PUC

Series: Monografias em Ciência da Computação
Nº 4/79

MICROPROGRAMMING: PRINCIPLES AND DEVELOPMENTS

by

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: Monografias em Ciência da Computação
Nº 4/79
Series Editor: Daniel A. Menascé

March, 1979

MICROPROGRAMMING: PRINCIPLES AND DEVELOPMENTS*

by

Ayola N. Akonteh

* This work has been sponsored in part by FINEP.

Abstract:

This is Part I of a series on Microprogramming. This part analysis developments in microprogramming from Wilkes (1951) through its commercial application by IBM (1964) to today. It recognizes the software/hardware duality and shows microprogramming as a tradeoff that offers an alternate and efficient level of program implementation. The rest of the study analysis the basic micro-instruction design, encoding, implementation and timing.

Key words:

Asynchronous polyphase, complementarity, duality, micro-programs, microsoftware, micro-operations, micro-instructions.

Resumo: Este trabalho é a Parte I da série de monografias sobre microprogramação. Esta parte analisa o desenvolvimento da micro-programação começando com o trabalho de Wilkes (1951) passado através a comercialização da IBM (1964) até hoje. Este trabalho reconhece a dualidade de software/hardware e mostra microprogramação como um compromissso que oferece um nível alternativo e eficiente para implementação de programas. O resto do texto faz uma análise dos projetos básicos de microinstruções, a implementação e "timing" deles.

Palavras chave:

Multifase assincrona, complementaridade, dualidade, micro-programas, microsoftware, microoperação, micro-instruções.

Table of Contents

MICROPROGRAMMING

PART I : PRINCIPLES AND DEVELOPMENT

The evolution of principles of microprogramming follows similar trends as found in higher level language programming. These trends include Microinstruction codification, timing, etc and their use in the development of microprogram controls. Generally derived principles differ mostly in the level and environment (ROMs,RAMs, etc., as opposed to main memory) of implementation.

1.1 -Foundations of Microprogramming.

Microprogramming as a concept evolved from a necessity to systematize the internal operations (register -to-register data transfers, signal control, etc.) of digital computers. Such a scheme inevitably calls for some tradeoffs including a departure from the traditional ad hoc hardwired logic control. Immediate returns from microprogrammed control include savings through shorter and flexible systems design formats that can be implemented through emulation (e.g. IBM 360/series) and more in material (less number of logic gates) that may further provide a higher performance. The added flexibility which allows users to alter the structure of a machine through microprogramming into a system with predefined features is not only revolutionary but may in fact lead to the realisation of a universal machine (see parts 0 and III).

Before tracing the historical development of microprogramming it is relevant to explore at an abstract level the philosophical implication of the process. Microprogramming is not defined at this stage since it would be both premature and lack the necessary background information necessary to appreciate such a definition. There exist a discernable, but ill-defined complementarity between software and hardware that suggest in part some duality in computation. This duality may be different for instance from the particle/wave, heat/mass, light/dark, etc duality but can rightly be called software/hardware

