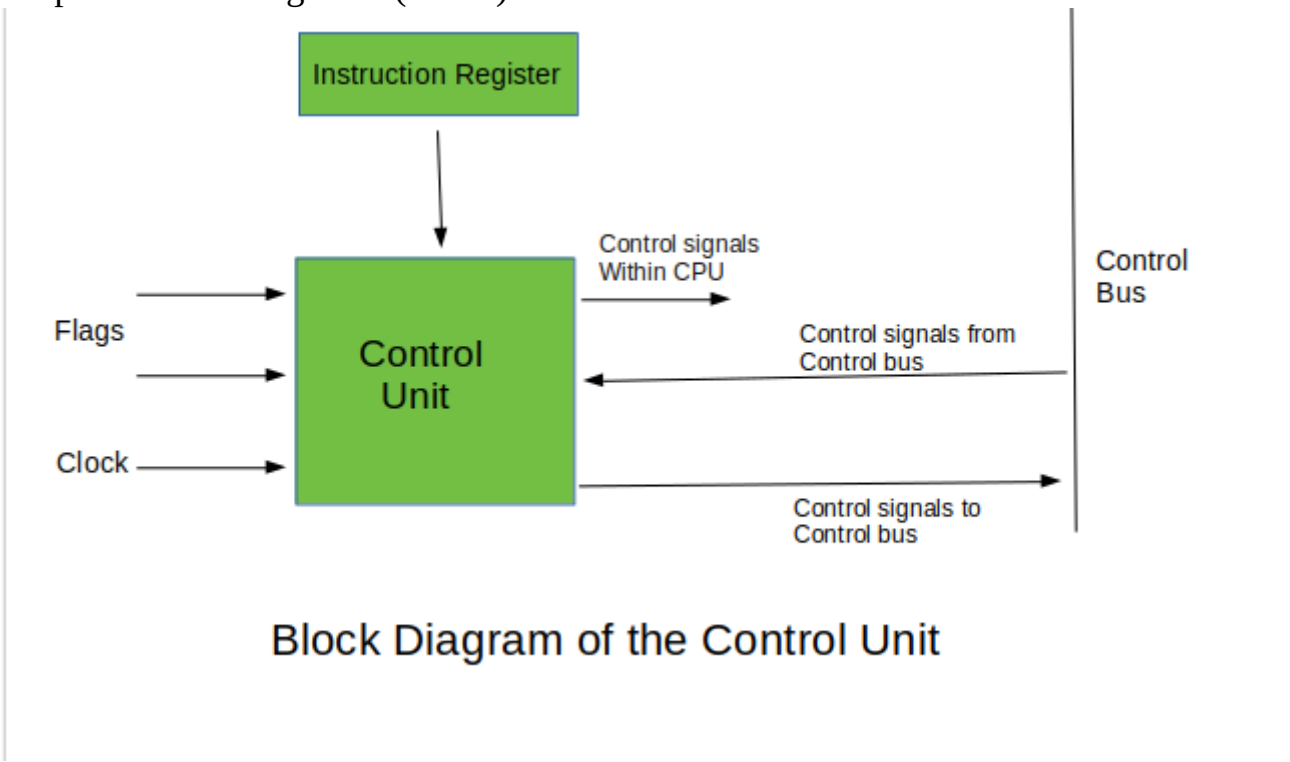
Introduction of Control Unit and its Design

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.
A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)



Block Diagram of the Control Unit

Instruction Register
Control signals
Within CPU
Control
Bus
Flags
Control
Unit
Control signals from
Control bus
Clock
Control signals to
Control bus
Block Diagram of the Control Unit

- To execute the instructions, the processor must have some means of generating control signals needed in the proper sequence.
- Computer designers use a wide variety of technology to solve this problem.
- This approach fall into two categories:
- i. Hardwired Control
- ii. Microprogrammed control

Hardwird Control

The control units use fixed logic circuits to interpret instructions and generate control signals from them.

The fixed logic circuit block includes combinational circuit that generates th required control outputs for decoding and encoding functions.

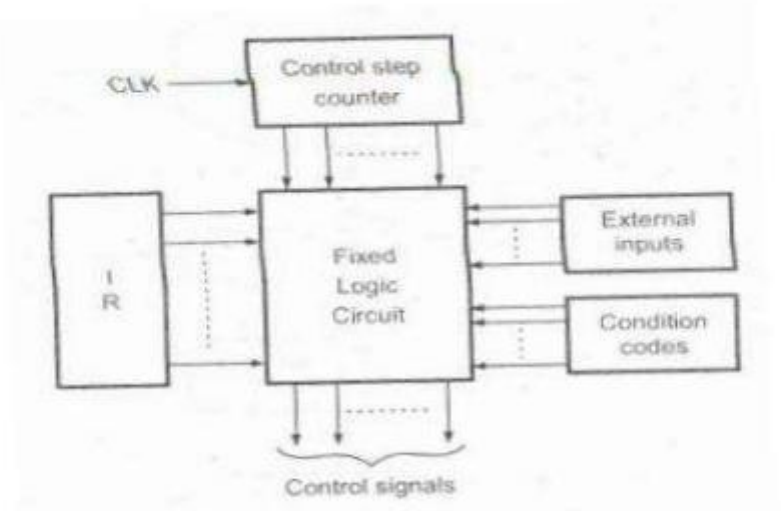


Fig. 3.10 Typical hardwired control unit

n decoder decodes the instruction loaded in the IF

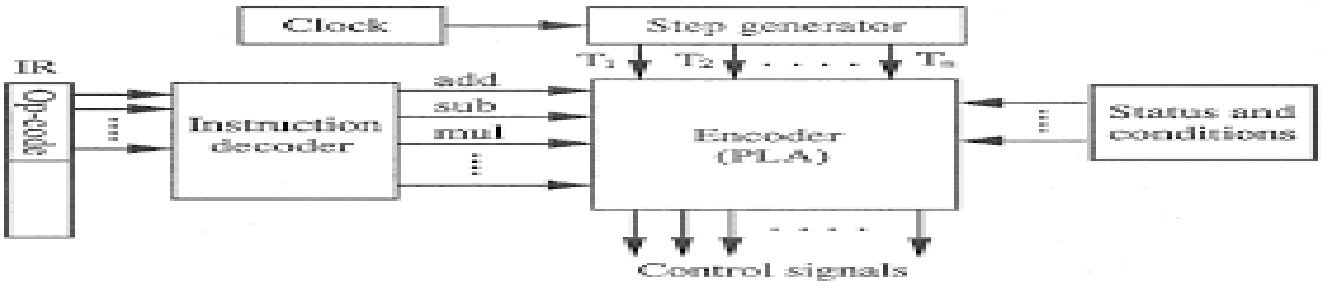
Hardwired control

- Each steps in this sequence is completed in one clock cycle.
- A counter may be used to keep the track of the control steps.
- In the hardwired control, the control unit use fixed logic circuits to interpret instructions and generate control signals from them.
- The required control signals are determined by the following information.

