   ;TEST FOR FLAG VALUE

005E D1 87      SLIP   CMPIY POINTL ;HERE'S THE TEST
0060 D0 15             BNE   OUTPUT ;BRANCH IF FAILED
0062 C8                INY
0063 D0 F0             BNE   POP
0065 E6 88             INCZ  POINTH
0067 A5 86             LDAZ  END
0069 C5 88             CMPZ  POINTH
006B B0 E8             BCS   POP
                ;ABOVE TEST OK - CHANGE AND REPEAT
006D C6 8A             DECZ  MOD    ;CHANGE 1 OUT OF 3 POSITIONS AND
006F 10 AD             BPL   PASS   ;DO NEXT THIRD
0071 A5 89             LDAZ  FLAG
0073 49 FF             EORIM $FF    ;INVERT FLAG FOR PASS TWO
0075 30 A1             BMI   BIGLP

0077 8C 1C A4   OUTPUT STY   ADDRL  ;SAVE LOW ORDER ADDRESS
007A A5 88             LDAZ  POINTH
007C 8D 1D A4          STA   ADDRH  ;SAVE HIGH ORDER ADDRESS
007F 20 13 EA          JSR   CRLOW
0082 4C 39 E8          JMP   MFAIL  ;DISPLAY MESSAGE AND ADDRESS
                ;AND RETURN TO AIM MONITOR

                ***** END PROGRAM  *****
```

KIMSI, S-100

MODIFICATION TO KIMSI TO ADD 4K OF RAM TO MEMORY SPACE BELOW MONITOR

by John R. Campbell
6278 Lake Lucerne Dr.
San Diego, Ca 92119

The KIMSI, as originally designed, allow addition of S-100 type interfaces to the KIM-1, but only in the address space from 2000 Hex and up. By making the following changes, 4K of RAM memory can be added to give a total of 4K from 0000 H to 13FF H, which is desireable to have.

1. The KIMSI is modified by cutting the trace between IC 8A-12 and IC 9B-3. A 7425 Dual 4 input NOR is added in the expansion area and is wired as shown. →

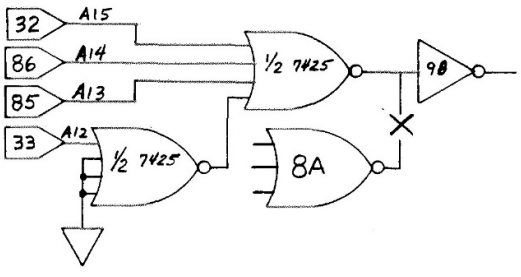
   This part of the modification enables the KIMSI and disables the KIM-1 for address space from 0000 H through OFFF H.

2. The second part of the modification moves the onboard KIM RAM from 0000 through 03FF to 1000 through 13FF. This is done by cutting the trace between IC U4-1 and IC U16-1. The proper place for cutting this trace is on top of the KIM-1 board near where the trace meets U16-1. On top of the board connect IC U16-1 to IC u4-5 and connect a 560 ohm resistor from IC U16-1 to Vcc (+5V) at the common ends of R36, 37, 38 and 35 (all 560 ohm).

3. Last, insert a 4K RAM board into the KIMSI with a starting address of 0000 H. Note that all 3 steps must be taken.

Incidently, the KIMSI diagram has an error: IC 1B-8 is connected to IC 11C-9 and IC9E-10 does not connect to IC 11C-9.

The KIMSI is manufactured by Forthought Products, Box 386, Coburg, OR 97401.



65XX chip family stuff

CPU BUG

by Heinz J. Schilling, DJ1XK
Im Gruen 15
D-7750 Konstanz 16
West Germany

This evening I was informed by Dr. Karl Meinzer (see BYTE 1/79: "IPS") that something seems to be wrong with the JMP Indirect instruction.

I have made some quick tests, and I must inform you that the JMP Indirect is indeed defective!

The MOS Programming Manual says (page 141, 9.8.1.):
"In the JMP Indirect instruction, the second and third byte of the instruction represent the indirect low and high bytes respectively of the memory location containing ADL. Once ADL is fetched, the program counter is incremented with the next memory location containing ADH."

