The System Architecture Evaluation Facility—An emulation facility at Rome Air Development Center

by N. BRUCE CLARK and MICHAEL A. TROUTMAN
Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

Military requirements for data processing systems with unusual characteristics to perform specialized jobs have led to research into advanced architectures by the Department of Defense (DoD). Some of the requirements for systems have no counterpart in the civilian industry. Command, control and communications systems are typically complex and must be reliable and available with a high degree of certainty. This places great stress on the development of new data processing systems. The architecture, as the bedrock of all systems, must be continually improved in order to accomplish the increasingly complex software functions now demanded. Spaceborne automated systems simply cannot have an onboard team of vendor maintenance engineers to diagnose problems and replace components; a fault tolerant architecture is needed. Advanced radar surveillance systems provide a tremendous potential for information gathering but must be supported by parallel architectures which are still in the research phase. The DoD is actively involved in research and development of advanced architectures for tomorrow’s data processing needs and the System Architecture Evaluation Facility (SAEF) is an example of the use of microprogrammable (and other special purpose) computers to reduce the cost and improve the efficiency of this research.

Direct experimentation with unique hardware architectures is extremely expensive and time-consuming. It is also wasteful of resources, as the prototypes are rarely usable systems and must be discarded. Rather than actually build hardware components, they can be emulated by microprogrammable computers. Emulation is similar to a simulation of hardware, but must be supported by parallel architectures which are still in the research phase. The DoD is actively involved in research and development of advanced architectures for tomorrow’s data processing needs and the System Architecture Evaluation Facility (SAEF) is an example of the use of microprogrammable (and other special purpose) computers to reduce the cost and improve the efficiency of this research.

Instead of the traditional software programs, the microprograms which determine the actual control signals generated for machine language instructions are modified (or rewritten) to execute a different instruction set, the one for the “simulated” machine. In effect, the microprogrammable computer is molded to look and act like the proposed design at the instruction set (machine language) level. Thus, machine level programs written for the proposed design will execute on the microprogrammable machine.

It is an arguable position that this is still a software simulation, since microprogramming is just a lower level of programming. The difference is that the level of detail being used to describe the target machine is lower than the level it is describing. This is in contrast to using assembly language or a higher order language to simulate the instruction set of a computer. This gives a tremendous advantage in the time versus detail tradeoff, and thus this type of simulation is usually referred to by the special designation of emulation. Well written emulations of most architectures can execute within one order of magnitude of “real-time” for the proposed design. Thus a software function which takes one minute of execution time in the target machine might take 10,000 to 100,000 minutes (one to ten weeks of 24-hour days) on a detailed simulation, but only ten minutes on an emulation. Obviously these figures vary widely depending on the architecture being emulated and the computer being used to emulate, but are representative of the speed advantages gained with emulation over simulation.

SAEF ELEMENTS

To provide the emulation capabilities described, the core of SAEF consists of two microprogrammable computers, the Nanodata QM-I and the Multiple Microprocessor System (MMS) (Figure 1). Also included is a Goodyear Staran Associative Processor which will aid in evaluating single-instruction stream-multiple-data stream (SIMD) architectures. A larger general purpose computer will be used for the hosting of software tools to be used in connection with SAEF. Finally, all of these elements will be connected via the ARPAnet to facilitate intercommunication.
SYSTEM ARCHITECTURE EVALUATION FACILITY (SAEF)

Figure 1—Projected facility.

QM-1

The QM-1 is a high-speed general purpose digital computer that operates under two levels of microprogram control. These two levels provide extreme flexibility in machine definition and allow the advantages of both vertical and horizontal control. Machine instructions resident in Main Store are executed and defined by microprograms in Control Store. Control Store is a vertical level which is in turn implemented by Nanoprograms in Nanostore. The Nano level is a true horizontal architecture which has ultimate control over the total resources of the machine.

