UNIT 6
Basic Processing Unit

Objectives

- How a processor execute 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.

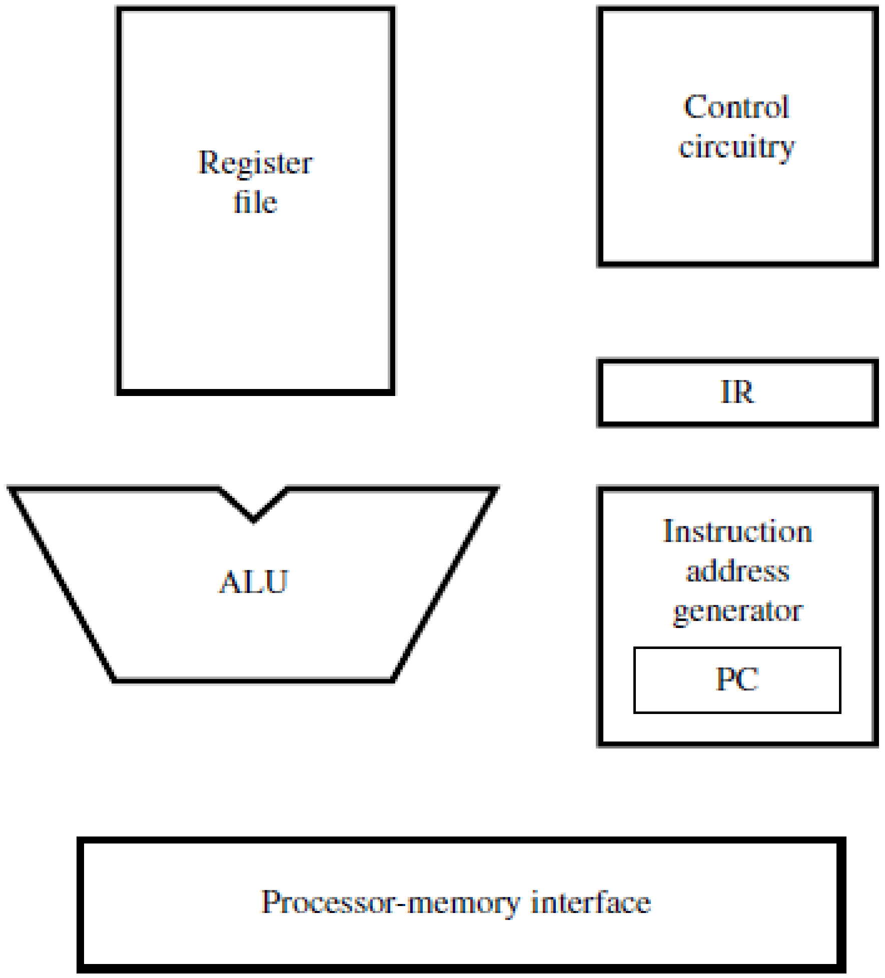
- Steps 1 and 2 are part of Fetch phase
- Step 3 continues the execution

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

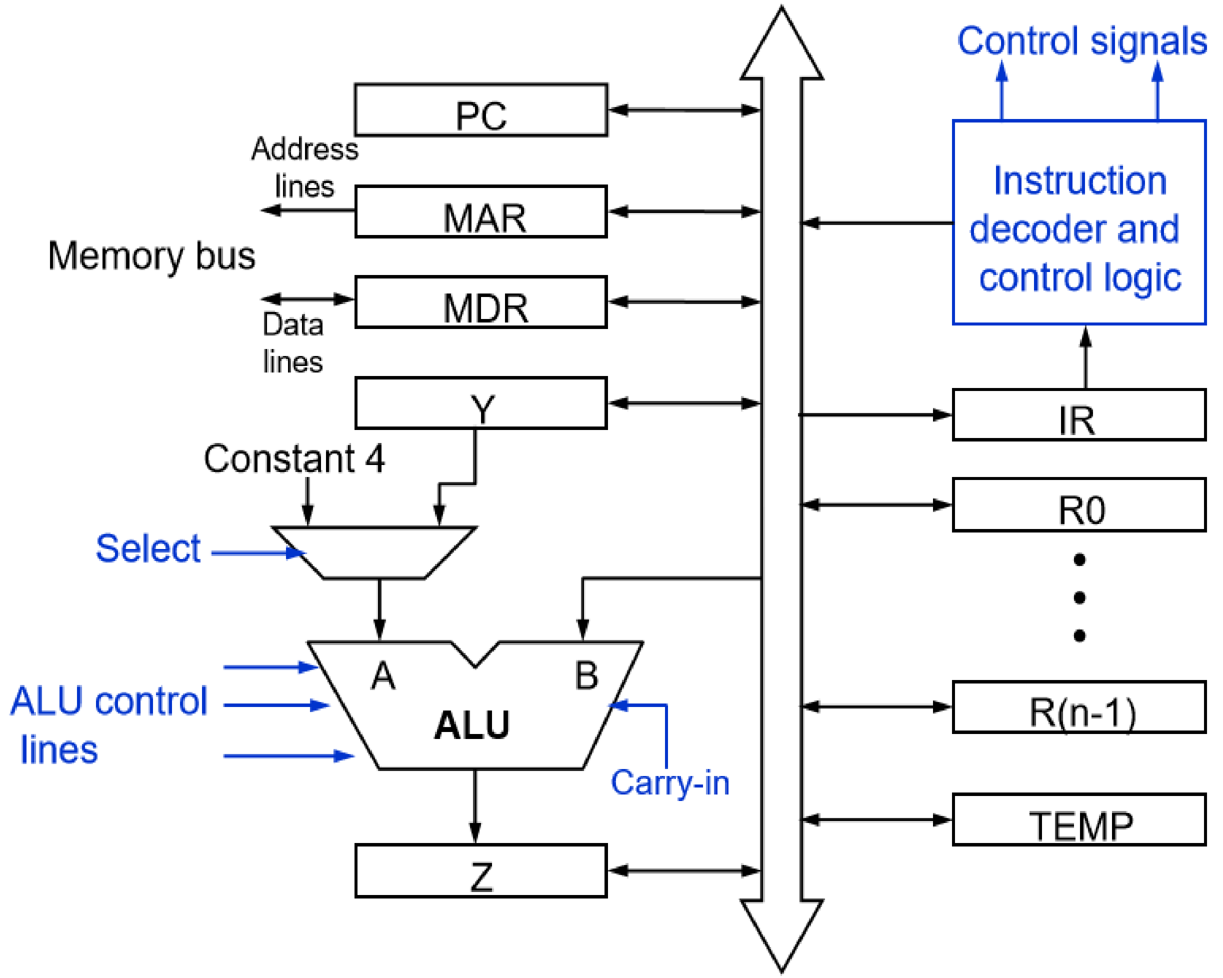
- Performing the operation specified in the instruction constitutes the ***Instruction Execution Phase***.

