PUC

series: Monografias em Ciência da Computação
Nº 6/79

MICROPROGRAMMING III
TRENDS IN EMULATION TECHNOLOGY

por

Ayola N. Akonteh

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225 - CEP 22453

Rio de Janeiro — Brasil

PUC-RJ - DEPARTAMENTO DE INFORMÁTICA

Series: Monografia em Ciencia da Computação
Nº 6/79

Series Editor: Daniel A. Menascé March, 1979

MICROPROGRAMMING

Part III: Trends in Emulation Technology*

by

Ayola N. Akonteh

* This work has been supported in part by FINEP.

ABSTRACT:

This Part III of a series on Microprogramming focuses on trends in emulation technology. The study develops a formal representation of an emulation environment and defines an emulator in terms of a transformation process $\tau(x,y)$ equivalent to an environment M_{xy} created by imbedding the state image of a target machine y into a host machine x. A performance index ρ_{xy} of the emulator is developed to indicate its relative versatility. The rest of the study analyses the static and dynamic process of emulation and trends in emulation technology characterised by developments in semiconductor technology that have made possible the application of processor-memory-switch concepts in emulation.

KEY WORDS:

Imbedded image, environment, host machine, interpreter, microdes, microstores, microprogram, multiprogramming, performance index, processor-memory-switch, virtual machine.

ABSTRATO

Esta Parte III de uma série sobre Microprogramação tem como objetivo o desenvolvimento de tecnologia de emulação. O estudo desenvolve uma representação formal de ambiente de emulação e define um emulador em termos de um processo de transformação $T(x,y)$ que pode ser equivalente a uma ambiente M_{xy} criado através da implantação da imagem do estado de uma máquina emulada y dentre uma máquina anfitriã x. Um índice de avaliação ρ_{xy} do emulador é desenvolvido para mostrar a sua flexibilidade. A última parte do estudo analisa os processos estáticos e dinamicos de emulação e a direção de tecnologia de emulação caracterizada por desenvolvimento na tecnologia de semicondutores. Esta tecnologia já facilitou a aplicação de conceitos de "processor-memory Switch" no emulação.

PALAVRAS CHAVES:

Imagens Embutidas, ambiente, interpretador da máquina anfitriã, micromemoria, microprograma, multiprogramação, índice de desempenho, "processor-memory-switch", máquina virtual

I N D E X

MICROPROGRAMMING

Part III: TRENDS IN EMULATION TECHNOLOGY

3.0.0 Introduction

The importance of emulation arises from several software-hardware complementarity factors which from todays technology can be used to augment the performance of a given digital system. This section of the microprogramming lecture series first formalises the process of emulation, then traces growth trends in emulation technology and explores its practical applications. As an introduction consider the following propositions*.

"For a defined functional software process P_x there is an equivalent hardware processor x, and for a defined functional software P_y there is a hardware processor y".

Now, P_y can be transformed in several ways, using some process $\tau(x,y)$ (figure 3.1), to render it interpretable

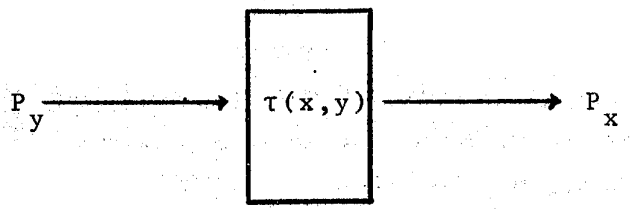

Fig. 3.1 - Transformation of P_y to P_x.

and executable by processor x. $\tau(x,y)$ is defined later in terms of an environment (M_{xy}) created by embedding the state image of the target machine y, into the host machine x.

---

* An analysis of this proposition can be found in Part I of this Lecture Series.

Among possible transformation processes for P_y, the following are interesting.

(i) - Reprogram and compile process P_y for processor x;

(ii) - Code-to-code translation and data conversion of program of processor y;

(iii) - Simulation of P_y (use software of processor x to simulate process P_y);

(iv) - Emulation of y (software-hardware simulation of y on x);

(v) - Use a processor y compatible with x.

Solution (iv) is of interest to this study on emulation.

3.0.1 - Definition:

Definition 1:

> Processor x (host) emulates processor y (target), if there exists a combined software - hardware restructuring of processor x such that any process P_y, can be interpreted and executed by processor x.

