COMPUTER ORGANIZATION AND ARCHITECTURE (BCS302)

UNIT-3: Control UNIT

ONESHOT

Today's Target

- ALL 7 MARKS AND 2MARKS COVERED
- ALL IMPORTANT QUESTION COVERED
- ALL AKTU PYQs' COVERED

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

INSTRUCTION CYCLE(AKTU 2019-20/2018-19/2022-23)

➢In computer architecture, the instruction cycle, also known as the fetch-decode-execute cycle, is a fundamental concept that describes how a computer processes a machine language instruction. The cycle consists of several steps that a central processing unit (CPU) goes through to execute a single instruction.

➢The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are.-

➢Fetch: In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). The PC is then incremented to point to the next instruction in memory.

➢Decode: In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.

Execute: In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program.

INSTRUCTION(2014-15/2016-17) (2 M)

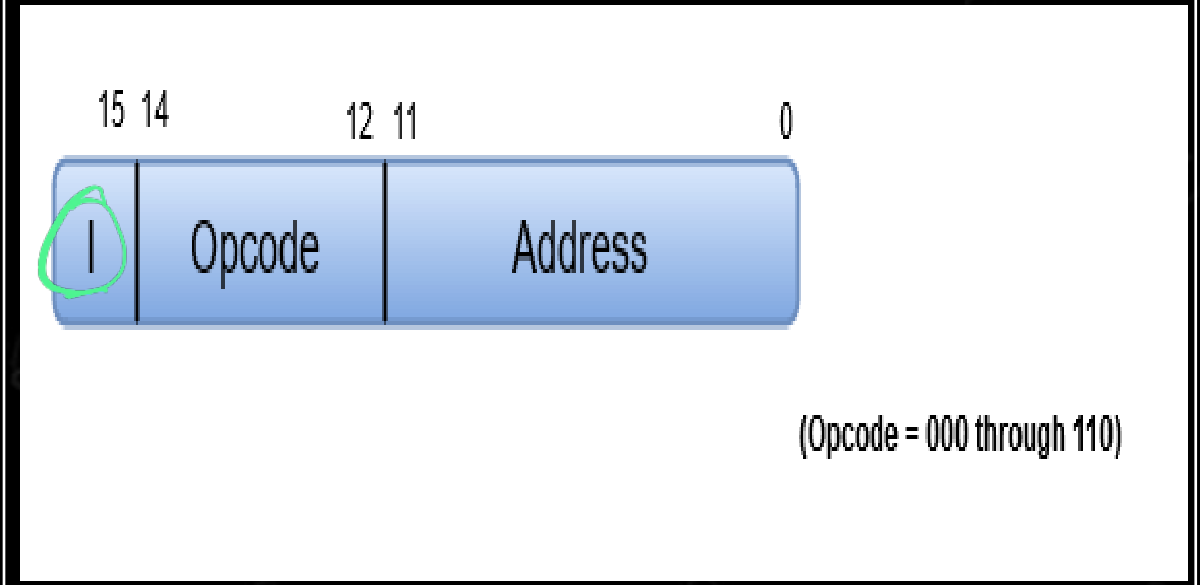
- Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.
- An instruction comprises of groups called fields. These fields include:
- The Operation code (Opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e., register or memory location.
- The Mode field which specifies how the operand will be located.

A basic computer has three instruction code formats which are:

- Memory - reference instruction
- Register - reference instruction
- Input-Output instruction

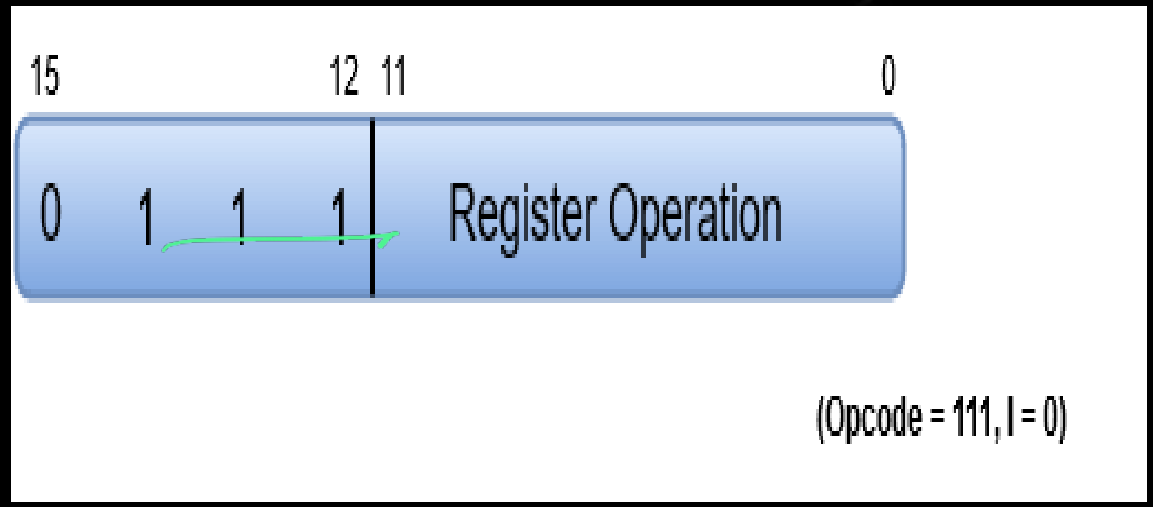
INSTRUCTION

Memory - reference instruction (16 bit)



Register - reference instruction

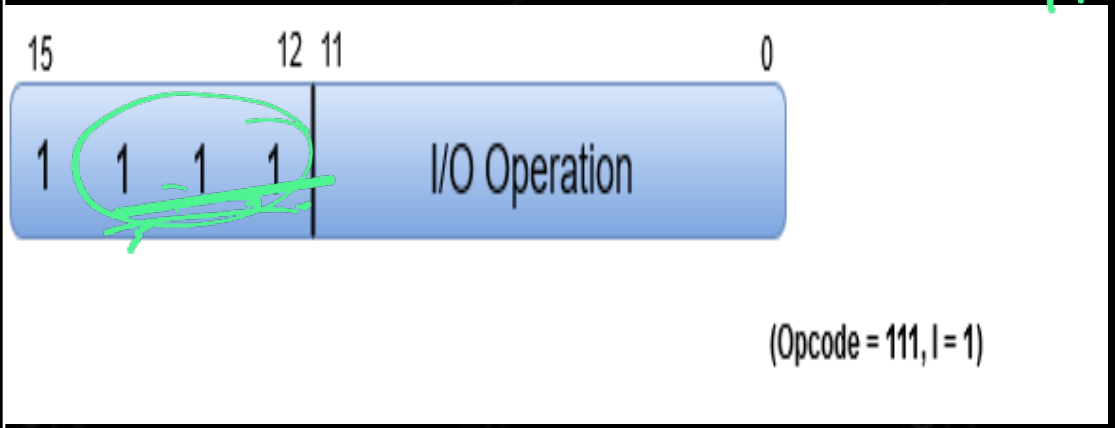
- The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction. A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register



INSTRUCTION

Input-Output instruction

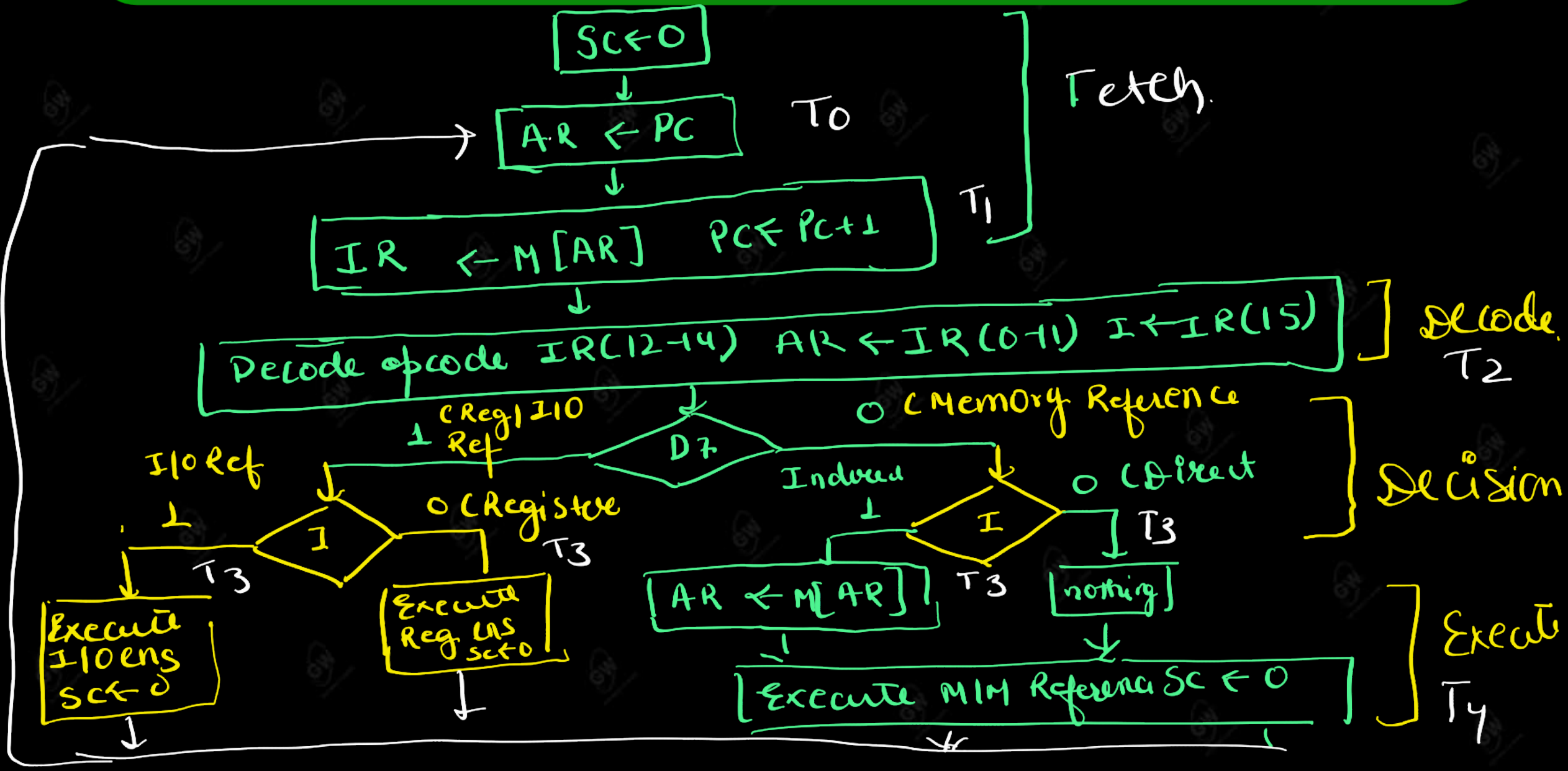
Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed



NOTE

- The Operation code (Opcode) of an instruction refers to a group of bits that define arithmetic and logic operations such as add, subtract, multiply, shift, and compliment.
- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type, and the bit in position 15 is taken as the addressing mode I.
- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

FLOWCHART OF INSTRUCTION CYCLE(AKTU 2021-22/2022-23)



Micro-operation(AKTU 2016-17/2017-18/2105-16)

➢Micro operations are basic, elementary operations that a computer's central processing unit (CPU) performs on data stored in its registers. These operations involve simple arithmetic, logical, or data transfer tasks that contribute to more intricate computations and instructions

➢**Significance And Role:**

➢Micro operations serve as the fundamental steps that processors take to perform a wide range of tasks. They are essential for

➢**Instruction Execution:** Micro operations are the underlying actions that allow the CPU to process and execute instructions provided by software programs.

➢**Data Manipulation:** These operations enable data manipulation within registers, making it possible to perform arithmetic calculations, logical comparisons, and other essential operations.

➢**Control Flow:** Micro operations help control the flow of instructions and data within the CPU, ensuring proper sequencing and synchronization

Micro-operation

Types Of Micro Operations:

Micro operations are categorized into various types based on their functionalities:

Arithmetic Micro Operations: These operations involve mathematical calculations such as addition, subtraction, multiplication, and division.

Logical Micro Operations: Logical micro operations include bitwise operations like AND, OR, NOT, and XOR, which are used for comparing and manipulating binary data.

Shift Micro Operations: Shift operations involve moving bits within a register left or right, effectively multiplying or dividing by powers of two.

Transfer Micro Operations: Transfer operations move data between different registers or memory locations within the CPU.

Arithmetic Micro-operation

Arithmetic micro-operations perform operations on the numeric data stored in the registers.

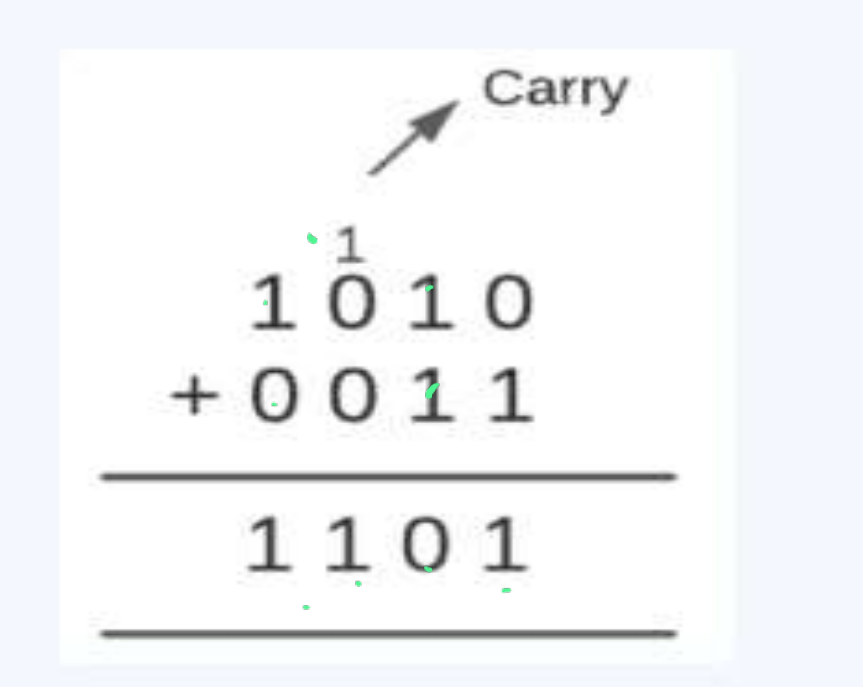
The basic arithmetic micro-operations are-

- Addition
- Subtraction
- Increment
- Decrement
- 1's complement
- 2's complement

Addition

- The Add arithmetic micro-operation adds the values of the two registers and stores the output in the desired register.
- The symbolic notation for the Add arithmetic micro-operation is-
- R3 <- R1 + R2
- the add arithmetic micro-operation remember the following rules:
- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 and 1 as carry (here, 0 is placed in the result and 1 is transferred as carry to the next column)

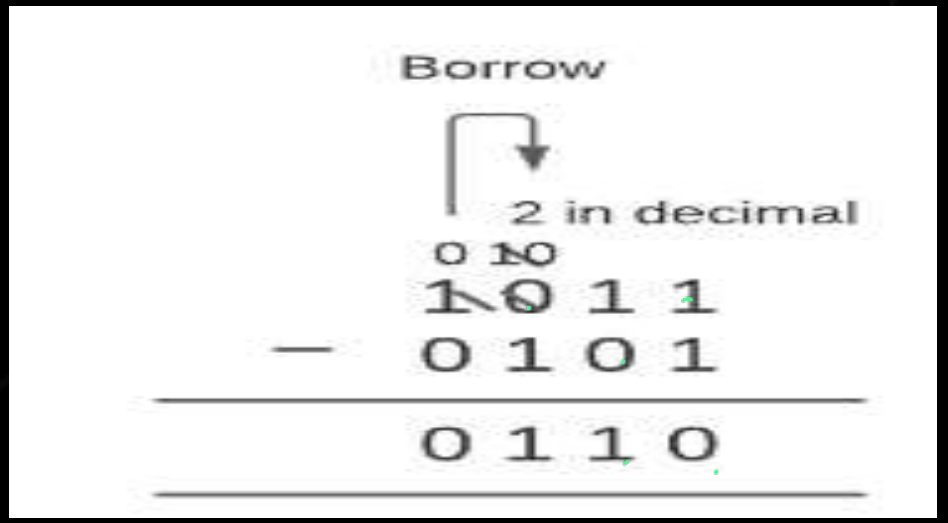
Arithmetic Micro-operation



SUBTRACTION

- The Subtract arithmetic micro-operation subtracts the values of the two registers and stores the output in the desired register.
- The symbolic notation for the Add arithmetic micro-operation is-
- R3 <- R1 - R2
- . For performing the subtract arithmetic micro-operation remember the following rules:
- 0 - 0 = 0
- 0 - 1 = 1 (because 10 is borrowed from next high order digit which is equal to 2 in decimal so 2 - 1 = 1)
- 1 - 0 = 1,1 - 1 = 0

Arithmetic Micro-operation



- Besides the above way, there is also an alternate way of doing the arithmetic subtraction. This way includes the use of the 2's complement.
- To subtract the values of two registers, we need to add the first register, the complemented value of the second register and one.
- The symbolic notation is-
- R3 <- R1 + R2' + 1

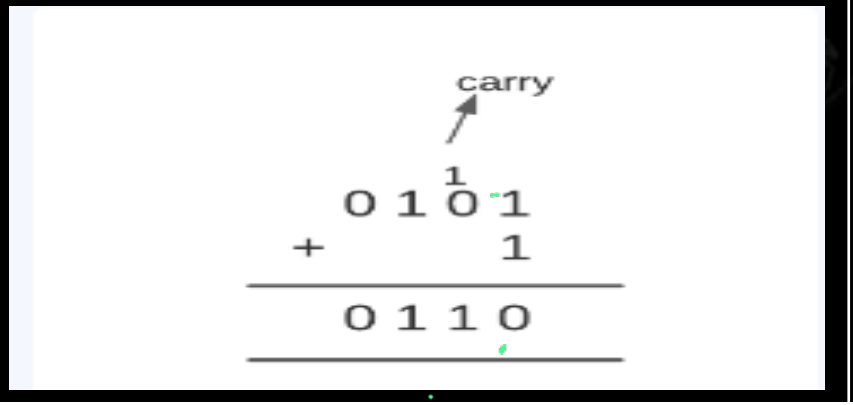
Arithmetic Micro-operation

➢INCREMENT

➢the Increment arithmetic micro-operation increments the value of a register by 1. This means this operation adds 1 to the value of the given register and stores the output in the desired register.