- 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

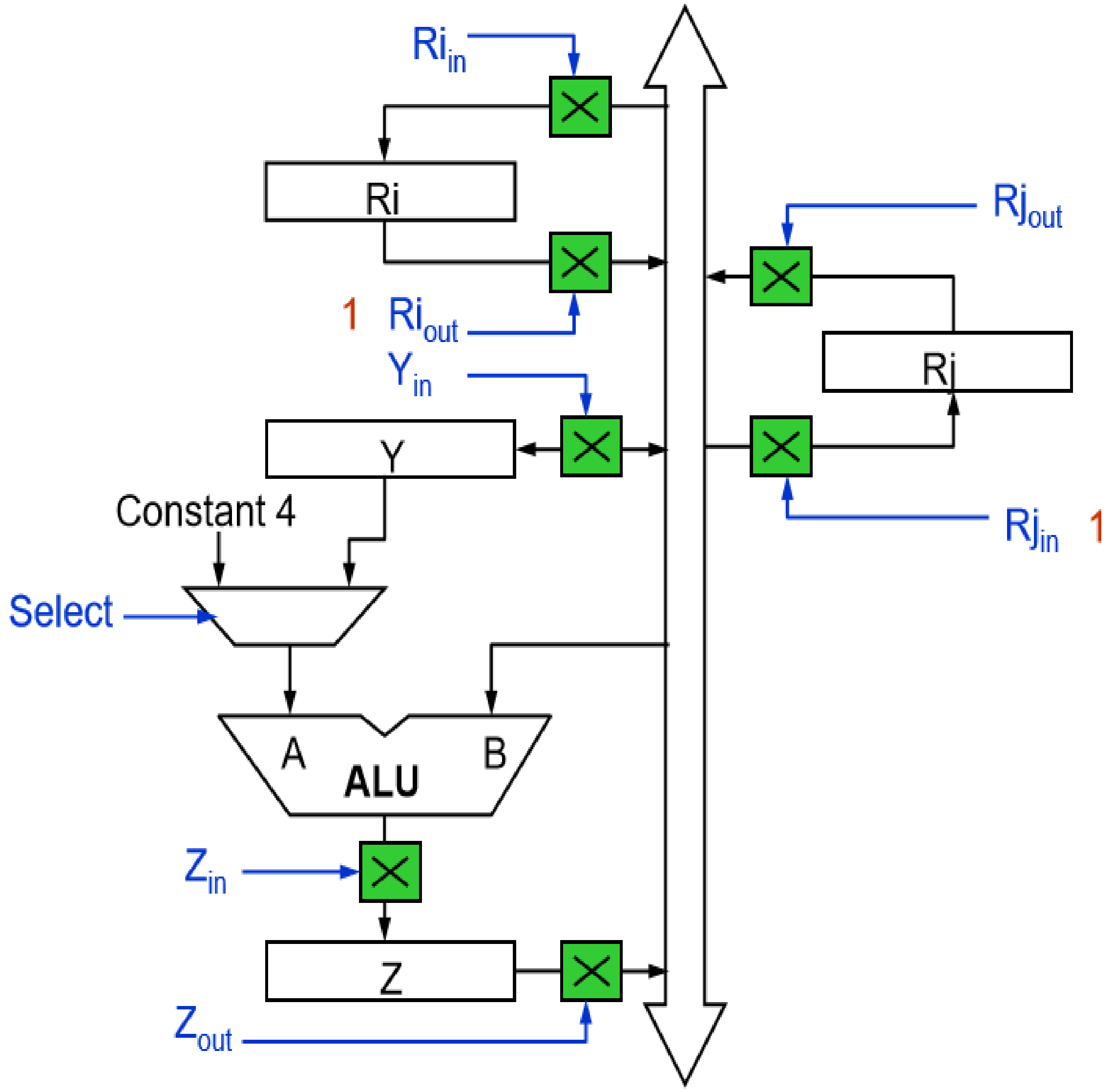


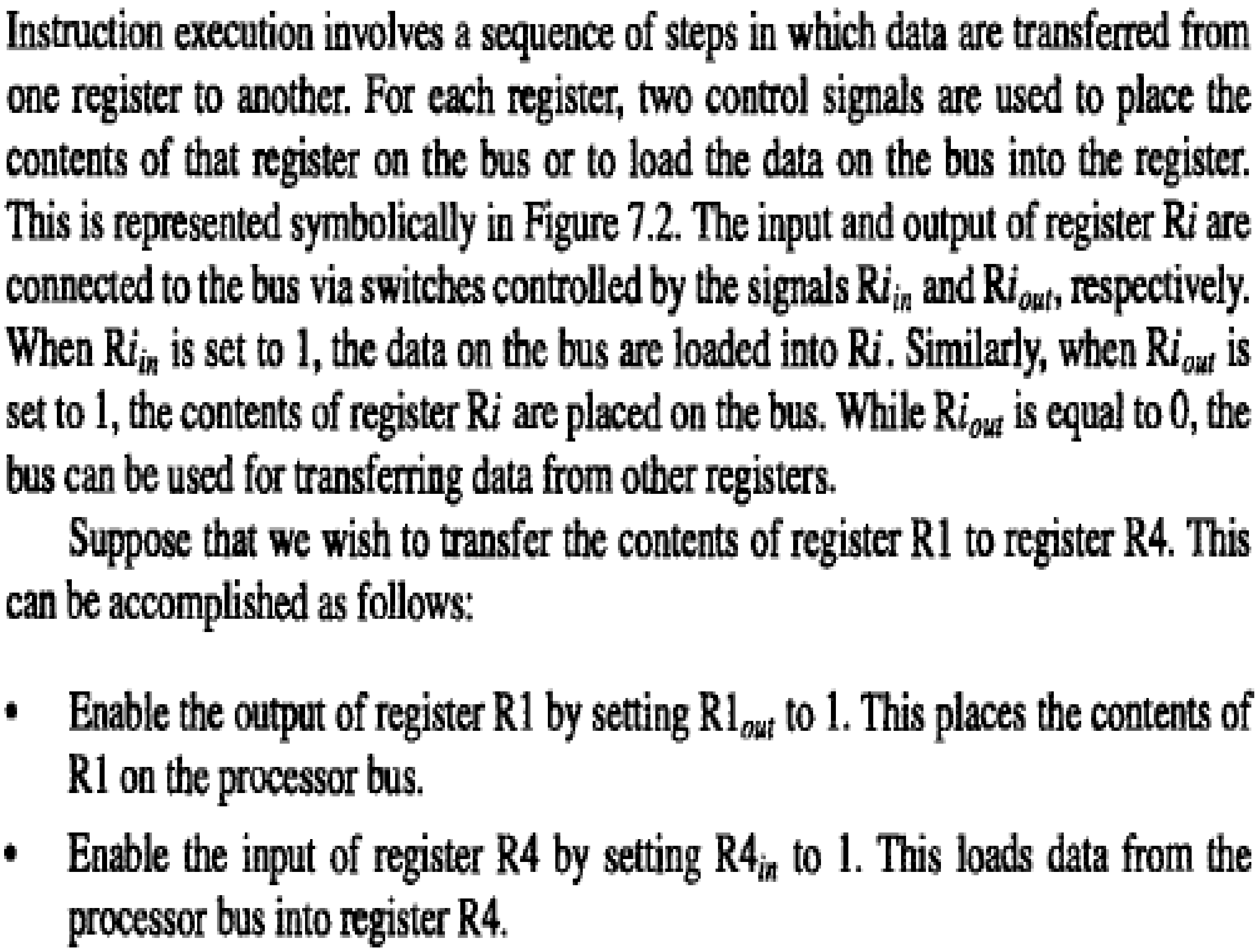
- **MAR:** Memory Address Register
- **MDR:** Memory Data Register
- **Y, Z, TEMP:** Temporary Registers used by processor.
- **R0 To R(n-1):** General Purpose Registers
- **PC:** Program Counter
- **IR:** Instruction Register

