Advances in Microcomputer Development Systems

Larry Krummel
and
Gaymond Schultz
Ramtek, Corp.

Introduction

The advent of the microprocessor has enabled designers to prove that they can now perform the same tasks previously accomplished with TTL or minicomputers but at the same time achieve much greater product complexity. Thus, results are not simply products that replace old ones with greater cost effectiveness, but ones which incorporate a magnitude of new features. The next generations of product design after the conversion to microprocessors will be characterized largely by a mastery of reconfiguration of memory and peripherals. However, a key problem that will remain in configuring microcomputer systems is the generation of microcomputer software and, in particular, its integration with the microcomputer hardware. Designers of first-generation microprocessor systems were often disturbed by their inability to significantly reduce the design and development time and indeed the cost of the product. The microprocessor-based system design learning curve was at least as steep as that of TTL design. The microprocessor represents a new logic family which requires new tools for development, because both hardware and software are required simultaneously for generation of the logic.

An equally important problem which has remained is the testing and debugging of the microprocessor product at the factory as well as in the field. Thus, a measure of a company's competitiveness will be its ability to develop new products much more rapidly than in previous times and to manufacture and service them effectively.

Historical approaches to design

Microprocessor chip families, as they are being utilized today, service two philosophically diverse areas: "microcomputers" and "user-packaged microprocessors." The term "microcomputer," as used here, means a card or system with microprocessor, memory, and I/O. These cards or systems resemble a minicomputer in almost every respect. As used here, the term "user-packaged microprocessor" denotes the microprocessor chips along with memory and I/O chips when designed and packaged by a user into a specific product. In this case the microprocessor is "buried" because it is not a separately packaged entity. Historically the user-packaged microprocessor grew out of hardware sequential logic, while the microcomputer grew directly from the minicomputer. Microcomputers do not represent a dramatic departure in concept from minicomputers. On the other hand, user-packaged microprocessors do represent a dramatic new concept—as well as a complete new set of problems.

Microcomputer characteristics

Microcomputer systems generally have the following characteristics:

1. They tend to be used in low volumes, and they are purchased by the user in an already packaged and working state.

2. They are usually RAM based, meaning that peripherals are required for loading software.

3. The development system is part of the product itself. A terminal is usually attached for controlling and loading software, which resides on the system for development and diagnostic purposes and which forms a portion of the product.

4. Testing of the product can usually be performed in a manner independent of the product application. In other words, testing is not application-specific.

The software development approaches for microcomputer systems follow the approach used with minicomputers.

User-packaged microprocessor characteristics

User-packaged microprocessors usually have the following characteristics:

1. They tend to be used in high volumes.
2. They are usually ROM based, meaning that very little RAM is available to test software and most likely no peripherals are available for loading software.
3. The development system must be totally separate from the product because the product generally does not contain the functions needed for a development system.
4. The peripherals for a microcontroller are usually special IC logic. The special IC logic usually constitutes the greatest percentage of hardware in the system. Thus more problems in development and production will occur in the peripheral IC logic than in the microprocessor chips or ROM memory chips. Microprocessor chip families are attempting to minimize TTL random logic designs, but today much TTL I/O logic surrounds the microprocessor.
5. Testing of software and hardware must be directed toward demonstrating that the product performs the specific tasks for which it was designed. Therefore, the testing is application-specific and concentrates on peripheral I/O logic.

Software development approaches for "user-packaged microprocessors" follow somewhat from the approaches used with sequential logic design. Software is written with the idea that it will run from ROM, not RAM. It is carefully segmented into ROM chip size boundaries (e.g., 1K byte boundaries) and addresses are assigned accordingly. Memory space is usually critical because of product volume, and therefore software is more tightly coded.

Figure 1 summarizes and contrasts the two categories from which present microprocessor usage has been derived.

