UNIT 6
Basic Processing Unit

Objectives

- How a processor execute an instruction?
- The internal functional units of processor and how they are interconnected?
- H/w for generating internal control signal.
- The micro programming approach.
- Micro program organization.

Fundamental Concepts

- The processor fetches one instruction at a time and performs the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- The processor uses the **Program Counter (PC),** to keep track of the address of the next instruction to be fetched and executed.

- After fetching an instruction, the contents of the PC are updated to point to the next instruction in sequence.

- A branch instruction may cause a different value to be loaded into the PC.

- When an instruction is fetched, it is placed in the **Instruction Register (IR)**, from where it is interpreted, or decoded, by the processor's control circuitry.

- The IR holds the instruction until its execution is completed.

1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are interpreted as an instruction to be executed. Hence, they are loaded into the IR. Symbolically, this can be written as

$$IR \leftarrow [[PC]]$$

2. Assuming that the memory is byte addressable, increment the contents of the PC by 4, that is,

$$PC \leftarrow [PC] + 4$$

3. Carry out the actions specified by the instruction in the IR.

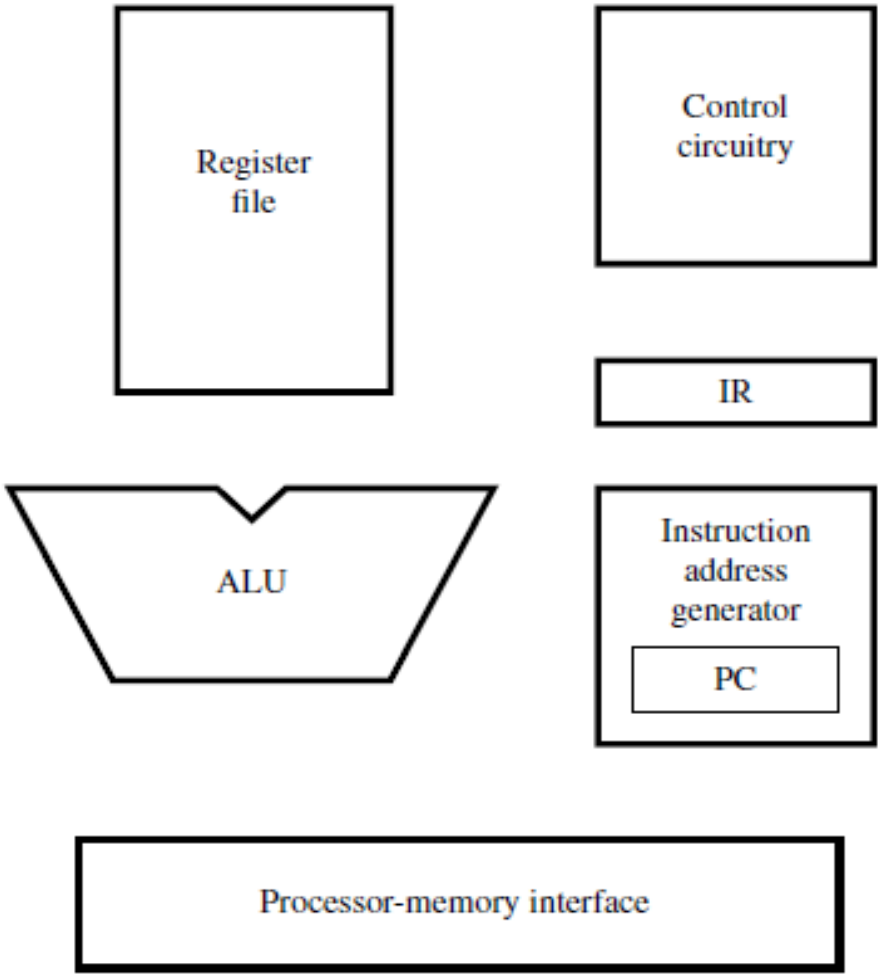
- Steps 1 and 2 are part of Fetch phase
- Step 3 continues the execution

- Fetching an instruction and loading it into the IR is usually referred to as the ***Instruction Fetch Phase***.

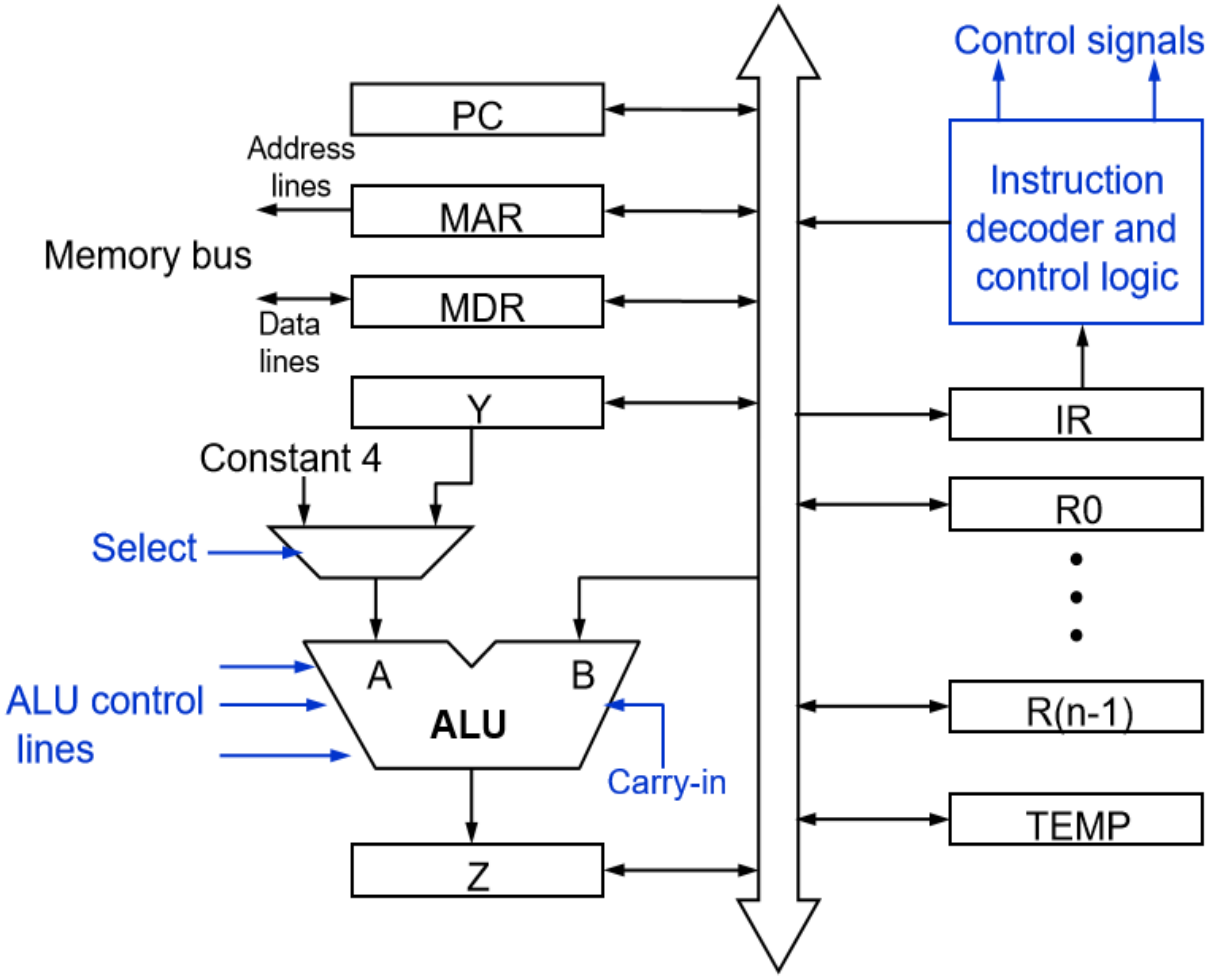
- Performing the operation specified in the instruction constitutes the ***Instruction Execution Phase***.

