SU-SEL-75-045

A TALE OF THREE EMULATORS

by

Lee W. Hoevel
and
Walter A. Wallach, Jr.

November 1975

Technical Report No. 98

The work described herein
was partially supported by
the U. S. Air Force Office
of Scientific Research under
Grant No. AFOSR-75-2865.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY . STANFORD, CALIFORNIA

SU-SEL 75-045

A TALE OF THREE EMULATORS

by

Lee W. Hoevel
and
Walter A. Wallach, Jr.

November 1975

Technical Report No. 98

DIGITAL SYSTEMS LABORATORY
Stanford Electronics Laboratories
Stanford University
Stanford, California

The work described herein was partially supported by the U. S. Air Force Office of Scientific Research under Grant no. AFOSR-75-2865.

Digital Systems Laboratory
Stanford Electronics Laboratories

Technical Report No. 98

November 1975

A TALE OF THREE EMULATORS

by

Lee W. Hoevel
and
Walter A. Wallach, Jr.

ABSTRACT

This is a preliminary report on the development of emulator code for the Stanford EMMY.

Emulation is introduced as an interpretive computing technique. Various classes of emulation and their correlation to the image machine are presented.

Functional and structural overviews of three emulators for the Stanford EMMY are presented. These are IBM System/360; CRIL; and DELtran. Performance estimates are included for each of these systems.

CONTENTS

BASIC CONCEPTS

'These are the best of timings,
these are the worst of timings'

This is a preliminary report on the general structure of various emulators for the Stanford EMMY. It is too early to provide firm space and time measurements (indeed, it may be that programs will have to be measured pragmatically to obtain solid engineering data); however, planning estimates are included.

Emulation is the implementation of an image machine by mapping the states of this image machine into substates of a given host machine (in this case, the Stanford EMMY), and then programming the host machine to perform state transitions over this substate as required by the image architecture. Three image architectures will be discussed:

1) IBM System/360 -- A general-purpose register-oriented image machine.
2) CRIL -- A Polish Suffix, stack-oriented image machine whose design is tuned to a specific source language.
3) DELtran -- An experimental "mixed" image machine combining many of the desirable features of both a stack-oriented and a register-oriented architecture.

The emulators described here are all 'Class B" in terms of the following criterion:

Class A -- transforms all image substates precisely as would a true image machine (i.e., duplicates failure modes).
Class B -- transforms all image substates corresponding to "correct programs" as would a true image machine (i.e., may fail differently).
Class C -- transforms a selected subset of image substates as would a true image machine.
Class D -- transforms a selected subset of image substates in a manner such that the behavior of a true image machine could be predicted.
Class F -- bears little or no relation to a true image machine.

This level precision is not actually required for the global purposes of the Stanford Emulation Laboratory. The final classification of these emulators may drop to Class C, or even Class D, when they are actually tested on a working EMMY host.

This report will be updated to reflect significant changes, additional data. and hard results as they become available. The substance of the work to date, however, should be useful in understanding the nature of emulator structure and planning specific, comparative experiments on the yet-to-be-realized EMMY lab system.

SYSTEM 360 EMULATOR

INTRODUCTION

This section describes a Class B S/360 emulator for the Palyn EMMY. Based on EMMY CPU timing and the code developed thus far, performance should be approximately that of a model 50. A timing summary is included in the appendix for both Model I and Model II control store.

The reader is assumed familiar with both the Palyn EMMY [1], the IBM System/360 [3], and the IBM S/360 Model 50 [5].

The emulator implements the 360 basic instruction set with certain extensions (available as options on certain 360 models).

1) Address Specification Interrupts
   - no boundary restrictions are enforced since the memory controller performs all necessary alignment. Timing estimates assume optimal alignment.
2) Optional Features
   - Storage protect not supported and storage key field of PSW ignored.
   - Decimal feature not supported - decimal instruction will result in an operation specification exception.
   - Floating Point feature not supported at this time. Floating Point instruction will result in an operation specification exception.
   - Extended Control and Dynamic Address Translation not supported.
3) Internal Timer not supported. Main Storage locations X'50' thru X'52' have no special significance.