- An instruction can be executed by performing following operation:

1. Transfer a word of data from one processor register to another or to ALU.
2. Perform an arithmetic or logical operation and store result in processors register.
3. Fetch the content of given memory location and load them into a processor register.
4. Store the word of data from a processor register into a given memory location.

Register Transfers

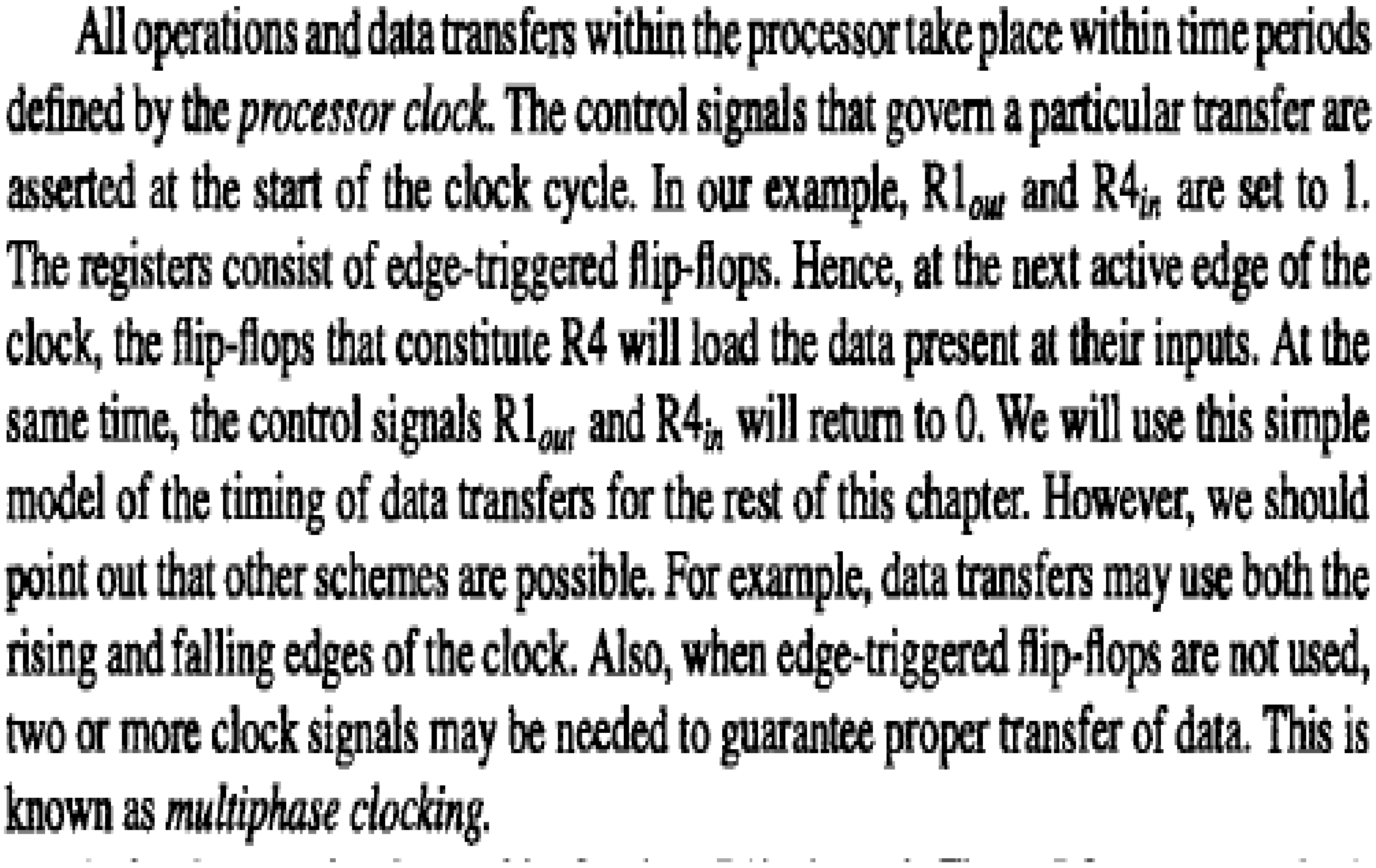




Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register. This is represented symbolically in Figure 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:

