FROM THE FACTORY: Problems with some 6502 chips.

I have been informed of two possible problem areas with the 6502 CPU. Some of the early CPU's didn't set the zero flag correctly after register to register transfers,(TAY, TAX,TYA etc This would make it impossible to run Tiny Basic. All 6502 CPUs could have problems setting the zero flag after decimal addition when the answer equalled zero. Test your CPU with a simple program to see if you have these problems. The decimal addition problem can be gotten around by an "OR" immediate with "00" after the operation which would then set the zero flag correctly. The only method of solving the register transfer not setting zero flag would be to install a new CPU. (You would then pick up the "ROR" rotate right instruction in the process).

Copying KIM cassette tapes:

If you've tried copying KIM tapes from cassette to cassette then you already know that the copies will not be read correctly by KIM. The fix? Simply tie a .02 uf capacitor to ground from the junction of R6 and R34. Now make the new master tape. Copies from this new master should be read correctly by KIM.

KIM-2 Memory expansion:

If you wish your KIM-2 (4K) memory module to reside in the 0400-13FF position already decoded by KIM, then read on; Wire - or the KIM K1, K2, K3, K4 decoder outputs together, add a 3.3K pull-up resistor to +5 volts and tie these four lines to the address line 12 input on the KIM-2, set all the on-board DIP switches to the "off" position, and you're all set!

.....

INTRODUCTION TO KIM-1 ARCHITECTURE

J. Butterfield
14 Brooklyn Avenue
Toronto M4M 2X5
Canada

This is intended to be a beginner's guide to the way KIM-1 is put together. It's mostly a hardware description with (hopefully) explanatory notes.

Because every KIM-1 owner has three fat manuals on his system, complete with detailed drawings, I'll try to save space here by referring to these manuals wherever possible.

The Address and Data Busses.

Let's start with page 24 of the KIM-1 User Manual (KIM-1 Block Diagram, Figure 3.1). We're going to concentrate on those two pipes on the left: the Address and Data Busses.

Every microsecond, the 6502 microprocessor sends out an address over the sixteen lines of the address bus. Sixteen lines are enough to address any of the 65,536 memory locations that could be fitted to KIM-1 (you only have 3,328 active addresses in the basic unit). In addition, there are a couple of other lines that accompany this bus: a Read/Write line (R/W) to tell whether the microprocessor wants to read or write memory; and a timing line, ϕ2 (phi, pronounced fy, two; it's confusing on the diagram because the phi looks like a zero).

The address bus goes to all memories. The idea is that when an address is generated, one memory only (whether RAM, ROM, I/O or Timer) suddenly says, 'that's me!' and connects to the data bus. If the R/W line says, 'read', the memory concerned places its data onto the bus; if 'write', the memory takes the data from the bus (placed there by the microprocessor) and stores it. For every address, only one memory unit responds; the rest stay silent.

This points out a fundamental difference between the address and data busses. The address bus goes one way only: from the processor to the memory. But the data bus information flows both ways.

This calls for a special kind of circuitry to connect to the data bus, called 'tri-state'. Every device on the bus might be (1) sending; (2) receiving; or, (3) ignoring the bus. (That isn't exactly how tri-state is defined, but it's a good way to remember it).

LOOKING FOR PROMS???? I just picked up four 82S129 bipolar 256x4 proms to make SUPERTAPE and other such used software a permanent part of KIM. S.D. SALES CO., P.O. BOX 28810, DALLAS, TEXAS, 75228, also has 5204 EPROMS (512x8, 1 usec.) for $7.95, 1702A (1 usec. 256x8) for $6.95,8T97B's for $1.25, etc. etc. All their parts are guaranteed and they deliver in about a week. Definitely good people! Get their catalog! If you order from this, include 5% for shipping, $0.75 for orders under $10.00, and tax if you're in Texas.
—the editor—

Since address bus signals go one way only, it doesn't need to use tri-state circuitry, which simplifies things. However, it could cause you minor problems if you wanted to expand your KIM-1 system so as to have two processors sharing the same memory, or if you wanted to drive a TV display directly from memory. In cases like that, the problem is called DMA (Direct Memory Access) and hints on how to solve it can be found on page 112 of the Hardware Manual.

Turning to page 25 of the KIM-1 User Manual, we can see the Address and Data busses in a little more detail. A little closer examination turns up a problem: the whole address bus doesn't connect to the memory modules. In fact, only 10 bits (numbered AB0 to AB9) out of the 16 are connected. That's only enough to define 1,024 addresses. How can it work?

