AKTU

B. Tech II-Year

Gateway Classes

Semester -III

CS IT & CS Allied Branches

BCS302- COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT-3 Control Unit

Hand Written Notes

Gateway Series for Engineering

- Topic Wise Entire Syllabus
- Long - Short Questions Covered
- AKTU PYQs Covered
- DPP
- Result Oriented Content

Download App

For Full Courses including Video Lectures

AKTU

B. Tech II-Year

Gateway Classes

BCS302- COMPUTER ORGANIZATION AND ARCHITECTURE

Hand Written Notes

Unit-3

Control Unit

Syallbus

Control Unit: Instruction types, formats, instruction cycles and sub cycles (fetch and execute etc), micro operations, execution of a complete instruction. Program Control, Reduced Instruction Set Computer, Pipelining. Hardwire and micro programmed control: micro programme sequencing, concept of horizontal and vertical microprogramming.

Download App

For Full Courses including Video Lectures

COMPUTER ORGANISATION AND ARCHITECTURE (BCS 302)

UNIT - 3 CONTROL UNIT

Instruction Cycle

- 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 thro-ugh to execute a single instruction
- Memory address registers (MAR): It is connected to the address lines of the system bus. It specifies the address in memory for a read or right operation.
- Memory Buffer Register (MBR): It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- Program Counter (PC): Holds the address of the next instruction to be fetched.
- Instruction Register (IR): Holds the last instruction fetched.

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 identify-ing the opcode and any operands that are needed to execute the instruction.

Execute: In the execute cycle, the CPU perform-s 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.

- There are also some additional steps that may be performed during the instruction cycle, depending on the CPU architecture and instru-ction set:

- Fetch operands: In some CPU's, the operands needed for an instruction are fetched during a separate

cycle before the execute cycle. This is called the fetch operands cycle.

- **Store results:** In some CPUs, the results of an instruction are stored during a separate cycle after the execute cycle. This is called the store results cycle.

Fetch — IR ← Memory location (1010) Next memory location where next instruction (which is executed to be next) is PC → present (1110)

Decode — Interprets Instruction — opcode / operand

Execute — instruction → Arithmetic, logical etc. / Read / write / control flow

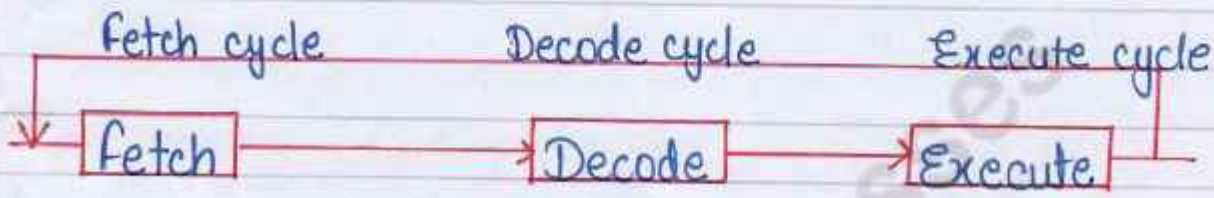
★ Instruction Cycle (Phases)

- **Interrupt handling:** In some CPUs, interrupt handling may occur during any cycle of the instruction cycle. An interrupt is a signal that the CPU recieves from an external device or software that requires immediate attention. When an interrupt occurs, the CPU suspends the current instruction and executes an interrupt handler to service the interrupt.

Gateway Classes

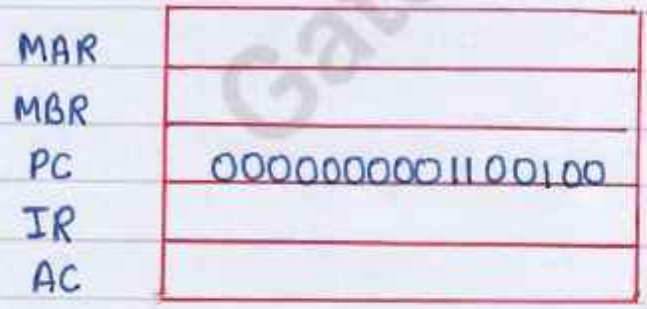
- These cycles are the basic building blocks of the CPU's operation and are performed for every instruction executed by the CPU. By optimizing these cycles, CPU designers can improve the performance and efficiency of the CPU, allowing it to execute instructions faster and more

Block Diagram



The Fetch Cycle – Sequence counter value initialize to 0

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the Program counter (PC).



Beginning

Step 1: The address in the program counter is moved to the memory address register (MAR),

as this is the only register which is connected to address lines of the system bus.

MAR ← PC

MAR	0000000000 11 00100
MBR	
PC	0000000000 11 00100
IR	
AC	

First Step.

Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register (MBR). (These two actions can be performed simultaneously to save time.).

- MBR ← MAR

MAR	0000000000 11 00100
MBR	0001000000 1 00000
PC	0000000000 1 1 00101
IR	
AC	

Second Step.

Step 3 : The content of the MBR is moved to the Instruction register.

MAR	0000000000110010 0
MBR	0001000000100000
PC	0000000000110010 0
IR	0001000000100000
AC	

- This all done in , t1 and PC ← PC + 1 Program counter is incremented by one, to get ready for the next instruction

Decode Cycle

- Decode Cycle (T2)
- The Instruction Register (IR) is a 16-bit register capable of storing instructions in 16-bit format. The three components of the 16-bit instruction format (0 to 11 bit) are:
  - Addressing Mode (15th Bit)
  - OPCODE (12,13,14 bit)
  - Operand Address (0 to 11)
- The CPU's control unit decodes the program instructio -n once it is loaded into the instruction

register (IR). The Decode Phase is the second phase of the instruction cycle.

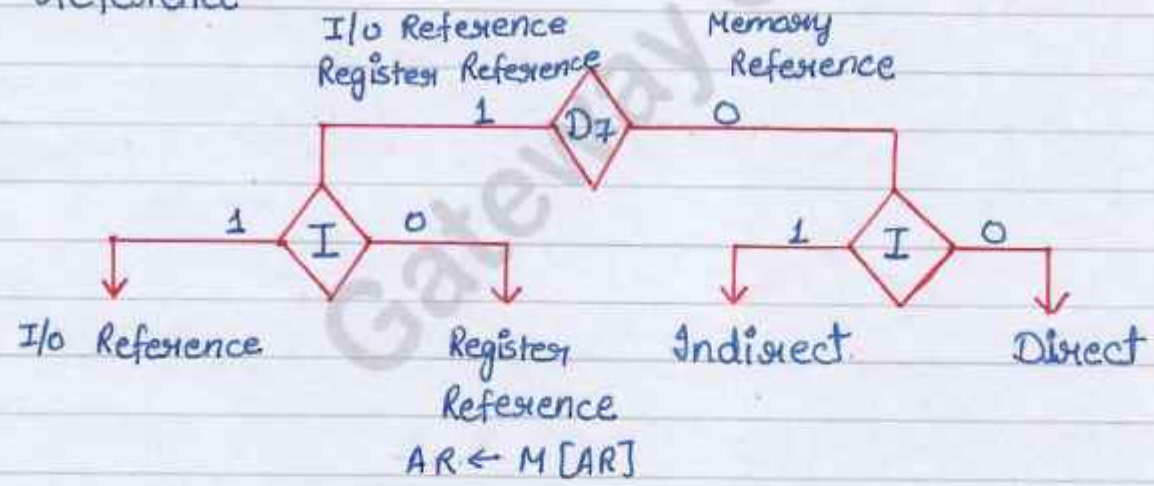
- The instruction is decoded by the CPUs control unit based on the operation code (OPCODE), determ -ined by three bits. (bit 12, 13, 14).
- Decode Instruction Type occuring at Clock Pulse (T-3).
- The CPU's control unit must first decode the type of instruction. The three types of education are:
  - Memory reference instruction
  - Register reference instruction
  - Input / output instruction
- The 3x8 decoder determines the type of instruct -ion. The D7 value determines the type of instruction. The three-bit OPCODE can have distin -ct values, starting with (D0, D1, D2, ..., D6, D7). Alternatively, the OPCODE can have values ranging from D0 - 000 to D7 111.
- If D7 is set to zero, the instruction type is a memory reference. When D7 is set to 1, the instruction type can be a register refe -rence type or an input/output instruction. The instruction type is a register reference type if D7 equals one and I equals 0. If D7 and I are both 1, the instruction

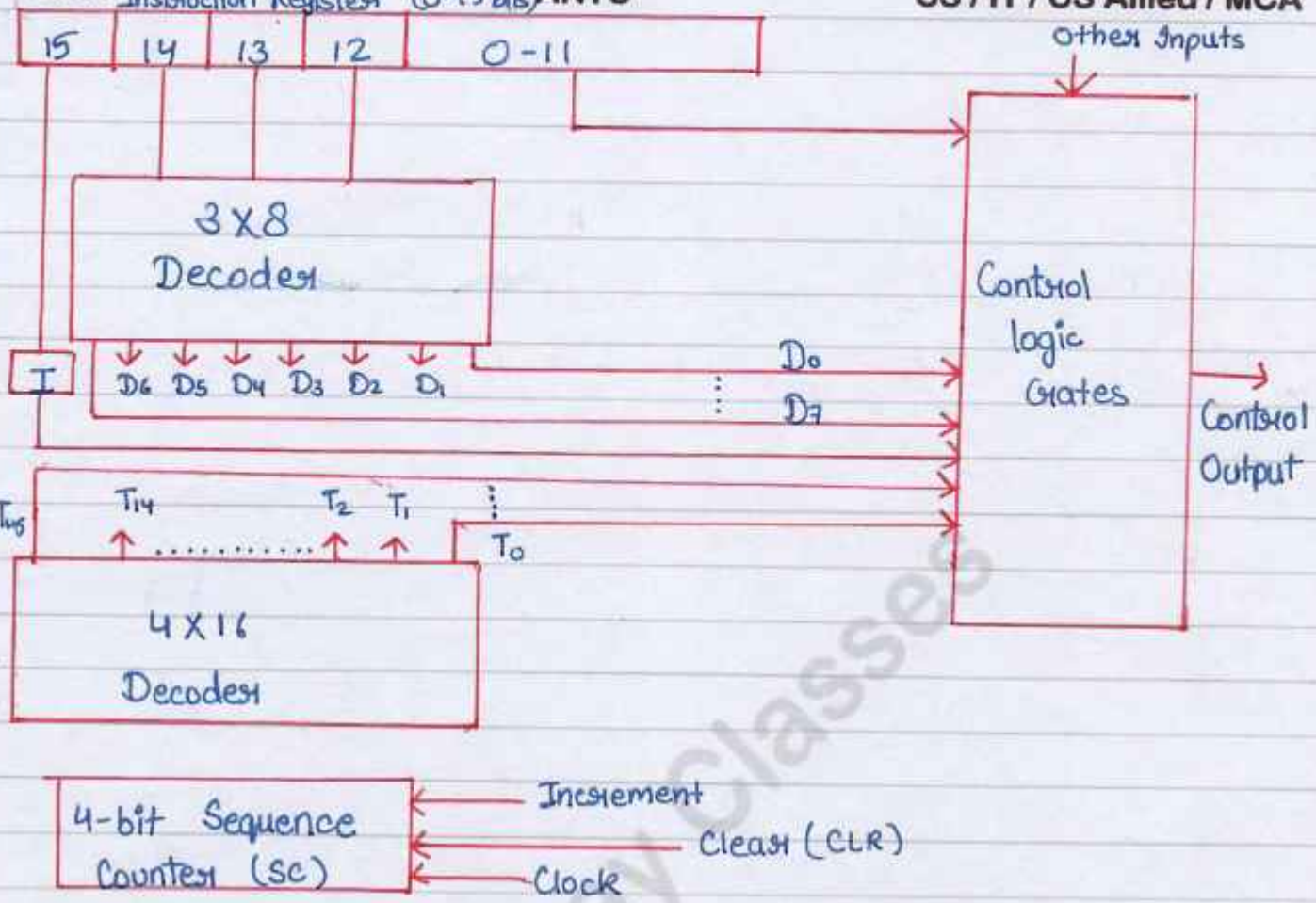
type is input / output.

