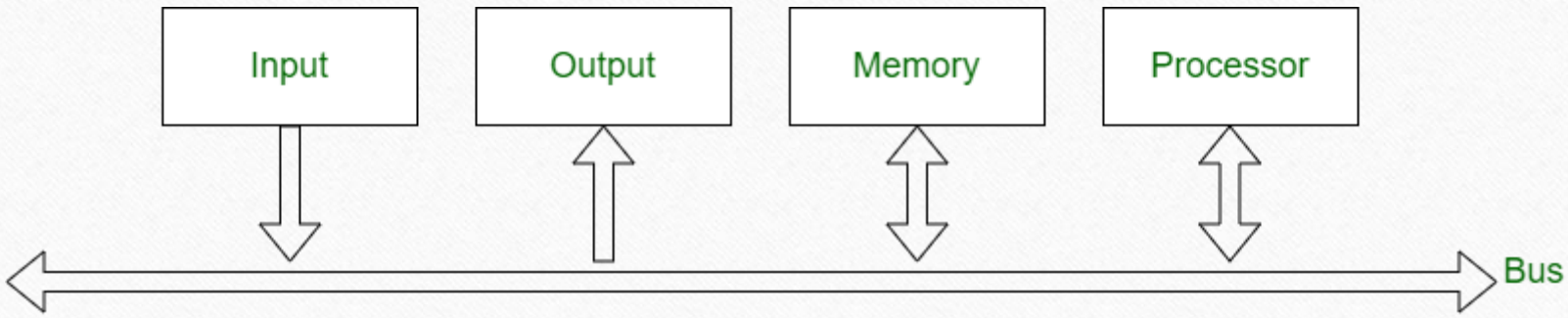
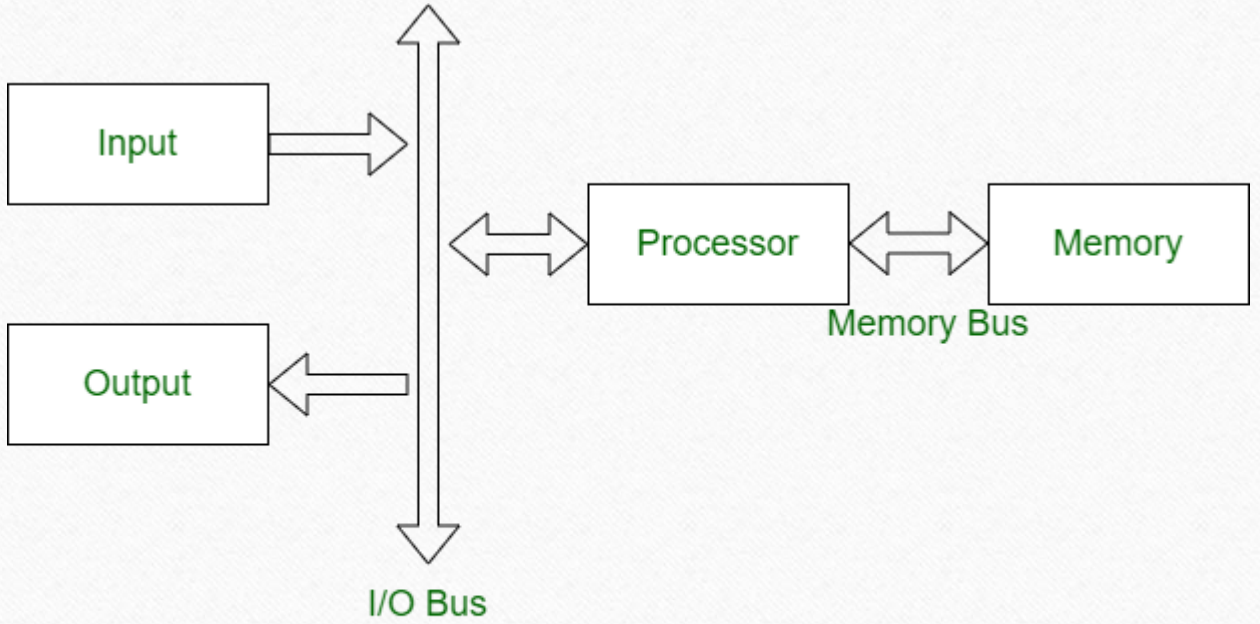
Fundamental Concepts and processor organization

Single bus Structure



Single Bus Structure

Double bus structure



Double Bus Structure

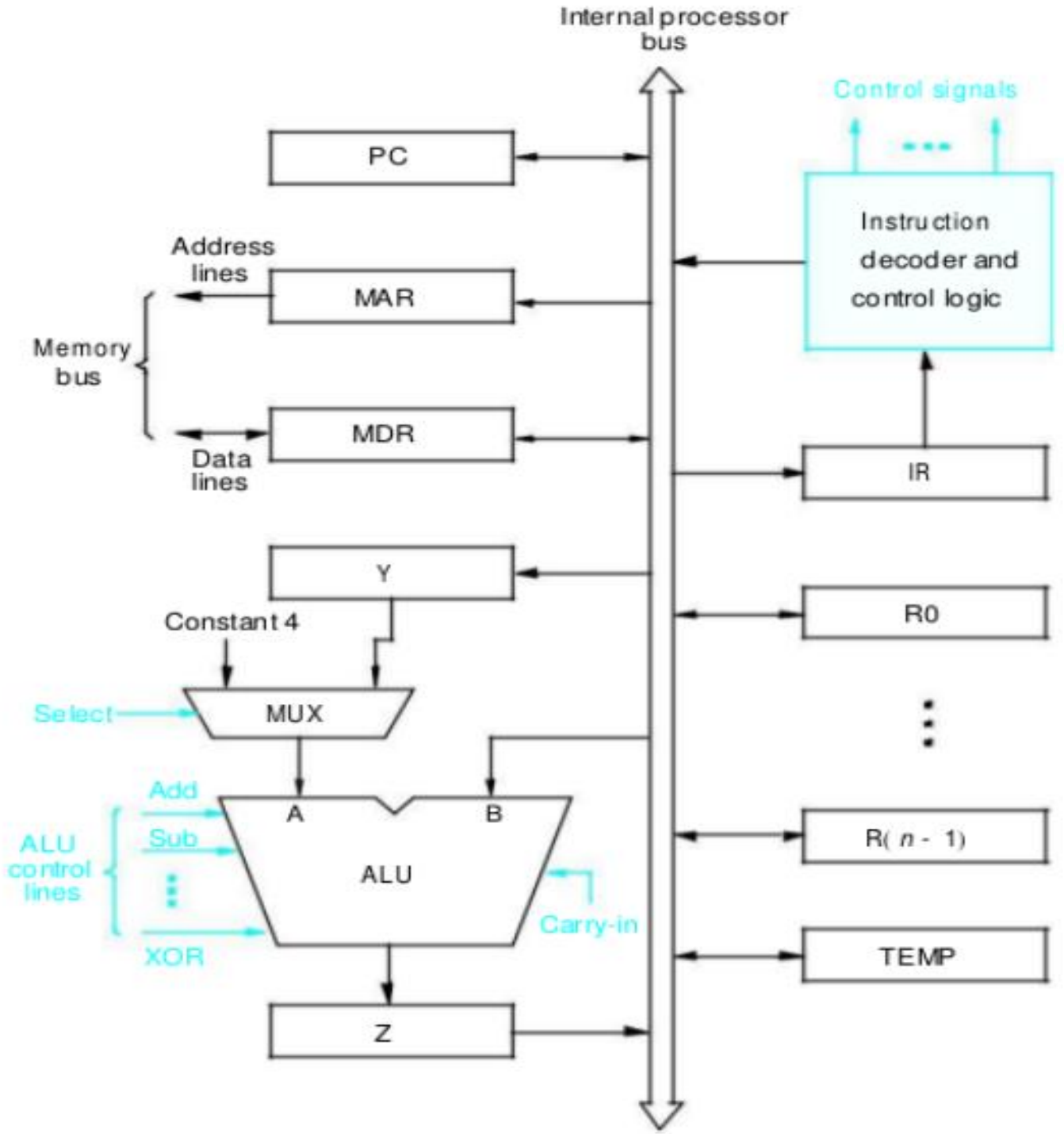
Differences between Single Bus and Double Bus Structure

S. No.	Single Bus Structure	Double Bus Structure
1.	The same bus is shared by three units (Memory, Processor, and I/O units).	The two independent buses link various units together.
2.	One common bus is used for communication between peripherals and processors.	Two buses are used, one for communication from peripherals and the other for the processor.
3.	The same memory address space is utilized by I/O units.	Here, the I/O bus is used to connect I/O units and processor and other one, memory bus is used to connect memory and processor.
4.	Instructions and data both are transferred in same bus.	Instructions and data both are transferred in different buses.
5.	Its performance is low.	Its performance is high.
6.	The cost of a single bus structure is low.	The cost of a double bus structure is high.