Well, buried in the control logic block at the upper right is an 'address decoder'. Although figure 3.2 doesn't show the connections, this circuit is looking at address bus lines AB10, AB11, and AB12. Depending on what it sees in this part of the address, it may fire up either of the ROMs (via leads K6 and K7), or the Timer/IO/RAM section of the 6530's (via lead K5), or the 1K RAM (via lead K0). Each memory unit will work only if its respective K lead is energized. So now, each unit is responding to 10+3 or 13 bits of address information.

The address decoder, incidentally, can be seen in more detail on the next page, Figure 3.3, Control and Timing. It's block U4; you can see the three lines of the address bus coming in from the left, and the select signals (K0 through K7) going out at the bottom. Incidentally, outputs K1 through K4 are not used on the basic KIM-1; but they are available to you for expansion purposes. Each lead is capable on controlling 1K (1,024) bytes of memory.

What about the other three bits of the address bus, AB13, AB14, and AB15? KIM-1 ignores them. They are not needed for the basic system, so they're not used.

This leads to an odd feature of the basic KIM-1 system. Since the three high-order bits of the address are ignored, the memory starts repeating itself after reaching 1FFF hex. For example, try storing something in address, say, 0123; then look at the contents of 2123, and 4123, etc. They're all the same location!

This explains something that can be quite troublesome to the beginner. Both the Hardware Manual (page 40) and the Programming Manual (chapter 9, pages 124-146) state that the interrupt and reset vectors are located at hexadecimal addresses FFFA to FFFF. Yet reading the KIM-1 Monitor program listings shows them to be at 1FFA to 1FFF. Very puzzling--until you realize that in the basic KIM, they are the same locations!

It also outlines something you'll need to take into account when you add memory. If you use the 4K decoded for expansion with lines K1 to K4, no special reconfiguration is necessary ... you're staying in the area where the 'duplicated' addresses won't bother you. But to go to addresses at hex 2000 and higher, read Chapter 6 of the KIM-1 User Manual very carefully.

Interrupts, Reset, and the SST

The 6502 has three pins which force it to branch to a given location. They are called RST (reset), NMI (non-maskable interrupt), and IRQ (interrupt request). They all differ in detail, but the manner in which they cause a branch or jump in program execution is similar; we'll discuss them together.

First: when the appropriate input pin is 'kicked', the microprocessor hardware, after doing a couple of odd jobs, digs out the address that it will jump to from a fixed location. This location varies depending on which pin was activated, but it will be in the range FFFA to FFFF hexadecimal (see page 146 of the Programming Manual).

Next step: because of the way the KIM-1 addressing is wired, you may recall that address FFFA is the same as address 1FFA. This is part of ROM (the 6530-002 chip), so that the contents of these vectors are fixed. They point at programs to handle each activity.

Let's summarize what we have so far in a table:

Pin	Address of Vector Hardware	KIM-1	Contents (Vector)	What you'll find at the vector location
NMI	FFFA-FFFB	1FFA-1FFB	1C1C	Program NMIT (Jump indirect)
RST	FFFC-FFFD	1FFC-1FFD	1C22	Prgm RST (A reset program)
IRQ	FFFE-FFFF	1FFE-1FFF	1C1F	Program IRQT (Jump indirect)

So the branches we take are completely predefined for us by the Monitor program. In the case of Reset, the system will reset the stack pointer and then proceed to the main monitor program.

But the other two are different. Programs NMIT and IRQT, to which the two respective vectors point, are nothing more than indirect jumps. And the address to which you will jump, in both cases, is stored in RAM; so you can change these addresses to make the two interrupts branch anywhere you want. (Normally, you'll want the address of monitor program SAVE, at 1C00, stored in both vectors).

So there are two kinds of vectors: the hardware vectors at 1FFA to 1FFF which you can't touch since they are in ROM; and the software vectors at 17FA to 17FF which you must set up each time you turn on the system. (Oddly enough, the software vectors include a Reset vector that isn't used).

It's good practice, every time you power up, to key in:

AD 1 7 F A DA 0 0 + 1 C + 0 0 + 1 C + 0 0 + 1 C AD

setting the vectors and thus enabling the ST key and SST switch.

Now let's take a little excursion into software. When you push the GO button (or hit the G key if you're lucky enough to have teletype), the Monitor does a few last things before it gives up control to your program. Refer to page 39 of the KIM User Manual (Special Memory Addresses). You'll see that locations 00F1 to 00F5 are associated with various registers of the microprocessor.

When you push the GO button, the Monitor digs out these locations and puts them in the microprocessor registers. (You can read the program that does it at locations 1DC8 to 1E87 - it's quite clever).