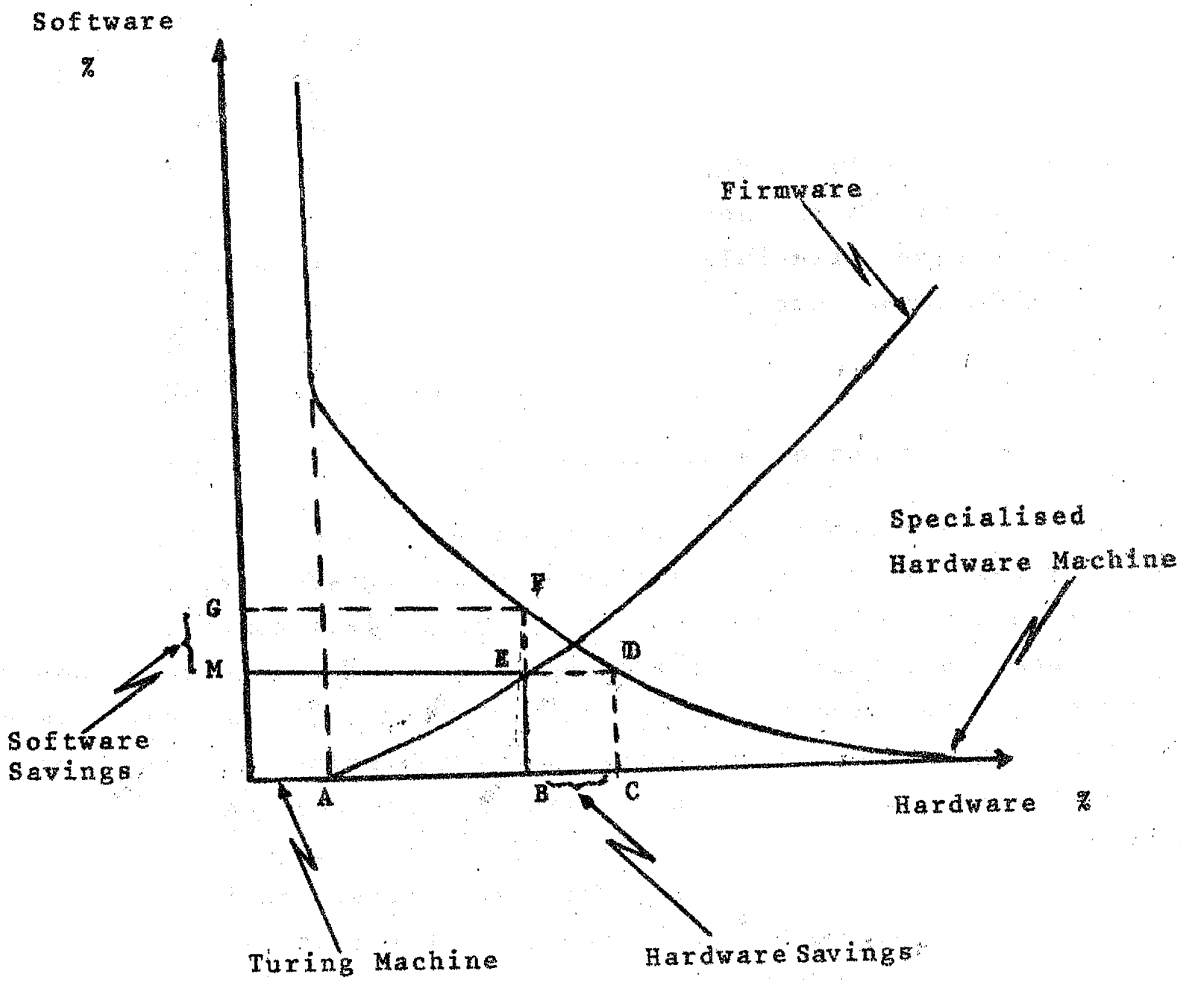


Figure 1.1. Conceptual frame of Software-Hardware duality.

duality. More formally this concept is stated below (see also figure 1.1).

Definition 1: For every defined functional hardware processor x, there exist an equivalent software process P_x, and for a defined functional software process P_y, there is an equivalent hardware Processor y.

The key to unlocking the form and characteristics of this dual existence of computational methods seems to lie in its transformation agent (microprogramming) whose characteristics are reflective of both hardware and software. Hence, it seems important to study microprogramming in light of software and hardware in order to understand and optimize them. The implications of the process are frightening because of the great flexibility in restructuring both software and hardware through instant rewiring of a machine effectuated through microprogramming. The realisation of this dream depends on the state of semiconductor technology, which for the moment poses no major problem.

1.1.1 - Trends in Hardware Developments.

The concept of a stored-program in a non-volatile Read-Only-Memory (ROM) was first proposed by Professor Maurice V. Wilkes of the Mathematical Laboratory (University of Cambridge) at Manchester University Computer Conference (1951). The original proposal called for a systemization of program control then accomplished through ad hoc random logic gates. This systematized control program Wilkes called a 'micro-programme' from which evolved the word microprogramming. Details on the Wilkes model as well as a formal definition of microprogramming are contained in Part 0 of this study.

Developments in microprogramming in the 1950s and early 1960s was confined to academic interest. Figure 1.2 shows in a very rough manner some of these trends and the degree of apparent reception of the concept. The upturn seems to have come with its industrial application by IBM when it announced the 360/Series in

1964 as a family of microprogrammed machines.