But this is only correct al long as the location containing ADL is not the last byte of a page!

In this special case the incrementation works like a wrap around in the page as the handling of the carry seems to be processed incorrect.

The ADL is fetched from the last byte of the page, but ADH is fetched from the first byte of the same page instead of the first byte of the next page. This error occurs with CPU chips from MOS and from Synertec, it will be the same with Rockwell chips eventually.

So it is wise not to use the JMP Indirect instruction in the form of 6C FF xx.

6522 INFO & DATA SHEET CORRECTIONS

THE EDITOR

In issue #13 we presented a 6522 I/O board design. If you've looked over the 24 page 6522 spec sheet, you've probably commented on the complexity of the device.

While I was at MOS Technology, I had occasion to go through the spec sheet and confirm many of the chips operating modes. A number of typographical & operational errors were found and noted, (thanks to feedback from a number of sharp users). Things may make a little more sense after our discussion of the problem areas with the 6522 VIA chip and documentation.

page 3 - the peripheral B port is capable of sourcing 3.0 ma (not 30 ma).

page 13 - last sentence should read "Bit 7 will be read as a logic 2.

page 16 - section 4 should read-"If ACR5=0, T2 acts..."

page 24 - the delay time for T_{sr1}, T_{sr2}, and T_{sr3} should be 300 ns minimum and not 300 ns maximum.

page 10 - in mode 010, CB1 generates 9 clock pulses for controlling external devices. This is a serious bug in the chip.

page 10 - in mode 011, the shift register DOES stop the shifting operation after 8 bits have been shifted in. Reading or writing the shift register resets the Interrupt Flag and initializes the SR counter as well as re-starting the shifting action.

page 11 - in mode 101, CB2 remains at the state of the last bit shifted until a new bit is shifted out.

figure 11 - data becomes valid approx 1.5 usec following the negative transition of CB1.

figure 12 - output data is valid on the rising edge of CB1

page 12 - in mode 111, the SR counter sets the SR Interrupt flag each time it counts 8 pulses and DOES disable the shifting function.

Perhaps a little explanation on the 6522 timers is in order. They're different from the 6530 style in that they are full 16 bit counters as opposed to the 6530 style 8 bit counters with prescaling. This gives the 6522 timer the capability for much better resolution (to lus. with a 1 MHZ clock) over the entire range from 1 us. to 65,536 us. (65.5 milliseconds).

There are two timers in the 6522, each slightly different in its abilities. Timer 1 can handle normal 16 bit timer functions as well as operating in the "free running" mode, generating a square wave clock on the output of PB7 independent of any processor intervention. Handy for test signals around the workbench as well as for clocking peripheral devices such as A/D's etc. Timer 2 can operate as a pulse counter where it keeps track of negative going pulses coming in on the PB6 line as well as the normal "one shot" interval timer mode.

The shift register is probably the most misunderstood function in the VIA. This 8 bit synchronsus serial port was designed to facilitate inter-system communications, not as a "normal" asynchronsus werial I/O port. The serious bug in the shift-register (mentioned previously) makes this function even less useful. There are, however, other uses for the shift register. How about clock or music generation? I did think about using this shift register as the main element in a mini-floppy interface but gave up the idea after an investigation of the timing requirements of the floppy.

more next time.

EXTENDING THE RANGE OF KIM-1 TIMER TO 1:32640

by Cass Lewart

many systems based on the 6502 microprocessor e.g. the popular KIM-1, contain one or more firmware timers. When a value K is stored in a specific location, the timer starts a countdown lasting K time periods P, where P can assume 1, 8, 64, or 1024 usec depending on the time location chosen. A typical program using the firmware timer would look as follows:

```
A2 XX            LDX, K
8E 06 17  START  STX  TIMER  start timer
2C 07 17  CHECK  BIT  TIMOUT check if timer
                             finished
10 FB            BPL  CHECK  if not, check
                             again
```

With K assuming values between 0 - FFhex, the range of the timer is 1:256 (K=0 results in a countdown of FFhex + 1). This timing range may be inadequate for some applications and can be extended to 1:32640 by simply adding two statements at the end of the previous program:

```
CA       DEX
D0 F5    BNE START
```

The number of time intervals will be now:

$(K+1)\ \frac{K}{2}$ $\qquad 0 < K < FFhex$

$(FFhex+2)\ \frac{(FFhex+1)}{2}$ $\qquad K = 0$

The following table shows the delay introduced by the timer program for selected values of K. These figures do not include the overhead caused by the testing and looping instructions.