The resultant emulator E_{xy} is a representative state image of the target processor y, fashioned in software and hardware and imbedded in processor x. Each subsequent process P^k_{xy} executed by the resultant machine M_{xy} is a function of the combined instruction set $I_x \epsilon x$ and $I_y \epsilon y$ (interpretable by x). Thus the execution environment for a k^{th} process can be functionally described as

$$P^k_{xy} = \phi(I_x, I_y, M_{xy}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 3.1$$

$$k = 1,2 \ldots\ldots\ldots\ldots n$$

for $\forall I_x$, and $\forall I_y \notin I_x$

The machine M_{xy} can be obtained as follows:

(i) - Improvements in data flow of host machine x;

(ii) - Microprogramming implementation of other functions or extensions of existing instruction set in x;

(iii) - Interpretation of instruction set of target machine y;

(iv) - Other forms of modifications in the host machine x.

3.1 - The Concept of Emulation

Emulation as a concept is partially traceable to virtual machines defined in software. The actual implementation of emulation however, is an abstraction based on software-hardware equivalence principles implementable at the microprogramming (firmware) level.

Consider for example, a defined architecture x_o, by abstraction several configurations (family X) can be developed , each capable of executing some or all of the instruction sets of other members of the family X and of other processors y (of family Y). The basic processor x_o, is a virtual architecture (conceptual machine); an example of which is found in the IBM 360 which never existed in physical form but which was developed into the models 45, 58, 65 etc. Each physical processor $x \epsilon X$ developed independently, or through emulation of x_o, and used to emulate another processor $y \epsilon Y$ is regarded as a host machine and $y \epsilon Y$ (the emulated processor the target machine.

The emulator E_{xy} derived from a software-hardware intergration of y into x is functionally equivalent to $\tau(x,y)$ and representable as follows:

$$\tau(x,y) = E_{xy} = \phi(M_{xy}, I_{xy}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 3.2$$

where the instruction set I_{xy} for each emulator of the family can be defined as:

$$I_{xy} = \sum_{x,y \notin x} (I_x + I_y + \sigma_{xy}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 3.3$$

σ_{xy} represents software-hardware modifications in processor x to imbed the state image of processor y.

A relative performance index, the versatility and computational power ρ_{xy}, can be derived from equation (3.3) as follows:

$$\rho_{xy} = \sum_{x,y \notin x} I_{xy} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 3.4$$

Hence, for a defined processor x, ρ_{xy} can be determined from equation (3.4). ρ_{xy} as a performance index is bounded as follows:

$$1 \leq \rho_{xy} < N$$

For $\rho_{xy} = 1$ the machine is single purpose and specialised.

For $1 < \rho_{xy} \leq N$. The machine can be regarded as a multiple emulator (the ML-900 and B1700 offer this possibility).

As $N \to \infty$, the machine M_{xy} approximates a universal machine.

Microprogramming remains a useful emulation tool with a capability to restructure a given processors environment. RCA, Standard Computer Corporation, Interdata, etc have exploited the versatility offered by microprogramming together with increasing overlaps in processor architectures to emulate well known machines that include members of the IBM 360 family.

3.2.0 - Emulation and Simulation

In this study emulation is considered a software-hardware process different from a purely software simulation process. Functionally these two processes can be described as follows:

$$\text{Simulation} = f(P_x, P_y) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 3.5$$

$$\text{Emulation} = h(P_{xy}, M_{xy}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 3.6$$

P_x, P_y, P_{xy}, M_{xy} are as defined above.

In those cases of emulation where hardware modifications are required, compromises in costs, material, etc may result in a totally different machine. Thus, emulation goes a step futher than simulation in not reproducing only the behavioral characte - ristics of a given software process but also in modifying some aspects of the processor architecture.

Simulations normally consist of two parts:

(i) - An internal CPU program directed towards each target ma- chine simulated;

(ii) - A supervisory program to handle input/ontput and other co-ordinating activities.

The two parts are composed of several general purpose and specialised subroutines; among which are the following.

a) An interpreter that fetches, interprets ,and manipula- tes an instruction to obtain the right execution subrou tine in the host machine.

b) Execution routines not related to I/0.

c) Control programs to handle cases other than fetch , address computation and execution of simulated opera - tions.

Despite their usefulness as development tools, simulations have two major drawbacks:

(i) Lack of parallelism

(ii) Inability to exploit certain hardware features of a given ma chine.