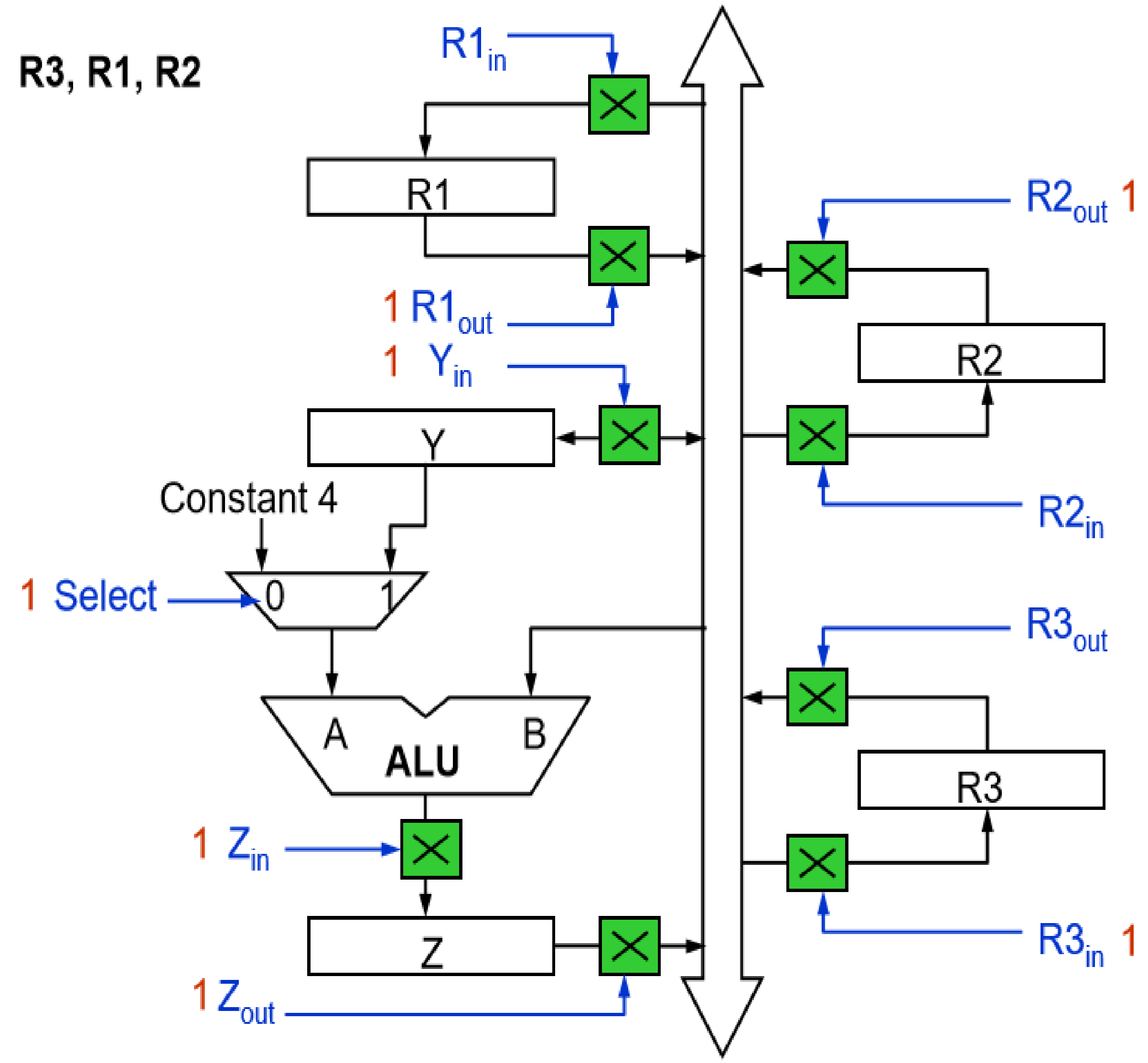
- Enable the output of register R1 by setting $R1_{out}$ to 1. This places the contents of R1 on the processor bus.
- Enable the input of register R4 by setting $R4_{in}$ to 1. This loads data from the processor bus into register R4.



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

Add R3, R1, R2



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}, $R3_{in}$

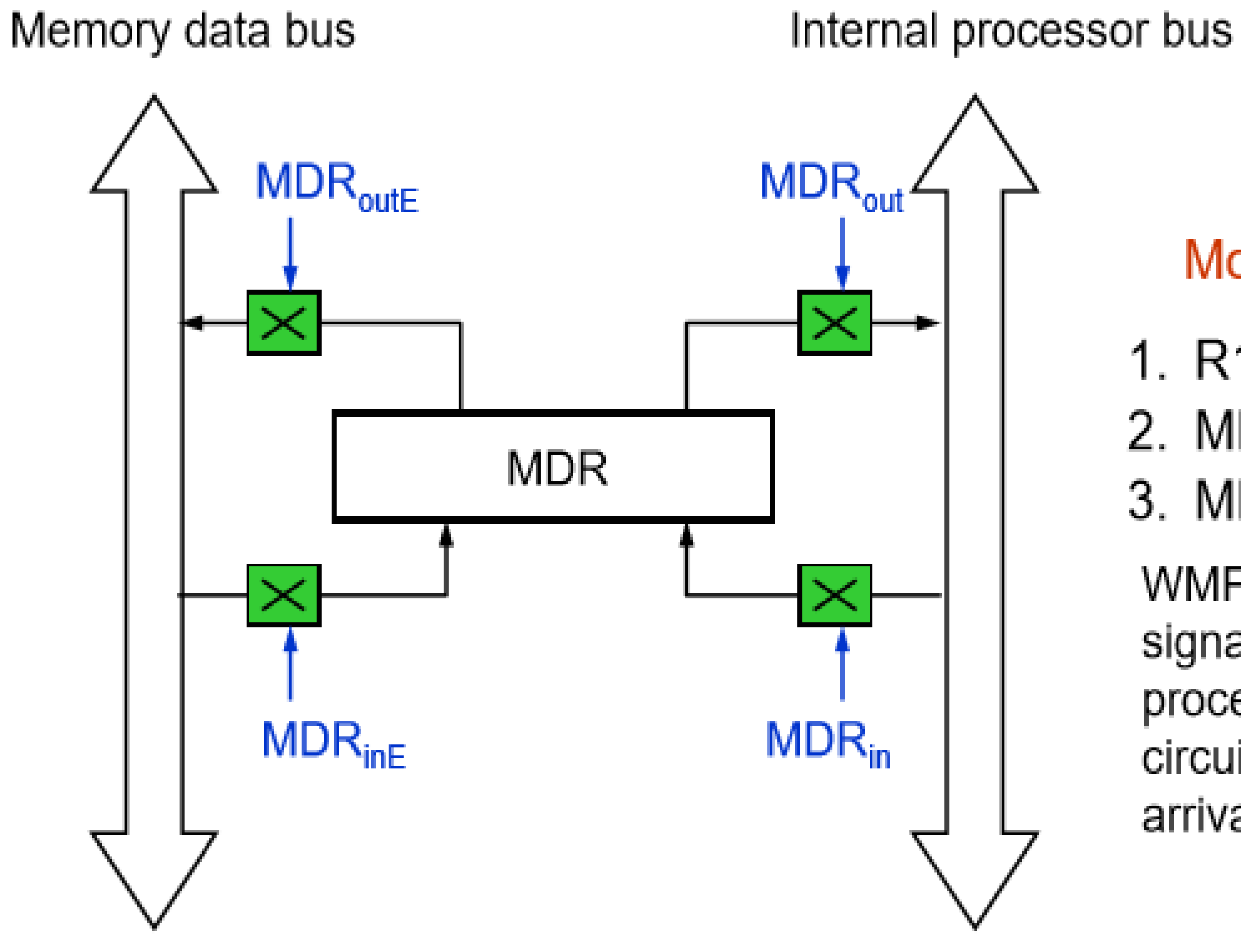
Read operation

- **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]

MFC: Processor waits until it receives an indication that requested read operation has been completed.