- With few exceptions, the operation specified by an instruction can be carried out by performing one or more of the following actions:
  - Read the contents of a given memory location and load them into a processor register.
  - Read data from one or more processor registers.
  - Perform an arithmetic or logic operation and place the result into a processor register.
  - Store data from a processor register into a given memory location.

Main Hardware Components of a Processor

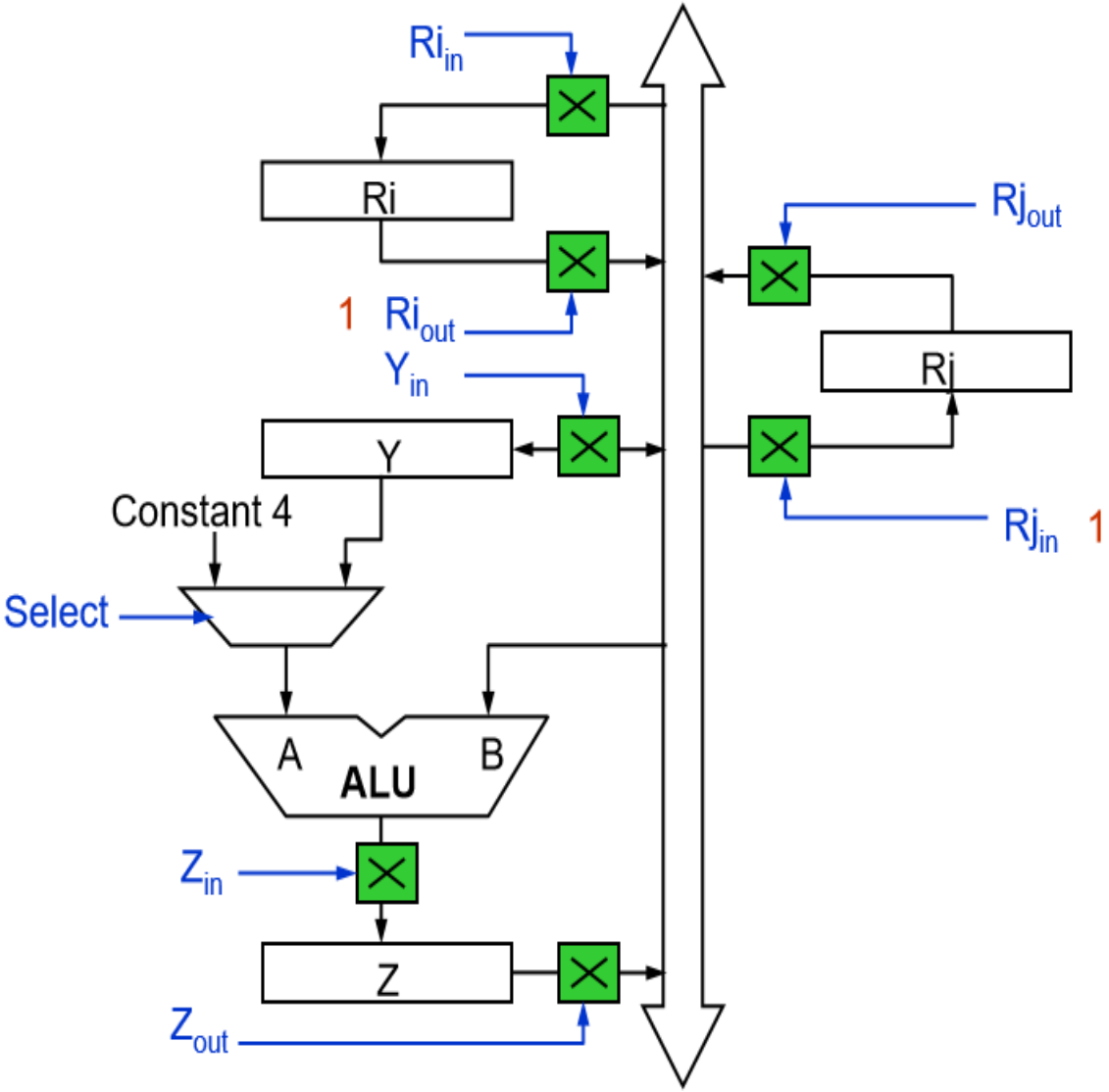


Single-Bus Organization of the Datapath



- **MAR:** Memory Address Register
- **MDR:** Memory Data Register
- **Y, Z, TEMP:** Temporary Registers used by processor.
- **R0 To R(n-1):** General Purpose Registers
- **PC:** Program Counter
- **IR:** Instruction Register

Register Transfers



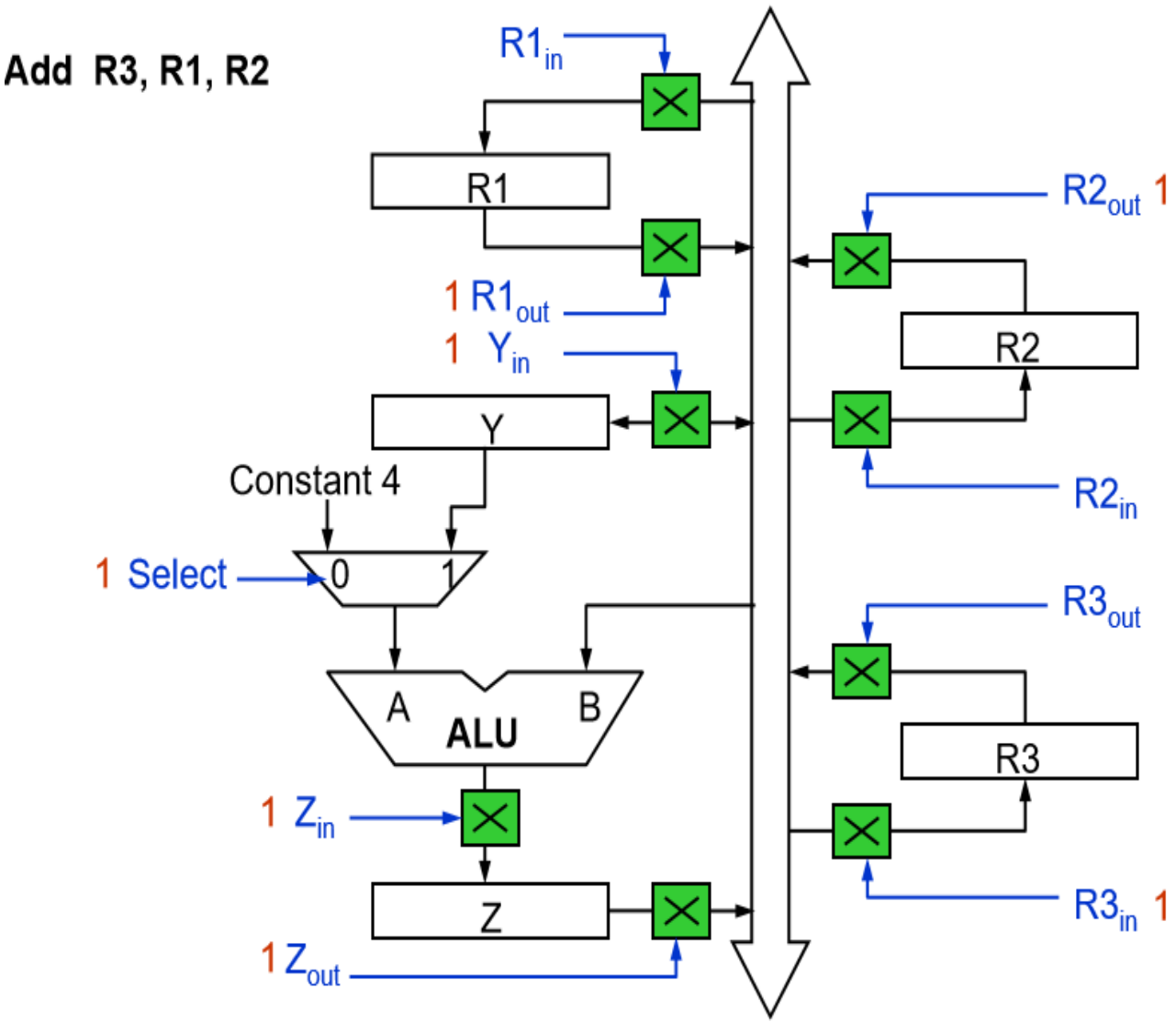
Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register. This is represented symbolically in Figure 7.2. The input and output of register R*i* are connected to the bus via switches controlled by the signals Ri_{in} and Ri_{out}, respectively. When Ri_{in} is set to 1, the data on the bus are loaded into R*i*. Similarly, when Ri_{out} is set to 1, the contents of register R*i* are placed on the bus. While Ri_{out} is equal to 0, the bus can be used for transferring data from other registers.

