u-Panel

KIM users who don't have X-ray vision will find the following article useful.

George E. Lang
204 Sweetbriar St.
Pittsburgh PA 15211

Can you see the reaction of every register in your microprocessor as you are single-stepping through the program? If so, skip this article; if not, read on.

On my KIM-1 microprocessor, all the registers are internal to the chip and are not directly visible. This is a problem common to most other chips and systems as well. There is a means for "looking at" all registers in single-step mode, but after many hours of doing this, I decided to let "constructive laziness" take over and to make a front panel showing all registers. This display I call the *u-Panel.*

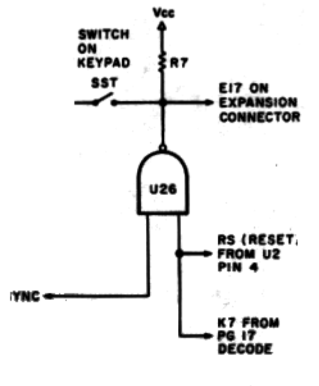


Fig. 1. Within the KIM, a signal is generated at the output of U26 with each instruction execution, and is made available at pin 17 of the expansion connector.

My criteria for the panel are as follows.

1. No object program modification.
2. Cost (as inexpensive as possible).
3. No physical modifications to KIM.
4. Simple to load.
5. Easy to connect.
6. Looks like a front panel.

KIM Single-Step Operation

In the KIM monitor SST (single-step) mode, the monitor is displaying the current address and the current contents (data or instruction). When the GO button is pressed, the monitor is exited, the instruction is executed and, at the same time as execution, the NMI (non-maskable interrupt) line is pulled low thus enabling the interrupt. The interrupt is not executed, however, until the instruction currently in the processor is finished.

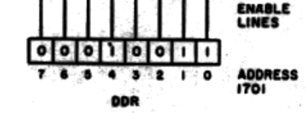


Fig. 2. The data direction register and I/O register shown being set up for saving all the KIM registers.

When the instruction "execute" cycle is done, the KIM looks at the NMI vector stored at 17FA-17FB (hex), the contents of which are normally 1C00 (hex), and loads these contents into the program counter. The interrupt is now serviced. During all this action, the program counter and stack pointer are pushed onto the stack.

When the interrupt program is executed, the stored registers are first pulled from the stack and stored in zero page locations, along with the accumulator and the X and Y index registers. These locations are the ones "looked at" during KIM SST operations. The monitor program, which was entered through the interrupt, then takes over the job of displaying the data and address, scanning the keys, etc. When the GO key is pressed again, an RTI (return from interrupt) is executed, and exit from monitor occurs; the whole process begins again.

Looking at the schematic shown in Fig. 1, I saw that a signal is generated on the output of U26 every time an instruction is executed. This signal combines the SYNC signal and an enable from the ROM chip and is fed into an N-O (normally open) switch (the SST). It also runs to E17 on the expansion edge connector. That signal is then used to trigger (with the SST switch closed) the NMI response previously described.

I decided to use the IRQ (interrupt request), which has a vector at 17FE-17FF (hex), and an external N-O switch was connected between E4 (IRQ input) and E17 (output of U26) on the expansion connector. With the switch open, the program runs normally without interruption. With the switch closed, and upon an "execute" by the microprocessor (the GO button being pressed), the IRQ vector is loaded. Therefore, I can place my SST routine starting address into the IRQ vector and will execute the routine every time an IRQ is generated.

The program to enable the SST and output the registers resides on page 1 (0100-01FF hex). There are two reasons for this:

1. The stack sits on page 1, and I prefer that if anything is going to get overwritten due to stack overflow it be the short SST and not the long object program.
2. Page 0 is used for data. That's why there is zero page addressing.

There are two pages left—page 2 (0200-02FF hex) and page 3 (0300-03FF hex). If the user's program is to use continuous memory, the only place left for user program is starting on page 2.

