COMPANY CONFIDENTIAL

Engineering News

15 Oct. 1977 Vol. 4, No 11

Staff: Burgess Laughlin (Editor) and Scott Sakamoto (Art Director) Ext. 5674 D.S. 50-462

TEKTRONIX PRODUCT LIABILITY TEAM

THE PRODUCT LIABILITY SEMINAR

Product Safety Engineering sponsored product liability seminars on June 23, 1977 in the Beaverton and Wilsonville auditoriums. The purpose of the seminars was to encourage communication between the engineering community and Tektronix representatives most concerned with product liability and safety.

A short film prepared by Employers Insurance of Wausau (Tektronix' liability insurance carrier) presented the basic concepts and problems of product safety and liability. The film emphasized the team effort required to deal with product liability problems. Members of the product liability team include Tektronix field representatives (who must quickly report any situations where product liability claims may arise), design engineers, legal counsel and the insurance representative.

PANEL MEMBERS

Chairperson of the seminar was Rich Nute (corporate product safety engineer). Panel members included Nancy Mowlds (Tektronix' insurance manager), Eric Jorgensen (corporate legal counsel), and Larry Krogh (a product safety specialist from Employers Insurance of Wausau).

PRODUCT SAFETY ENGINEERING

Rich introduced the panel members and then briefly described the major functions of Product Safety Engineering (PSE). PSE (located in building 58) reviews every new product at each milestone of the new product introduction phase system. PSE's review includes evaluations of hazards under normal use and forseeable misuse, and documentation of constructions which prevent hazards. Pete Perkins is the manager of the Product Safety Engineering group.

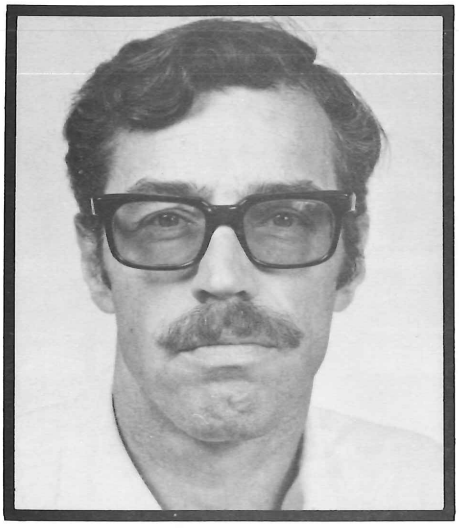
INSURANCE

Nancy Mowlds briefly described Tektronix' liability insurance program and pointed out that there have been no successful suits brought against Tektronix. We have been named in suits but the charge against us, in each case, was dropped before the case went to trial. She also reiterated the need for team work between manufacturing, marketing, engineering, product safety, and our insurance carrier. (For more information, see the TEKTRONIX LIABILITY INSURANCE insert on page 4.).

QUESTIONS AND ANSWERS

Where is the tradeoff between safety and costs?

Unfortunately there is no direct answer to the question. Adding the latest safety features, as well as documenting the safety decisions, may mean higher costs and therefore a higher price for the product. The consumer may not be willing to pay the extra cost if there is a less expensive (though less safe) alternative on the market.



Rich Nute, corporate product safety engineer.

Larry Krogh (Employers of Wausau) told the story of a manufacturer of power hand tools who was the first in the field to use double insulation rather than third-wire grounding to protect the user from shock. The double insulation is a superior system, but more expensive. The manufacturer went broke because consumers chose the less safe but cheaper models.

The manufacturer's documentation should show the safety-versus-costs tradeoffs that were made in product development. One factor that influences court decisions is what standards and what precautions other manufacturers followed at the time the product was sold. If a product can be made safer with only a slight increase in costs and if a manufacturer fails to adopt those extra safety precautions, then juries and courts may impose liability on the manufacturer.

What kind of safety precautions can the designer assume the user will take?

For most of Tektronix' products we assume that they will be used indoors and not in a driving rain or while the user is standing in a puddle of water.

Does Tektronix print safety warnings in other languages for products sold in other countries?