➢The symbolic notation for the Increment arithmetic micro-operation is-

➢R1 <- R1 + 1

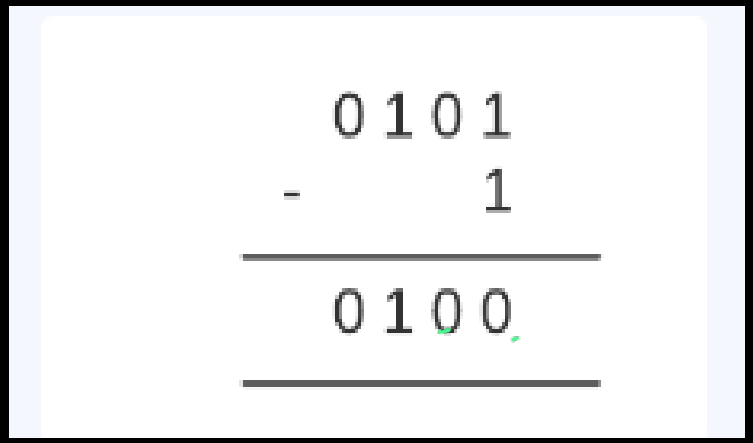


Decrement

The Decrement arithmetic micro-operation decreases the value of a register by 1. This means this operation subtracts one from the value of the given register and stores the output in the desired register.

The symbolic notation for the Increment arithmetic micro-operation is-

R1 <- R1 - 1



Arithmetic Micro-operation

1's Complement

The 1's complement arithmetic micro-operation complements the contents of a register. In this micro-operation, 0 is converted to 1 and 1 is converted to 0.

2's Complement

The 2's complement arithmetic micro-operation first complements the contents of the given register and then adds 1 to it. This micro-operation is also known as Negation.

The symbolic notation for the 2's complement arithmetic micro-operation is-

R2 <- R2' + 1

Arithmetic Micro-operation

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.

REGISTER TRANSFER OPERATION

Register Transfer language

➢The symbolic notation used to describe the micro-operation transfer among registers is called RTL (Register Transfer Language). The use of symbols instead of a narrative explanation provides an organized and concise manner for listing the micro-operation sequences in registers and the control functions that initiate them

➢A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.

➢It is a convenient tool for describing the internal organization of digital computers in concise and precise manner.

➢REGISTER

➢Computer registers are designated by upper case letters (and optionally followed by digits or letters) to denote the function of the register.

➢ For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR.

➢Other designations for registers are PC (for program counter), IR (for instruction register, and R1 (for processor register).

➢The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.

REGISTER TRANSFER OPERATION

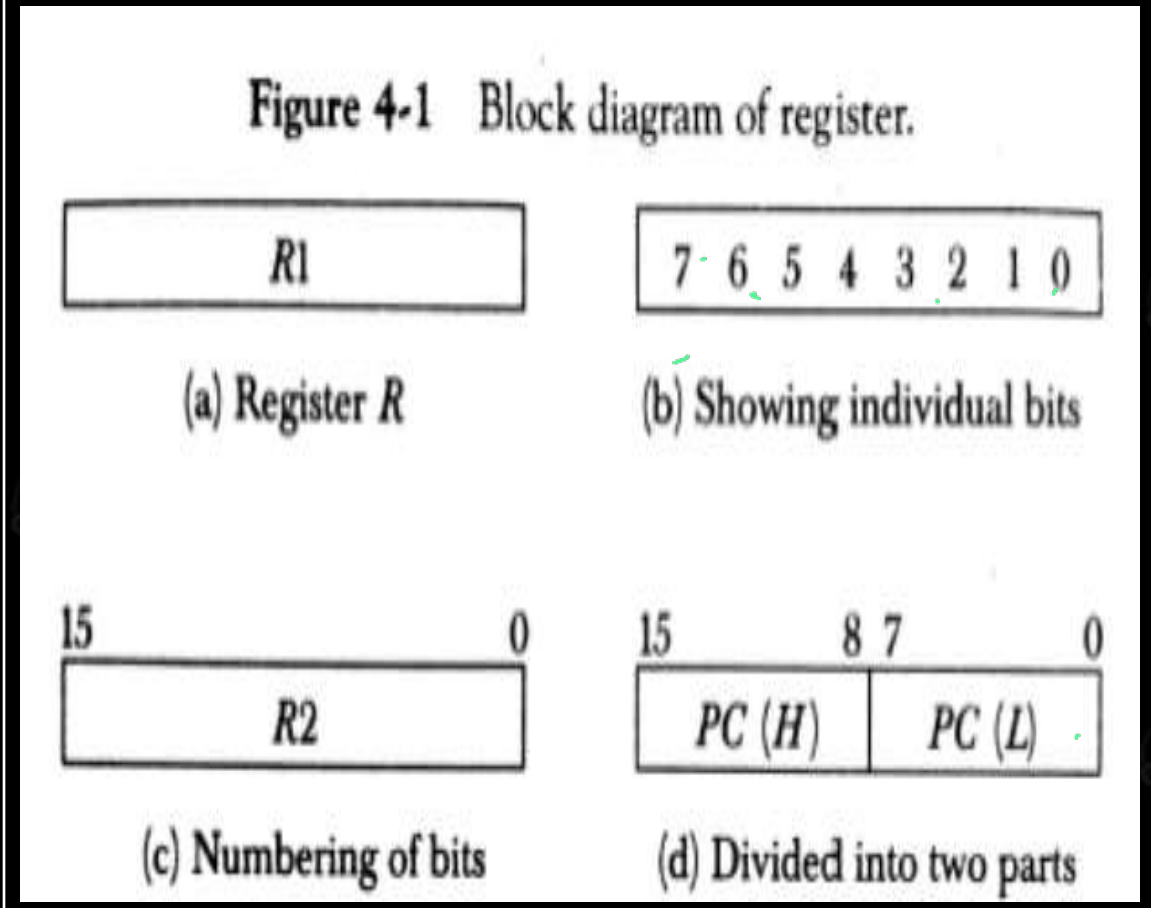


Figure 4-1 Block diagram of register.

➢. Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

➢ The statement R2← R1 denotes a transfer of the content of register R1 into register R2. ⍰ It designates a replacement of the content of R2 by the content of R1.

➢ By definition, the content of the source register R 1 does not change after the transfer.

➢ If we want the transfer to occur only under a predetermined control condition then it can be shown by an if-then statement.

➢if (P=1) then R2← R1

REGISTER TRANSFER OPERATION

- P is the control signal generated by a control section.
- We can separate the control variables from the register transfer operation by specifying a Control Function. Control function is a Boolean variable that is equal to 0 or 1. control function is included in the statement as P: R2← R1
- Control condition is terminated by a colon implies transfer operation be executed by the hardware only if P=1.Every statement written in a register transfer notation implies a hardware construction for implementing the transfer. Figure 4-2 shows the block diagram that depicts the transfer from R1 to R2

- The n outputs of register R1 are connected to the n inputs of register R2.
- The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
- Register R2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
- As shown in the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t.

REGISTER TRANSFER OPERATION

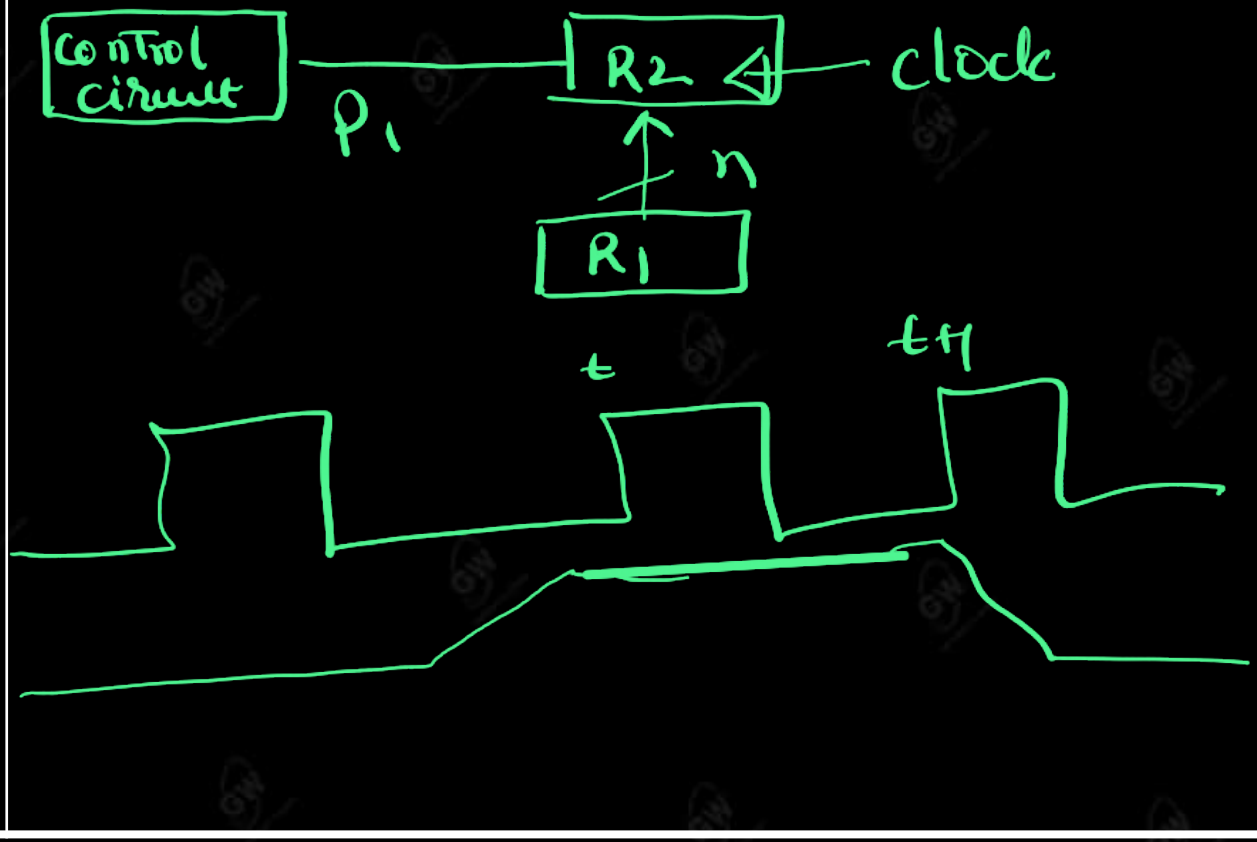
➢The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.

➢P may go back to 0 at time t+1; otherwise, the transfer will occur with every clock pulse transition while P remains active.

➢ Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t +1

➢A comma is used to separate two or more operations that are executed at the same time.

. The statement T : R2← R1, R1← R2 (exchange operation) denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T=1



Register transfer micro-operation

Symbol	Description	Example
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow <--	Denotes transfer of information	R2 <-- R1
Comma ,	Separates two micro operations	R2 <-- R1, R1 <-- R2

shift Micro-operation

Shift micro-operations are used when the data is stored in registers. These micro-operations are used for the serial transmission of data. Here, the data bits are shifted from left to right. Or right to left . These micro-operations are also combined with arithmetic and logic micro-operations and data-processing operations.

There are three types of shift micro-operations-

- Logical Shift
- Arithmetic Shift
- Circular Shift

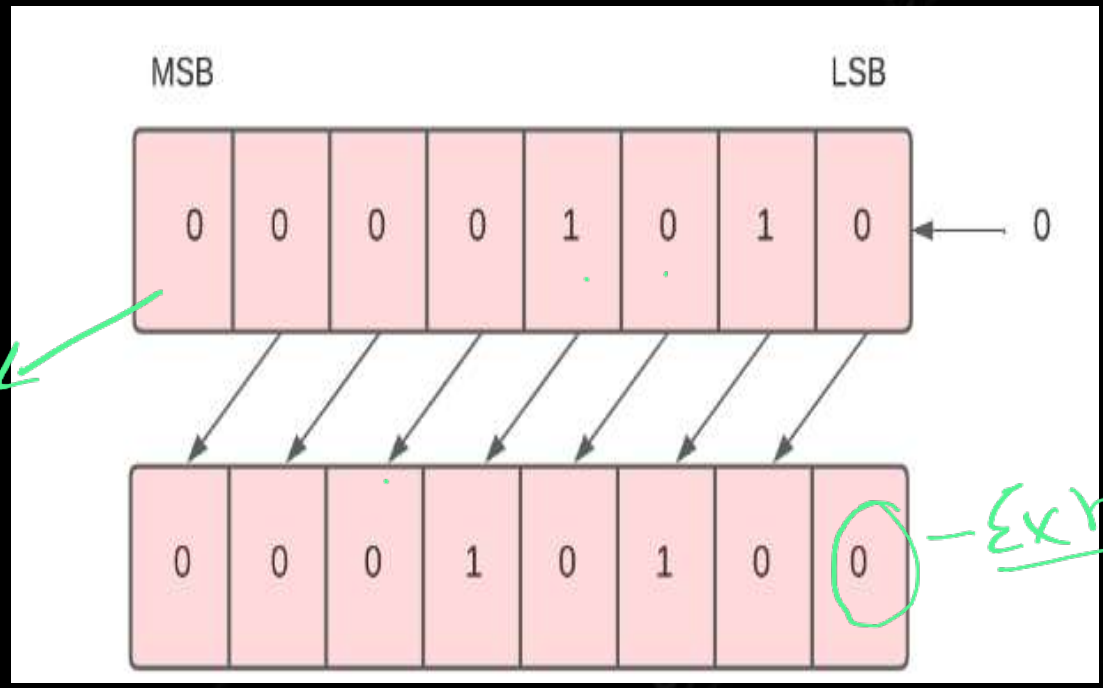
Logical Shift

There are two ways to implement the logical shift.

Logical Shift Left, Logical Shift Right

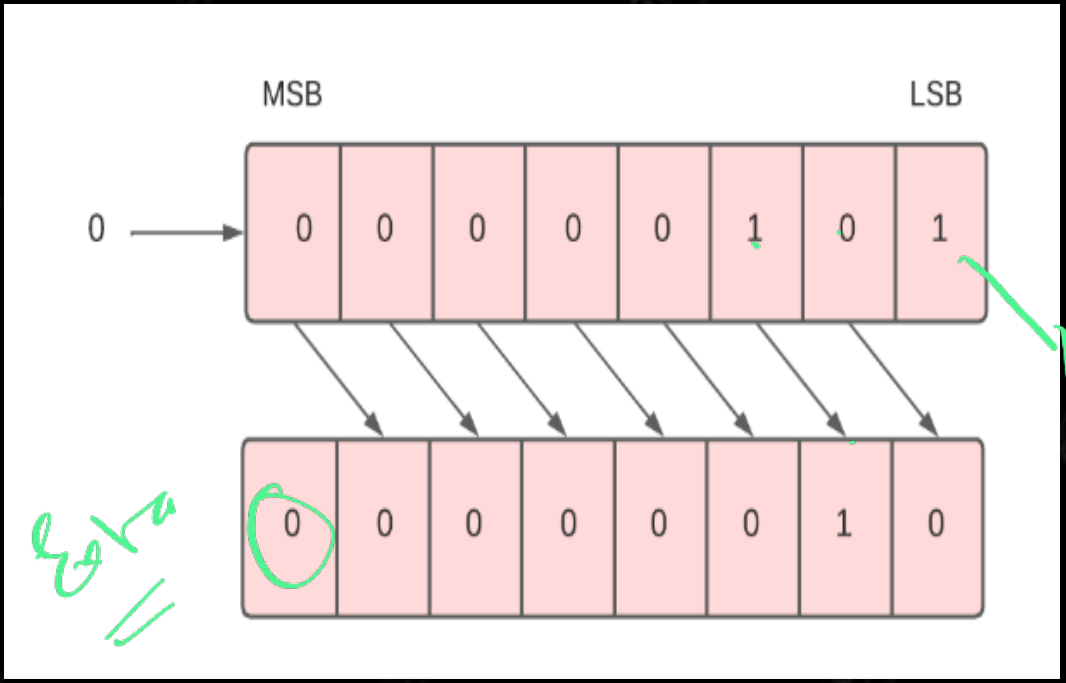
. logical Shift Left

- Each bit in the register is shifted to the left one by one in this shift micro-operation. The most significant bit (MSB) is moved outside the register, and the place of the least significant bit (LSB) is filled with 0.



shift Micro-operation

➢Each bit in the register is shifted to the right one by one in this shift micro-operation. The least significant bit (LSB) is moved outside the register, and the place of the most significant bit (MSB) is filled with 0.



Arithmetic Shift

The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position. There are two ways to implement the arithmetic shift.