3.4.0 - Interpretation and Emulation

Emulation has traditionally been used in digital systems as an interpretive tool. Hence, the words emulation and interpretation are functionally synonymous though their design and levels of implementation are markedly different. Interpreters are customarily software programs that reside in main memory, while emulators are generally a set of microprograms residing in control memory. Interpretation is formally defined below to shade light on its attributes.

Definition 2:

An interpreter is a system that executes a program in a representational frame by dynamically mapping each statement (instruction), at the point of its execution into an execution sequence of an environment that recognises the semantics of the mapped statement.

The emulation process, usually based on a dynamic mapping and execution of static processes, may consist of an environment described as follows:

(i) Data and control state images (working registers - accumulators, index registers, program counters, interrupt registers, etc).

(ii) A set of primitive operations that can be used to test and modify state images.

(iii) Control rules that sequence primitive operations based on current status of control state image.

Associated with each emulated environment are both static and dynamic processes, some of which are examined later.

3.4.1. - Dynamic and Static Processes of Emulation.

Emulation derives its dynamic characteristic from certain alternable processes, eg mapping, decoding, bit extraction, etc that constantly modify emulation environments. Static processes tend to be invariable though together with dynamic processes, they offer a degree of efficiency in emulation. In order to maintain such efficiency it is necessary that the following are possible.

(i) Data and static image of target (emulated) machine can easily be imbedded into the host (microprogrammed) machine x;

(ii) Decoding and control sequencing functions can be implemented efficiently in the host machine x;

(iii) Microinstruction semantics can operate on the imbedded state image of emulated machine y in same way that an instruction does on its state image.

Technological advances in emulation now incorporate certain generalising factors to these concepts which include the following.

* A generalised decoding structure;
* A reconfiguration of the state image, control structure and primitive operations of the emulation environment in a way that closely matches the emulated environment.
* A clean means of dynamically modifying microinstraction semantics based on parameters specified in the emulated instructions.

The net effects of technological advances in emulation is in recent microprogrammable microprocessors which now contain the following features.

(i) Flexible bit extraction and manipulation or generalised decoding.

a) Barrel shifters and mask capability (found in the B1700 and the FCPU)

b) Inserts of data in an arbitrary field of a register (FCPU)

(ii) Residual control as a means of reconfiguring the environment

a) Set-up gating patterns between registers and buses.

b) Set-up modes for arithmetic computation (e.g. As complement, BCD, etc - found in the B1700 and FCPU)

c) Set-up word length of data that will be applied to operations, memory access and store (B1700, FCPU)

d) Pseudo interrupt registers for imbedding control structure of emulated machines.

(iii) Microistructions as parameterised templates.

a) Indirect address of general registers, shift counters, ALU, etc (ML-900 of Standard Computer has these factures)

b) Execute commands (Found in the B1700 and FCPU).

3.4.2 Environmental factors of Emulation

An important curiosity in emulation is to determine a relevant environment for the emulator An analysis of the above advances, made in the area, shade considerable light on the environment factors (microprogrammed architecture) for general purpose

emulators. Some of these environmental factors can be summarised as follows:

(i) A primitive unit of information, the bit string;

(ii) Capability for dynamic reconfiguration of both internal and external environments of microprogrammable processors, ie word lengths, number of general registers, control structure, arithmetic modes, etc;

(iii) A capability for constructing complex address mapping functions;

(iv) A flexible and complex control structure for multiple interpretation, and general emulation. These features may include coroutines data flows, parallelism,etc or additional capabilities.

The present trend in programming languages towards a more complex structure (coroutines, data flow models, parallelism,etc) needs to be reflected in the architecture into which they compile; a fact that needs to be reflected in the emulation environment.

3.4.3 Microprogramming Techniques for Large Scale emulations.

Besides techniques for general emulation, several techniques now exist to facilitate microprogramming in large scale emulations. These techniques, for the most part, are based on unique characterisations of microprograms that include the following.

(i) High speed working registers - such a feature is needed for both control purposes and by the very nature of microprograms that are characterised by high speed execution.

(ii) Larger Control stores - lack of space has tended to reduce

. the efficiency of microprograms which in some cases have been limited to 256 words.

(iii) An N- way branch (case condition) - this offers an ability to test several conditions and branch to any section of that services then.

(iv) Micro-subrotine - ability to invoke a function or reference data indirectly at a lower level (a library call say).

(v) Memory Management -

Multiprogramming now a common practice requires more than an emulator within a control memory that contains device control programs. There are imminent problems of paging, memory protection, reallocation that are likely to pose same problems often met in higher level programming.