4) Timing estimates are included for EMMY CPU with Model I and Model II Control Storage. In both cases, optimal Main Storage operand alignment is assumed.

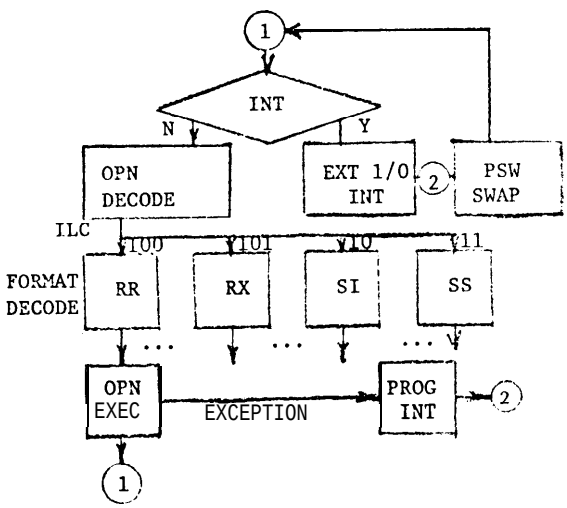
Based on IBM studies [4], about 20 instructions account for 70% of the instruction usage in 360 instruction streams. It was therefore decided that these instructions should be as fast as possible. A reasonable optimization of these and certain VFL instructions should result in Model 50 performance.

PHILOSOPHY

Image instruction execution consists of three phases, Operation Decode (DECODE), Format Decode and Effective Address Calculation (EAxx), and Execution and Prefetch (OPxx). In addition, frequently used functions, such as condition code setting and exception testing for arithmetic operation, are handled by common routines. These appear in line for the RR-format instructions to allow these to achieve the fastest possible speed.

The five classes of 360 interrupts are handled by a common routine. This minimizes microstorage overhead.

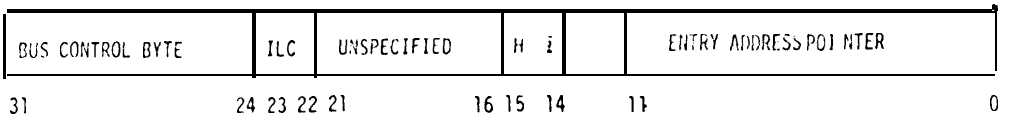
I/O is handled in the same manner for both low and high speed devices. Initially, all I/O will be to the Data Point. As more devices are added to the bus, I/O can either be mapped by the emulator, or by the Datapoint. Handling of low speed devices may also be done on a word by word multiplexed basis, with the EMMY CPU acting as a channel (it may also be necessary to handle high speed devices this way).



As devices are added to the bus, it may well be desirable to implement device controllers around microprocessors, to allow some degree of device emulation with lower CPU overhead.

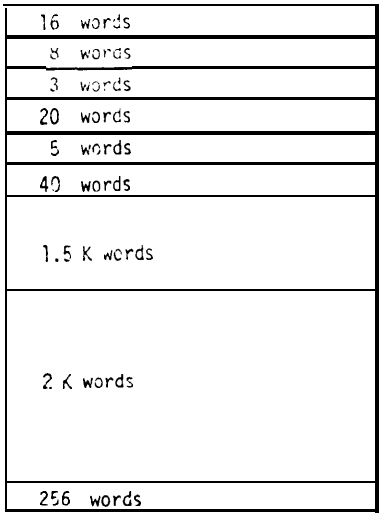
ORGANIZATION OF MICROMEMORY

The first 16 words of Micromemory are the 16 general registers of 360 architecture. The next 8 words represent the 4 floating point registers (floating point is not yet implemented). The 360 PSW is stored in Micromemory in decoded format. The high order half remains as it appears in mainstorage upon retrieval. The low order half is parsed as follows:



31:24 Initial BUS Control byte for Main Store accesses
23:22 Actual 360 ILC
21:16 unspecified
15:14 must be *zeros*
11:0 Semantic Routine entry point

FIGURE 3. Semantic Pointer



360 General Purpose Registers
360 Floating Point Registers
Program Status Word (decoded)
IPL and Interrupt code
Operation Decode
Format Decode

Semantic Routines

unused

Semantic Pointers

FIGURE 4. Micromemory Usage

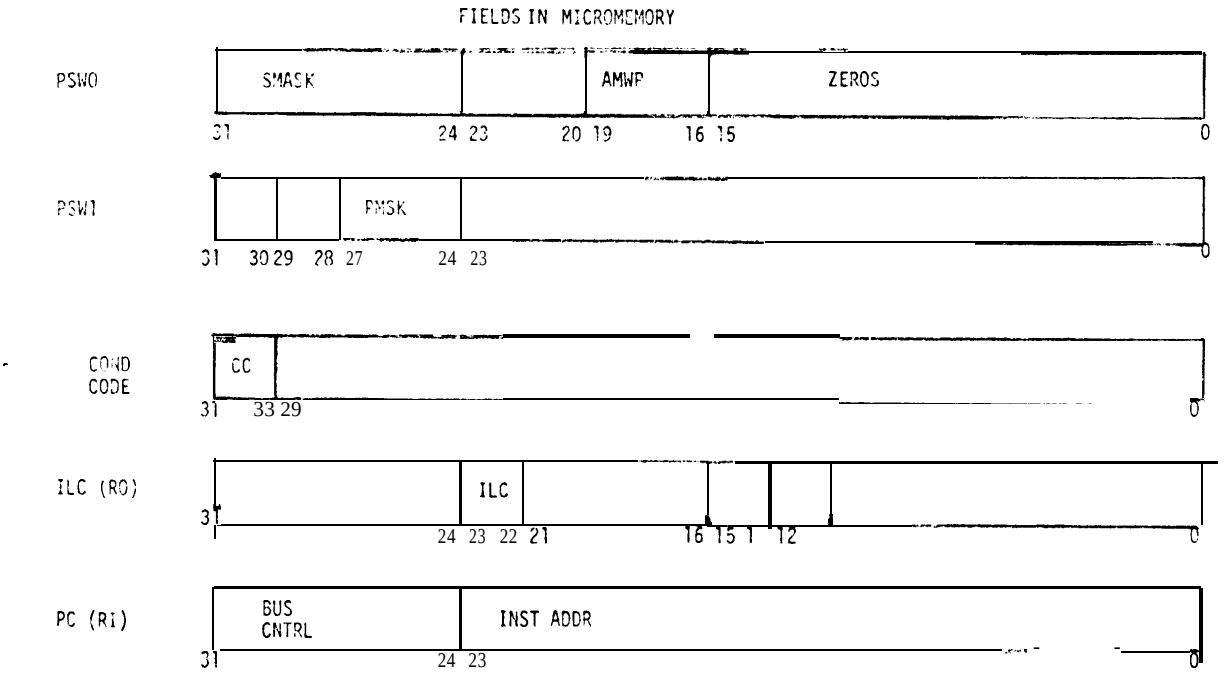
Next Instruction Address--mainstorage address of the next target instruction in host register 1 (R1-PC).

360 condition code (CC) occupies its own word in micromemory, as does the Program Mask.

ILC is stored in host register Ø (RØ=MAR).

The high-order 256 words of control store contain pointers to the entry points of various execution routines. Bits 31-12 of these pointers are not used to address microstore, and may contain information pertinent to each specific execution sequence. The emulator code resides in the low-order part of the block of micromemory between these storage areas.

It is assumed that the image machine will be interrupted infrequently. Therefore, the extra overhead required to reformat the 360 PSW at each interruption is acceptable.



