EMMY—An emulation system for user microprogramming*

by M. J. FLYNN and C. NEUHAUSER
Stanford University
Stanford, California

and

ROBERT M. McCLURE
Palyn Associates, Inc.
San Jose, California

INTRODUCTION

A relatively unique emulation laboratory facility is being developed at Stanford University to support research in computer architecture and language processing. The center of this system is a "universal host" computer which is capable of emulating (or simulating) the behavior of many other computers.

The facility will serve three purposes. It will allow researchers to:

1. access a variety of computera rchitectures—facilitating inter-architecture comparisons—providing for the processing of archival code for obsolete computers;
2. analyze the effectiveness of various computer architectures and compilers through the use of "software probes";
3. develop new "soft" computer architectures which reflect the artifacts of specific higher level languages—ultimately each higher level language would have its own coded machine architecture dynamically loaded into a host system.

For some time microprogramming techniques have been used in the design of computer control units. [1,2,3] However, in the past the principal interest has centered around read-only-memory microprogram systems.

The introduction of very high speed, read-write storage based on large scale integration (both bipolar and fast MOS technologies) represented a significant change in the above environment. Now microprograms—and data—could be rapidly loaded into the "control storage" which we term "microstorage" here. The environment of the Seventies introduced the possibility that one "host" system could serve as an emulator for a wide variety of "image" systems. This led to the introduction of two new architectural concepts: the soft computer architecture; [4] and dynamic microprogramming. [5]

The soft computer architecture—as represented by the Nanodata QMI [6] machine and to the Burroughs B1700 [7] takes advantage of this fast read-write capability, coupling it with a number of innovative processor features including:

1. field handling and selection
2. high speed shifting ability
3. extensive bit testing
4. flexible specification of data paths (residual control)

Each of these features can be used effectively in implementing interpretive emulation processes.

A soft architecture is enhanced through a technique known as "dynamic microprogramming"—in which the read-write micromemory is identified as the primary storage media of the system as well as the medium which contains the emulator code. Such architectures are arranged to both execute microinstructions and fetch data out of this fast storage media. The "control storage" then becomes a microstorage which more closely resembles an explicit Cache than a simple ROM. The advantage of dynamic microprogramming is that data access times can be shortened by having the data present in this high speed storage media, thus resulting in improved system performance.

EMMY

Emmy is the name given to the processor which forms the nucleus of our facilities at Stanford. It was designed, with severe cost constraints, to be an efficient as well as unbiased host machine. The goal was for a CPU design that could fit on one large printed circuit board with an inherently high instruction processing rate. Further, the design would have to accommodate the flexibility required to emulate a variety of conventional machines as well as to allow the development of new, abstract, language oriented machines. As a result, EMMY is both a soft architecture and dynamically micro-

* The general architecture of EMMY was based on a series of studies conducted at the Johns Hopkins University and was supported, in part, by the U.S. Atomic Energy Commission under contract AT(11-1 3288).

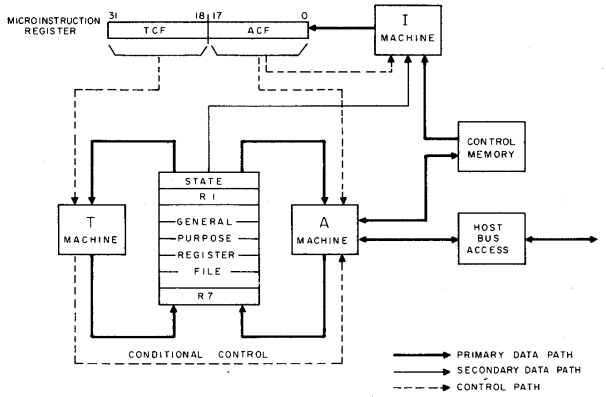

Figure 1—Structure of host machine

programmable. It is based upon a series of simulated systems which have been developed by our research group over the past several years. The EMMY machine is a 32 bit system which has 4,096 words (32 bit) of fast microstorage (access time 60 nanoseconds—cycle time less than 200 nanoseconds). The system is implemented using a very high speed technology—e.g., the switching technology has an internal cycle time of 25 nanoseconds. It is highly organized about LSI both in memory area and in processor implementation, and typically executes an instruction once every 200 to 400 nanoseconds. A strong influence on the design was the desire to minimize the amount of logic required to implement it, since cost and size were considered very critical.

An unusual feature, for a machine of this size, is that there exists a high degree of parallelism within the individual instructions. The host machine contains three separate, yet interdependent, finite state machines, each receiving control input from the current microinstruction and each controlling a resource associated with one class of instructions (Figure 1). These machines are designed as:

1. T-machine (controls functional resources),
2. A-machine (controls memory resources), and
3. I-machine (controls fetching of the next microinstruction).