(iv) Microinterrupts - These are a useful tool in both a multi-emulation and multi-microprogramming environment.

As a lower level program implementation, microprogramming in large scale emulators benefits from resolved complex problems of higher level programming but at the same time inherits some of its deficiencies.

3.4.4 Implied Technological requirements of emulation

Historically, microprogramming (in emulation and interpretation) has relied on ROMs, which provide means of encoding and decoding of information; and on multiplexors to extract and assemble test conditions as well as select control information. A more advanced introduction has been RAMs (alterable read-only memories) which afford easy corrections , extensions and possible exchange in microprogramming functions.

However, increasing implementation of control at the microprogramming level, suggests a need for speed in large microstores. This can be derived primarily from larger and denser memories. A viable alternative is cache memory that combines a very fast primary store with slower secondany stores. Solid state disks may also be suitable because of their low latency which can be used in the fast swapping medium of demand paging. The growth in microcodes and and microstores will probably lead to other technological considerations. For example, multiprogramming environments that enable the emulation of several machines, the use of several microprogramming languages, etc.In all, there exist an apparent need for the following:

(i) Larger microstores (control memories);

(ii) Flexible microprogramming enviornment;

(iii) Easy manipulation of codes, bit strings, etc;

(iv) Management of a multiprogrammed environment ;

Progress made in these areas is examined together with how embodied concepts can be used to develope a new architectural philosophy for microprocessors.

3.5.0 Emulation Technology

One of the earliest formsof control technology is in the Whirlwind I, (built around 1950 at MIT), in wich control is encoded in a diode changeable memory array. The works of Wilkes (1952) and later Stringer have extended control to a systematic encoding which forms the conceptual frame of microprogramming.

Later, the coming of fast ROMs (1966) followed by RAMs (1970) made possible the design of processors with fixed instruction sets that permit more complex operations , (divide, multiply, double precision, etc.) The earlist use of this technology is in the IBM Systems 360 where most modes are implemented using control interpreters. The tendency from implementing the instruction set of one processor (emulation) is to implement that of several processors. IBM made this extension in implementing the 7090 on the 360/45.

Standard Computer Corporation (California) went a step further to build a machine that would interprete variable instruction sets (first in its IC Model 4 and later in the MLP-900) and attempted to implement the IBM 360 and PDP-10. These attempts failed possibly because of the level of details required in an emulation process and the available technology of the time.

3.5.1 Semiconductors Technology and Emulation

The key to developements in microprogramming (an emulation tool) seems to hinge on two important factors.

(i) A software representational freedom to enable the development of multiple instruction sets.

(ii) The state of semiconductor technology

The Standards MLP-900, the Saab FCPU, Burroughs B1700 and more recently Intel 3000 do show a degree of representational flexibility that facilitates development of other systems. Prior to the 50s high speed control stores used diodes matrix (whirlwind, and Wilkes Schemes) and integrated circuit technology in the mid-60s marked by the coming of the ROM and the RAM. IBM Engineers however, did not have access to this technology in the design of the System 360 and though the system 370 could have benefited from this technology, it did not. Today, there exist at least three (3) most significant advances in emulation technology among others less significant.

(i) Memories (microstores)- ROMs, RAM, PRAMs, etc.;

(ii) Programmable logic arrays (PLAs) and Field Programmable Logic Arrays (FPLAs);

(iii) Shifters.

3.5.2 Advances in Microstore

Though the bit count for control programs has decreased as a result of greater efficiency in microprogramming, microstores of today aré far more advanced than existed several years ago. For example, the Whirlwind I required 4800 bits of control store as opposed to 35000 bits in the PDP-11/40. Great advances in intergrated circuit technology (since the mid 60s) made possible first ROMs, then RAMs and presently EPROMs and PROMs with improved memory arrays. In a short time also (10 years) semiconductor manufacturers have doubled the number of devices per chip e.g. the 4-bit package RAM (1300 devices) introduced about 21/2 years after 1- K RAM (4000 devices). Progress in control stores implies improved performance of microprogramming (and hence emulation) and additional capabilities. ROMs and other microstores provide tables to encode, decode, sequence control, etc.

With recent advances in LSI and VLSI, there is the increased probability that microstores will be an influential factor in program implementation.

