Instrumented architectural level emulation technology

by HARRISON R. BURRIS
TRW Defense and Space Systems Group
Redondo Beach, California

ABSTRACT

The advent of general purpose emulators as tools for computer architecture research and system development is briefly traced. The concepts of an architectural level emulation and of instrumenting an emulation are introduced. An operational emulation-based computer system development facility is described. The results of a 1976 Independent Research and Development program aimed at improving emulation development time, emulation execution time, and instrumentation flexibility are discussed.

INTRODUCTION

There are, at present, a number of operational or planned computer architecture and system development facilities that are based upon general purpose emulation technology and have requirements to evaluate many different computer architectures and software packages. These emulation oriented facilities have found application across all segments of the computer industry including: TRW's Computer System Development Facility, Stanford's Emme, the Argonne Microprocessor, Army's Teleprocessing Design Center, and Reconfigurable Computer Facility. In spite of the successful implementations, further development of emulation technology is still required if system designers (hardware and software) are to be provided a reliable, cost effective system design tool. During the past year, TRW Defense and Space Systems Group conducted an Independent Research and Development program (for which the author was principal investigator) that had the objectives of (1) implementing an emulation based architectural research capability and (2) exploring the feasibility of several proposed extensions to that technology that would either increase the range of problems to which emulation could be applied or would increase the flexibility of emulation technology from a user standpoint. This paper is, in part, a report on the result of this IR&D effort.

Emulation center evolution

In 1951, Wilkes suggested microprogramming as a more efficient means of designing a computer sequence controller. Microprogramming also facilitated engineering changes to the architecture. By the mid 1960's, another benefit of microprogramming had been identified; by emulating earlier machines, microprogrammed computers could execute the software of the machines they replaced as well as software written in their own "native mode" instruction sets. This software compatibility prevented the loss of significant investments in software. The advent of the user microprogrammable computer (writable control store) in 1970 made possible a third application of microprogramming, the emulation of many computer architectures by means of a single host machine. The existence of general purpose emulators made possible an emulation-based system development facility. The next steps in the evolution path probably occurred in many locations at about the same time. The development of the Army Teleprocessing Design Center provides a clear history of the next steps.

In December 1970, the U.S. Army Computer System Command and elements of the Department of the Army Staff began exploring the application of emulation to a software development facility. The rationale was that a software developer supporting many different computers could replace separate systems at his development facility with a single general purpose emulator. This machine could then be microprogrammed to emulate whichever of the machines was required for developing a piece of software. Further consideration of the potential for emulation within the Army led to the proposal of additional benefits: (1) Software development tools (e.g., traces, variable range checks and activity counters) can be embedded in the emulation permitting software testing without destroying the integrity of the software under test (e.g., address realignment due to breakpoint code), and (2) The hardware performance (e.g., signal timing traces) can also be modeled by emulation, thereby permitting evaluation of hardware and hardware/software design trade-offs. In 1973, the Army Teleprocessing Design Center was implemented under the direction of the Office of the Project Manager for Army Tactical Data Systems with the mission of providing both hardware and software evaluations.

INSTRUMENTED ARCHITECTURAL LEVEL EMULATION

Architectural level

Bell and Newell, in their book on computer architecture, defined a hierarchy of levels at which computer
structure may be described, two of which (programming level and logic design level) are of daily interest to the system designer. For the purposes of this paper, the architectural level model of a computer structure is considered to include both the programming level and logic design level since it permits hybrid models, portions of which are implemented at each of these levels of representation.

**Emulation**

Computer system simulations in which the processing functions performed by one machine (the target) are replicated on another computer (the host) have been termed functional simulations.\(^{14}\) While both interpreters and emulators are functional simulations of a target machine, they differ in the method of implementation. The interpreter is a software program that is written in a language executable by the host computer and which accepts as input data and then executes computer programs written for the target computer. An emulator is a set of equivalences (microprograms) between operation codes of a target machine language and the processing logic of the host computer. When the microprograms for a target language are loaded into the sequence controller of the host computer, programs written in the target machine language are decoded and executed by the hardware of the host machine. Traditionally, the suitability of an interpreter or emulator for a particular project was determined by the trade-off: interpreters are fast (relatively low cost) to implement and modify and are more general (more space available for code), but have slow execution speeds: Emulators are slow (relatively expensive) to implement and modify, and restricted as to complexity modeled (limited control store), but execute considerably faster than interpreters for the corresponding target and host machines.\(^{1,16}\) In the immediate future, this trade-off will probably remain valid even for the high level language (Direct Executing Language) machines, since it would seem that the execution speed advantage remains with directly emulating the target machine instruction set instead of emulating a high level language and using an interpreter program written in the HLL to simulate the processing of the target machine.