Contd.,

- 1) contents of the control step counter
- 2) contents of the instruction register
- 3) contents of the condition code flags
- 4) External input signals such as MFC and interrupt request.

Hardwired Control

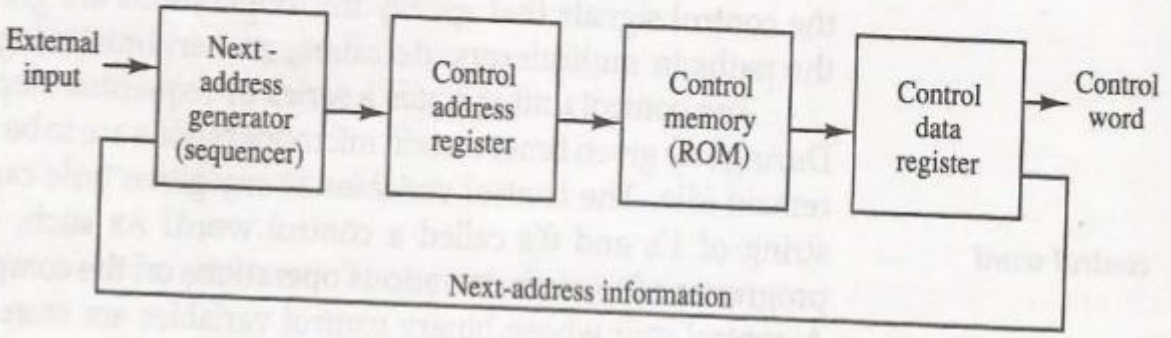


Block Diagram of Microprogarmmed Control Memory

- 4) Control Data Register (=Pipeline Register)
  - » Hold the microinstruction read from control memory
  - » Allows the execution of the microoperations specified by the control word *simultaneously* with the generation of the next microinstruction

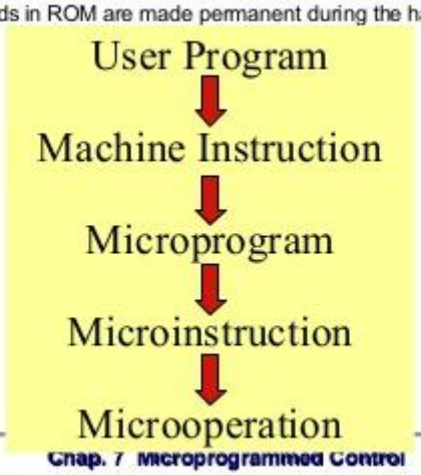
- Example(RISC Architecture Concept)
  - RISC(Reduced Instruction Set Computer) system use hardwired control rather than microprogrammed control

Figure 7-1 Microprogrammed control organization.



Microprogram control unit

- Microinstruction : (*Control Word in Control Memory*)
  - The instruction store in control memory is called microinstruction (specifies one or more microoperations)
- Microprogram
  - Microprogram is a sequence of microinstruction just like as program is a sequence of program. It is two type as follow:
    - » Dynamic microprogramming : (*Control Memory* = RAM)
      - RAM can be used for writing (*to change a writable control memory*)
      - Microprogram is loaded initially from an auxiliary memory such as a magnetic disk
    - » Static microprogramming : (*Control Memory* = ROM)
      - Control words in ROM are made permanent during the hardware production.



Micro programmed Control Organization

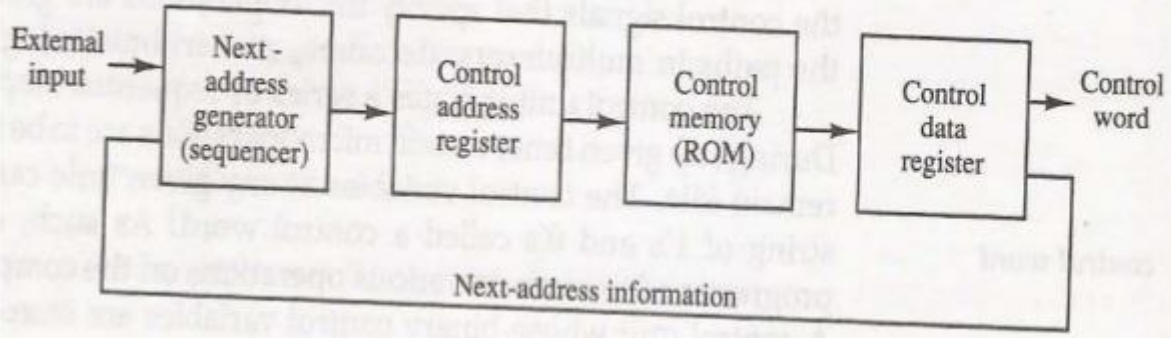
- Microprogrammed control Organization :(***Fig. 7-1)***
  - 1) Control Memory
    - » Computer Memory *employs a micro programmed control unit which have two separate memory*
      - Main Memory : for storing user program (*Machine instruction/data*)
      - Control Memory : for storing microprogram (*Microinstruction*)
  - 2) Control Address Register
    - » Specify the address of the microinstruction3) Sequencer (= *Next Address Generator*)
    - » Determine the address sequence that is read from control memory
    - » Next address of the next microinstruction can be specified several way depending on the sequencer input : *p. 217, [1, 2, 3, and 4]*

Block Diagram of Microprogarmmed Control Memory

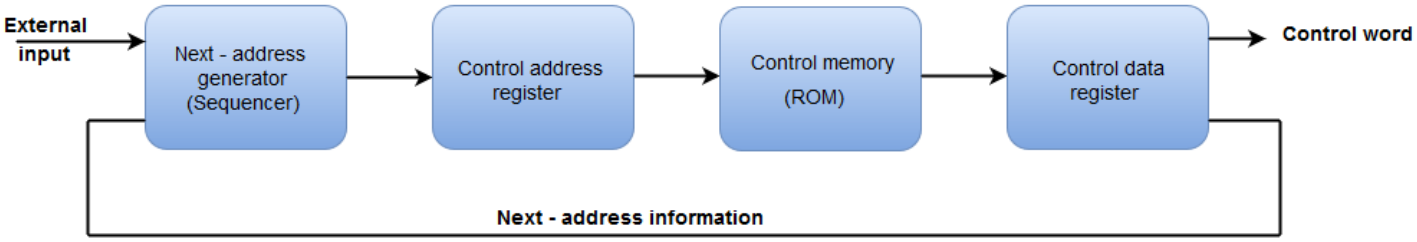
- 4) Control Data Register (=*Pipeline Register*)
  - » Hold the microinstruction read from control memory
  - » Allows the execution of the microoperations specified by the control word *simultaneously* with the generation of the next microinstruction

- Example(RISC Architecture Concept)
  - RISC(Reduced Instruction Set Computer) system use hardwired control rather than microprogrammed control

Figure 7-1 Microprogrammed control organization.