- After deciding on the type of instruction and making a subsequent decision, the decode phase concludes.
The value of 9 in the 15th bit determines the decode phase.

- If D==1 then if I==1 then input/output reference and If D==1 and I==0 the register reference.

D==0 the if I==1 then Indirect memory reference and if d==0 and if==0 then direct memory reference



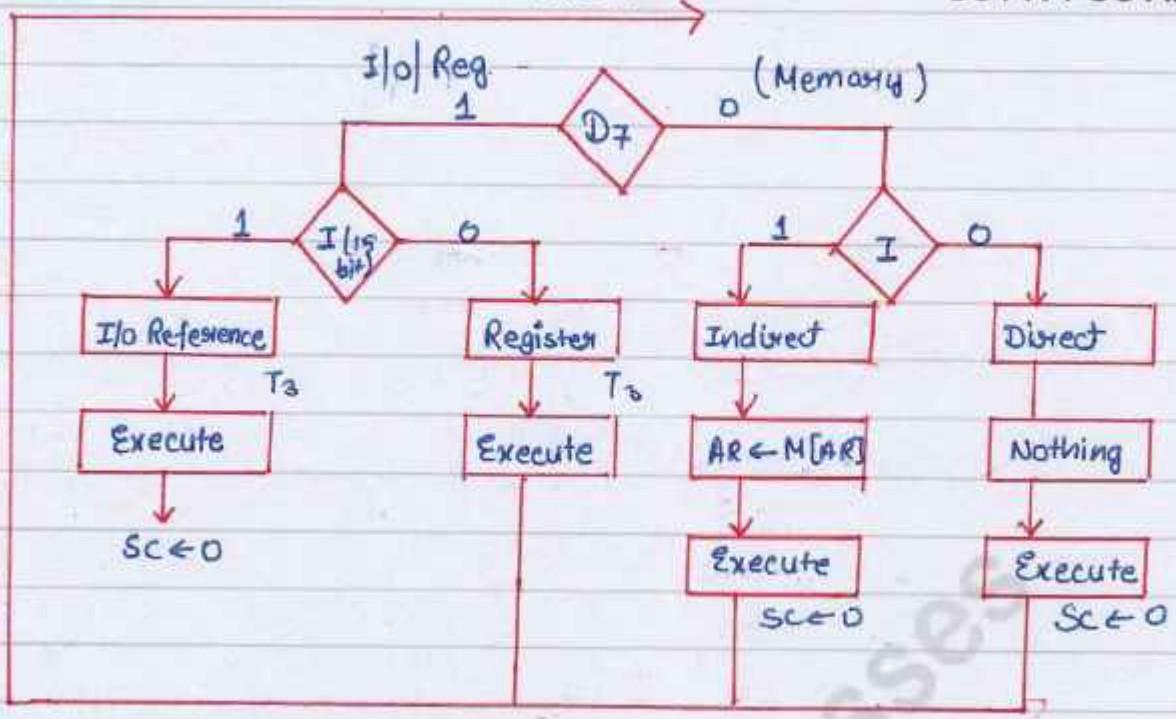


Execute Cycle (t3)

- D7 == 1 then if i == 0 then register reference
- EXECUTE THE INSTRUCTION T3 AND SC=0.

- and if D == 1 and i == 1 the input/output refrence
- EXECUTE THE INSTRUCTION T3 AND SC=0.

- D==0 the if I == 1 then indirect memory reference AR ← M[AR] EXECUTE INSTRUCTION SC=0.
- and if d==0 and if i == 0 then direct memory reference EXECUTE INSTRUCTION SC=0



AKTU Question

Q1. Explain different cycle/phases of instruction cycle.

OR

Define instruction cycle

AKTU 2019-20, 2018-19
2022-23.

Instruction

- 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 :

Gateway Classes

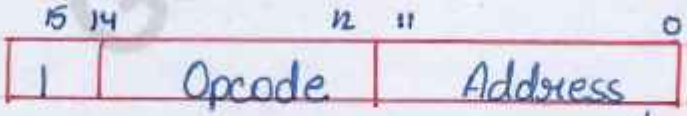
- The operation code (OPCODE) field which specifies the operation to be performed.
- The address field which contains the location of the operand i.e., register or memory location.
- The Mode field which specifies how the operand will be located.

MODE	OPCODE	OPERAND / address of OPERAND

- A basic computer has three instruction code formats which are:

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

Memory - reference Instruction

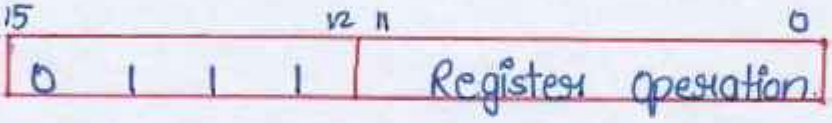


(Opcode = 000 through 110)

Register - reference Instruction

- The Register - reference instructions are represented by the Opcode 111 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.



(opcode = 111, I = 0)

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.

(Opcode = 111, I = 1)

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.

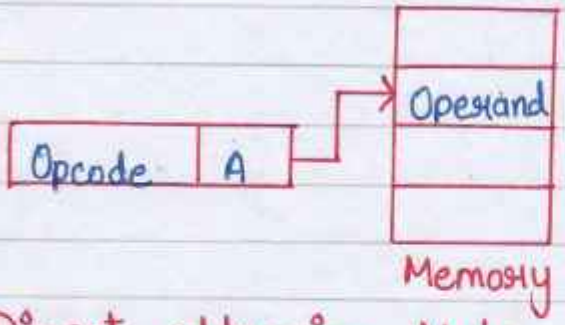
Addressing MODE

- The different ways of specifying the location of an operand in an instruction are called as addressing modes.

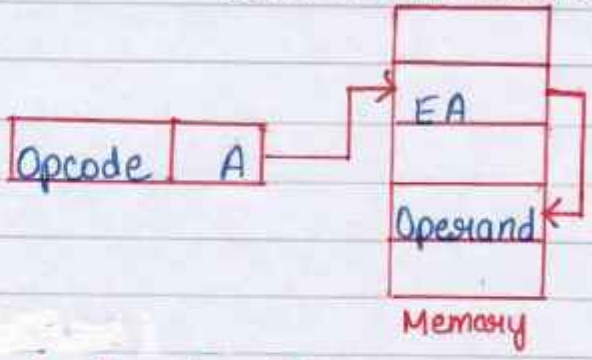
- Implied / Implicit Addressing Mode
- Stack Addressing Mode
- Immediate Addressing Mode
- Direct Addressing Mode
- InDirect Addressing Mode
- Register Direct Addressing Mode
- Register InDirect Addressing Mode
- Relative Addressing Mode
- Indexed Addressing Mode
- Base Register Addressing Mode
- Auto-Increment Addressing Mode
- Auto-Decrement Addressing Mode