A current paper\(^{16}\) succinctly illustrates that microprogramming (emulation) is becoming more user-oriented in a manner similar to the evolution experienced with "conventional" software. Recent developments in high level microprogramming languages,\(^{17\text{--}19}\) the availability of larger (and writable) control stores and other techniques being reported in this paper are beginning to eliminate the development cost and generality advantages of interpreters, and it seems entirely consistent to predict that within three to five years, emulation will be the implementation method of choice whenever the functions of one computer are to be replicated by another. However, until some of these newer developments have matured and are generally available, the above trade-off will remain an important consideration in deciding to emulate or interpret a target machine architecture.

**Instrumentation**

Computer instrumentation is the control and performance of (1) measuring, (2) screening, (3) recording, (4) processing and (5) displaying data describing the operation of the target computer. This ordering of activities does not necessarily reflect the operational relationship in an instrumentation system. While measurement is first and display last, the intermediate steps vary widely. For example, all measurements may be recorded and uninteresting data eliminated (screened) as part of the processing. Also, the instrumentation process may be interrupted by recording data and then processing and displaying it sometime later (off-line). Conventionally, instrumentation is performed by means of hardware monitors (probes and logic analyzers) attached to the physical hardware of the target system, or by software monitors (programs executed by the system being measured). A discussion of these techniques including identification of their deficiencies and proposed solutions is found in Reference 20, and a detailed treatment of computer instrumentation that is perhaps as thorough as is possible for such a new and rapidly evolving topic appears in a recent book by Svobodova.\(^{21}\)

The particular measurements to be performed during an instrumentation exercise will depend upon the question being investigated. Typical classes of measurements are: (1) interval timing (2) event counting, such as utilization figures for hardware resources, and software characteristics such as page fault frequencies and branching, (3) discrete values, such as those resulting from computations and replacements, and (4) extents such as the amount of primary or secondary storage consumed by a process. It is often very difficult or even impossible to obtain some of these measures on particular target computers because there is no way to attach a probe to the signal of interest since it is deep within the physical hardware.\(^{22\text{--}24}\) Also, there is nothing to instrument if the particular configuration of equipment for which the measurements are desired is either not in existence (under development) or not available. If an ALE modeling all of the signals and architectural entities of interest to the investigator is constructed, the desired measurements can be obtained by instrumenting the emulation of the target system rather than by instrumenting the actual hardware.

The idea behind instrumenting an emulation of a target system instead of instrumenting the target system itself is this. General purpose hardware and software instrumentation techniques are available. These methods can be applied to the host computer system as well as to the target computer system. The emulation "programs" representing the target system are microcode and, in some cases, special "emulation mode" hardware resources. The instrumentation of the emulated target machine hardware such as the program counter then reduces to instrumenting the memory location or register of the host machine where the emulated target machine program counter resides.

The concept of instrumenting an emulation of a machine is best illustrated with an example. If the contents of a buffer register and the timing sequence of signals on two
control lines are of interest, an ALE, including register transfer entities for these three machine features, must be created. Then instead of placing probes on the register and two signal lines, the memory locations with which the emulator represents the three register transfer level entities are examined by the instrumentation. The contents of the buffer register and presence of 1's or 0's on the signal lines will be the same as though the actual hardware had been probed.

If the ALE has already been implemented, instrumentation of the ALE can also be faster and less costly to perform than installing the probes on the physical hardware of the target machine since the only task involved is identifying addresses of memory locations within the host computer corresponding to target machine resources. This same flexibility that makes initial instrumentation of the emulation easier than instrumenting the hardware also makes it easier to reconfigure the probe points. With an appropriate instrumentation control program, dynamic reconfiguration during a measurement experiment is also possible.