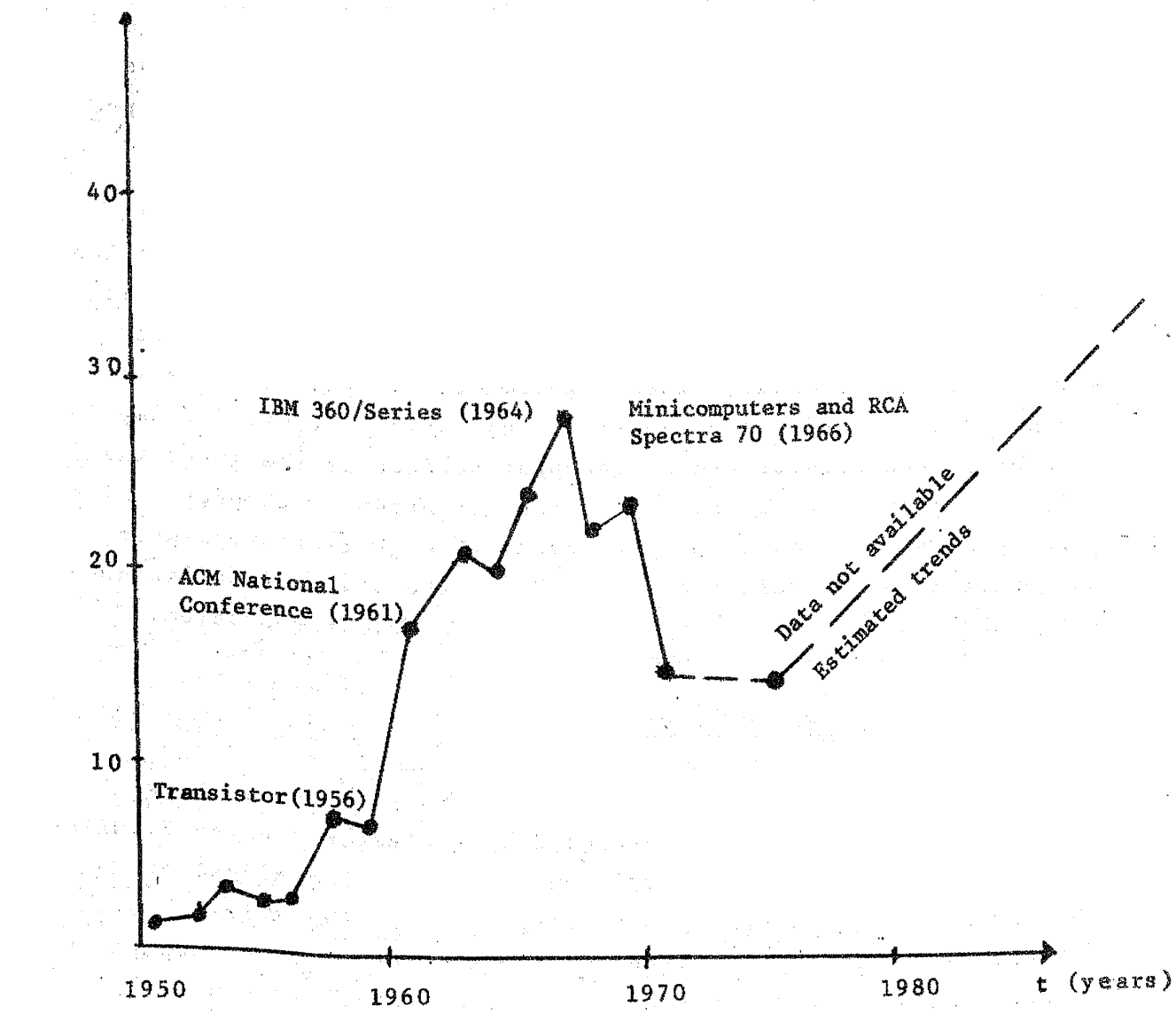


Figure 1.2 Trends in Microprogramming

The RCA announcement of the spectra/70 was perhaps a greater surprise because of the striking resemblance in its architecture and to the IBM 360/Series . Since these early developments several manufacturer have produced microprogrammed machines: (e.g. HP2100, Burroughs B1700, Digital Scientific Meta 4, etc).

1.1.2 - Trends in Software Developments

Though software developments have lacked behind hardware, there now exist substantial effort in developing microsoftware (Assemblers, translators, etc.) and in some cases higher micro-programming languages (for example Euler);developed translators (for example MLTG*) convert higher microprogramming languages to microinstructions (a sequence of bits) that are interpretable by a machine to effectuate an operation.

Some attempts have been made to design higher level microprogramming languages (Schlaeppi) with yet very moderate success. The subject of microprogramming languages is treated in Part II of this study. There exist interesting developments in the area of microprogramming;for a relatively short time, the field has attracted a lot of interest.

The period 1951-1959 seems characterised by research and lack of implementation. The first generation of microprogramming seem to be 1960 with the coming of the Read-Only-Memories (ROMs) and the second generation by the coming of Read-Write-Memories (RAMs) in the 1970s together with dynamic programming capabilities. The development of higher level microprogramming programs signals yet another distinct phase which can be denominated third generation microprograms. The forth generation shows a tendency to intergrate hardware and software through firmware. If this in fact ocurred, the level of function implementation would be optimized and dependent only on the efficiency afforded (e.g. operating systems may be totally implemented at the microlevel). A perhaps more interesting conjecture is the possibility of switching levels of function implementation (say from higher level language to microprogramming).

1.1.3 - Some Applications of Microprogramming

Several applications of microprogramming exist and are treated in detail in separate studies. However, some of the important areas of applications are summarised below. These include:

---

*MLTG - Microprogramming Language Translator Generator (Sawai, et al 1977).

1. Simulation and Emulation - very important integration of software-hardware to effectuate certain functions or reproduce the instruction set of other machines (e.g. Spectra/70 emulation of the IBM 360/Series) (see Part III of this study).

2. Signal Processing - Particularly useful in the area of sensor data processing, military operations, nuclear monitoring, etc; all of which may need real time execution.

3. Macrodiagnostics - Packages for microdiagnostics are particulary useful for field operations in testing faults on machines. Such packages should be simple and be independent of any hardware to enable use by field Engineers.

4. Communication systems Control - Particularly useful for telephone switching, telemetry decoding, data routing, etc. Can also be used to monitor real time applications of process control.

