SU-SEL-75-047

# HIGH PERFORMANCE EMULATION

by

Walter A. Wallach, Jr.

November 1975

Technical Report No. 102

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 75-047

# HIGH PERFORMANCE EMULATION

by

Walter A. Wallach, Jr.

November 1975

Technical Report No. 102

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University
Stanford, California

# High Performance Emulation

by Walter A. Wallach Jr.

Technical Report No. 102
Stanford Electronics Lab
Digital Systems Lab
Stanford, California 94305
November 10, 1975

## Abstract

The Stanford EMMY is examined as an emulation engine. Using the 360 emulator and the DELtran interpreter as examples, the performance of the current EMMY architecture is examined as a high performance emulation vehicle. The problem of using a sequential, vertically organized processor for high speed emulation are developed and discussed.

A flexible control structure for high speed emulation studies is derived from an existing high performance processor. This structure issues a stream of microinstructions to a central command bus, allowing user-defined execution resources to execute then in overlapped fashion, These execution resources may be added or deleted with little or no processor rewiring.

## 1.0 Evaluation of an Existing Machine

Several existing processors claim to be "Universal Host" machines. While all have their strong points, none is particularly well suited to high speed emulation. Most of these machines have been designed for low to medium performance production work, or as research tools where speed is not essential.

At present, one of the most promising architectures available is that of the Stanford EMMY [I]. EMMY is a fixed architecture, vertically organized, dynamically microprogrammed processor, It consists of three machines or phases- I-machine, T-machine and A-machine. The operation of these machines is essentially sequential, with some overlap of nicronemory refresh.

The I-machine controls microinstruction sequencing and fetching, the T-machine performs data transformations between file regfisters, and the A-machine transfers data between memory resources.

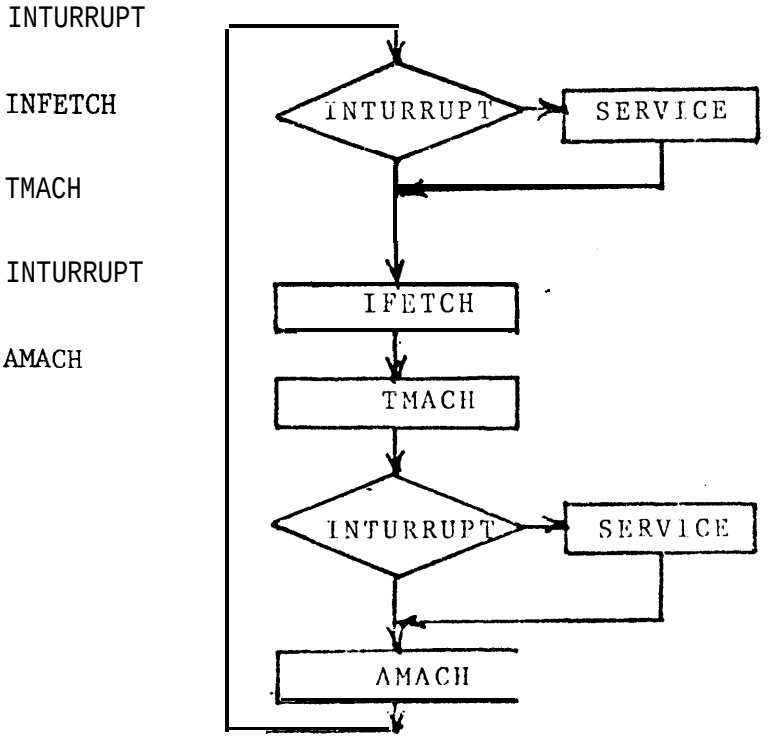



Figure 1.1 Control Flow in EMMY

The prototype version at Stanford has a 35ns internal cycle and 200ns control store cycle. tlicroinstructions require between 280 and 1250ns to execute, with a typcal instruction average being 455ns [3]. A proposed printed circuit version would feature an internal cycle of about 30ns and a micromemory cycle of 65ns, giving a significant overall average performance improvement,

Several emulators have been developed for this architecture [2], and performance estimates have been evaluated [3]. A System/360 emulator run on the Stanford machine processes 360 instructions at about 100 thousand instructions per second (KIPs). The proposed printed circuit version should achieve about 143 KIPs.

Other microcode has been develped to interpret certain meta languages called Directly Executed Languages (DELs). A DEL is essenially an instruction set taylored to interface a particlar higher level language such as Fortran or Algol, to the actual execution hardware. The effect is to emulate a Virtual Fortran machine" or "virtual Algol machine" [ 7].