Gateway Classes

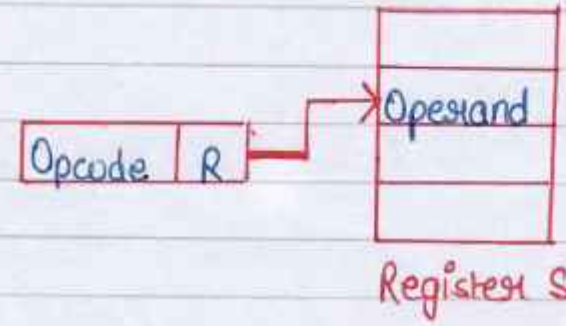
COA AKTU CS / IT / CS Allied / MCA



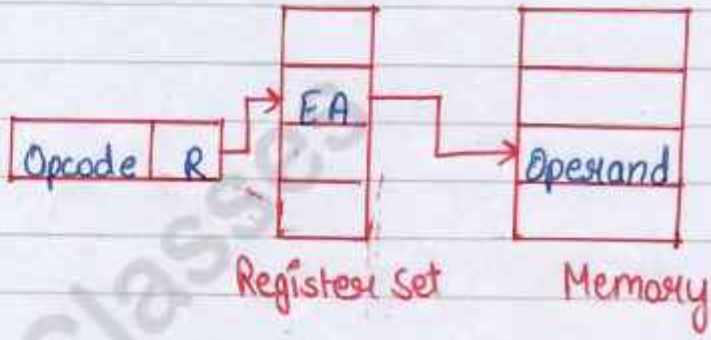
Direct addressing Mode



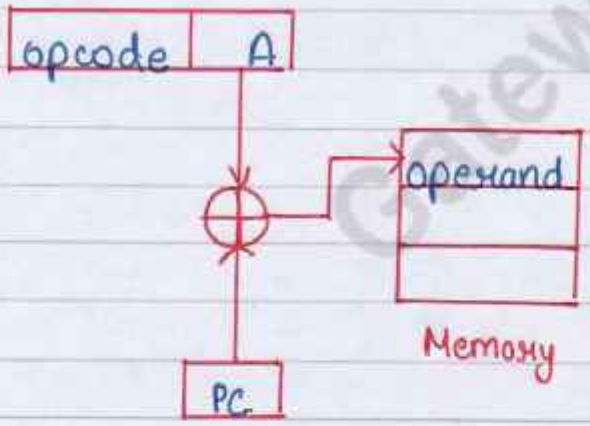
Indirect addressing Mode



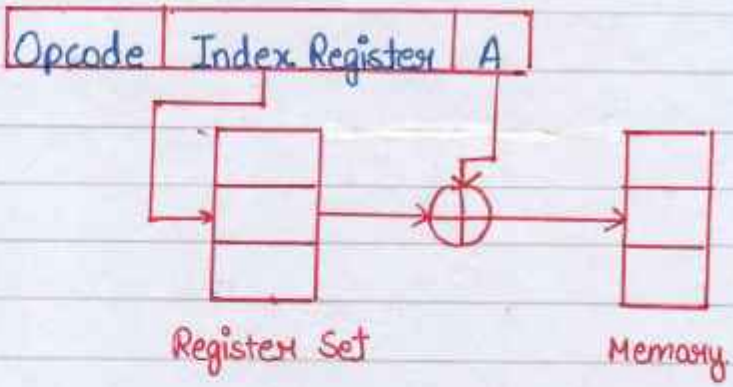
Register Direct addressing Mode



Register Indirect addressing Mode



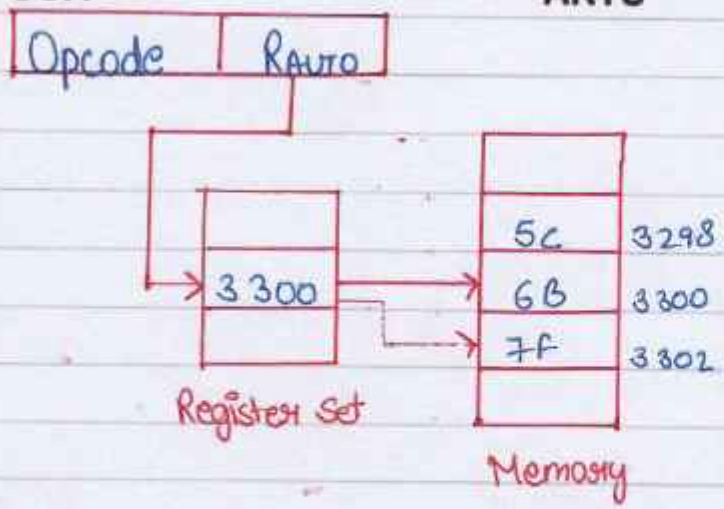
Relative addressing mode

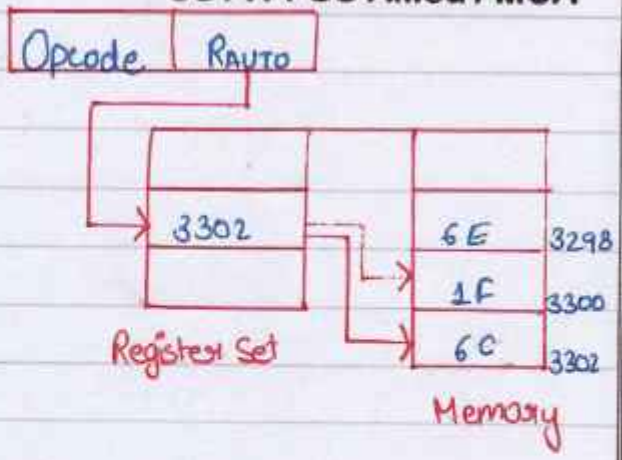


Indexed addressing mode

Gateway Classes

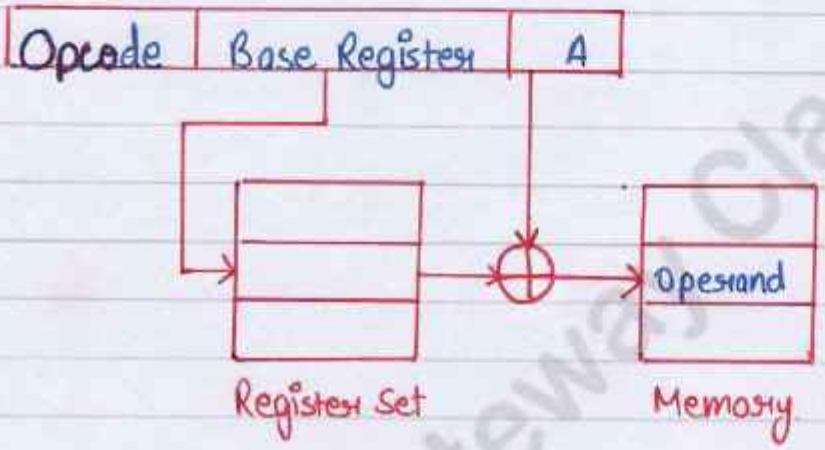
COA AKTU CS / IT / CS Allied / MCA





Auto Increment addressing mode

Auto decrement addressing mode



Base register addressing mode

Implied addressing mode

example - Complement accumulator

Immediate addressing mode

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

Difference Between :-

Parameters	Direct Addressing mode	Indirect addressing mode
Address field	Address field contains the effective address of operand	Address field contains reference of effective address.
Memory References	Requires only one memory reference	Requires two memory references.
Processing speed	The addressing mode has fast addressing compared to indirect addressing mode.	It is slower than direct addressing mode.
Classification.	No further classification	Further classified into two categories - Memory Indirect and Register Indirect Addressing Mode.
Calculation	No further calculation is required to perform the operation.	Requires further calculation to find the effective address.
Address space	It occupies a smaller amount of space than the indirect mode.	It occupies the large amount of space than the direct mode.

Gateway Classes

COA | AKTU | CS / IT / CS Allied / MCA

Overhead	No additional overhead is involved while searching for operand	Additional overhead is involved while search -ing for operand.
Advantage	Easy as no intermediary is involved	Availability of large address space.

AKTU Question

Q1 What is instruction in terms of computer organi zation explain the purpose of various element of an instruction with the help of sample instr -uction format.

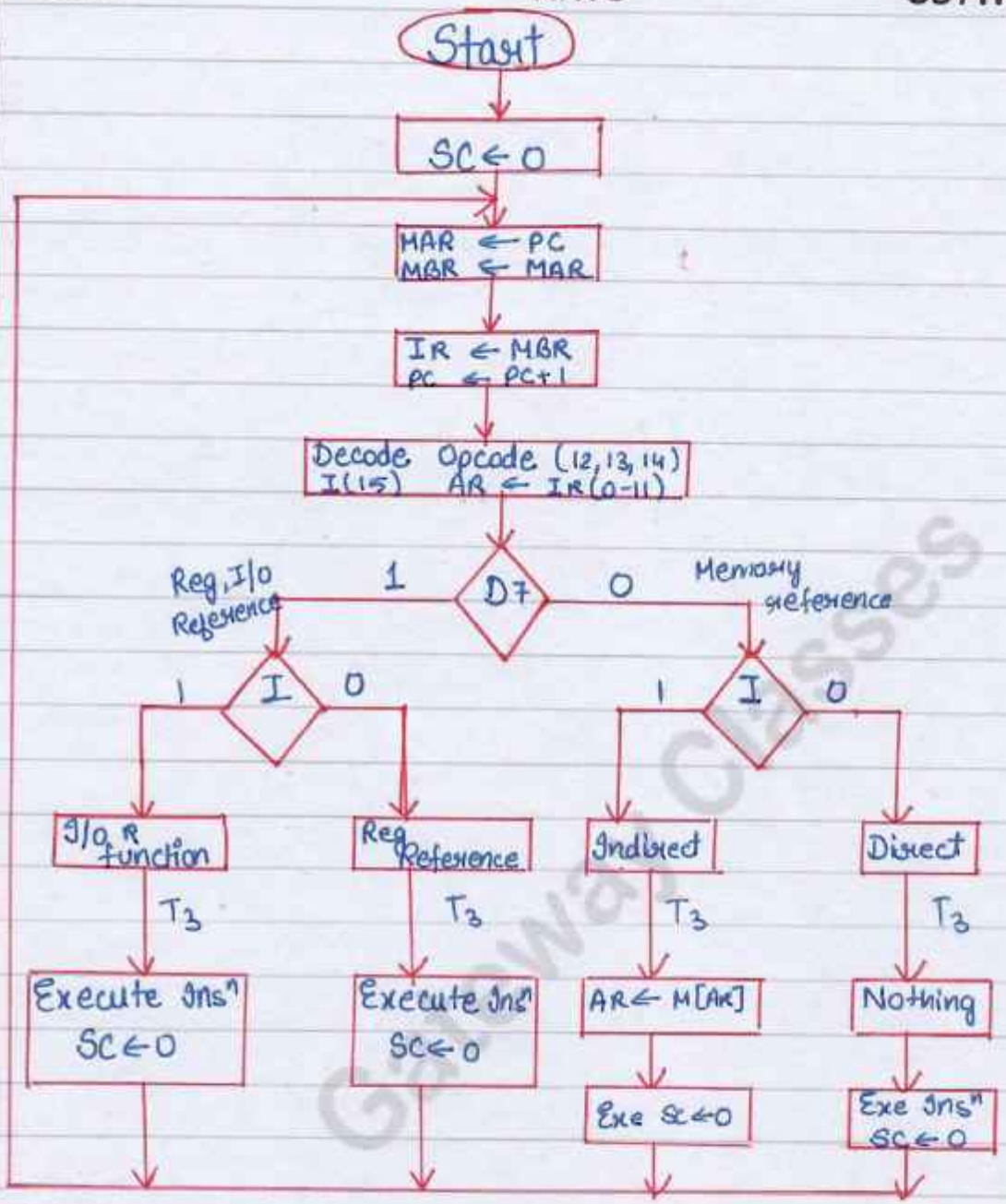
AKTU 2014-15, 2016-17.

Q2 Draw a flowchart to explain the instruction cycle in detail with diagram.

Fetch → Getting the Instruction from memory
Decode → what instruction want (ADD, SUB) + operand
Execute

Direct Indirect

Ins^n → Instruction
I/O → Input/output Reference
Reg → Register Reference
PC → Program Counter
MAR → Memory Address Register
MBR → Memory Buffer Register
IR → Instruction Register



How to fetch word from Memory

- To fetch instruction / data from memory, processor transfers required address to MAR (whose output is connected to address-lines of memory-bus). At the same time, process

-or issues Read signal on control-lines of memory-bus.

- When requested-data are received from memory they are stored in MDR.
- From MDR, they are transferred to other Registers.
- MFC (Memory function Completed): Addressed-device sets MFC to 1 to indicate that the contents of the specified location
- → have been read & are available on data-lines of memory-bus.
- Consider the instruction Move $(R_1), R_2$. (R_1) REPRESENT Indirect addressing mode R1 holda address
- Steps
- MAR ← $[R_1]$ and Read operation performed.
- loaded into MDR and wait for MFC Response from memory
- R_2 ← MDR
- Consider the instruction Move $(R_1), R_2$. (R_1) Represent

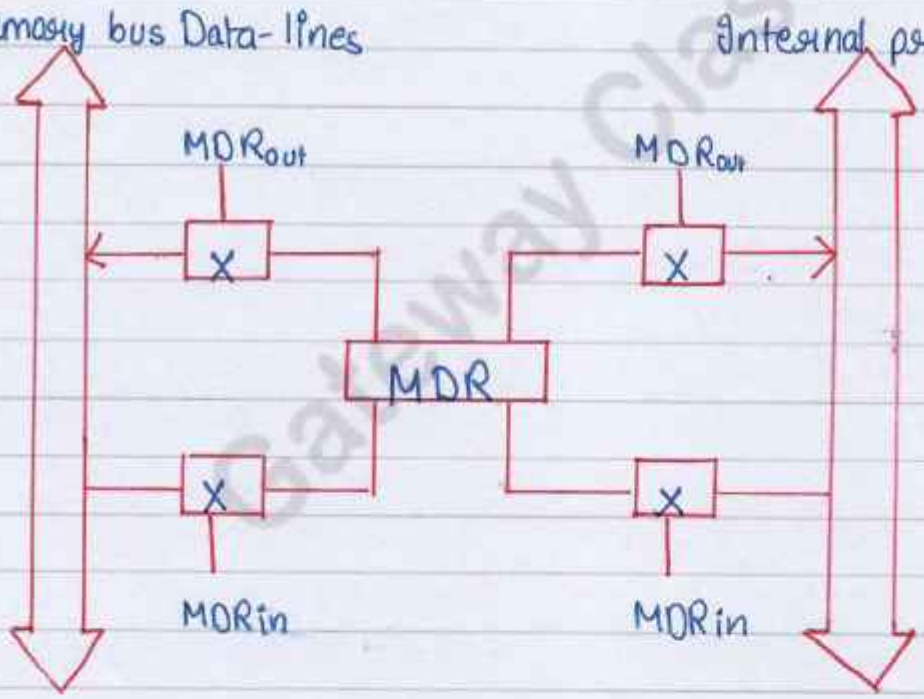
indirect addressing mode R1 holda address

- The sequence of steps is:

1) R1out, MARin, Read ; # desired address is loaded into MAR & Read command is issued

2) MDRinE, WMFC ; # load MDR from memory bus & wait for MFC response from memory

MDRout, R2in ; # load R2 from MDR.



Branch Instruction

- Branching instructions refer to the act of switching execution to a different instruction sequence as a

result of executing a branch execution.

The three types of branching instructions are:

- Jump (unconditional and conditional)
- Call (unconditional and conditional)
- Return (unconditional and conditional)

⇒ Jump Instructions - The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and conditional Jump Instructions.

- Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

Opcode	Operand	Explanation	Example
JMP	address	Jumps to the address	JMP 2050

- Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

Opcode	Operand	Explanation	Explanation
JC	address	Jumps to the address if carry flags is 1	JC 2050
JNC	address	Jumps to the add -ress if carry flag is 0	JNC 2050

⇒ Call Instructions - The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call instructions and conditional Call Instructions.

- Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.

Opcode	Operand	Explanation	Example
CALL	address	Unconditionally calls	CALL 2050

• Conditional Call instructions: Only if the condition is satisfied, the instructions executes.

Opcode	Operand	Explanation	Example
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050

⇒ Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

a) Unconditional Return Instructions: The program sequence is transferred conditionally from the subroutine to the calling program.