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

- **MOVE R2,(R1)**

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

Execution of Complete Instruction

Consider instruction:

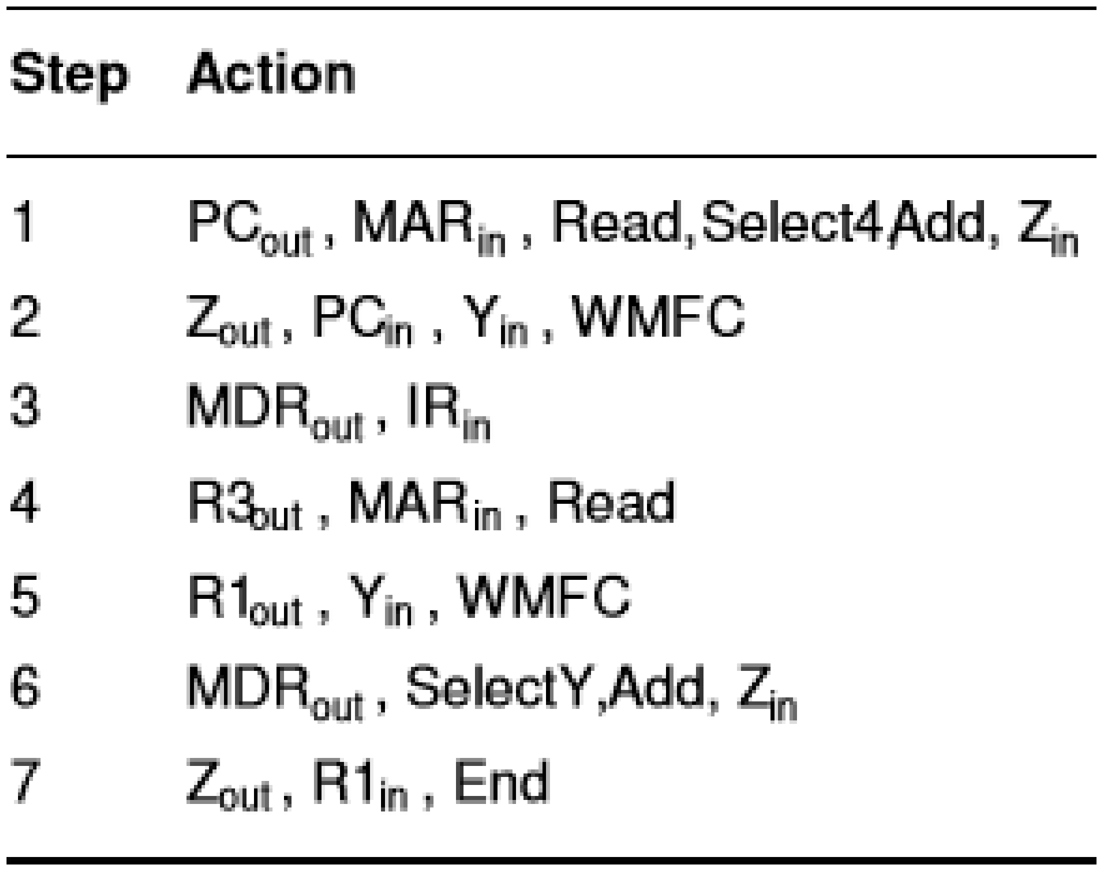
Add (R3) ,R1

(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

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

Hardwired Control

- 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**

- **Hardwired Control:** In hardwired control, control signals required inside processor can be generated using control step counter and decoder/encoder circuit.

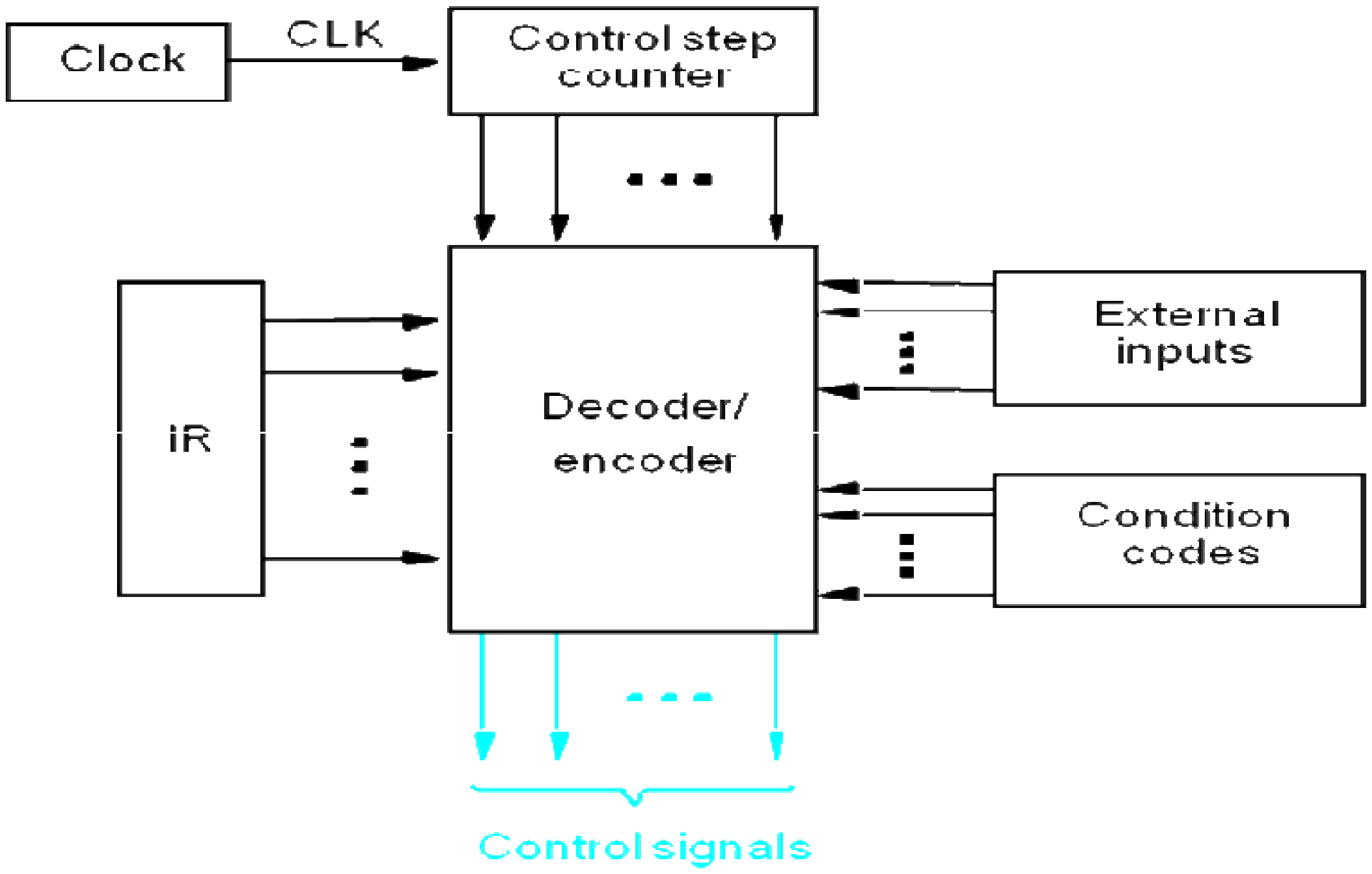
- **Micro-programmed Control:** In micro-programmed control, control signals are generated by a program similar to machine language programs.

- Eg. Consider set of microinstructions on slide no. 20. Consider each step in a sequence is completed in one clock period. Processor uses counter to keep track of each clock period.