➢Arithmetic Shift Left

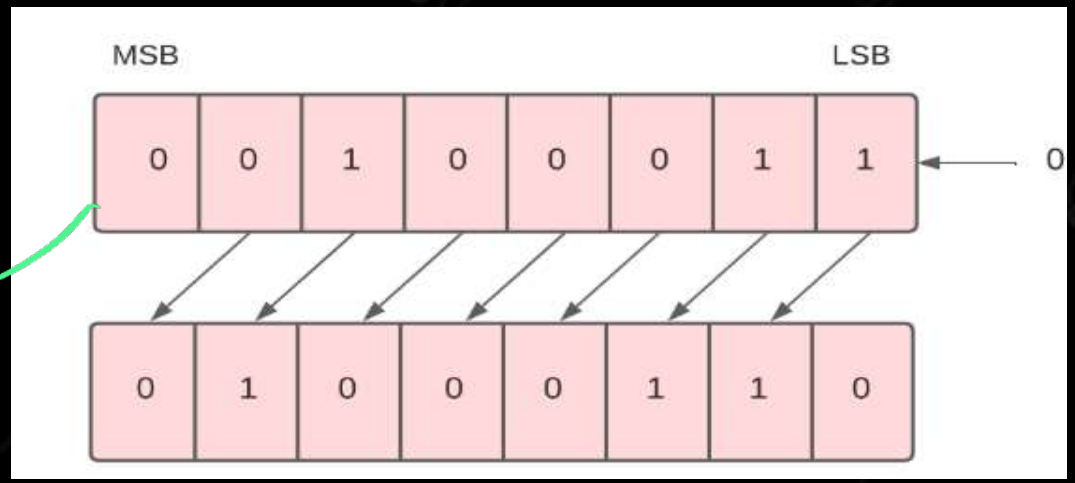
➢Arithmetic Shift Right

shift Micro-operation

➢Arithmetic Shift Left

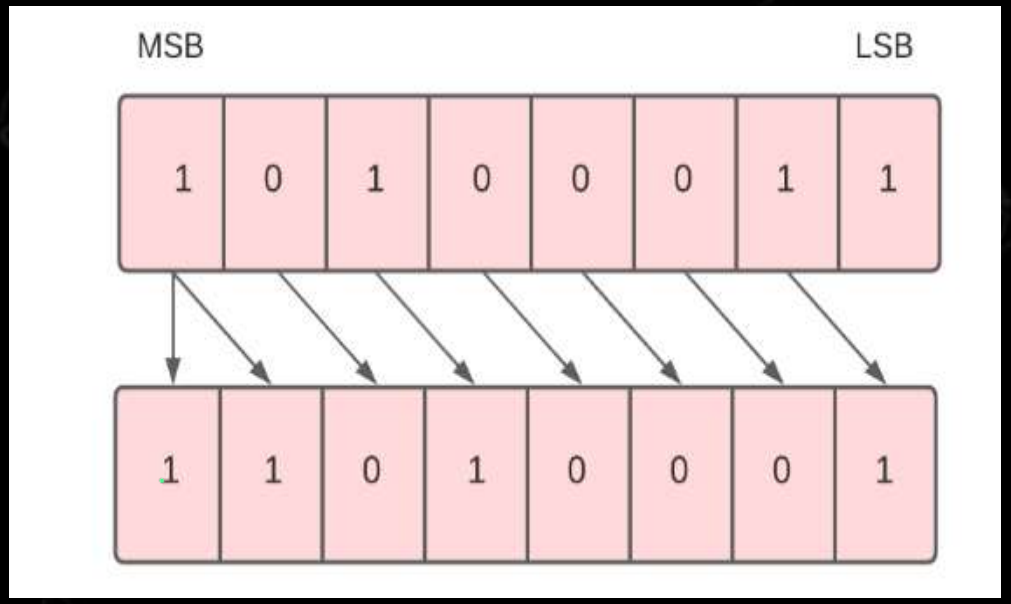
➢The arithmetic shift left micro-operation is the same as the logical shift left micro-operation. Each bit in the register is shifted to the left one by one in this shift micro-operation. The most significant bit (MSB) is moved outside the register, and the place of the least significant bit (LSB) is filled with 0.

➢.



Arithmetic Shift Right

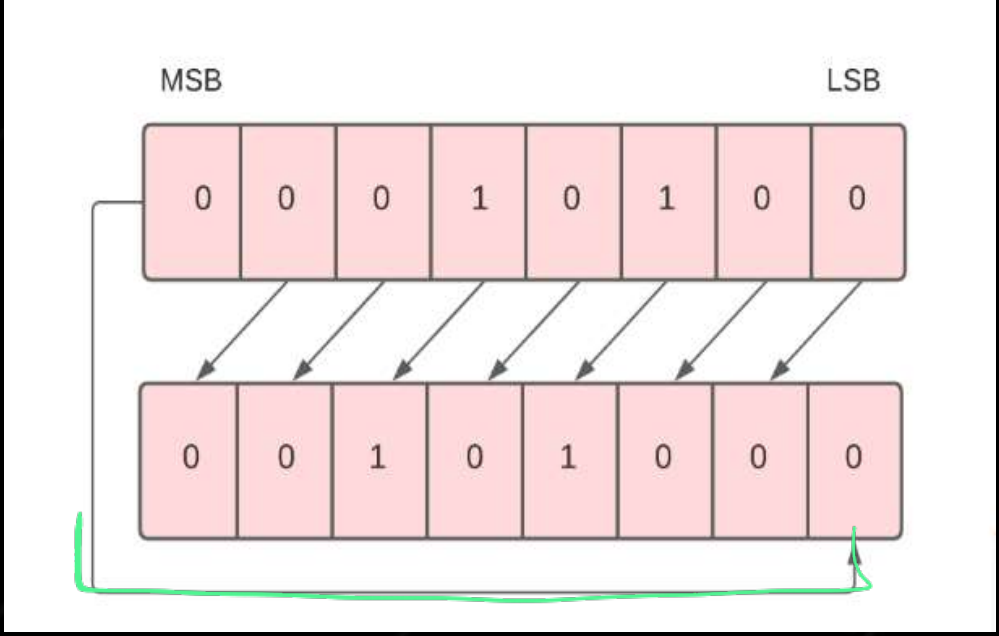
➢Each bit in the register is shifted to the right one by one in this shift micro-operation. The least significant bit (LSB) is moved outside the register, and the place of the most significant bit (MSB) is filled with the previous value of MSB



shift Micro-operation

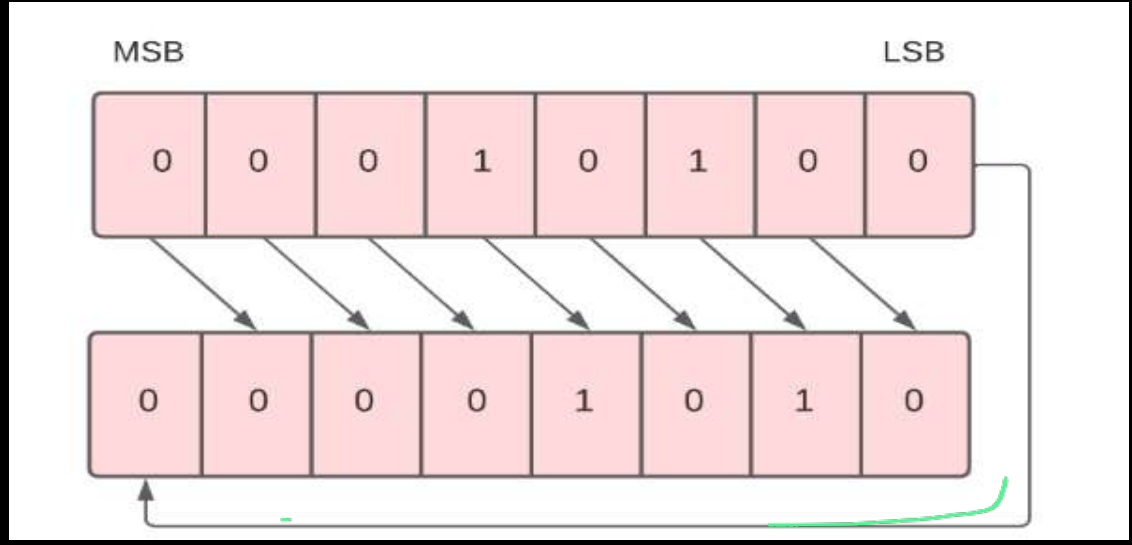
Circular Shift Left

➢Each bit in the register is shifted to the left one by one in this shift micro-operation. After shifting, the least significant bit (LSB) place becomes empty, so it is filled with the value at the most significant bit (MSB).



➢Circular Shift Right

➢Each bit in the register is shifted to the right one by one in this shift micro-operation. After shifting, the most significant bit (MSB) place becomes empty, so it is filled with the value at the least significant bit (LSB).



LOGIC Micro-operation

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = x.y$	$F \leftarrow A \wedge B$	AND
$F_2 = x.y'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'.y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y'$	$F \leftarrow A \oplus B$	Exclusive OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive NOR

LOGIC Micro-operation

$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (x.y)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	F ← all 1's	Set to all 1's

LOGIC Micro-operation

AND

X	Y	X’	Y’	F1=X.Y	F4=X’.Y	F2=X.Y’
0	0	1	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	0	0	1
1	1	0	0	1	0	0

LOGIC Micro-operation

TRANSFER A

X	F3=X
0	0
0	0
1	1
1	1

LOGIC Micro-operation

TRANSFER B

Y	F5=Y
0	0
1	1
0	0
1	1

LOGIC Micro-operation

Exclusive OR

X	Y	F6
0	0	0
0	1	1
1	0	1
1	1	0

LOGIC Micro-operation

OR

X	Y	X’	Y’	F7=X+Y	F11=X+Y’	F13=X’+Y
0	0	1	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	1	1	0
1	1	0	0	1	1	1

LOGIC Micro-operation

NOR

X	Y	X+Y	F8=(X+Y)’
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

LOGIC Micro-operation

Exclusive NOR

X	Y	F9
0	0	1
0	1	0
1	0	0
1	1	1

LOGIC Micro-operation

Complement B

Y	F10=y’
0	1
1	0
0	0
1	1

LOGIC Micro-operation

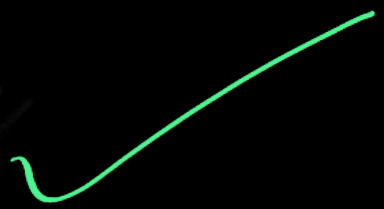
Complement B

Y	F10=y’
0	1
1	0
0	0
1	1

LOGIC Micro-operation

- Complement B

X	F12=X’
0	1
0	1
1	0
1	0



LOGIC Micro-operation

➢NAND

X	Y	F14=(x.y)’
0	0	1
0	1	1
1	0	1
1	1	0

(RISC AND CISC) AKTU2015-16/ 2016-17/2017-18/2019-20/ 2020-21/2021-22/ 2022-23)

- RISC stands for Reduced Instruction Set Computer Processor, and CISC stands for Complex Instruction Set Computer Processor, both are used to increase CPU performance.
- RISC reduces the cycles per instruction at the cost of the number of instructions per program.
- CISC tries to minimize the number of instructions per program but at the cost of increasing the number of cycles per instruction.

2 Marks

Reduced Instruction Set Computer (RISC)

- RISC is a microprocessor architecture that uses a simple set of instructions that can be substantially modified.
- It is designed to reduce the time it takes for instructions to execute by optimizing and reducing the complexity of instructions. It means that each instruction cycle has only one clock per cycle, and each cycle consists of three parameters: fetch, decode, and execute.
- The RISC processor can break multiple complex instructions into a simple one. RISC chips require several transistors, making them less expensive to develop and reducing instruction execution time.
- RISC instruction has a fixed length encoding of instruction and each instruction executes in a single clock
- RISC architecture focus on reducing the complexity of instructions and working with simpler instruction set having limited number of addressing modes and allowing them to execute more instruction in the same amount of time
- Program written for RISC architecture tend make more space in memory but RISC processor increased clock rate allow it to execute its program in less time than a CISC processor take to execute its program

Characteristics/Features of RISC Processor

1.) RISC processors use one clock per cycle (CPI) to execute each instruction in a computer. Each CPI also comprises the methods for fetching, decoding, and executing computer instructions.

2.) Multiple registers in RISC processors allow them to hold instructions, reply fast to the computer, and interact with computer memory as little as possible.

3.) The RISC processors use the pipelining technique to execute multiple parts or stages of instructions to perform more efficiently.

4.) RISC has a simple addressing mode and fixed instruction length for the pipeline execution.

5.) It uses LOAD and STORE instruction to access the memory location.

6)Simple instruction used for RISC architecture

7) It support few data type and synthesis complex data type

8) It used fixed length of instruction for pipelining

ADVANTAGES OF RISC

ADVANTAGES

- **Simpler instructions:** RISC processors use a set of simple instructions, which makes them easier to decode and execute quickly. This results in faster processing times.
- **Faster execution:** Because RISC processors have a simpler instruction set, they can execute instructions faster than CISC processors.
- **Lower power consumption:** RISC processors consume less power than CISC processors, making them ideal for portable devices.

Complex Instruction Set Computer (CISC)

- Intel developed the CISC processor.
- It has an extensive collection of complex instructions that range from simple to very complex and specializes in the assembly language level, which takes a long time to execute the instructions.
- So, CISC approaches reducing the number of instructions on each program and ignoring the number of cycles per instruction.
- It emphasizes building complex instructions directly in the hardware because the hardware is always faster than software. However, CISC chips are relatively slower than RISC chips but use little instructions.
- Examples of CISC processors are AMD, Intel x86, and the System/360.
- CISC has complex instruction set variable length encoding of instruction and instruction execution take varying number of clock cycle .Due to large number of addressing modes for the operations and instruction, the CISC computer generally require fewer instruction to perform the computation.
- Program written for CISC architecture tend to take less space in memory .

FEATURES/CHARACTERSTICS OF CISC

Some of the crucial features of the CISC processor are:-

1.) CISC may take longer than a single clock cycle to execute the code.

2.) The length of the code is short, so it requires minimal RAM.

3.) It provides more accessible programming in assembly language.

4.) It focuses on creating instructions on hardware rather than software because they are faster to develop.

5.) It comprises fewer registers and more addressing nodes, typically 5 to 20.

6.) CISC processor provides direct manipulation of operand residing in memory

7.) if the frequency of complex operation is high , then the performance of the CISC machine is better to implement

Complex Instruction Set Computer (CISC)

Advantages of CISC

- **Reduced code size:** CISC processors use complex instructions that can perform multiple operations, reducing the amount of code needed to perform a task.
- **More memory efficient:** Because CISC instructions are more complex, they require fewer instructions to perform complex tasks, which can result in more memory-efficient code.

Disadvantages of CISC

- **Slower execution:** CISC processors take longer to execute instructions because they have more complex instructions and need more time to decode them.
- **More complex design:** CISC processors have more complex instruction sets, which makes them more difficult to design and manufacture.
- **Higher power consumption:** CISC processors consume more power than RISC processors because of their more complex instruction sets

DIFFERENCE BETWEEN-(AKTU 2016-17/2017-18/2019-20/2022-23)

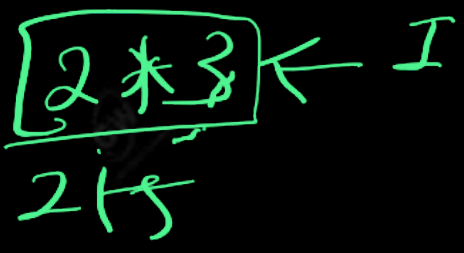
(2 Marks / 7 Marks)

RISC	CISC
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word.	Instructions are larger than the size of one word
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers

DIFFERENCE BETWEEN-

RISC	CISC
Simple and limited addressing modes.	Complex and more addressing modes.
RISC is Reduced Instruction Cycle.	CISC is Complex Instruction Cycle.
The number of instructions are less as compared to CISC.	The number of instructions are more as compared to RISC.
It consumes the low power.	It consumes more/high power.
RISC is highly pipelined.	CISC is less pipelined.
RISC required more RAM.	CISC required less RAM.
Here, Addressing modes are less.	Here, Addressing modes are more.

PIPELINING(AKTU 2022-23/2021-22/2018-19/2019-20)



- To improve the performance of a CPU we have two options:

1) Improve the hardware by introducing faster circuits.
2) Arrange the hardware such that more than one operation can be performed at the same time. Since there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option

Pipelining

- **It** is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.
- Instruction pipelining is a technique that implements a form of parallelism called as instruction level parallelism within a single processor.

PIPELINING

➢Let us see a real-life example that works on the concept of pipelined operation.

➢ Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(**I**), Filling water in the bottle(**F**), and Sealing the bottle(**S**). Let us consider these stages as stage 1, stage 2, and stage 3 respectively.

➢ Let each stage take 1 minute to complete its operation. Now, in a non-pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Similarly, when the bottle moves to stage 3, both stage 1 and stage 2 are idle.

➢But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3. Hence, the average time taken to manufacture

PIPELINING

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Thus we can say pipelining increase effeciency

PIPELINING

- In pipelined architecture,
- The hardware of the CPU is split up into several functional units.
- Each functional unit performs a dedicated task.
- The number of functional units may vary from processor to processor.
- These functional units are called as stages of the pipeline.
- Control unit manages all the stages using control signals.
- There is a register associated with each stage that holds the data.
- There is a global clock that synchronizes the working of all the stages.
- At the beginning of each clock cycle, each stage takes the input from its register.
- Each stage then processes the data and feed its output to the register of the next stage.

3

Fetch, Decode
Execute

5 stages PIPELINING

RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set.