Now, since I want to dump all of the registers to an external device before returning back to

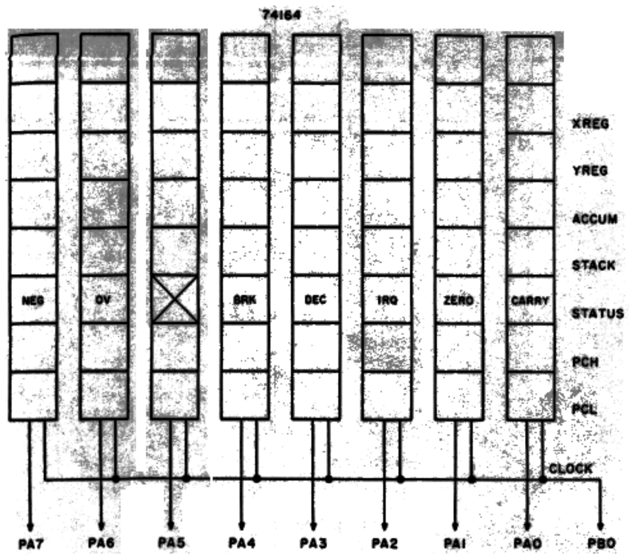


Fig. 3. KIM registers.

the monitor, I can't use the NMI interrupt. This is one of the reasons I selected the IRQ interrupt. Also, now that the IRQ is being used, the routine in ROM that saves all the registers and inserts them in zero page locations is missing. Therefore, this must be the first part of the new SST routine.

The I/O (input-output) ports on the KIM must be preselected by first setting the appropriate DDR (data direction register) bits. The DDR's bits are in direct correspondence to the I/O register bits. That is, if you wanted the least significant bit of the I/O port to be an input and the rest of the seven bits to be outputs, the word 01 (hex) 0000 0001 (binary) would be inserted into the DDR. If I wanted an 8-bit output, the DDR would be set to FF (hex) 1111 1111 (binary). In Fig. 2 the DDR has been loaded with 13 (hex) 0001 0011 (binary). Note how the I/O register is set.

Looking now at Fig. 3, you will see that the registers to be output are as follows:

1. Accumulator
2. X index register
3. Y index register
4. Status register
5. Stack pointer
6. PC high (most significant bits of the program counter)
7. PC low (least significant bits of the program counter)

These registers are found in the storage locations down on page 0. They are consecutively loaded from 00EF-00F5 (hex) in this sequence:

00EF	PC lo
00F0	PC hi
00F1	Status
00F2	Stack
00F3	Accumulator
00F4	Y index
00F5	X index

The Hardware

The front of the *u-Panel* is shown in the accompanying photo. The accumulator is at the top, followed by the X and Y registers. The program counter is the long one in the middle, with the status register at the lower left and the stack pointer at lower right.

As you can now see from Fig. 3, I am loading eight bits of information (the register) seven levels deep. Each shift register is a 74164, and the serial inputs of each are connected to a specific corresponding bit of the I/O port, PA7-PA0. The clock inputs of the 74164s are connected in parallel, and a pulsed signal is applied from a pin of the other I/O port (PB0).

With seven synchronizing pulses from PB0, the shift registers will shift any data applied to the serial inputs down the line, thus loading all the information sequentially applied from the I/O ports into the seven-level-deep output pins of the shift register. These output pins are then connected to 7406 inverting drivers to provide the necessary current to light the front panel LEDs. See Fig. 4.