KIM-1 TIMER LOCATION INTERVAL P	1704 1us	1705 8us	1706 64us	1707 1024us
K (HEX)				
01	1us	8us	64us	1024us
10	136us	1.09ms	8.7ms	13.9ms
20	528us	4.22ms	33.8ms	541ms
40	2.08ms	16.6ms	133ms	2.13sec
60	4.66ms	37.2ms	298ms	4.77sec
80	8.26ms	66ms	528ms	8.45sec
A0	12.9ms	103ms	824ms	13.2sec
C0	18.5ms	148ms	1.19sec	19.0sec
00	32.9ms	263ms	2.11sec	33.7sec

SYM AND AIM TIMER LOCATIONS

by Marvin L. De Jong
School of the Ozarks
Point Lookout, Mo 65726

Enclosed find a short table that may be of some use to SYM-1 and AIM 65 owners. Both the SYM-1 and AIM 65 have 6532 chips which in turn have interval timers that are almost indentical to the timers on the KIM-1. In fact, in many programs written for the KIM-1, one can merely substitute the address given in the table if he is using an AIM 65 or SYM-1.

If the program involving a KIM-1 timer is using the interrupt mode, that is, PB7 is connected to the IRQ line or the NMI line, then SYM-1 users are out of luck as far as using the 6532 is concerned. Perhaps they could jumper a lead from the IRQ pin on the 6532 to the IRQ pin on the 6502, but I am certainly not recommending that without a SYM-1 with which to experiment. The AIM 65 people are still in luck, for the 6532 interrupt is connected on board to the IRQ pin of the 6502. So AIM 65 users can make use of all the KIM-1 programs that use interval timers by substituting the addresses shown in the table.

Of course AIM 65 and SYM-1 users can rewrite any timer routine using the 6522 chips that both these systems include.

TIMER	KIM-1 ADR.	AIM 65 ADR.	SYM-1 ADR.
T0001	$1704*	$A494*	$A41C**
T0008	$1705	$A495	$A41D
T0064	$1706	$A496	$A41E
T1024	$1707	$A497	$A41F
READ STATUS	$1707	$A497	$A407
READ TIME	$1706	$A486	$A406

*Add eight (in hexadecimal) to the address to enable the interrupt feature on the KIM-1 and AIM 65.

**The interrupt line on the SYM-1 is not connected.

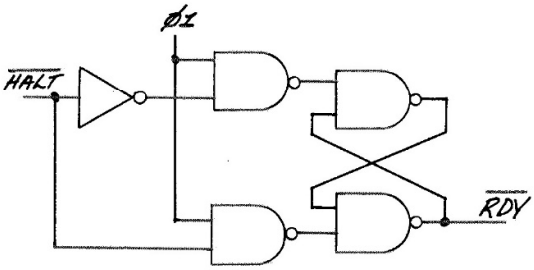
USE OF THE RDY LINE TO HALT THE PROCESSOR

by Conrad Boisvert
Applications Manager
Synertek

The RDY line, available on the expansion connector, is used to halt the processor. This line is normally high and is driven to the low state in order to halt, and then driven high again to restart.

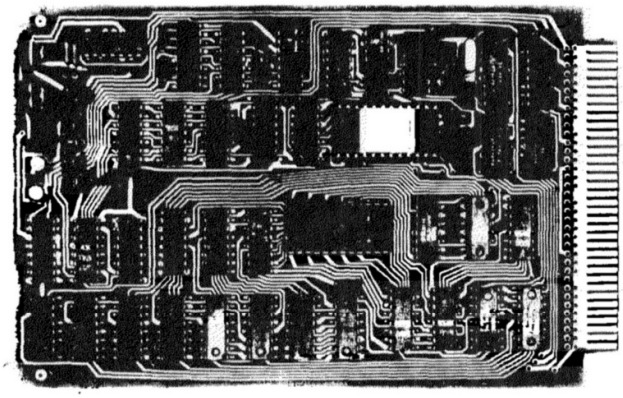
The timing of the RDY line transition must not be random, relative to the processor clock. If it is, then the processor will occasionally fail to re-start. To solve this problem, it is necessary to time the RDY line transitions so that they occur during Ø1 timing, only.