The QM-1 is primarily composed of a hierarchy of stores. At the lowest level is Nanostore consisting of 1K of 360-bit words with an access time of 75 ns. One nanoword is made up of a 72-bit K vector and four 72-bit T vectors (only one T vector is in control at any one time). Nanostore is a true read/write memory giving the programmer the ability to dynamically change its contents. Sharing control of the QM-1 with nanostore is F Store. F store consists of 32 six-bit registers which are used for residual control purposes. These registers determine the bus connections between the various units of the QM-1 as well as maintain the state of the machine. Moving away from the low level control of the QM-1, local store consists of 32 18-bit registers. The majority of these registers are general purpose but several of them have specific functions such as microinstruction registers and microprogram counters. External store is a group of 32 18-bit registers which provides specific functions including I/O interfacing, special main store addressing and generation of addresses for interrupts. Control store is a 16K by 18-bit read/write memory having an access time of 75 ns. This memory can be used for data storage and target register storage in addition to the vertical microinstructions. At the next level up is main store consisting of a maximum 256K 18-bit words. This is a read/write core memory having an access time of 750 ns.

The QM-1 contains several other functional units which are not considered part of the store hierarchy. These include a full 16-function 18-bit ALU, a 36-bit double shifter/shifter extension, an Index ALU for fast indexing and logical operations on local store, an RMI unit for rotating/masking/indexing the output of main store, and an ALU for operating on the six-bit F store.

The QM-1 is operable in both a stand-alone mode and in a time share mode connected to a DECsystem-20. In stand-alone, the QM-1 supports a full complement of peripherals. An operating system is available which maintains control over these devices as well as providing editor, assemblers and other useful routines. Also included are complete emulation debug and support packages which are independent of the operating system. These packages provide simple interfaces between an emulation and QM-1 resources and allow highly interactive sessions between an emulation and its user/developer. While in a stand alone mode, the QM-1 is directly usable by a single user thru the system console. When several users wish simultaneous access to the QM-1 it can be operated in a time-share mode.

In time-sharing the QM-1, it is connected to a DECsystem-20 via a common main store and the DECsystem-20 I/O bus. This system, which is known as Q-PRIM, provides an interactive microprogrammable environment in much the same way as when the QM-1 is stand-alone. However, in this case the QM-1 is treated as an I/O device by the DECsystem-20 operating system, TOPS-20. In this mode the QM-1 will have no peripherals of its own but will rely on TOPS-
20 to provide all its I/O capabilities. This resource will also be available to remote users because of the ARPAnet connection to the DECsystem-20.

The Q-PRIM software consists of four major modules. These are the QM-1 supervisor or "microvisor," the TOPS-20 QM-1 driver, Q-PRIM Exec and Q-PRIM debugger. The QM-1 microvisor interacts directly with the user's emulation and the TOPS-20 QM-1 driver. It is a small module which communicates between the QM-1 and the DECsystem-20. The microvisor provides context switching capabilities and handles the virtual memory addressing and paging from the QM-1 side. Accessing the QM-1 from TOPS-20 processes is done through the QM-1 driver. The driver communicates with the microvisor and TOPS-20 system calls. It is responsible for initializing the microvisor, controlling, scheduling, and swapping users, accumulating accounting data, and passing along I/O requests. The Q-PRIM Exec provides an environment on the DECsystem-20 which supports each of the emulations executing on the QM-1. The Exec provides a variety of commands that allow a user to build and interact with his emulated system. The Q-PRIM debugger is a table-driven, interactive, symbolic debugger that permits a user to debug target-machine programs in terms of symbols defined for the target machine. Data representations are controllable, thus allowing the user to tailor the emulation interface to more closely match the target machine.