Opcode	Operand	Explanation	Example
RET	none	Return from the subroutine uncon-ditionally	RET

b) Conditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

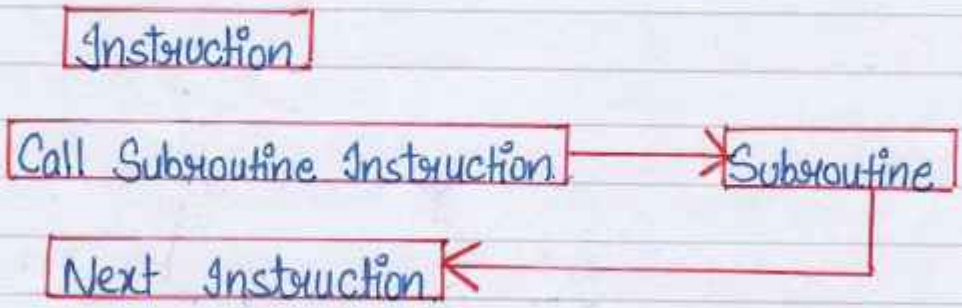
Opcode	Operand	Explanation	Example
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC

Subroutine Instruction

- A set of instructions that are used repeatedly in a program can be referred to as a subroutine. Only one copy of this instruction is stored in the memory. When a subroutine is required it can be called many times during the Execution of a particular program. A call subroutine instruction calls the subroutine. Care should be taken while returning a subroutine as a Subroutine can be called from a different place from the memory.

Gateway Classes

- The content of the PC must be saved by the call subroutine instruction to make a correct return to the calling program.



Main Difference

Branch Instruction :

- The program counter is changed to the new location.
- The branch instruction is machine language or assembly level instruction.
- It is used to change the sequence of instruction.

Call Subroutine Instruction :

- The program counter is pushed onto the stack, and the program counter is then is changed to the first instruction of the subroutine.
- It is a control transfer instruction.
- It is used to call a subroutine.

Question

Assuming that all registers initially 0 what is the value of R_1 after the following instruction sequence is executed

- MOV R_1, #6 → MOV/Put the value 6 in R_1
- MOV R_2, #5 MOV/Put the value 5 in R_2.
- ADD R_3, R_1, R_1 $R_3 \leftarrow R_1 + R_1 \Rightarrow R_3 \leftarrow 6+6 = 12$
- SUB R_1, R_3, R_2 $R_1 \leftarrow R_3 - R_2 \Rightarrow R_1 \leftarrow 7$
- MUL R_3, R_1, R_1 $R_3 \leftarrow R_1 * R_1 \Rightarrow R_3 = 7*7 = 49$

Ans. $R_1 = 7$.

AKTU Questions

Q1. What are the steps of fetching word from memory.
AKTU 2014-15, 2019-20.

Q2. Difference between branch instruction and call subroutine instruction.
AKTU 2019-20

Micro Operation

- 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 under-lying 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.

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 incl-

ude 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.

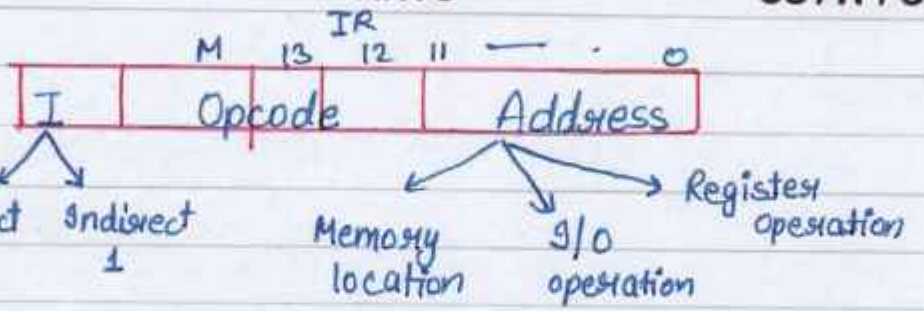
Transfer Micro operations: Transfer operations move data between different registers or memory locations within the CPU.

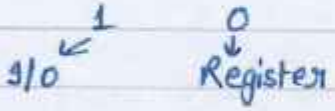
Question based on Control Unit

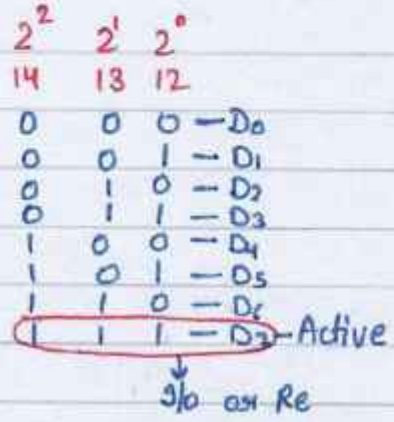
In an instruction format, there are 16 bits in an instruction word. Bit 0 to 11 convey the address of the memory location for memory related instructions. For non-memory instruction these bits convey various register or I/O operations. Bits 12 to 14 shows various basic memory operations such as ADD etc. Bit 15 shows if the memory is accessed directly or indirectly for such an instruction format draw the block diagram of the control unit of a computer and briefly explain how an instruction will be decoded and executed by the control unit.

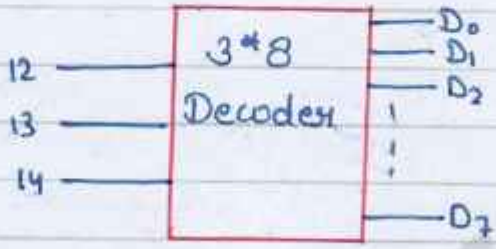
Gateway Classes

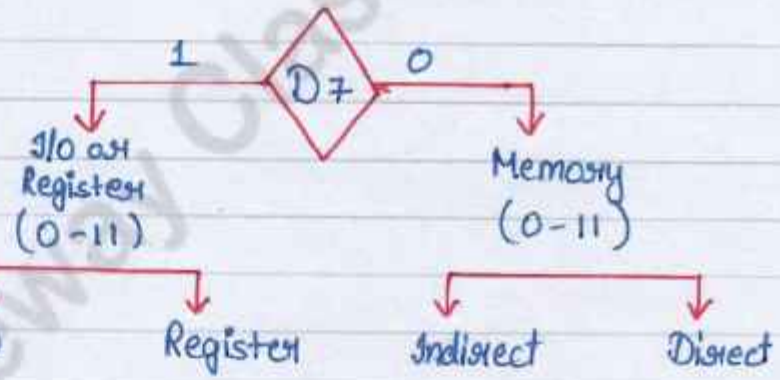
COA | AKTU | CS / IT / CS Allied / MCA

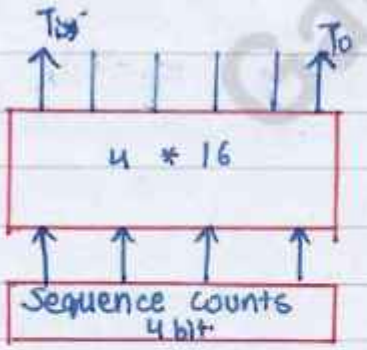


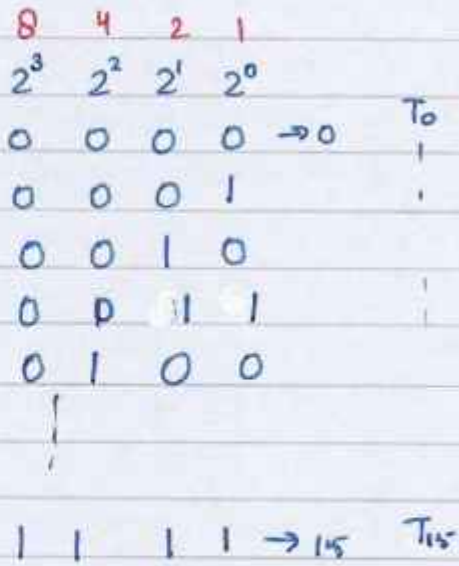


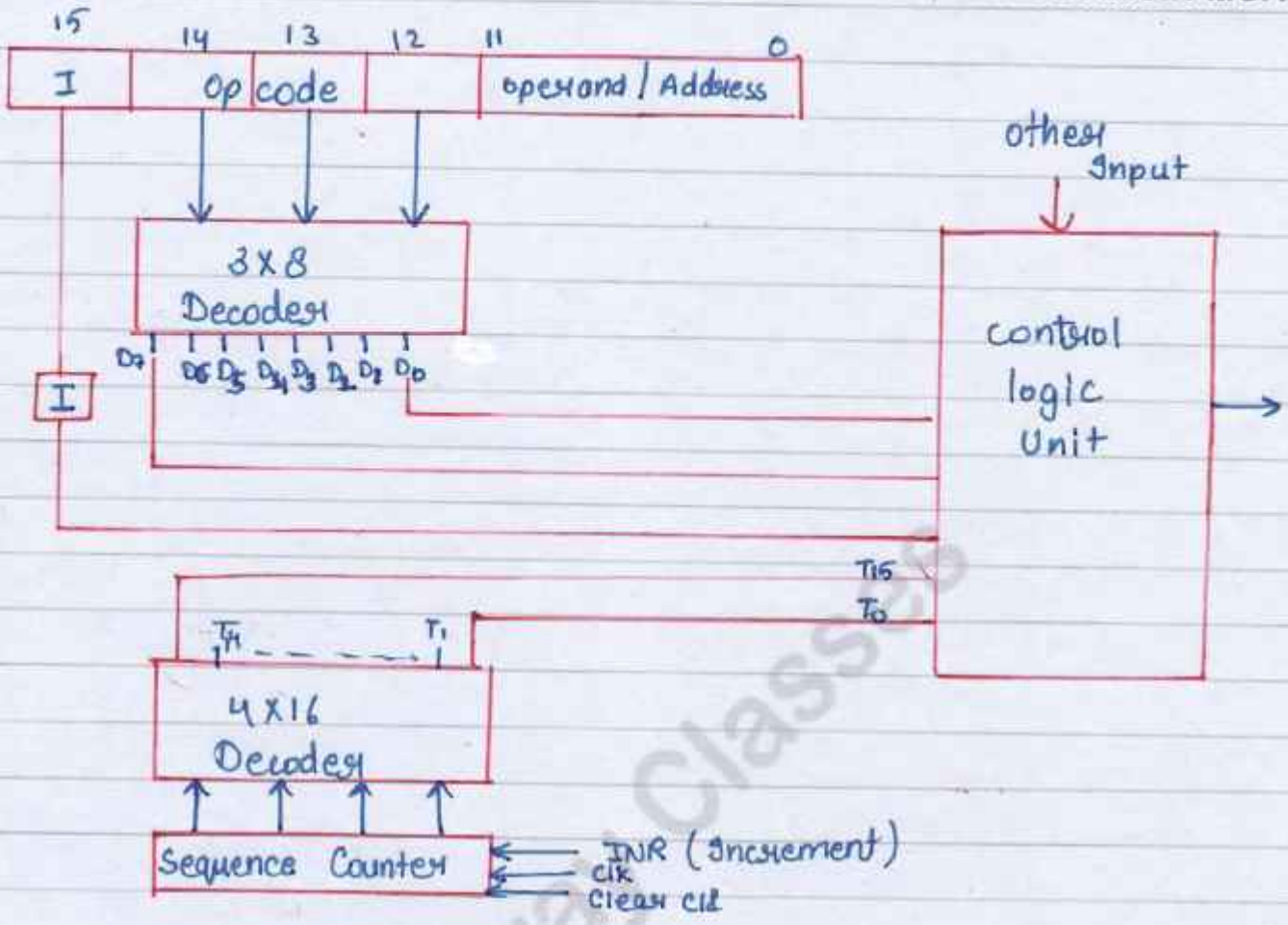












Control Unit

- The basic components of control units are.
- 2 decoders, one is 3X8, and the other is 4X16.
- One sequential counter
- One control logic gate
- and one Instruction Register.
- Above diagram shows that,

Gateway Classes

- It contains two decoders, One Instruction Register, one sequence counter, and control logic gates.
- Fetched instructions from the memory are place in the instruction register. (IR)
- An instruction register contains three fields, Operand (0-11 bits), Opcode (12-14 bits), and Mode (15 bit).
- Operand provides the operand or address of operand and directly connected with control logic gates.
- The operation code (opcode) bits are connect with 3 x 8 decoder. And outputs of the decoder are denoted by the symbols D0 through D7.
- Mode Bit 15 represents the addressing modes and directly connected with control logic gates.
- The sequence counter (SC) can count from 0 through 15 in binary and connected with 4 x 16 decoder.
- As, the 15th bit is 1. So, it represents indirect addressing mode. 12, 13, 14 bits of opcode are 001

which represent ADD operation..