MINICOMPUTERS	HARDWARE SEQUENTIAL LOGIC
MICROCOMPUTERS	MICROCONTROLLERS
• RAM based	• ROM based
• Low volume	• High volume
• Development capability is part of product	• Development capability is completely separate from product
• Peripherals are mechanical	• Peripherals are IC's
• General purpose testing	• Application-specific testing

Figure 1. Historical product approaches

Historical approaches to software

The cost of software development has dictated the procedures for implementing small volume systems. Indeed, before the microprocessor there were very few software systems which were reproduced in large numbers. Hence, almost all software development techniques have been oriented to reducing initial development cost.

The general procedure was to create system software which relieved the programmer of time-consuming decisions. This not only reduces development time but it reduces the programmer's awareness of the physical characteristics underlying his program. It also has the effect of increasing the hardware content to reduce the software development cost. In large volume systems, the increased hardware costs tend to be unacceptable, and new techniques for software generation may be required.

A quick review of system software tools will give some perspective on the increasing remoteness of the programmer from the hardware.

Machine language. The user must enter the zeros and ones for each instruction, and he must also predict in advance how much memory each program will require so forward jumps can be assigned.

Assembly language. The user is given a nmemonic representation of instructions to aid in code generation, and symbolic names are used instead of absolute values.

The assembler is a piece of system software which has no relation to the program being written. It is a tool, and indeed it is only useful with other tools such as editors. An operating system only begins to appear on the horizon.

Macro assembler. The macros may generate multiple instructions with a single command. The user becomes less aware of the actual code generated.

Relocating assembler. The user no longer has to know in advance where each variable and each program resides.

Compiler. The relation between function and code is completely controlled by the compiler. The same is also true of where and how memory is assigned except in the most general sense.

Figure 2 summarizes the historical development approaches which characterize minicomputer-like systems and "user-packaged" systems. The minicomputer type of system is characterized by its greater usage of software. It often makes use of higher-level languages. Code is usually relocatable within the large RAM-based system. File systems are common because disks or tapes are used for mass storage. These kinds of systems allow for a fast turnaround of errors, and therefore not as much time is spent coding the software. In most cases, an error can be quickly and easily corrected by a "patch" in RAM. Because of all the software aids and the "looseness" of the code, testing tends to be a short time per instruction.

MINICOMPUTERS	HARDWARE SEQUENTIAL LOGIC
• High-Order Language	• Prom Programmers
• Relocatable Code	• Scopes
• Editors	• Special Control Panels
• File System	
• Fast TurnAround of Errors	• Small Programs Hand Coded
• Patch and Run	• Careful Consideration
• Short Test Time	• Long Time to Test

Figure 2. Historical development approaches

The other side of the picture is also shown in Figure 2. Hardware sequential logic has undergone a significant evolution in method of implementation largely as a result

of the introduction of the semiconductor ROM. Finally, the microprocessor has offered a revolutionary approach. Historically, sequential logic was developed almost entirely with an oscilloscope. As PROMs were introduced, PROM programmers were needed and most designers constructed special "homemade" control panels for aiding in the debug process. The PROMs contained small programs which were usually hand coded with great care. Mistakes in either hardware or software were difficult to find and slow to correct. As a result, testing tended to require a long time per instruction.

The same things can be said for the first rounds of user-packaged microprocessors systems in many companies. Of course the problems are compounded when user-packaged microprocessors are combined with PROM-implemented sequencers in a single system.

Developing user-packaged microprocessor systems

The initial hardware debug effort for any microprocessor system is simply one of stimulating each memory and peripheral device on the system and verifying its operation. It is important to be able to stimulate the system *even if it is not working.* This can only be accomplished if the user can perform the following four primary functions:

- Write memory
- Read memory
- Write I/O devices
- Read I/O devices

Other functions which use combinations of these four may speed the debug process. For instance, a memory test which writes and reads memory over a range of addresses is very useful in initial hardware debug. If implemented correctly, the memory test will verify address decoding as well as memory handshake circuits. Also, the user will appreciate the four primary functions on a repeated basis for use as a "scope loop" in order to track down specific hardware malfunctions, once they are isolated within the system. All of these functions are possible via in-circuit emulation, since the emulator can generate the bus signals even if there is no operational user software. After the initial hardware debug phase the software must be integrated in order to exercise the system.