**MMS**

Another key component of SAEF will be the MMS which is currently in the design phase. The MMS will consist of 64 microprogrammable microprocessors each operating autonomously or connected as part of an SIMD or MIMD architecture. It will contain a highly flexible and versatile interconnection system under software control which facilitates communication between MMS processors. The MMS will be able to effectively emulate shared memory, bus oriented, and crossbar switch interconnection schemes used in distributed multiprocessor systems. Control over the MMS will be accomplished by a Facility Control Processor (FCP) which is expected to be a minicomputer. This system will be usable in both a stand alone mode with the user communicating directly with the FCP, and in a remote mode via its connection to the ARPAnet. For the purpose of this discussion the MMS can be broken down into four sections—(1) FCP, (2) Emulation Engine Support, (3) Processing Elements and (4) Memory Subsystem.

The primary function of the FCP is to maintain control over the operation of the MMS. The FCP will provide both user and ARPAnet interfaces to the MMS. It will contain a host of run time tools which will allow the loading, modification, and control of individual PEs. Other support tools will include microassemblers, assemblers, compilers and software packages for processing of performance data. In its job of control over the MMS, the FCP is aided by the emulsion engine support.

Emulation engine support is broken into four areas. The Time Align Controller maintains master pseudo time for the MMS. The Emulated Local I/O Processor will create an I/O environment for each individual PE. The job of the Shared Resource Controller is to manage memory and communication paths. Last, the function of the Performance Monitor Processor is to collect all Performance Monitor System (PMS) event data from the PEs and emulation support and store this data on mass storage for processing by the FCP. Each of these devices communicates with controllers which are distributed among the PEs.

Each of the 64 PEs in the MMS will consist of a microprogrammable microprocessor and control hardware for I/O, memory, pseudotime, and messages. The microprocessors will be composed of microstore and an RALU based on bit slice architecture. A 16-bit RALU is the most likely size, with hardware aid for more efficient emulation of smaller word size architectures. Emulation of larger machines will be done with multiple instruction cycles. The I/O and memory controllers work in unison to provide an environment with memory mapped and I/O space I/O, local memory and shared memory. The pseudotime controller coordinates with the master time align controller for keeping PEs in step and the message controller handles communication between the local I/O memory unit and the appropriate emulation support processor.

The memory subsystem is partitioned into 64 each 32K word blocks each associated with a particular processing element. Individual blocks may be subpartitioned in any manner desired between local and global memory. Arbitration for the memory is handled by a portion of the shared resource controller and a local arbitration unit. The memory was partitioned in this way so as to give each PE fast access to 32K local words. Nonlocal accesses will be slower, because they take place through a shared bus.

The MMS as described allows for very detailed system emulations. In addition to emulating the computer architecture and peripherals as usual, one also has the capability to emulate the exact protocols of interprocessor communication and memory accessing. This is made possible by the programmable nature of many of the controllers located throughout the MMS. These features also enable the efficient emulation of I/O devices and virtual memory because the microprogrammable microprocessors are not burdened with these tasks, they can actually be done in parallel by the programmable controllers.

**STARAN**

Although not an emulation machine, a Goodyear Aerospace Corporation STARAN S-1000 associative processor interfaced to the HIS 6180 Multics system is included in SAEF as an aid in evaluating SIMD architectures. The associative processor can be operated in two modes, a stand-alone mode and an on-line mode to the Multics time-sharing system. In the latter mode, a Multics user is able to control the STARAN from his terminal as he would if he were using the STARAN in stand-alone mode. He can create program and data files using the capabilities of Multics and transmit them to STARAN. Currently the associative processor cannot be time-shared; that is, only one user at a time may utilize the STARAN. All communications between
ST ARAN and Multics are via a 12-bit parallel buffered I/O channel.

The ST ARAN basically consists of a conventionally addressed control memory for program store and data buffering, four associative memory arrays, a control logic unit for sequencing and decoding instructions from control memory, and a control logic unit associated with a special parallel I/O (PIO) capability. The associative array memories are the heart of the ST ARAN system. The array memories provide content-addressability and parallel processing capabilities. Each array consists of 65,536 bits of multi-dimensional access (MDA) memory organized as a matrix of 256 words by 256 bits with parallel access to up to 256 bits at a time in either word (horizontal) direction, bit-slice (vertical) mode or mixed mode (combination of the two). In addition to the MDA memory, each array contains 256 bit-serial processing elements. These processing elements provide the parallel processing capabilities for each array. Processing in the ST ARAN system can be overlapped with some arrays performing I/O while others are executing arithmetic and logic instructions.