Following are the 5 stages of the RISC pipeline with their respective operations:

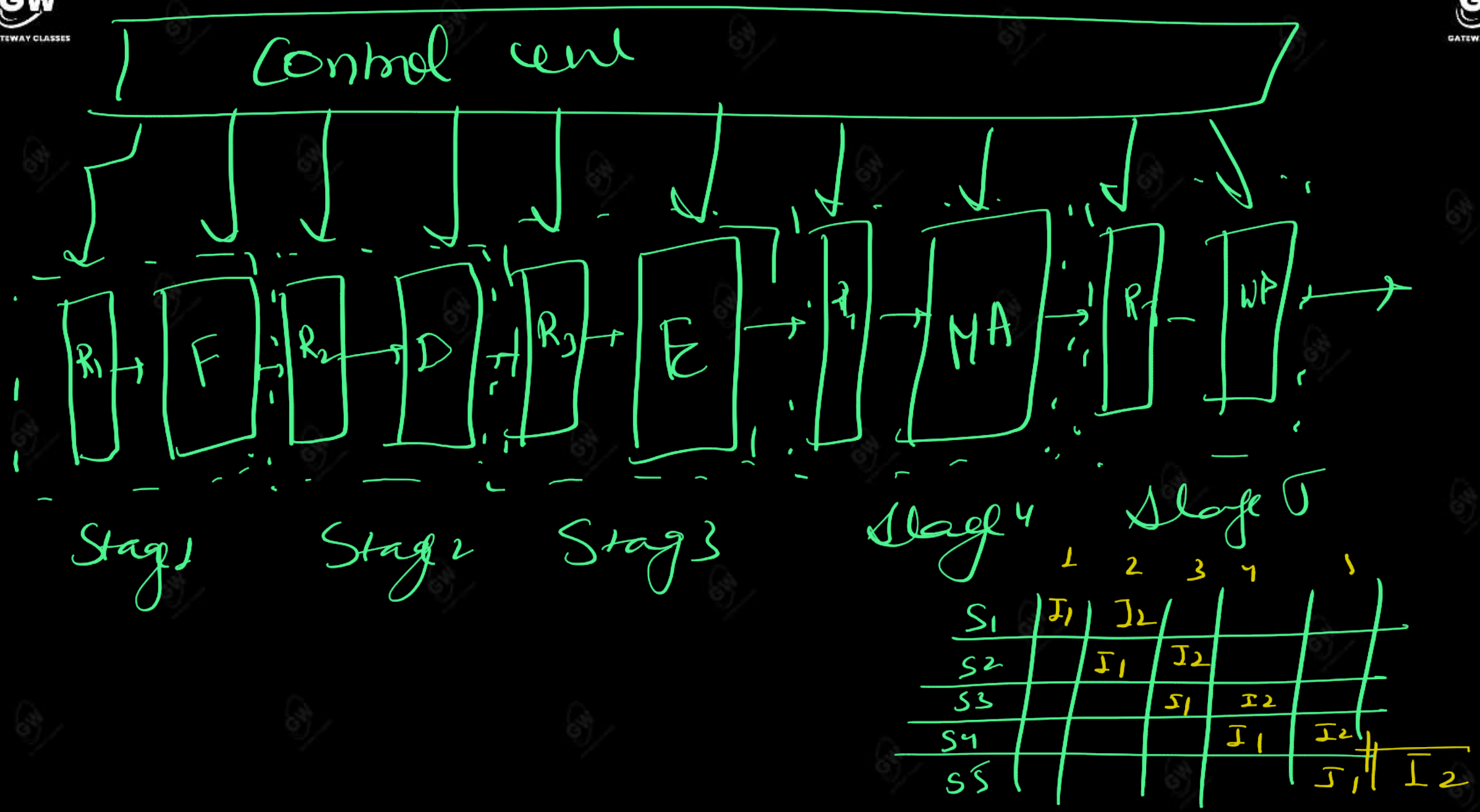
Stage 1 (Instruction Fetch) In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode) In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

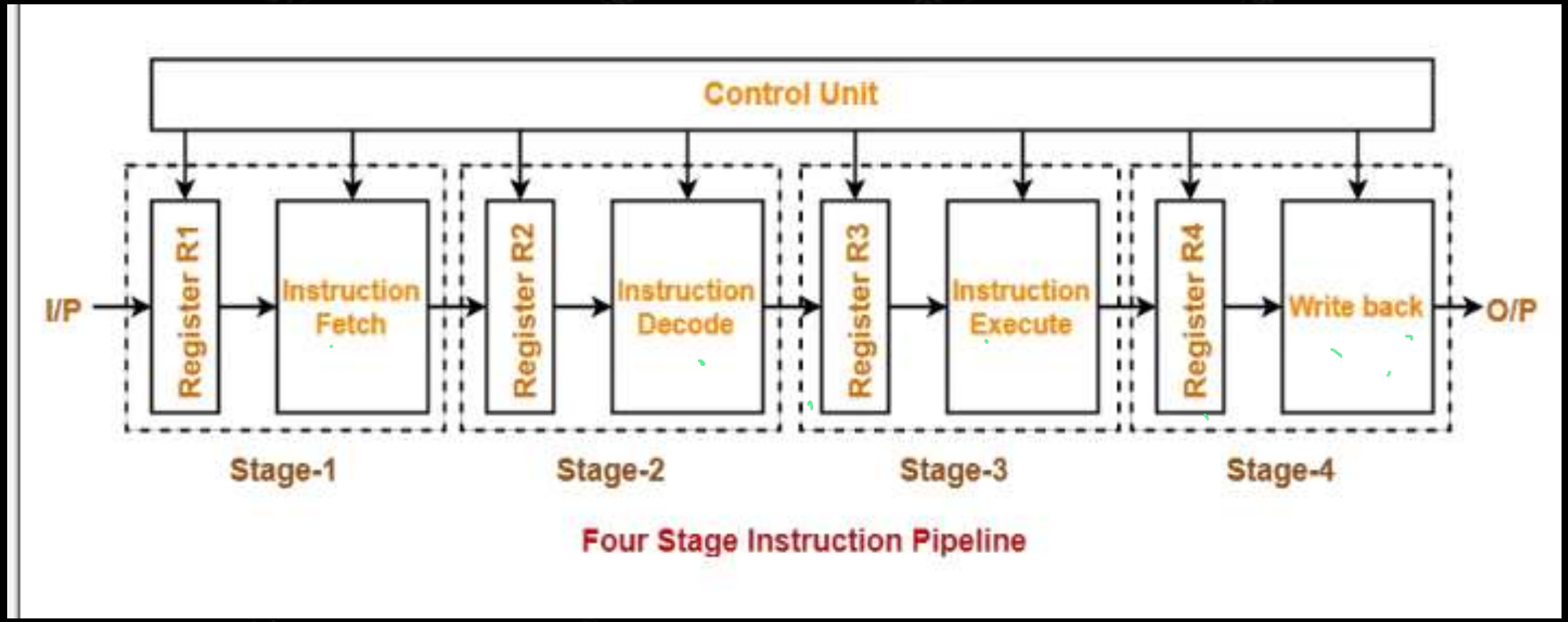
Stage 3 (Instruction Execute) In this stage, ALU operations are performed.

Stage 4 (Memory Access) In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back) In this stage, computed/fetched value is written back to the register present in the instructions.


Control unit
R1
F
R2
D
R3
E
R4
MA
R5
WB
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
1 2 3 4 1
S1 I1 I2
S2 I1 I2
S3 I1 I2
S4 I1 I2
S5 I1 I2

Give a space time diagram for visualizing the pipeline behavior for a four stage pipeline(AKTU 2017-18)



Four Stage Instruction Pipeline

4 STAGE PIPELINING

(Fetch)

Stage-01: At stage-01, First functional unit performs instruction fetch. It fetches the instruction to be executed.

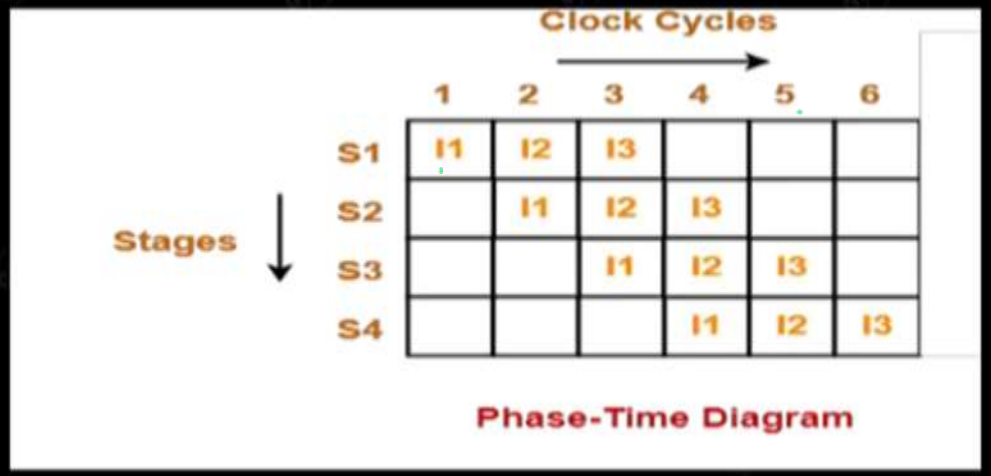
Stage-02: At stage-02, Second functional unit performs instruction decode. It decodes the instruction to be executed.

decode

Stage-03: At stage-03, Third functional unit performs instruction execution. It executes the instruction.

Exec

Stage-04: stage-04, Fourth functional unit performs write back. It writes back the result so obtained after executing the instruction.



PERFORMACE OF PIPELINING PROCESSOR(AKTU 2017-18)

(Performance metrics)

Performance of a pipelined processor Consider a 'k' segment pipeline with clock cycle time as 'Tp'. Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n – 1' instructions will take only '1' cycle each, i.e, a total of 'n – 1' cycles. So, time taken to execute 'n' instructions in a pipelined processor:

I_1, I_2, I_3, I_4

3 stages

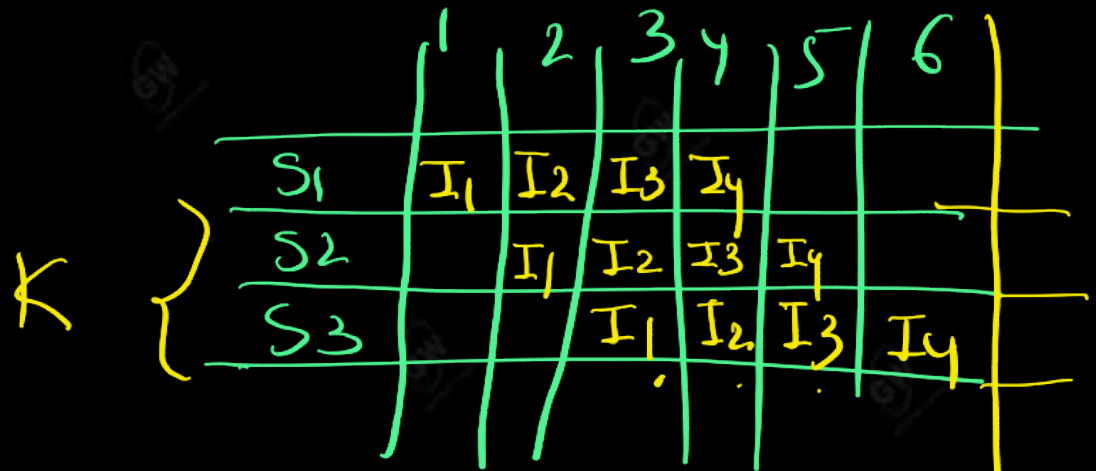
n

$ET_{pipeline} = k + n - 1$ cycles

$= (k + n - 1)\ Tp$

In the same case, for a non-pipelined processor, the execution time of 'n' instructions will be:

$ET_{non\text{-}pipeline} = n * k * Tp$



SPEED UP OF PIPELINING PROCESSOR(2021-22)

So, speedup (S) of the pipelined processor over the non-pipelined processor, when 'n' tasks are executed on the same processor is:

S = Performance of non-pipelined processor /Performance of pipelined processor

As the performance of a processor is inversely proportional to the execution time, we have,

S = ETnon-pipeline / ETpipeline

=> S = [n * k * Tp] / [(k + n – 1) * Tp]

S = [n * k] / [k + n – 1]

When the number of tasks 'n' is significantly larger than k, that is, n >> k

S = n * k / n

S = k

$$\frac{n * k * Tp}{[k + n - 1] * Tp}$$

1 Lakh

1

PIPELINING

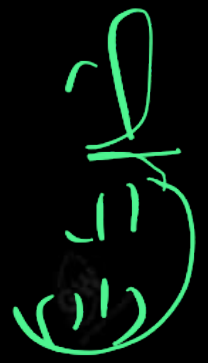
GW GATEWAY CLASSES

where ‘k’ are the number of stages in the pipeline. Also, Efficiency = Given speed up / Max speed up = S / Smax We know that Smax = k So, Efficiency = S / k Throughput = Number of instructions / Total time to complete the instructions So, Throughput = n / (k + n – 1) * Tp Note: The cycles per instruction (CPI) value of an ideal pipelined processor is 1

Efficiency = S/k

THROGHPUT OF PIPELINING PROCESSOR

Performance of pipeline is measured using two main metrices as Throughput and latency.



Throughput:

- It measure number of instruction completed per unit time.
- It represents overall processing speed of pipeline.
- Higher throughput indicate processing speed of pipeline.
- Calculated as, throughput= number of instruction executed/ execution time.
- It can be affected by pipeline length, clock frequency. efficiency of instruction execution and presence of pipeline hazards or stalls.

LATENCY OF PIPELINING

Latency:

- It measure time taken for a single instruction to complete its execution.
- It represents delay or time it takes for an instruction to pass through pipeline stages.
- Lower latency indicates better performance .
- It is calculated as, Latency= Execution time/ Number of instruction executed.
- It in influenced by pipeline length, depth, clock cycle time, instruction dependencies and pipeline hazards..

Classification/Different of pipelining(AKTU 2020-21/2018-19)

➢Arithmetic Pipelining

➢Arithmetic Pipelines are commonly used in various high-performance computers. They are used in order to implement floating-point operations, fixed-point multiplication, and other similar kinds of calculations that come up in scientific situations.

➢Let's look at an example to better understand the ideas of an arithmetic pipeline. We perform addition and subtraction of floating points on a unit of the pipeline here.

➢The inputs in the floating-point adder pipeline refer to two different normalized floating-point binary numbers. These are defined as follows:

➢$A = X * 2^x = 0.9504 * 10^3$

➢$B = Y * 2^y = 0.8200 * 10^2$

$0.9504 * 10^3$

$0.0820 * 10^3$

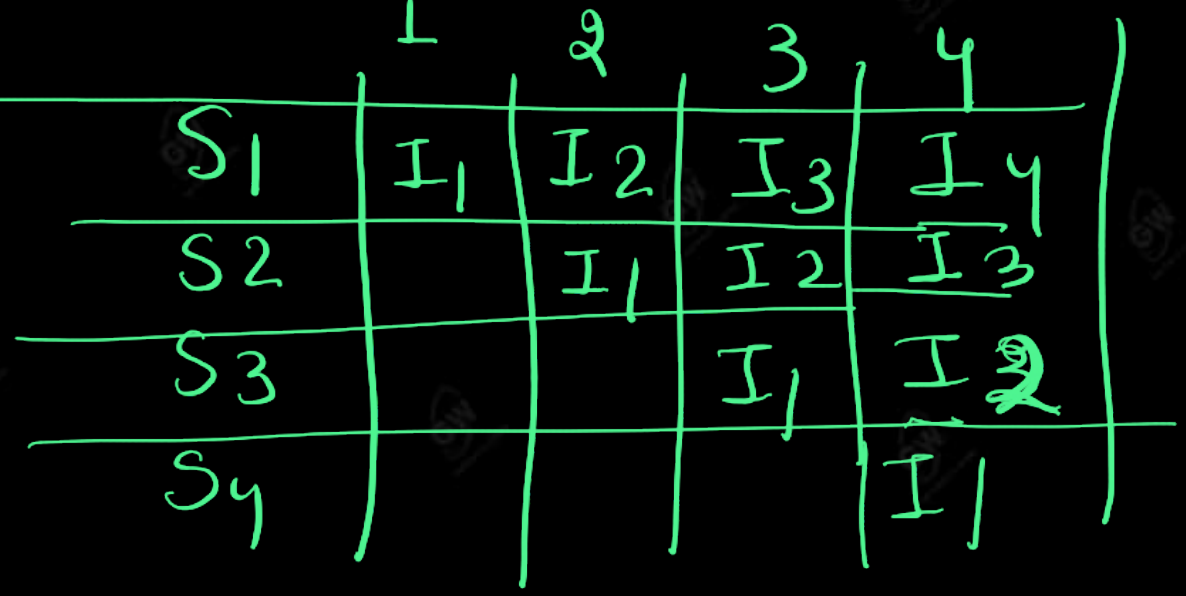
Classification of pipelining

➢ Arithmetic Pipelining

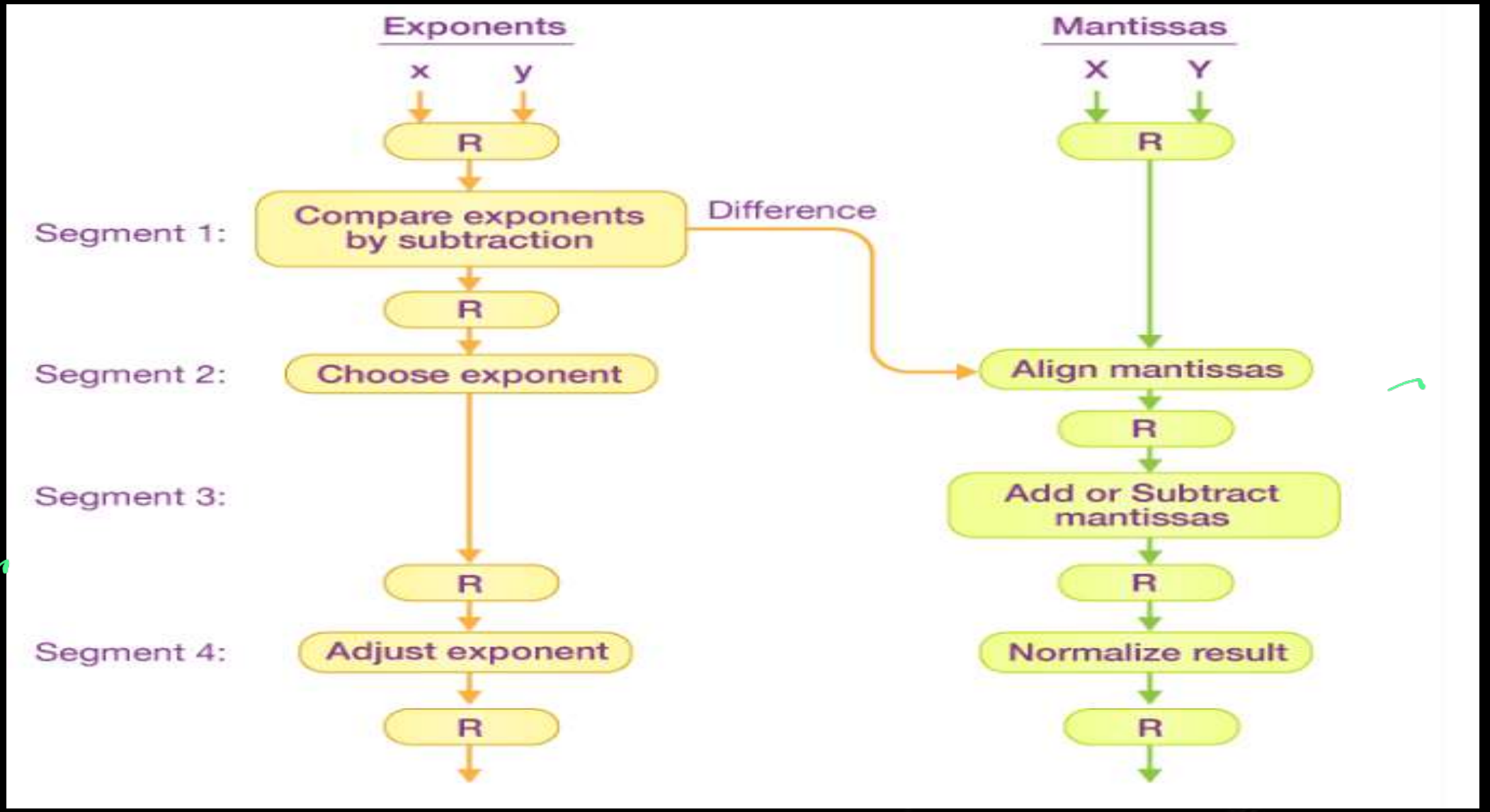
Where x and y refer to the exponents and X and Y refer to two fractions representing the mantissa.

The floating-point addition and subtraction process is broken into four pieces. The matching sub-operation to be executed in the specified pipeline is contained in each segment. The four segments depict the following sub-operations:

1. Comparing the exponents using subtraction S1
2. Aligning the mantissa S2
3. Adding or subtracting the mantissa S3
4. Normalizing the result S4



Classification of pipelining



Classification of pipelining

➢Arithmetic Pipelining

Note: The registers are placed after every sub-operation in order to store the intermediate results.

1. Comparing Exponents by Subtraction

The difference between the exponents is calculated by subtracting them. The result's exponent is chosen to be the larger exponent.

The exponent difference, 3 – 2 = 1, defines the total number of times the mantissa associated with the lesser exponent should be shifted to the right.

2. Aligning the Mantissa

As per the difference of exponents calculated in segment one, the mantissa corresponding with the smaller exponent would be moved.

$A = 0.9504 * 10^3$

$B = 0.08200 * 10^3$

Classification of pipelining

3. Adding the Mantissa

Both the mantissa would be added in the third segment.

$C = A + B = 1.0324 * 10^3$

4. Normalizing the Result

After the process of normalization, the result would be written as follows:

$C = 0.1324 * 10^4$

Classification of pipelining

Advantages of Arithmetic Pipeline

- **Improved throughput:** Arithmetic pipelines allow multiple arithmetic operations to be executed simultaneously, leading to increased throughput and faster computation.
- **Reduced latency:** The pipelining of arithmetic operations reduces the overall latency, making computations more efficient.
- **Parallelism:** Pipelining enables parallel execution of arithmetic stages, utilizing hardware resources effectively.
- **Complex operations:** It simplifies complex arithmetic operations by breaking them down into smaller, manageable stages.
- **Enhanced performance:** Arithmetic pipelines improve modern computer architectures' overall performance.

Classification of pipelining

Instruction pipelining

Pipeline processing can happen not only in the data stream but also in the instruction stream. To perform tasks such as fetching, decoding and execution of instructions, most digital computers with complicated instructions would require an instruction pipeline.

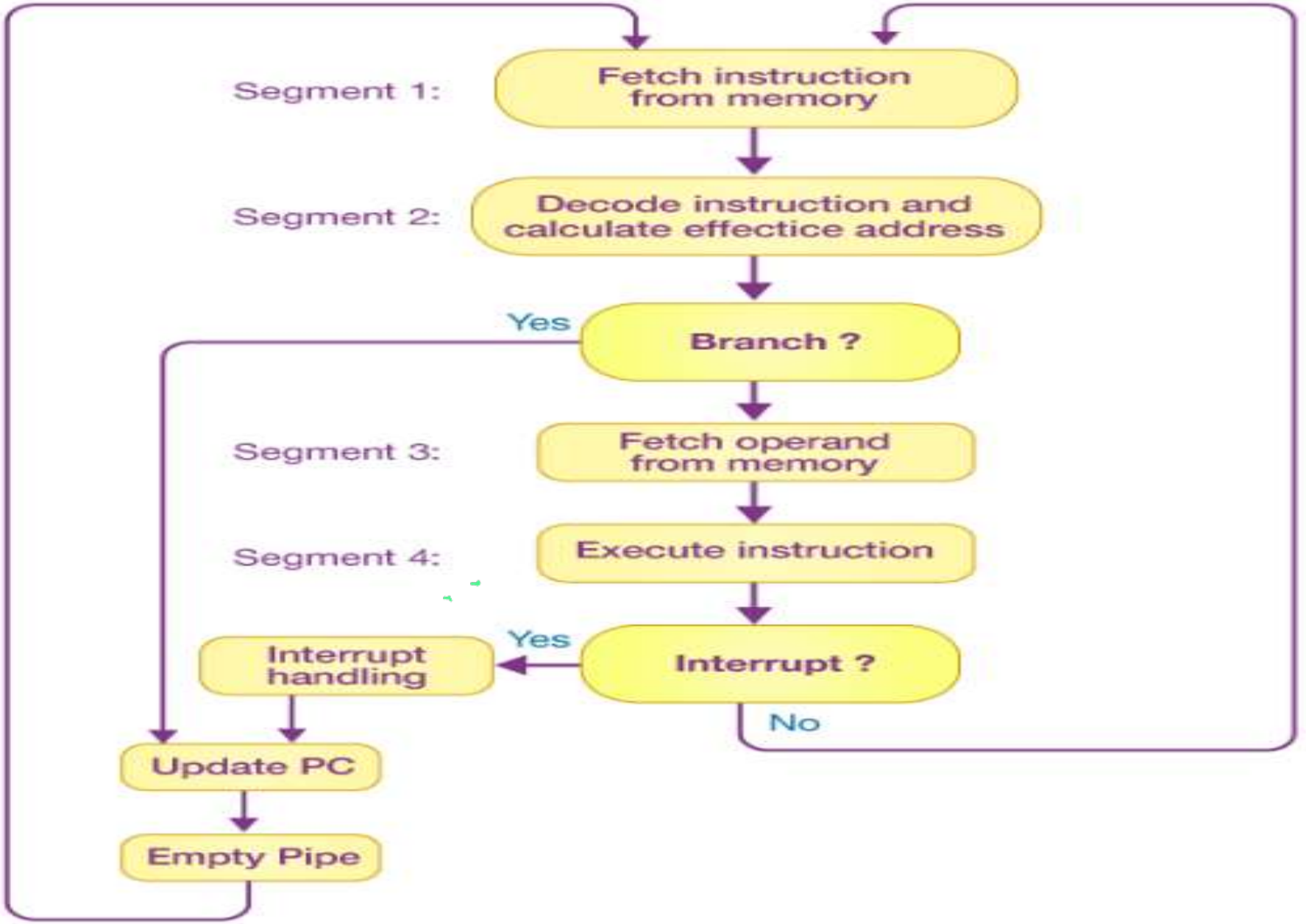
In general, each and every instruction must be processed by the computer in the following order:

1. Fetching the instruction from memory
2. Decoding the obtained instruction
3. Calculating the effective address
4. Fetching the operands from the given memory
5. Execution of the instruction
6. Storing the result in a proper place

} single segment

Classification of pipelining

- Each step is carried out in its own segment, and various segments may take different amounts of time to process the incoming data. Furthermore, there are occasions when multiple segments request memory access at the very same time, requiring one segment to wait unless and until the memory access of another is completed.
- If the instruction cycle is separated into equal-length segments, the organisation of an instruction pipeline will become much more efficient. A four-segment type of instruction pipeline refers to one of the most common instances of this style of organisation.
- A four-segment instruction pipeline unifies two or more distinct segments into a single unit. For example, the decoding of the instruction and the calculation of the effective address can be merged into a single segment.


Segment 1:
Fetch instruction from memory
Segment 2:
Decode instruction and calculate effectice address
Yes
Branch ?
Segment 3:
Fetch operand from memory
Segment 4:
Execute instruction
Yes
Interrupt handling
Interrupt ?
No
Update PC
Empty Pipe

Classification of pipelining

instruction pipelining

four-segment instruction pipeline is illustrated in the block diagram given above. The instructional cycle is divided into four parts:

Segment 1

The implementation of the instruction fetch segment can be done using the FIFO or first-in, first-out buffer.

Segment 2

In the second segment, the memory instruction is decoded, and the effective address is then determined in a separate arithmetic circuit.

Segment 3

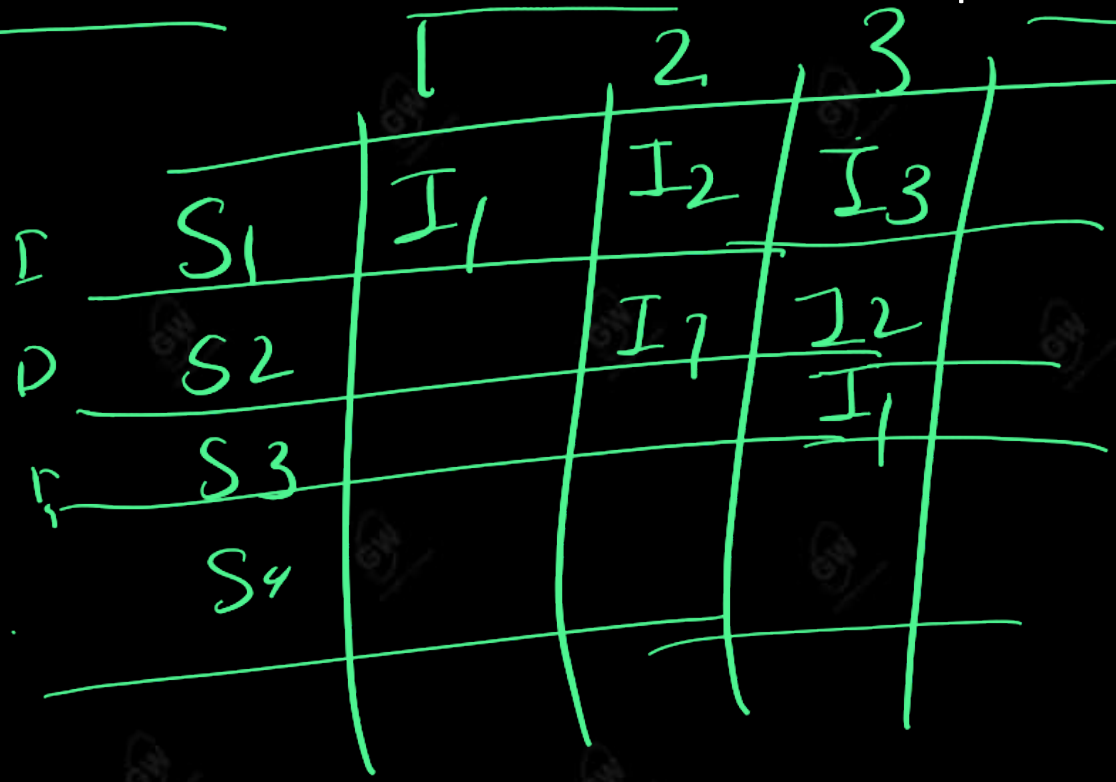
In the third segment, some operands would be fetched from memory.

Segment 4

The instructions would finally be executed in the very last segment of a pipeline organisation

Difference-

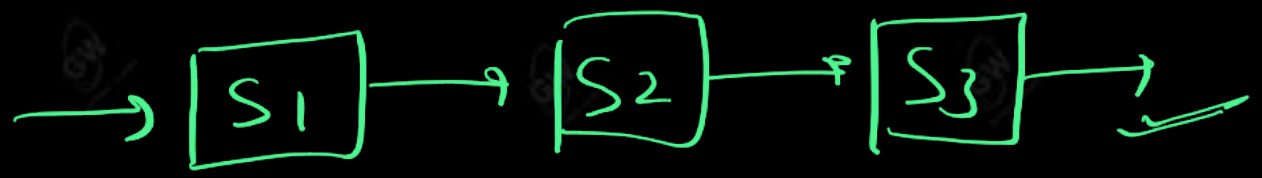
Arithmetic Pipelining focuses on parallelizing arithmetic operations, while Instruction Pipelining optimizes the execution of instructions. Arithmetic Pipelines break down arithmetic operations into stages, whereas Instruction Pipelines break down the instruction execution process into stages.



Difference between-(AKTU 2019-20)

Linear Pipeline	Non-Linear Pipeline
In linear pipeline a series of processors are connected together in a serial manner.	In Non-Linear pipeline different pipelines are present at different stages.
Linear pipeline is also called as static pipeline as it performs fixed functions.	Non-Linear pipelines is also called as dynamic pipeline as it performs different functions.
The output is always produced from the last block.	The output is not necessarily produced from the last block.
Linear pipeline has linear connections.	Non-Linear pipeline has feedback and feed-forward connections.
It generates a single reservation table.	It can generate more than one reservation table.
It allows easy functional partitioning.	Functional partitioning is difficult in non-linear pipeline.

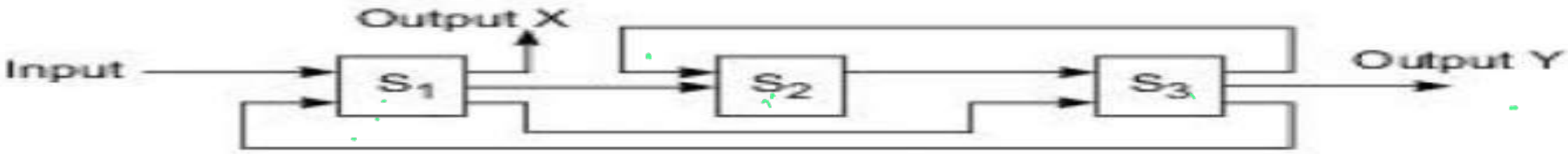
Linear pipeline



	1	2	3
S1	X		
S2		X	
S3			X

Reservation

Non-Linear pipeline



(a) A three-stage pipeline

Time

Stages	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

(b) Reservation table for function X

Time

Stages	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

(c) Reservation table for function Y

Difference between-(AKTU 2019-20)

Aspect	Pipelining	Parallelism
Nature of Concurrency	Achieves concurrency by breaking down the execution of a single instruction into stages and processing multiple instructions in different stages simultaneously.	Achieves concurrency by executing multiple instructions or tasks simultaneously, either at the instruction level, thread level, or data level.
Dependency Handling	Dependencies between instructions must be carefully managed to prevent hazards, such as data hazards and control hazards, which can affect the pipeline's efficiency.	Dependencies between instructions or tasks must also be considered, but different techniques, such as out-of-order execution or dynamic scheduling, can be employed to handle dependencies.
Resource Usage	Uses a single set of hardware components that are shared among different stages of the pipeline.	Involves multiple sets of hardware components that can operate concurrently on different instructions or tasks.

Difference between-

Aspect	Pipelining	Parallelism
Workflow	Each stage of the pipeline is responsible for a specific task, and as soon as one stage completes its task, it passes the result to the next stage.	Simultaneous execution of multiple instructions or tasks, either at the instruction level, thread level, or data level.
Advantages	- Improved throughput and overall system performance. - Increased instruction throughput by overlapping stages. - Utilizes resources more efficiently.	- Faster execution of tasks by performing them simultaneously. - Better resource utilization. - Enhanced overall system throughput.