ADD operation also denoted by D1.0 to 11 bits are 01000101011 which represents the address of a operand.

If this instruction executed at time T2. Then it can symbolically represent as D1 T2 and SC becomes zero.
This instruction means that apply ADD operation on address (01000101011) of memory.

AKTU Question

Q1 In an instruction format, there are 16 bits in an instruction word. Bit 0 to 11 convey the address of the memory locati-on for memory related instructions. For non memory location for memory related instruction. For non memory instruction these bits convey various register or I/O operations. Bits 12 to 14 shows various basic memory operations such as ADD etc. Bit 15 shows if the memory is accessed directly or indirectly. For such an instruction format draw the block diagram of the control unit of a computer and briefly explain how an instruction will be decoded and execut-ed by the control unit.

Arithmetic Micro-operations

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

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-
- $R3 \leftarrow R1 + R2$
- Here R1 and R2 are the registers whose conte-nts we want to add and,
- R3 is the desired register for storing the outp-ut.

- Note : We can either store the output in another register or the same register, i.e. R1 or R2.

- For example, consider the value of register R1 as 1010 and the value of register as 0011. For performing 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).

- If we add R1 and R2, the output will be -

```
      1  <- carry
    1 0 1 0
  + 0 0 1 1
  ---------
    1 1 0 1
```

Subtraction :

- The subtract arithmetic micro operations subtracts the values of the two registers and stores the output in the desired register.

- The symbolic notation for the subtract arithmetic micro operation is -
  - $R_3 \leftarrow R_1 - R_2$

- Here R_1 and R_2 are the registers whose contents we want to subtract and,
- R_3 is the desired register for storing the output.
- Note: We can either store the output in another register or the same register, i.e. R_1 or R_2.
  $R_3 \leftarrow R_2 - R_1$ or $R_2 \leftarrow R_2 - R_1$ or $R_1 \leftarrow R_2 - R_1$
- For example, consider the value of register R1 as 1011 and register R2 as 0101. 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
- If we subtract R2 from R1, the output will be -