The sequential control portion of ST ARAN consists of a PDP-11/20 minicomputer with 8K of memory and associated peripherals. The sequential processor also contains logic to interface with other ST ARAN elements. It runs system software programs such as the assembler and macro preprocessor, operating system, file handling programs, diagnostic programs and debugging routines.

Data manipulator

Another element of SAEF is the Data Manipulator which provides a flexible bit manipulation capability. The basic approach follows that described by Dr. Tse-Yun Feng of Wayne State University. Currently the Data Manipulator is attached to ST ARAN and allows the programmer to establish a relationship between input and output words such that, for each of the bit positions in the output word, any bit location in the input word may be specified as its data source. In addition, both input and output data can be masked.

Host computers

In order to provide many of the support tools required by SAEF a larger host computer must be included. At the present we will be using the Honeywell 6180 Multics and DEC System-20 time share systems located at RADC. These two computers will provide capabilities otherwise unattainable on the other elements of SAEF, either because of their small size or specialized nature. Hosting tools on a common computer also has the added benefit of reducing the number of operating systems the user has to learn. This is a primary concern as ease of use is the most important factor for SAEF. Because these hosts provide multiprogramming environments, several users of SAEF may be working on some aspect of a system emulation concurrently. Other obvious advantages are access to the ARPAnet and the amount of mass storage available on these computers. The host computers will communicate with the remainder of SAEF through the local ARPAnet connections.

Progression of SAEF

SAEF as described above will be developed over the next several years. At the present, SAEF consists of the DEC System-20 and HIS 6180 both connected to the ARPAnet, the ST ARAN and Data Manipulator with their connection to Multics, and the QM-1 in a stand-alone mode (Figure 2). Multics is the primary support host with its Meta Assembler, compilers, and editors. A 1200-baud serial line exists from Multics to the QM-1 for downloading purposes. The DEC System-20 currently supports a preliminary PRIM system utilizing a resident simulation environment instead of emulation by the QM-1. The Q-PRIM system is expected to be operational near the end of 1979. The MMS is currently in the design phase and is projected to be built by the end of 1981.

SUPPORT TOOLS

The hardware elements and software directly supplementing those elements are the core of SAEF. Several additional software support tools are in being or currently under development for use in SAEF, but are not exclusively limited to the facility and may, in fact, be most beneficial in contexts other than SAEF. Specifically, this section discusses the development of a hardware description language called SMITE for writing emulations, and study on the concepts of an automatically retargetable compiler which will accept machine descriptions written in a hardware description language like SMITE and produce an emulation of the machine.

Inherent in the design requirements for any usable item, be it a facility such as SAEF or any of its individual support tools, is its ease of use. No tool, no matter how vital, will be consistently and easily used if it is poorly interfaced with

SYSTEM ARCHITECTURE EVALUATION FACILITY (SAEF)

From the collection of the Computer History Museum (www.computerhistory.org)
the human link. The necessity to describe machine architectures is a valid research requirement, but the reality of writing an emulation in microcode, where typical productivity is a fraction that of assembly language programming, seriously impairs the utility of the facility. In order to overcome this problem, the development of a "high-order" language for machine description has been undertaken at TRW under-contract from RADC. The result is Advanced SMITE (Software Machine Implementation Tool for Emulation), an ISPS based language which allows machine descriptions at the register transfer level to be compiled into microcode for the QM-1.6 To support SMITE, a software system resident on the QM-1 is necessary. This system, called SASS (SMITE Applications Support Software), uses an augmented instruction set which is a superset of MULTI, the vertical level instruction set of the QM-1 normally considered to be a "native" instruction set, although it is in itself defined by the nanostore instruction set. SASS is a modification of the vendor supplied run-time package called TASK/PROD. SASS is resident on the QM-1, while the SMITE compiler is written in Fortran and resides on a CDC-6600 located at the Air Force Weapons Laboratory in New Mexico. Compilation of source code is accomplished remotely through the ARPAnet, after which the object code is transferred to the Honeywell 6180 (via file transfer protocol on the ARPAnet) and then to the QM-1 for execution. By the summer of 1979, a new version of SMITE with language enhancements will be delivered written in PL/1 and installed on the Honeywell 6180 (Multics) system at RADC. The advanced version of SASS will provide interactive debugging and performance monitoring capabilities not now possible.