Suppose that we wish to transfer the contents of register R1 to register R4. This can be accomplished as follows:

- Enable the output of register R1 by setting $R1_{out}$ to 1. This places the contents of R1 on the processor bus.
- Enable the input of register R4 by setting $R4_{in}$ to 1. This loads data from the processor bus into register R4.

All operations and data transfers within the processor take place within time periods defined by the *processor clock*. The control signals that govern a particular transfer are asserted at the start of the clock cycle. In our example, $R1_{out}$ and $R4_{in}$ are set to 1. The registers consist of edge-triggered flip-flops. Hence, at the next active edge of the clock, the flip-flops that constitute R4 will load the data present at their inputs. At the same time, the control signals $R1_{out}$ and $R4_{in}$ will return to 0. We will use this simple model of the timing of data transfers for the rest of this chapter. However, we should point out that other schemes are possible. For example, data transfers may use both the rising and falling edges of the clock. Also, when edge-triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper transfer of data. This is known as *multiphase clocking*.

Arithmetic Operation



R3=R1+R2

Set of control signals or microinstructions generated are

1. **$R1_{out}$, Y_{in}**
2. **$R2_{out}$, SelectY, Add, Z_{in}**
3. **Z_{out}, $R3_{in}$**

Examples

1. **Add R4,R2,R5**

2. **SUB R3,R2,R1**

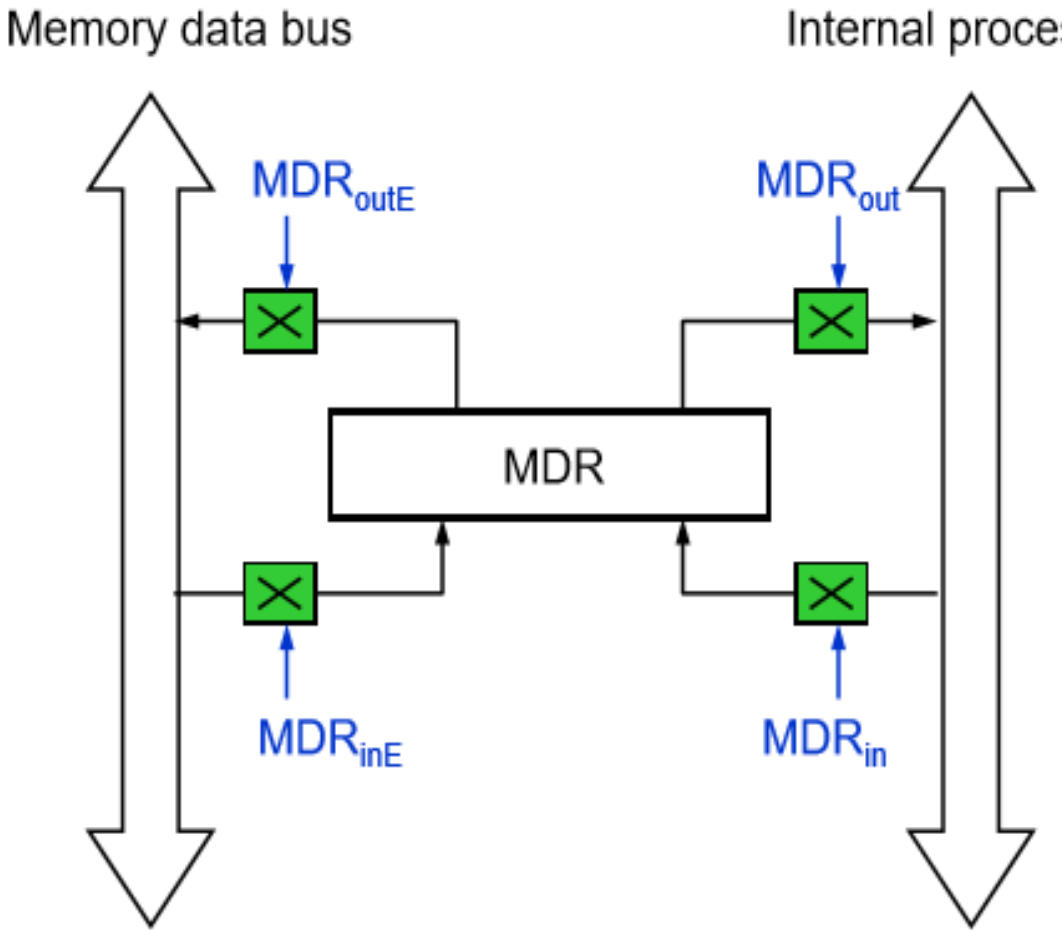
Read operation from Memory

- **MOVE [R1],R2**

1. **MAR<- [R1]**
2. **Start a Read operation on the memory bus**
3. **Wait for the MFC response from the memory**
4. **Load MDR from the memory bus**
5. **R2<- [MDR]**

Memory Function Completed (MFC) Signal: Processor waits until it receives an indication that requested read operation has been completed. This signal will be 1, required data is read and available on data line to access.

Fetching a Word from Memory



Move (R1), R2

1. R1$_{out}$, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

WMFC: is the control signal that causes the processor's control circuitry to wait for the arrival of the MFC signal.