The following circuit can be used to accomplish this:



In this circuit, HALT is the low-going signal indicating that the processor is to be stopped.

MVM 1024 MICROPROCESSOR VIDEO MODULE



KIM-1 GOES VIDEO

$235.00

The MVM-1024 is a video display that departs from the usual DMA page memory structure. Two on-board bi-directional ports hold the cursor position, eliminating the need to use microprocessor registers to form a memory pointer and external RAM to save the cursor position. The cursor display is a blinking over- and under-line. Reverse video characters can be generated independent of cursor function.

The MVM-1024 is ideal as a parallel access display. The upper / lower case capability, together with its unique organization make it a natural for text editing applications. The board uses more expensive low-power Schottky logic and low-power memory. Designed for no specific microprocessor, the interface can be adapted to any available microprocessor. It can support separate IN and OUT data buses, or a single bi-directional data bus.

JOHNSON COMPUTER

P.O. BOX 523 MEDINA, OHIO 44256

(216) 725-4560 OR 725-4568

comments...

COMMENTS FROM

by Les Jacobson
3841 Fetlock Cir.
Colorado Springs Co 80918

I have some information which I would like to have you pass on to the other readers of USER NOTES. It may keep them from repeating some of the mistakes which I have made.

First, DESPITE the full page ads in October and November issues of BYTE, Commodore is NOT able to supply the KIM-3B, nor the KIM-4, nor the KIM-5 nor the KIM-6. During telephone conversations with their Marketing dept., in December, I was advised that these items would be available after the first quarter of 1979. This (again) despite their ad's statements that ALL of these were available for immediate shipment.

Today (March 13th) I phoned them again. This time I learned that Commodore has decided not to construct and offer EITHER NOW OR IN THE FORSEEABLE FUTURE any of the above boards. Further inquiry lead me to locate probably the last remaining KIM-6 in the U.S. Falk-Baker Co. in Nutley, N.J. (201) 661-2430, has a limited supply of the KIM-4 motherboards. So does the NCE/CompuMart in Ann Arbor, Mich. (800) 521-1534. In fact, NCE is discounting their remaining KIM-4s and the very few KIM-3Bs that they still have by almost 30%.

My intent is to attempt to locate a manufacturer who can duplicate the KIM-6 so that I can prototype additional memory. With the recent prices for good 2114s having dropped, I believe that 16K of memory should be buildable for less than $200.

It appears that the KIM is not the only thing which Commodore is not supporting. I'm a Senior Software Systems Engineer with the Aerospace division of Ford. My primary work is for the Dept. of Defense but I interface with other government agencies. The other day I visited the National Bureau of Standards in Boulder. Since my interests are software I was immediately involved with their latest experiments and applications. In support of one of their projects, NBS purchased 29 PET computers. Their problem, I learned, was that NOT ONE of these units functioned as advertised. Commodore had been called to correct the problem and hadn't bothered to extend the courtesy of a response. NBS engineers told me that the problems were shoddy workmanship, poor printed circuit board construction, and the use of many sub-standard chips. I was told that NBS had stopped payment on the purchase and was preparing to return all of the units to Commodore.

Interestingly enough, the owner of our local Computerland is returning his entire consignment of PETs to Commodore for the identical reason.

I support your search for a RPN calculator chip interface for the KIM. RPN is precicely the concept utilized by the large scale machines because of the inherent efficiency. It may mean dedicating up to 4 memory locations to retain precision, but that appears to be trivial.

For anyone still wanting to go "glass" TTY instead of the clanking monster, SWTP makes a very nice unit for almost exactly the same price as a working TTY. And it has many more features than the Teletype does. In addition it displays the more reasonable 24 by 80 format which is far more useable unless you don't care that your computer only plays games.