This means something quite important to the programmer who's doing testing. You can set the initial values of any of the registers before you run - by storing the appropriate values in F1 to F5.

Note that the Monitor does all this. As far as the microprocessor is concerned, these locations have no special status; they are no different from any other part of memory. If a program puts something in the accumulator, the contents of F3 don't change, and vice versa.

Now, we've talked about what happens when your program starts. What happens when it stops? It would be handy (for debugging and other reasons) if the registers could be dumped out to the same locations (F1 to F5).

Well, program SAVE at location 1C00 does exactly that. So if possible, we should try to terminate a program by having it exit to location SAVE. Here are several ways of doing it:

1. Finish your program with the instruction JSR SAVE1 (20 05 1C). The accumulator will be lost, but the other registers will be saved correctly.
2. Finish your program with a BRK (00) instruction, and be sure your software vector at 17FE-17FF set to the address of SAVE (1C00).
3. To stop your program while it is running, press the ST button; but be sure you have previously set the software vector at 17FA-17FB to the address of SAVE

4. If you have t the SST switch to on, the program will stop a one instruction -- provided you have set the sof re vector at 17FA-17FB to address SAVE. More about how this happens soon.

How does the SST switch do that, anyway? Let's return to the hardware.

The diagram on page 26 of the KIM User Manual (Control and Timing) contains the basic information, especially in the area of U26 (a NAND gate).

We can see that the SST switch, when operated, will kick the NMI interrupt pin when the following two conditions are satisfied:

1) Lead K7 is not energized. In other words, the address bus is not in the range 1C00 to 1FFF;

2) The SYNC signal is high, which means the processor is fetching an instruction. (See page 44 of the Hardware Manual).

Put these together, and what do they mean? Just this: that if the SST switch is on, and we fetch an instruction that is not in the Monitor, the NMI interrupt will be signalled. The instruction in progress will be completed. Then, if we have set up our vector correctly, the micro will return to the executive, neatly dumping the registers as it does so. (Pressing GO for the next instruction reloads the dumped values so that the next step continues with exactly the conditions left by the previous one).

This is a really neat mating of hardware with software. The hardware allows the Monitor to run freely; but any other instructions are immediately interrupted, and the Monitor takes over again. The software, on the other hand, stacks away the register contents, and restores them when the GO button is pressed again. Elegant, huh?

Last word on interrupts: if you plan to expand the KIM-1 system, all of these features are available for you to modify. Chapter 6 of the KIM User Manual (Expanding Your System) gives quite a bit of material on this.

You might like to run the following note: "Tired of squinting at the 6500 Instruction summary--and mistaking B for 8, among other things? You might like a copy of my large type summary, which regroups by usage, and lists mnonics, hex code and formula for the effect of the command. For a Xerox copy, send a self-addressed stamped envelope with an extra 9¢ stamp(loose) for one copy or 13¢ stamp for 2 copies to: Mike Firth/6500
4712 Northway Dr.
Dallas, TX 75206 "

Digital Tape Drives for KIM-1

Robert E. Haas
2288 Blackburn St.
Eugene, OR 97405

As I stated in a previous newsletter, I am developing an assembler for KIM, and I planned to use the KIM tape I/O system for source input and object output. However, I have encountered several problems including low reliability, slow speed, and general frustration with the limitations of audio cassette recording. I have instead decided to interface a commercial-quality digital storage device to my system. I have chosen the Mini-Raycorder Model 6409 by Raymond Engineering, Middletown, Ct. This drive uses a miniature version of the standard Phillips cassette, a digital-certified version of the one being used in some newer dictating equipment. The drives cost $350 in singles, declining to less than $200 in large quantities. I would be willing to organize a group purchase of the drives to take advantage of quantity pricing. I will distribute my interface hardware and software design, as well as my assembler to anyone participating in the group purchase, for the cost of distribution.

The hardware portion of the interface consists of 4 bits of latched output and 5 bits of input (already available on the basic KIM-1). The dirves give the computer full control over tape motion (forward/rewind), and read/write status, and let the computer sense cassette-in-place, tape position (beginning or end), cassette write-protect, and cassette side. The technical specs of the drives are:

Recording mode: bit serial, bi-phase, 800 bits/inch.
Tape speed: 3 in/sec forward, 20 in/sec rewind.
Transfer rate: 2400 bits/sec (300 bytes/sec).
Tape length: 50 feet. Capacity: 60,000 bytes/side (2 sides)

Anyone interested in the group purchase should send me a stamped business-sized self-addressed envelope, and indicate how many drives you wish to purchase.