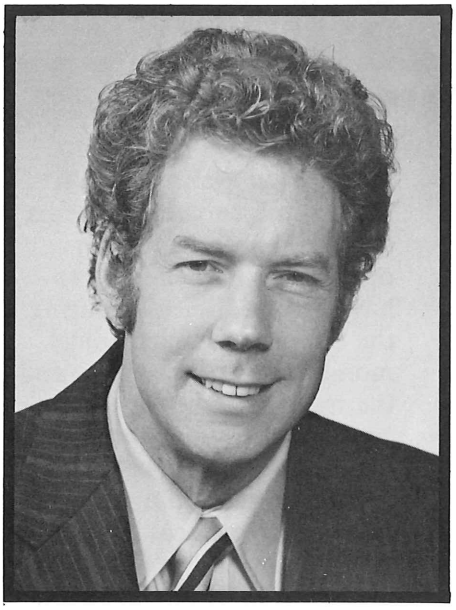
No, we don't. Most of the people using our equipment know English well enough to use the manuals and therefore to read the warnings.

How are liability cases handled in other countries?

The U.S. is the only highly industrialized country that awards damages through jury trials. So, defendants in damage suits in this country face problems not faced by defendants in other countries. Awards or settlements here may be as much as ten times larger than they are in other countries. Monetary settlement of liability cases has not been a major problem in other countries.

Are we liable for OEM equipment we use in our products?

Yes, everyone involved in the manufacture, design and distribution of a product is responsible for the safety of that product. If we buy an assembly from an OEM, we are responsible for it. Of course, we can also turn around and sue the assembly manufacturer to recover our loss if we have been sued.



Eric Jorgensen, corporate legal counsel.

Do we have a way to tell our customers that we've discovered a hazard in a product after it has been sold?

Yes. So far we have had very few hazards reported. Each instance has been handled on an ad hoc basis.

The Product Safety Engineering group has drafted a "hazard correction procedure" that defines the spectrum of options available for letting customers know about hazards. The options range from recall and repair to a simple warning of the hazard.

Products are reviewed for safety during NPI process, but when are manuals and advertising reviewed?

They aren't reviewed on a regular basis. However, Product Safety Engineering has been working closely with the manual formatting committee and the individual manuals groups, and contact is maintained with the advertising people.

Advertising is important in a product liability program. Here's an example of how unintentionally misleading advertising can affect a product. In one of its advertising brochures, a company showed a product in the back of a boat being rowed across a lake. UL withdrew its listing of the product because the device was never intended to be used in that kind of environment.

MORE INFORMATION?

For more information call Pete Perkins on ext. 7374.

Tektronix Liability Insurance

Tektronix does have product liability insurance, but this in no way relieves us of our responsibility to produce safe products. Can you imagine what it would do to our reputation if one of our instruments injured someone, or if we were involved in a lengthy litigation because of our products?

What does our product liability insurance cover?

It covers all sums which Tektronix becomes legally obligated to pay for bodily injury or property damage arising out of goods or products manufactured, sold, handled, or distributed by us when they are put into use by others and away from the premises.

It also covers defense of all suits brought against Tektronix alleging a cause of action which comes within the policy's coverage, even if such allegations are groundless, false or fraudulent.

What limits do we have?

Our primary policy, which is with Employers of Wausau, has a $1,000,000 limit. There is a $25,000 deductible. We have additional coverage on our Umbrella Liability policy, which takes over if the primary limit is exhausted.

Who is covered by our product insurance?

Any officer, director, stockholder, or employee, while acting within the scope of his duties is insured. If you, as an individual, are sued in a products case, our insurance company would provide your defense and pay any judgments.

What product liability claims have we paid?

So far we have not had to pay a single claim. We were named in a suit several years ago when a patient in an operating room claimed to have received injuries during the operation. Our medical monitor was in the room and the manufacturers of all equipment in the operating room were named in the suit. The judge ruled that the case was one of medical malpractice and not a malfunction of any product.

What can be done to prevent product liability claims?

A good communication network is essential between engineering, marketing, manufacturing, insurance, product safety, and our insurance company. We must keep records of all product-safety complaints. They should be reported to Rich Nute.

Quality control in the areas of design, purchasing production, packaging, and advertising is especially important. From the beginning of basic research we need detailed written records on each product. Any decisions made during the design phase involving cost factors versus safety factors should be documented. This assumes, of course, that the most safe alternative was selected. Otherwise we would have no defense against a products claim.