CONTROL UNIT

➢Control Unit :

The unit which directs the operation of the processor & is a part of the CPU is known as Control Unit. It generates control signals for the operations of a computer

Types of Control Unit :

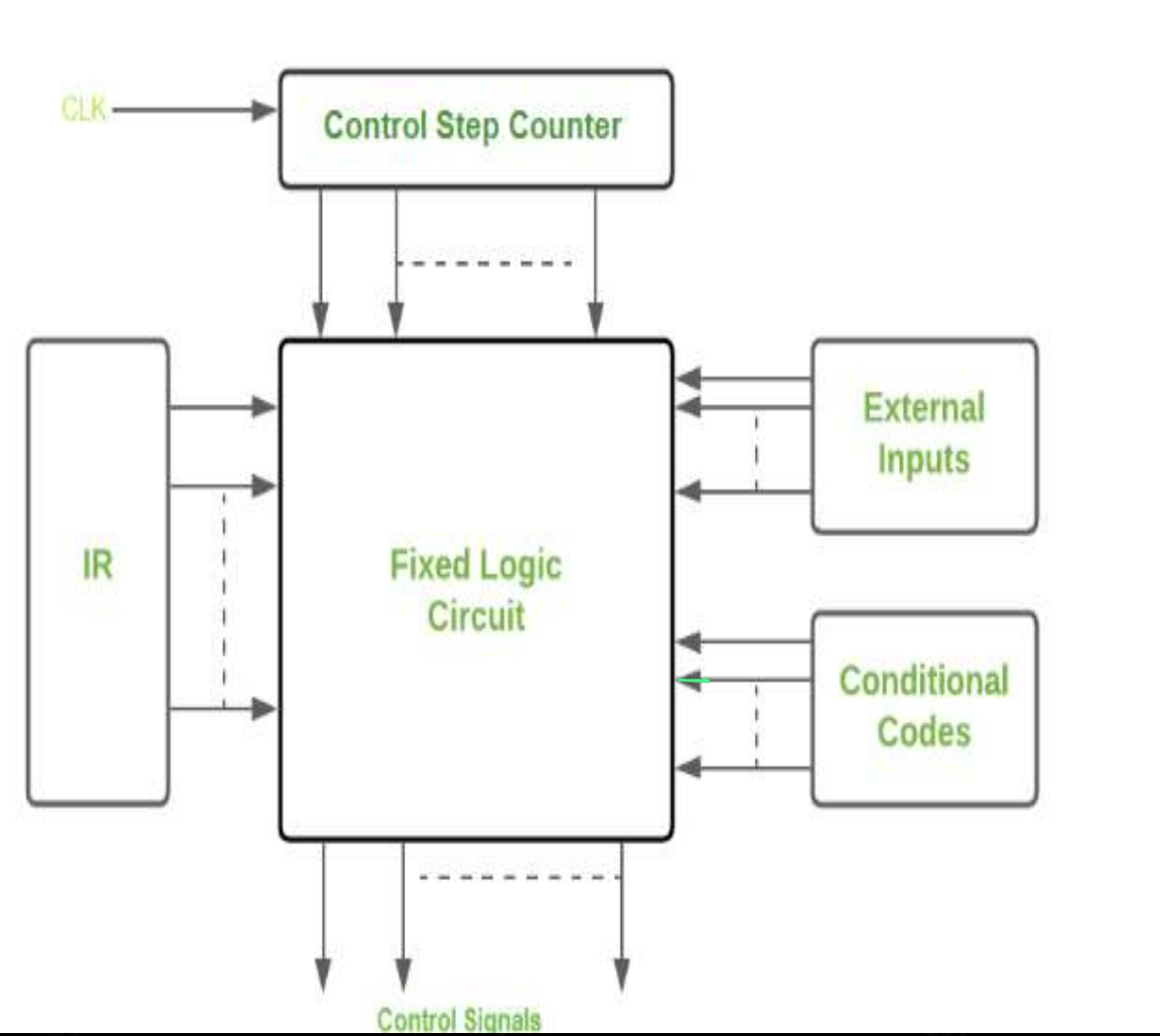
There are two types of control units as follows.

- ➢Hardwired control unit
- ➢Micro-programmed control unit

Hardwired control unit(2016-17/2017-18/2021-22)

- The name suggests, these control units are designed using hardware components such as sequential logic circuits or some finite state machines, for example, logic gates, flip-flops, decoders, resistors and other digital circuits to generate a specific sequence of control signals.
- To interpret the instructions & generate control signals, the control unit uses fixed logic circuits.
- To generate signals, the fixed logic circuits use the contents of the control step counter, Instruction Register (IR) & code flag, and some external input signals such as interrupt signals. The fixed logic circuit in the diagram is a combinational circuit made from decoders & encoders.
- It generates the output based on the state of its input(s). The decoder decodes the instruction loaded in IR (Instruction Register) & generates the signal that serves as an input to the encoder.
- Also, external input & conditional codes act as an input to the encoder. The encoder then accordingly generates the control signals based on the inputs. After the execution of each instruction, another signal: the end signal is generated which resets the state of control step counter & makes it ready for the next instruction

Hardwired control unit



BLOCK DAIGRAM OF HARDWIRED CONTROL UNIT

Advantages of Hardwired Control Unit :

Here, we will discuss the advantages of the Hardwired Control Unit as follows.

- Because of the use of combinational circuits to generate signals, Hardwired Control Unit is fast.
- It depends on number of gates, how much delay can occur in generation of control signals.
- It can be optimized to produce the fast mode of operation.
- Faster than micro- programmed control unit.
- It does not require control memory. (ROM)

Disadvantages of Hardwired Control Unit :

- The complexity of the design increases as we require more control signals to be generated (need of more encoders & decoders)
- Modifications in the control signals are very difficult because it requires rearranging of wires in the hardware circuit.
- Adding a new feature is difficult & complex.
- Difficult to test & correct mistakes in the original design. It is Expensive.

Method to design hardwired control

1 state table method

2 Delay element method

3 sequential counter method

4 PLA METHOD

STATE TABLE METHOD

T - States	Instructions			
	I_1	I_2	...	I_N
T_1	$C_{1,1}$	$C_{1,2}$	...	$C_{1,N}$
T_2	$C_{2,1}$	$C_{2,2}$	...	$C_{2,N}$
...	...	...	...	...
T_M	$C_{M,1}$	$C_{M,2}$	...	$C_{M,N}$

$C_{1,1}$ means control signal to be produced in T-state(T_1) of Instruction (I_1)

STATE TABLE METHOD:

- Here the behavior of control unit is represented in the form of a table, which is known as the **state table**.
- Here, each row represents the T-states and the columns represent the instructions.
- Every intersection of the specific column to each row indicates which control signal will be produced in the corresponding T- state of an instruction.
- Here the hardware circuitry is designed for each column(i.e. for a specific instruction) for producing control signals in different T-states.

STATE TABLE METHOD:

➢ **Advantage –**

➢ It is the simplest method.

➢ This method is mainly used for small instruction set processors(i.e. in RISC processors)

. Disadvantages

➢ In modern processors ,there is a very large number of instruction set. Therefore, the circuit becomes complicated to design, difficult to debug, and if we make any modifications to the state table then the large parts of the circuit need to be changed.

➢ Therefore ,this is not widely used for these kinds of processors.

➢ There are many redundancies in circuit design like the control signals are required for fetching the instruction is common and which is repeated for N number of instruction. So the cost of circuitry design may increase

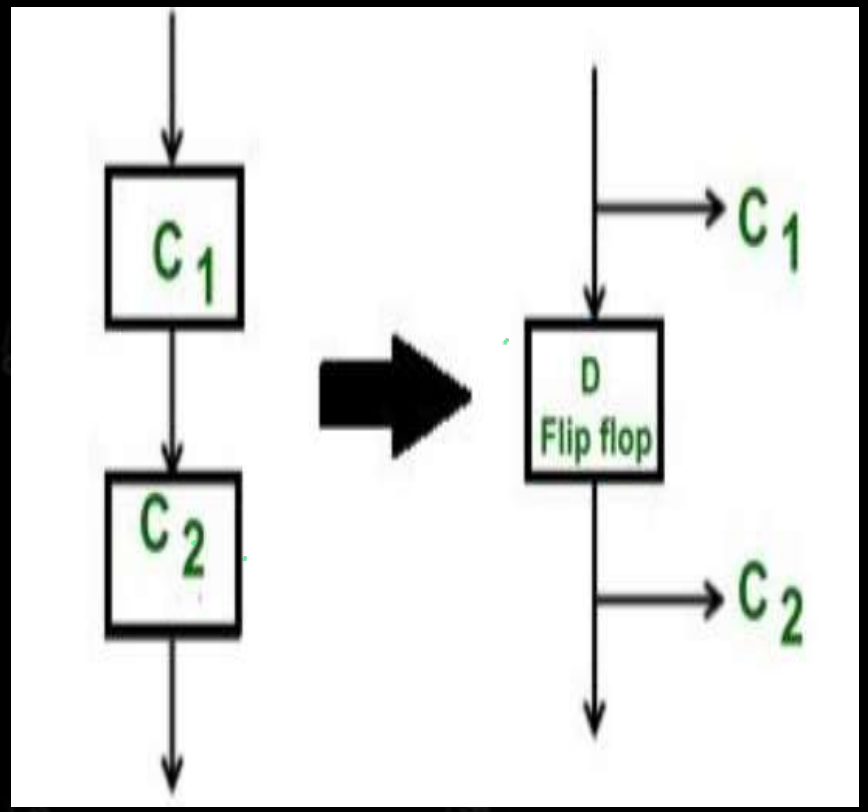
DELAY ELEMENT METHOD:

- Here the control unit behavior is represented in the form of a **flowchart**.
- Each step in the flowchart represents a control signal that needs to be produced for processing the instructions.
- If all the steps of the instructions are performed, this means the instruction is executed completely.
- **Control signals perform micro-operations and each micro-operation requires one T-state.**
- For the micro-operations which are independent, they are required to be performed in different T-state.

Therefore, for every consecutive control signal an exactly 1-state delay is required, which can be produced with the help of D FF.

- Therefore. D Flip-Flops are inserted between every two consecutive control signals

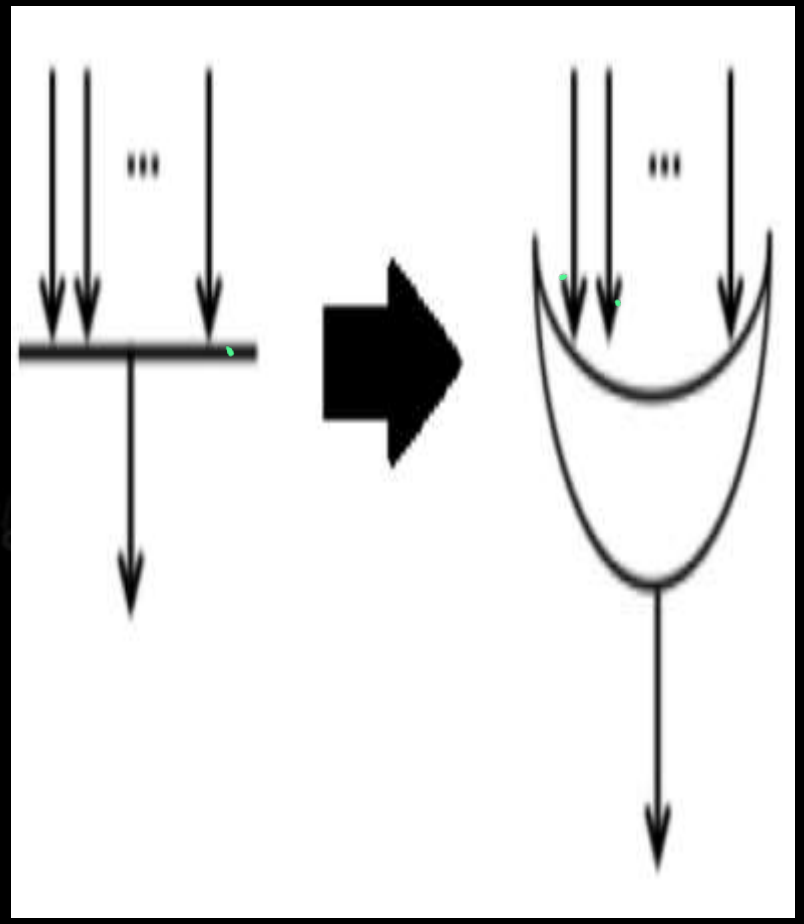
Hardwired control unit

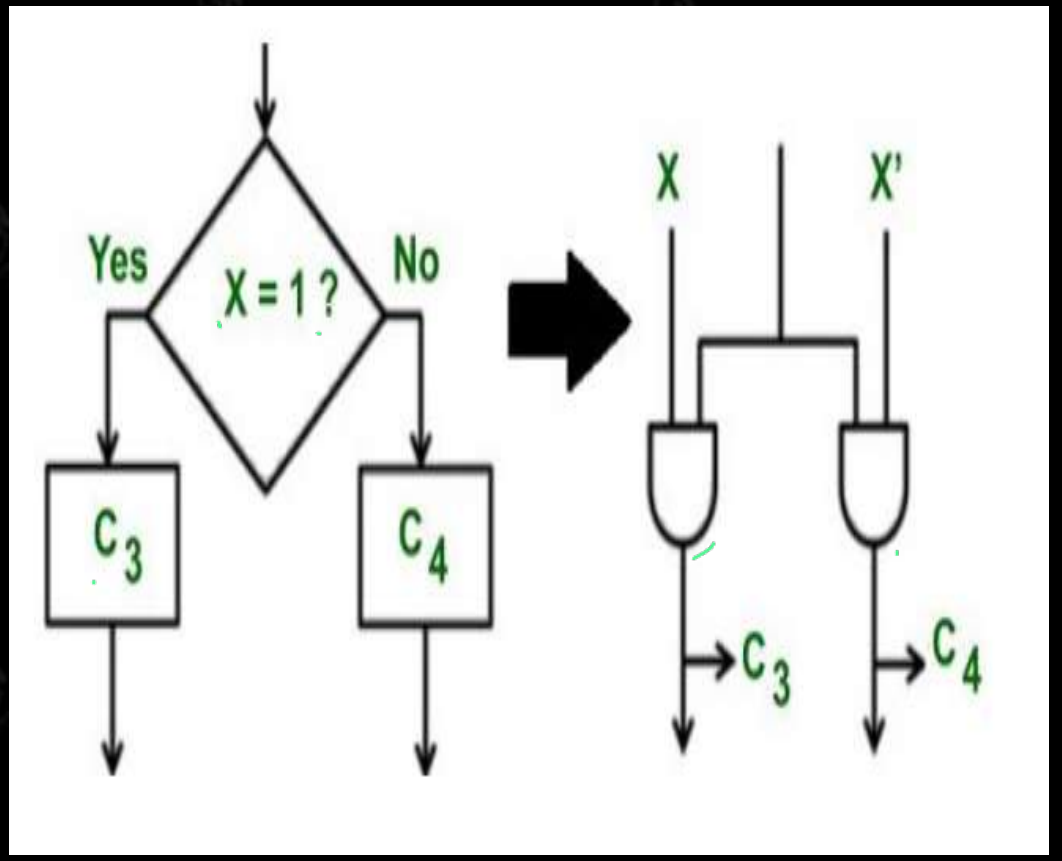


DELAY ELEMENT METHOD:

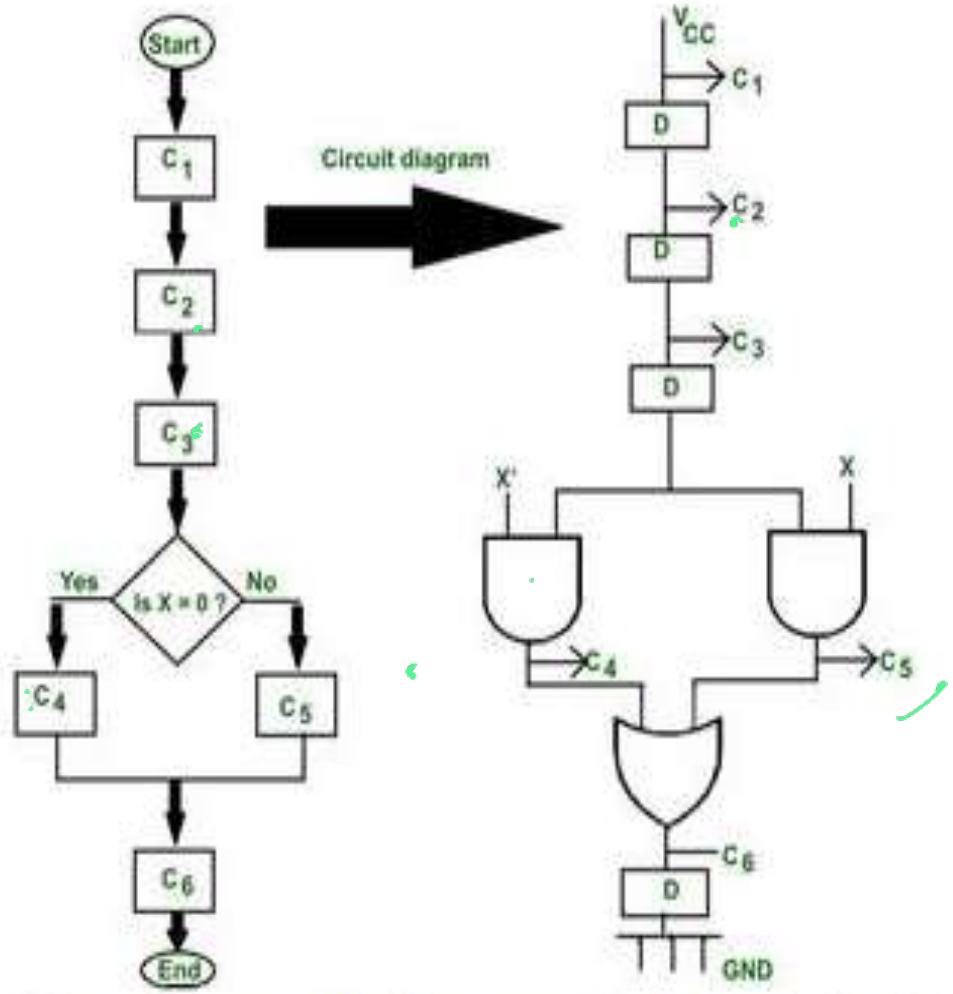
- As we can observe, the D FF is introduced between each pair of control signals .Therefore, after a control signal is generated, then the delay element before that control signal is not in use until before the next instruction required that control signal. Therefore, of all D Flip-Flops, only one will be active at a time. Therefore, this method is also known as **one hot method**.
- In a flowchart, if there is a multiple entry point for control signal then to combine two or more paths, we use an OR gate.