Microinstructions in the host machine are formatted so that, in general, one half of the instruction (the T-control field or TCF) controls the T-machine and the other half (the A-control field or ACF) controls the A-machine. The I-machine may be controlled by either or both halves of the microinstruction.

Both the T- and A-machines manipulate data residing in the eight general purpose registers. The A-machine also moves data between micromemory and the registers and initiates communications with external memory units on the host bus. I-machine operation controls the fetching of the next microinstruction from micromemory. Host machine state information necessary to control the I-machine is contained in register 0 of the register file. Since this state register is directly accessable to the microprogrammer, flexible procedure oriented operations are possible.

Instruction set structure

Microinstructions (Figure 2) are 32 bits in length—the leftmost 14 bits, the TCF field, being dedicated to the control of the T-machine and the remaining 18 bits, the ACF field, being dedicated to the control of the A- and I-machines. Note that although there is a high degree of parallelism in these instructions, the TCF and ACF fields are vertically encoded independently of each other. The resulting microinstruction set is a relatively simple programming medium.

T-machine instructions

T-machine instructions are designed to provide the basic functional operations that the microprogrammer needs to emulate the functional and control aspects of a target machine. Instructions for the T-machine may be divided into the following classes:

1. logical,
2. arithmetic,
3. shift and rotate,
4. extended arithmetic, and
5. field insert and extract.

Instructions in the first four classes have a standard format which specifies opcode, subopcode, two register operands and indicates the possible use of immediate data. When immediate data is specified, the 18 bit field usually used to control the A-machine is expanded into a 32 bit quantity of immediate data. The extended arithmetic instructions subopcodes are designed to give the microprogrammer powerful single cycle operations with which to build complex target machine instructions, such as multiply and divide, by repetition.

Field insert and extract instructions are full word instructions which the microprogrammer may use to isolate and

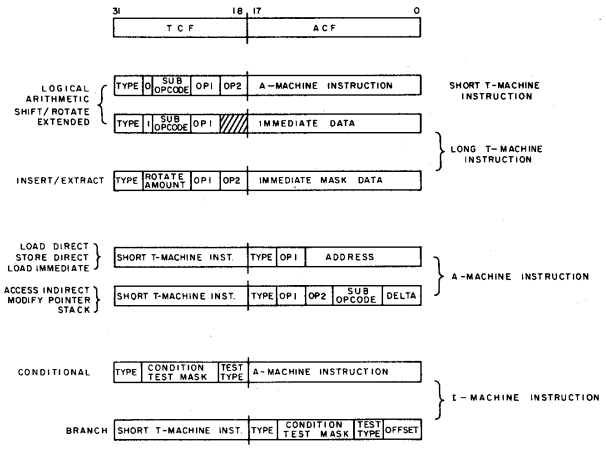

Figure 2—Structure of host machine instruction set

move fields of a data word residing in the registers. The insert instruction, for example, takes a word from one register, rotates it by a specified amount (0-31 bit positions) and places the result in a designated register under masking specified by the ACF field. This instruction is useful in breaking down target machine instructions and in matching host machine resources to target machine requirements when their word lengths differ.

A-machine instructions

A-machine instructions are used by the microprogrammer to access micromemory, manipulate address pointers, and communicate with external devices on the host bus system. A-machine instructions fall into the following classes:

1. move registers directly to and from micromemory,
2. load a register with immediate data,
3. access memory resources indirectly,
4. manipulate pointers, and
5. maintain stacks in micromemory

Access to external memory is designed so that once the operation is initiated the instruction address counter may continue to advance while awaiting the completion of the operation. This is an important source of parallelism in the emulation of instruction and data fetch in many target machines. A-machine stack operations allow the microprogrammer to access and maintain stacks in micromemory. Pointer manipulation instructions involve register incrementing, decrementing, addition, and conditional branching on results. Stack and pointer operations are particularly useful for operand indexing and sequencing of interpretive subroutines.

I-machine instructions

Fetching of the next microinstruction is controlled by the I-machine. Microinstructions are fetched sequentialy from micromemory unless the I-machine is specifically directed to fetch from a different location. Since the machine state, which includes the microinstruction address, is contained in one of the general purpose registers, the programmer may change the usual sequence by using the current microinstruction to modify the state register.