We should avoid broad and absolute terms to describe our products in advertising and selling. Warnings about possible hazards and safety instructions should be clearly stated in manuals.

And, finally, if one of our products does injure anyone or damage any property, it must be reported to me, Rich Nute, or Eric Jorgensen, immediately. Even a delay of one day can be important.

Nancy Mowlds, Tektronix Insurance Manager.

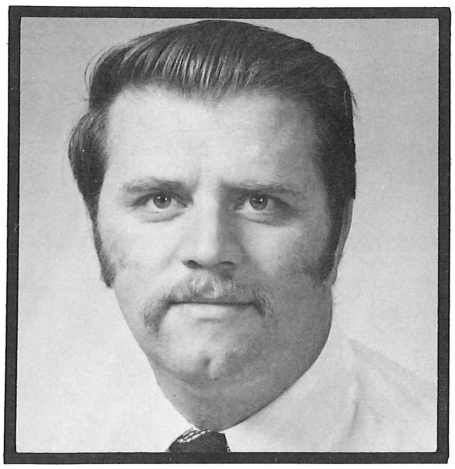
Product Safety Engineering and the NPI Process

The following discussion of the role of Product Safety Engineering in the new product introduction (NPI) process has been abstracted from the NPI Guidebook.

CONCEPT PHASE

Product Safety Engineering's job includes:

- Reducing hazards in the product.
- Making sure the product conforms to federal laws and regulations, and to county and city ordinances.
- Making sure the product conforms to published safety standards.
- Obtaining third-party certification (by independent laboratories such as Underwriters Laboratories and the Canadian Standards Association). This usually meets the requirement of the safety laws, regulations and standards. Third-party certification is a definite plus for product sale.
- Minimizing Tektronix' and its employees' liability if a claim is filed.



Pete Perkins, product safety engineering manager.

The design engineer should communicate with Product Safety Engineering from the beginning of the concept phase. The product proposal should answer these questions:

1. Is the product intended for third-party certification (UL or CSA for example). If not, why not?
2. What published standards is the product intended to meet? For example, will the product meet an IEC standard, even though not intended for third-party certification?
3. Are there special limits on safe use of the product? (Is it safe in a mine? in an operating room with flammable anesthetics? is it double insulated?)

DESIGN PHASE

Product Safety Engineering reviews the product design, and documents the review. In layout and design of a new or modified product, the requirements of laws, regulations, ordinance, standards (such as UL, CSA, and IEC), and internal safety policy must be considered. The project manager maintains communication with Product Safety Engineering during the design phase. As the working model materializes, it should be available for Product Safety Engineering's investigation.

EVALUATION PHASE

Product Safety Engineering checks evaluation phase (A-Phase) prototypes and documents its findings. If these A-Phase units fairly represent the final product, then two units are sent to UL for listing (certification). Tektronix also sends UL a statement that identifies the features that will be improved before the final product is released. However, if many items need correction, one unit from the A-Phase and later a second unit from B-Phase (verification phase) will be sent to UL. If A-Phase units are very deficient in safety features, both units sent to UL will be B-phase units.

VERIFICATION PHASE

Product safety engineers (together with product design and evaluation engineers) appraise verification phase (B-phase) prototypes for safety. Units for third-party certification laboratories (such as UL) are selected and submitted at this time.

Product Safety Engineering documents the safety status of the product at this point. This document goes to the engineering release meeting. These questions are answered: Is the product sufficiently safe? Are there any remaining deficiencies? Does the product conform to the intended standards? ■■

Development Tools for Microprocessor-Based Instruments

Allen Hollister

Allen presented this paper at the Instrumentation Society of America conference in Las Vegas in May. The paper will be printed in Instrumentation Technology magazine later this year.

If you would like more information about the topics discussed here, call Allen on extension 6250 or drop by 39-092.

ABSTRACT

This paper discusses some of the problems (and solutions) which show up in a microprocessor-based instrument design. The tools, both hardware and software, which are needed to solve these problems along with the limitations of each tool are discussed. Most of the problems in a microprocessor-based instrument design are software in nature. The tools for their solution tend to be oriented for software design and troubleshooting. An effort has been made to define the terminology used in this specialized area of computer programming.