Hardwired control unit





Hardwired control unit



Delay element method for generating control signals.

Suppose the processor has two instructions add or subtract (Therefore an opcode of 1 bit is needed in which 0 opcode for add instruction and 1 for subtract is used.

➢ **Flowchart design –**

Say C_1,C_2,C_3 is the control signals for fetching the instruction. When X= 0, then C_4 control signal is produced (i.e. decoding) which is used for performing add operation, and when x=1, then control signal C_5 will be produced for performing the subtract operation. And c_6 control signal is used for storing the result and the process ends

some basic terms AKTU 2015-16/2020-21/2018-19

Micro programmed control unit:

A control unit whose binary control variables are stored in memory is called a micro programmed control unit

Control Memory: (CROM)

Control Memory is the storage in the micro-programmed control unit to store the microprogram.

Writeable Control Memory:

Control Storage whose contents can be modified, allow the change in microprogram and Instruction set can be changed or modified is referred as Writeable Control Memory.

Control Word:

The control variables at any given time can be represented by a control word string of 1 's and 0's called a control word .

Control variable The control function that specifies a micro operation is called as ***control variable***

SOME BASIC TERMS

Microcode:

➢Micro-code is a layer of hardware –level instruction and /or data structures involved in the implementation of higher level machine code in many computer and other processor.

➢It helps to separate the machine instruction from the underlying electronics so that instruction can be designed and altered more easily

Micro program:

➢ A sequence of microinstructions constitutes a microprogram.

Micro instruction:

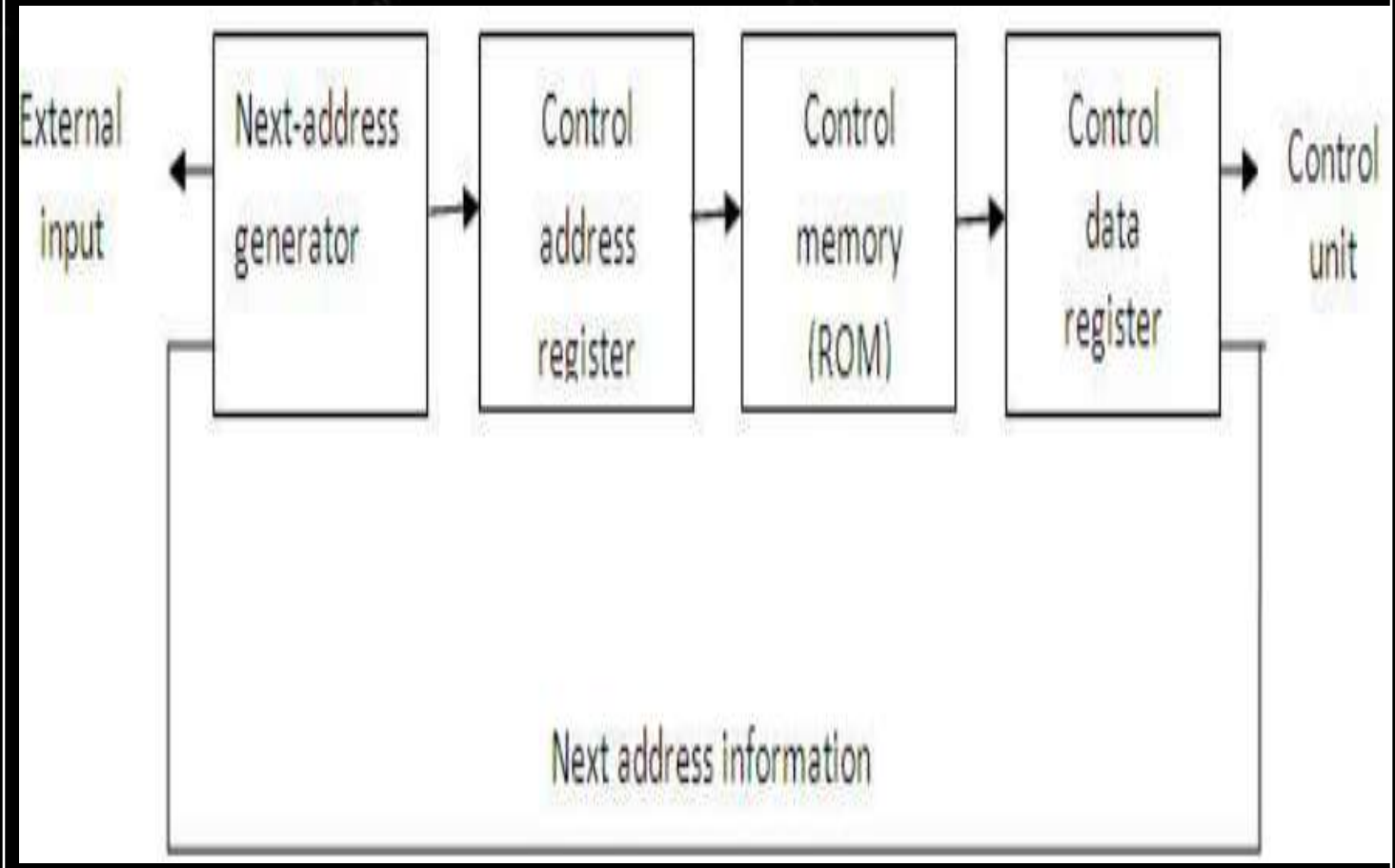
➢A symbolic microprogram can be translated into its binary equivalent by means of an assembler.

➢ Each line of the assembly language microprogram defines a symbolic microinstruction

Micro programmed control unit (AKTU 2019-20/2021-22)

- A control unit whose binary control variables are stored in memory is called a micro programmed control unit
- Control information is stored in control memory.
- Control memory is programmed to initiate the required sequence of micro-operations.
- The key characteristics are
- Speed of operation is low when compared with hardwired
  - o Less complex ,Less expensive
  - o Flexibility to add new instructions

.



block diagram of Micro programmed control unit:

- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control memory address register(CAR) specifies the address of the microinstruction, and the control data register (CDR)holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more micro-operations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory. .
- While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- Thus a microinstruction contains bits for initiating micro-operations in the data processor part and bits that determine the address sequence for the control memory.
- The next address generator is sometimes called a microprogram sequencer. It is used to generate the next micro instruction address.

Typical functions of a micro-program sequencer are

incrementing the control address register by one,

note

- computer with a micro programmed control unit will have two separate memories: a main memory and a control memory
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:
- fetch the instruction from main memory
- evaluate the effective address
- execute the operation
- return control to the fetch phase for the next instruction

Address sequencing AKTU 2021-22

Microinstructions are stored in control memory in groups, with each group specifying a *routine.*

 To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1:

➢ An initial address is loaded into the control address register when power is turned on in the computer.

➢ This address is usually the address of the first microinstruction that activates the instruction fetch routine.

➢The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.

➢At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

➢The control memory next must go through the routine that determines the effective address of the operand.

➢ A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.

Address sequencing

➢The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.

➢When the effective address computation routine is completed, the address of the operand is available in the memory address register..

Step-3:

➢The next step is to generate the microoperations that execute the instruction fetched from memory.

➢The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.

➢Each instruction has its own micro-program routine stored in a given location of control memory.

➢The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a *mapping process.*

➢A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4:

➢Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.

➢ Micro-programs that employ subroutines will require an external register for storing the return address.

Address sequencing

- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return

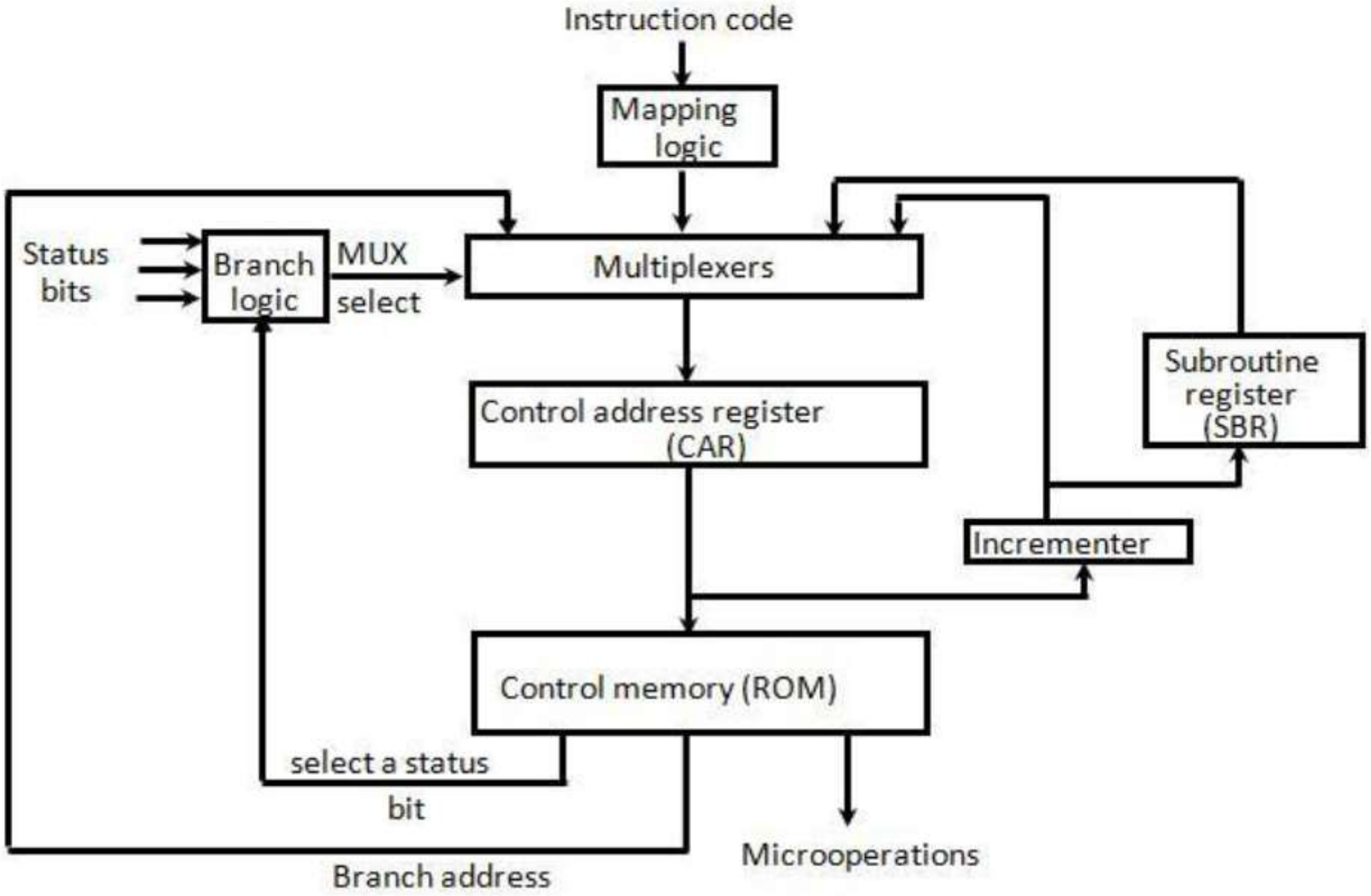


Figure 4.2: Selection of address for control memory

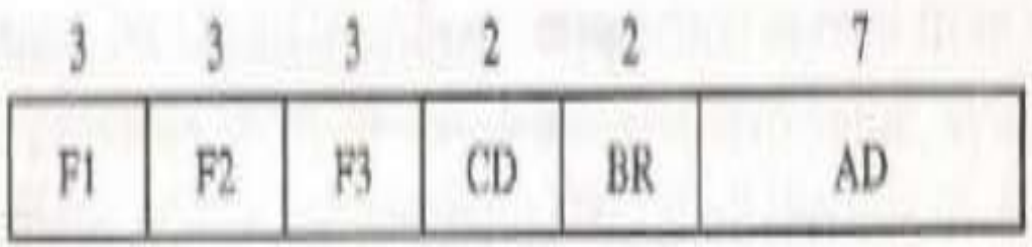
Address sequencing

- Above figure 4.2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.
- The microinstruction in control memory contains a set of bits to initiate micro operations in computer registers and other bits to specify the method by which the next address is obtained.
- The diagram shows four different paths from which the control address register (CAR) receives the address.
- The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.
- Branching is achieved by specifying the branch address in one of the fields of the microinstruction.
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- An external address is transferred into control memory via a mapping logic circuit.
- The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from the subroutine.

Address sequencing

- The branch logic of figure 4.2 provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions. The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

Microinstruction Format (AKTU 2017-18)



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{-}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

Microinstruction Format

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Microinstruction Format

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	*DR*(15)	I	Indirect address bit
10	*AC*(15)	S	Sign bit of *AC*
11	*AC* = 0	Z	Zero value in *AC*

BR	Symbol	Function
00	JMP	*CAR* ← *AD* if condition = 1
		CAR ← *CAR* + 1 if condition = 0
01	CALL	*CAR* ← *AD*, *SBR* ← *CAR* + 1 if condition = 1
		CAR ← *CAR* + 1 if condition = 0
10	RET	*CAR* ← *SBR* (Return from subroutine)
11	MAP	*CAR*(2-5) ← *DR*(11-14), *CAR*(0,1,6) ← 0

DIFFERENCE BETWEEN-(AKTU 2015-16/16-17/19-19/21-22/22-23)

Hardwired Control Unit	Microprogrammed Control Unit
Hardwired control unit generates the control signals needed for the processor using logic circuits	Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardware	This is slower than the other as micro instructions are used for generating signals here
Difficult to modify as the control signals that need to be generated are hard wired	Easy to modify as the modification need to be done only at the instruction level

DIFFERENCE BETWEEN-

Hardwired Control Unit	Microprogrammed Control Unit
More costlier as everything has to be realized in terms of logic gates	Less costlier than hardwired control as only micro instructions are used for generating control signals
It cannot handle complex instructions as the circuit design for it becomes complex	It can handle complex instructions
Only limited number of instructions are used due to the hardware implementation	Control signals for many instructions can be generated

DIFFERENCE BETWEEN-

Hardwired Control Unit	Microprogrammed Control Unit
Used in computer that makes use of Reduced Instruction Set Computers(RISC)	Used in computer that makes use of Complex Instruction Set Computers(CISC)
It can use limited instructions	It can generate control signal for many instructions

TYPES OF MICROPROGRAMMED CONTROL UNIT(AKTU2021-22/2020-21/2019-20/2017-18)

ack Organization?

- The micro-programmed control unit can be classified into two types based on the type of Control Word stored in the Control Memory, viz., Horizontal micro-programmed control unit and Vertical micro-programmed control unit.
- In the Horizontal micro-programmed control unit, the control signals are represented in the decoded binary format, i.e., 1 bit/CS. Here 'n' control signals require n bit encoding. On the other hand.
- In a Vertical micro-programmed control unit, the control signals are represented in the encoded binary format. Here 'n' control signals require $\log_2 n$ bit encoding.

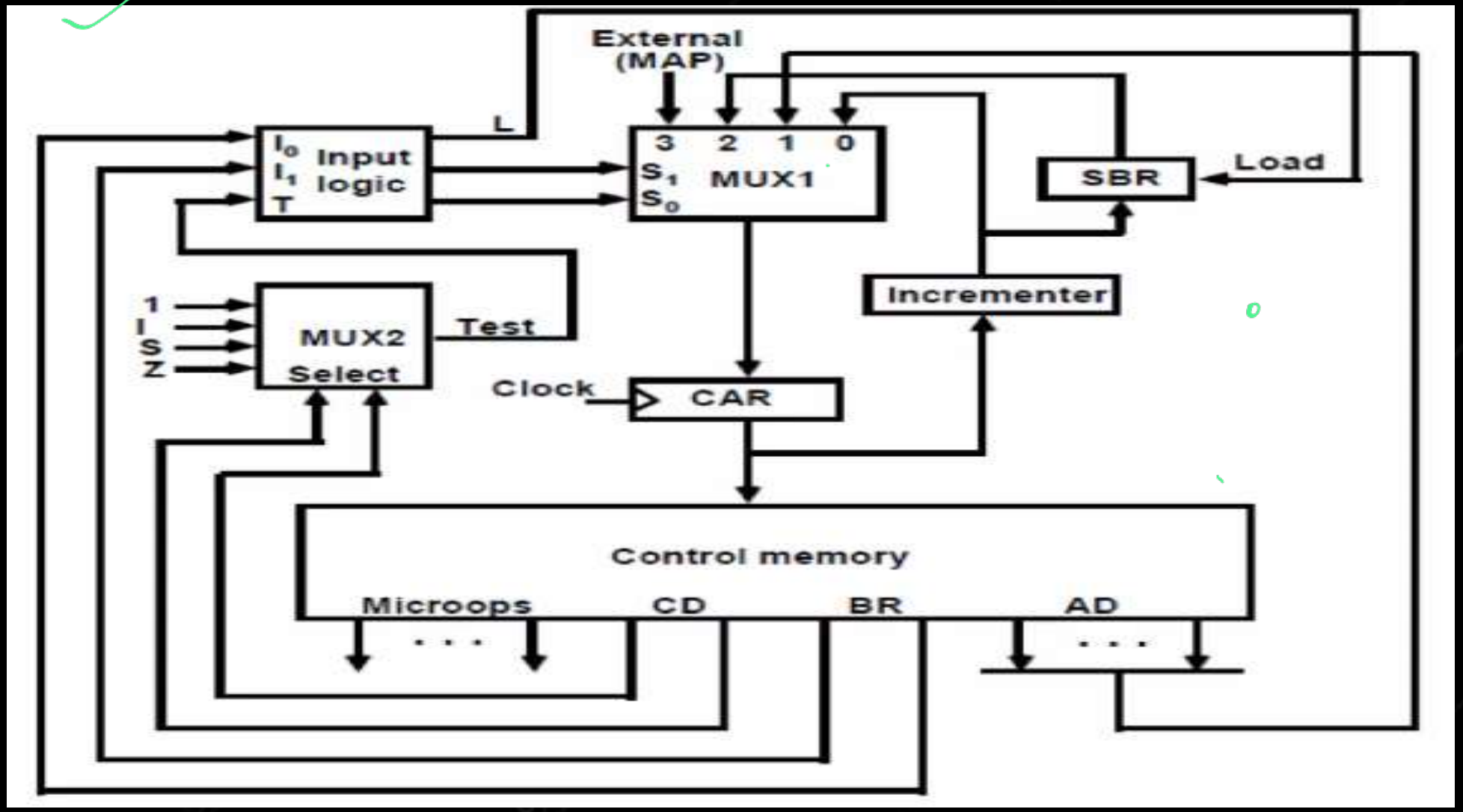
TYPES OF MICROPROGRAMMKED CONTROL UNIT

S. No	Horizontal μ-programmed CU	Vertical μ-programmed CU
1.	It supports longer control word.	It supports shorter control word.
2.	It allows a higher degree of parallelism. If degree is n, then n Control Signals are enabled at a time.	It allows a low degree of parallelism i.e., the degree of parallelism is either 0 or 1.
3.	No additional hardware is required.	Additional hardware in the form of decoders is required to generate control signals.
4.	It is faster than a Vertical micro-programmed control unit.	it is slower than a Horizontal micro-programmed control unit.