Spectrum of microprocessor development aids

It is not surprising that today's development aids for microprocessor systems have evolved from timesharing and minicomputer systems on one extreme and from hardware sequential logic on the other. Figure 3 indicates the spectrum of microprocessor development aids presently offered. The left-hand columns list aids which are more hardware oriented; the right-hand columns list aids which are more software oriented.

On the hardware development end, oscilloscopes and logic analyzers give the user pulse characteristic and pulse trace capability. Pulse tracing, like the software tracing counterpart, allows the user to "look back" to see what occurred prior to a point in time. These aids are obviously useful for debugging specific logic functions once a problem has been isolated. They are of little value in isolating a problem from the beginning because they do not provide a way to stimulate the system, and if the system is not operating it can not be expected to stimulate itself. Tracing microprocessor software flow with these devices is an inefficient use of time.

Recently a new group of instruments has been introduced which replace the logic analyzer and to some extent the oscilloscope for displaying microprocessor states. Called "microprocessor analyzers," they display instructions, addresses, and "1"/"0" states for handshake lines or any external logic. The Hewlett-Packard 1611 is an example of such an instrument. Again these types of devices have

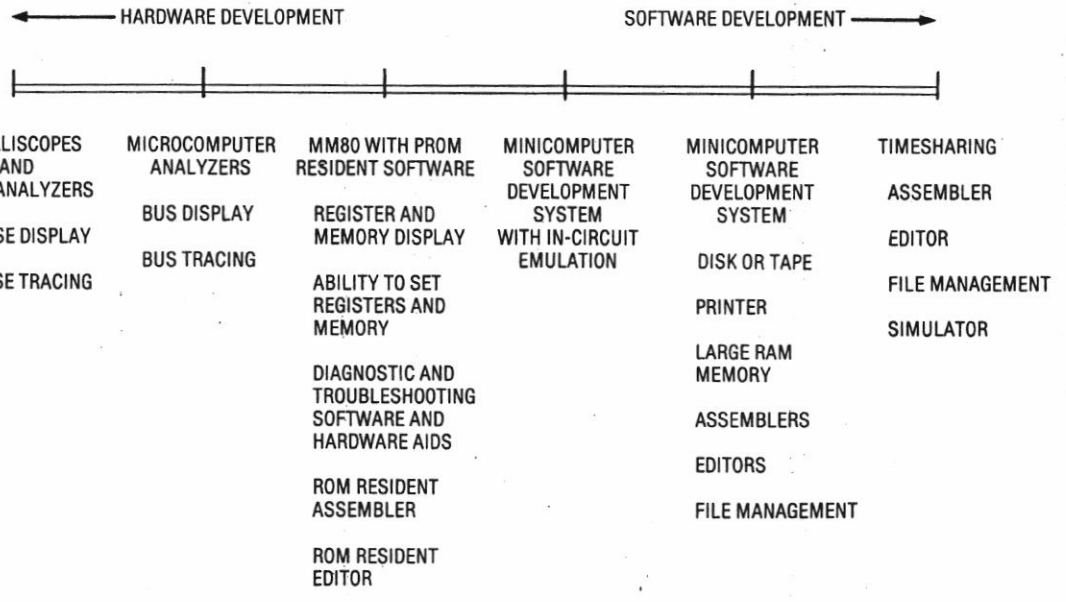

Figure 3. Spectrum of microprocessor development aids

no way of stimulating the system. They allow the user to view the program flow in mnemonic format, but if the system is "dead" little will be accomplished. In summary, all of these lack the means of stimulating the system even though it cannot execute software on its own. A desirable characteristic of microprocessor analyzers is that they are portable and they operate like an instrument. In other words they provide easy-to-use pushbutton operation. The user doesn't require knowledge of a lot of software and bulky peripherals such as floppy disk just to perform simple functions.

On the other end of the spectrum are the "minicomputer-like" software development systems. The Intel MDS is an example. These systems are pointed at solving the software development problems, and they do a good job in this area. They make use of the floppy disk and large amounts of RAM. They are characteristically large and stationary. Historically these systems have been complex to use because they are primarily intended for programmers, not technicians. As a result, they are not ideally suited for hardware debug, especially in product final test areas and in the field. In other words they lack the instrument-like operation.