3.5.3 Programmable Logic Array (PLAs)

PLAs are two-level combinatorial logic circuits that are wired for a specific application by masking or metalization. The basic concepts in PLAs structure is the old minimization form of combinatorial digital circuits, though with outward characteristics likened to ROMs, The major difference between. PLAs and ROMs lies in the higher device count required per unit area of ROM to match PLAs in several applications.

Though promising, PLAs suffer several defects, some of which are as follows:

(i) PLAs are not dynamically alterable;

(ii) No natural addressing scheme exist for each of the make -or - break points in PLAs structure.

(iii) They offer no real advantage over ROMs in ilconditioned functions (e.g. purity testers). In decoding examples, it offers an advantage because these functions can be minimized.

The most apparent dynamic component substitute for PLAs is an associative memory, but it is not cost effective.

3.5.4 Shifters.

Shifting is an important feature in the interpretation (emulation) of variable data formats. Several advances have been made in the development of shifters with the signetic 8243 Shifter as one of the most significant. The Signetic 8243 takes 8-bit as an input and shifts left from 0 to 7 positions, zeroing out the rightmost bits. Large shifters can be constructed using the Signetic 8243 Shifter as a building block.

The ability to implement fast shifts makes variable length bytes extraction (an important feature in emulation) a much easier task. For example, fast shifters may be used to extract specific fields for interpretation or data of special formats such as floating point numbers.

3.6 Microcomputers in Emulation

Emulation using microcomputer architecture involves first an imbedding of the state image of the emulated computer y into the host microcomputer x. The embedded image should include working registers of y, its internal working state and memory system. An important factor in this form of emulation is a "Control Process" that activates a decoding tree that analyses each instruction to determine its appropriate semantic routine. The semantic routine then interpretes the instruction and depending on the results of this interpretation, returns to the control process to select the next instruction to be interpreted. This basic cycle (Figure 3.2) is traditionally known as DIL (Do Interpretive LOOP (Tucker, 1965)).

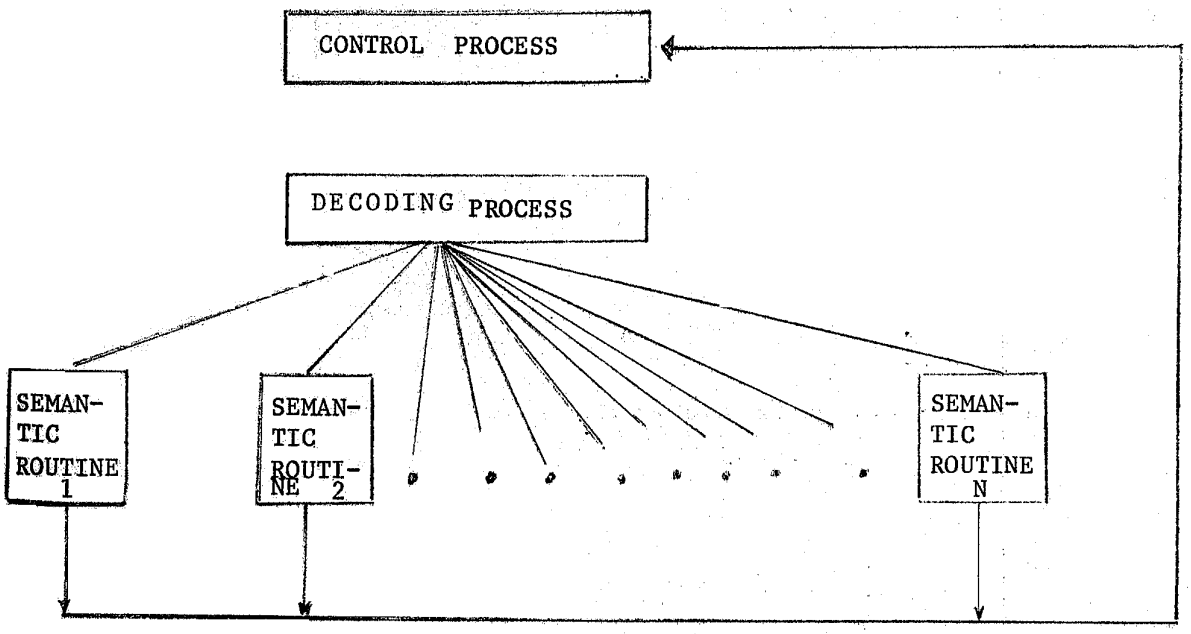

Figure 3.2 Do Interpretive Loop

3.6.1 The Emulation Environment

The emulation environment using microcomputers consists of embedding routines (of target microcomputer y), the control process, a decoding process, and an implicit extraction process. This commutative process is shown in figure 3.3 below.