**COMPUTER SYSTEM DEVELOPMENT FACILITY**

The Computer System Development Facility (CSDF) used for the IALE development IR&D is configured from mainframe computer systems maintained in TRW’s Mini-computer System Facility. A block diagram of the CSDF is presented in Figure 1. The host machine for the emulation is a Nanodata QM-1. The QM-1 is an extremely flexible two-level (micro and nano) user microprogrammable machine of the type characterized by Flynn as a soft computer architecture. The QM-1 design supports dynamic modifi-

![Diagram](image-url)
cation of both the micro and nano level programs. The microinstruction set is not fixed, but instead depends upon the mapping to the nano-level instructions. This is extremely important for the emulation of a large number of radically different target architectures. The processing and display of instrumentation data is performed by an Interdata 8/32\textsuperscript{27} configured with a Genisco Graphic Display System.\textsuperscript{28} At present, the connection between the measurements recorded by the QM-1 and the processing and display of the Interdata 8/32 is by means of a 9-track tape at 1600 BPS. An 8-bit parallel 9600 CPS bus between the two processors is currently being installed and dynamic interaction between the QM-1 and the 8/32 should be possible by March 1977. As part of the IR&D effort, it was desired to validate the instrumentation results obtained by the IALE (this is discussed in more detail later). For this purpose, an IMSAI 8080\textsuperscript{29} microprocessor system with much of the architectural generality of larger computer systems (as opposed to the microprocessor system development kits) was selected as the target architecture. The target architecture is shown in Figure 2. The unit was specially designed to support hardware instrumentation and both a minicomputer (PDP11/20) and a Hewlett-Packard logic analyzer were used to collect measurements.

Concept of operation

The following description of the use of the system development facility is presented to illustrate the capability desired. The system developer constructs the desired target system emulation at the level of detail appropriate to his experiment (hardware alternative selection, software debugging, etc.) by assembling already defined emulation building blocks from a library supplemented as necessary with newly defined emulations (which could be added to the library once they pass the accuracy of emulation test\textsuperscript{1}). Next, the desired measurements are specified in terms of the target machine architecture (e.g., busses, registers, status indicators, etc.) to be examined and the conditions (events) to be reported. The processing to be performed (e.g., calculation of channel and CPU active and wait state percentages) on the measured data and the method of display (e.g., tabular, Kiviat diagram\textsuperscript{30,31} etc.) is specified. The emulation and measurement microcode is loaded into the QM-1 micro and nano control stores under control of a "micro operating system", MOS. The target machine software is loaded into upper mainstore (lower mainstore contains the overlay library used by the micro operating system). The remaining instrumentation software is loaded into the instrumentation processor and the emulation experiment is initiated. The results of the instrumentation would be displayed to the user and the measurement, processing, and display could be interactively redefined during the experiment. While emulation microcode could be compiled or assembled under control of a more extensive version of MOS, the more straightforward method of iterating emulations is used. The system user must return to the support software provided by NANODATA to redefine the emulation. This process can still be initiated from the console, however, permitting a fully interactive evaluation session with a series of emulation experiments.

IALE feasibility demonstration

The concept of the feasibility demonstration was to implement an Instrumented Architectural Level Emulation of a particular target hardware configuration, build the target hardware system, perform instrumented experiments on both implementations, and compare the results. Considerable effort was expended during the project planning\textsuperscript{32} to design a set of feasibility demonstration experiments which would meet the twofold objectives of demonstrating the feasibility of IALE and also providing data on a target system application of intrinsic interest. The IMSAI 8080 microprocessor system to be used as the target hardware was assembled by the author using a mother board design intended to provide maximum access to each card and 2-level wirewrap sockets were used on the printed circuit boards instead of soldertail sockets to provide an easy means of connecting instrumentation probes (see Figure 3). The 8080 software selected for the demonstration involved encryption and fault tolerant communications processing. Detailed applications oriented analysis of the results of these experiments have been presented elsewhere.\textsuperscript{33,34} Figure 4 shows the process used to conduct each experiment of the feasibility demonstration. Aside from the lower path through the instrumented target system hardware, the process is the same as would be followed in performing any emulation-based experiments with the computer system development facility.