Developing software in large RAM-based minicomputer-like systems is not totally consistent with most "user-packaged" final products which are ROM-based. One reason is that most ROM-based products are documented and released to manufacturing on a hardware basis. The PROMs contain software, but they are treated like a piece of the hardware. They are documented as assemblies and controlled by ordinary engineering documentation control procedures. The user must be able to correct an error in one PROM without affecting all the other PROMs in the system. Thus software must be planned and generated on PROM chip address boundaries. The user cannot simply reassemble the whole memory; he must set aside spare locations in each section and always work within a section. It would help if the development system were consistent with this important philosophy. Large RAM-based systems simply do not encourage users to plan their systems for PROMs. As a result, they are often forced into a significant software redesign effort when integrating into PROMs.

The needs for user-packaged microprocessor systems fall closer to the middle of the spectrum shown in Figure 3. They are truly half hardware and half software. The development system should show the characteristics of being an instrument. For ease of operation it must utilize in-circuit emulation—which might as well be a standard feature since it is always required. A typical company can use one or two of these "instruments" with full software development features (which extends the application to the right hand direction in Figure 3). However, the same company will need a number of lower cost "instruments" for product testing and field service. The Ramtek MM80 is one example of such a product.

Timesharing—a software-only tool

The use of timesharing for software development is especially useful for static software—i.e., the input/output relationship is unaffected by time. A floating point package is static in the sense that the output of a floating point add should be the same given the same input regardless of other variables. The careful use of timesharing can avoid large initial capital outlays, and it will also easily support a number of programmers on the same project. A timesharing system provides file space and software facilities (editing, etc.) which will usually surpass any microcomputer development system.

The reasons for not using timesharing for software are twofold: cost and real time. If the timesharing user is not aware of his costs, he can spend more on timesharing in a month than what he would spend on a stand-alone development system. If the user is aware of timesharing cost factors, then timesharing can be a very effective tool for non real-time functions. Debugging of real-time functions was done on timesharing before in-circuit emulation became available, since the timesharing simulator provided the user with information which was simply unavailable even on the minicomputer-like development systems. When in-circuit emulation became available as a peripheral to such devices as the Intel MDS or as the primary mode function of the Ramtek MM80, then timesharing simulation became a second-best method of debugging real-time software.

The concept of in-circuit emulation

In-circuit emulation, as the name implies, involves simulating all the operations of a CPU in its normal physical position. The purpose is to emulate the CPU while monitoring it with surrounding circuitry which it is not desirable to build into the user's product. In order to be most effective in performing in-circuit emulation with a particular microprocessor it is necessary to be able to implement the following functions:

1. Detect the beginning of each instruction execution sequence.
2. Suspend instruction execution at a defined point in the instruction execution sequence where execution can begin later.
3. Perform single simulated memory read, memory write, I/O read, and I/O write sequences.
4. Recover the contents of the CPU registers without destroying their contents.

The concept of in-circuit emulation may be more easily understood by considering an actual implementation for the 8080 microprocessor. Figure 4 shows the architecture of the Ramtek MM80, a single-processor approach to in-circuit emulation.

As shown in Figure 4, the MM80 uses only a single 8080 CPU which resides on a bus containing memory and I/O devices used by the MM80. The in-circuit emulation functions consist of buffers for the address and data busses, and mode control logic. When the 8080 CPU is operating as if it were in the user's system (user mode), all of the memory and I/O devices inside the MM80 are disabled by the mode control logic and the address stack is enabled in order to store instruction addresses for tracing software.

The user may set or examine his memory or I/O devices from the MM80's console. The MM80 performs memory transfers with the user's memory system by utilizing the 8080 move memory to accumulator and move accumulator to memory instructions depending on the direction of the transfer. These transfers correspond to the display and set commands which are issued at the MM80 console. In executing these commands, the MM80's software first issues an I/O command to its mode control logic followed by the move instruction. The mode control logic

disables the MM80 memory during the portion of the move instruction when the transfer occurs (last cycle), and it turns on the logic interconnection to the user system which is represented by the in-circuit emulator cable. User I/O devices are loaded or examined in a similar manner, except that the input and output instructions are substituted for the memory move instructions.