Microprogrammed Control Unit of a Basic Computer:



Microprogrammed Control vs Hardwired Control

Microprogrammed Control	Hardwired Control
It is a microprogram in control store that generates control signals.	It is the sequential circuit that generates control signals.
Speed of operation is low, because it involves memory access.	Speed of operation is high.
Changes in control behavior can be implemented easily by modifying the microinstruction in the control store.	Changes in control unit behavior can be implemented only by redesigning the entire unit.

Microprogrammed control	Hardwired control
It is the microprogram in control store that generates control signals.	It is the sequential circuit that generates control signals.
Speed of operation is low, because it involves memory access.	Speed of operation is high.
Changes in control behavior can be implemented easily by modifying the microinstruction in the control store.	Changes in control unit behavior can be implemented only by redesigning the entire unit.

Comparison between Hardwired and Microprogrammed Control Unit

Sr. No	Attribute	Hardwired	Microprogrammed
1	Speed	Fast	Slow
2	Cost of implementation	More	Cheaper
3	Implementation approach	Sequential circuit	Programming
4	Flexibility	Not flexible	Flexible
5	Ability to handle complex instruction	Difficult	Easier
6	Design process	Complicated	Systematic
7	Decoding and sequencing logic	Complex	Easy
8	Application	RISC µp	CISC µp
9	Control memory	Absent	Present
10	Chip area	Less	more

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

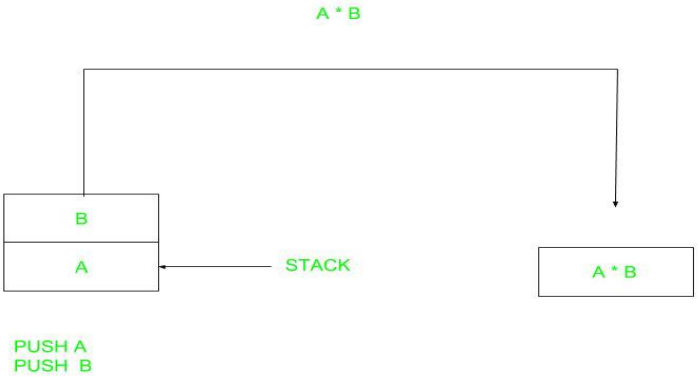
1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**



A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3. **Two Address Instructions –**
   This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

opcode	Destination address	Source address	mode

Here destination address can also contain operand.

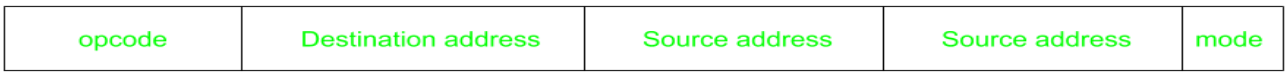
Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.



Expression: X = (A+B)*(C+D)

R1, R2 are registers

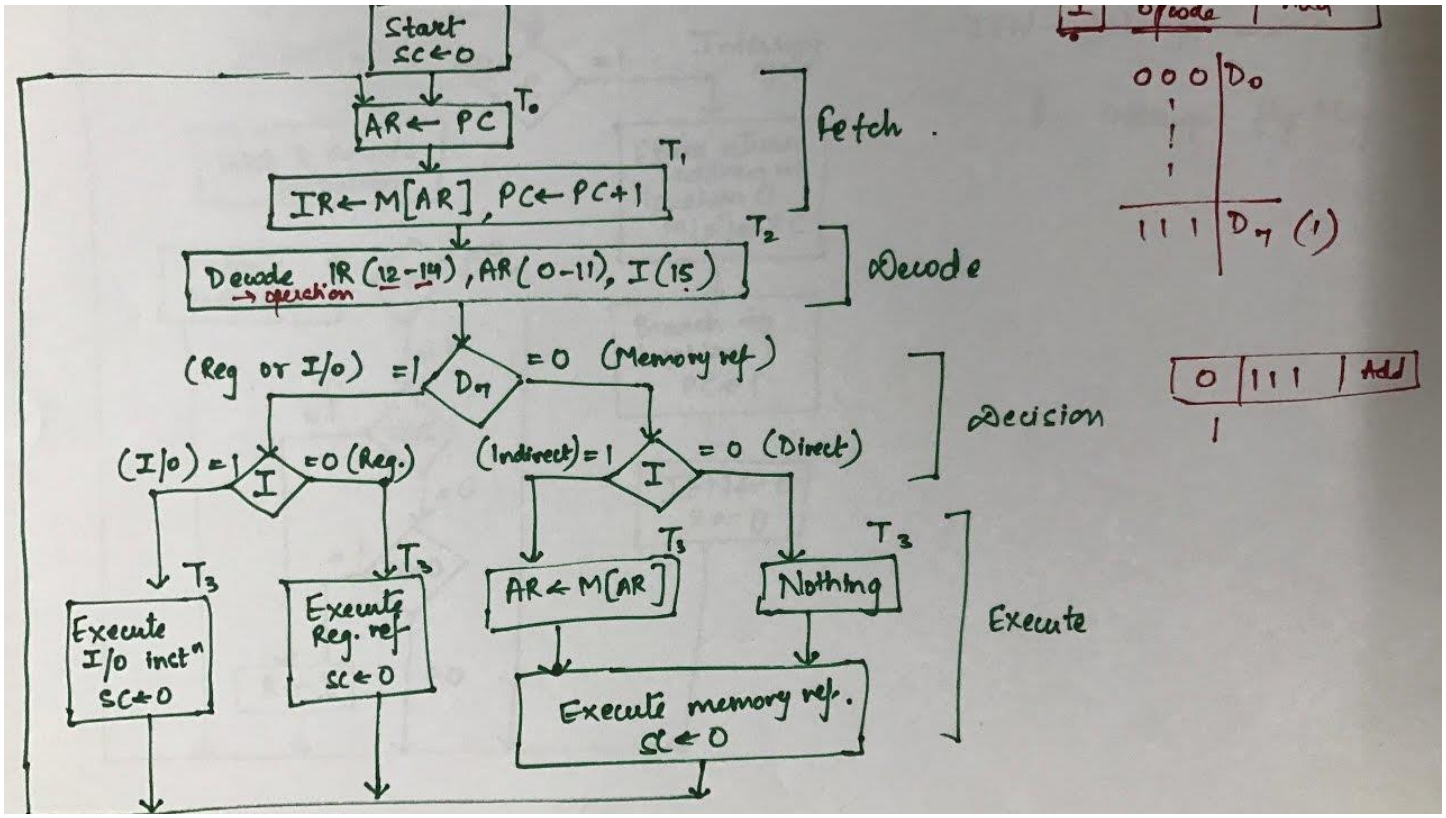
M[] is any memory location

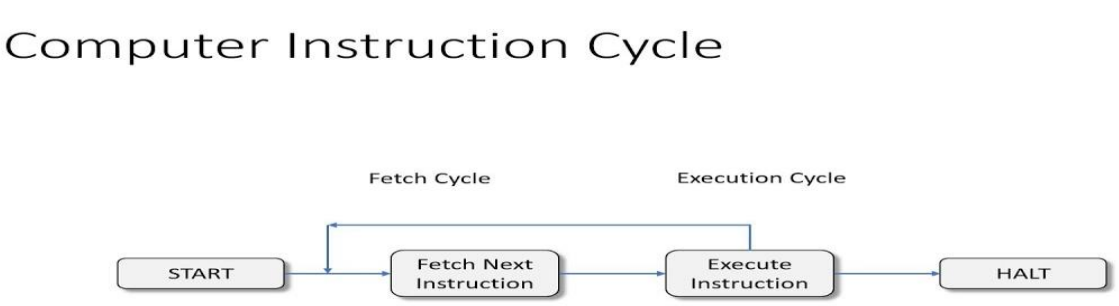
ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