A Fortran DEL (DELtran) is under development. Typical DELtran operations execute on the Stanford machine in between 4 and 10 us (roughly eauivalent to 360 instuctions emulated on the same machine). This is in excess of 150 thousand Fortran statements per second. However, these figures are misleading. Processing a DEL at 150 thousand statements per second could be equivalent to processing traditional machine code at 1 HIP, depending upon the efficiency of compiled code.

## 1.1 Characteristic Structure of Emulators

Development work on EMMY has revealed several inportant points. Foremost is that a great deal of work is still to be done in the area of host machine architecture. As the first architecture specified by emulator writers, rather than hardware designers, EMMY achieved many of its goals, notably ease of use and flexibility. EMMY demonstates respectable speed; we know of no other universal host machines that emulate 360 code at model 50 rates.

A great deal of the image instuction cycle is spent parsing and decoding image instructions. For the 360 emulator, this accounts for better than half the execution time of all except variable field length instructions. Additionally, the parsing of some fields is independent of instruction format, and often several fields may be processed simultaneously. In a sequential machine, however, this is inpossible; only one microinstruction can be executed at any one time.

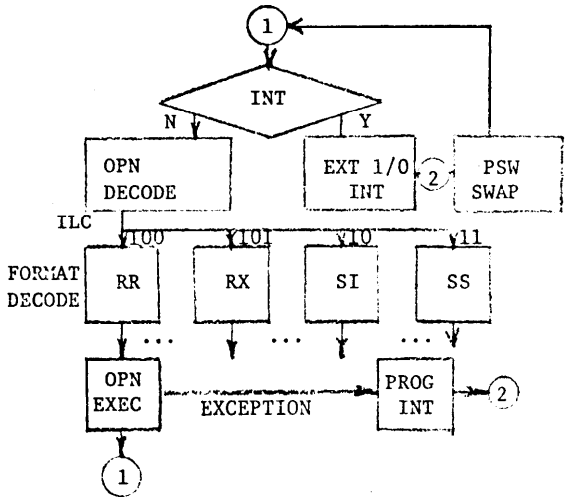



Figure 1.2 Control Flow in Emulated 360 Instructions

Eased on emulator code characteristics, it is possible to overlap the decoding of one instruction with the execution of the previous instruction. Concurrent main memory operation allows some overlap of instruction processing and next instruction prefetch in currently availably machines, however this is not enough to achieve high performance operation. Processing image instructions at one NIP requires an average instruction executon of less than one microsecond, If two instructions are being processed concurrently, the average execution times could approach two microseconds,.

A significant problem with existing Universal Host machines is that architectural changes require costly rewiring of the CPU. Execuion resources cannot be easily added or deleted. Resource requirements must be anticipated and included in the original design as an integral unit. Attempted changes require extensive redesign.

## 2.0 A Look at Existing High Performance Organizations

The fact that high speed emulation can be achieved is already established. The IBIJ System 370 Model 168 processes instructions within the desired range. It is not, however, a Universal Host, being uniquely well mapped to the 360-370 architecture, but not very well suited to general purpose emulation,. It consists of functionally independent modules coupled through a rather tight control structure. These modules include an instruction fetching, parsing and decoding element, various types of execution elements, such as fixed point, floating point, and variable field length, and a memory control element.