In the case of the 8080, its status information represents the beginning of an instruction sequence by the M1 line. Each cycle within an instruction is represented by the sync signal. The MM80 can step one instruction at a time by detecting M1.

When the MM80's 8080 CPU is required to change from a mode of operation within its own memory and I/O devices (called MM80 environment) to the user's memory and I/O devices (called user environment), it first saves it program state and then loads the user's last program state into all of its registers. The MM80 then issues an I/O command to its mode control logic followed by a jump instruction. The jump instruction serves to load the program counter with the address of the first user instruction to be executed. At the end of execution of the jump instruction the MM80 mode control logic switches the 8080 to the user environment and disables the MM80 environment.

Control may be returned to the MM80 (called monitor mode) by discontinuing operation with the user system at the beginning of a user instruction (M1 time) and forcing an 8080 RST instruction on the data bus during the next instruction fetch. This "simulated" instruction serves to cause the 8080 to begin executing MM80 software which will save the state of all the user's 8080 register contents.

These basic operations form the fundamental capability needed to allow the user to easily work with memory or I/O devices in his system by entering simple commands on the MM80 console. Some of the commands perform complex sequences which save the user considerable time. A few examples are summarized below:

1. Fill memory with a pattern.
2. Search memory for a pattern.
3. Verify that all of memory contains a particular pattern.
4. Continuously read or write memory or I/O devices with a pattern.
5. Run a memory test using 1's, 0's, and binary incrementing patterns.
6. Run a memory test using the "marching ones" patterns.

The important result is that these commands allow a test technician to perform comprehensive product tests without having to understand how to write software.

Another important approach to implementing the in-circuit emulation functions is the multiple processor architecture. The Intel MDS system is one example of such a system. In these systems a "master" processor provides a basic facility that controls a general-purpose set of development resources, and a slave processor tailors the programming, in-circuit emulation, and diagnostic functions to a particular class of microprocessors. This configuration is very much like a minicomputer with