FIELDS IN REGISTERS

OPERATION

Instruction Prefetching:

Instructions are prefetched by the previous execution routine, thereby ensuring that instructions are immediately available to the Decode routine. The Conditional Branch instruction initially assumes no branch is to be taken, and prefetches the next sequential instruction upon entry to the execution routine. The branch target instruction is then prefetched. In this way, if it is determined that a branch should be taken, the penalty will be minimized.

The Decode routine performs two levels of operation decoding as well as initial parse of the 360 instruction. The first decode is that of the instruction length code. This is used as an increment to a 4-element transfer vector table. In this way, the proper routine to parse the instruction, update program counter, and calculate addresses will be selected. (Actually, the final entry of this table in the actual RR format decode routine, thus eliminating a branch and speeding the decode of this class of instruction.)

The second level of decode involves the selection of semantic pointer to the execution entry point from microstore. These pointers are stored in the high order 256 words of microstore. Since only the low order 12 bits of these pointers are actually used to address microstore, the remainder can contain special information pertinent to each specific instruction execution, such as actual instruction length code and memory control bytes. Loading of the host MAR with this pointer sets the I-codes. The actual ILC for each instruction is set in this way.

Format decoding proceeds with as much generality as possible. This requires some functions to be repeated in several execution sequences, however, the simplification of the decoding process seems a worthwhile tradeoff. Predefined holding registers contain pertinent portions of the parsed and decoded 360 instruction. Exit from the Format Decode is a register transfer to RØ, the micro control register.

An interesting tradeoff was made in the RR format decode. A 500 ns reduction in execution time (4.3 ns to 3.8 ns) was achieved by doing a partial

decode in the RR decode routine. Parsing and Program Counter updating is complemented in the execution routine. This is done utilizing unused portions of already existing microinstructions in the execution code, resulting in a size reduction as well as the above mentioned speed increase. The same technique could not effectively be employed in other decode sequences.

Upon entry to the execution routine, a holding register still contains the semantic pointer word. In addition, the I-codes of RØ have been loaded with the information in bits 21-16 of the semantic pointer. Useful information from the holding register is usually retrieved immediately, and this register used as a scratch in the execution sequence.

Again, execution will include a prefetch of the next target instruction.

CODING AND TESTING

Performance

Sample code for the Decode, Format Decode and certain representative instructions has been developed. Performance estimates, for a CPU with Model II *control store and no overlap or concurrency, indicates approximately Model 50 execution speed. The RR Format instructions are equivalent; AR instruction takes about 3.18 µs, while a model 50 required 3.75 µs. Rx format instructions are also somewhat slower; an Add Word required 6.30 µs, while the model 50 requires 5.50 µs. SS format instructions are significantly faster than Model 50, however.

Testing of the routines will consist of walking through each routine as it is coded to ensure the proper information is available at the proper times. In addition, individual segments of code will be tested on the machine for proper information transfer. Testing of the integrated Emulator will consist of the running of Benchmark software, significantly the PL360 monitor subsystem developed by McClure.

CRIL EMULATOR

INTRODUCTION

CRIL is an intermediate, executable text developed by ICL for the CORAL source language and existing line of ICL microprogrammable hosts. The CRIL image machine contains a dynamic evaluation stack (up to 17 elements deep) and 8 data sector base registers. Its machine language manipulates these resources directly, and is organized along the lines of a Polish Suffix notation.

Instructions are of varying length, and more than one instruction may be packed into a single word of program store. Instructions are packed from right-to-left (least significant bits on the right with respect to the normal order of execution. Fields within an instruction are ordered right-to-left with respect to the normal sequence of interpretation. Each instruction explicitly indicates whether the next instruction is to be Fetched from a new word in program store (Fetch mode), or is contained in the Currently obtained program word (Continue mode).

The first instruction in a program word begins six bits in from the right (least significant) end of the word. Bits from other packed instructions may wrap around the left (most significant) end of the instruction word into these six low-order bit positions.