Differences between Single Bus and Double Bus Structure

S. No.	Single Bus Structure	Double Bus Structure
7.	Number of cycles for execution is more.	Number of cycles for execution is less.
8.	Execution of the process is slow.	Execution of the process is fast.
9.	Number of registers associated are less.	Number of registers associated are more.
10.	At a time single operand can be read from the bus.	At a time two operands can be read.
11.	Advantages- •Less expensive •Simplicity	Advantages- •Better performance •Improves Efficiency

Processor Organization (Single bus organization of a data path inside a processor)



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.**

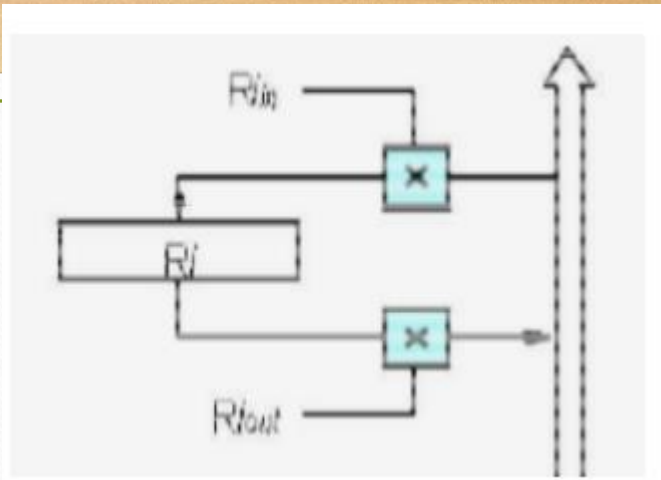
Register Transfers

- Register is a very fast computer memory, used to store data/instruction in-execution.
- A Register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing binary information of n-bits.
- The data transfer from one register to another is named in representative design using a replacement operator. The statement is

R2←R1

It indicates a transfer of the content of register R1 into register R2. It labelled a replacement of the content of R2 by the content of R1. The content of the source register R1 does not shift after the transfer.

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 register.**
- **The input and output of register Ri in and Ri out is set to 1,**
- **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 Ri are placed on the bus.**
- **While Ri out is equal to 0, the bus can be used for transferring data from other registers.**

Typically, most of the users want the transfer to occur only in a predetermined control condition. This can be shown by following if-then statement:

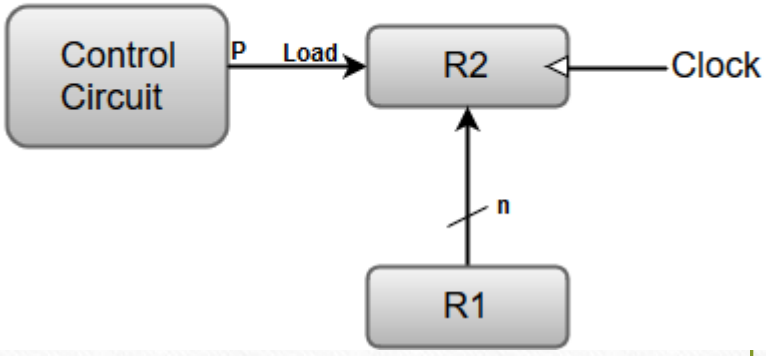
If (P=1) then (R2 ← R1); Here P is a control signal generated in the control section.

For instance, the following statement defines the data transfer operation under a specific control function (P).

P: R2 ← R1

- Here, the letter 'n' indicates the number of bits for the register. The 'n' outputs of the register R1 are connected to the 'n' inputs of register R2.
- A load input is activated by the control variable 'P' which is transferred to the register R2.

Transfer from R1 to R2 when P = 1:



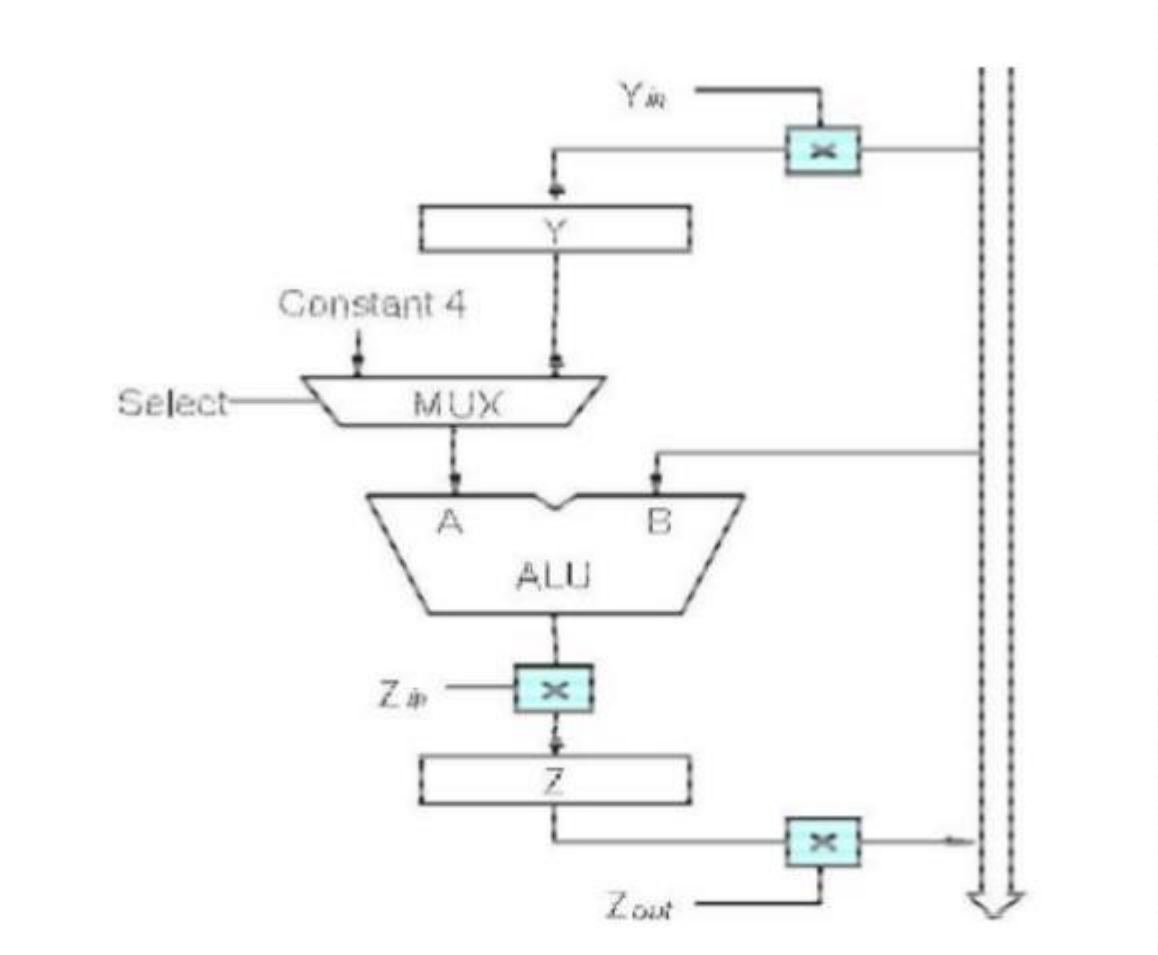
Example: Move R1, R4

Transfer the contents of register R1 to register R4.

This can be accomplished as follows.

- **Enable the output of registers R1 by setting R1out to 1. This places the contents of R1 on the processor bus.**
- **Enable the input of register R4 by setting R4in 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.

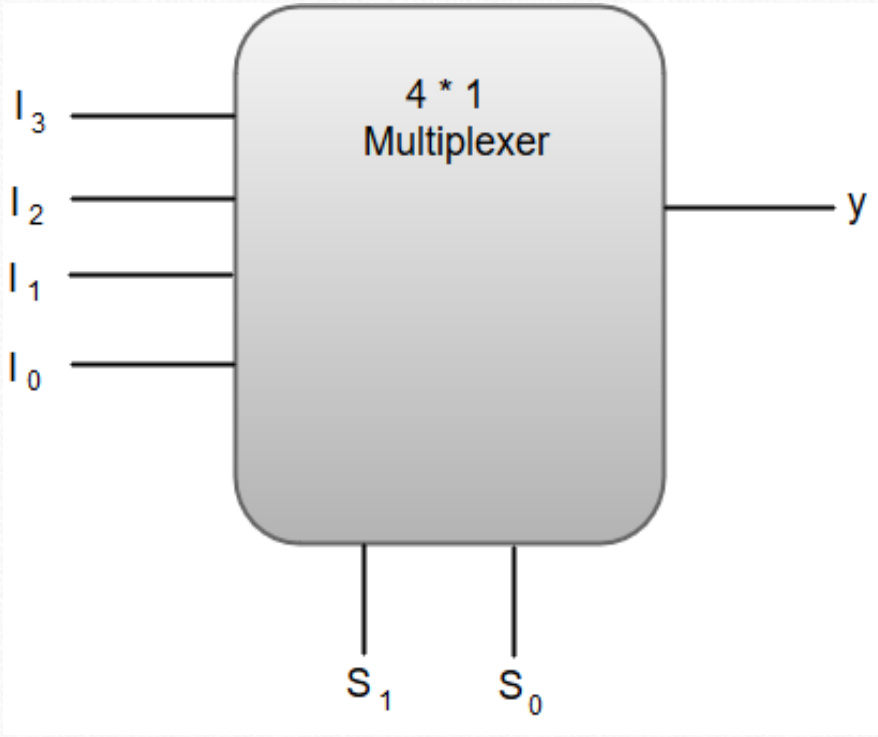
Performing an Arithmetic or Logic Operation



Multiplexers

- A Multiplexer (MUX) can be described as a combinational circuit that receives binary information from one of the 2^n input data lines and directs it to a single output line.
- The multiplexer is often called as data selector since it selects only one of many data inputs.