in decimal.

```
  0  2
  1  0  1  1
- 0  1  0  1
  0  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 -
- $R3 \leftarrow R1 + R2' + 1$

Here, R1 and R2 are the registers whose contents we want to subtract and,

R3 is the desired register for storing the output.

Using this method, we get the same output as R1 - R2.

Note : we can either store the output in another register or the same register, i.e. R1 or R2.

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 -

micro operation is - $R1 \leftarrow R1 - 1$

Here, R1 is the register whose value we want to decrement and,
R1 is also the desired register for storing the output.

Note: We can store the output in another register or the same register.

For example, consider the value of register R1 as 0101. For performing the decrement arithmetic micro-operation, we will subtract from R1.

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ -\quad\quad\ 1 \\ \hline 0\ 1\ 0\ 0 \end{array}$$

The decrement arithmetic micro-operation is carried out with the help of a combinational circuit or a binary up-down counter.

1's Complement

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

The symbolic notation for the 1's complement arithmetic micro-operation is - $R1 \leftarrow R1'$ or $R2 \leftarrow R1'$

$R1 \leftarrow R1 + 1$ or $R_2 \leftarrow R1 + 1$

Here, R_1 is the register whose value we want to increment and,

R_1 is also the desired register for storing the output.

Note: We can store the output in another register or the same register.

For example, consider the value of register R_1 as 0101. For performing the increment arithmetic micro-operation, we will add 1 to R1.

$$\begin{array}{cccc} & \overset{1}{} & \overset{\rightarrow \text{carry}}{} & \\ 0 & 1 & 0 & 1 \\ + & & & 1 \\ \hline 0 & 1 & 1 & 0 \end{array}$$

The increment arithmetic micro-operations is carried out with the help of a combinational circuit or a binary up-down counter.

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 decrement arithmetic

Here, R1 is the register whose value we want to complement and,
R1 is also the desired register for storing the output.

Note: We can store the output in another register or the same register.

For example, consider the value of register R1 as 0101. For performing the 1's complement arithmetic micro-operation, we will just convert 0 to 1 and 1 to 0.

Therefore, 1's complement of R1 will be 1010.

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 - $R2 \leftarrow R2' + 1$ or $R3 \leftarrow R2' + 1$

Here R2 is the register on whose value we want to perform 2's complement and,

R2 is also the desired register for storing the output.

Note: We can store the output in another register or

the same register.

For example, consider the value of register R2 as 0101. For performing the 2's complement arithmetic micro-operation first, we will find the 1's complement of R2. The 1's complement of R2 will be 1010. Then we will add 1 to it.

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ +\qquad 1 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

- The signals that implement these operations propagate through gates in this case, and the result of the process can be transferred into a destination register via a clock pulse immediately after the output signal propagates through the combinational circuit.
- Besides the above-described arithmetic micro-operations, there are two more arithmetic micro-operations— multiply and divide. These two operations are valid arithmetic operations, but they are not part of the required set of micro-operations. A series of add and shift micro-operations are used to perform the multiply micro-operation.

Gateway Classes

COA | AKTU | CS / IT / CS Allied / MCA

Symbolic Representation	Description
$R_3 \leftarrow R_1 + R_2$	The contents of R_1 plus R_2 are transferred to R_3.
$R_3 \leftarrow R_1 - R_2$	The contents of R_1 minus R_2 are transferred to R_3.
$R_2 \leftarrow R_2'$	Complement the contents of R_2 (1's complement).
$R_2 \leftarrow R_2' + 1$	2's complement the contents of R_2 (negate).
$R_3 \leftarrow R_1$	R_1 plus the 2's complement of R_2 (subtraction).
$R_1 \leftarrow R_1 + 1$	Increment the contents of R_1 by one.
$R_1 \leftarrow R_1 - 1$	Decrement the contents of R_1 by one.

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 consise manner for listing the micro-operation sequences in registers and the control functions that initiate them.

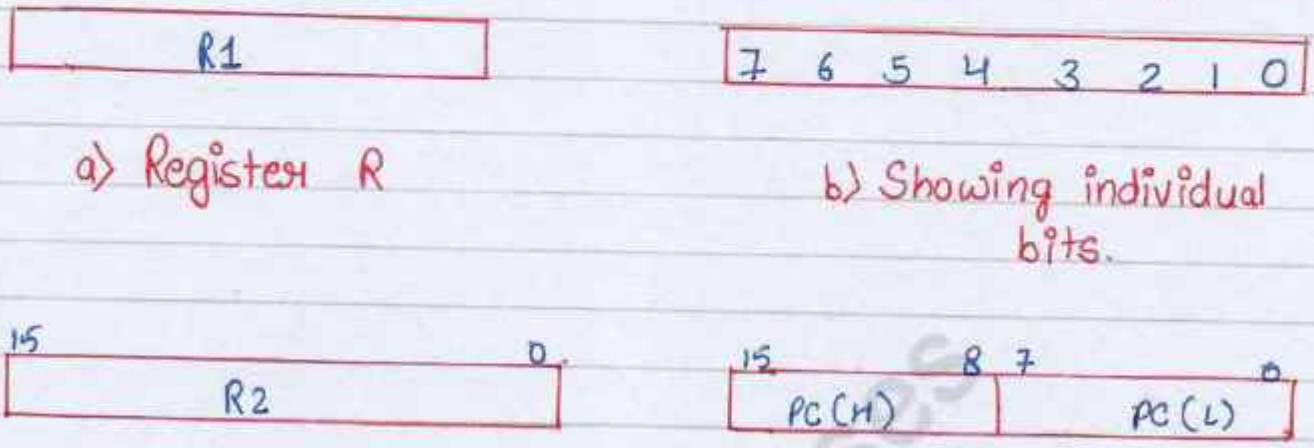
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- It is a convenient tool for describing the internal organization of digital computers in concise and precise manner.

Register

- Computer registers are designated by upper case letters 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 towards the left.

Figure 4.1 Block Diagram of register.



a) Register R

b) Showing individual bits.

c) Numbering of bits

d) Divided into two parts

- The most common way to represent a register is by a rectangular box with the name of the register inside, as in fig. 4.1 (a). The individual bits can be distinguished as in (b).

The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c).

16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L and bits 8 through 15 are assigned the symbol H

The name of the 16-bit register is PC. The symbol PC (0-7) or PC(L) refers to the low-order byte

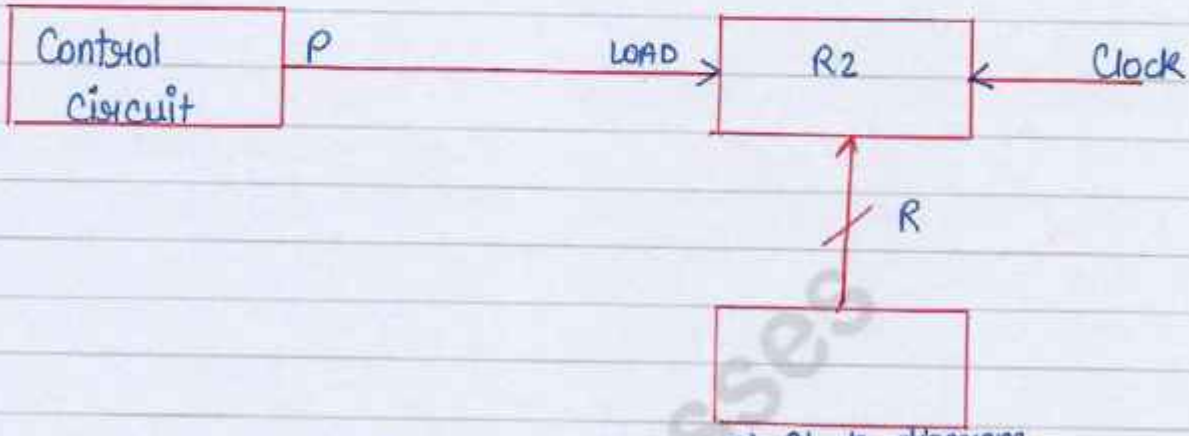
and PC (8-15) or PC (H) to the high order byte.

Register Transfer micro-operation.

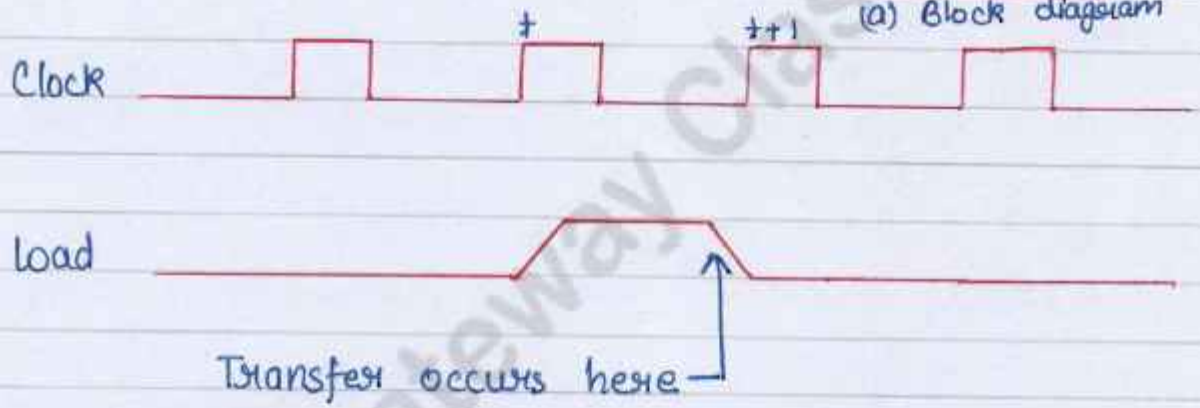
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.
- The statement $R2 \leftarrow R1$ denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R1.
- By definition, the content of the source register R1 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 $R2 \leftarrow R1$
- 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.

- Fig. 4.2 shows the block diagram that depicts the transfer from R1 to R2.

Fig. 4.2 Transfer from R1 to R2 when $P = 1$



(a) Block diagram



(b) Timing diagram.

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.

Gateway Classes

- Register R2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchro-nized 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.
- 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.

Gateway Classes

COA | AKTU | CS / IT / CS Allied / MCA

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

Symbol	Description	Example
Letters (and numeric)	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

AKTU Questions

Q1. Explain arithmetic micro-operations and register micro-operation.

Q2. Explain the steps of instruction cycle.

Shift Micro-operation

Shift micro-operation are used when the data is stored in the registers. The micro-operations are used for the serial transmission of data. Here, the

data bits are shifted from left to right. Operations are also combined with arithmetic and logic micro-operations and data-processing operations.

There are three type of shift micro-operations

- Logical Shift
- Arithmetic Shift
- Circular Shift

Logical Shift

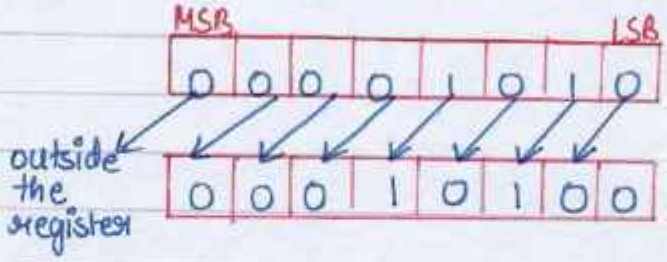
There are two ways to implement the logical shift.
- Logical Shift Left
- Logical Shift Right

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.

- For example, in the below data, there are 8 bits 00001010. When we perform a logical shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftmost bit i.e. 0 will be moved outside, and at the rightmost

place on LSB, 0 will be inserted as shown below

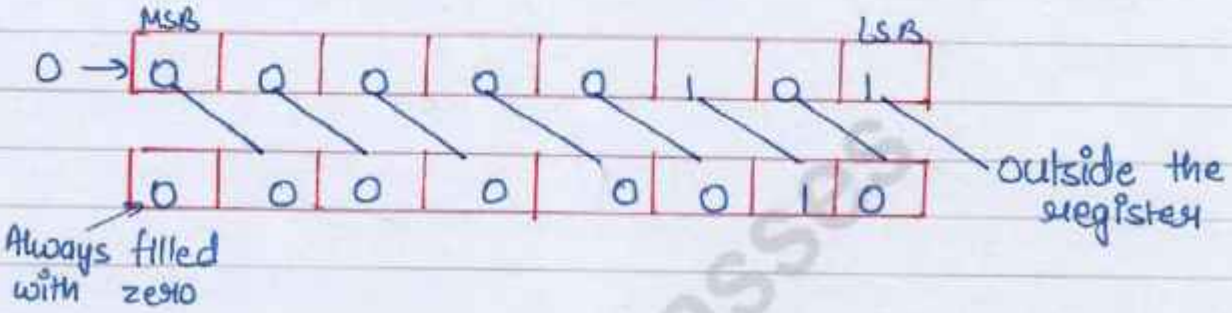


- To implement the logical shift left micro-operation, we use the shl symbol.
- For example, R1 ← shl R1.
- This command means the 8 bits present in the R1 register will be logically shifted left, and the result will be stored in Register R1.
- Moreover, the logical shift left micro-operation denotes the multiplication of 2. The example we've taken above when converted into decimal forms the number 10. And the result after the logical shift operation when converted to decimal forms the number 20.

Logical 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 0.

For example, in the below data, there are 8 bits 00000101. When we perform a logical shift right on these bits, all these bits will be shifted towards the right. The LSB or the rightmost bit i.e. 1 will be moved outside, and at the leftmost place or MSB, 0 will be inserted as shown below.



- To implement the logical shift right micro-operation, we use the shr symbol.
- For example, $R_1 \leftarrow$ shr R1.
- This command means the 8 bits present in the R1 register will be logically shifted right, and the result will be stored in register R1.
- Logical right shift micro-operation generally denotes division by 2. The inputted bits when converted into decimal form the number 5. And the outcome when converted into decimal forms the number 2.

Arithmetic shift Micro-operation

Arithmetic Shift

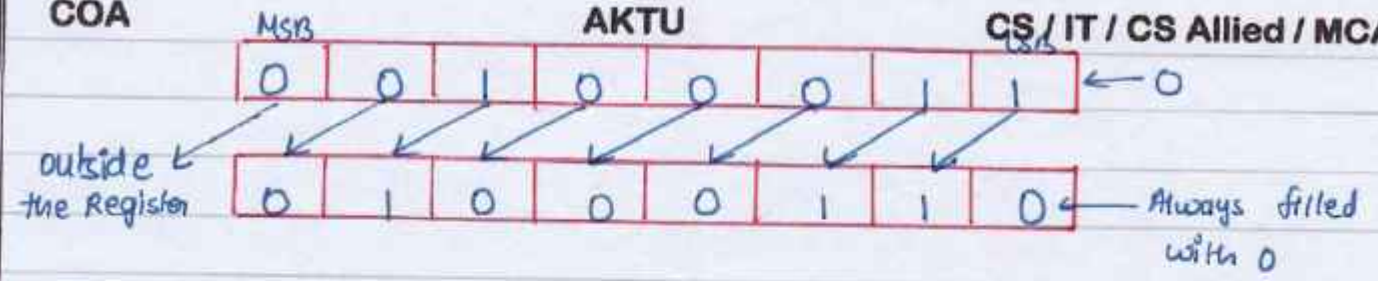
The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position. There are two ways to implement the arithmetic shift.

- Arithmetic Shift Left
- Arithmetic Shift Right

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.

- For example, in the below data, there are 8 bits 00100011. When we perform the arithmetic shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftm-ost bit i.e. 0 will be inserted as shown below



- The given binary number (00100011) represents 35 in the decimal system. And the binary number after logical shift left (01000110) represents 70 in a decimal system. Since 35*2 = 70. Therefore, we can say that the arithmetic shift left multiplies the number by 2.

- To implement the arithmetic shift left micro-operation, we use the ashl symbal.

- For example, R1 ← ashl R1.

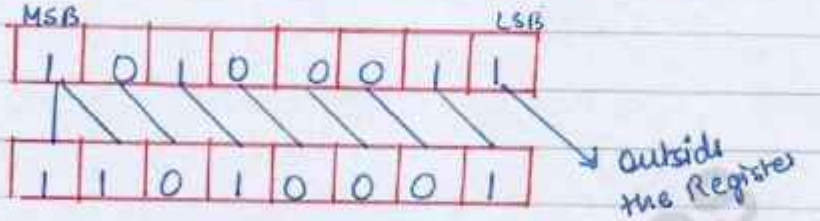
- This command means the 8 bits present in the R1 register will be arithmetic shifted left, and the result will be stored in register R1.

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.

- For example, in the below data, there are 8 bits

10100011. When we perform an arithmetic shift right these bits, all these bits will be shifted towards the right. The LSB or the rightmost bit i.e. 1 will be moved outside, and at the leftmost place or MSB, the previous MSB value, i.e. 1 will be inserted as shown below



- Arithmetic right shift divides the number by 2.
- To implement the arithmetic shift right micro-operation, we use the ashr symbol.
- For example R1 ← ashr R1.
- This command means the 8 bits present in the R1 register will be arithmetic shifted right, and the result will be stored in register R1.

Circular Shift Micro-operation

- The circular shift, also known as the rotate shift, moves the bits in the register's sequence around both ends, thus ensuring no loss of information. There are two ways to implement the circular shift.

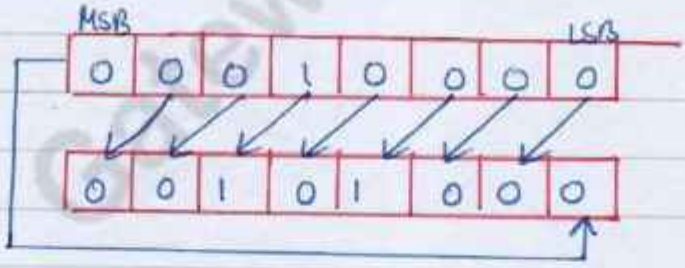
Gateway Classes

- Circular Shift Left
- Circular Shift Right

Circular Shift Left:

- Each bit in the register is shifted to the left 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).

- For example, in the below data, there are 8 bits 00010100. When we perform a circular shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftmost bit i.e. 0 will be placed at the rightmost place or LSB.



- To implement the circular shift left micro-operation, we use the cil symbol.

- For example, R1 ← cil R1.

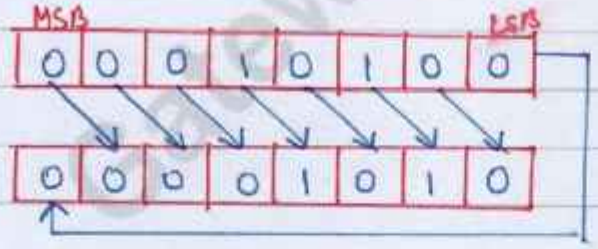
- This command means the 8 bits present in the R1.

register will be circular shifted left, and the result will be stored in register R1.

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).

- For example, in the below data, there are 8 bits 00010100. When we perform a circular shift right on these bits, all these bits will be shifted towards the right. The LSB or the the leftmost bit, i.e. 0, will be placed at the rightmost place or MSB.



- To implement the circular shift right micro-operation, we use the cir symbol.

- For example, R1 ← cir R1.

- This command means the 8 bits present in the R1 register will be circular shifted right, and the

* result will be stored in register R1.

Logic Micro-operation.

- Logic micro-operations are used on the bits of data stored in registers. These micro-operations treat each bit independently and create binary variables from them.