I promise not to write often. But I will attempt to keep you posted on my successes and failures.

By the way, Micropolis has impressed me the most of any company with their disk configuration and reliability. Has anyone successfully interfaced their hardware to the KIM bus? If so, how about letting the rest of us in on it.

Keep up the good work, and thanks for warning the others of the holes in the KIM path.

ALTERNATE SOURCE FOR OSI BOARDS

by Robert F. Solomon
5868 JoAnne Court
North Ridgeville, Ohio 44039

GREAT NEWS FOR OSI OWNERS! As most owners of OSI computers know, delivery on OSI boards range from two weeks to infinity; with emphasis on the latter. While attempting to locate a bare OSI 420 board, I found out that D&N Micro Products, 3932 Oakhust Dr., Fort Wayne, In 46815 made OSI compatible boards. I called them and learned they made an 8K RAM board in kit form at a reasonable price. I promptly sent them a money order and received the kit within 5 days. After assembling it and tracing a shorted foil (my fault, not theirs) it worked beautifully. They also make a Real Time clock and a proto board; with a couple more boards just going into production. I am well pleased with their RAM board and love their delivery.

Before I go into my present activities, I would like to explain my system. (Mainly to show that I am not working with a super-sophisticated system, but more on the level of what I consider the system of the average tinkerer. I have an OSI system with 16K,video board and cassette interface. It is based on the 404V board. I have an RO-15 Teletype for hard copy. The entire system has been built from kits. My major accomplishment has been to interface the computer to an electronic organ. This is not synthesized music but a case where the computer actually plays the organ.

Most of my programming has been in machine language and Tiny Basic. I am now getting Focal up and running. As of right now, Focal appears to be working, but as yet I have not exercised all of the functions to make sure everything works. I hand loaded the entire program from the KIM based listing of the Aresco version 3D. I have all the patches made to make it operate on the OSI.

I am planning to summarize these patches and submit an article to NOTES for publication. At present, I am starting work on a PLL FSK interface to operate with the OSI 430 I/O system to up my cassette records to 1000 baud. I plan on publishing this also.

I think that is enough rambling for now, I hope my contributions and thoughts will encourage other OSI users to get on the NOTES bandwagon and share their experiences.

(EDITORS NOTE -
Bob has just sent in an article on adapting a KIM 6530-003 to the OSI system for the purpose of reading KIM compatible cassettes. That article will appear in #16. Thanks Bob!)

music

ADDITIONS TO THE MTU MUSIC SOFTWARE PACKAGE

by Bruce Nazarian
WD8DRK

As promised, here are a few changes you should try to make to Hal Chamberlain's DAC software. I sent these to him and he told me he liked them and would be using them in the demo ROM for the DAC system. I guess being a musician has its advantages!! So, here they are, and you may wish to put these in the User Notes as well.

PROGRAM CHANGES FOR KIM4V
(For use with the MTU DAC music system)

THESE CHANGES WILL CORRECT AN ERROR IN ASSEMBLY

ADDRESS	SHOULD READ:
1788	3E, not 2E
178D	3E, not 2E
1792	3E, not 2E
1797	3E, not 2E
179C	3E, not 2E
17A1	3E, not 2E
17A6	3C, not 2C
17AB	3C, not 2C

THESE CHANGES ARE SUBJECTIVE MUSICAL CHANGES IN CONTENT...You may like them, and then again, you may not. Your ears will tell you yes or no. hi.