Out of these four input data lines, a particular input data line will be connected to the output based on the combination of inputs present at these two selection lines.



The function table for a 4 * 1 Multiplexer can be represented as:

S1	S0	y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

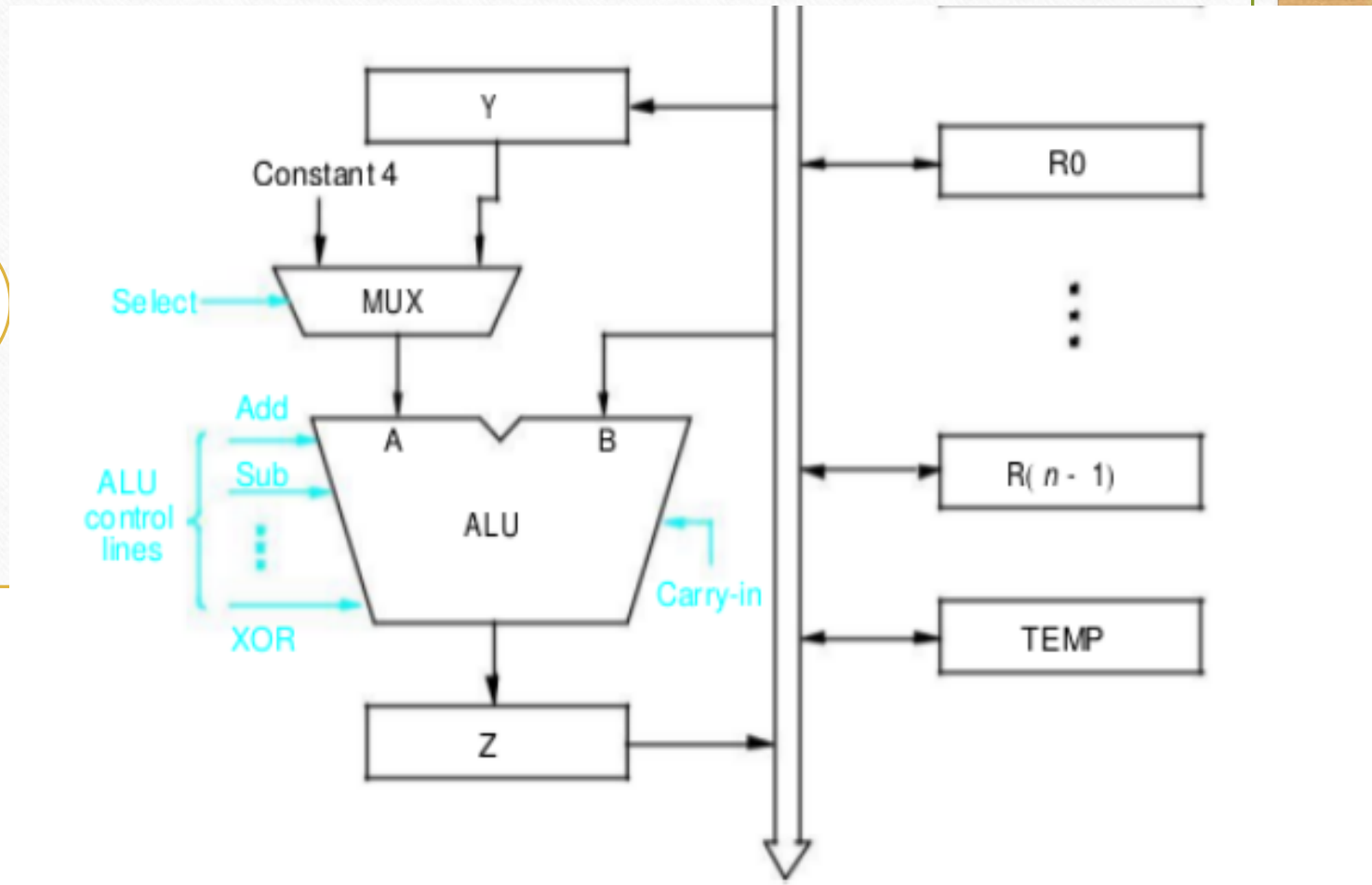
Performing an Arithmetic or Logic Operation

- The ALU is a combinational circuit that has no internal storage.
- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

1. **R1out, Yin**
2. **R2out, SelectY, Add, Zin**
3. **Zout, R3in**

Performing an Arithmetic or Logic Operation

1. R1out, Yin
2. R2out, SelectY, Add, Zin
3. Zout, R3in



- The MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.
- The constant 4 is used to increment the contents of the program counter.

All other signals are inactive.

- ✓ In step 1, the output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.
- ✓ In Step 2, the multiplexer's select signal is set to Select Y, causing the multiplexer to gate the contents of register Y to input A of the ALU.
    - At the same time, the contents of register R2 are gated onto the bus and, hence, to input B.
    - The function performed by the ALU depends on the signals applied to its control lines.
    - In this case, the ADD line is set to 1, causing the output of the ALU to be the sum of the two numbers at inputs A and B.
    - This sum is loaded into register Z because its input control signal is activated
- ✓ In step 3, the contents of register Z are transferred to the destination register R3.

This last transfer cannot be carried out during step 2, because only one register output can be connected to the bus during any clock cycle.

Fetching a Word from Memory(Read)

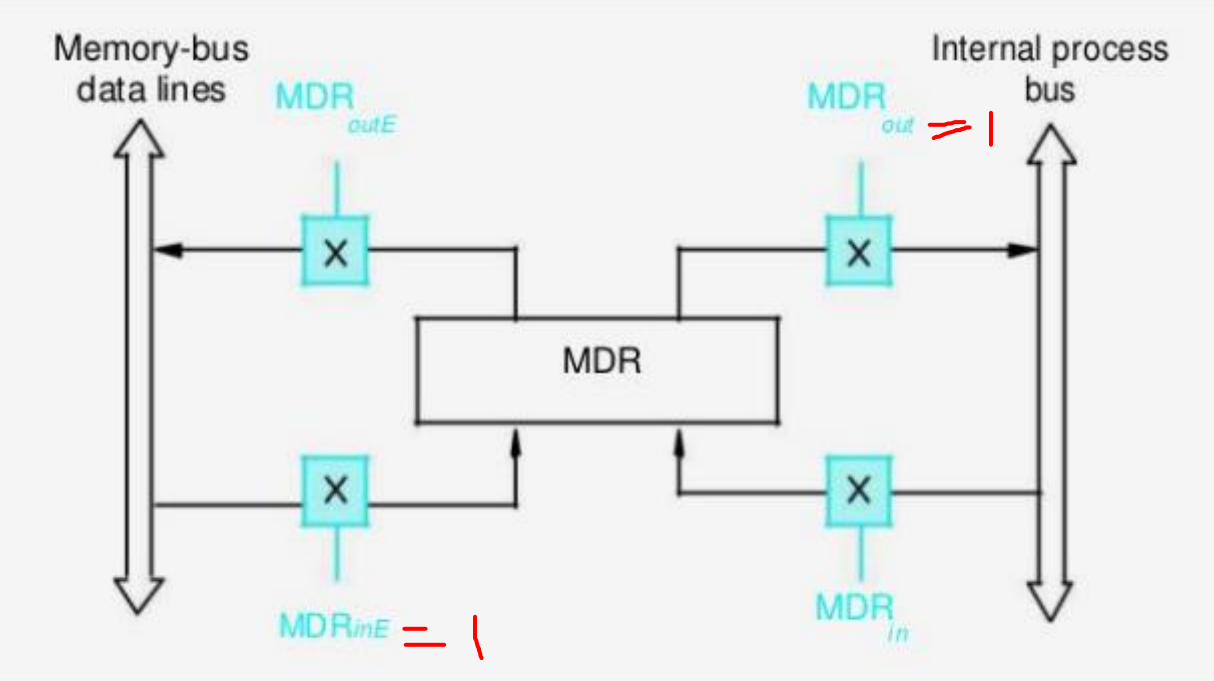


Figure: Connection and control signals MDR.