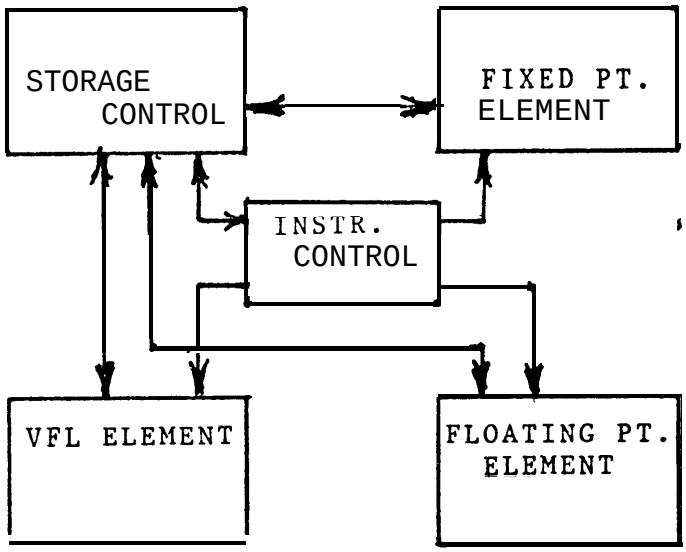



Figure 2. 1 Block Diagram of IBM 370/168

Each micro-order essentially establishes an entire operation. Thus by properly overlapping the operation of the various elements, the instruction processing rate MAY approach the micromemory access rate. The key is asynchronous operation of the functional elements. By naximizinp activity overlap, execution resources are fully utilized, Asynchronous organization, however, introduces certain new problems, such as resource conflict and sequential dependencies.

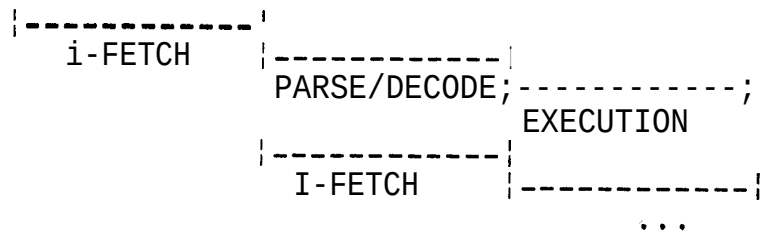



Figure 2.2 Resource Execution Overlap

## 3.0 Control Structure for a High Speed Emulation Vehicle

In order to achieve high speed emulaton at realistic cost, it is essential that many micro-resources operate concurrently. Individual resources need not be excessively fast, but will be fully utilized,

It is desirable to be able to modify the host processor easily; e.g., adding of special mapping hardware at the micro-level to enhance emulation of complex image machines, Resources found to be inadeauate can be deleted or improved. (This is particularly desirable, since the resources reauired for high performance emulation are not yet clearly defined). The flexability of an easily modified host processor enhances its utility in a variety of environments, including the study of f'ail-soft architectures, secure architectures, and mulitprocessors.

The organization proposed here consists of a central control structure, called the C-machine, which issues microinstructions to a. common command bus. User defined execution modules scan this bus and latch recognized microinstructions into internal control registers, executing the specified operation while the next microinstruction was fetched and desplayed on the bus. These modules may range from a simple shifter to a complex, microprogrammed machine. Local storage resources will include 'writable control store, a register file, a flag register (FR), a residual control facility (RCR), and an instruction register (IR).