TYPES OF MICROPROGRAMMKED CONTROL UNIT

S. No	Horizontal µ-programmed CU	Vertical µ-programmed CU
5.	It is more flexible than a vertical micro-programmed control unit.	It is less flexible than horizontal but more flexible than that of a hardwired control unit.
6.	A horizontal micro-programmed control unit uses horizontal micro-instruction, where every bit in the control field attaches to a control line.	A vertical micro-programmed control unit uses vertical micro-instruction, where a code is used for each action to be performed and the decoder translates this code into individual control signals.
7.	The horizontal micro-programmed control unit makes less use of ROM encoding than the vertical micro-programmed control unit.	The vertical micro-programmed control unit makes more use of ROM encoding to reduce the length of the control word.

MICROPROGRAM SEQUENCER(2021-22/2020-21/2018-19)



MICROPROGRAM SEQUENCER

BR FIELD		INPUT			MUX1		loadSBR
		I_0	I_1	T	S1	S2	/
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	X	1	0	0
1	1	1	0	X	1	1	0

CD TABLE

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{-}5) \leftarrow DR(11\text{-}14)$, $CAR(0,1,6) \leftarrow 0$

MICROPROGRAM SEQUENCER

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.
- The block diagram of the microprogram sequencer is shown in fig

- **The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.**
- **There are two multiplexers in the circuit.**
- **The first multiplexer selects an address from one of four sources and routes it into control address register *CAR*. The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.**
- **The output from CAR provides the address for the control memory The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register *SBR*.**

MICROPROGRAM SEQUENCER

- The other three inputs to multiplexer come from
- The address field of the present microinstruction
- From the out of SBR
- From an external source that maps the instruction
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- If the bit selected is equal to 1, the T variable is equal to 1; otherwise, it is equal to 0.
- The *T value together with two bits from the BR (branch) field goes to an input logic circuit.*

- The input logic in a particular sequencer will determine t
- The input logic circuit in above figure has three inputs I0, I1, and T, and three outputs, S0, S1, and L.
- Variables S0 and S1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the selection variables determine the path in the multiplexer.
- For example, with S1,S0 = 10, multiplexer input number 2 is selected and establishes transfer path from SBR to CAR.
- The truth table for the input logic circuit is shown in Table b.
- The type of operations that are available in the unit.

MICROPROGRAM SEQUENCER

- Inputs I1 and I0 are identical to the bit values in the BR field.

- The bit values for S1 and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.

- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).

NUMERICALS ON MICROPROGRAMMED CONTROL UNIT

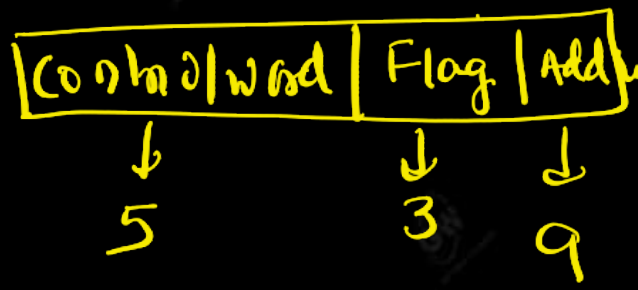
Q1 A vertical micro programmed control unit support 512 instructions . The system is using 8 conditional flag and contain 31 control signal .Each instruction on an average has 1 micro operation .calculate the approximate size of memory in bytes(AKTU 2021-22)

No of instruction = 512
& Conditional flag
31 control signal
Each instruction has 1 uop

Number of bit needed to Represent
& condition Flag = 2^3 = 3bit
total uop = 512 * 1 = 512 uop
↓
9 bit ← 2^9

31 control Signal
need bit = 5 bit

No of bit to
Represent 1 uop
1 micro in
5 + 3 + 9 = 17bit



No of bit needed
to Represent 29 uoper.
2^9 * 17 bit

$\frac{2^9 * 17}{8}$ = 1088 byte

PIPELINE FORMULAS and CONCEPTS

Performance of Pipelined Execution-

The following parameters serve as criterion to estimate the performance of pipelined execution-

Speed Up

Efficiency

Throughput

$$\text{Speed Up (S)} = \frac{\text{Non-pipelined execution time}}{\text{Pipelined execution time}}$$

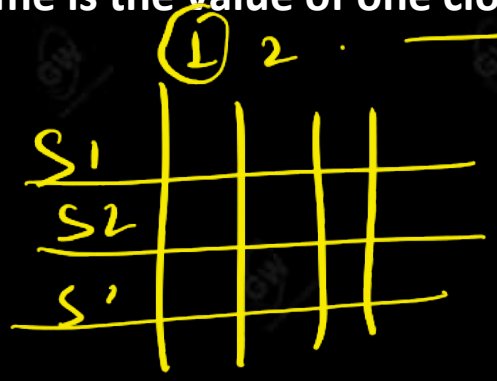
$$\text{Efficiency } (\eta) = \frac{\text{Speed Up}}{\text{Number of stages in Pipelined Architecture}}$$

$$\text{Throughput} = \frac{\text{Number of instructions executed}}{\text{Total time taken}}$$

PIPELINE FORMULAS and CONCEPTS

Point-01: Calculating Cycle Time-

- In pipelined architecture,
- There is a global clock that synchronizes the working of all the stages.
- Frequency of the clock is set such that all the stages are synchronized.
- At the beginning of each clock cycle, each stage reads the data from its register and process it.
- Cycle time is the value of one clock cycle.



- . There are two cases possible-

Case-01: All the stages offer same delay-

If all the stages offer same delay, then-

Cycle time = Delay offered by one stage including the delay due to its register

Case-02: All the stages do not offer same delay-

If all the stages do not offer same delay, then-

Cycle time = Maximum delay offered by any stage including the delay due to its register

PIPELINE FORMULAS and CONCEPTS

Point-02: Calculating Frequency Of Clock-

➢Frequency of the clock (f) = 1 / Cycle time

Point-03: Calculating Non-Pipelined Execution Time-

➢In non-pipelined architecture,

➢The instructions execute one after the other.

➢The execution of a new instruction begins only after the previous instruction has executed completely.

➢So, number of clock cycles taken by each instruction = k clock cycles

➢Non-pipelined execution time

➢= Total number of instructions x Time taken to execute one instruction

➢= n x k clock cycles

Point-04: Calculating Pipelined Execution Time-

➢In pipelined architecture, Multiple instructions execute parallelY.

➢Number of clock cycles taken by the first instruction = k clock cycles

➢After first instruction has completely executed, one instruction comes out per clock cycle.

➢So, number of clock cycles taken by each remaining instruction = 1 clock cycle

➢Thus, Pipelined execution time

➢= Time taken to execute first instruction + Time taken to execute remaining instructions

➢= 1 x k clock cycles + (n-1) x 1 clock cycle

➢= (k + n – 1) clock cycles

PIPELINE FORMULAS and CONCEPTS

Point-04: Calculating Speed Up-

Speed up

= Non-pipelined execution time / Pipelined execution time

= n x k clock cycles / (k + n – 1) clock cycles

= n x k / (k + n – 1)

= n x k / n + (k – 1)

= k / { 1 + (k – 1)/n }

For very large number of instructions, n→∞. Thus, speed up = k.

Practically, total number of instructions never tend to infinity.

Therefore, speed up is always less than number of stages in pipeline

Numerical on pipeline

Register

Problem-01:

Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 ns. Given latch delay is 10 ns. Calculate-

- Pipeline cycle time
- Non-pipeline execution time
- Speed up ratio
- Pipeline time for 1000 tasks
- Sequential time for 1000 tasks
- Throughput

(1) Pipeline cycle time = Max of any stage + Register delay

Max {60, 50, 90, 80} + 10

90 + 10 = 100 ns

(2) Non-pipeline cycle time = time to go through all stage

60 + 50 + 90 + 80 = 280 ns

Numerical on pipeline

$$\text{speed up} = \frac{\text{Non-pipeline Execution time}}{\text{pipeline Execution time}}$$

$$\frac{280\,ns}{100\,ns} = 2.8$$

pipe time for 1000 task = $(k + n - 1)\,TP$

$(4 + 1000 - 1) * 100$

$1003 * 100 = 100300$

Numerical on pipeline

Non-pipeline Execution = One enstruction Execution time * Numbere of enstruction

280 * 1000

280000ns

throughput = $\frac{\text{No of enstruction}}{\text{Execution time for Executing three ins}} = \frac{1000}{100300}$

Numerical on pipeline

Numerical on pipeline

Problem no.8 (AKTU) 2021-22 / 22-23

There is an instruction pipeline with four stages. The stage delay for each stage is 5nsec , 6nsec, 11 nsec, and 8 nsec respectively .consider the delay of inter stage register in the pipeline is 5 nsec. Determine the approximate speedup of the pipeline in the steady state under ideal condition as compared to the coressponding non-pipelined implementation

Pipeline Execution Cycle time: $\max(5, 6, 11, 8) + 5 = 11 + 5$

16 ns

Non-pipe = Delay time all stage $5 + 6 + 11 + 8 = 30$

$$\text{Speed up} = \frac{\text{Non-pipeline Execution}}{\text{pipeline execution}} = \frac{30}{16} = 1.8$$

Numerical on pipeline

Instruction Format

➢Instruction formats refer to the way instructions are encoded and represented in machine language. There are several types of instruction formats, including zero, one, two, and three-address instructions.

➢An opcode is a collection of bits that represents the basic operations including add, subtract, multiply, complement, and shift.

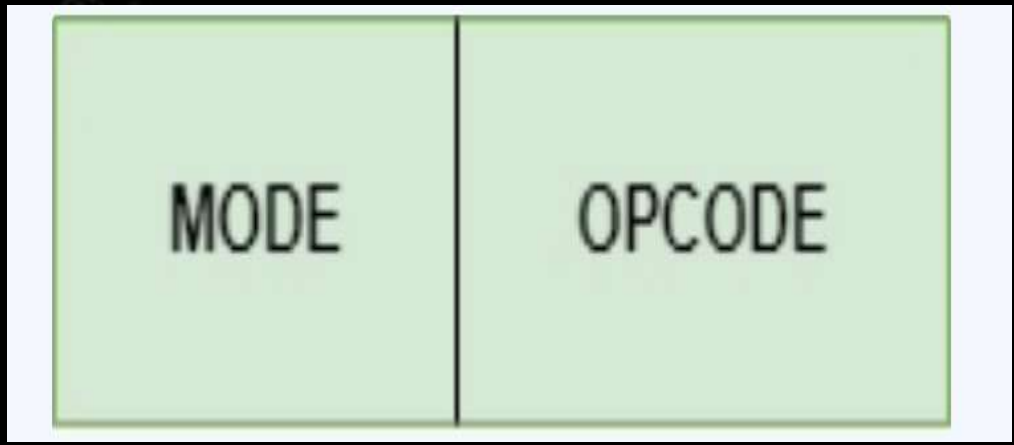
➢Operands are definite elements of computer instruction that show what information is to be operated on

a+b

ab+

1. Zero address instruction

➢This instruction does not have an operand field, and the location of operands is implicitly represented. The stack-organized computer system supports these instructions. To evaluate the arithmetic expression, it is required to convert it into reverse polish notation.



Instruction Format

Example of zero address

TOS: Top of the Stack Expression: X = (A+B)*(C+D) Post fix :

X = AB+CD+*

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← (A + B)
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← (C + D)
MUL	TOS ← (C + D) ∗ (A + B)
POP X	M [X] ← TOS

Zero-address instructions:

Advantages:

They are simple and can be executed quickly since they do not require any operand fetching or addressing. They also take up less memory space.

Disadvantages:

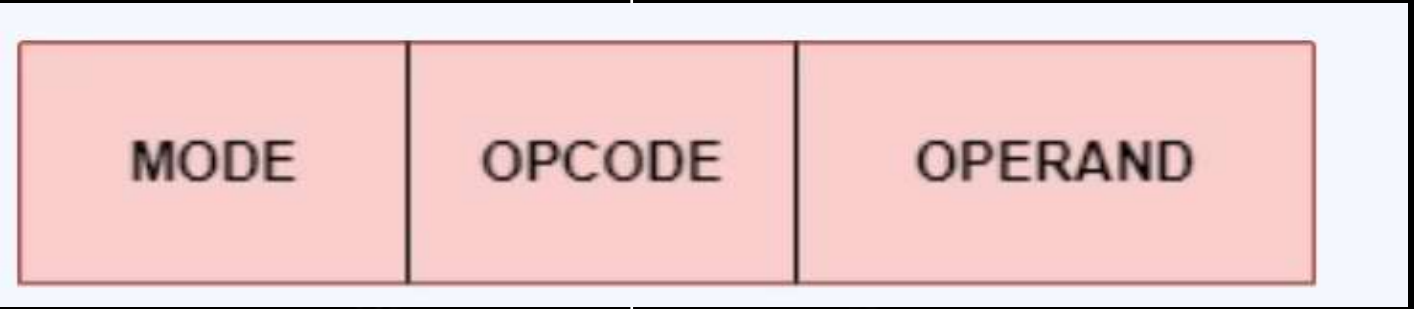
They can be limited in their functionality and do not allow for much flexibility in terms of addressing modes or operand types

Instruction Format

One address instruction

- This instruction uses an implied accumulator for data manipulation operations.
- An accumulator is a register used by the CPU to perform logical operations.
- In one address instruction, the accumulator is implied, and hence, it does not require an explicit reference.

For multiplication and division, there is a need for a second register. However, here we will neglect the second register and assume that the accumulator contains the result of all the operations



Instruction Format

One address instruction

➢Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

➢LOAD A	AC ← M [A]
➢ ADD B	AC ← A C+ 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

One-address instructions:

Advantages:

They allow for a wide range of addressing modes, making them more flexible than zero-address instructions. They also require less memory space than two or three-address instructions.

Disadvantages:

They can be slower to execute since they require operand fetching and addressing

Instruction Format

Two address instruction

➢These instructions specify two operand or address, which typically refers to a memory location or register. The instruction operates on the contents of that operand, and the result may be stored in the same or a different location

Mode	opcode	Destination address	Source address

Two address instruction

Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

MOV R1, A	R1 ← M [A]
ADD R1, B	R1 ← R1 + M [B]
MOV R2, C	R2 ← M [C]
ADD R2, D	R2 ← R2 + M [D]
MUL R1, R2	R1 ← R1∗R2
MOV X, R1	M [X] ← R1.

The MOV instruction transfers the operands to the memory from the processor registers. R1, R2 registers.

Transfer data from Reg to Memory loc

Instruction Format

Two-address instructions:

Advantages:

They allow for more complex operations and can be more efficient than one-address instructions since they allow for two operands to be processed in a single instruction. They also allow for a wide range of addressing modes.

Disadvantages:

They require more memory space than one-address instructions and can be slower to execute since they require operand fetching and addressing

Three address instruction

These instructions specify three operands or addresses, which may be memory locations or registers. The instruction operates on the contents of all three operands, and the result may be stored in the same or a different location

Mode	opcode	Destination address	Source address	Source address

Three address instruction

$X = (A+B) * (C+D)$

R1 R2

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

Three address instruction

Advantages:

- They allow for even more complex operations and can be more efficient than two-address instructions since they allow for three operands to be processed in a single instruction. They also allow for a wide range of addressing modes.

Disadvantages:

- They require even more memory space than two-address instructions and can be slower to execute since they require operand fetching and addressing.

- Overall, the choice of instruction format depends on the specific requirements of the computer architecture and the trade-offs between code size, execution time, and flexibility

Ins" format

④ Hardwire & Microprog + block

① - Microprogram sequencer + ins"
② Instruction cycle
③ RISC & CISC

GATEWAY CLASSES

GATEWAY CLASSES

AKTU Full Courses (Paid)

Download Gateway Classes Application

From Google Play store

All Subjects

Link in Description

Thank You