Write Operation to Memory

- **MOVE R2,(R1)**

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{inE}, Write
3. MDR_{outE} , WMFC

Execution of Complete Instruction

Consider instruction:

Add (R3) ,R1

(Which adds the contents of a memory location pointed to by R3 to register R1)

Executing this instruction require following steps:

1. Fetch the instruction
2. Fetch the first operand (R3)
3. Perform the addition
4. Load the result into R1

Execution of Complete Instruction

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Example

1. **SUB (R2) ,R4**

BRANCH Instructions (Unconditional)

Steps	Action
1	PC_{out} , MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

The offset X used in the branch instruction is usually difference between the branch target address and the address immediately following the branch instruction

Generating Control Signals

- To execute instructions, the **processor requires some unit to generate control signals in proper sequence.**

- Proposed approach system is of two kinds:

  - **Hardwired Control**
  - **Micro-programmed Control**

1. **Hardwired Control:** In hardwired control, control signals required inside processor can be generated using a dedicated hardware device.

2. **Micro-programmed Control:** In micro-programmed control, control signals are generated by a special program similar to machine language programs.

1. Hardwired Control

- In hardwired control, control signals required inside processor can be generated using generated using a **dedicated hardware device**.

- Hardware device **Control Step Counter** and **Decoder/Encoder Circuit**.

Example

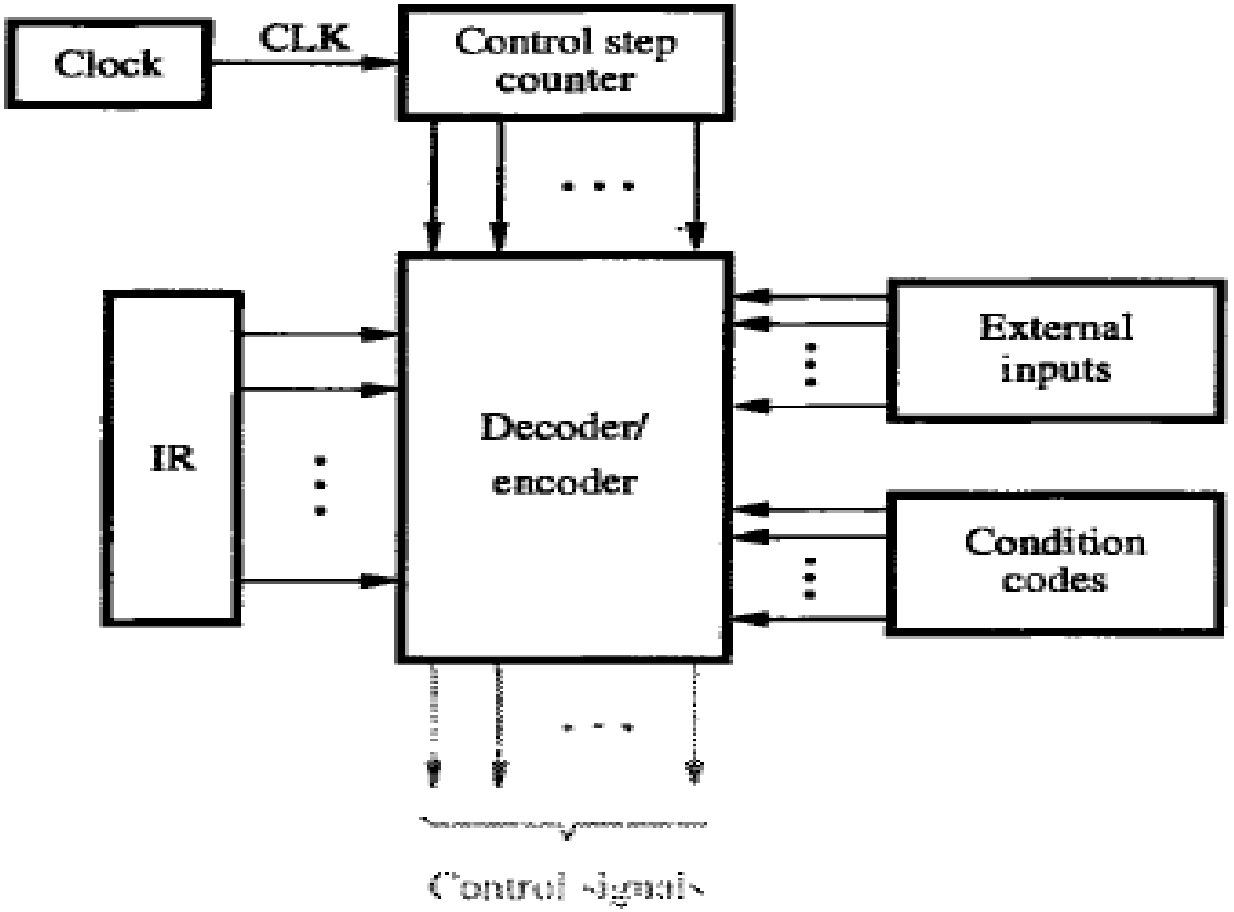
Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

- Eg. Consider set of microinstructions on previous slide. **Consider each step in a sequence is completed in one clock period.** Processor uses counter to keep track of each clock period.

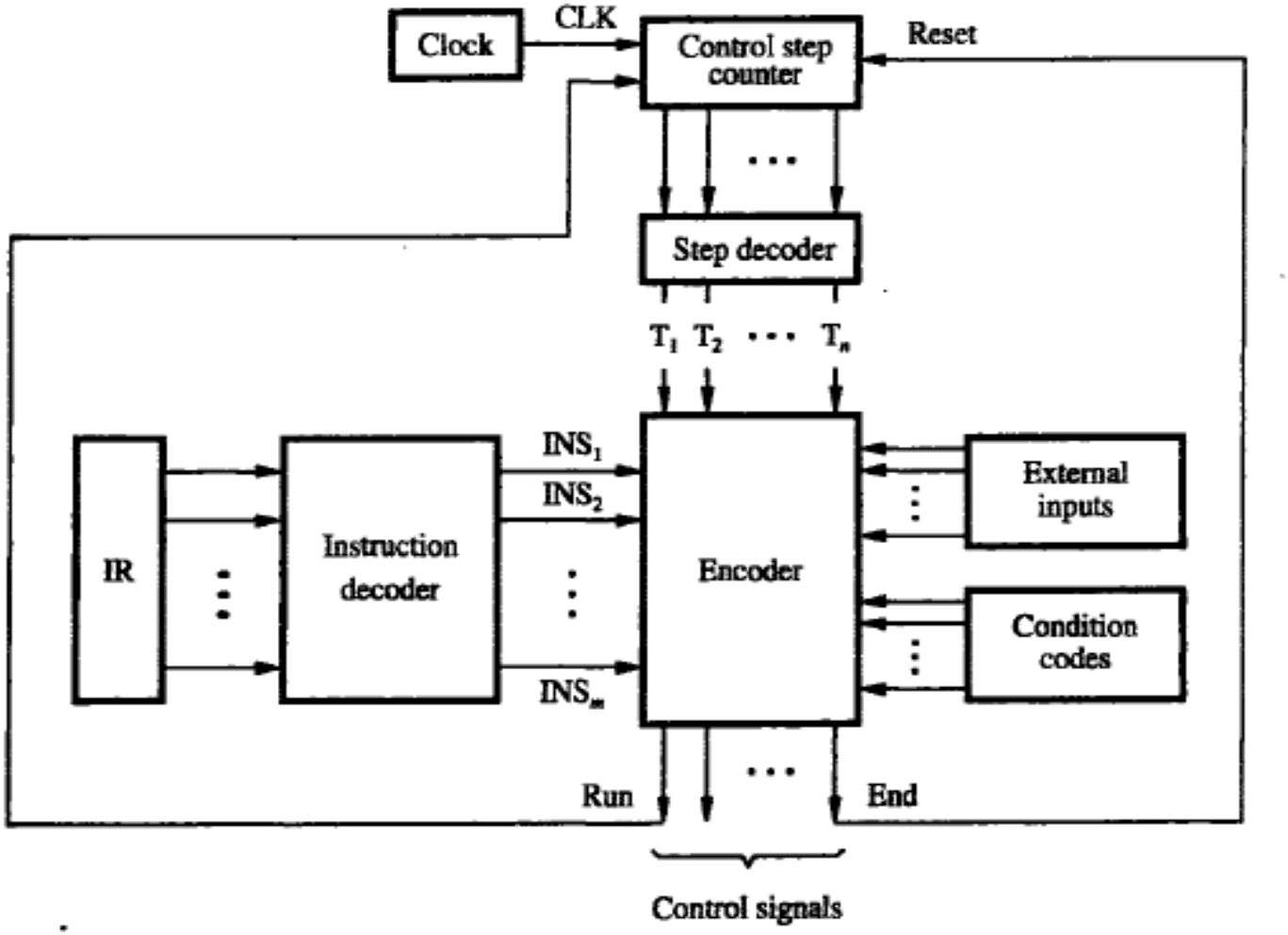
- Each step or count corresponds to one control step. The required control signals are determined by following information:

  - Content of **Control Step Counter**
  - Contents of **Instruction Register**
  - Contents of **Conditional Flags**
  - **External Control Signals (eg. MFC, Interrupt requests)**

Control Unit Organization



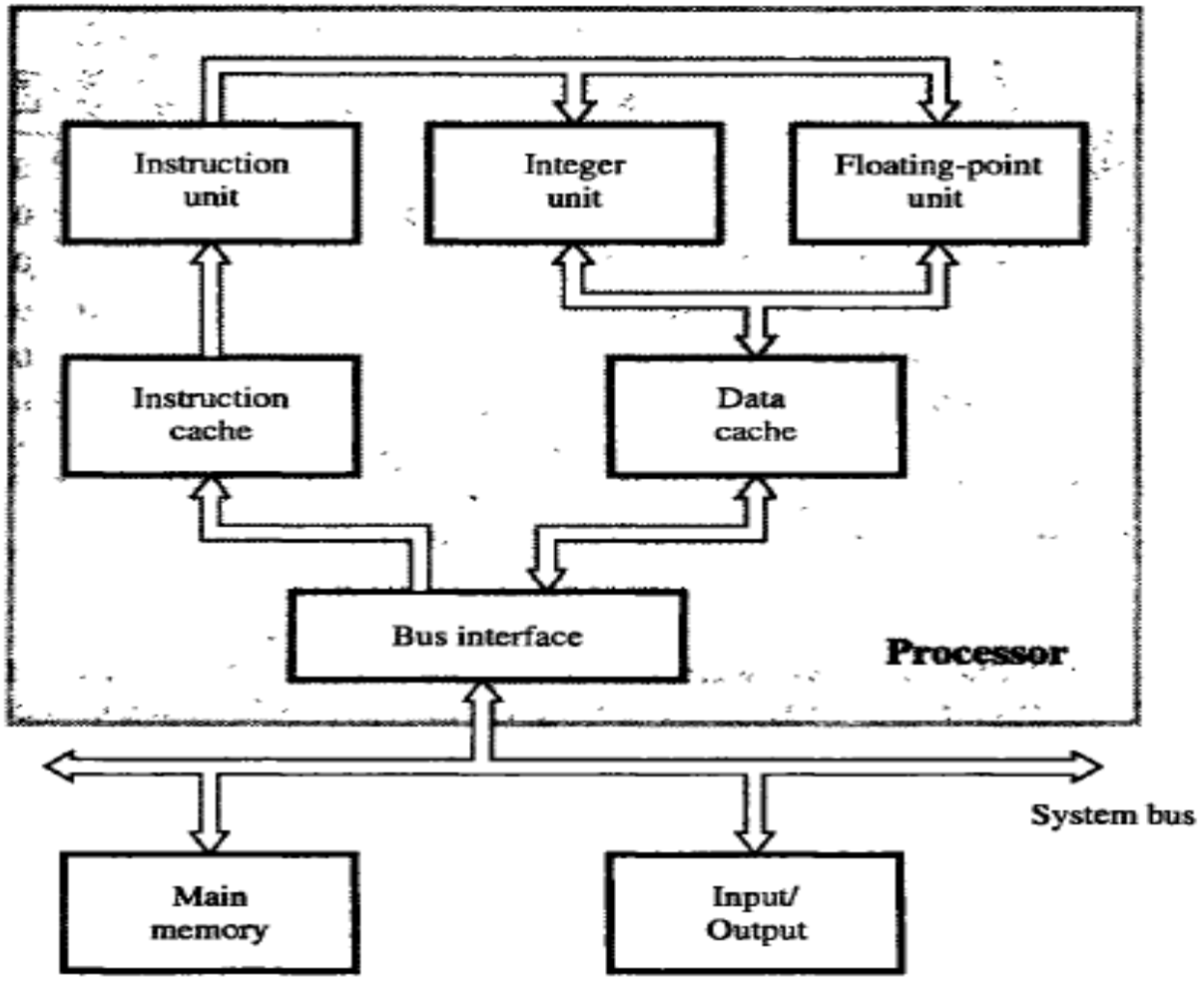
Detailed Control Design



A Complete Processor

- It consists of
  - Instruction Unit
  - Integer Unit
  - Floating-point Unit
  - Instruction Cache
  - Data Cache
  - Bus Interface Unit
  - Main Memory Module
  - Input/Output Module.

Block Diagram of a Complete Processor



- **Instruction Unit-** It fetches instructions from an instruction cache or from the main memory when the desired instructions are not available in the cache.
- **Integer Unit**– To process integer data.
- **Floating Unit**– To process floating –point data.
- **Data Cache**– The integer and floating unit gets data from data cache.
- **Instruction Cache-** The integer and floating unit gets next instruction from instruction cache.

- Every instruction in a processor is implemented by a sequence of one or more **sets of concurrent micro operations.**

- **Each micro operation is associated with a specific set of control lines** which, when activated, causes that micro operation to take place.

- Since the number of instructions and control lines is often in the hundreds, the **complexity of hardwired control unit is very high.**

- Thus, **it is costly and difficult to design.**

- **The hardwired control unit is relatively inflexible** because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

2. Micro-programmed Control

- In micro-programmed control, control signals are generated by a **Special Program** similar to machine language programs.