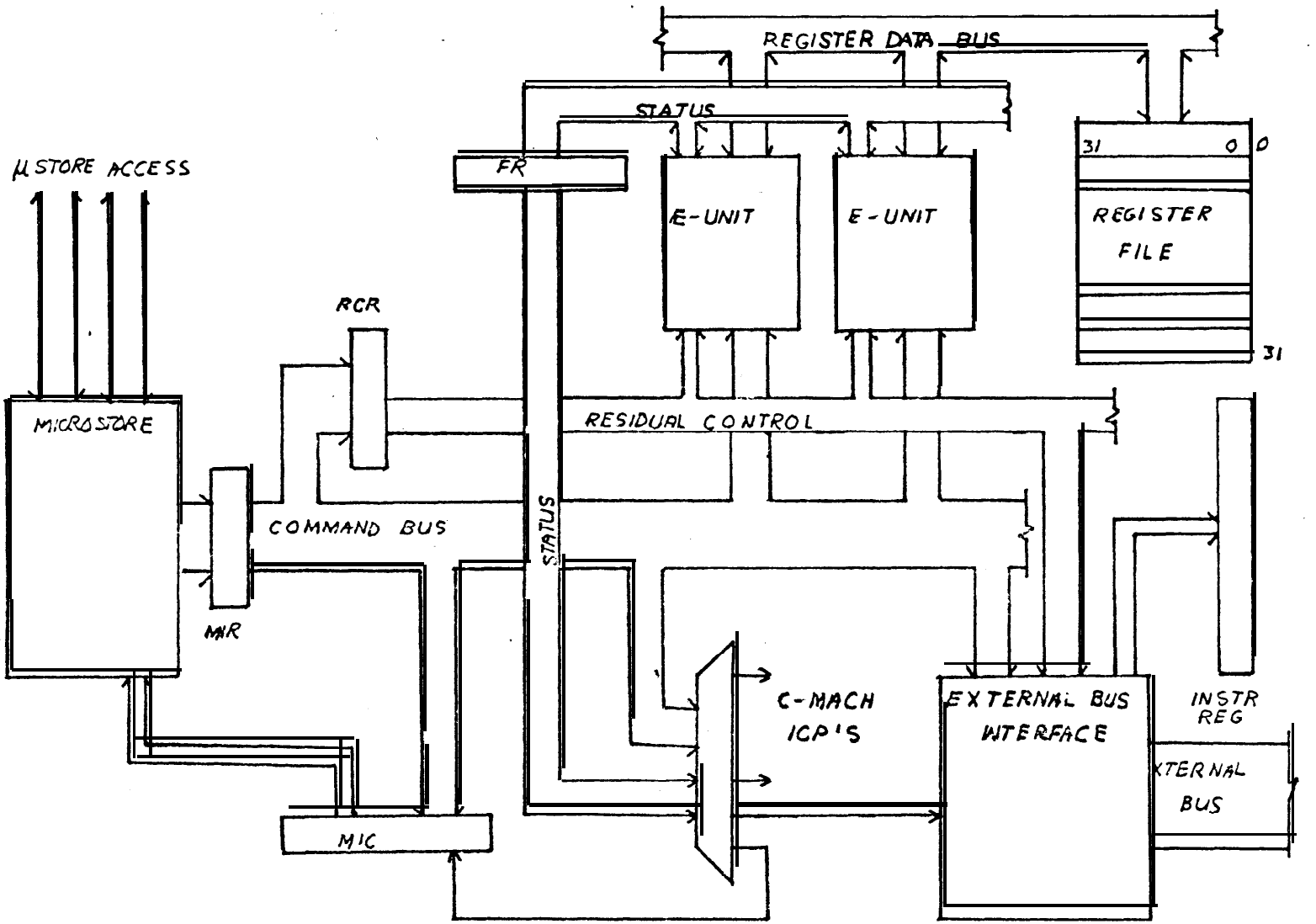
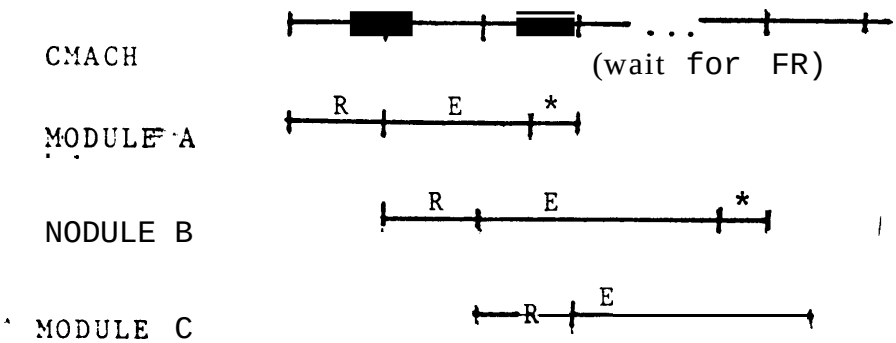



Figure 3 MICROMACHINE LAYOUT

## 3.1 C-machine and llicroconmand Bus

The C-machine is responsible for sequencing, fetching, and issuing microinstructions. It also initializes the residual control facility, though this could alternatively be performed by a user defined module, In the absence of branch instructions, the C-machine fetches and issues a continuous stream of microinstructions to the bus. The speed of this stream is limited only by the cycle tine of the microstore, In the event that a branch is encountered, both the target and the next sequential instructions trill be fetched while the branch test proceeds, thereby minimizing performance degradation.




R= recognition E-execution cycle. *=set FR bit

Figure 3.2 Microinstruction Execution Overlap

An execution module will have a tine equal to one micromemory cycle to recognize and latch its microinstruction, Execution will proceed concurrent with the next instruction being displayed on the bus, Thus, several module executions may be overlapped. The execution latency of each nodule nay be known to the microprogrammer, or may be data dependent. In the event that these latencies are unknown, a flag register bit is used to indicate execution completion. The C-machine, after starting several operations, will then wait (by looping on the current microaddress) until all operations are complete, indicated by

appropriate FR bits being set.

## 3.2 Branch Instructions

Several microinstructions wi11 be recognized by the C-machine. These are primarily branch-type instructions. The first will specify a mask to be compared to the FR, and a nicroaddress. Four test options can be specified, branch if selected bits all zero or all one, donot branch if selected bits all zero or all one. This is useful in synchronizing several nodules and performing iterative operations.