IALE TECHNOLOGY ADVANCES

As part of this project, improvements to the basic IALE concept were identified which had potential for improving
(1) emulation development time, (2) emulation execution time, and (3) modeling and instrumentation flexibility. The feasibility of each of these potential improvements and any restrictions they imposed upon the emulation process were explored. Several of the improvements under consideration had been previously employed with interpreter based systems, but had unique ramifications when applied to emulation based systems.

Microprogram operating system

Simulation at the circuit sublevel is used for some types of hardware analysis. If these simulations are to be performed by means of emulation, it is often necessary to somehow modularize the microcode since even the soft computer architectures do not provide sufficient control store for a program of this size. Even in cases where an

Figure 3—Target hardware and probe points

Figure 4—Feasibility demonstration process
entire architecture at the appropriate level of detail can be emulated by wholly resident microcode if many separate architecture alternatives are to be evaluated, the capability to construct the emulation from a library of predefined building blocks that are combined by a linkage editing or parameter passing process considerably reduces the emulation development time. For these two reasons, a microprogram operating system (MOS) was considered critical to the IALE effort.

The MOS is loaded by the QM-1 support system, uses a directory and library of emulation modules contained in lower main store to build the emulation from a predefined root module, and this continues to provide microstore memory management during execution of the emulation. The only interrupt presently recognized is F switch 1 which is used as a run/halt control. Eventually, the MOS will also service interrupts from the Instrumentation Processor. Input/output functions including those of the measurement processes are supported by the individual emulation and measurement modules.

The particular microcode instruction set defined for the feasibility demonstration did not support relocatable microcoding. This necessitated programming the microprogram overlays for fixed overlay points within the micro-storage space, and proved to be a serious restriction upon the flexibility of which programs could be operated with each other. For example, two overlays moving in and out of the same overlay space severely slow down the speed of execution. An experiment examining the 8080 CPU and a circuit level model of a memory control module executed a segment of 8080 machine code 5727.46 times slower than the target machine when the CPU and control module emulation overlays were coded at the same overlay point (and had to be overlain whenever execution passed from one to the other). When the memory control module was recoded so that both emulations could be coresident, time to execute the 8080 program was reduced to 91.89 times that required by the target machine. This result demonstrates the impact that overlay contention could have on a fairly typical computer system experiment, and indirectly indicates the value of a relocatable program approach to microprogramming. Since relocation was not available on the QM-1, a set of fixed overlay spaces were defined with sets of building blocks associated with each. This, of course, restricted the building blocks that could be assembled into emulations. Approaches to implementation of relocatable microcode such as suggested in Jones39 are considered to be one of the highest potential areas for further development.

The CPU and memory control module were also recoded as a single microcode program with a processing flow identical to the building block model except that the parameter passing support of the MOS was not needed and the emulation could execute as a stand-alone. This simulation was 86.19 times slower than the target machine. Therefore, for the particular segment of 8080 machine code being executed, a 6.61 percent execution time penalty is incurred to obtain the increased flexibility of modular emulations.

**Emulation building blocks**

One of the most significant trends in digital logic design has been the willingness of logic designers to forgo the speed advantages of discrete component systems for the design ease of MSI and LSI packages. The target system emulation design process can benefit from the use of predefined building blocks of larger functions in much the same way as the target processor hardware design process. A considerable reduction in emulation development time and cost could be achieved if an emulation designer could assemble already programmed building blocks into the desired emulation instead of having to code each emulation from scratch. This is just a further extension of the general argument for doing away with hand-crafted software.

As an example of the process, consider a small special applications device consisting of two identical micro-processor chips, a solid state memory chip, a contention resolving chip and a clock circuit. The logic designer can assemble the system design by joining together the chips without repeating the design of their insides. An emulation designer should be able to assemble the emulation of the system design, by providing code joining already working emulations of each of the chips being used. Furthermore, once this five-chip emulation is assembled, it can become a building block available for use in still bigger emulations.