Within the state register is an eight bit condition code field representing various aspects of the previous T-machine operation (Figure 3). Instructions are provided to allow the microprogrammer to test these condition codes and control the operation of the A-and I-machines. These instructions are classified as:

1. conditional,
2. branching, and
3. looping

A conditional instruction is one in which the TCF field of the microinstruction specifies the testing of the condition codes and controls the subsequent execution of the A-machine. If the indicated condition is found to hold then the instruction for the A-machine, as specified in the ACF field, is executed, otherwise it is skipped. Using this facility the microprogrammer is able to specify conditional jumps, stacking operations, memory accesses and so forth.

A branch instruction may be specified in the A-machine control field (ACF) and allows the programmer to test the condition codes and perform a short relative jump from the current location based on the results. This instruction is used to provide control of the I-machine concurrently with T-machine operation.

Pointer modification instructions, which control the A-machine, may also provide looping capability. The results of each pointer modification operation may be tested for one of the common arithmetic conditions (e.g., less than zero), and the results of the test may cause a short relative jump. This instruction allows the microprogrammer to control repetitive operations such as normalize and multiply. In fact, the emulation of a target machine multiply instruction requires only one microinstruction since the extended arithmetic instruction "multiply step" and the looping instruction may be combined.

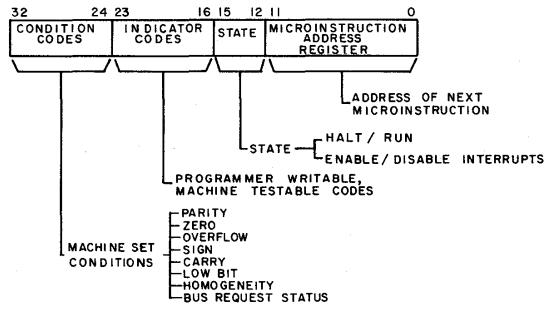

Figure 3—Layout of host machine state register

DATA FLOW DESCRIPTION

The general purpose register file (Figure 3) consists of seven 32 bit working registers and one status register. The status register (reg. 0) contains status and machine state information including the micromemory address pointer. The seven working registers are all full accumulators. There are two registers designated Register A and Register B at the input to the T-machine. These are temporary holding and shifting registers between the register file and the Arithmetic Logic Unit (ALU) in the T-machine. The ALU always accepts one operand from register A, and may accept the second operand from register B, the immediate field from the microinstruction, micromemory, or an outside resource. A multiplexer switch that gates the appropriate second operand to the ALU is also termed the "expansion unit" in that it can gate partial word operands left or right justified with zero or one fill in the remaining bits.

The ALU result is gated back to the appropriate (desig-

nated in the microinstruction) register in the register file. For arithmetic and logical operations a second cycle then gates the condition code into the correct field in register 0.

The Micro-Instruction Register holds the current micro-instruction being executed. As noted earlier, each instruction is divided into two parts: T-control field (TCF) and the A-control field (ACF). The TCF controls data transformation resources and the ACF controls auxiliary operations such as loads, branches, and I/O operations. For some instructions the ACF may not be executed due to the result of the TCF execution satisfying a given condition. Immediate fields from Micro-Instruction Register may be gated to either the ALU or micromemory.

The micromemory address counter is located in the right-most 12 bits of register 0.

All registers save those in the register file are transparent to the microprogrammer, although he should obtain a somewhat qualitative understanding of EMMY's architecture and its hardware operation.

REGISTER 0 (STATE REGISTER)

Register 0 contains 4 main fields. They are the Condition Code, Indicator Code, Machine State, and the Micromemory Address Counter.

The CCODE is set by arithmetic, logical and various internal operations. The CCODE comprises bits 31 through 24 of register 0. Bits 31 through 25 make the arithmetic condition code and are set only by arithmetic and logical operations. The significance of each bit is listed in Figure 3.

Bit 24 is set when micromemory is busy. The microprogrammer must test this bit to determine completion of memory cycle.

The indicator code, ICODE, (bits 23 through 16) is for programmer access only. Various TCF instructions may access this field for the purpose of setting flags or any other purpose the programmer deems reasonable (or unreasonable if he so desires). Thus the CCODE code is set by the machine while the ICODE is set by the programmer. However, both are testable by the branch and both are saved on an interrupt.

The Machine State is depicted by bits 15 through 12. The functions are indicated on Figure 3.

The Micromemory Address Counter, MAC, (bits 11 through 0) points to the next instruction to be fetched in the MIR from micromemory. After the MIR is loaded, the MAC is incremented, and the instruction execution begins.

ADDRESSING SCHEME

Certain devices within the EMMY and all external devices have an address assigned to them. All are connected to a common address and data bus. When an address is gated onto the address bus, each device looks to see if it is the device being selected. The following table lists all the internal device addresses. All other addresses are either unused or the address of an external device.