A second branch instruction will cause the C-machine to wait until some specified event occurs, such as "all modules finish execution? This differs from the branch in that no alternate microinstruction need be fetched and no alternate address need be specified.

In order to allow alternate microinstruction fetching, nicrostore nust be interleaved. The microprogrammer must also take care that the target instruction fetch nay proceed in parallel with the fetch of the next sequential instruction, Interleaving several ways will be required to prevent micronemory data accesses from degrading performance,

Interleaving will certainly aleviate some of the problems associated with access conflicts, Both high and low order interleaving will be required, along with certain programming practices, to ensure that access conflicts donot significantly afect performance. Simulation studies will be required to determine the required degree. If use of the register file keeps simultaneous microstore access requests to a minimum, the decree of interleaving can also be kept to a minimum,

## 3.3 Local Storage

Microstore provides a useful storage resource. Past experience with EMMY indicates that ample microstorage is available for both data and executable code. To keep conflict to

a minimum, it should not be the primary data resource, however.

The primary data source for the execution modules will be the register file. Therefore, a large number of registers will be required, larger than the current version of EMMY. Each execution module may have access to the entire file, or selected portions of it (at the user's option). Overlapping these portions provides another means of internodule communication and data transfer. Partitioning the file may also keep access conflicts to a minimum.

The file is implemented using several multiport register file chips, effectively interleaving this resource. Another way to view this is to say that several register files are provided, say four sets of eight registers each. An option of accessing 32 individual registers, 16 double length registers, or 8 quadruple length registers will provide a good multiprecision capabiliy. The length of individual registers must be kept to a minimum, with the basic register being 8 bits. (instruction fields to be parsed rarely exceed 8 bits).

Register resources can also be added as execution modules, then used as a register file extension, Such a module would have to include a port which other modules could attach to.

The functional specification of the residual control facility is not yet complete. It must contain bit vectors which will dictate instruction parsing, data path width, arithmetic precision, and so forth. The exact mechanism of control is still being formulated, however, and it will undoubtedly include unspecified control to be interpreted by each selected nodule,

The final host resource is an instruction register. Since the purpose of this processor is to fetch, decode, and execute image instructons, such a resource will clearly be needed. It was decided to include it seperate from the register file to allow the inclusion of an instructon parsing unit. This unit will place selected fields of the image instruction in predefined host registers. Decoding of these fields can be overlapped.. Operation execution will then proceed, overlapped, if possible, with next image instruction fetch and parse.

## 3.4 External Eus Interface Module

The processor will attaches to the outside world via an external bus similar to a Unibus R . Main Storage as well as peripheral devices will attatch to this bus. The host will communicate with the bus through an interface module,

The interface module, adressable as a microresource, will be responsible for starting I/O operations between external devices, such as disk to main storage, and between external devices and host local storage. It is be desirable to have this nodule be as powerful as possible.

The module will be able to perform block data transfers and data caching, Bit vector selection will be performed prior to data transfer to local storage. Three levels of data streaming are included to enhance image instruction fetching and operand processing when operands are word vectors.

It may prove necessary to implement some of this function in a main memory control unit. Providing it in the interface module, however, will be more in line with treating peripheral storage as a single level and letting a bus management device decide which storage medium contains a requested piece of data,

## 3.5 Microinstruction Format

In the processor configuration described so f'ar, the number and characteristics of the individual execution modules would be unknown when the control machine itself was desipned. In fact, this is essential to the flexibilty of the design. However, there must be a way to route microinstructions to the proper module.

Each microinstruction consists of a prefix code followed by control information. The nodule which is to execute the current instruction will recognize the prefix code. The use of prefix codes ensures that no module wi11 recognize another nodule's

identification code and control bits as its own id code. And, since prefix codes are variable length codes, modules which require long control words will have short id codes.