- The response time of each memory access varies (cache miss, memory-mapped I/O etc.)
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).

- **Step 1: MAR ← [R1]**
- **Step 2: Read Operation Started**
- **Step 3: Wait for MFC**
- **Step 4: Load the data into MDR**
- **Step 5: R2 ← [MDR]**

Example: Move (R1), R2

Example: Move (R1), R2

- The output of MAR is enabled all the time.
- Thus the contents of MAR are always available on the address lines of the memory bus.
- When a new address is loaded into MAR, it will appear on the memory bus at the beginning of the next clock cycle.
- A read control signal is activated at the same time MAR is loaded.

- This means memory read operations requires three steps, which can be described by the signals being activated as follows

R1 out ,MAR in ,Read
MDR inE ,WMFC
MDR out ,R2 in

Storing a word in Memory (Write)

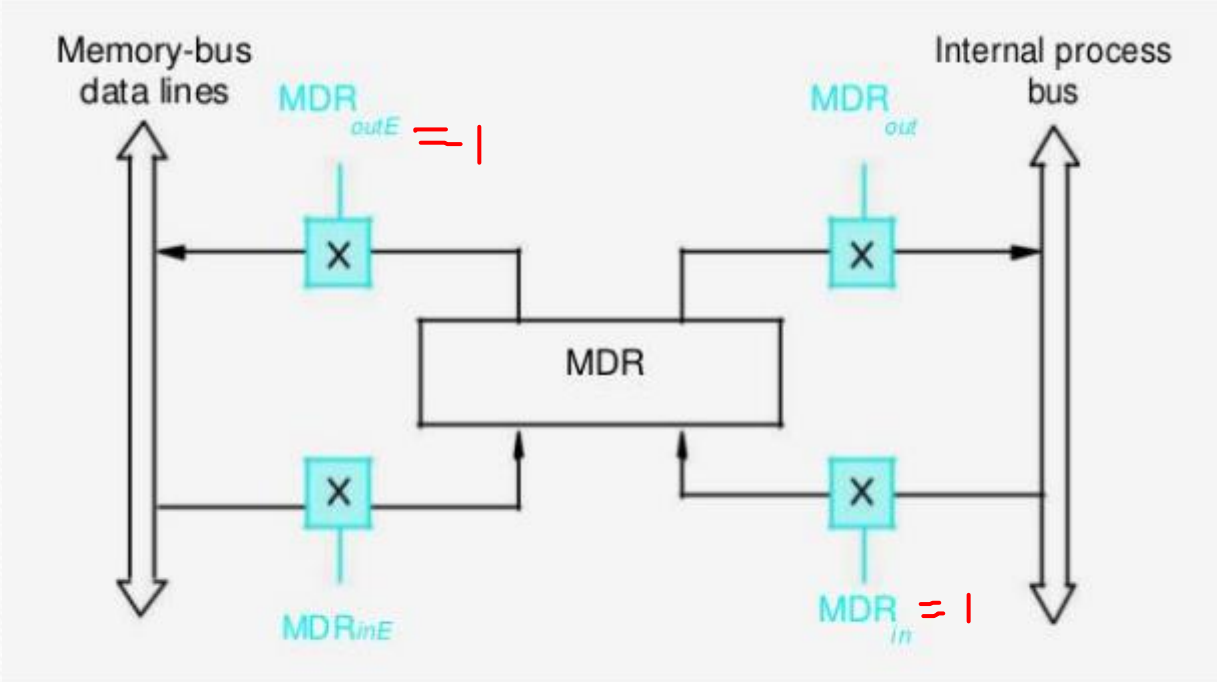


Figure: Connection and control signals MDR.

Writing a word into a memory location follows a similar procedure.

- The desired address is loaded into MAR.
- Then, the data to be written are loaded into MDR, and a write command is issued

Example: Move R2,(R1) requires the following steps

R1 out ,MAR in
R2 out ,MDR in ,Write
MDR outE ,WMFC

Execution of a complete instruction.

To execute an instruction, the processor has to perform the following steps:

- Fetch the contents of the memory location pointed to by the PC. The contents of this location is the instruction to be executed; hence they are loaded into the IR. (Fetch phase)

  $IR \leftarrow [[PC]]$

- Increment the PC to point to the next instruction. Assuming that the memory is byte addressable, the PC is incremented by 4. (Fetch phase)

  $PC \leftarrow [PC] + 4$

- Carry out the operation specified by the instruction in the IR.(Execution phase)

Execution of a complete instruction.

- Consider the instruction Add (R3), R1 This adds the contents of a memory location pointed to by R3 to register R1.
- Executing this instruction requires the following actions

1. **Fetch the instruction**
2. **Fetch the first operand (the contents of the memory location pointed to by R3)**
3. **Perform the addition**
4. **Load the result into R1**

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1.	PCout , MARin , Read, Select4,Add, Zin
2.	Zout , PCin , Yin , WMFC
3.	MDRout , IR in
4.	R3out , MARin , Read
5.	R1out , Yin , WMFC
6.	MDRout , SelectY, Add, Zin
7.	Zout ,R1n ,End

First 3 steps are common for all instruction(fetch instruction and increment pc value

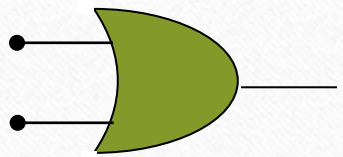
Micro-operations

- The microoperations most often encountered in digital computers are classified into four categories:
  1. Register transfer microoperations
  2. Arithmetic microoperations (on numeric data stored in the registers)
  3. Logic microoperations (bit manipulations on non-numeric data)
  4. Shift microoperations

Logic Microoperations

OR Microoperation

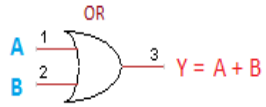
- Symbol: ∨, +
- Gate:



Truth table

Two Input OR gate		
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Symbol



- Example: $100110_2 \vee 1010110_2 = 1110110_2$

OR

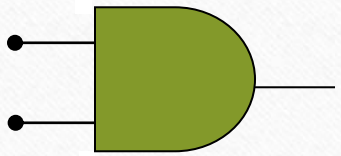
R1←R2+R3, R4←R5 ∨R6

ADD

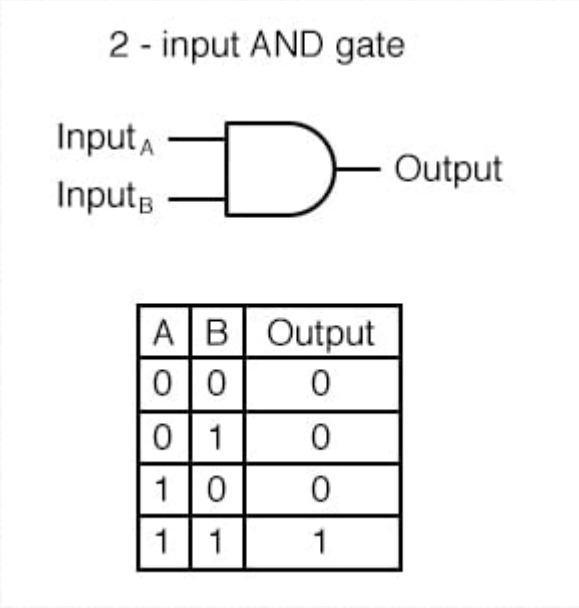
Logic Microoperations

AND Microoperation

- Symbol: $\wedge$
- Gate:



- Example: $100110_2 \wedge 1010110_2 = 0000110_2$

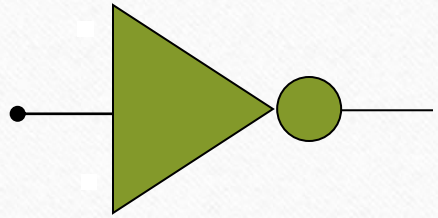


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

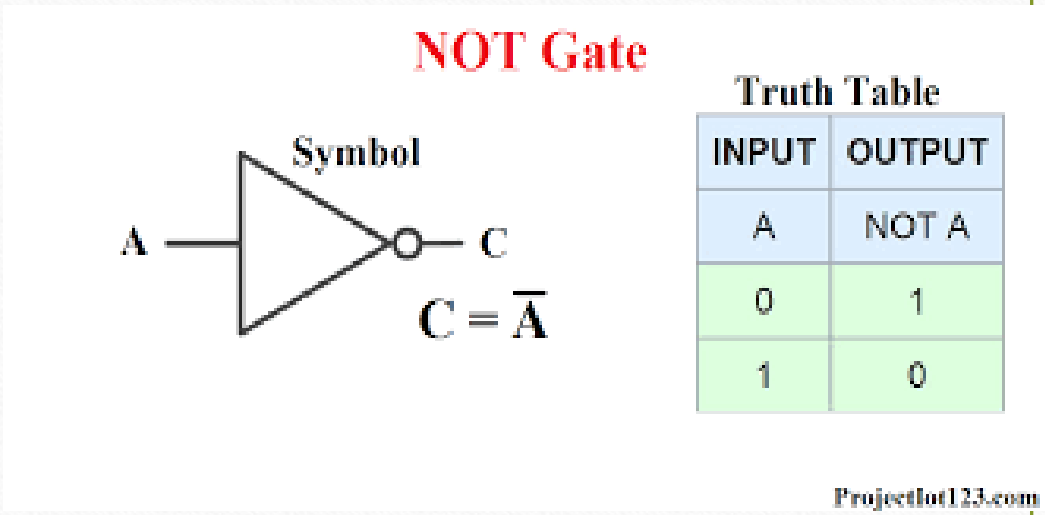
Logic Microoperations

Complement (NOT) Microoperation

- Symbol: $\overline{}$
- Gate:



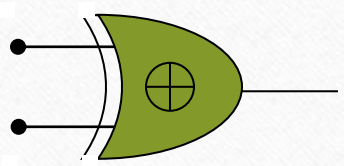
- Example: $1010110_2 = 0101001_2$



Logic Microoperations

XOR (Exclusive-OR) Microoperation

- Symbol: $\oplus$
- Gate:



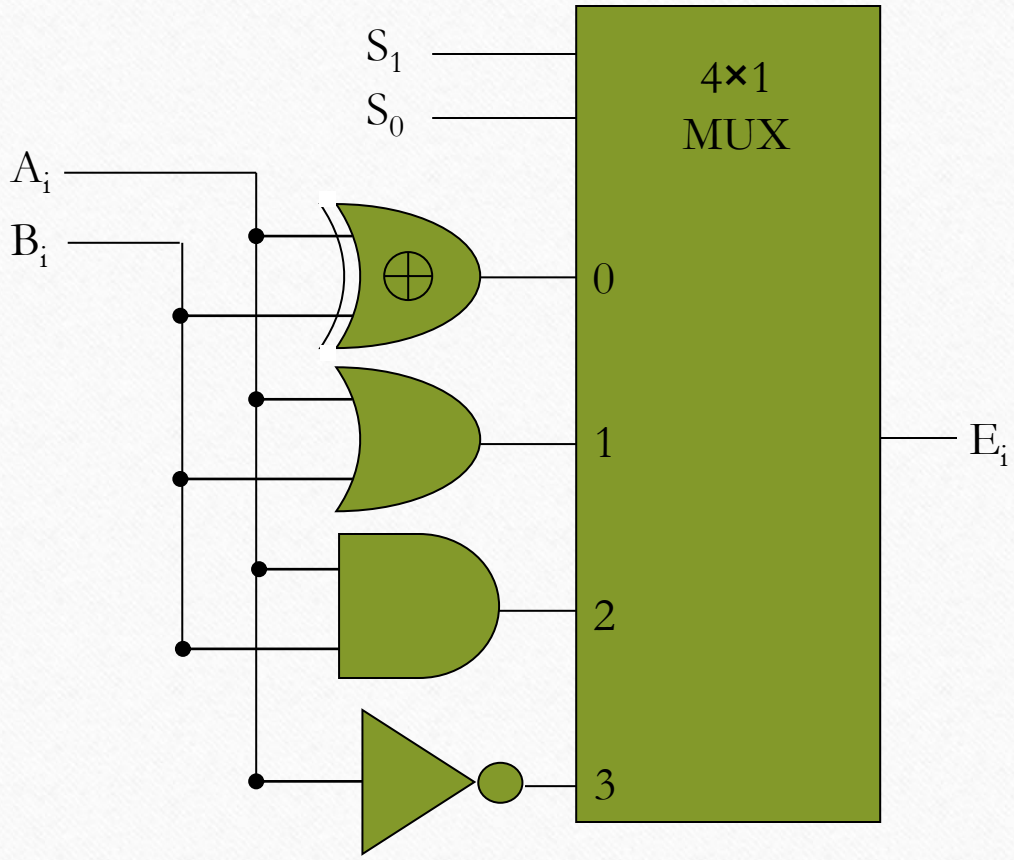
- Example: $100110_2 \oplus 1010110_2 = 1110000_2$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Logic Microoperations Hardware Implementation

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function
- Most computers use only four (AND, OR, XOR, and NOT) from which all others can be derived.

Logic Microoperations
Hardware Implementation cont.



S_1	S_0	Output	Operation
0	0	$E = A \oplus B$	XOR
0	1	$E = A \vee B$	OR
1	0	$E = A \wedge B$	AND
1	1	$E = \overline{A}$	Complement

Logical *Microoperations* (Continued)

- Let R1 = 10101010, and R2 = 11110000
- Then after "Operation", R0 Becomes:

R0	Operation
01010101	**$R0 \leftarrow \overline{R1}$**
11111010	**$R0 \leftarrow R1 \vee R2$**
10100000	**$R0 \leftarrow R1 \wedge R2$**
01011010	**$R0 \leftarrow R1 \oplus R2$**

Shift Micro-Operations

Shift micro-operations are those micro-operations that are used for the serial transfer of information.

These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations

There are three types of shift micro-operations:

1. **Logical Shift**
2. **Arithmetic Shift**
3. **Circular Shift**

1. Logical Shift:

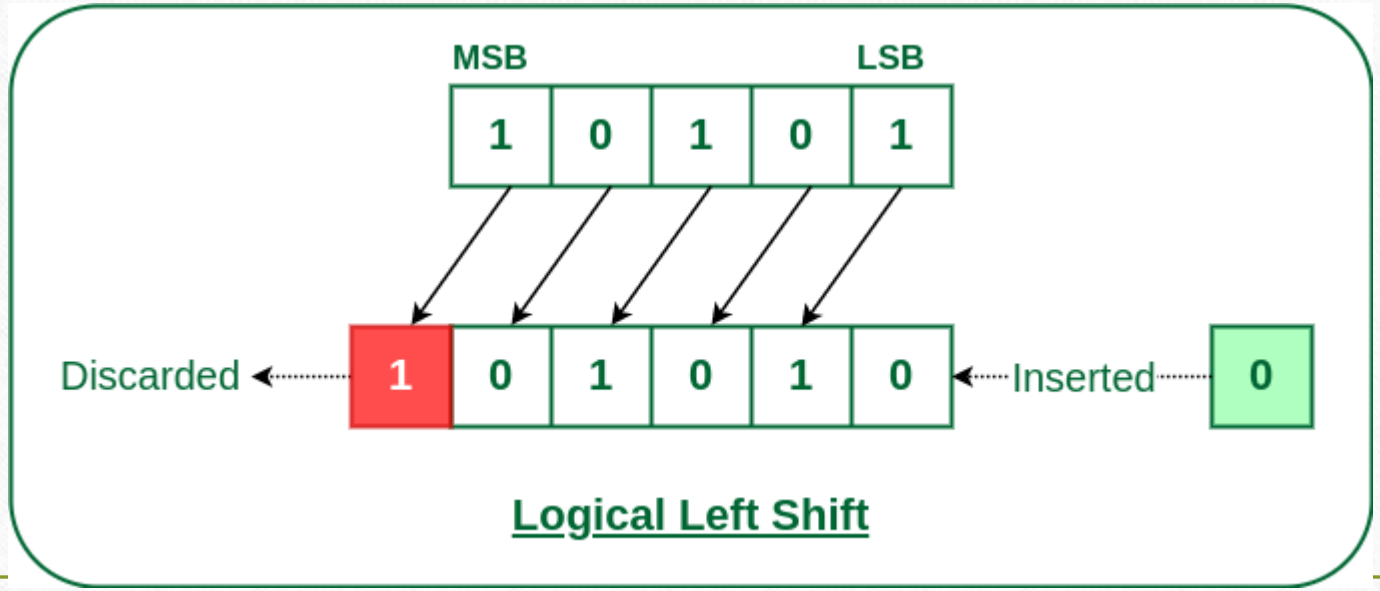
It transfers the 0 zero through the serial input. We use the symbols ‘<<‘ for the logical left shift and ‘>>‘ for the logical right shift. Following are the two ways to perform the arithmetic shift.

1. **Logical Left Shift**
2. **Logical Right Shift**

Logical Left Shift:

In this shift, one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.