Address	Device
FF0000-FF0FFF	Micromemory
FF1000-FF1007	Register file
FE0000	Address display register
FE0001	Data display register
FE0002	Data/Address switch register
FE0003	Push button register
000000-03FFFF	Main memory addresses

INTERRUPTS

When the interrupt system is enabled and an interrupt is received, Register 0 is saved at a micromemory location (with the Micromemory Address Counter field incremented by (1) corresponding to the type of interrupt. Register 0 is then reloaded from an associated location and execution resumes. By reloading register zero, the programmer can (obviously) change all the information contained in register zero, that is, the condition and indicator codes, the machine state, and the Micromemory Address Counter.

A (partial) table of defined micromemory interrupt locations is listed below. Register zero is saved at the odd location (the listed address plus one) and is reloaded from the even location (the listed address).

Location	Interrupt Type
44	Console interrupt
46	Main memory interrupt
48	Console interrupt
4A	Block Transfer interrupt
4C	Bus time-out interrupt

PERFORMANCE

Microinstructions which reference other registers are executed in the 200-250 ns range with the exception of long shifts which require an extra 25 ns per bit shifted after 2 bits, extract-insert, which require long shifts, and a few of the extended instructions. Referencing micromemory requires an additional 200 ns. Referencing the external bus requires a varying amount of time depending on the unit referenced and the function performed. Memory activity specifically is conducted asynchronously, and the CPU need not wait for a memory read or write. Memory completion testing may be either explicit or implicit. A single statement of performance is difficult to make. A few examples might be most informative:

Example Timings of Emulated Instructions

Multiply—32×32=64 bit* (2's complement)	7.2 us
Divide—64/32=64 bit* (correct sign, 2's comp. rem.)	8.4 us
Binary to Decimal—20b to 7d* (unsigned)	5.0 us

Decimal to Binary—11d to 32b* (signed 2's comp., 360 style dec.)	9.2 us
MVC—360, 32 bytes non-aligned (assumes 1.0us main memory for both instructions and data)	12.8 us
AR—360 RR add (assumes 1.0us memory for instructions, regs. in micromemory)	7.4 us

IMPLEMENTATION

The natural choice of TTL was initially selected. It became apparent very early, however, that to meet the performance objective of an average of 200ns per microinstruction, it would be necessary to both use a large percentage of Schottky TTL and also provide two adder paths with attendant interlocking difficulties.

Since we have had extensive experience with high-speed current-mode logic, we took a second look at the cost and advantages of using MECL-10K logic. We found that the logic problems were simpler and the performance target could more easily be met. The additional care required in mechanical and electrical design did not prove to be a serious problem.

The resulting design requires a minimum number of logic design tricks. It is a straightforward synchronous design with a 25 ns clock. The T operations, A operations, and I operations are viewed as three cooperating finite state machines. Each of these three machines is implemented as a 16 state machine (although not all states are currently used in any of them). By implementing these three functions as independent, automata, simultaneous use of CPU resources is achieved with minimum difficulty. The bus control logic is independent of all of these and serves to further maximize overlap.

Physically, the CPU is on a single PC card of approximately 12″×15″. The micromemory and console logic each have a PC card.

* not including main memory access and instruction interpretation

CONCLUSIONS

EMMY is a low-cost, soft machine developed using high speed technology. This system has uniform 32 bit instructions and data paths.

The instruction format exhibits threefold parallelism: transformational specification, auxiliary (move and pointer handling) and an implied next instruction fetch. This parallelism together with fast native performance (200-400 nsec/instruction) produces respectable emulation capability across a variety of target machines.

REFERENCES

1. Wilkes, M. V., "The Best Way to Design an Automatic Calculating Machine," *Manchester University Computer Inaugural Conference*, 16-18, July 1951.
2. Husson, S. S., *Microprogramming Principles and Practice*, Prentice-Hall 1970.
3. Tucker, S. G., *Microprogram Control for System 360*, Vol. 6, No. 4, 1967.
4. Flynn, M. J. and M. D. MacLaren, "Microprogramming Revisited," *ACM National Conference Proceedings*, Vol. 22, Thompson Books, Washington, D.C. 1967, pp. 457-464.
5. Cook, R. and M. J. Flynn, "System Design of a Dynamic Microprocessor," *IEEE Transactions on Computers*, Vol. C-19, No. 3, pp. 213-222, March 1970.
6. Q M-1, Nanodata Corp., Buffalo, New York.
7. *B1700, Systems Reference Manual*, Burroughs Corp., Detroit, Michigan, 1972.