The width of the microinstruction bus has not been specified, nor has the microword length. Use of a 16 bit microinstruction bus and 32 bit microword offers some interesting possibilities, from the standpoint of cost effectiveness, Since fetching a microword provides two instructions, a slower (and less costly) microstore may be used, The microaddress counter can still address the actual 16 bit microinstruction, and branches be made to the second half of a microword, but such practice entails a penalty of one microcycle,

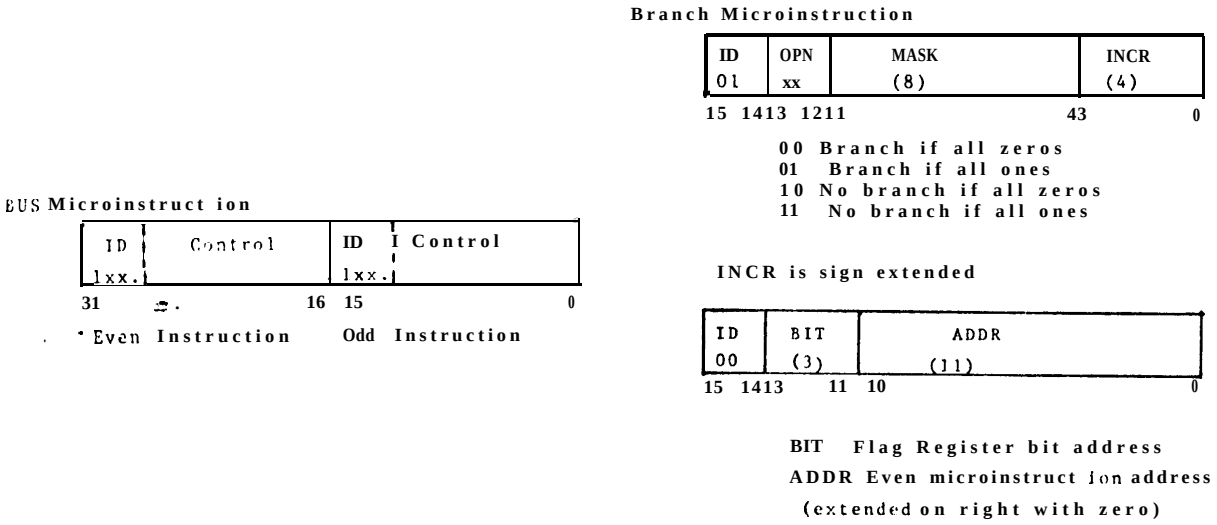



Figure 3.3 Microinstruction Formats

(Mote: a prefix code is a member of the class of variable length instantaneously decodable codes where no valid code word is the prefix of another valid code word. In other words, if a valid code word of n bits exists, no valid code word has these same n bits as its first n bits).

The specification of some execution modules will be made and these resources provided at the outset,. They may, at a later date, of cource be replaced. It is expected that, as more experience with this type of structure is gained, newer and more useful modules will be developed. These will undoubtedly surpass the original modules.

## 3.6 MicroMachine Control

Rather than include a maintenance console, it is desirable to emulate console functions on some type of programmable terminal. The console will be used to monitor emulator operation and diagnose faulty host operation. This arrangement has proven extremely powerful on the Stanford EMMY.

Support hardware for the emulated console feature will be implemented as a plugin execution module. This will provide access to all local storage resources (an additional means would have to be provided to allow access to the microadress counter),

## 4.0 Summary and Conclusion

The system outlined here provides a flexible, powerful means to control multiple execution resources. A straightforeward way in which these resources can be changed has been designed in. This provides a method for utilizinp special purpose hardware; such as mapping hardware and functional transformation hardware.

Depending upon the investment in execution resources, host performance will be in the .5 to 5 MIP range, This represents an order of magnitude increase in speed over existing host architectures, resulting from exploitation of parallelism and overlap in the interpretation process.

References


1. Neuhauser, Charles,"An Emulation Oriented, Dynamic Microprogrammable Processor (Version III)", TIJ #65, Digital Systems Lab, Stanford University, Stanford, California 94305 October, 1975.

2. Hoevel, L. W., and Wallach, W. A. , "A Tale of Three Emulators", TR # 98, Digital Systems Lab, Stanford University, Stanford, California 94305, November, 1975.

3. Wallach, W. A.,"System/360 Emulator Performance Estimate", Technical Note No.66, Digital Systems Lab, Stanford University, Stanford, California 94305, November, 1975.

4. Hoevel, L. W., and Wallach, W. A.,"Proposed Enhancements to EMMY", TN # 67, Digital Systems Lab, Stanford University, Stanford, California 94305, November, 1975.

5. IBDI, "S/370 Model 168 Functional Characteristics", Order No. GA22-7011.

6. IBM,"S/360 Model 91 Functional Characteristics", Order No. GA22-6907.

7. Hoevel, L. W.,"Languages for Direct Execution", Proceedings of Sigmicro 7, October 1974.