- There are a total of 16 micro-operations available

1. Clear

- The clear logic micro-operation is used to clear the register to 0. To use this micro-operation, we need to feed 0 to the register.

- For example, F ← 0 means the value of the register F is set to 0 or is cleared. The previous value of register F will be removed.

Boolean expression -

- The boolean expression for the clear logic micro-operation is F0 = 0

- The AND logic micro-operation performs the logical AND between the bits of the data stored in the two registers. The symbol to represent the

logical AND is A.

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

Case 1: Both x and Y values are true.

- In the first case, if the values of both two registers are true then the result of AND operation is 1; else, it is 0. F1 represents the truth table of AND logic micro-operation in the above truth table.
- For example, F ← A ∧B means the registers A and B value will undergo AND micro-operation, and the output will be stored in register F.
- Boolean expression –
- The Boolean expression for the AND logic micro-operation will be F1 = x.y

Case 2: x is true and y is false.

- The logical AND operation we discussed above

gives output 1 when both x and y are true. There is also another AND operation which includes x but not y. Also known as inhibition, here are performing the AND operation, the first value is taken from the x variable or register. The second value is taken as the complement of the y variable or register. If the value of the x register is true and of the y register is false, then the result of AND operation is 1; else, it is 0.

- F2 represents the truth table of inhibition AND logic micro-operation in the above truth table.

- For example, $F \leftarrow A \wedge B'$ means the value of the registers A and complement B will undergo AND micro-operation, and the output will be stored in register F.

Boolean Expression-

- The boolean expression for the AND logic micro-operation will be $F2 = x.y'$

Case 3: x is false and y is true

- The third case of logical AND operation includes y but not x. Also known as inhibition, here for

performing the AND operation, the first value is taken as the complement of the x variable or register, and the second value is taken from the y variable or register. If the value of the x register is false and of the y register is true, then the result of AND operation is 1; else, it is 0.

- F4 represents the truth table of inhibition AND logic micro-operation in the above truth table.
- for example, F ← A'.1B means the value of the complement register A and as it is B will undergo AND micro-operation, and the output will be stored in register F.

Boolean expression -

- The Boolean expression for the AND logic micro-operation will be F4 = x'.y

2. Transfer A.

The Transfer A logic micro-operation transfers the contents of register A (first register) to the output register. F3 represents the truth table of Transfer A logic micro-operation in the BELOW truth table. Since there is a transfer of data from the first register to the

output register in this micro-operation its truth table is the same as the taken values of the x variable (0, 0, 1, 1)

- For example, F ← A means the value of register A is moved to register F. The previous value of register F will be removed.

Boolean expression -

- The Boolean expression for the Transfer A logic micro-operation is F3 = x

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

3. Transfer B

- The transfer B logic micro-operation transfers the contents of register B (second register) to the output register. F5 represents the truth table of transfer B logic micro-operation in the above truth table is the same as the taken values of the y variables (0, 1, 0, 1).

- For example $f \leftarrow B$ means the value of register B is moved to register f. The previous value of register f will be removed.

Boolean Expression -

The Boolean Expression for the Transfer B logic micro-operation is $f5 = y$.

y	f5 = y
0	0
1	1
0	0
1	1

4. Exclusive OR

- Also known as XOR, this logic micro-operation performs the logical XOR between the data bits stored in the two registers. The logical XOR means either x should be true or y but not both. The symbol to represent the Exclusive OR is $\oplus$.

- F6 represents the truth table of Exclusive OR logic micro-operation in the above truth table. The output will be 1 when either $x = 1$ and $y = 0$ or $x = 0$ and $y = 1$.

- For example, $F \leftarrow A \oplus B$ means the registers A and B value will undergo XOR micro-operation, and the output will be stored in register F.

Boolean expression -

The Boolean expression for the Exclusive OR logic micro-operation will be $F6 = x.y' + x'.y$.

X	Y	F6	X'	Y'	XY'	X'Y	X'Y + XY'
0	0	0	1	1	0	0	0
0	1	1	1	0	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	0	0	0	0

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

The OR logic micro-operation performs the logical OR between the data bits stored in the two registers. The symbol to represent the logical OR is V.

OR is V.

Case 1 : Either x or y. or both x and y values are true.

In the first case, if either the value of x register is false, or the value of x regist is false, and y register is true, or both the values of x and y registers are true, then the result of OR operation is 1 else is 0. F7 represents the truth table of OR logic micro-operation in the above truth table.
For example, F ← A V B means the registers A and B value will undergo OR micro-operation, and the output will be stored in register F.

Boolean expression – The boolean expression for the OR logic micro-operation will be F7 = x + y.

Case2 : If y, then x else not.
In the second case, the output for 1. follows the condition that if the value of the y register is true, then the value of the x register must be true. If this condition is satisfied, then the output is 1.

If the value of the y register is false, then

we don't need to look for the value of the x register, and the output is 1 else the output is 0.
To perform this logic micro-operation, we need to perform the logical OR of the values of the x register and the complement value of the y register.
In the above truth table, F11 represents the truth table of this logic micro-operation.
For example, $F \leftarrow A \vee B'$ means the value of the registers A and complement B will undergo OR micro-operation, and the output will be stored in register F.

Boolean Expression – The Boolean expression for this OR logic micro-operation will be $F11 = x + y'$.

Case 3: If x, then y else not.

In the third case, the output for 1 follows the condition that if the value of the x register is true, then the y-register's value must be true. If this condition is satisfied, then the output is 1. If the value of the x register is 0, then we don't need to look for the value of the y register, and the output is 1. Else the output is 0.

To perform this logic micro-operation, we need

to perform the logical OR of the complemented value of the x register and the value of the y register.

In the above truth table, F_{13} represents the truth table of this logic micro-operation.

For example, $F \leftarrow A' \vee B$ means the complemented register A and B value will undergo OR micro-operation, and the output will be stored in register F.

Boolean Expression -

The Boolean expression for this OR logic micro-operation will be $f_{13} = x' + y$

7. NOR - The NOR logic micro-operation is simply the opposite of OR logic micro-operation. As the name suggests, it is Not OR. The output of OR micro-operation is 1 when the value of either x register or both x and y registers are true. In contrasts, in NOR, the output is 0 when the value of either x register or y register or both x and y registers are true, and it is 1 when both x and y registers are false. In the truth table, f_8 represents the truth table of NOR logic micro-operation.

for example, $F \leftarrow (A \vee B)'$ means the registers A and B value will undergo NOR micro-operation, and the output will be stored in register F.

Boolean Expression - The Boolean expression for the NOR micro-operation is $F8 = (x+y)'$

x	y	x + y	$F8 = (x+y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

8 Exclusive NOR:

We perform the Exclusive NOR micro-operation, the output will be 1 when the values of both the x and y registers will be the same. They can be true or false, but they have to be the same.
F9 represents the truth table of Exclusive NOR logic micro-operation in the above truth table. The output will be 1 when either $x=0$ and $y=0$ or $x=1$ and $y=1$.

for example, $F \leftarrow (A \oplus B)'$ means the registers A and B value will undergo Exclusive NOR micro-

operation, and the output will be stored in register F.

Boolean Expression - The Boolean expression for the Exclusive NOR logic micro-operation will be $F9 = x.y + x'y'$

X	Y	F9	X'	Y'	XY	X'Y'	$(A \oplus B)' = XY + X'Y'$
0	0	1	1	1	0	1	1
0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	1	1	0	0	1	0	1

9. Complement B

The complement B logic micro-operation transfers the complemented contents of register B (second register) to the output register. First, the content of the register is complemented and then moved to the desired register.

In the truth table, F10 represents the truth table of complement B logic micro-operation. Since there is a transfer of complemented data from the second register to the output register in this micro-operation, its truth table is just the opposite of the taken values of the y variable

(1, 0, 1; 0).

For example, $F \leftarrow B'$ means the complemented value of register B is moved to register F. The previous value of register F will be removed.

Boolean Expression -

The Boolean expression for the complement B logic micro-operation is $F10 = y'$

Y	$F10 = y'$
0	1
1	0
0	0
1	1

10. Complement A -

The complement A logic micro-operation transfers the complemented contents of register A (first register). register. First, the content of the register is complemented and then moved to the desired register.

F12 represents the truth table of complement A logic micro-operation in the above truth table.

Since there is a transfer of complemented data from the first register to the output register in this micro-operation, its truth table is just the opposite of the taken values of the x variable (0, 1, 0, 1).

For example, $F \leftarrow A'$ means the complemented value of register A is moved to register F. The previous value of register F will be remov-ed.

Boolean Expression -

The boolean expression for the complement A logic micro-operation is $F12 = x'$

X	$F12 = X'$
0	1
0	1
1	0
1	0

11. NAND

The NAND logic micro-operation is simply the oppos-ite of AND logic micro-operation. As the name suggests, it is Not NAND. The output of AND micro-operation is 1. when the value of both

x register is true, and it is 1 when either x is false, or y is false, or both are false.

In the below truth table, F14 represents the truth table of NAND logic micro-operation.

For example, $F \leftarrow (A \wedge B)'$ means the registers A and B value will undergo NAND micro-operation, and the output will be stored in register F.

Boolean expression -

The boolean expression for the NAND logic micro-operation is $F14 = (x.y)'$

X	Y	X.Y	$F14 = (X.Y)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

12. Set to all 1's

The set to all 1's logic micro-operations is used to set all the register bits to 1. To use this micro-operation, we just need to feed 1.

to the register. In the below truth table f_{15} represents the truth table of set to all 1's logic micro-operation.

For example, $F \leftarrow 1$ means the value of the register F is set to 1. The previous value of register F will be removed.

Boolean expression - The boolean expression for the set of all 1's logic micro-operation is $f_{15} = 1$.

X	$f_{15} = 1$
0	1
0	1
1	1
1	1

Summary

Boolean Function	Micro-operation	Name
$f_0 = 0$	$F \leftarrow 0$	Clear
$f_1 = x.y$	$F \leftarrow A \wedge B$	AND
$f_2 = x.y'$	$F \leftarrow A \wedge \bar{B}$	
$f_3 = x$	$F \leftarrow A$	Transfer A
$f_4 = x'.y$	$F \leftarrow \bar{A} \wedge B$	
$f_5 = y$	$F \leftarrow B$	Transfer B
$f_6 = x \oplus y'$	$F \leftarrow A \oplus B$	Exclusive OR

Boolean function	Micro-operation	Name