5. Control and Monitoring of Computer Systems - This is one of the most important applications of microprogramming and in fact the basis of its evolution. Certain architectural specifications (register sizes, intra registrar data transfers, CPU cycles, etc) can be implemented through microprogramming. Schedulers, macros, despatchers, interrupt handlers, etc can similarly be implemented. In recent days great effort has gone into optimizing compilers through microprogramming their repetitive routines.

1.2.- Organization of a Microprogrammed Computer

The microprogrammable computer organization consist of four major functional units:

. Memory Unit
. Arithmetic and Logic Unit.

. Control Unit
. Input/Output Unit

Some analysts consider Input and Output as two separate units. This is not important for this study and does not distract from the basic communication patterns realisable within different units of the computer through electronic signals. The orderly implementation of instructions, data flows, timing, etc. in a digital computer is the function of the computer control unit. The control unit may itself be directed(controlled) through nanoprograms or hardwired control logic. In controlling the operations of a computer, the control unit executes microprograms, contained in a control memory and each directed at realising some predefined macro-operation. Besides this, the control unit attends to several other control activities. The format of response by the control unit can be summarised as follows:

1. Determination of exact instruction to be executed.
2. Issue of control signals to open and close specific gates within the computer system - thus allowing only certains operations and data flows.
3. May order results to be restored in memory or other devices.
4. Issue an order for the next Fetch-Decode-Execute Instruction.

The above functions can be summarised in a general form as follow:

i) Fetch $R \leftarrow (M_i)$

ii) Decode

iii) $K_i \leftarrow K_{i+1}$ change control

iv) $S_i \leftarrow S_{i+1}$ change of state of the system.

1.2.1 - Logical Building Blocks

Some of the functional logic gates for the above operations include AND, OR, XOR, INVERT, etc; which can be used as building

blocks to build other functions (e.g. NOR,NAND,etc). Flip-flops, latches, triggers, etc which can be set (by the control unit) at each machine cycle, are also derivable from same basic blocks. A control gate may contain one or more input lines and has an output only when the signal is on. The following example is of a control gate..

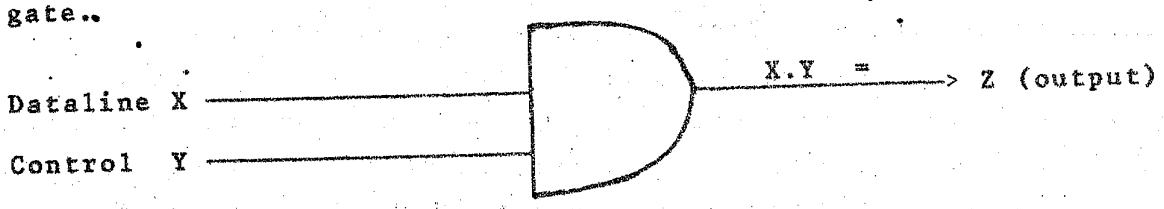


Figure 1.3. Basic Control (AND) gate.

Basically there exist two ways of implementing control in a digital computer - through traditional hardwired random logic or microprogramming. The following sections summarises some of the attributes of each of the two forms of control.

1.2.2 - Random Logic Control

Random Logic control denotes the ad hoc implementation of Computer Control using hardwired logical gates. Such connections could form either sequential or combinational networks that function like a finite state machine. There are several limitations to the use of random logic for control in digital systems most of which are examined in Part 0 of this study.

1.2.3 - Microprogram Control

An alternate design and implementation of control in digital systems is through microprogramming.*

Micro-operations steps in each time interval are representable by a control word characterised by 1's and 0's. These control words can be programmed to initiate operations in various components of a system in a defined way. Any control unit with its micro-operations steps stored in memory is called a microprogrammed control unit. A sequence of control words (microinstructions) is called a microprogram. Hence the definition of

---

* Defined in Part 0 of this study.

microprogramming. This concept has only become practical with advances in memory technology which gave birth to the read-only memories (ROM) used in storing micro-operation steps in digital systems.

Dynamic microprogramming is an associated concept which allows certain desired microprograms to be loaded onto control memory from either a console or auxiliary memory. Writable control Memories (WCM) are used in dynamic microprogramming thus giving it a writable (modification) capability though in practice it is used only for reading. Through dynamic microprogramming various types of microprograms can be brought into memory to effectuate different functions (e.g. Fortran, Algol, Basic, etc. to compile other programs) at appropriate intervals.

1.3 - Microinstruction Design

A microinstruction (control word), contained in a control memory, is made of a series of bits each of which denotes a specific control signal . Each control word is subdivided into a group of bits (fields) each of which field provides a distinct and separate function. In general, bits of a control word may provide one or more of the following functions:

1. Micro-operations* for control unit itself with specification of the next microinstruction address format;
2. Micro-operations for the registers and memory of the system (micro-operations may be divided into fields);
3. Address field for branch microinstructions;
4. Other special fields that may contain data for transfer to a specified destination (addresses).

---

* A micro-operation is an elementary operation, realisable in a clock pulse, on information stored in one or more registers with results replacing previous information of the registers or stored in other registers. Examples of micro-operations include (shift, clear, count, load, etc.).