ADDRESS	SUBSTITUTE	ADDRESS	SUBSTITUTE
179F	24 for 1E	0088	06 for 14
17A9	24 for 1E	0092	10 for 1E
17B8	14 for 22	009C	16 for 24
17BE	30 for 2C	0204	14 for 22
17C2	2C for 22	0209	1A for 28
17C3	32 for 2C	020E	1A for 28
17D1	10 for 1E	0234	1E for 24
17D6	10 for 1E	025C	1E for 24
17DB	14 for 22	0298	1E for 24
		029D	22 for 24

Also the following:

```
02C9   18 2C 2C 44 44
02C3   18 36 36 4E 4E
02D3   18 32 36 4E 52
02D8   48 30 3C 4E 5C
02DD   18 40 00 00 58
02E2   24 3A 00 00 52
02E7   0C 36 00 00 4E
02EC   30 2C 44 4C 52
```

THESE CHANGES ARE FOR THE "EXODUS" SONG TABLE IN THE MTU ADVANCED MUSIC SOFTWARE PACKAGE, AVAILABLE FOR THE MTU DAC MUSIC SYSTEM. (K-1002)

I don't know if you have seen the article in MICRO, Ish 2 which attempt to explain how to use the DAC music system..It is a good piece of writing, well aimed at people who do not know that much about the semantics of musical part-writing, but ole Armand (Camus, the author) made a few good old-fashioned boo-boos in there...He states address 001D will change the TEMPO of the tune--well, maybe in his software it is, but in the listing I have, the TEMPO byte is location 0016. Also, he states that the execution point may be changed from the beginning ($0200) to another point as long as you start out with a correct duration byte. Correct, but the addresses are not 0017 & 0018 in my listing...the starting address should be in locations 0014 (SAL) and 0015 (SAH). The little chart he has made regarding the available memory locations and their use in the song tables is right on the money!! Hope you haven't been confused by this. I really was for a few minutes until I dug back in the lisings I had. Maybe there is a difference in the software that was made available with the DAC that Tripp was selling and the one that MTU is doing on their own?

interface

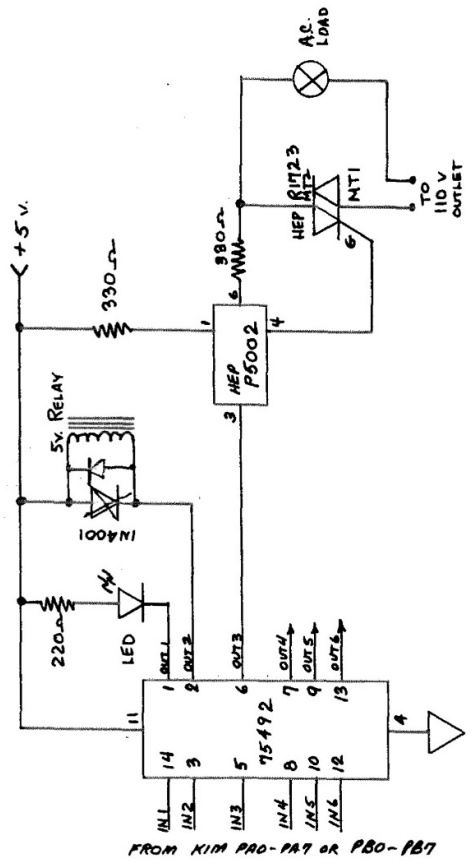
A SIMPLE MICROPROCESSOR INTERFACE CIRCUIT

by Cass R. Lewart

The following simple and inexpensive interface circuit will let KIM control LEDs, relays or AC operated appliances.

The computer ports are directly connected to inputs of SN75492. This is a popular MOS/LED driver IC, described by me in earlier issues of KUN, which can sink up to 200 mA on each of its six outputs. A typical use for this IC is as digit driver in multiplexed LED calculator displays. It can also be used to drive individual LEDs, relays or optocouplers. To calculate the value of load resistors it should be remembered that the voltage drop between any output and ground of SN75492 is 700 mV. The HEP P5002 or Motorola MOC3010/11 is an optocoupler interfacing an infrared emitting diode to a low power Triac. The low power Triac in the optocoupler in turn controls a larger Triac e.g. HEP R1723 to turn on and off AC appliances, motors, heaters, etc.

If more than 6 ports of a computer are being used for control, additional SN75492s can be installed. The same port can also drive more than one output e.g. an AC load via an optocoupler and an LED as activity indicator.



KIM SOFTWARE ON CASSETTE

We know that you have better things to do with your time than punching hex code into your machine. Because of this, we have made some of the longer programs available on KIM cassette.