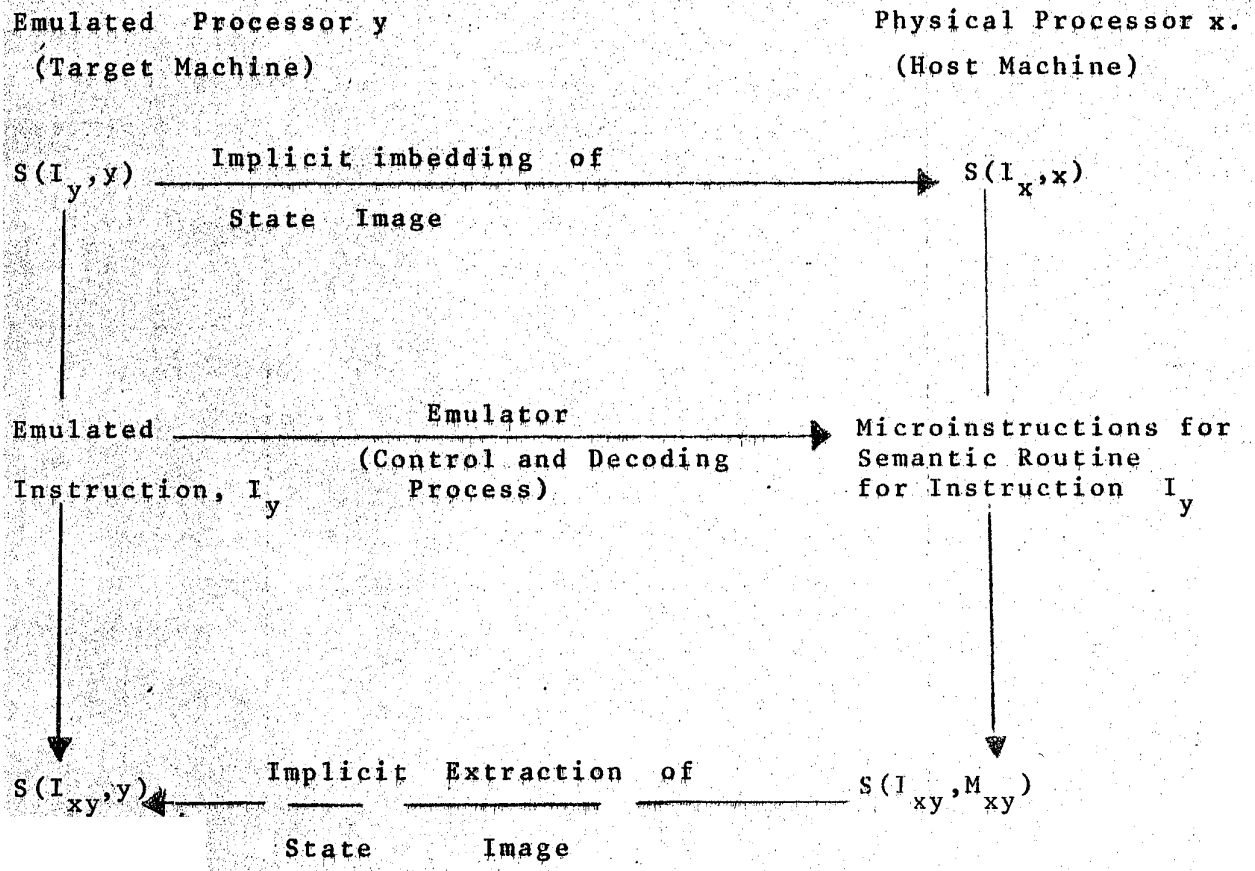

Figure 3.3 Commutative State diagram of Conventional Emulation.

In microcomputers designed to emulate specific computer architectures or families of computers, the embedding of the state image S_y of a target machine y is straight forward. This is more often so because of the following:

(i) Dedicated internal registers exist for holding commonly accessed state information on y.

(ii) The control and decoding processes are usually in hardware and receive data from dedicated internal registers.

(iii) The intergration of the decoding process into the control structure of the microprocessor creates an extended control structure.