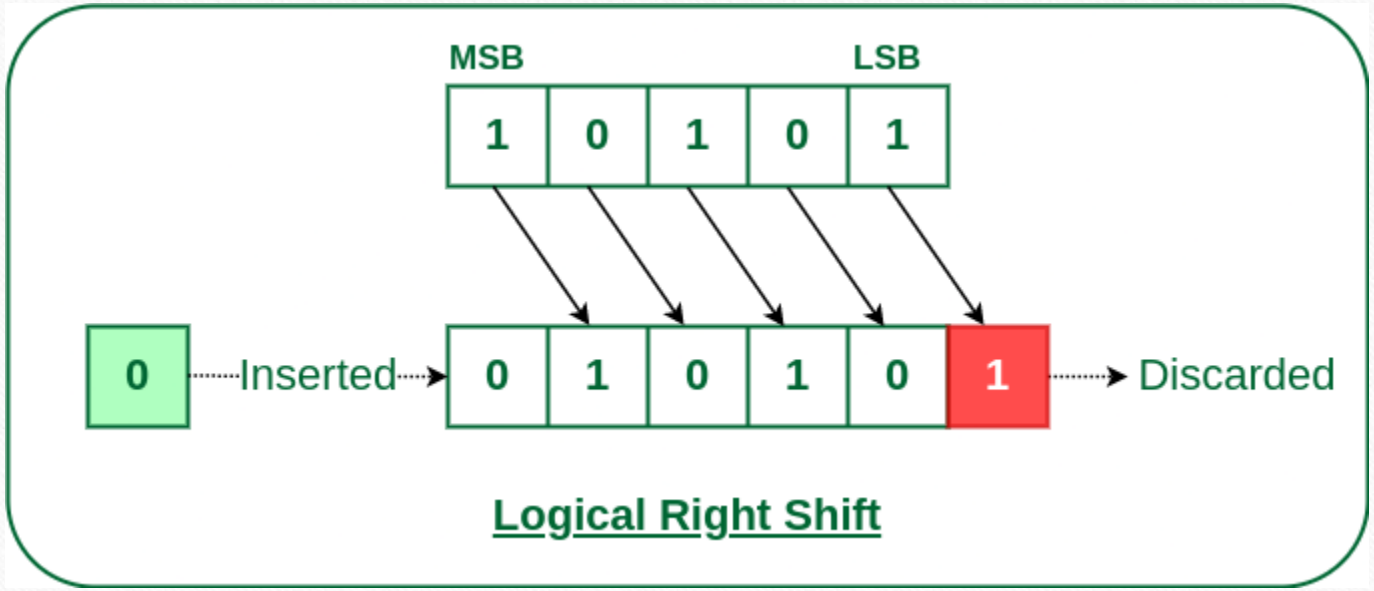
The left shift operator is denoted by the double left arrow key (<<). The general syntax for the left shift is shift-expression << k.



Logical Right Shift

In this shift, each bit moves to the right one by one and the least significant bit(LSB) is rejected and the empty MSB is filled with zero.

The right shift operator is denoted by the double right arrow key (>>). The general syntax for the right shift is “shift-expression >> k”.



2. Arithmetic Shift:

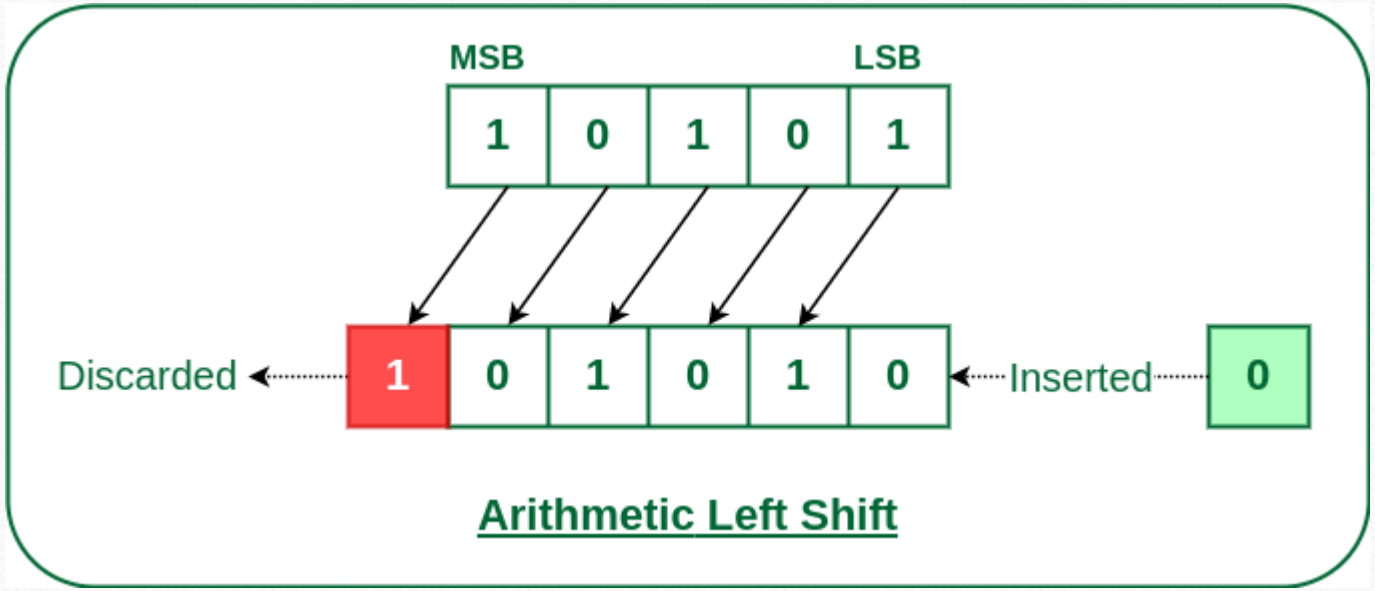
The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position.

Following are the two ways to perform the arithmetic shift.

1. **Arithmetic Left Shift**
2. **Arithmetic Right Shift**

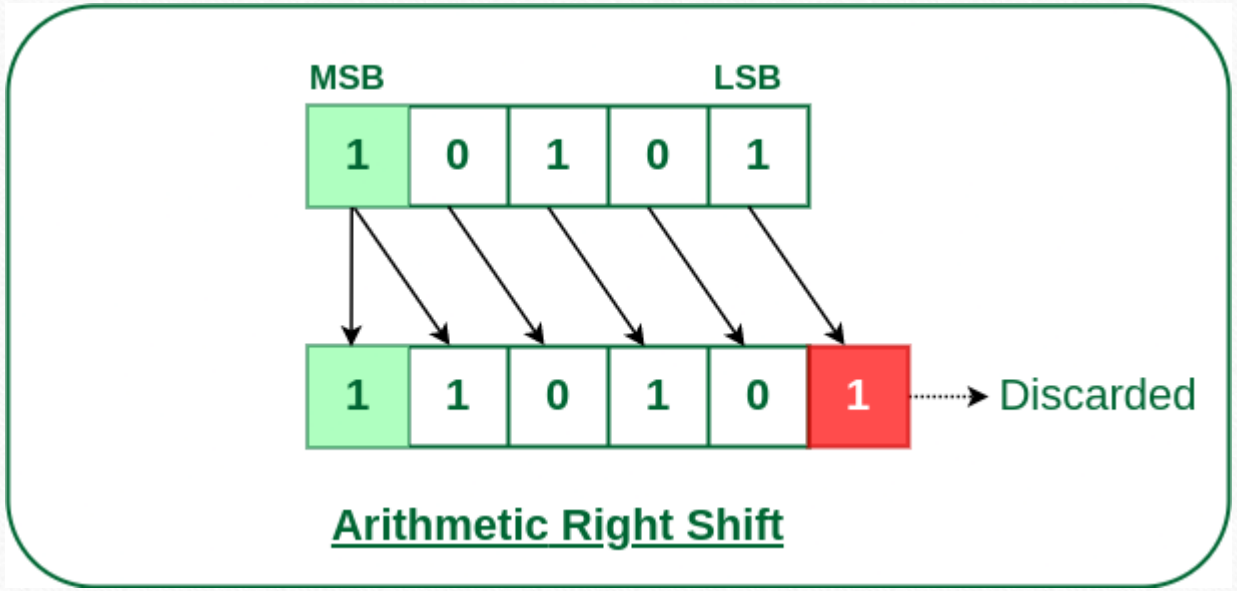
Arithmetic Left Shift:

In this shift, each bit is moved to the left one by one. The empty least significant bit (LSB) is filled with zero and the most significant bit (MSB) is rejected. Same as the Left Logical Shift.



Arithmetic Right Shift:

In this shift, each bit is moved to the right one by one and the least significant(LSB) bit is rejected and the empty most significant bit(MSB) is filled with the value of the previous MSB.



3. Circular Shift:

The circular shift circulates the bits in the sequence of the register around both ends without any loss of information.