Hardware vs. Micro-programmed Control

- Microinstructions are fetched, decoded, and executed in the same manner as regular instructions.
- This extra level of instruction interpretation is what makes microprogrammed control slower than hardwired control.
- The advantages of microprogrammed control are that it can support very complicated instructions and only the microprogram needs to be changed if the instruction set changes (or an error is found).

Instruction cycle



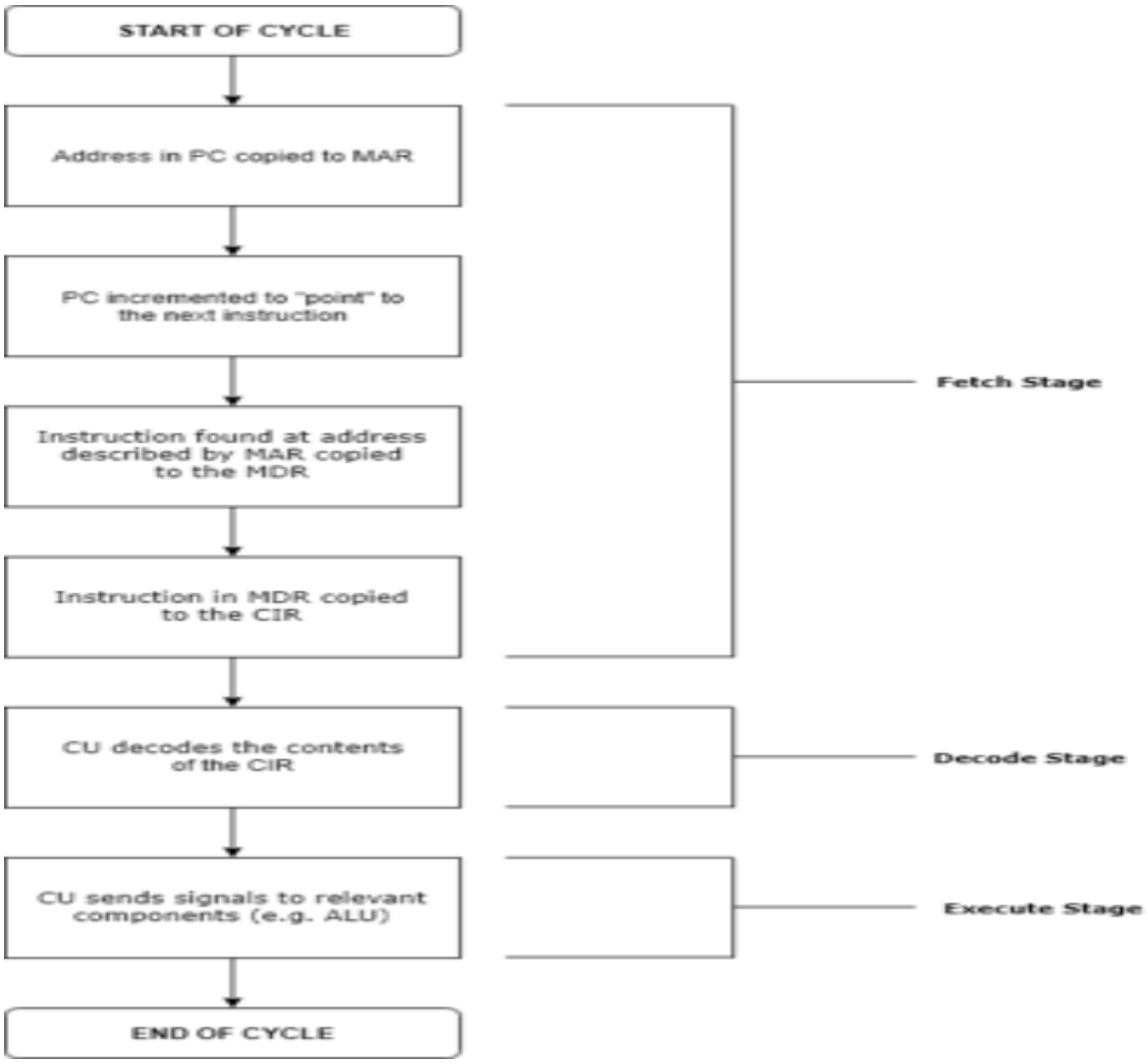


The instruction cycle (also known as the fetch–decode–execute cycle or simply the fetch-execute cycle) is the cycle which the central processing unit (CPU) follows from boot-up until the computer has shut down in order to process instructions. It is composed of three main stages: the fetch stage, the decode stage, and the execute stage.

This is a simple diagram illustrating the individual stages of the fetch-decode-execute cycle.

In simpler CPUs, the instruction cycle is executed sequentially, each instruction being processed before the next one is started. In most modern CPUs, the instruction cycles are instead executed concurrently, and often in parallel, through an instruction pipeline: the next instruction starts being processed before the previous instruction has finished, which is possible because the cycle is broken up into separate.

Execution of a complete instruction



Microinstruction Sequencing:

A micro-program control unit can be viewed as consisting of two parts:

1. The control memory that stores the microinstructions.
2. Sequencing circuit that controls the generation of the next address.

A micro-program sequencer attached to a control memory inputs certain bits of the microinstruction, from which it determines the next address for control memory. A typical sequencer provides the following address-sequencing capabilities:

1. Increment the present address for control memory.
2. Branches to an address as specified by the address field of the micro instruction.
3. Branches to a given address if a specified status bit is equal to 1.