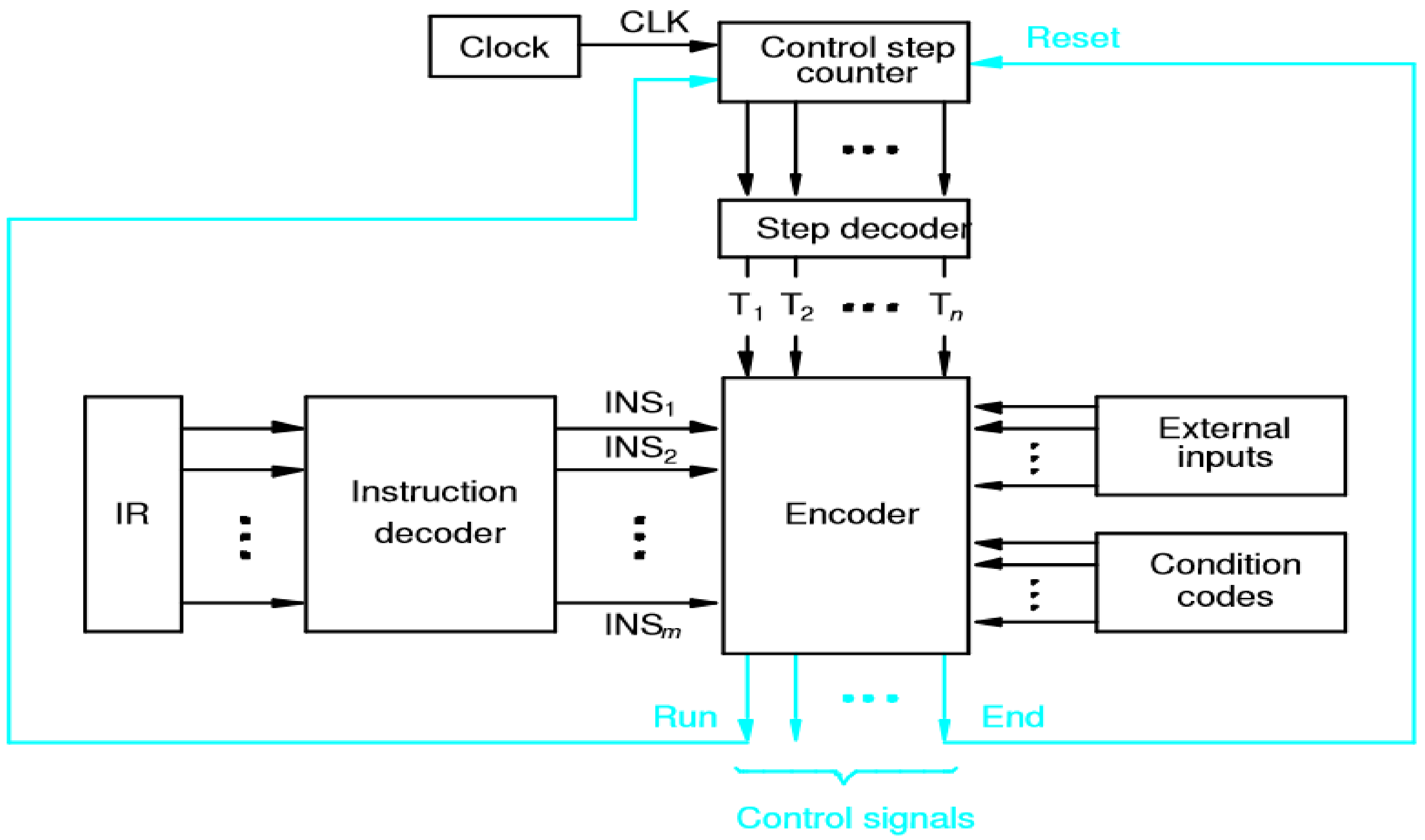
- Each step or count corresponds to one control step. The required control signals are determined by following information:

  - **Content of control step counter**
  - **Contents of IR**
  - **Contents of Conditional flags**
  - **External control signals (eg. MFC, Interrupt requests)**