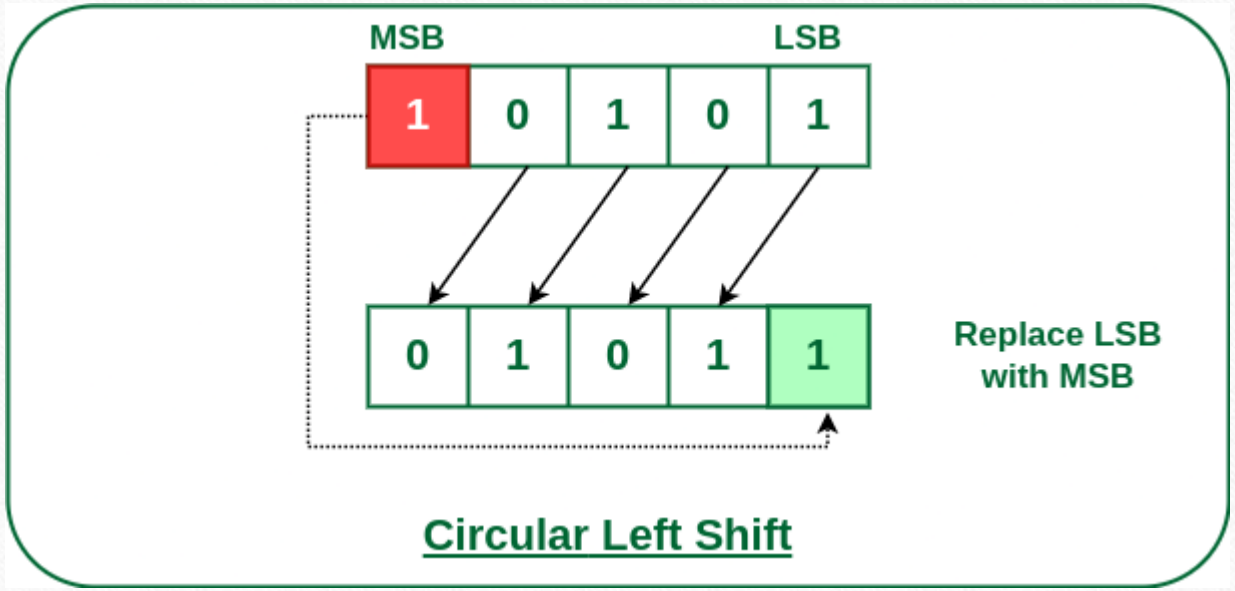
Following are the two ways to perform the circular shift.

1.Circular Shift Left

2.Circular Shift Right

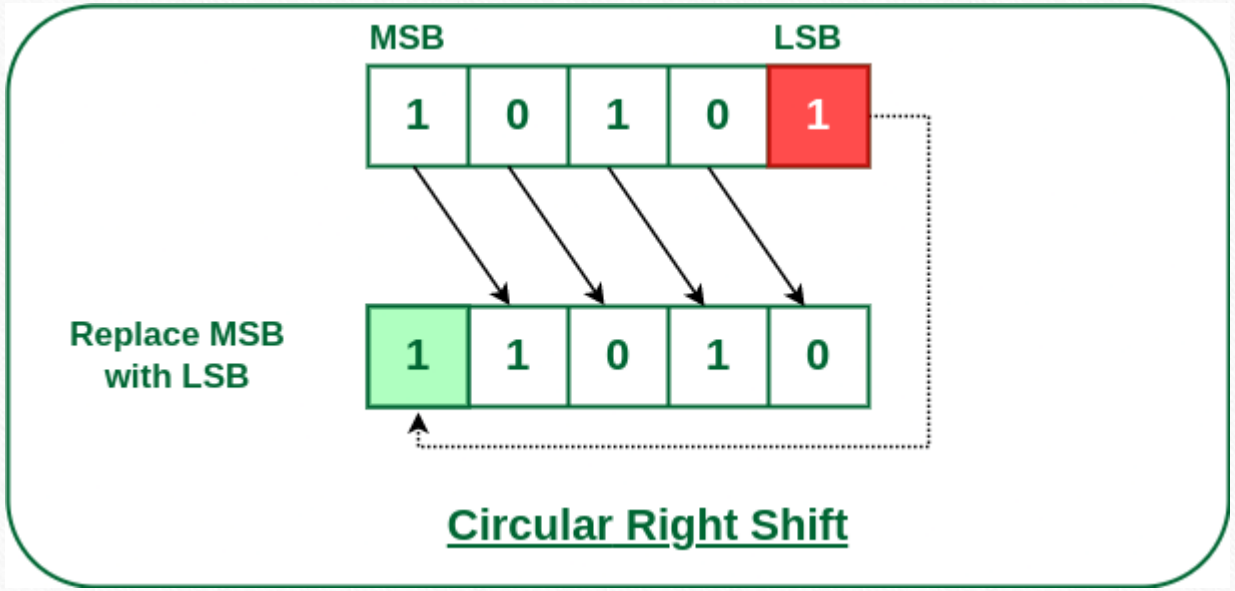
Circular Left Shift:

In this micro shift operation each bit in the register is shifted to the left one by one. After shifting, the LSB becomes empty, so the value of the MSB is filled in there.



Circular Right Shift:

In this micro shift operation each bit in the register is shifted to the right one by one. After shifting, the MSB becomes empty, so the value of the LSB is filled in there.



Arithmetic Micro-operations

In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

The basic Arithmetic Micro-operations are classified in the following categories:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Shift

Some additional Arithmetic Micro-operations are classified as:

1. Add with carry
2. Subtract with borrow
3. Transfer/Load, etc.

The following table shows the symbolic representation of various Arithmetic Micro-operations.

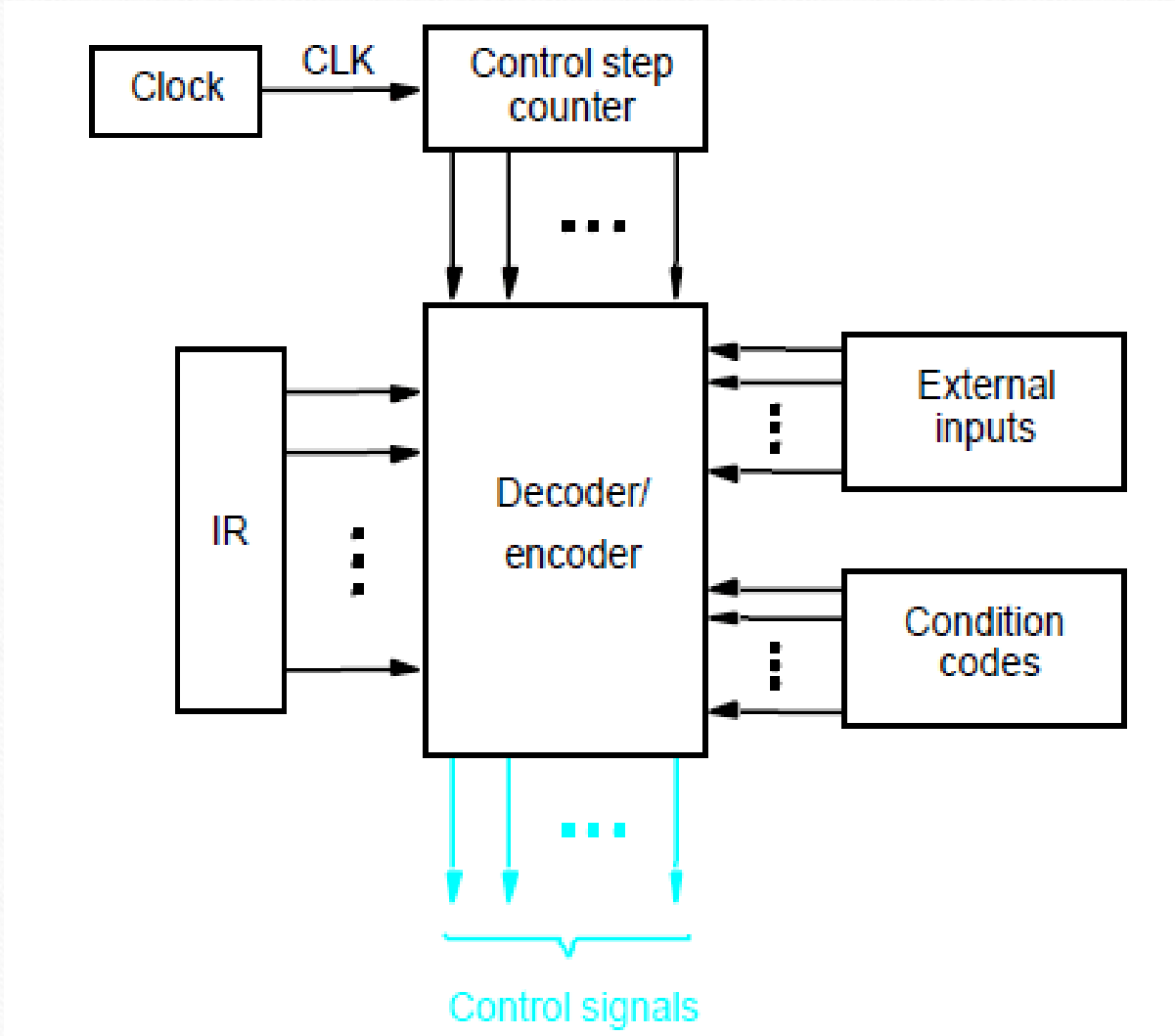
Symbolic Representation	Description
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

Control Unit

- To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence.
- There are two approaches used for generating the control signals in proper sequence

1. Hardwired Control Unit
2. Micro-programmed Control Unit.

Hardwired Control Unit



Hardwired Control Unit

- **The decoder/encoder** is a combinational circuit that generates a set of required control signals.
- **A control step counter** is used to keep track of the control steps.
- Each count of this counter corresponds to one control step.
- The required **control signals** are determined by the following information:
  1. contents of the control step counter
  2. contents of IR register
  3. contents of the condition code flags
  4. External input signals, like MFC and interrupt request.