INTRODUCTION

Today, every designer needs to know how to use microprocessors. If you're not already familiar with them, this paper can serve as an introduction to the theory and can provide a few hints on how to handle them using some of the newest tools for developing microprocessor-based instruments.

A PRIMER ON MICRO-COMPUTERS

A Microcomputer

At the onset, we need to define some terminology. If you have worked with minicomputers before, then a microcomputer will look like a less powerful minicomputer. Microcomputers have a smaller instruction set and they are slower, but they are still basically minicomputers. On the other hand, with a hardware background, a microcomputer will look like a sequential state machine that can replace thousands of random-logic chips with just a few chips.

A Microcomputer System

Physically, a microcomputer consists of a microprocessor chip, input/output capability, and associated hardware such as clocks and power supplies. A microcomputer will also contain either read-only memory (ROM) or random access memory (RAM) or both.

Microprocessor

The microprocessor is the heart of the system. The microprocessor gets an instruction from memory, does what the instruction tells it to do and then begins the cycle again by fetching the next instruction.

ROM and RAM

For most systems, instructions reside in ROM, but some systems use RAM. Besides its control lines, a ROM has a set of inputs called address lines, and a set of outputs called the data-out lines. See figure 1.

When the ROM sees an address word, a data word (a logical bit pattern) appears on the output. That word was programmed into the ROM at the time of its manufacture. The microprocessor uses the data-out words as instructions. It's important to remember that you can read data out of a ROM, but you cannot write into it. If the address lines shown in figure 1 contain 1001 (where a3 is 1, a2 is 0, a1 is 0, and 10 is 1), the data lines will be 1101 (d3 will be 1, d2 will be 1, d1 will be 0, and d0 will be 1). The number 1101 is the data word. A RAM has the same address input lines, and control lines, but you can write data into memory as well as read from it. Figure 2 shows an example of RAM.

Typical Microcomputer System

Figure 3 shows a typical microcomputer system. In figure 1 the address word and the data word were each four-bit words. But, the more popular microcomputers have an 8-bit data word (8 data lines) and a 16-bit address word. While those words are binary numbers, most people transform these numbers into hexadecimal (base 16) code to make them easier to understand. As an example, instead of saying that the next instruction is at memory address 1001 1101 0011 1111, it is easier to say the address is 9D3F. This convention makes it easier for people to understand, but the computer still needs to see logical ones and zeros on the 16 address lines.

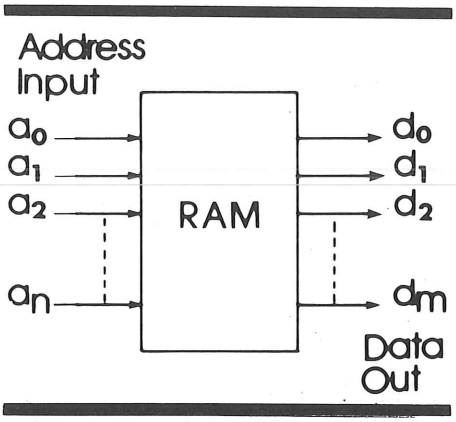


Figure 2. RAM nomenclature.

How to Program a Microcomputer

The microprocessor-based instrument designer's biggest task is programming the microcomputer and verifying those instructions. The instruction sets for today's microcomputers have from 40 to 200 instructions. Each instruction has two parts: an operation code and an operand (see figure 4).

The op code tells the microprocessor what operation to perform. The operand, however, tells the microprocessor what to operate on. Examples of opcodes are:

Add
Branch on a condition
Add immediate
Load
Jump
Decrement
No operation

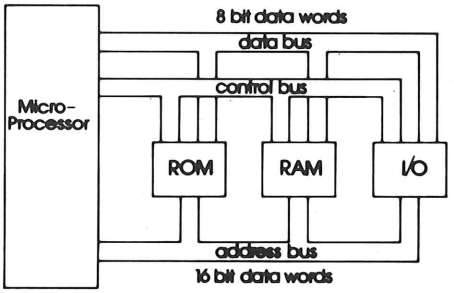
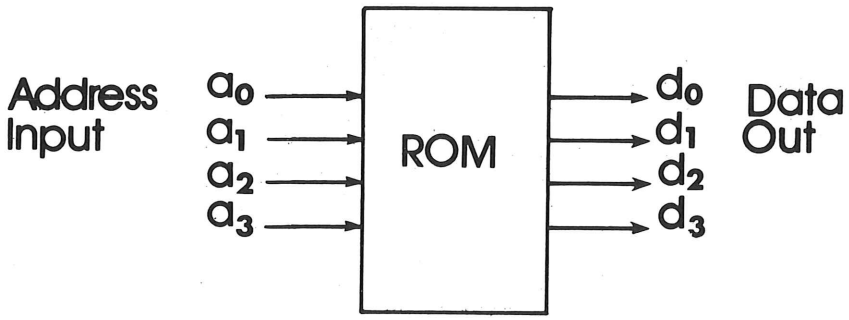


Figure 3. A typical microprocessor system.

Figure 4. An instruction.

Figure 1. A ROM example.

	Address					Data				
Hex Code	DEC Code	a_3	a_2	a_1	a_0	d_3	d_2	d_1	d_0	Hex Code
0	0	0	0	0	0	1	0	0	0	8
1	1	0	0	0	1	1	0	0	0	8
2	2	0	0	1	0	1	0	1	0	A
3	3	0	0	1	1	0	0	1	1	3
4	4	0	1	0	0	0	1	0	0	4
5	5	0	1	0	1	0	1	0	1	5
6	6	0	1	1	0	0	0	1	1	3
7	7	0	1	1	1	1	1	1	1	F
8	8	1	0	0	0	1	0	0	0	8
9	9	1	0	0	1	1	1	0	1	D
A	10	1	0	1	0	1	0	0	1	0
B	11	1	0	1	1	0	0	1	1	3
C	12	1	1	0	0	0	1	1	0	6
D	13	1	1	0	1	1	0	0	0	8
F	15	1	1	1	1	1	0	1	0	A



Every op code is represented by a specific number.

The operand is either an address or a piece of data but for some op codes there is no operand. As an example, an instruction, Add 2E tells the microprocessor to go to address 2E (in hex, but 0010 1110 in binary), get the data stored there and add it to the data stored in the accumulator (a register in the microprocessor). The accumulator stores the results of arithmetic operations performed by the processor.

Add Immediate 2E tells the microprocessor to add the number 2E to the data stored in the accumulator. In the instruction Add 2E, the operand (2E) was an address. On the other hand, in the instruction Add Immediate 2E, the operand (2E) is the data itself.

The instruction Jump 2E causes the microprocessor to go to address 2E to obtain its next instruction. There is an unconditional jump in the program flow. It is equivalent to the GO TO statement in FORTRAN. (The instruction set will also have some conditional jumps, equivalent to the IF statement in FORTRAN.

AN EXAMPLE

Figure 5 illustrates a very simple timing loop program. The program begins at memory location 5000, where a value of $A6_{16}$ (166_{10}) is loaded into the accumulator. The next instruction, Decrement Accumulator, subtracts one from the accululator. The instruction stored at memory address 5003 tests if the accumulator has reached zero. If not, it branches back to location 5002 and subtracts one again.

The program will loop through this sequence 166 times before the accumulator reaches zero; each time through the loop, it will use up 6 microseconds assuming a 1 MHz clock. Therefore, a full millisecond will be consumed in this delay loop. When the accumulator finally goes to zero, the program will continue at location 5005. A delay loop such as this is useful when it is necessary to wait a short time to let a switch settle, for example.

Two types of operands are illustrated in this example. In the instruction at location 5000 (Load Accumulator Immediate), the operand contains the actual data to be loaded. In the instruction at location 5003 (Branch if Not Zero to 5002), the operand FD is a relative address. In other words, FD does not represent an actual address; it represents the number of words, in two's complement arithmetic, that are to be jumped, minus three to reach location 5002. (The program counter is at 5005 while this instruction is executed thus it must jump back three places to get to location 5002.)

Programming Problems

Even in this short example we can see some of the problems involved in writing and storing a problem. In the simplest case we might have a front panel with 24 switches on it, each of which could be set to a logic 0 or 1 and could represent an instruction (16 switches for address, and 8 for data). Thus, you could set each switch, push another button to load the instruction of data into memory, and repeat this process for each instruction. This would get tedious very fast, especially if you have an average length program, say 1000 instructions. And boredom would be the least of your problems if the power went out...thus erasing your program.