1.3.1 - Address Mapping

Associated with each macroinstruction is an unconditional branch to a ROM address to execute a subroutine associated with its opcode. The address of the first microinstruction of the routine is a function of the opcode bits and the size of the ROM address register (points to specific control words in the control memory). The process by which the ROM addresses are determined for each microinstruction associated with a given machine instruction is called mapping. Figures 1.5 - 1.7 show a graphical representation of this process which in summary words is

Macroinstruction (Assembly Language Implementation) —unconditional Branch→ ROM Routine —→ Microinstruction (Address obtained through mapping) —→ Micro-operation

1.3.2 - Microinstruction Format

In general microinstruction formats vary a lot - the following though only illustrative is general enough to contain most of the fields normally considered in the design of a microinstruction

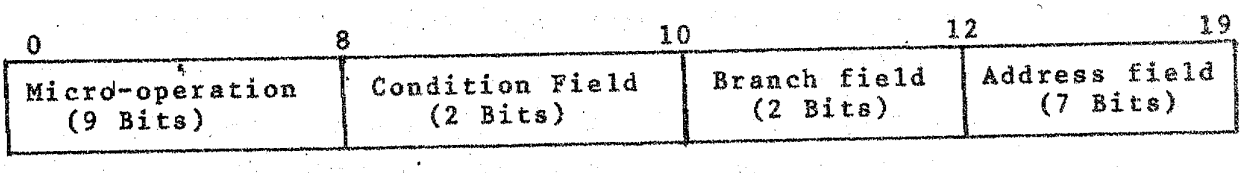


Figure 1.4 General Format of a microinstruction

The activities associated with each of these fields are summarised in figure 1.4. There exist several other forms of encoding of microinstructions, two of the most common forms of which are horizontal and vertical microinstructions encodings

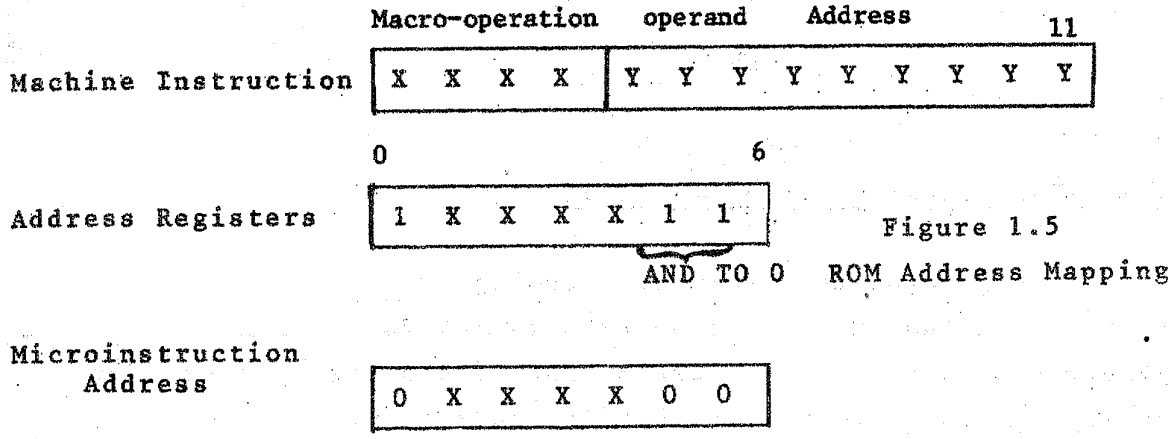


Figure 1.5 ROM Address Mapping

The mapping process can be summarised as shown in figure 1.6 belows.