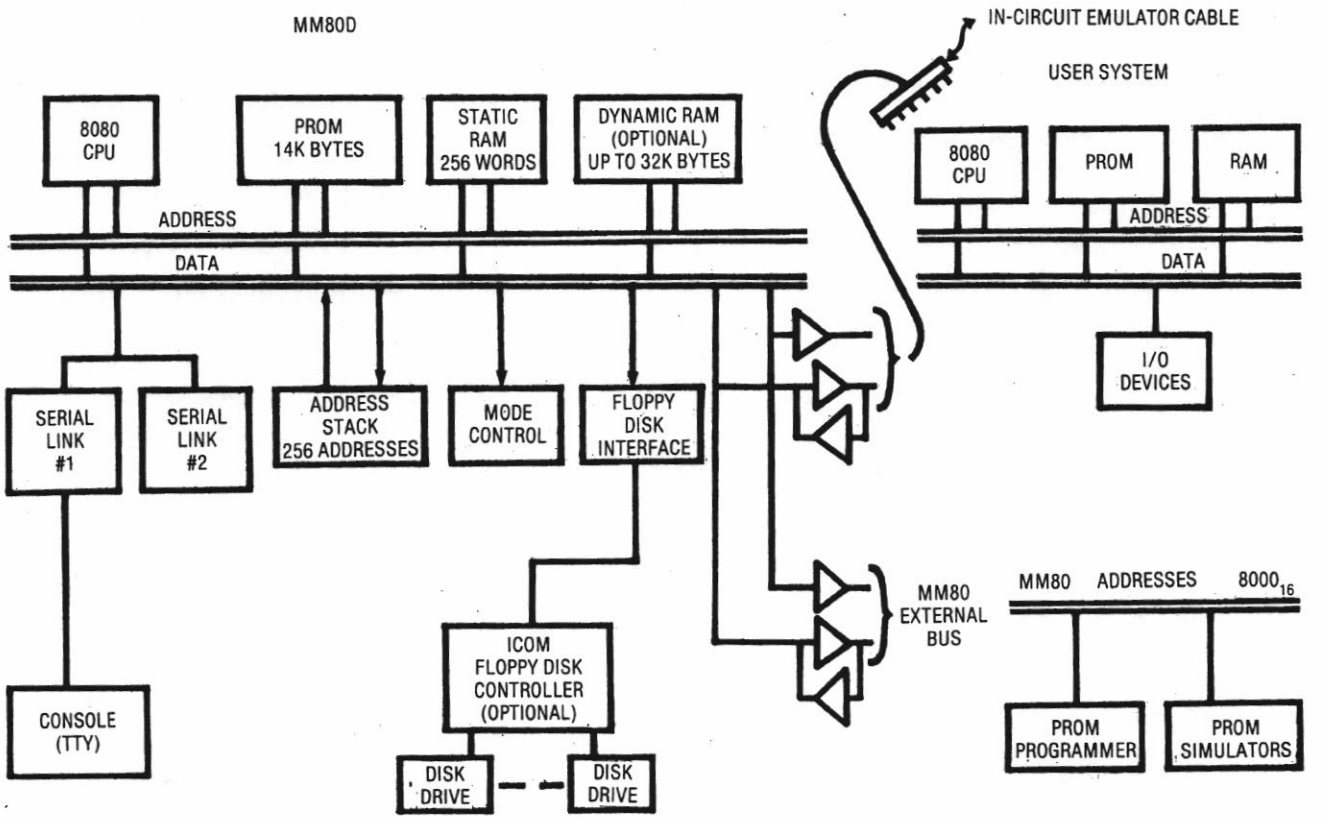

Figure 4. MM80 single processor architecture

in-circuit emulation as a peripheral which is capable of gaining full use of the master processor's environment. Figure 5 depicts the block diagram for a typical multiprocessor architecture. Note the many similarities between the two approaches of Figure 4 and Figure 5. The multiprocessor-organized bus shown in Figure 5 is somewhat more complex because the in-circuit emulator must become the system master when using main memory and peripherals. The "slave" in-circuit emulator of Figure 5 performs its emulation functions in much the same manner as described for the single processor approach.

The advantage of the dual processor approach is that more than one microprocessor family may be serviced without redesign of basic general-purpose development resources such as file management, text editing, and system I/O. Peripheral device controllers need be designed only once to be applicable to a wide range of microprocessor families with varying bus structures and instruction sets.

The primary disadvantages to the user of the dual processor approach are the increased size, cost, and complexity.

The significance of in-circuit emulation

In-circuit emulation, more than anything else, has improved the development of microprocessor systems. Its most important benefit is allowing the convergence of hardware and software in the final product without the need for extraneous and unwanted hardware or software. It has eliminated the need for intermediate steps toward a final user-packaged system.

It means that the product designer will not be burdened with added functions which are required only for system debugging and are not needed by the final product. ICE is a universal solution. One set of development tools can be used for *all* possible applications of a particular microprocessor.

The hardware designer or production test technician can work with the product and debug it even though it is not operating well enough to execute software. In-circuit emulation is an ideal way to evoke and record responses in a microprocessor system, because the CPU is the one point where communication must take place with all the memory and peripherals in the system.

Hardware debug has been an external function which should cause minor changes to the system operation (e.g., probe capacitance). The function of any individual IC was generally simple enough so that its output could be predicted easily by its input. The microprocessor is a much more complex IC, and the engineer cannot easily predict output without the tracing functions which are associated with in-circuit emulation.

On the other hand, software debug has traditionally been an internal function in that the debugging routines generally altered the system under test. The programmer had to be very aware of register and memory modifications caused by the debug routines. Very frequently the size of the debug routine placed severe restrictions on memory space. A relocating loader was especially important since the debug package had to be loaded with the routines undergoing checkout.