There is another disadvantage to front-panel programming. In writing the initial machine language program in hex, you have to remember the op code number in hex....with 200 opcodes, it is not a trivial problem.

A third problem is keeping track of all addresses. This is very difficult. You can't calculate some of the addresses until the code is written. For example, in a jump instruction you wouldn't know where to jump to. Furthermore, if you need to add code, then you may have to change some addresses. Because of those problems, coding in machine language is extremely difficult.

Memory address (hexadecimal)	Opcode (binary)	Operand (binary)	Hexadecimal Equivalent	Description
5000	1000 0110	1010 0110	86 AG	Load Accumulator Immediate
5002	0100 1010		4A	Decrement Accumulator
5003	0010 0110	1111 1101		Branch If Not Zero to 5002
5005	. . .	. . .	26 FD	

Figure 5. Sample time delay program for a Motorola 6800.

The problem of initially storing the program could be solved if you had a paper tape reader and the software required to operate the reader. Then you could punch the code onto paper tape and have a permanent record. This is easier to do than using a bunch of switches, and you do have a record if the power fails. However, paper tape has the disadvantage of not being easy to edit in case you make a mistake. Paper tape readers are also slow and inaccurate.

DEVELOPMENT TOOLS

Fortunately, some tools have been developed to simplify the programming process. The first tool we'll consider is an assembler. An assembler allows us to do three things:

- use mnemonics for op codes
- use labels for addresses from which the assembler will calculate all of the actual address locations. If you add code to the program the assembler will automatically recalculate the addresses
- define constants which may be used anywhere in the program

If we wrote our sample program in assembly language, it would look like the program shown in figure 6.

```
        ORG   $5000
START   LDA   A #166
LOOP    DEC   A
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BNE   LOOP
```

Figure 6. Sample assembly language program.

This program, after it has been through an assembler, will generate the same machine code we generated by hand earlier. BNE (Branch If Not Equal to Zero), for example, is a lot easier to remember than 26 hex.

Another advantage of using the assembler is that we don't have to calculate address locations. The assembler will do that. A mistake in coding is easy to change and rerun through the assembler which then generates all new machine codes with new addresses.

If the assembler resides in a computer other than the computer for which the code is being developed, the assembler is a cross assembler. For example, a cross assembler may reside in a time-share computer. The development process is: log onto the timeshare system, write the program into timeshare memory, run it through the assembler, then write the program out in one of several ways. You can punch a paper tape with the machine code. You can then transfer the program from the tape into the memory of the system you're developing, and run it.

A second way of outputting the assembled program is to load it directly into your system's memory. This method would require using a modem, a direct line, or programming some programmable read-only memory (PROM) and plugging that into the system you are using.

If the assembler runs on the machine for which it generates code, it is a resident assembler. The resident assembler is usually cheaper to run than a cross assembler. Using the resident assembler avoids timeshare costs, but it may be slower than the cross assembler on a large computer.

HIGH LEVEL LANGUAGES

For large programs, coding for assemblers can be very tedious. It takes a long time to write the program and to debug it (we all make mistakes). High level languages (HLL's) have been developed to solve those problems. Using a HLL, you can write code more quickly and find bugs more easily. You will also be able to create programs that will run on more than one kind of microprocessor.

FORTRAN is an example of a HLL used in the scientific computer world. For microcomputer development the most used HLL today is Intel's PL/M. The prime difference between a HLL and an assembly language is that an assembler generates one machine language statement for each assembler statement, but a HLL generates many machine statements for each source statement. Thus the statement:

A =B/C

in FORTRAN (which says "divide B by C, and store the results in A") might generate 30 assembler statements. Now if you want to do this task in assembly language, you have to write those assembler statements yourself. Each one requires time to write, and each one is a possible bug. If you use the HLL, then the compiler (a program that generates machine code from a HLL program) automatically generates the 30 assembler statements required to divide B by C.

But there are disadvantages to using a HLL. Today, HLL generates more code than if the program were written in an assembly language. That's especially true for programs of less than about 5000 words of machine code. The HLL may run slower than assembly language programs do. But, as memory costs continue to drop and development time continues to rise, HLL's will become more attractive.