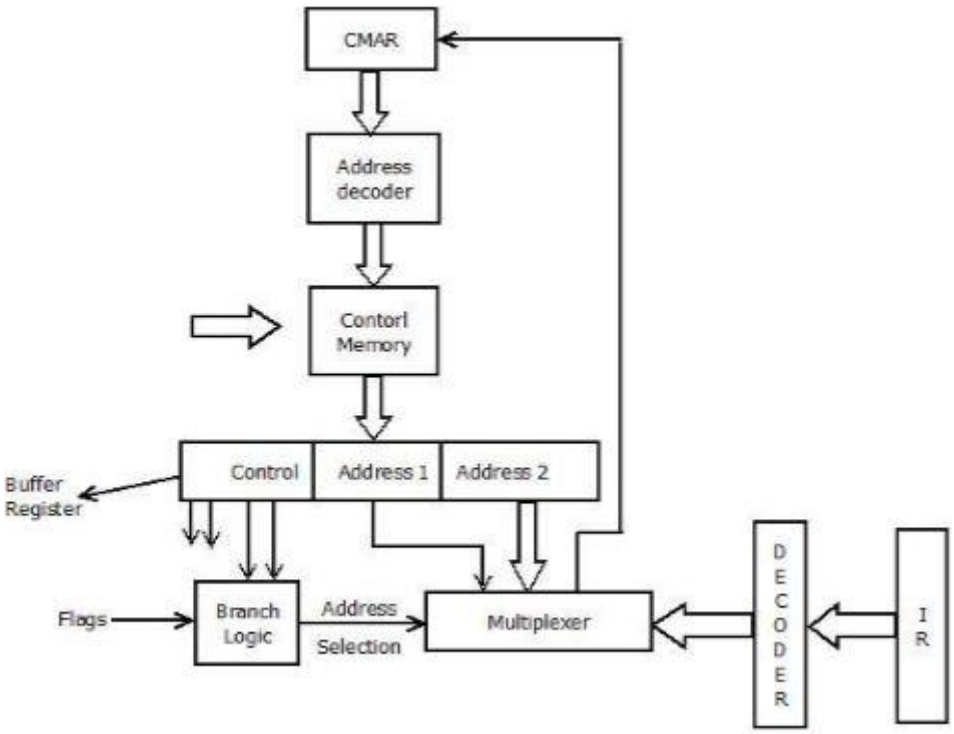
4. Transfer control to a new address as specified by an external source (Instruction Register).
5. Has a facility for subroutine calls and returns.

Depending on the current microinstruction condition flags, and the contents of the instruction register, a control memory address must be generated for the next micro instruction.

There are three general techniques based on the format of the address information in the microinstruction:

1. Two Address Field.
2. Single Address Field.
3. Variable Format

Two Address Field:



The simplest approach is to provide two address field in each microinstruction and multiplexer is provided to select:

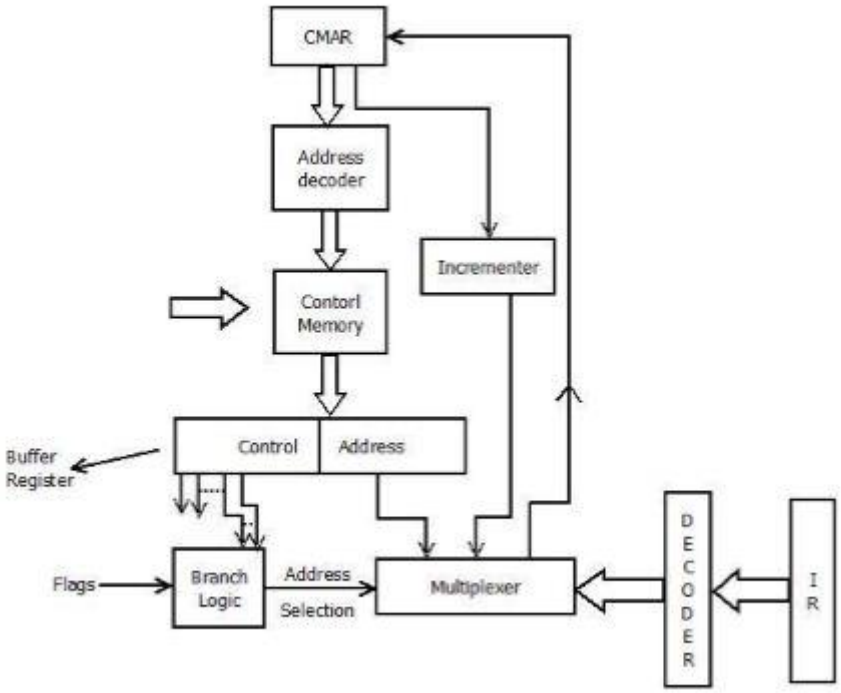
- Address from the second address field.
- Starting address based on the OPcode field in the current instruction.

The address selection signals are provided by a branch logic module whose input consists of control unit flags plus bits from the control partition of the micro instruction.

Single Address Field:

Two-address approach is simple but it requires more bits in the microinstruction. With a simpler approach, we can have a single address field in the micro instruction with the following options for the next address.

- Address Field.
- Based on OPcode in instruction register.
- Next Sequential Address.



The address selection signals determine which option is selected. This approach reduces the number of address field to one. In most cases (in case of sequential execution) the address field will not be used. Thus the microinstruction encoding does not efficiently utilize the entire microinstruction.

Variable Format:

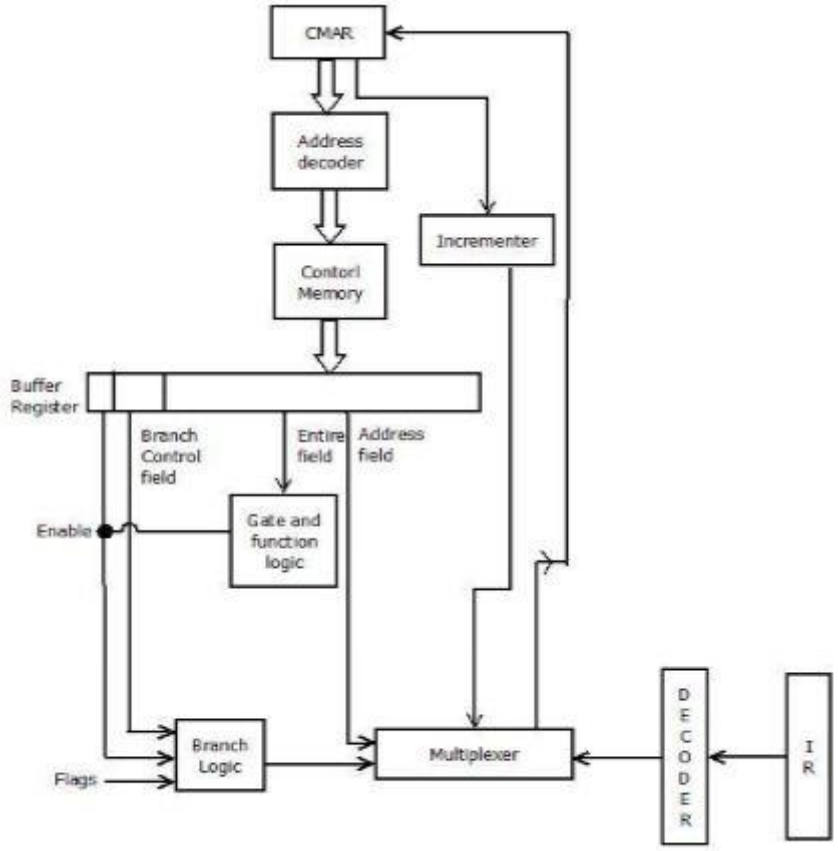


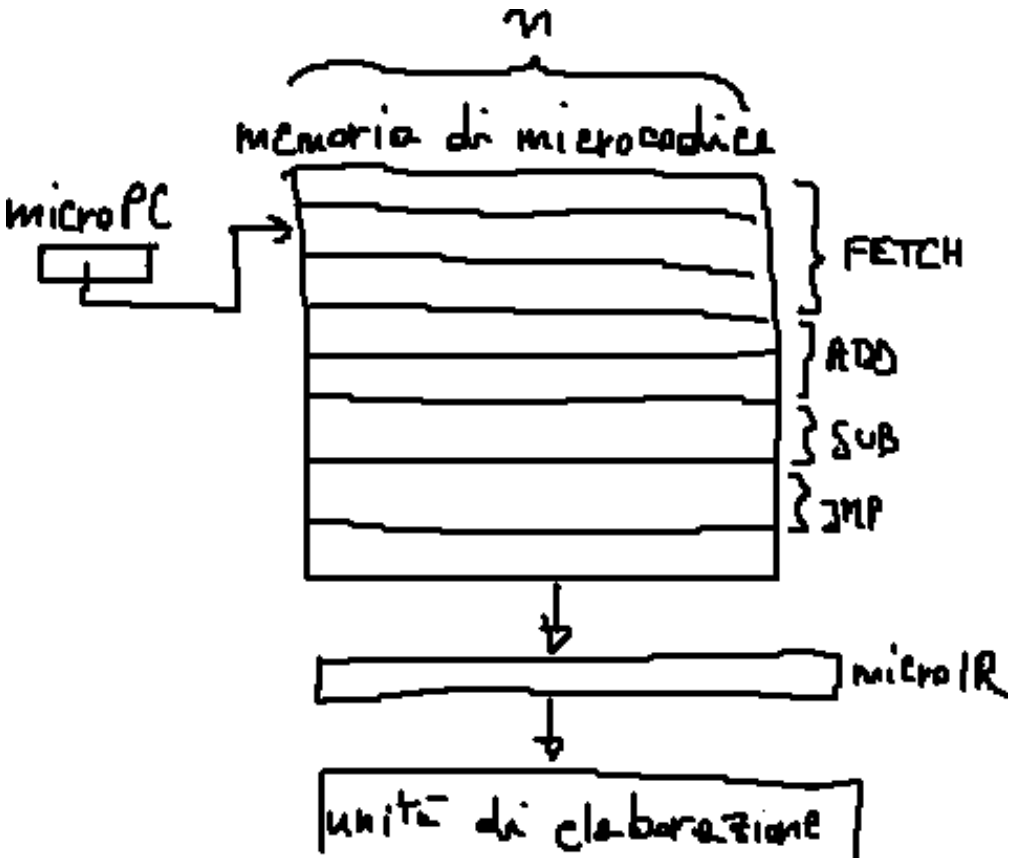
Fig. 3.41: Branch control logic, variable format

In this approach, there are two entirely different microinstruction formats. One bit designates which format is being used. In this first format, the remaining bits are used to activate control signals. In the second format, some bits drive the branch logic module, and the remaining bits provide the address. With the first format, the next address is either the next sequential address

or an address derived from the instruction register. With the second format, either a conditional or unconditional branch is specified.

Horizontal Microprogramming

In horizontal microprogramming, each bit is identified specifically with a single control point, which indicates that the corresponding micro-operation is to be executed. Since each microinstruction is capable enough to control several resources simultaneously, it has the potential advantage of more efficient hardware utilization and in addition, it requires smaller number of microinstructions per microprogram. It allows higher degree of parallelism with a minimum amount of encoding and separate control fields. However, developing microprograms that use resources optimally or efficiently is a complex task. Horizontal microprogramming offers great flexibility because each control bit is independent of each other. It has greater length so it typically contains more information than vertical microinstructions.



Vertical Microprogramming

Vertical microprogramming employs a variable format and higher degree of encoding, as opposed to horizontal microprogramming. It not only shortens the length of the microinstruction, but also prevents the increasing memory capacity from directly affecting the microinstruction length. Each vertical microinstruction generally represents a single micro-

operation. A code is used for each micro-operation to be performed and the decoder translates the code into individual control signals. Because only the micro-operation to be performed is specified, the microinstruction fields are fully utilized. Plus vertical microprograms are easier to write than their horizontal counterparts. Vertical microinstruction resembles the conventional machine language format comprising one operation and a few operands. It is consequently easy to use for microprogramming. It generally consists of four to six fields that require approximately 16 to 32 bits per instruction.

Difference between Horizontal and Vertical Microprogramming

Encoding

Vertical microprogramming employs a variable format and a higher degree of encoding, as opposed to horizontal microprogramming. In vertical microprogramming, the control bits are encoded with each code being used for each action to be performed and an instruction decoder decodes the code into multiple control signals. On the contrary, horizontal microprogramming involves horizontal microinstructions that use no encoding at all. They represent each control bit in the datapath assigned with a separate bit in the microinstruction format. Every bit in the control field is attached to a control line.

Sequence

Horizontal microprogramming generally follows a sequential approach to specify the next microinstruction in a microprogram, similar to conventional machine language format. Each bit is identified specifically with a single control point, which indicates that the corresponding micro-operation is to be executed. Special conditional and unconditional branch microinstructions are then required to break the sequence. Vertical microprogramming may use a relatively addressing scheme in which a few bits are required to specify a relative forward or a backward jump. This requires address computation at every step.