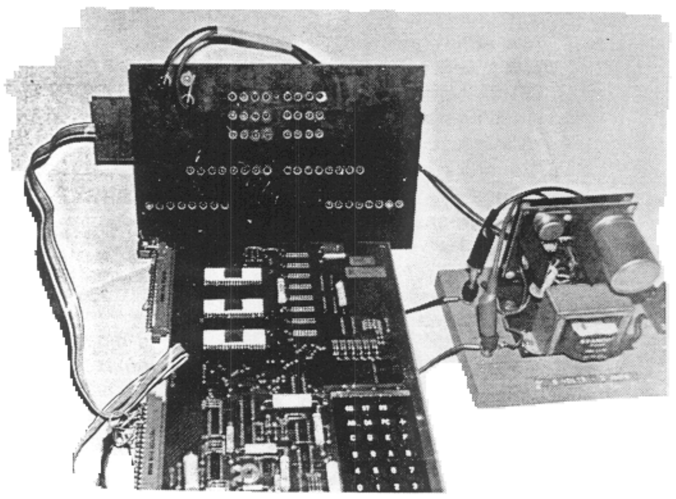
The Software

Turning now to the SST program listing, you can see the first part is the packing segment to place all register information into zero page memory locations 00FE-00F5 (hex). The next three parts are for setting the DDRs of the two I/O ports and for pulling PB0 low. The next part is to take the information (data) sequentially from the zero page locations, place the data on the I/O pins and pull PB0 high to shift the shift registers up one place.

Next comes a check to see if all locations have been loaded. If not, go back, increment to the next location, output to the 74164s, toggle PB0 and repeat until all seven levels are full. When all of the levels have been filled, exit from the SST program and enter the KIM ROM monitor at location 1C4F (hex), which is called BEGIN.

With all shift registers filled with KIM register data, and the KIM monitor in a loop awaiting another GO command, you can now observe all of the registers on the front of the *u-Panel*. This information will stay until pushed out by your pressing the GO button and executing another instruction.

The front layout of the *u-Panel* is shown in the photo. It can be modified to suit. I have placed an edge connector and three female banana jacks to interface with the KIM merely because I had them around. A single larger edge connector



Completed front panel connected to, and working with, KIM.

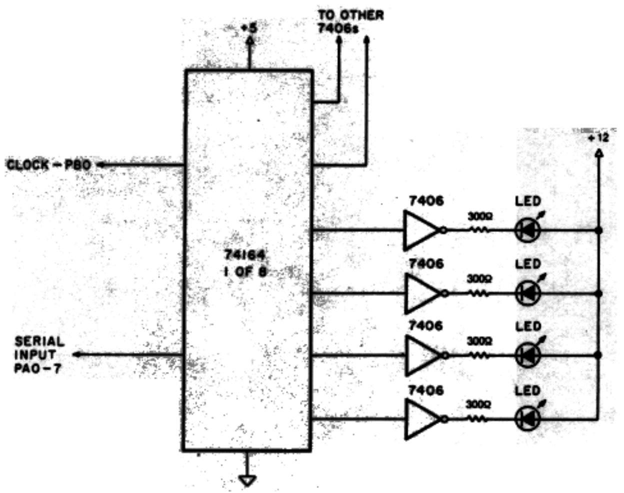


Fig. 4. Front panel LED interface circuit.

```
0130  STA  85
0131  F3
  32  PLA  68
  33  STA  85
  34  F1
  35  PLA  68
  36  STA  85
  37  EF         Loads 00EF-00F5
  38  STA  85    on IRQ escape.
  39  FA
  3A  PLA  68
  3B  STA  85
  3C  F0
  3D  STA  85
  3E  FB
  3F  STY  84
  40  F4
  41  STX  86
  42  F5
  43  TSX  BA
  44  STX  86
  45  F2
--------------------------------------------------
  46  LDA  A9
  47  #01        Sets PB0 to output
  48  STA  8D
  49  03
  4A  17
--------------------------------------------------
  4B  LDA  A9
  4C  #00
  4D  STA  8D    Stores 0 in PB0
  4E  02
  4F  17
--------------------------------------------------
  50  LDA  A9
  51  #FF        Sets PA0-PA7 as output
  52  STA  8D
  53  01
  54  17
--------------------------------------------------
  55  LDX  A2    # of locations to output
  56  #07
  57  LDA  B5<-┐ Start at F5
  58  EE       │
  59  STA  8D  │ Output F5
  5A  00       │
  5B  17       │
015C  LDA  A9  │ Set PB0 high
  5D  #01      │
  5E  STA  8D  │ Load PA0-PA7 into shift register
  5F  02       │
  60  17       │
---------------│----------------------------------
  61  NOP  EA  │
  62  NOP  EA  │ Settling time
  63  NOP  EA  │
---------------│----------------------------------
  64  LDA  A9  │
  65  #00      │
  66  STA  8D  │ Set PB0 low
  67  02       │
  68  17       │
---------------│----------------------------------
  69  NOP  EA  │
  6A  NOP  EA  │ Settling time
  6B  NOP  EA  │
---------------│----------------------------------
  6C  DEX  CA  │ Set X to (last reg.) - 1
  6D  BNE  D0--┘ If X not 0, go back and
  6E  E8         output register.
  6F  JMP  4C    If X zero, jump to 'START'
  70  4F         in KIM ROM listings.
  71  1C
--------------------------------------------------
NMI = 17FA - 17FB = 1C00
RST = 17FC - 17FD = 1C00   VECTORS
IRQ = 17FE - 17FF = 0130
--------------------------------------------------
Note: All programs must have CLI (clear interrupt)
in program initialization.
--------------------------------------------------
      EXPANSION CONNECTOR
    E4          E17
    ↑            ↑
    SINGLE STEP SWITCH
    └───◄──2──┘
```