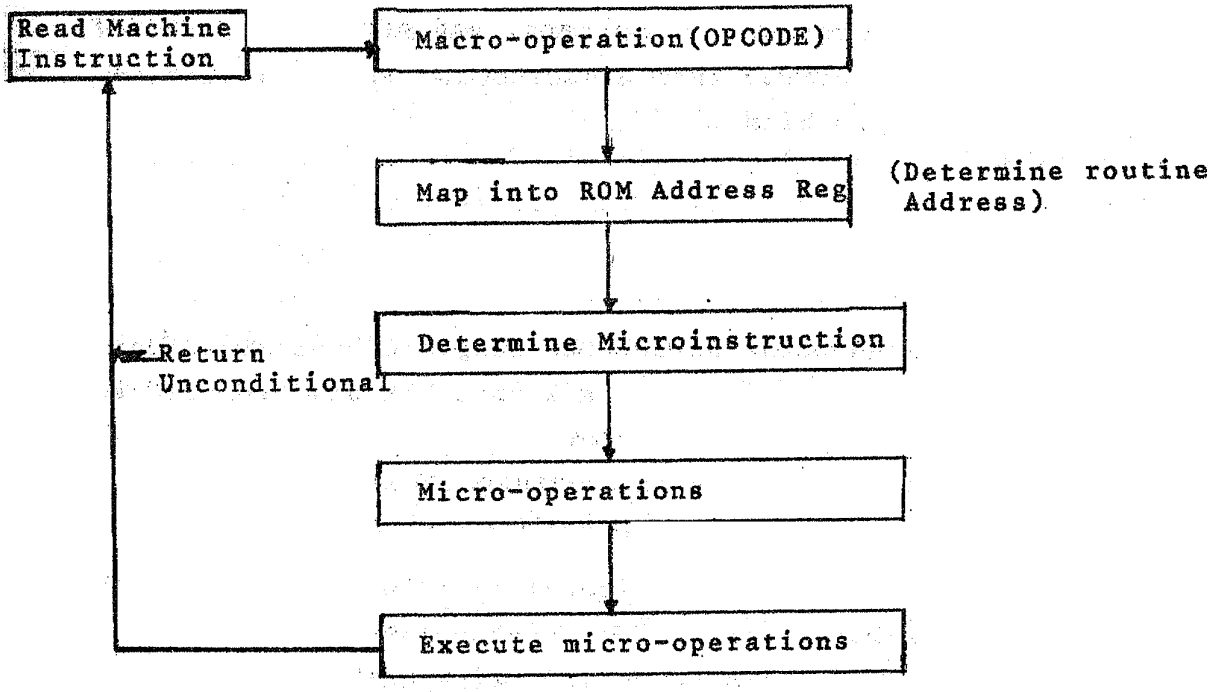


Figure 1.6 Summary of the mapping process.

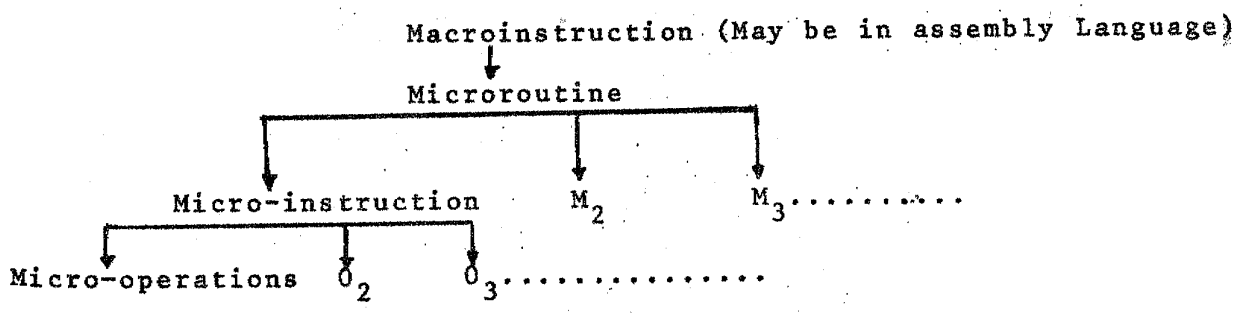


Figure 1.7 - Instruction Mapping Flowchart

1.3.3 - Horizontal Microinstructions

In general this form of encoding controls several hardware resources and thus is of a greater length than a vertical micro-instructions. Horizontal microinstructions normally represent micro-operations that are executed concurrently.

1.3.4 - Vertical Microinstruction

This type of encoding is similar to the classical machine instruction with an opcode, an operand, and one or more fields. It is usually much shorter than a horizontal microinstruction (varies from 12 to 24 bits).

1.3.5 - Other Microinstruction encoding schemes

Encoding patterns effect the length of the microinstruction. The most elementary case is where a 1 in a bit position represents an executable micro-operations (similar to Wilkes model where the presence of a bit represented the opening of a particular gate). Two other interesting levels of encoding exist.

In a single level (or direct) encoding, micro-operations that a single hardware resource can perform are encoded in a single field e.g. an ALU that can perform 8 different arithmetic and logic operations is encoded in 3 bit (2^3=8). This is shown in the figure 2.8.