To illustrate the basic syntax of the SMITE language, the following example describes a trivial eight-bit-wide machine consisting of three registers, 64 words of memory, and four instructions:

```
EXAMPLE: PROCESSOR;
DECLARE MEM(0:63)(0:7) MEMORY,
   ACC(0:7) REGISTER,
   IR(0:7) REGISTER,
DO FOREVER;
BEGIN;
IR~MEM(PC);
CASE IR(0:1);
   ACC~MEM(IR(2:7)); "LOAD ACC"
   MEM(IR(2:7))~ACC; "STORE ACC"
   ACC~ACC+MEM(IR(2:7)); "ADD ACC"
   ACC~ACC+1; "INCR ACC"
END CASE;
END;
EXAMPLE: END;
```

Once the description of an architecture has been implemented on a microprogrammable computer, there will be a need to write software for that emulated machine. In the same manner that support tools are necessary for writing machine descriptions, tools are necessary for applications software (the word applications is used to distinguish from the emulation software, or machine description, even though the "application" may be an operating system for the emulated machine). If no support tools are available, the system designer is thrown back to the dark ages of writing machine code for an emulated machine! Cross assemblers make this situation slightly more bearable, but a need exists for a compiler which would automatically compile to the object code of the machine described to it. It has been feasible to rewrite the back end of a compiler for new models of machines as they evolve, but the use of emulation and a high-level language like SMITE means that "new" machines are available from perturbations of the emulations due to fine tuning a design. The several man-months' worth of effort required to modify a compiler is clearly unacceptable.

Recognizing the need for such a compiler, RADC has contracted for the development of a support tool to be called the Retargetable Compiler. As an interim, users of SAEF are writing applications software in the assembly language of the emulated machine and using the Meta-Assembler, developed by McDonnell-Douglas, to create the object code.

### RESEARCH AREAS

SAEF is currently projected for use in two distinct areas of research. The most obvious research is into unique machine architectures for special purpose data processing systems and requires no further elaboration. The existence of SAEF also provides for a different type of research which may best be described as the implementation of the "Software First" concept. In the development of computer systems, it has traditionally been necessary and expedient to separate hardware and software functions early in order to define the "machine" and begin work building it. Since a major portion of the cost of systems involved hardware, the software was considered of secondary importance, and was developed to fit the machine. The advent of microprogramming and the tremendous decrease in the percentage of cost devoted to hardware implies that the software now can (and should) be designed to fit the problem being solved and the hardware is then molded to fit the software necessary for that problem. The concept of "Software First" has now been made possible through emulation of hardware and systems development can now be accomplished in a much more orderly and logical fashion. Research into systems development is being conducted at RADC under the name of Total Systems Design (TSD) Methodology.7

The TSD Methodology represents a departure from the traditional concepts of computer systems development. Instead of initially dividing the system into hardware and software subsystems and developing each independently until the integration phase, TSD encourages design of the system independent of the ultimate realization of individual functional elements.

The flow of the TSD Methodology breaks into three major divisions. The first portion addresses the general area termed requirements definition. Next is an area of detailed analysis which takes the requirements definition and results in allocated functions implementable in hardware and soft-
ware. Finally, there is an expression of the design process which uses emulation as a substitute for actual hardware until the system is validated to an extent justifying hardware acquisition.

REFERENCES