This idea of a library of emulations imposes some additional design requirements on the emulations. The emulation code should be reentrant so that several copies may be active within a single larger emulation. For instance, the two identical micro-processors in the above example should require only one copy of the emulation code. Also, this library concept is a strong argument for the use of separate instrumentation routines since if the instrumentation is included in the emulation building block, it would have to be sufficient for the most general cases; which would be unacceptably slow and probably would make the code too big for most target systems of interest.

**Multilevel emulation**

When the host system is of a newer technology than the target machine, (e.g., an IBM 360 emulating a 1401) software often executes significantly faster on the emulation than on the target machine. However, when target and host are of the same technology or the host is being used to evaluate advanced target technologies, speed of execution can become a problem. In almost all cases, it seems likely that when emulations are used to simulate target machines (or subsystems) at the logic level, a speed penalty would be incurred. Thus, in the context of system development and computer architecture research, one of the problems with both interpreters and emulators is that they are slow in execution. The degree of slowness for both methods depends upon (1) the level of target machine detail being simulated and (2) the closeness of the mapping of the target hardware resources into the host hardware resources. Im-
proving execution speed by augmenting the host hardware is rejected since once a host has been selected, little can be
done to improve execution speed for general purpose
emulation experiments requiring several different target
machine architectures. If many emulation experiments are
to be performed on a single target machine architecture
(e.g., a software development facility), then it may be
desirable to improve execution speed by replacing complex
firmware procedures with hardware, such as the transform
boards often used in CDC 5000 series based emulators.\textsuperscript{5,13}
However, if this approach were adapted when several
alternative architectures were being considered, it might
become necessary to construct a different transform board
for each. This is nearly brassboarding each alternative—one
of the very things IALE is attempting to avoid.

The other factor, level of detail being simulated, provides
a general means of increasing execution speed. An
approach to the definition of emulations (also applicable to
interpretation) which I call multi-level emulation, can
considerably reduce the time required to execute a particular
emulation experiment by modeling in detail only those
operations of the target system that are of interest and
emulating at a higher level of abstraction the remainder of
the system necessary to provide the input data for the lower
level model.

Different types of system performance analyses (i.e.,
throughput, reliability, utilization, security) require the
measurement of different system parameters. The hardware
characteristics which must be observable in order to evaluate
these parameters also vary. This observation implied
that an increase in emulation speed due to reduced emulation
complexity is obtainable by restricting an ALE to
representations of target system hardware entities that are
required for a particular measurement experiment instead
of executing a general purpose (complete in every detail at
the lowest logic level) ALE of the target system for all
experiments. For example, during an experiment concern­
ing the utilization of a memory contention controller and a
shared memory, it was possible to model the memory
contention controller at the circuit level while the CPU, I/O
logic, and memory need only be emulated at a level of
detail sufficient to generate the memory reference
strings\textsuperscript{36–38} needed to drive the memory contention
resolver.

Figure 5 illustrates a multi-level ALE which could be
instrumented to perform the evaluation suggested above.
The CPU's and the remainder of the system, other than the
memory control hardware of principal interest, are emu­
lated at the RT level. This emulation, in terms of such
constructs as program status words, floating point registers
and interrupt status words would execute the object code
equivalent of target system programs. Whenever a refer­
ence is made to memory (next instruction or operand store/
fetch) the address is loaded into the architectural equivalent
of a memory address register (MAR). The MAR and the
equivalent memory buffer register MBR are crossover
paths between the CPU and the model of the memory
control hardware. The logic gate (AND, OR, XOR, etc.)
representation of the MAR is loaded with the address
contained in the RT level MAR and a cycle of emulation
through the memory control logic is begun. At the conclu­
sion of that cycle, control returns to the RT level CPU
emulation which resumes processing where it was halted
while the memory access was emulated.

For one of the alternative contention resolution circuits
evaluated a driver program of 8080 software executed 3354
times slower than the target machine when a circuit level
approximation\textsuperscript{a} of the CPU was used. When the same 8080
software was executed by an emulation using a register
transfer model of the CPU, the execution time was reduced
to 91.89 times that of the target machine, an improvement
by a factor of 36.5 times. For the above comparison, only
one CPU was modeled (no contention effects).