Design

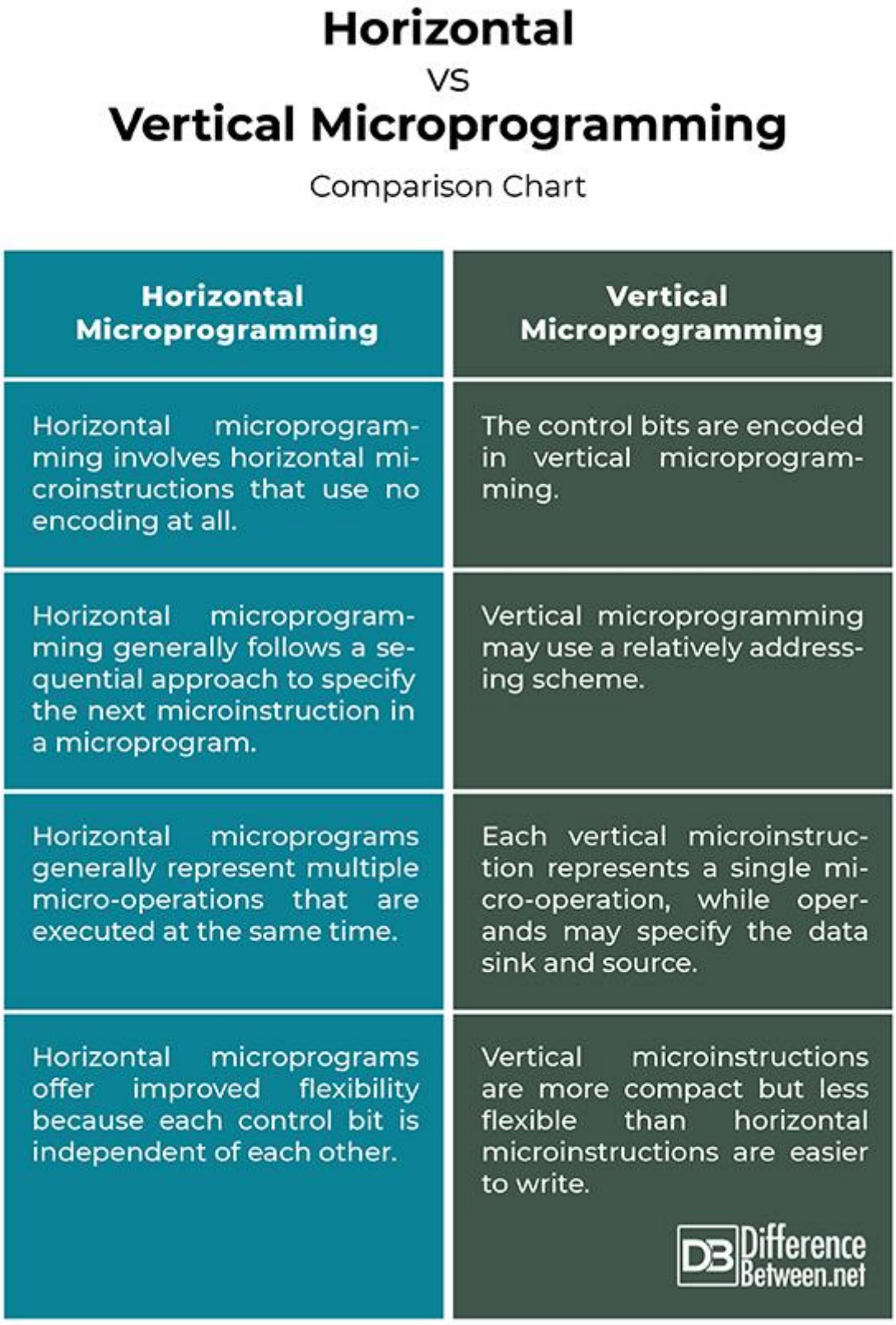
– Vertical microprograms have a better code density which is beneficial for the size of the control store. Vertical microinstruction resembles the conventional machine language format comprising one operation and a few operands. Each vertical microinstruction represents a single micro-operation, while operands may specify the data sink and source. Horizontal

microprograms, on the other hand, generally represent multiple micro-operations that are executed at the same time. In extreme cases, each horizontal microinstruction controls several hardware resources simultaneously.

Flexibility

– Horizontal microprograms offer improved flexibility because each control bit is independent of each other. It has greater length so it typically contains more information than vertical microinstructions. Horizontal microinstructions with 48 or more bits are quite common. Horizontal microprograms have the potential advantage of utilizing hardware more efficiently and on top of it, it requires smaller numbers of microinstructions per microprogram. Vertical microinstructions, on the other hand, are more compact but less flexible than horizontal microinstructions. The vertical approach is consequently easy to use for microprogramming.

Horizontal vs. Vertical Microprogramming: Comparison Chart



Summary of Horizontal and Vertical Microprogramming

As opposed to horizontal microinstructions, the vertical microinstruction represents single micro-operations. Horizontal microprograms allow higher degree of parallelism with a minimum amount of encoding and separate control fields whereas the control bits are encoded in vertical microprograms. The choice between the two approaches needs to be made carefully. However, in practical, designers use a combination of horizontal and vertical microinstruction formats so that the resulting structure is compact yet efficient.

Adressing Modes and Instruction Cycle

The operation field of an instruction specifies the operation to be performed. This operation will be executed on some data which is stored in computer registers or the main memory. The way any operand is selected during the program execution is dependent on the addressing mode of the instruction. The purpose of using addressing modes is as follows:

1. To give the programming versatility to the user.
2. To reduce the number of bits in addressing field of instruction.

Types of Addressing Modes

Below we have discussed different types of addressing modes one by one:

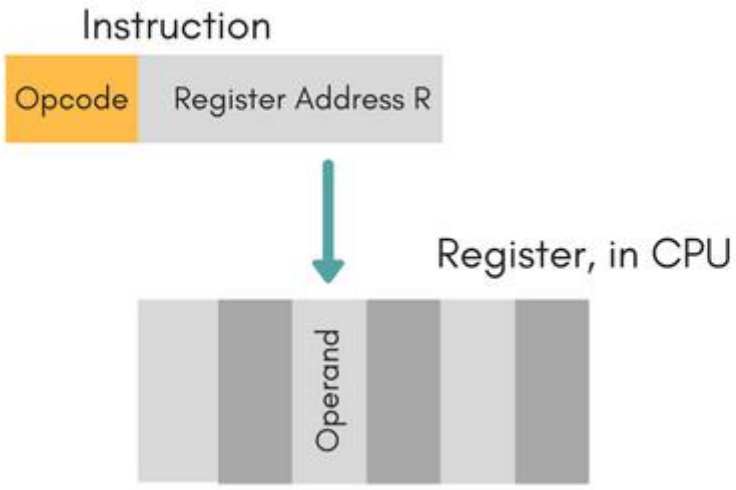
Immediate Mode

In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

For example: ADD 7, which says Add 7 to contents of accumulator. 7 is the operand here.

Register Mode

In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.



Advantages

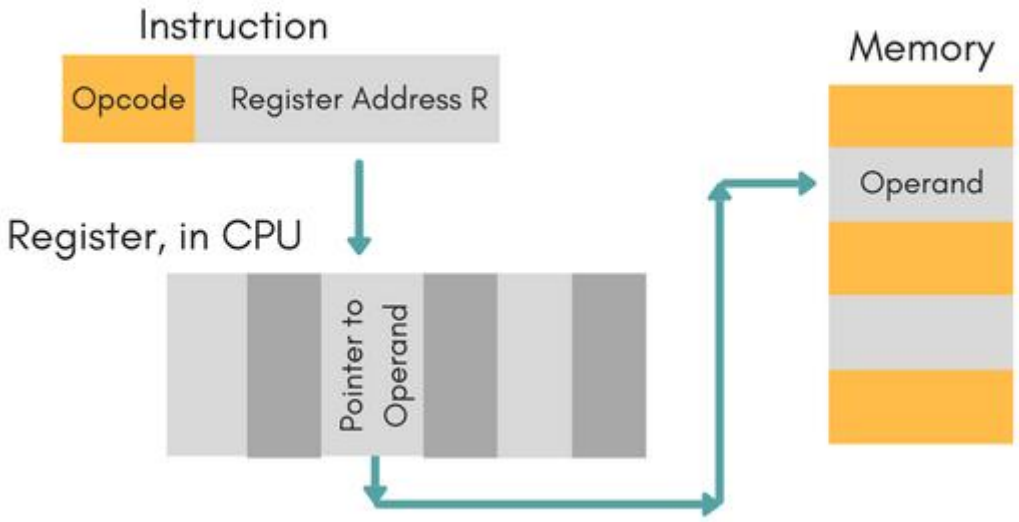
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