These cassettes are original dumps, not copies, made with top quality 5-screw housing cassettes in the standard KIM tape speed. Thirty seconds of sync characters precede the program to enable you to tune up your recorder or PLL.

Are you AIM & SYM owners interested in having some of these programs available for your machines?

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

OUR PRESENT OFFERINGS INCLUDE:

KIMATH (specify $2000 or $F800 version).....$12.00
(includes errata sheet for manual)

HEXPAWN (from issue #13).....$5.00

DISASSEMBLER (from issue #14).....$5.00

BANNER (from issue #14).....$5.00

PAYMENT MUST BE IN U.S. FUNDS

OVERSEAS CUSTOMERS PLEASE INCLUDE $1.00 EXTRA PER CASSETTE FOR EXTRA POSTAGE.

PET 8K	$ 695
PET 16K NEW Full-size Keyboard	$ 890
PET 32K NEW Full-size Keyboard	$ 1065
PET Dual Disk (343,000 bytes online)	$ 1160
PET Printers (May 1979 availability)	
2021 Electrostatic	$ 495
2022 Tractor Feed	$ 890
2023 Pressure Feed	$ 760
KIM-1	$ 159
SYM-1	$ 238
KL-512 Power Supply for KIM, SYM, and extra RAM	$ 34
Memory Plus	$ 238
KIM-4 Motherboard	$ 95
Synertek ROM BASIC	$ 139
Synertek KTM-2 Keyboard Terminal	$ 290
Problem Solver KM8B KIM RAM	$ 149
SEA-16 -- NEW 16K Static RAM Uses new MOS Tech. very low power 2114 (1.35 amp/16K). For SYM, AIM, KIM	$ 325
Seawell Buffered Motherboard Assembled, with space for 4K RAM. For SYM, AIM, KIM	$ 99
Other Seawell products available soon.	

CASSETTE TAPES

Premium quality, low noise, in 5 screw housing. Includes labels. -- All tapes guaranteed --

C-10	10/5.95	50/25.00	100/48.00
C-30	10/7.00	50/30.00	100/57.00

Norelco-style hard cassette boxes	10/1.25
Soft poly cassette boxes	10/1.00
2716 (Intel) or 2516 (TI) +5V EPROM	$ 45
2114 L NEW low power MOS Technology	$ 6.95
6550 RAM (for Commodore PET)	$ 16.20
6522 VIA or 6520 PIA	$ 10.50
6500 Programming Manual (MOS)	$ 6.50
6500 Hardware Manual (MOS)	$ 6.50
First Book of KIM	$ 8.95
Programming a µComputer: 6502 Foster	$ 8.95
Programming the 6502 R. Zaks	$ 9.95
KIM Microchess (Jennings)	$ 13.00
PET Microchess (Jennings)	$ 17.95

Write for: 6502 and S-100 product list
PET Software List

A B Computers 215-699-8386

115 E. Stump Rd. Montgomeryville, PA 18936

6502 CONSULTING SERVICE

HAVE COMPUTER/WILL CONSULT

CALL ERIC (216) 237-0755

"6502 HARDWARE AND SOFTWARE DESIGN EXPERIENCE"

REVIEWS ETC.

REVIEW: Programming the 6502, by Rodnay Zaks
(SYBEX, 305 pp.)

review by Jim Butterfield

The 650X community is in need of good reference and/or tutorial books on their chip. Unfortunately, this book doesn't make the grade.

There are too many mistakes and oversights in the book to make it serve as a useful reference or teaching guide. Some of the problems are relatively minor goofs that may be corrected in a future edition: for example, page 15 notes that binary 10000000 equals a value of minus zero (!), and page 181 says (twice!) that the BIT instruction uses relative addressing.

More seriously, there seems to be a lack in the author's depth of understanding. Exercise 3.17 asks the reader, "Why is the return from a subroutine so much faster than the call?" Why indeed? The 6502's JSR (Jump Subroutine) and RTS (return from subroutine) in fact have identical execution speeds. On the same subject, Zaks suggests that a handy way for a calling program to pass parameters to a subroutine is through the stack. He doesn't mention the formidable coding problems that this creates.