Finally, there are a number of error conditions which must be checked for when the CRIL machine is running in a special "Debug-Trace" mode. These conditions are transparent, however, when running in a normal "Production" mode. In terms of our classification of emulators, then, the CRIL machine can be run in a Class A or Class B mode.

PHILOSOPHY

Image machine instruction execution proceeds in three phases: Operation Decode, Operation Execution, Next Instruction Fetch. The fetch of the next instruction is included, rather than the fetch of the current instruction, due to the nature of the CRIL sequencing rules. Since EMMY lacks a right-circular shift, it was necessary to convert the circular CRIL decode rules into logical right shift rules. The emulator nucleus developed so far depends on the following assumptions:

1) No program word is so packed that either:

   a) cyclic looping occurs within the word (i.e., all of the instructions in the word are Continue-mode, in which case a non-terminating loop will result since every CRIL instruction capable of modifying the Program Counter is Fetch mode).

or

   b) any bit is used as part of the encoding of more than one instruction (i.e., instructions do not "overlap").

2) All bits not part of an instruction encoding (i.e., 'unused') are low (zero).

3) Only Class B emulation (Production mode) is required.

STORAGE ORGANIZATION

All internal CRIL resources are mapped into EMMY's Micro Store as indicated in the table below:

CRIL Resource	EMMY Resource
(8) Addressing Registers	Micro Store, words 0:7
(17) Evaluation Stack cells	Micro Store, words 8:24
Top-Of-Stack Pointer	Micro Register 7
Ra operand	Micro Register 4
Rb operand	Micro Register 5
Program Counter	Micro Register 1
Instruction Register	Micro Register 3

Additionally, Micro Register 2 is used for general indexing during all phases of instruction execution, and Micro Register 6 is used as a "Pre-Fetch" all for the next image program word.

CODING TECHNIQUES

Since CRIL instruction fields are interpreted naturally from right-to-left, beginning six bits from the low order position of an instruction word, each Fetch cycle starts by:

1) Moving the program word containing the current stream of CRIL instructions into the image instruction register (IR) while fetching the next program word into the prefetch register (PF) and updating the program counter register (PC).

2) Duplicating this program word in the indexing register (XR) adjacent to the instruction register (IR).

3) Right aligning the low order bit of the first CRIL instruction in the current program word by using a shift left (double) to simulate a shift right circular (single).

Each n-bit field extraction is subsequently achieved by:

1) Isolating the low-order n bits of the instruction register using either an EMMY Extract or And instruction.

2) Shifting the instruction register right (single) n bits.

Operation decode consists of a field extraction followed by a relative branch forward by the value of the field.

PERFORMANCE

The entire emulator, including I/O, should fit easily into the available 4K Micro Store. Typical execution times for "simple" operations range from 3 to 6 μs on the Stanford EMMY (l.t-2.5 on a production EMMY). This is certainly cost-effective in comparison to existing CRIL emulators which run at 15-30 μs per instruction and require more costly host support: Exact instruction timings, as well as a preliminary version of the emulator itself may be found in [6].

DELtran EMULATOR

INTRODUCTION

DELtran is an intermediate, executable text language developed specifically for evaluating BASIC FORTRAN on the Stanford EMMY. Its design follows the general precepts set forth in [7]. The DELtran image machine contains a dynamic evaluation stack (number of elements memory-limited), and a program-dependent number of Randomly Accessable "register" cells. Each of these storage resources can be manipulated directly by DELtran instructions.

Instruction units are of varying lengths, and more than one instruction can be packed into a single word. Also, instruction units can extend across program word boundaries, although the individual syllables of which they are constructed must lie entirely within a single word (note: an n-bit syllable with m trailing zero's may be packed into the low order n-m bits of a program word).

RAM cells are accessed indirectly through a table of dynamic descriptions (or SCOPE), and may refer to either scalar or array variables. Parameter-passing is achieved by coping dynamic descriptors for the actual arguments into the space reserved for the dynamic descriptors for the formats (e.g., a true call-by-reference).