MONITORS

After you get your program written and assembled, you must verify operation and then get the logical errors out.

One way to verify and debug is to use a monitor program. The monitor usually resides in a ROM set aside for this purpose. (This ROM probably would not be shipped in the final product to the customer.) The user communicates with the monitor program through a computer terminal. The monitor has several functions:

- to control the terminal,
- to accept data from the terminal and deposit it in memory,
- to display selected portions of memory, preferably making use of a dissassembler which will decode the machine code to produce the mnemonic for that code, and give any equivalent addresses in hex. This is called "dissassembly."

The monitor program should be able to transfer control to the user program and then come back to the monitor after some condition is met such as:

- receiving a control command,
- executing a certain number of lines of code,
- hitting a breakpoint.

A breakpoint is an address (set by monitor command) that causes the program to go back to the monitor whenever the program reaches that address. The microprocessor registers will also be saved so they may be examined after the program returns to the monitor control.

The monitor's functions also may include trace capability. The trace function follows the program through execution and writes out each instruction (as it is executed) along with the contents of each register resulting from the instruction execution.

The monitor greatly aids in debugging software and is the minimum needed for that function. By itself it can't do the whole job. For example, when the monitor is performing some of its functions, the system cannot run in real time. If the monitor is busy doing a trace, the user program cannot run at full speed. This makes it more difficult to check out things like software timing loops. For that reason, the microprocessor system designer may add extra hardware such as hardware breakpoints and hardware trace.

HARDWARE TRACE

Breakpoint and trace functions in hardware allow the program to run at full speed. When the designer uses hardware trace, he may also add features such as the ability to store selected bus cycles, selected op code cycles, or all bus cycles.

How does the hardware trace work? Hardware trace is a special kind of logic analyzer which has memory and the ability to trigger (stops writing to memory). In hardware trace, the designer sets the condition that will trigger the memory. Then the monitor gives control to the user program which runs in real time. When the logic analyzer trigger condition is met, the logic analyzer records the data around that point and returns control to the monitor.

ICE

There is one other tool which is supplied with some development systems, that is in-circuit emulation (ICE). ICE lets the designer unplug the microprocessor in the target system (the system being tested), and plug in a debug cable that goes to the development system. The development system emulates (looks just like) the target system microprocessor to the target system. See figure 7.

ICE allows three modes of operation:

- programs stored in RAM within the development system can operate the target system with all data and I/O coming from the target system,
- programs that are resident in the target system can operate the target system (again, all I/O and data come from the target system).
- a combination of the two above.

Combining the two methods allows the designer to patch in parts of a program to try it out.

MICROPROCESSOR DEVELOPMENT AIDS

Some manufacturers have recently produced complete microprocessor development aids (MDA) which include all of the features we've talked about: an assembler, a HLL, a monitor, a terminal, ICE, and hardware trace. The MDA's allow other peripherals such as line printers, floppy disks, paper tape reader/punch and a programmable ROM programmer (PROM) to be attached. The MDA's bring together the essential tools for developing microprocessor systems. In many cases, MDA's are the optimum solution for microprocessor development.

HARDWARE AIDS

We haven't looked much at the hardware side of things (although some of the tools discussed so far are a definite aid in troubleshooting hardware). Most of the problems encountered will be in software or in software/hardware interactions. Still there will be problems for which some hardware tools will be necessary. This is especially true when the hardware system is first being brought up. Problems like getting appropriate timing to the memory elements, and finding glitches must be solved.

A logic analyzer is one of the better tools for hardware development. It can asynchronously sample many lines of data (typically 16), because the sample clock is derived from the logic analyzer rather than from the microprocessor clock. The logic analyzer can also store this data in memory for later viewing. Data is captured around some trigger condition. It is possible to capture data either before or after the trigger has occurred. The trigger itself usually is the output of a word recognizer.

A word recognizer looks at multiple lines and generates a trigger when certain bit patterns appear on those lines. As an example if you had four lines hooked up and you had the word recognizer set to 10X1 (where X is a don't care condition), then you would get a trigger out when either a 1001 or a 1011 appeared on those lines. The logic analyzer then displays the results in timing diagram form. Thus the designer can look at timing on many lines at once, search for and find glitches, examine things such as handshake logic, and debug random logic in general. It is possibly the most useful tool for the hardware designer.