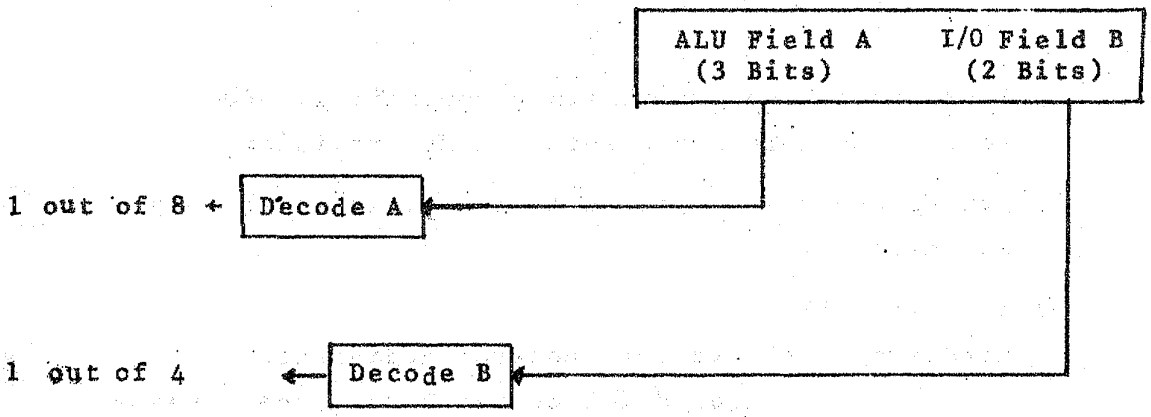


Figure 2.8 Single Level Encoding

In a two level encoding mutually exclusive microoperations maybe combined to form single level encoded fields. Such a process called bit steering allows any combination of fields to be decoded together by shifting the bit format. In such a case the bit interpretation is linked to the machine state e.g. Input/ Output. Encoding may reduce microinstruction size considerable; though at the expence of additional hardware (e.g. gates, latches, decoding nets, etc). Excessive encoding may also obfuscate machine operations.

1.3.6 - Microinstruction Sequencing

At least two sequencing techniques exist for systematically executing microinstructions (control words). One of this uses a microprogram counter (MPC) and the other a 'linked list' concept.

In the MPC approach, the address of a subsequent instruction is 1 greater than the current except under conditional execution. Conditional micro-operations may result from the following:

a) Structure of the microinstruction;
b) From the state of hardware.

The associated conditional addresses can be obtained as follows:

1) Micro-operation may contain a conditional address portion that is moved into the MPC register.

2) General purpose register that calculates the new addresses.

3) Some special memory location - Some machines use a hardware stack to save control addresses. Such a process would follow the traditional push and pop procedures initiated by interrupts or conditional branching.

In the 'linked list' approach, the address of the instruction is stored in the present microinstructions thus forming a chained list of addresses. Two implementations exist for this addressing format.

a) Include several addresses associated with certain conditions in the micro-instruction.

b) Special bits that could be set or encoded in one way for specific conditions (e.g. 4 bits would give 16 possible conditions).

1.4 - Factors in Microinstruction implementation

A known characteristic of microinstruction implementation is the time delays in its execution sequence. There exist at least two types of implementation - 1) serial and 2) parallel. They both indicate the degree of overlap in the execution phases of a current microinstruction and the fetch-decode phase of the next. Note that fetching next instruction involves the following

a) update of the microprogram counter, MPC ← MPC+1;

b) Selecting instruction pointed to in ROM by MPC is ROM(MPC) ;

c) Reading this microinstruction into the microinstruction.

register MIR, MIR ← ROM(MPC) ;

d) Decoding the executing micro-operations associated with microinstruction.

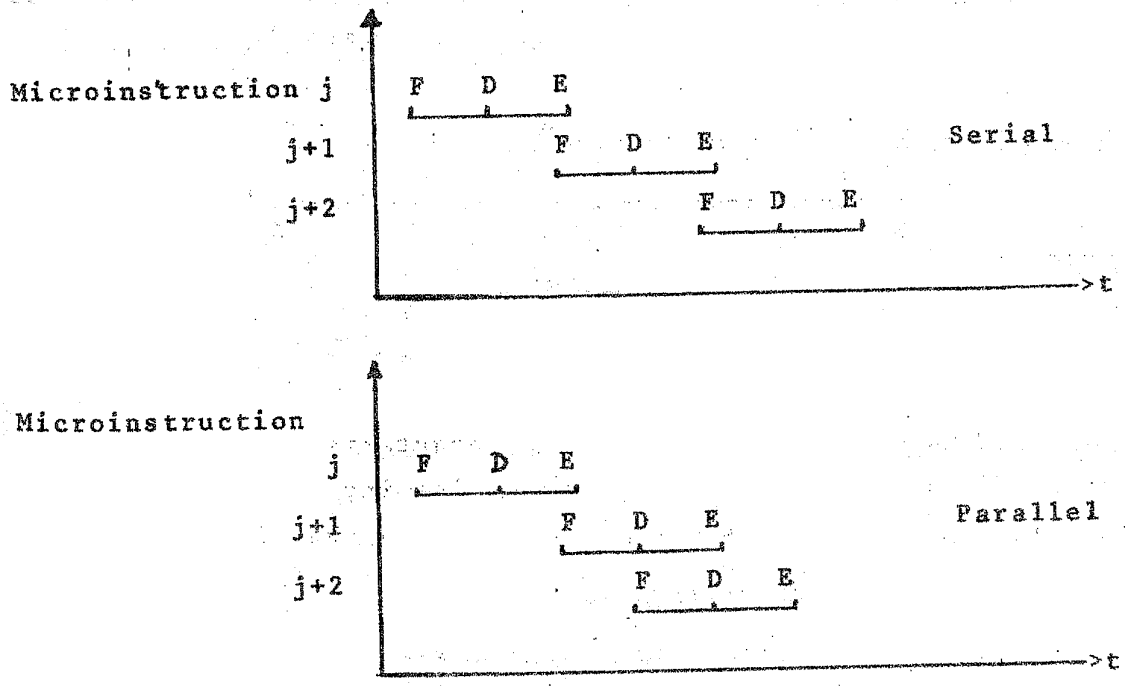
Signals and data propagation delays depend on the complexity of the hardware.