PHILOSOPHY

The execution of a typical DELtran instruction unit takes place in three phases:

1) Lead Syllables parse/decode--during this phase, the general type and lexical format of an instruction are determined;

2) Reference Syllable parse/decode--during this phase, the actual operands of an instruction unit are recognized and a "standard interface" established for the next phase;

3) Operator Syllable parse/decode/execute--during this phase, the image machine state is transformed according to the DELtran operator specified by that instruction and the dynamic values in the interface established by the previous phase.

DELtran syllables are encoded so that the zero-value means "fetch a new program word" and frequently-used interpretations are assigned codes with trailing zeros. This promotes efficient packing of syllables into program words.

DELtran resources map into the host machine as follows:

DELtran Resource	EMMY Resource
Evaluation Stack cells	Micro Store; anchored at high end, grows toward low end.
Dynamic descriptors (Scope)	Micro Store; anchored just above emulator code, grows toward high end.
Top-Of-Stack pointer	Micro Register 1
Current Program Word	Micro Register 3
Operand 1 Value (p)	Micro Register 4
Operand 2 Value (q)	Micro Register 5
Result Location (r)	Micro Register 6
Addressing State (s)	Micro Register 7
Program Counter	Micro Store, word 0

Additionally, Micro Register 2 is used for general indexing operations, and Micro Register 7 contains auxiliary DELtran State information.

CODING TECHNIQUES

A double-shift left operation is used to move the next syllabel in a DELtran instruction stream from the Current Program Word into the indexing register. Reference Syllables are then checked for validity by decrementing the index register and testing for a negative value (it will be positive or zero unless the syllable value was zero originally). If the resulting value is equal to minus

one, a new program word is fetched, and the entire operation repeated. Semantic Syllables are checked for zero-value through indexing into a "jump table", each element of which is an entry point into the routine for the code corresponding to the offset of the element. In this case, the zeroth element in the table is the entry point to a routine which fetches a new program word.

PERFORMANCE

The basic emulator, excluding the evaluation stack and Scope areas, will require less than 800 words of Micro Store. Allowing 1.2K words for dynamic data, the emulator should require about 2K words. Typical instruction units will execute in 6 to 10 µs, however, this does not indicate the comparative performance of DELtran to 360 machine code. To perform a comparative analysis the ratio of DELtran instruction units to 360 instructions, as generated by a BASIC FORTRAN compiler, must be taken into consideration. Preliminary estimates based on very small fragments show that one DELtran instruction unit is worth at least 3 or 7 360 instructions (functional surrogate fragment), and possibly 8 to 10 360 instructions (process control surrogate fragment).

On a "production" EMMY, with a 100 ns Micro Store and 30 ns internal cycle, the DELtran machine would execute instruction units in 4 to 7 µs. If an additional one or two Micro Registers were available, this would drop to 2.5 to 5 µs.

REFERENCES

1. Neuhauser, Charles, "An Emulation Oriented, Dynamic Microprogrammable Processor (Version 3)", Stanford Electronics Laboratory Technical Note #65, October 1975.

2. Neuhauser, Charles, An Emulation Oriented, Dynamic Microprogrammable Processor Version II, Hopkins Computer Report #281, Johns Hopkins University, Baltimore, MD.

3. IBM, IBM System/360 Principles of Operation, Order No. GA22-6821-8, November 1970.

4. Connors, W.D., Mercer, U.S. and Sorline, T.A., S/360 Instruction Usage Distribution, TR 001 2025, System Development Division, IBM, Poughkeepsie, N.Y., May 8, 1970

5. IBM S/360 Model 50 Functional Characteristics, Order No. A22-6989-Ø.

6. Palyn Associates, Inc., "A Preliminary Study of CRIL on EMMY", Sept. 1975.

7. Hoevel, Lee, "Languages for Direct Execution", Proceedings of the 7th Annual Workshop on Microprogramming (SIGMICRO 7).