Program listing.

would have been better. There is no critical wiring with the possible exception of the current requirements for the 56 LEDs. You must use adequate-sized wire.

The ICs were cemented directly to the phenolic board, and wire-wrap sockets were inserted. I like wire wrap because there is no possibility of heat burnout of chips, and (for me) it is faster than soldering. If you do not want to use wire wrap, careful soldering will do as well. The LEDs were also cemented to the board. The whole building and testing procedure took about six hours.

Operation

Once the whole thing is built and tested, load the SST program starting at 0130 (hex) and put the user program anywhere there is spare memory. Again 0200 (hex) is recommended. The actual location is not important as far as the SST routines are concerned. To test a program under SST, place the added SST switch ON (closed), set the IRQ vector 17FE-17FF (hex) to 0130 (hex) instead of 1C00 (hex) and load your program.

Whenever the GO key on the KIM is pressed, all registers will be seen on the *u-Panel*, and the next instruction and address will be seen on the KIM LED displays. Leave the KIM SST switch OFF as it will not be needed. One last item—at the beginning of your program, place a CLI (clear interrupt) instruction. This will ensure th
you will start in the right mo

I placed the SST program
cassette tape to facilitate lo
ing and reduce loading time.
loading the SST on tape a
"plugging in" the *u-Panel*, I c
be debugging programs
about two minutes. This is
reasonable time savings o
hand-loading the SST progra

Summary

Now look back to my obje
tives stated at the beginning
this article. Did I achieve ther

1. No program modificati
. . . I think 99 percent on th
one. The only addition is t
CLI instruction.

2. Cost . . . with a litt
scrounging, you can duplica
this unit for $25 or less. (A co
plete parts list is shown
Table 1.)

3. No physical modificatio
. . . 100 percent on this one.

4. Simple to load . . . wi
only 64 program steps and th
availability of cassette, it
simple.

5. Ease of connection . . . or
edge connector and two pow
supply connections (both c
which are on the KIM anyway

6. Looks like a front panel . .
I think it does!

I have built the *u-Panel* to b
used by students in a clas
room-laboratory situation fo
instructional purposes. I fe
that you, as well as m
students and I, will benefit b
seeing all registers at once.
hope you get as much out c
u-Panel as I have. ■

56	300Ω resistors
8	74164 Shift registers
10	7406 Inverting drivers—open collector
56	LEDs. If you wish to have different colors for the registers then order: 8 each accumulator, X and Y registers, stack pointer and status register. (different colors) Total 40; 16 for program counter; Grand total 56
18	Sockets, 14-pin DIP
1	Phenolic or Bakelite board for front panel
1	15-pin connector to interface with KIM
2	44-pin edge connectors for KIM expansion and application connectors. Do this only if you want to have separate connectors for the *u-Panel*, otherwise order 1.
1	SPST (single pole-single throw) switch
	Hookup wire
	Glue

Table 1. Parts list.