Another hardware tool is the microprocessor analyzer. This tool performs essentially the same function as the hardware trace. So, if your MDA has hardware trace, you don't need a microprocessor analyzer to bring up a system. The microprocessor analyzer could be very useful, however, for servicing and perhaps for manufacturing if all of the features of a complete MDA are not wanted.

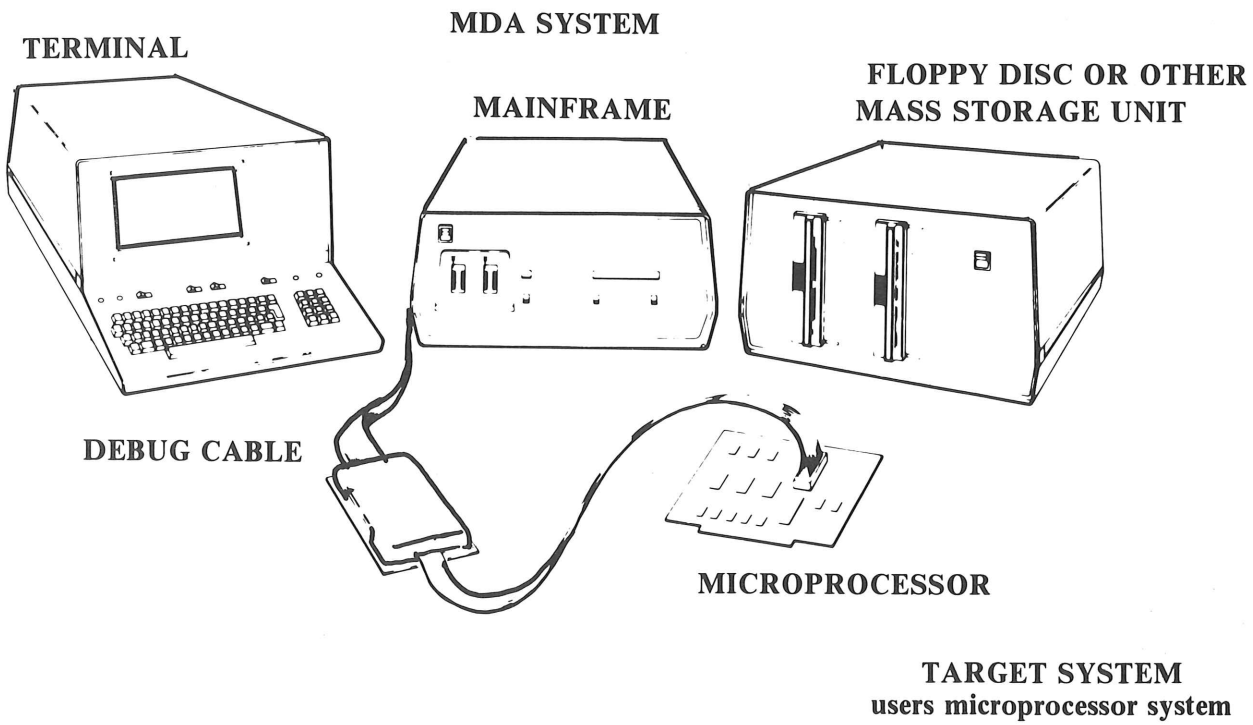


Figure 7. In-circuit emulation.

A Reminder

"Since our earliest days, we have considered patents to be important for the protection of our business. This is as true today as it was in the past. I would like to take this time to remind each of you that good patents come not only from good ideas, but from good habits and procedures.

First, an engineering notebook can be a valuable piece of evidence. Engineering notebooks are useful for many things, but particularly for jotting down the essence of an idea or bit of work which could lead to a patent. The second step is to file a disclosure with the Patents and Licenses Department (ext. 5266). They will be happy to help you.

Early completion of those two steps allows time for patentability evaluation and economic evaluation of the idea. Not all important ideas are patentable. However, some will be and I would like to ask each of you to make an effort to see that these are submitted so Tek may gain the benefit of patent protection."

Bill Walker

60 553

Maureen Key