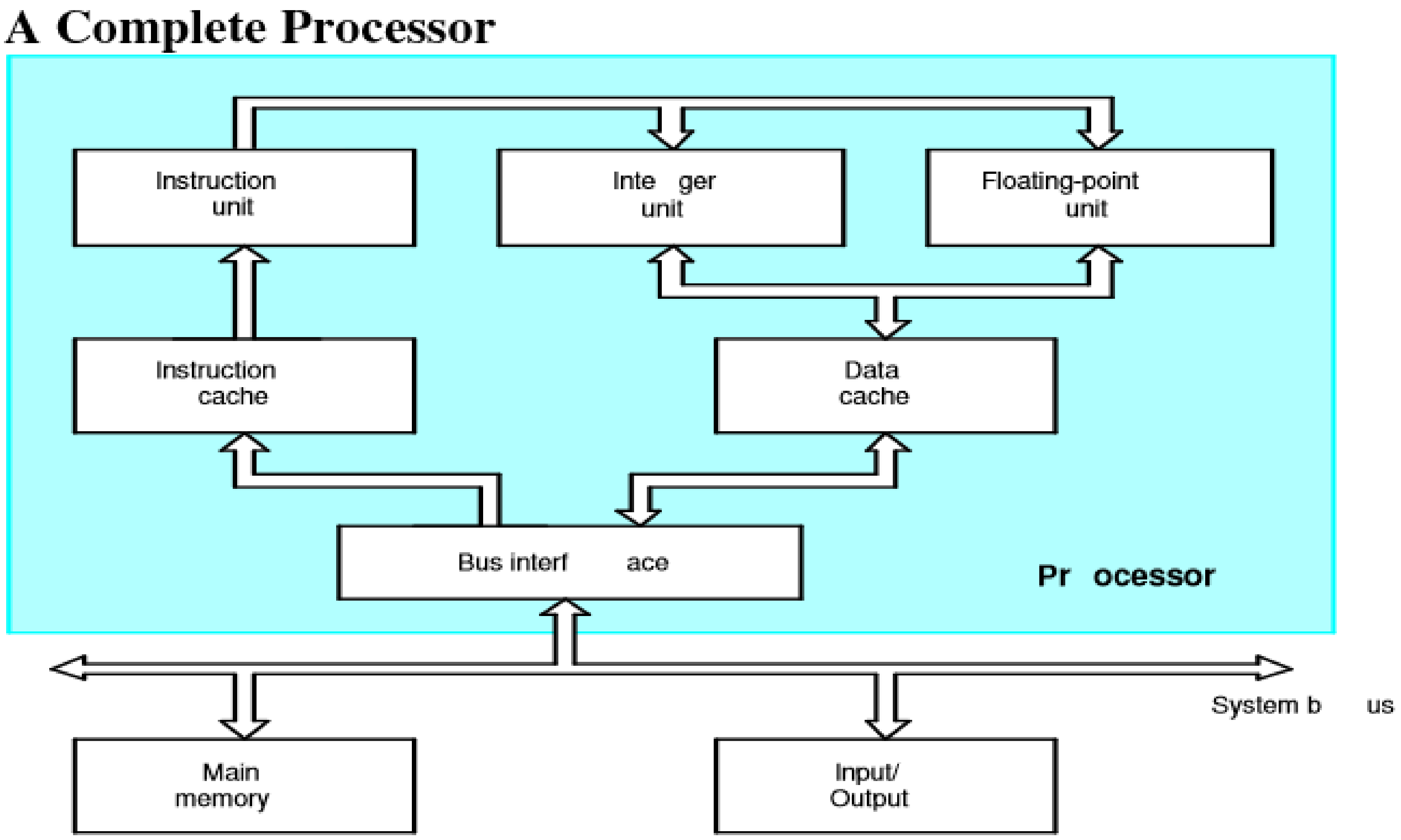
Control Unit Organization



Detailed Control Design



Block Diagram of a Complete Processor

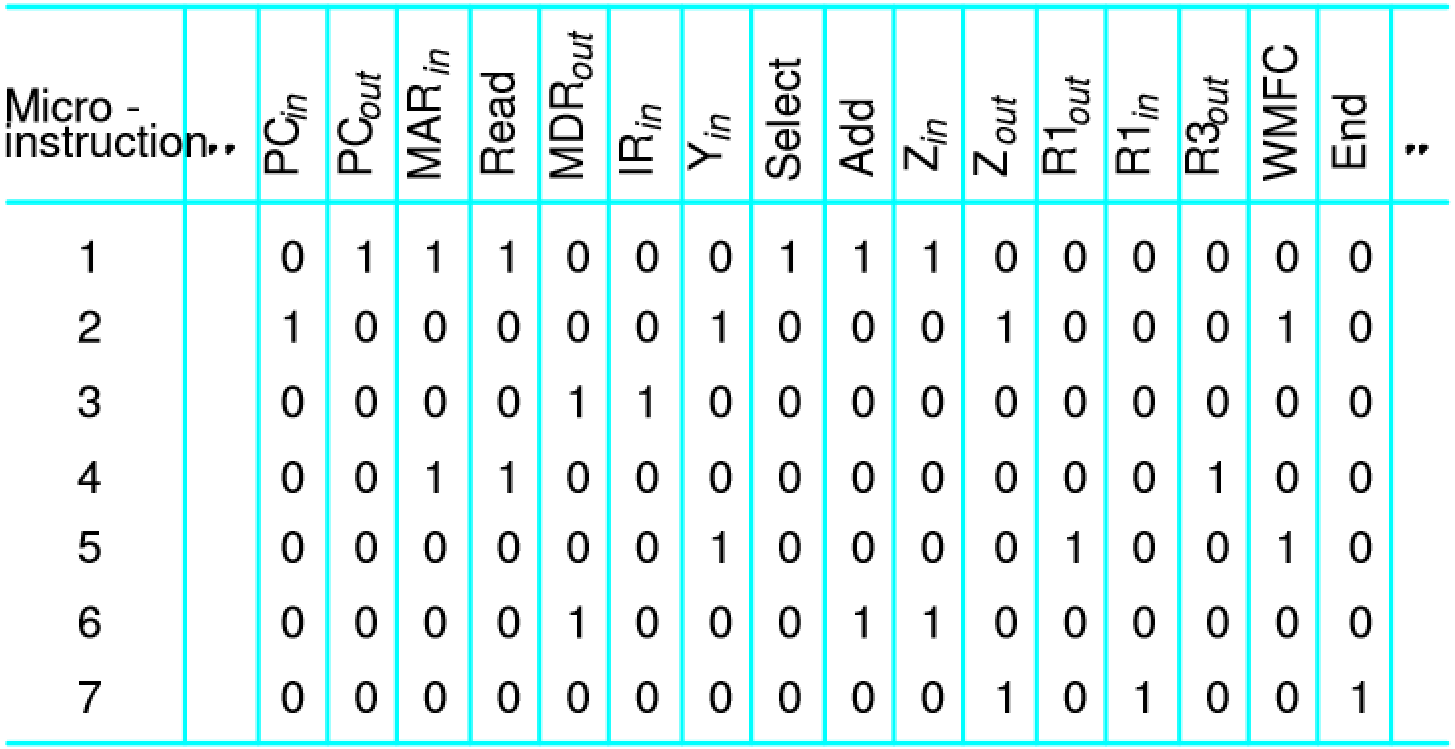


Micro-programmed Control

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

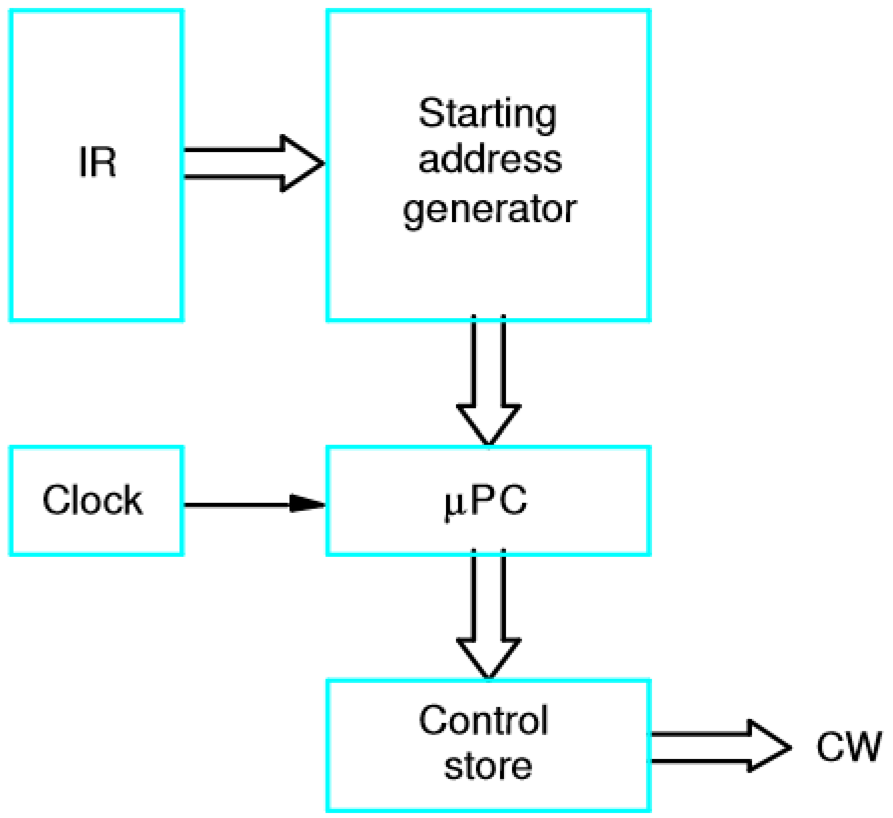
- Consider set of microinstructions on slide no. 20.