Register Indirect Mode

In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



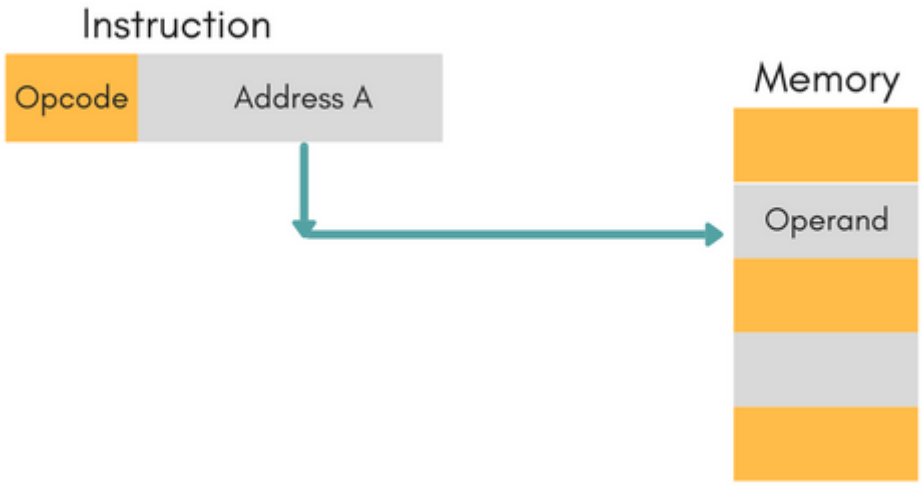
Auto Increment/Decrement Mode

In this the register is incremented or decremented after or before its value is used.

Direct Addressing Mode

In this mode, effective address of operand is present in instruction itself.

- Single memory reference to access data.
- No additional calculations to find the effective address of the operand.

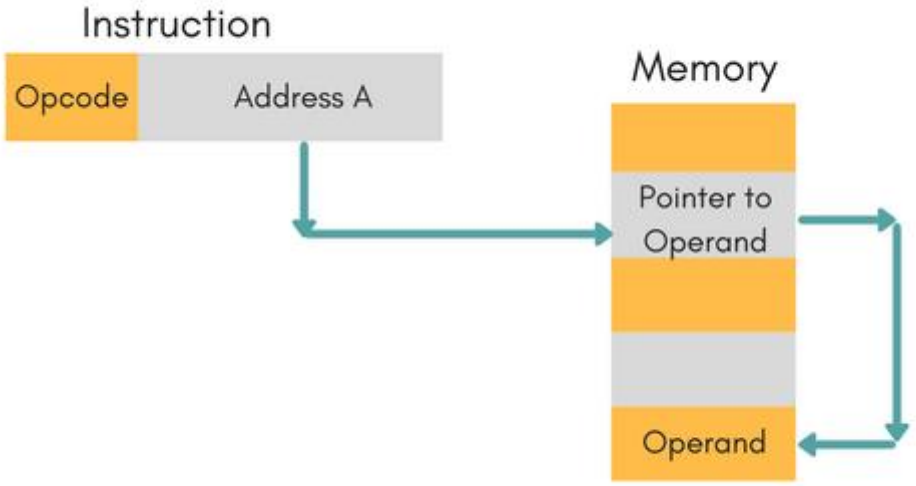


For Example: ADD R1, 4000 - In this the 4000 is effective address of operand.

NOTE: Effective Address is the location where operand is present.

Indirect Addressing Mode

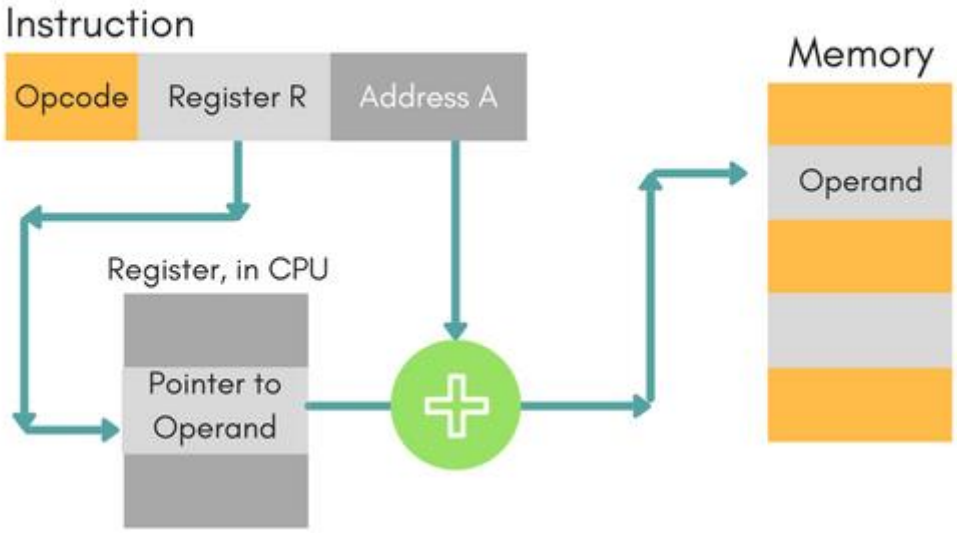
In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.



Displacement Addressing Mode

In this the contents of the indexed register is added to the Address part of the instruction, to obtain the effective address of operand.

EA = A + (R), In this the address field holds two values, A(which is the base value) and R(that holds the displacement), or vice versa.



Relative Addressing Mode

It is a version of Displacement addressing mode.

In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.

EA = A + (PC), where EA is effective address and PC is program counter.

The operand is A cells away from the current cell(the one pointed to by PC)

Base Register Addressing Mode

It is again a version of Displacement addressing mode. This can be defined as EA = A + (R), where A is displacement and R holds pointer to base address.

Stack Addressing Mode

In this mode, operand is at the top of the stack. For example: ADD, this instruction will *POP* top two items from the stack, add them, and will then *PUSH* the result to the top of the stack.

Instruction Cycle

An instruction cycle, also known as **fetch-decode-execute cycle** is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer.

Following are the steps that occur during an instruction cycle:

1. Fetch the Instruction

The instruction is fetched from memory address that is stored in PC(Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. Decode the Instruction

The instruction in the IR is executed by the decoder.

3. Read the Effective Address

If the instruction has an indirect address, the effective address is read from the memory. Otherwise operands are directly read in case of immediate operand instruction.

4. Execute the Instruction

The Control Unit passes the information in the form of control signals to the functional unit of CPU. The result generated is stored in main memory or sent to an output device.

The cycle is then repeated by fetching the next instruction. Thus in this way the instruction cycle is repeated continuously.