**Emulation timing**

When a functional simulation is used to determine how
much time (or how many machine cycles) a target program
will take to execute, it is generally satisfactory to maintain
a simple counter or "virtual clock" which is incremented
by the number of cycles required for each machine instruc­
tion. When the interaction of many signals (events) in a
piece of hardware is being simulated, obtaining the correct
timing between the signals is more complex. Approaches
used include an event queue which synchronizes the mod­
eled signals with a virtual clock that is regularly incre­
mented,\textsuperscript{19} or that synchronizes the signals with an external
clock that is regularly sampled by the host.\textsuperscript{4} An additional
method of time-scaling an emulation is to include the signal
synchronizing in the code for each step of an emulation.\textsuperscript{14}
This latter method is particularly useful for interfacing an
 emulation with physical hardware (e.g., universal interface
board) under conditions where the emulation produces

\textsuperscript{a} Some unaccountable aggregation of logic gates probably occurred since,\textsuperscript{13} the
internal design of the chip had to be assumed.
signals at precisely the same timing as the equipment being emulated (time-synchronous emulation). An emulation providing at least one level of detail at which events (e.g., occurrence of input and output signals) in the target system are related to the occurrence of the corresponding events in the emulation by a constant ratio (the scaling constant) is defined as a time-scaled emulation. When the scaling constant (emulated time/target time) is equal to one, the emulation is termed time-synchronous.

The register transfer level 8080 CPU emulation was time-scaled to the 8080 instruction set with a scaling factor of 9. This was achieved without an attempt to optimize the emulation speed. However, it seems to indicate that the QM-1 (without hardware augmentation) is not sufficient to perform a time-synchronous emulation of the 8080 chip. Considerations of target/host architecture mapping and relative technologies are again a factor.

As a counterexample, an error detection and correction process used for 1200 BPS communication links in the TACFIRE and TOS² Army Tactical Data Systems was emulated by the QM-1 at a 0.87 scaling factor. The QM-1 could easily be time synchronized as a driver to examine the performance of these tactical computer networks.

Figure 6 illustrates the process used to develop a time-scaled emulation. The accuracy of emulation test requires that all input-output transformations of the target system be replicated in the emulation before time-scaling is attempted.

### Emulation instrumentation

This section could equally well be titled “Instrumentation Emulation” because what was developed was a way of instrumenting an emulation by means of emulated instrumentation. Svobodova has suggested augmenting computer architectures with hardware to support measurement such as interval times and event counters. What we propose is to emulate these measurement hardware facilities along with the emulation of the computer hardware they are intended to measure. This emulated measurement hardware can then be used by external measurement equipment and internal software in the manner proposed by Svobodova. A significant advantage to emulation of the measurement hardware is that the amount and composition of the measurement hardware can be varied to suit the desired data acquisition requirements of a particular experiment. This flexibility in part anticipates a new law of computer performance analysis: For any computer architecture having N general purpose interval counters, there will be at least one experiment of critical interest requiring measurement of N+1 intervals.

The choice of using separate measurement programs instead of building the measurement hardware into the target architecture emulation was made to preserve the maximum generality of the system. This choice, as well as a desire to provide for external measurement (i.e., by DMA) decided another feature of the emulation architecture. At the microcode (microstore) level, each storage device of the target architecture is represented by a location in a resource vector of words in microstorage (i.e., a COMPOOL of emulator data sets). Instrumentation code outside of the building block can refer to the target hardware by a displacement into this table. Since the displacement is relative to an address maintained by the MOS for each building block and this address is updated whenever this resource vector is overlaid at a new point, the instrumentation is unaffected by microstore memory management actions. This organization of the emulation building blocks into executable code and separate data storage also permits reentrant coding by the creation of a new copy of the resource vector for each new use of the building block (each building block also accesses the resource vector through updatable relative addressing). Storage for all status and scratch areas used in the building block is appended just below each copy of the resource vector. An optimum policy for determining when to overlay a building block while retaining the resource vector in microstore card and when to overlay a building block and copy the resource vector to mainstore has not yet been formulated. For some of the circuit level blocks, it was possible to retain only the active resource vector in microstore because of size limitations. At present, all resource vectors are removed to mainstore when their building block is overlaid and are restored when the block is again activated. Some circuit level building blocks (i.e., 8080 CPU) require overlays within the building block because of the length of the code; these overlays do not affect the resource vector. Some typical microstore sizings are shown in Table I.
Conditional measurements