Detailed Block Description

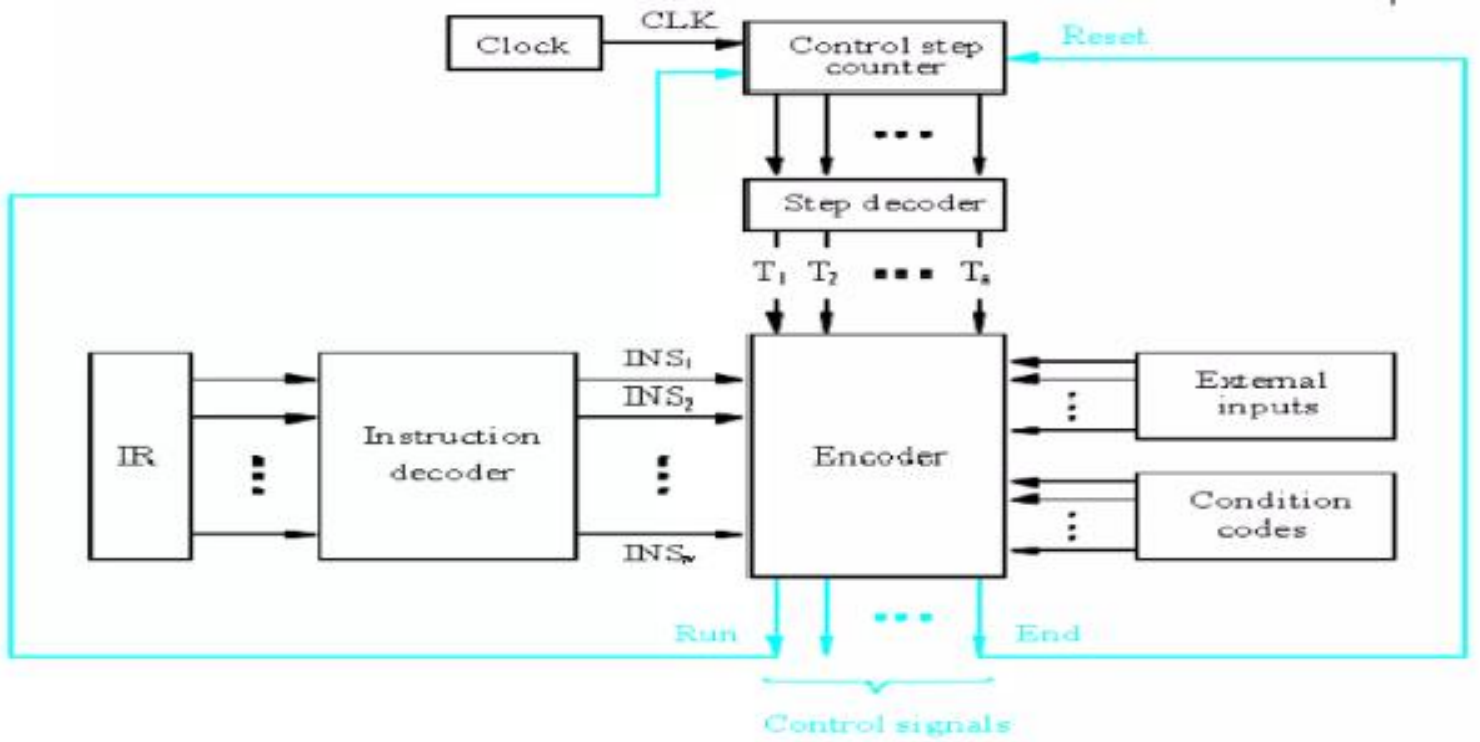


Figure 7.11. Separation of the decoding and encoding functions.

Detailed Block Description

- The step decoder generate a separate clock signal for each step, or time slot, in the control sequence.
- The instruction decoder decodes the instruction loaded in IR.
- The output of the instruction decoder consists of a separate line for each of the 'm' machine instruction.
- According to the code in the IR, only one line amongst all output lines of decoder is set to 1 and all other lines are set to 0.
- The input signals to encoder are combined to generate the individual control signals like add, read, etc.
- The End signal starts the a new instruction fetch cycle by resetting the control step counter to its starting value.
- When run=1, the counter to be incremented by one at the end of every clock cycle.
- When run=0, the counter stops counting and this is needed whenever the WMFC signal is activated.

Hardwired Control Unit

- Advantages of Hardwired Control Unit:
  1. Fast because control signals are generated by combinational circuits.
  2. The delay in generation of control signals depends upon the number of gates.

- Disadvantages of Hardwired Control Unit:
  1. More the control signals required by CPU, more complex will be the design of control unit.
  2. No Flexibility. Modification in control signal is very difficult i.e. it requires rearranging of wires in the hardware circuit.
  3. Difficult to add new feature in existing design of control unit.

Microprogrammed Control Unit

- Control signals are generated by **a program** similar to machine language programs.
- Sequence of control steps required to perform

  ADD (R3),R1 operations are

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	

Figure 7.15 An example of microinstructions

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1.	PCout , MARin , Read, Select4,Add, Zin
2.	Zout , PCin , Yin , WMFC
3.	MDRout , IR in
4.	R3out , MARin , Read
5.	R1out , Yin , WMFC
6.	MDRout , SelectY, Add, Zin
7.	Zout ,R1n ,End

Basic organization of Microprogrammed Control Unit

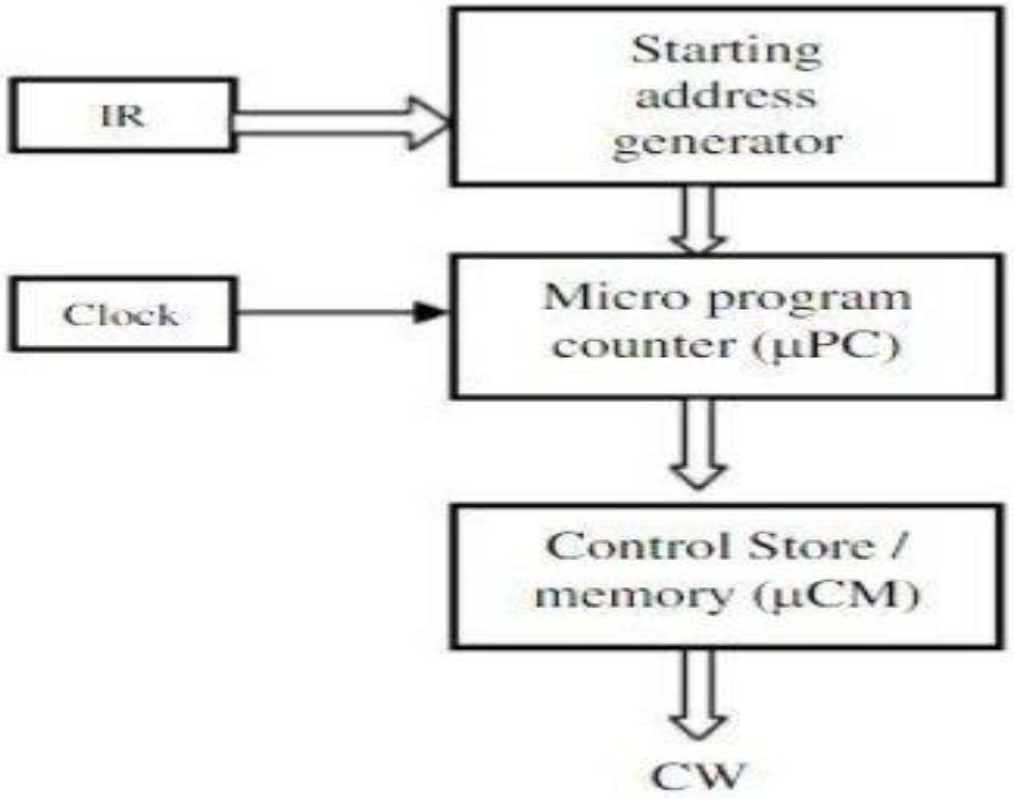


Figure 8

Microprogrammed Control Unit

- Control Word(CW):

  A word where individual bits represents the various control signals.

- Microroutine:

  A sequence of CWs corresponding to the control sequence of a machine instruction.

- Microinstruction:

  The individual control word in microroutine.

- Control Store:

  The microroutine for all instructions in instruction set of a computer are stored in a special memory called control store.