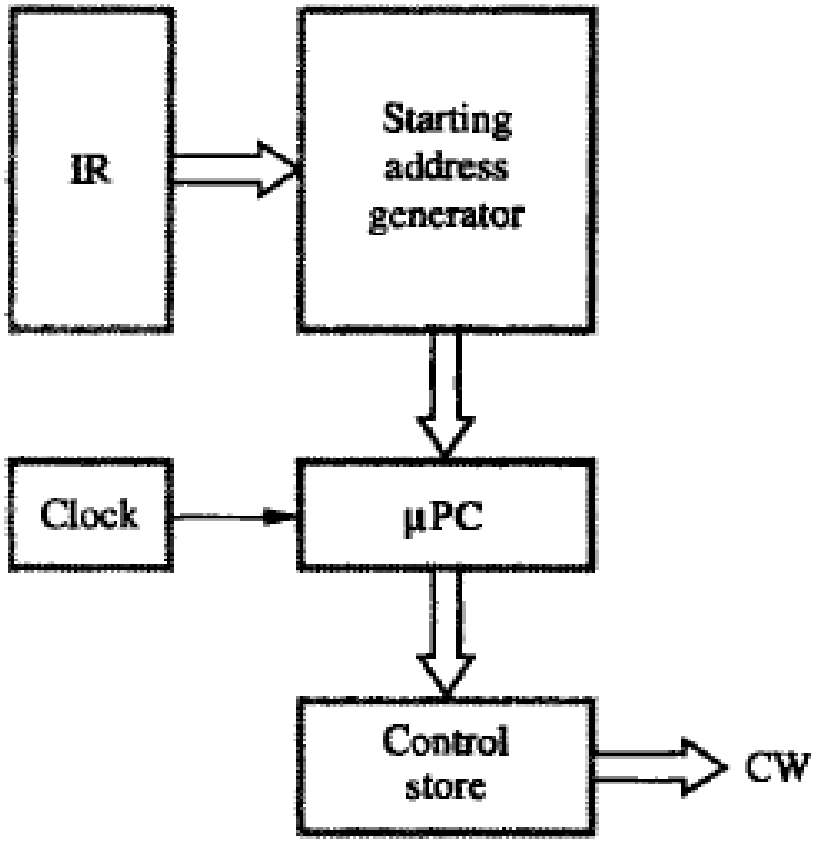
Micro-programmed Control

- Microprogramming is a method of control unit design which contains the **Control Signal Memory CM**.

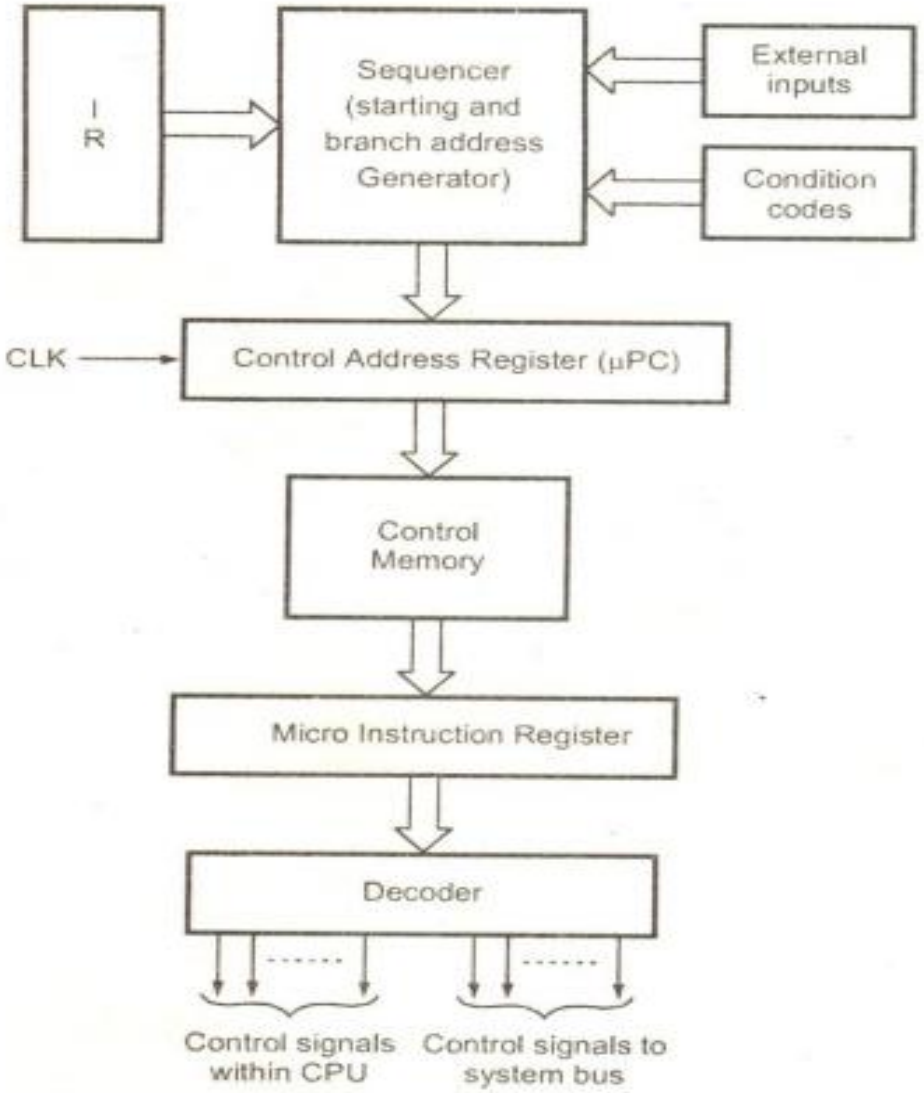
- The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM.

- A sequence of one or more micro operations designed to control specific operation, such as addition, multiplication is called a **Micro Program.**

Basic organization of a micro-programmed control unit



Micro-programmed Control Unit



- The address where these microinstructions are stored in CM is generated by **Microprogram Sequencer/Microprogram Controller**.

- The microprogram sequencer generates the address for microinstruction according to the instruction stored in the IR.

- The microprogrammed control unit contains
  - **Control Memory**
  - **Control Address Register**
  - **Micro Instruction Register**
  - **Microprogram Sequencer**

- The components of control unit work together as follows:
  - The **Control Address Register** holds the address of the next microinstruction to be read.
  - When address is available in control address register, the sequencer issues **READ command** to the Control Memory.
  - After issue of READ command, the word from the addressed location is read into the **Microinstruction Register.**
  - Now the content of the Micro Instruction Register generates control signals and next address information for the **Sequencer.**
  - The sequencer loads a new address into the control address register based on the next address information.

- **Advantages of Microprogrammed control**
  - It **simplifies the design** of control unit. Thus it is both, **cheaper** and **less error prone** implement.
  - Control functions are implemented in software rather than hardware.
  - The design **process is orderly and systematic**.
  - **More flexible**, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.
  - Complex function such as floating point arithmetic can be realized efficiently.

- **Disadvantages of Microprogrammed control**

  - A microprogrammed control unit is somewhat **Slower** than the hardwired control unit, because time is required to access the microinstructions from CM.

  - The **flexibility is achieved at some extra hardware cost** due to the control memory and its access circuitry.

Attribute	Hardwired Control	Microprogrammed Control
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible to accommodate new system specifications or new instructions	More flexible, to accommodate new system specification or new instructions redesign is required
Ability to handle large/complex instruction sets	Difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult	Easy
Design process	Complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instructionset size	Usually under 100 instructions	Usually over 100 instructions
ROM size	-	2K to 10K by 20-400 bit microinstructions
Chip area efficiency	Uses least area	Uses more area

Microinstructions

- A simple way to structure microinstructions is to **assign one bit position to each control signal** required in the CPU.

Example

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Micro-programmed control

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	:
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Micro-instruction Types

- **Vertical Micro-programming:** Each micro-instruction specifies single (or few) micro-operations to be performed.

- **Horizontal Micro-programming:** Each micro-instruction specifies many different micro-operations to be performed in parallel.