Zaks doesn't seem to realize the important difference in indexing behaviour between zero-page and absolute modes, namely that zero-page indexing can be used to achieve a negative index value. Anyway, he doesn't mention it; indeed, he makes little mention of zero-page indexing except to state that only the X register can be used as an index (which is, once again, wrong).

The list of problems goes on. Several examples are incorrect, and on at least one occasion, insult is added to injury by having an explanation of how the incorrect code works.

Perhaps the biggest problem is that Zaks doesn't seem to like the 6502. His tutorial style is to outline features he thinks "good" processors should have, and then conclude that the 6502 has a poor capability in that area. The word, "unfortunately", occurs over and over again indescribing the 6502: Unfortunately, it doesn't have both ADD and ADC; unfortunately it can't test bits in sequence (whatever that is); unfortunately, the 6502 has very few internal registers; unfortunately, only the A register can be shifted...the list goes on.

It reaches a climax on page 182 where Zaks first details indirect addressing on the 6502. He does this with seven sentences criticizing the way it's done. This is followed by, "In fairness, it should be noted that few microprocessors provide any indirect addressing at all."

Faint praise indeed for one of industry's biggest-selling microprocessors. A beginner reading this book might wonder whether he's made a mistake in opting for the 6502. Nowhere does the book mention the chip's speed and versatility.

Does the book have anything going for it? It covers the instruction set quite well, with addressing modes outline somewhat patchily. Many of the coding examples are well set out and explained. Interrupts are dealt with in a rather rough manner, and support chips are passed over briefly. There are eight pages of good appendices, and a thorough index.

It's still hard to find material dealing with the 6502. If you're desperate, this book will be of some help.

PRODUCT REVIEW

THE SEAWELL MARKETING 16K RAM BOARD

The SEA-16 is a KIM-4 compatible 16K Static RAM card from SEAWELL MARKETING, 315 N.W. 85th, Seattle, WA 98117 (206) 782-9480.

The card has been designed to fit in the standard KIM-4 backplane and cannot be used in the new HDE motherboard. The SEA-16 is a really nicely done board with solder-masking on both sides and labeling of all I.C.'s and DIP switches.

All of the 32 RAMs were socketed with low-profile Augat sockets (the good ones) which seemed indicative of the overall high quality of workmanship involved here.

Unfortunately, the documentation that accompanied this otherwise nicely done board consisted of a copy of the schematic and nothing else. I was left to decide for myself which way the write enable and bank select switches should be positioned for proper operation. Also, one of the RAMs failed almost immediately which indicated that this board had not been burned in at all.

In a phone conversation with Seawell Marketing shortly thereafter I was assured that this board had somehow "sneaked" past the usual burn-in procedure. It was further stated that the regular documentation package had just been printed up and I would receive it along with a replacement 2114 very shortly.

That was over a month ago and I still haven't received anything.

Seawell Marketing has done an otherwise first class job on this $325 RAM board except for the two points that I mentioned. Maybe they'll have gotten their act together by the time you read this.

ERIC

NEW PRODUCT

SPEAK & SPELL (TM) INTERFACE KIT

If you were wondering whether or not the new Texas Instruments' SPEAK & SPELL learning aid could be hooked up to a computer-wonder no longer! For apparently it already has been done.

After following up on an ad that was placed in ON-LINE*, I found out that Dave Kemp of East Coast Micro Products (1307 Beltram Ct, Odenton, MD 21113) is offering the SP-1, a bidirectional interface to the Speak & Spell for $49.00.

According to the flyer, "It (the SP-1) allows the computer to read speech data as it is being fetched from onboard ROM by the synthesizer, and it allows the computer to transfer data directly to the synthesizer to produce computer generated speech or sound effects."

I hope to be reviewing the SP-1 in an upcoming issue. It's really exciting to consider the possibilities of an under $100 digital speech synthesizer interface.

According to the information I received, the SP-1 will interface to a 6522 and includes some 6502 driver software (SYM).

*ON_LINE is a computer classified ad newsletter. For more info, contact Dave Beetle, publisher, 24695 Santa Cruz Hwy., Los Gatos, CA 95030

until next time .....