However, microcomputers may prove inefficient when used to emulate machines with completely different instruction formats. For example, mapping a 24-bit instruction into a 16-bit processor with dedicated internal registers may prove infeasible. The QM-1 and the MLP-900 designed for general emulation offer advantage of a capability to configure non-specific internal registers of microprocessors and their interconnections into specific configurations. This capability for a flexible configuration is referred to as residual control which may lead to

* ease of systems representation;
* code compactness;
* efficient use of processor resources

The idea of reconfigurability uses a similar principle as virtual memories (residual control) both being attempts to structure the features of a computer to suit the problem.

Using the concept of Processor-Memory-Switch (PMS), Lesser (1972) suggested a new view to the process of emulation. This view is schematically outlined in Figure 3.4 below.

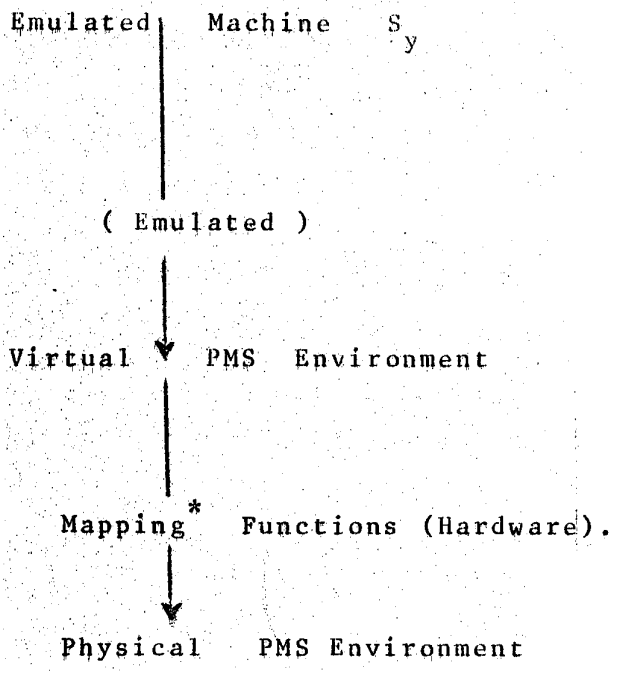

Figure 3.4 New view of the emulation process

* Similar to hardware in Virtual Memory System which manages page tables and programs mapping of virtual address to physical address.

The idea of PMS itself is an intergration of the concept of residual control, virtual memories and dynamic control (implemented though microprogramming)*

---

*Note:

(i) Residual Control as used here allows variations of the number of internal working registers of each processor

(ii) Virtual memory as used here allow for varying the structure of the memory for example memory size or word length.

(iii) Dynamic Control Structure allows for varying of the number of microprocessors and fucntional units, their interconnections and interactive patterns.

3.6.2 The concept of Virtual Processor-Memory-Switch.

The concept of virtual PMS environment leads to the new view of emulation illustrated in Figure 3.4 above. The virtual state image S_{xy} of the microcomputer system is created through a specification of a particular PMS environment with the additional level of hardware used to map micro-operations performed in the context of the virtual PMS environment on actual (physical) PMS environment. The virtual PMS environment provides extra representational dimensions that facilitate the emulation process.

A commutative state diagram of the emulation process using the concept of virtual PMS is shown in Figure 3.5 below. A specific virtual PMS is defined by defining an propriate control structure for a given microcomputer system. Such a structure is modifiable.