1.4.1 - Serial Implementation

This is a very simple implementation where the fetch phase begins at the execution phase of the present microinstruction. The advantage of the above realization is that no simultaneous fetch and execute exist and hence no hardware complications are likely to arise in either reading or execution of microinstruction.

1.4.2 - Parallel Implementation

In this form of implementation, the fetch phase of the next instruction to be executed is done in parallel with the execution of the current instruction. The advantage of this type of scheme is the saving in time. However, conditional branching may cause delays in determining the address of the next micro-instruction. In such a case a serial-parallel approach may be good.



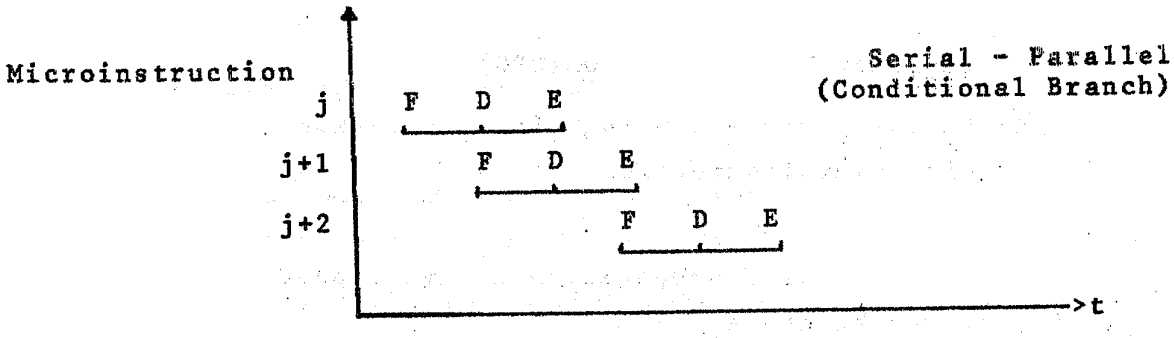


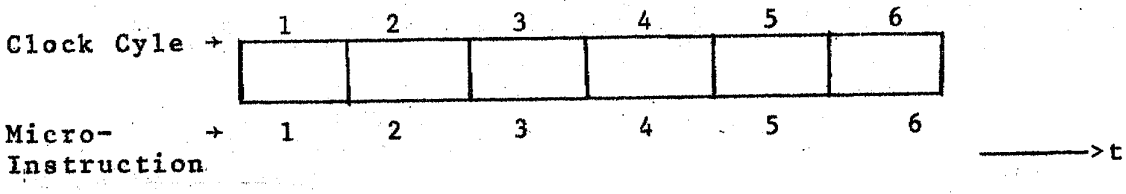
Figure 2. Characteristics of Serial-Parallel Implementation

1.4.3 - Monophase - Polyphase characterization of a microinstruction

The monophase - polyphase characteristics of micro-instruction refers to the number of distinct phases required to execute a given microinstruction.

1.4.4 - Monophase

In a monophase operation a whole microinstruction is effected by a (single) simultaneous issue of control signals. This type of operation is useful for simple operations such as simple data transfers which may involve only the opening of gates. The definite advantage here is simplicity of realization.



1.4.5 - Polyphase

Polyphase implementation is characterised by several distinct phases with the possibility of issuing control signals at each phase. There exist two types of polyphase implementation:

a) Synchronous Polyphase:

Microinstructions are executed in a single major clock cycle (which may contain several minor

cycles). The execution time for microinstructions is the same.

b) Asynchronous Polyphase:

The number of phases required to execute a microinstruction is a function of the complexity of the instruction. It has the advantage of possible parallel fetch but needs complex timing. Polyphase implementation is characterized by sequential logic.

Conclusions:

Microprogramming as an alternate level of program implementation offers promise for more efficient control of digital systems in the future. Moreover, it offers the best tradeoff for the software-hardware complementarity problem which will be an important design and economic consideration in future digital systems design.

Bibliography:

AKONTEH, A.N. — Trends in Emulation Technology Monograph
Dept. of Computer Science, PUC-RJ; 1979

BOULAYE, G.G. — Microprogramming
The MacMillan Press Ltda, NY (USA) 1975.

CHU, Y. — Computer Organisation and Microprogramming
Prentice-Hall, Englewood Cliff, NJ; 1972

HUSSON, S. — Microprogramming: Principles and Practices
Prentice Hall, Englewood Cliff, NJ; 1970.

KATZAN, H. — Microprogramming Primer
McGraw-Hill, NY, NY; 1977.

ROSIN, R.F. — Contemporary Concepts of Microprogramming...
Computer Surreys, Vol.1, No.4 pp 197-212
December 1969.

WILKES, M.V. — Growth of interest in Microprogramming
Computer Surveys Vol.1, No 3 pp 139-145
September 1969.