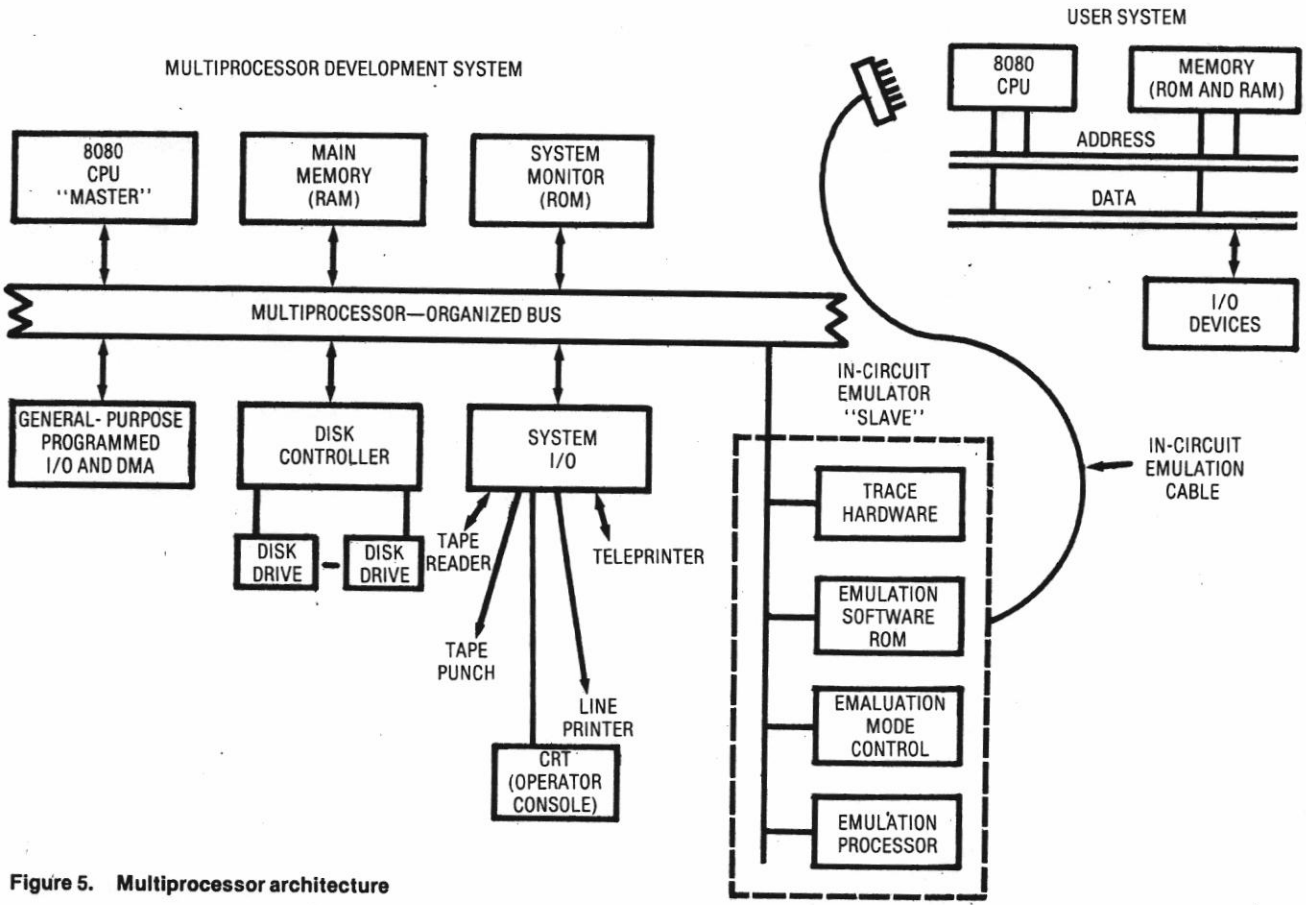

Figure 5. Multiprocessor architecture

The in-circuit emulation technique provides the programmer a complete debug package which is invisible to his own software. The programmer no longer has to consider register allocation requirements of the debug routine or residual debug inserts into the user code for tracing. The programmer writes his code the way it should be for production without memory or register restraints on how to debug.