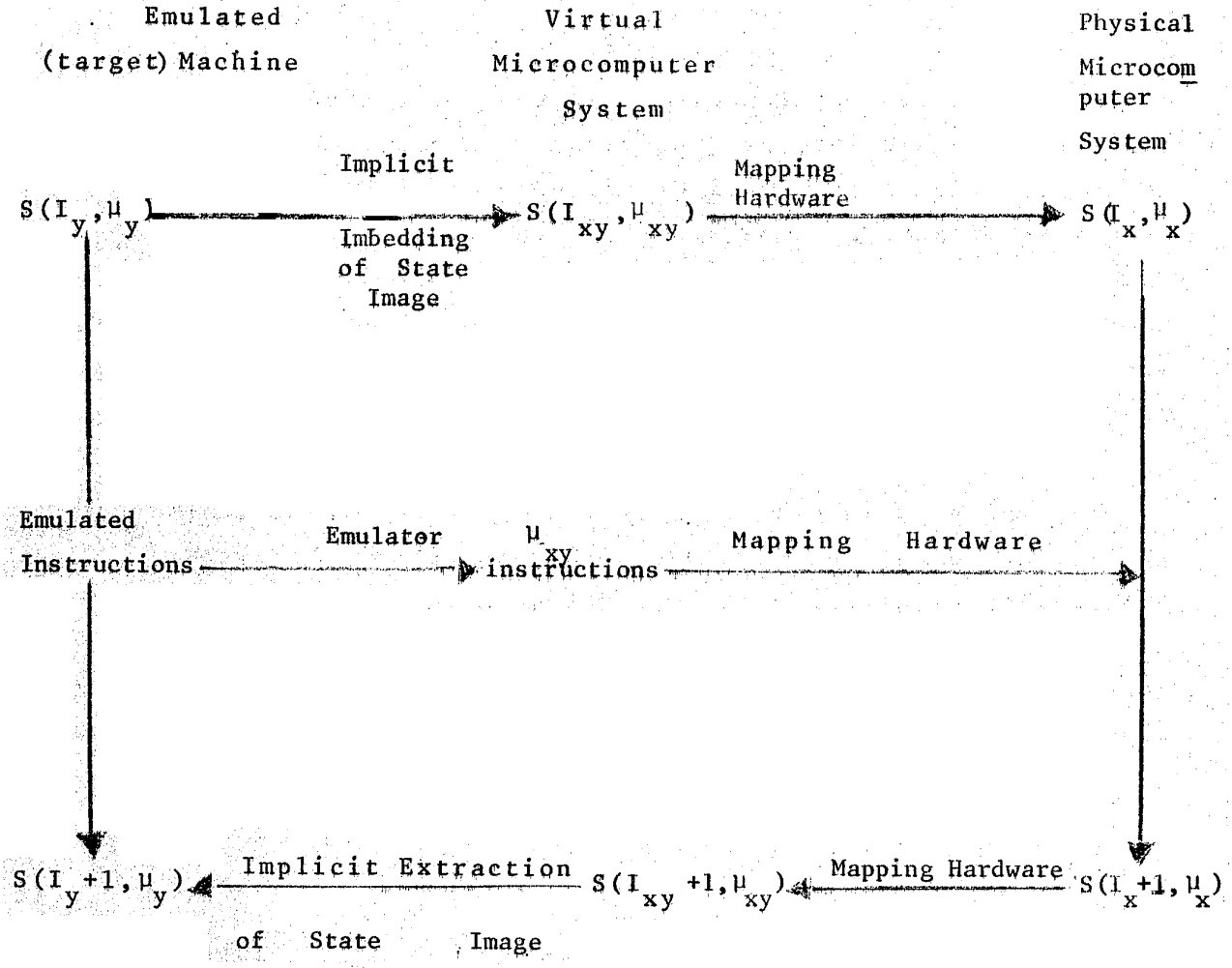

Figure Commutative state diagram of (virtual) Emulation.

to achieve particular results. The ability to modify the syntax for control is the key to tailoring a virtual PMS environment towards a particular emulated machine y.

CONCLUSIONS:

Rapid developments in semiconductor technology now offer the strong possibility of a universal processor or at least a multi-emulator based processor capable of executing the instruction sets of several other foreign (target) machines. Within such an emulated environment an alternate level of program implementation now exist through microprogramming, suitable for rapid service programs (operating systems, table look-ups, compiler routines, etc.).

The host machine (emulator(s)) can be evaluated through the performance index ρ_{xy}. More, the concept of PMS offers a flexibility in defining any emulation environment realizable with existing processor capability - a fact that seems economically attractive to create alternate digital systems through microprogram extensions of existing machines.

REFERENCES:

1. - Alexandridis, N.A. - Bit-Sliced Microprocessor Architecture
Computer, June 1978

2. - Fuller, S.H. et al - Microprogramming and its relationship to Emulation , Microprogramming and Systems Architecture
Infotech State of the Art Report 23

3. - Lesser, V.R. - Dynamic control structures and their use in emulation
Computer Science Report # CS 309
Stanford University, California 1972.

4. - Rosin,R.F. - Contemporary Concepts of microprogramming and emulation.
Computing Surveys, Volume 1, Number 4 (December, 1969)

5. - Tucker, S.G. - Emulation of Large Systems
Communications of the ACM Vol. 8, # 12 (1965).

6. - Wilkes, M.A. - Microprogramming principles and developments.
Microprogramming and Systems Architecture Infotech State of the Art Report 23.