Micro-programmed control

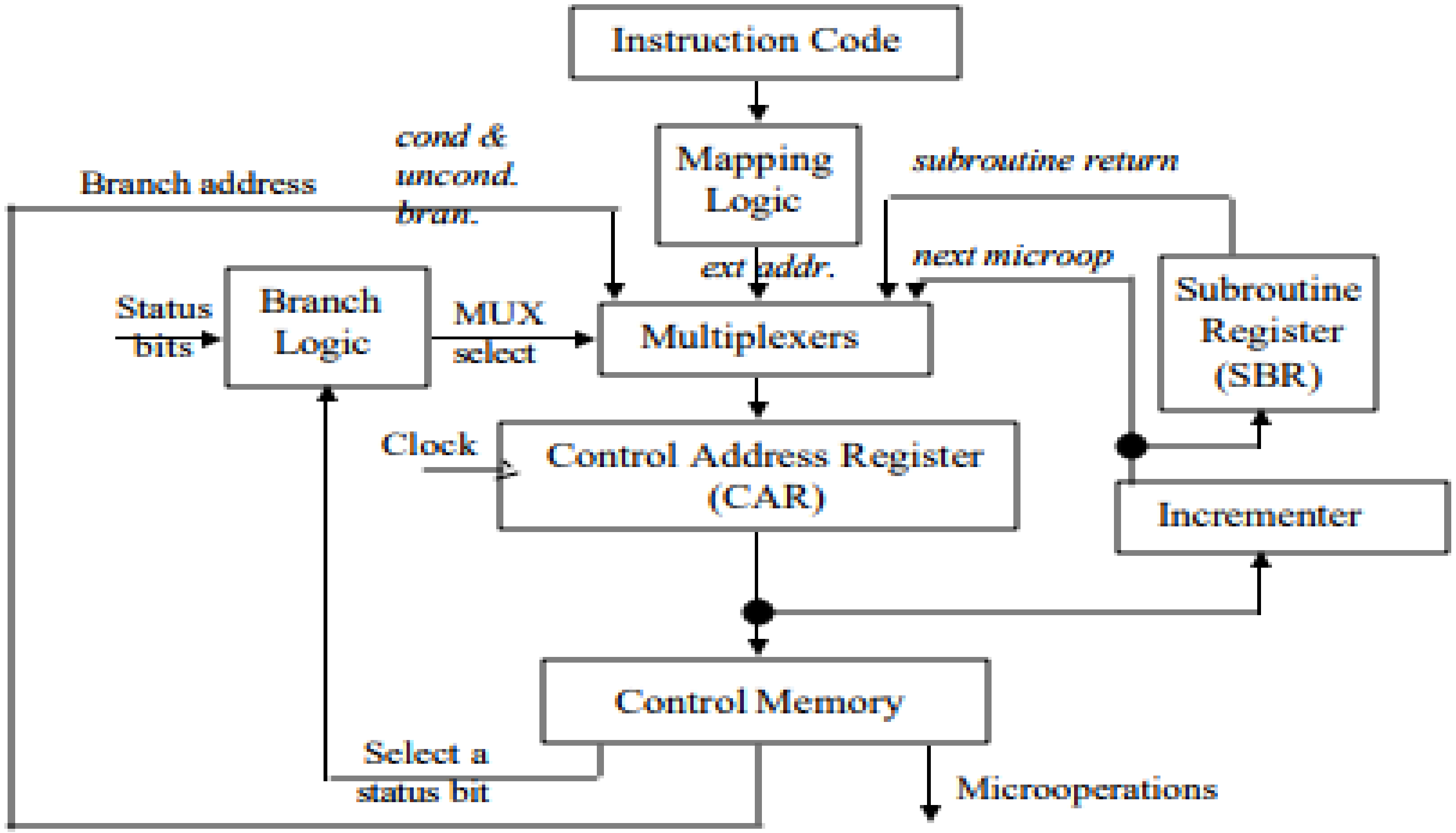


Micro - instruction..		PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0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

Basic organization of a micro-programmed control unit



Micro Instruction Sequencing



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.

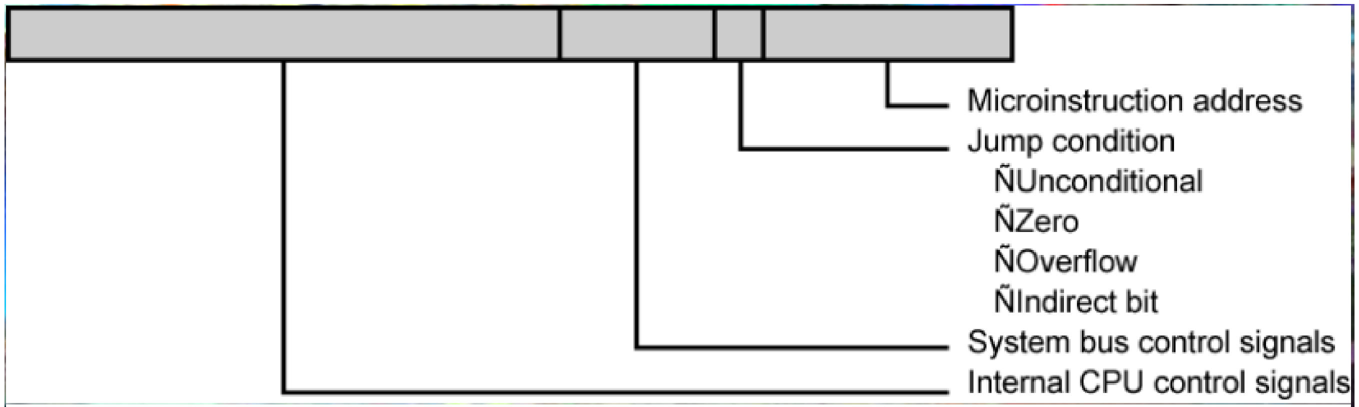
Vertical Micro-programming

- 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.

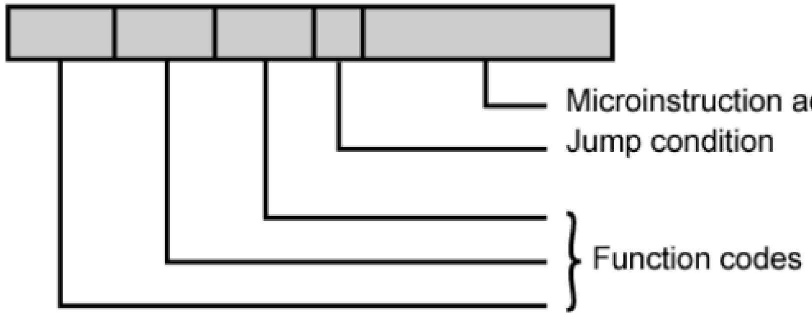
Horizontal Micro-programming

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

Typical Microinstruction Formats



(a) Horizontal microinstruction



(b) Vertical microinstruction

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 $n \times 2^n$ 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

Horizontal Micro Code

- In this types of code the micro code contains the control signal without any intermediary.

- Horizontal micro code instruction contain a lot of signals and hence due to that the number of bits also increase.

Vertical Micro code.

- In case of vertical micro code every action is encoded in density.

- Vertical micro code are slower but they take less space and their actions at execution time need to be decoded to a signal.