The future of in-circuit emulation

The first versions of development systems with in-circuit emulation have concentrated primarily on the needs of the engineer and programmer. Some of these products, like the MM80, are being applied in production test areas and in the field. One extension will be the use of in-circuit emulation in automatic testing of microprocessor products. The in-circuit emulator could contain test program software which automatically sequences through the commands now performed manually by the technician. Since the product does not have to be functioning to isolate problems, considerable testing could be accomplished. In order to develop test programs the monitor software must operate on the internal environment of the in-circuit emulator in the same manner it does on the user's system. It should also allow test programs to take advantage of routines which already exist as part of the monitor—e.g., routines which cause the user's system to be displayed or written.

It appears reasonable that the instrument-like in-circuit emulator of today will someday acquire all the features of the "microprocessor analyzer" as well as comprehensive software development features on a modular basis. Will this entire package become another oscilloscope plug-in? Perhaps the oscilloscope, which is required less now than in previous times, will become a small part of the microcomputer development system. ■

Larry L. Krummel is director of software development for Ramtek Corporation in Sunnyvale, California. His previous experience includes work on microprocessor IC designs and microprocessor development systems for AMI, development of a time-sharing operating system at TENET, and satellite control system design at TRW.

Krummel received his BS and MS in mathematics from Iowa State University. He also received an MS in operations research from Stanford University.

Gaymond W. Schultz is vice-president in charge of engineering at Ramtek Corporation, where he has also managed both microcomputer development system and graphic display system projects. Previously he managed calculator and microprocessor design groups at American Micro Systems and was manager of I/O processor design at TENET.

Schultz received his BSEE from Washington State University. An author of several technical papers, he is a member of the IEEE.

Today, Tomorrow, and Beyond-

COMMUNICATIONS COMPUTERS, CIRCUITS

TELECOMMUNICATIONS AND THE COMPUTER, 2nd Edition
James Martin — I.B.M. Systems Research Institute
Complete up-date on the Bell System with sections on transmission and switching, new data networks, satellites.
1976 670 pp. Cloth $28.00

COMPUTER DATA-BASE ORGANIZATION, 2nd Edition
James Martin — I.B.M. Systems Research Institute
Explores techniques for logical organization, physical organization, and overall design of the data base.
1977 544 pp. (est.) Cloth $26.50

COMMUNICATION SATELLITE SYSTEMS
James Martin — I.B.M. Systems Research Institute
How communications via satellite can utilize all forms of electronic communications.
1977 416 pp. (est.) Cloth $35.00

FUTURE DEVELOPMENTS IN TELECOMMUNICATIONS, 2nd Edition
James Martin — I.B.M. Systems Research Institute
Examines projects of the near future and new techniques that portend radical changes in telecommunications.
1977 624 pp. (est.) Cloth $34.95

COMPUTER COMMUNICATION NETWORK DESIGN AND ANALYSIS
Mischa Schwartz — Columbia University, New York City
One of the first texts in this growing field; stresses quantitative approaches to design of data communications networks.
1977 352 pp. (est.) Cloth $21.50

MICROPROCESSOR SYSTEMS DESIGN
Ed Klingman — Applied Cybernetics, Los Altos, California
Both an introduction to microprocessors and a source of information on over a dozen processors, including 3rd-generation devices.
1977 416 pp. (est.) Cloth $17.50

MICROCOMPUTERS/MICROPROCESSORS: HARDWARE, SOFTWARE AND APPLICATIONS
John L. Hilburn and **Paul M. Julich** — both of Louisiana State University
Emphasizes the latest large-scale IC technology with direct methods for nearly any minicomputer application.
1976 368 pp. Cloth $18.50

DIGITAL CIRCUITS AND LOGIC DESIGN
Samuel C. Lee — University of Oklahoma
Describes logic design applications for all the latest MSI and LSI integrated circuits, including microprocessors and microcomputers.
1976 594 pp. Cloth $26.95

Prices subject to change without notice.

For free 30-day examination copies of any of the above texts, include titles on separate page and mail with this coupon to: Robert Jordan, Prentice-Hall, Inc., Dept. J-729, Englewood Cliffs, New Jersey 07632.

Name

School

Address

City/State/Zip

Prentice-Hall