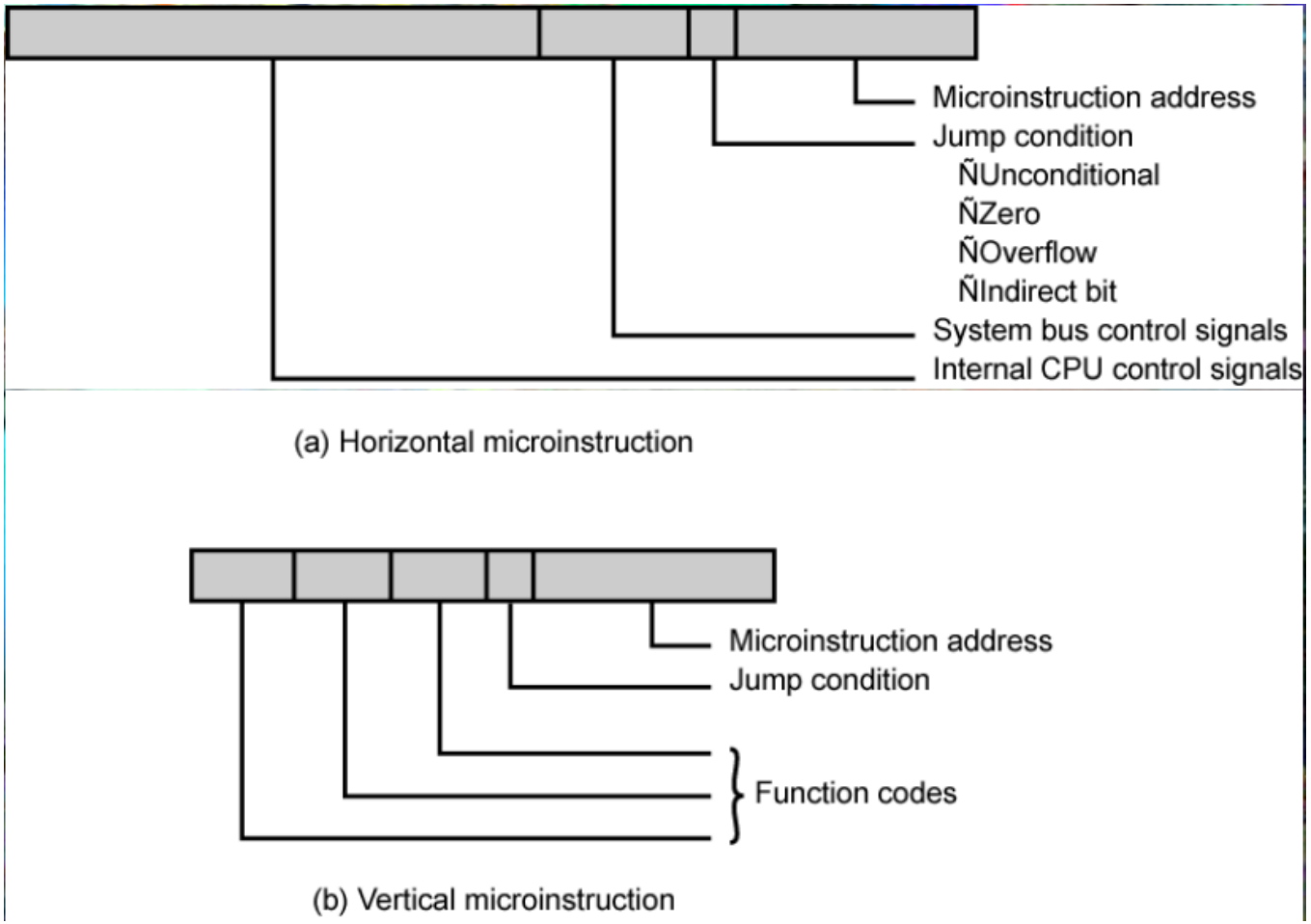
Vertical Micro-programming

- Width is narrow.
- ‘n’ control signals encoded into **‘log to base 2 of n’** bits.
- Limited ability to express parallelism.
- Considerable encoding of control information **requires external memory word decoder** to identify the exact control line being manipulated.

Horizontal Micro-programming

- Wide memory word.
- High degree of parallel operations possible.
- Little encoding of control information.

Typical Microinstruction Formats



Horizontal Microcode

- In the horizontal format, **each control signal is represented by a single bit in the control word**. Thus, if the design has 500 control signals, this will require 500 bits in each control word to store the control bits.

- In this format, the **control store looks horizontal in shape since the control words are wide**.

- **The disadvantage of the horizontal format is that the size of the control store is large**. However, it has the advantage of speed of operation as the control signals will be ready as soon as the control word is fetched from the control store.

Vertical Microcode

- In the vertical microcode organization, the following steps are performed:
  - Identify the number of distinct control words in the design.
  - Encode each distinct control word by assigning a unique n-bit code to it, where n is $\log_2$ (number of distinct control words).
  - Instead of storing the actual control signals that need to be generated, only the n-bit code is stored for each CW.
  - Use a $nx2^n$ decoder to generate a decoded signal for each distinct control word.
  - To generate the control signals, use an OR gate based on the decoded control word signals, for each control signal in the design.

Differences of horizontal and vertical micro code

S.No	Horizontal	Vertical
1	Long formats	Short formats
2	Ability to express a high degree of parallelism	Limited ability to express parallel microoperations
3	Little encoding of the control information	Considerable encoding of the control information
4	Useful when higher operating speed is desired	Slower operating speeds

Microprogram Sequencing

- The task of microprogram sequencing is done by **Microprogram Sequencer**.

- 2 important factors must be considered while designing the microprogram sequencer:
  - **The Size of the Microinstruction**
  - **The Address Generation Time**

- The **size of the microinstruction should be minimum** so that the size of control memory required to store microinstructions is also less. **This reduces the cost of control memory.**

- The **address generation time should be less**, microinstruction can be executed in less time resulting better throughout.

- During execution of a microprogram the address of the next microinstruction can be obtained from 3 sources:

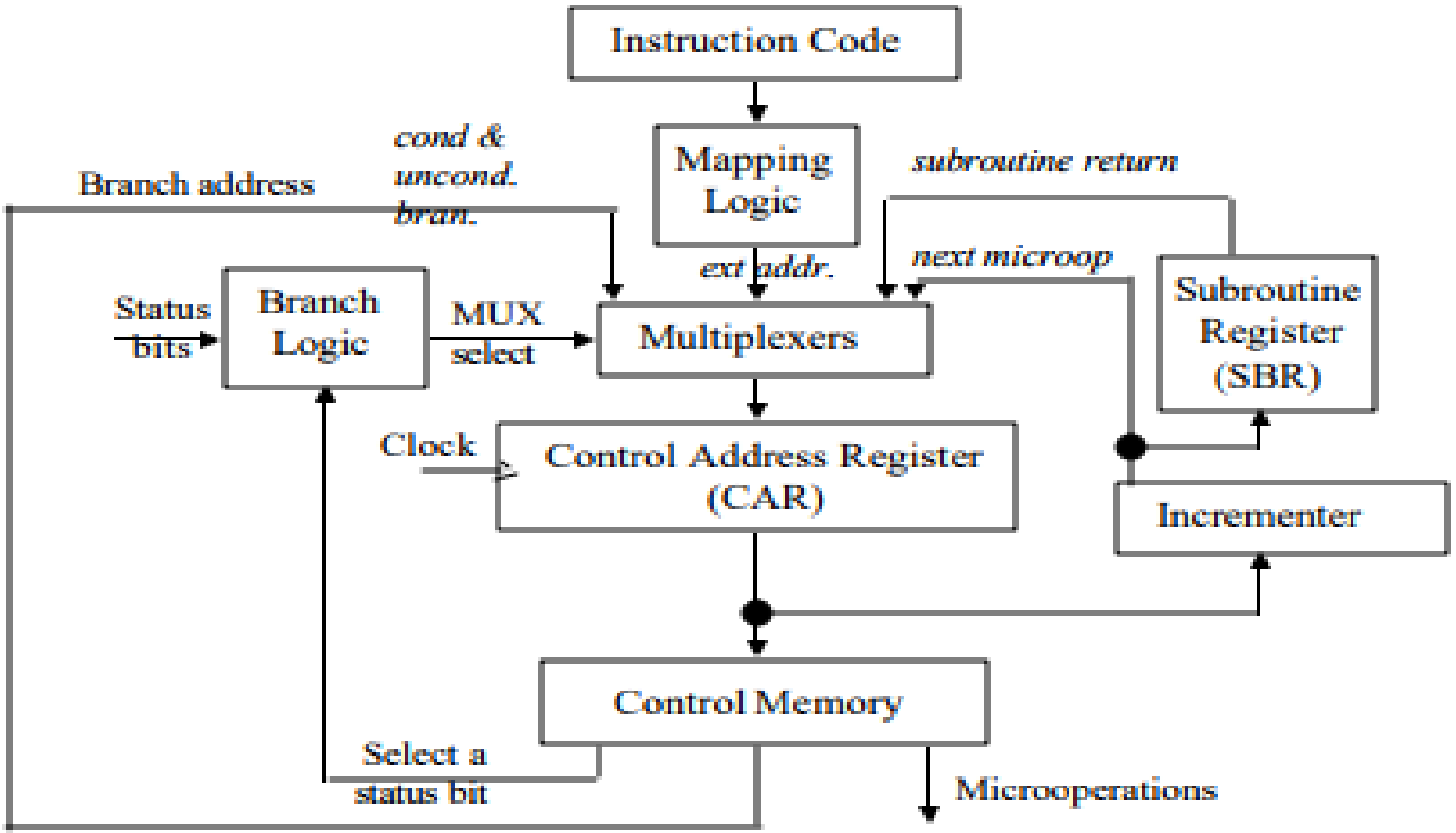
  i. Determined by Instruction Register

  ii. Next sequential address

  iii. Branch

- Microinstructions can be shared using microinstruction branching.

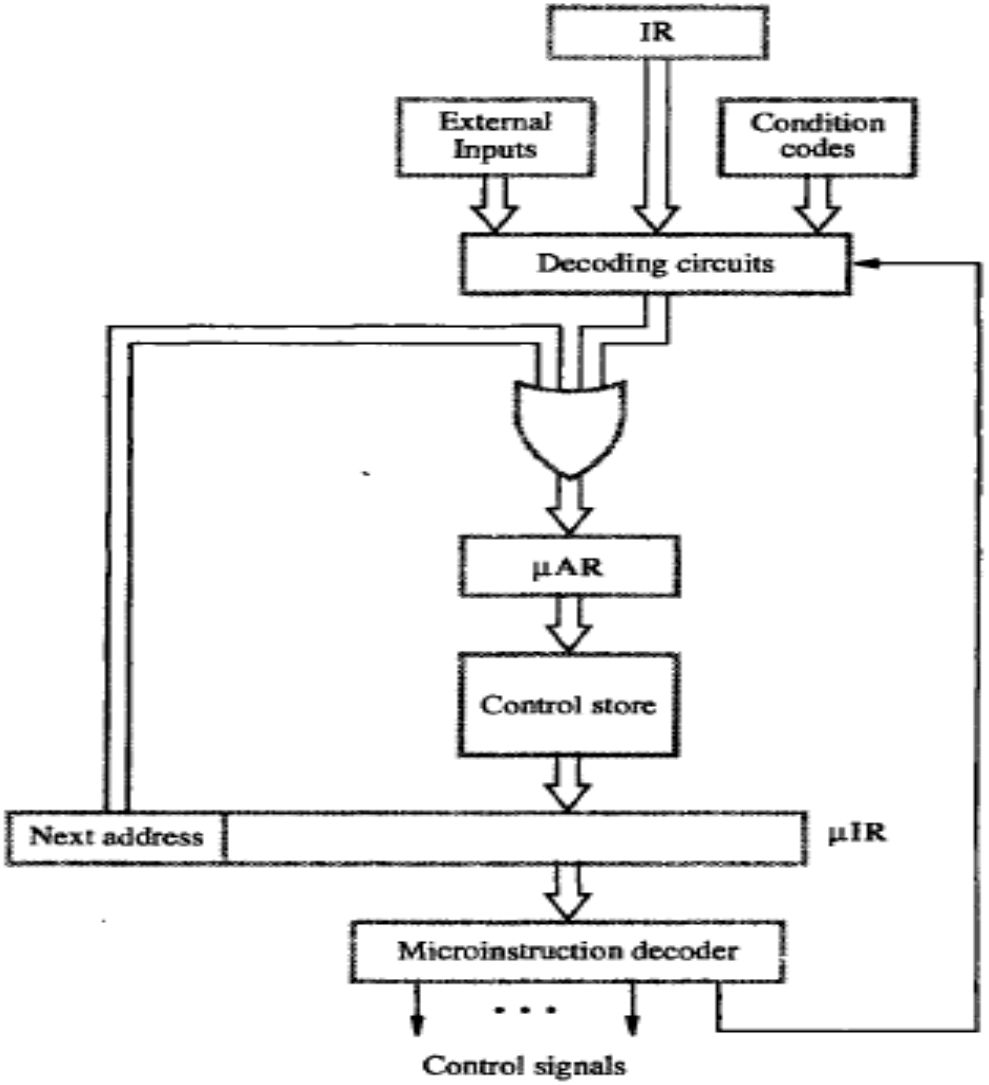
Micro Instruction Sequencing



Wide-Branch Addressing

- Generating branch addresses becomes more difficult as the number of branches increases.
- In such situations **Programmable Logic Array (PLA)** can be used to generate the required branch addresses.
- The simple and inexpensive way of generating branch addresses is known as **Wide-branch Addressing.**
- The opcode of a machine instruction is translated into the starting address of the corresponding micro-routine.
- This is achieved by connecting the opcode bits of the instruction register as inputs to the PLA, which acts as a decoder.
- The output of the PLA is the address of the desired microroutine.

Microinstruction with next address field



Pre-fetching Microinstructions

One drawback of microprogrammed control is that it leads to a slower operating speed because of the time it takes to fetch microinstructions from the control store. Faster operation is achieved if the next microinstruction is prefetched while the current one is being executed. In this way, the execution time can be overlapped with the fetch time.

Prefetching microinstructions presents some organizational difficulties. Sometimes the status flags and the results of the currently executed microinstruction are needed to determine the address of the next microinstruction. Thus, straightforward prefetching occasionally prefetches a wrong microinstruction. In these cases, the fetch must be repeated with the correct address, which requires more complex hardware. However, the disadvantages are minor, and the prefetching technique is often used.

Emulation

- Emulation allow us to **replace obsolete equipment with more up to date machines.**
- If replacement computer fully emulates the original one, then no software changes have to be made to run existing programs.
- Thus, **emulation facilitates transition to new computer systems with minimal disruption.**
- Emulation will be easy if machine involves are having same architecture.