Not all the measurements taken during an experiment represent data of interest to the experimenter. For example, when determining which instructions cause the heaviest loads upon the ALU, any opcodes not associated with ALU operations are not of interest and can be deleted from opcode occurrence measurements. As the number of system parameters being measured increases, the difference between the amount of data present and the particular combinations that are of interest rapidly increases. In order to retain efficient host execution speeds, conditional measurements are introduced. When implementing the instrumentation, a choice exists between (1) recording of all of the measurement data and then screening the unnecessary data at the instrumentation processor as part of the analysis process or (2) screening the measurement data as part of the measurement routine processing. The objective is to minimize the amount of instrumentation processing that must be performed by the host and the trade-off is between the amount of time required to output all the measurement data as opposed to the time required to screen the data that is recorded. Bulk recording of measurement data is the most straightforward to implement, but, in general, results in unsatisfactorily slow execution of the emulation. The incorporation of the screening process into the host was, therefore, selected for implementation during the feasibility research.

The standard instrumentation building block accesses the object (host resource representation of a) target machine entity to be measured and then controls the I/O process responsible for outputting the measured data in a specified format on a selected I/O device (disk, tape, line printer, CRT or host front panel). The conditional processing logic is interposed between the measurement and the action (e.g., I/O) performed as a result of the measured data matching the specified conditional phrase. Conditional phrases can be formed recursively by concatenations of conditional phrases and any logical operators. Classes of conditional phrases which were built into instrumentation blocks included: (1) signals contained by any target processor resources (memory, registers, busses, status indicators, etc.), (2) values of clocks (virtual or actual), and (3) control console inputs (and planned instrumentation processor inputs). The interpretations of the target processor signals depend upon the architectural level of interest. Typical software oriented representations could include: (1) references or stores to specified target processor resources, (2) values of the target machine program counter, and (3) values contained by target resources. When measured, data are treated as higher level representations (i.e., signals as values or addresses) a conditional phrase with a value specification can also contain range specifiers (LT, LE, GE, GT). The conditional processing logic was directly microcoded into the building blocks during the feasibility research; when the interactive instrumentation link becomes operational and interpretive capability is planned.

Fault injection

For development and testing of secure and fault tolerant computing applications, it is desirable to be able to inject abnormal conditions (faults) into the processing stream and then be able to determine the response of the hardware and software. Interpreters with fault injection capabilities were used during the development of several fault tolerant computer systems. The same fault injection capability is desired for architectural level emulations of the building block type. The fault injection system is very much the complement of the measurement portion of an instrumentation system. The measurement system asynchronously removes data from the emulation system while the fault injection system asynchronously inserts data (faults) into the emulation. As part of the fault injection capability, it was decided to permit instrumentation results (e.g., instruction counts, and time intervals) to be specified as the triggering event for fault injection. No unusual difficulties were encountered in the implementation of fault injection for emulations. The fault injectors operate as independent modules in the same manner as the instrumentation programs. For the dynamic fault injection routines, the injection trigger is a conditional measurement command whose response is the injection procedure.

An 8080 software version of the error detection and correction algorithm used in the Army Tactical Data Systems was tested on a data path between two 8080's each executing a copy of the program and passing a string of characters back and forth. One, two and three bit errors were introduced and the trace of control through the software and variation in hardware resource utilization under the different error conditions was measured. Hardware faults (stuck on one, stuck on zero, and indeterminate) were then injected into one of the CPU's and the response of the software was followed.

SUMMARY

The application of Instrumented Architectural Level Emulation to computer architecture and computer system development facilities was described. Several efficiency-improving methods were described and experimental implementations performed as part of an Independent Re-
search and Development study were discussed. The results of this study indicate that for a host architecture like that of architectures can be constructed from a library of module building blocks and (2) that separate conditional measurement and fault injection programs can operate upon these architectural level emulations.

REFERENCES

27. Interdata 8/32, Interdata Corp., Oceanport, New Jersey.