$F_7 = x + y$	$F \leftarrow A \vee B$	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
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (x.y)'$		NAND
$F_{15} = 1$		Set to all 1's

INTRODUCTION (RISC and CISC)

- RISC stands for Reduced Instruction Set Computer Processor and CISC stands for Complex Instruction Set Computer.
- 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.

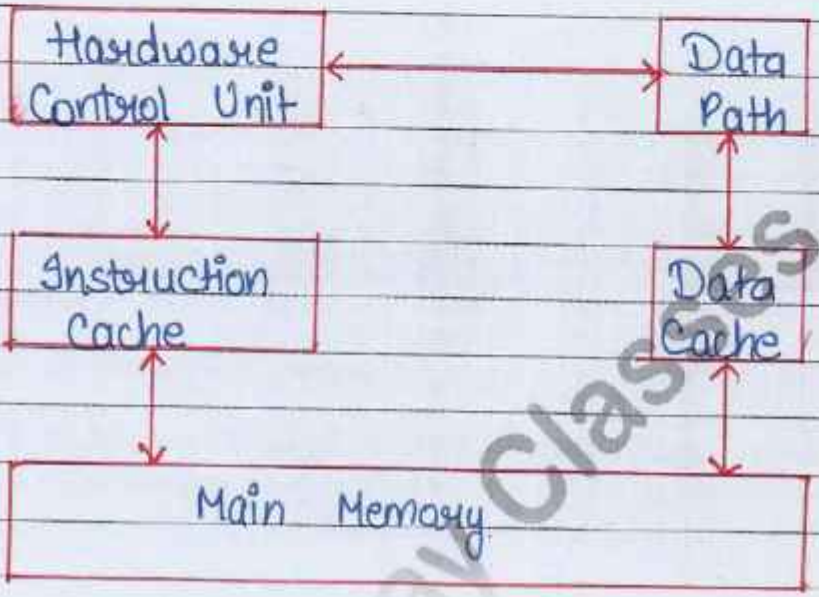
Reduced Instruction Set Computer (RISC)

- 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 consists of three parame-ters : fetch, decode, and execute.
- The RISC processor can also reduce multiple complex instructions into a simple one. RISC chips require several transistors, making them less expensive to develop and reducing instruct-ion execution time.
- Examples of RISC processors are PowerPC, microchip PIC, SUN's SPARC, RISC-V
- 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 instru-ction set having limited number of addressi-ng 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

RISC Architecture -



- RISC processor is implemented using the hardwired control unit.
- The hardwired control unit produces control signals which regulate the working of processors hardware. RISC architecture emphasizes on using the registers rather than memory.
- This is because the registers are the 'fastest' available memory source.
- The registers are physically small and are placed

on the same chip where the ALU and the control unit are placed on the processor.

- The RISC instructions operate on the operands present in processor's registers.
- All instructions in RISC are simple and execute one instruction per cycle. So here, the instruct-ons are hardwired and there is no need for control store. For each operation, we will have as defined hardwire. Making an instruction hardwired is making a function or operation in instruction permanent using connected circuits.
- Instruction and data path fetches the opcode and operands are stored.

Features of RISC Processor

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

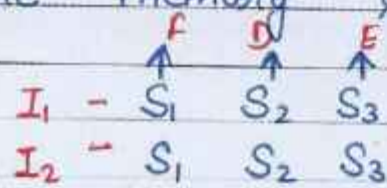
1. RISC processors use one clock per cycle (CPI) to execute each instruction in a computer. Each CPI also comprises the methods for fetching, decoding, and executing computer instru-ctions.

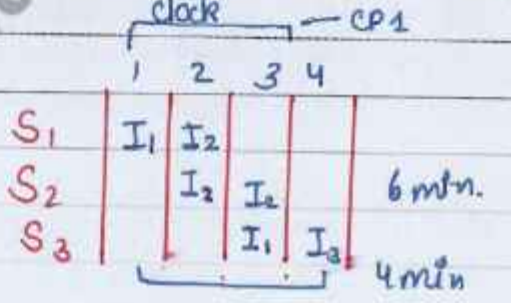
2. Multiple registers in RISC processors allow them to hold instructions, reply fast to the computer, and interact with the computer memory as little as possible.

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 types and synthesis complex data type.

8. It used fixed length of instruction for pipelining

9. RISC permits any register to use in any context

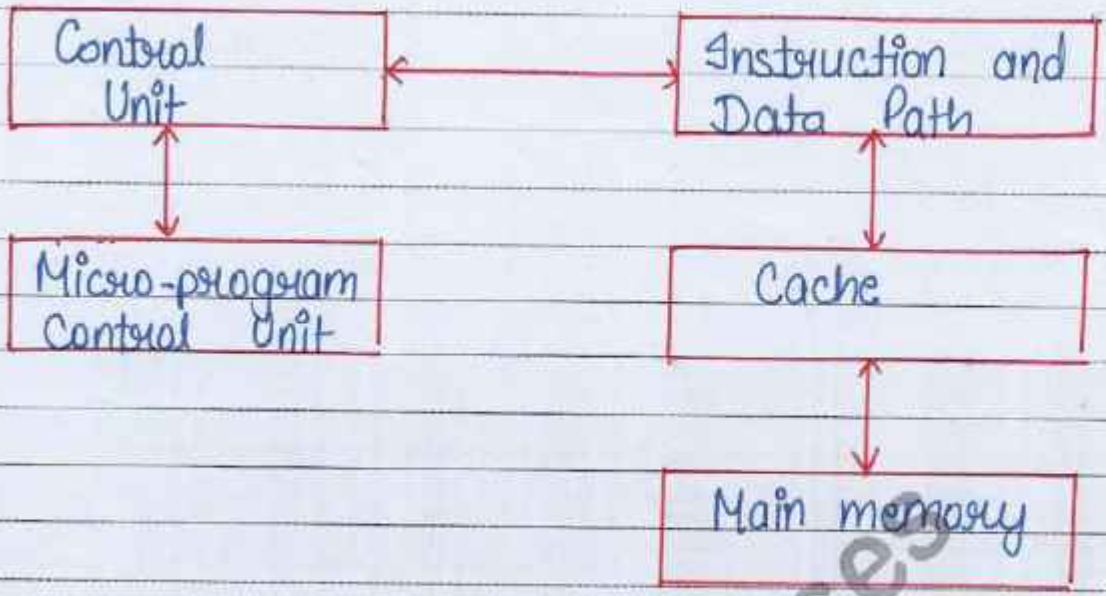
Advantages -

- **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)

- 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.

Gateway Classes



- The main objective of a CISC processor is to minimize the program size by reducing the number of instructions in a program. This is done by embedding some of the low-level instructions in a single complex instruction.

- If a program / software is getting simplified then the hardware has to get on work and must be able to perform the complex tasks. That's why a CISC processor has complex hardware.

- Here, you have a special microprogram control unit that uses a series of microinstructions of the microprogram stored in the "control memory" of the microprogram control unit and generate the control signals.

- The control units access the control signals produced by the microprogram control unit and operate the functioning of processors hardware.
- Instruction and data path fetches the opcode and operands of the instructions from the memory.
- Cache and main memory is the location where the program instructions and operands are stored.

Features of CISC Processor.

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.

Advantages of CISC :

- Reduced code size - CISC processors use complex instructions that can perform multiple operations.

- 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.

- Widely used - CISC processors have been in use for a longer time then RISC processors, so they have a larger user base and more available software.

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

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. (1 word = 2 Byte)	• Instruction are larger than the size of one word.
• Fixed sized instructions.	• Variable sized instructions.

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

• 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.
• Simple and limited addressing Cycle.	• Complex and more addressing modes.
• RISC is Reduced Instruction Cycle	• CISC is complex Instruction Cycle.
• The number of instructions are more as compared to CISC.	• The number of instructions are less 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.

Advantages of Having Variable Length Instruction Format

- **Code Density:** Variable-length instruction formats allow for more compact code representation. Since instructions can vary in size, it's possible to encode simple and common instructions with fewer bits, resulting in smaller program sizes. This can lead to more efficient use of memory and improved cache performance.

- **Reduced Memory Bandwidth:** Smaller instruction sizes mean that less memory bandwidth is required to fetch instructions. This can be especially advantageous in situations where memory bandwidth is a bottleneck, such as in embedded systems or systems with limited memory resources.

- **Improved Cache performance:** Variable-length instructions can enhance cache performance by allowing more instructions to be stored in a given cache size. This can result in better instruction cache hit rates and, consequently improved overall system performance.

- **Flexible Instruction Encoding:** Variable length instruction formats provide flexibility in encoding

complex or less frequently used instructions. This allows for a more efficient use of the available instruction encoding space, as the encoding length can be tailored to the complexity of the instruction.

- **Simpler Instruction fetch Logic:** Variable length instruction formats can simplify the instruction fetch logic in the processor. The variable-length nature allows the processor to fetch a variable number of instructions in a single fetch cycle, depending on the lengths of the instructions in the program.

Disadvantages of Having Variable Length Instruction Format:

- **Complex Instruction Decoding:** Decoding variable-length instructions can be more complex compared to fixed-length instructions. The processor needs to determine the boundaries of each instruction before decoding it, adding complexity to the instruction fetch and decode stages.

- **Difficulty in Pipelining:** Variable length instructions can make it challenging to design efficient instruction pipelines. In a pipelined processor, each stage of the pipeline typically processes one instruction at a time. With variable

length instructions, it becomes more difficult to predict the start of the next instruct-ion, potentially leading to pipeline stalls and reduced performance.

- **Increased Design Complexity:** The variable nature of instructions introduces complexity in the design of the instruction fetch unit, instruc-tion decoder, and the other components of the processor. This complexity can make the design and verification of the processor more challenging.

- **Potential for Higher Latency:** Due to the variable length nature, fetching and decoding instruc-tions may take more time than with fixed-length instructions. This can results in higher instruction fetch latency, impacting the overall performance of the processor.

- **Limited Parallelism:** Variable-length instructions may limit the degree of instruction-level parallelism that can be achieved. Fixed-length instructions are often easier to pipeline and parallelize since each instruction occupies a uniform amount of space.

- **Code Alignment Issues:** Variable-length instructions can lead to code alignment issues, especia-

-lly if the assembler does not properly align instructions. Misalignment can result in perform -ance penalties or even program failure on certain architectures.

- Less Predictable Memory Access - Variable length instructions may lead to less predictable memory access patterns. Fixed-length Instruc -tions, on the other hand, allow for more predictable fetching of instructions blocks, which can be advantageous for certain caching strate -gies.

- Compiler Complexities: Compilers may find it more challenging to generate optimal code for arch -itectures with variable-length instructions. The compiler needs to consider the variable size of instructions when optimizing code, which can be more complex than dealing with a fixed-length instruction set.

AKTU Questions.

Q1. Write a short note on RISC? Explain its characteristics. AKTU 2014-15, 15-16, 20-21

Q2. Write a short note on CISC? Explain its characteristics. AKTU 2015-16.

Q3. Explain comparison between RISC and CISC.

OR

Explain the difference between RISC and CISC based microprocessor AKTU 2016-17, 22-23, 19-20, 17-18

AKTU Questions

Q1. Explain the concept of pipelining. AKTU 2022-23, 18-19

Q2. Explain speed up performance metric for pipelined processor. AKTU 2021-22

Q3. Justify the content of pipelining using a suitable diagram. AKTU 2021-22

Q4. What is pipelining in computer architecture? 2019-20

Q5. Explain all phases of instruction cycle. AKTU 2018-19

Q6. How pipeline performance can be measured? Discuss. Give a space time diagram for visualizing the pipeline behaviour for a four stage pipeline. AKTU 2017-18

Pipelining

- 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.
- 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 the 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-piplelined operation, a battle 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 battle moves to stage 3, both stage 1 and stage 2 are idle.

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

Stage	1	2	3	4	5	6
I	B_1			B_2		
F		B_1			B_2	
S			B_1			B_2

non-pipeline 6 min

	1	2	3	4	5	6
I	B_1	B_2				
F		B_1	B_2			
S			B_1	B_2		

Pipeline 4 min

without pipelining = 9/3 min = 3m

I F S | | | | | | |
| | | I F S | | | |
| | | | | | I F S (9 min)

with pipelining = 5/3 min = 1.67 m.

I F S | |
| I F S |
| | I F S (5 min)

Thus we can say pipelining increase efficiency

- In pipelined architecture,
- The hardware of the CPU is split upto several functional units.
- Each functional units 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.

Design of basic pipeline -

- In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages / segments such that the output of one stage is connected to the input of the next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

Execution in a pipelined processor -

- Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram. For example, consider a processor having 4 stages and let there be 2 instructions to be executed. We can visualize the execution

Sequence through the following space-time diagrams

Non-overlapped execution:

Stage / Cycle	1	2	3	4	5	6	7	8
S_1	I_1				I_2			
S_2		I_1				I_2		
S_3			I_1				I_2	
S_4				I_1				I_2

Total time = 8 cycle

Overlapped execution:

Stage / Cycle	1	2	3	4	5
S_1	I_1	I_2			
S_2		I_1	I_2		
S_3			I_1	I_2	
S_4				I_1	I_2

5 cycle

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.

4 Stage Pipelining

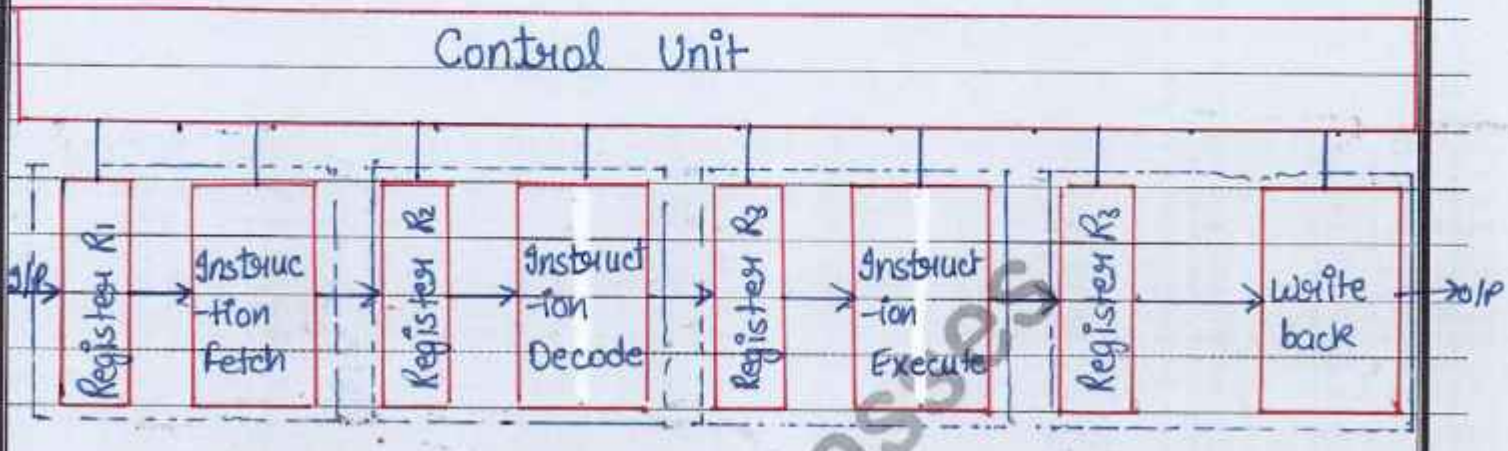
Instruction fetch (IF)
Instruction decode (ID)
Instruction Execute (IE)
Write back (WB)

To implement four stage pipeline,

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

- The hardware of the CPU is divided into four functional units.
- Each functional unit performs a dedicated task.



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.

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

Stage-04: Stage-04, fourth functional unit performs write back the result so obtained after executing the instruction.

Clock cycles →

stages ↓	1	2	3	4	5	6
S_1	I_1	I_2	I_3			
S_2		I_1	I_2	I_3		
S_3			I_1	I_2	I_3	
S_4				I_1	I_2	I_3

Phase time Diagram

	1	2	3	4	5	6	7
S_1	I_1	I_2	I_3				
S_2		I_1	I_2	I_3			
S_3			I_1	I_2	I_3		
S_4				I_1	I_2	I_3	
S_5					I_1	I_2	I_3

Performance of Pipelining Processor

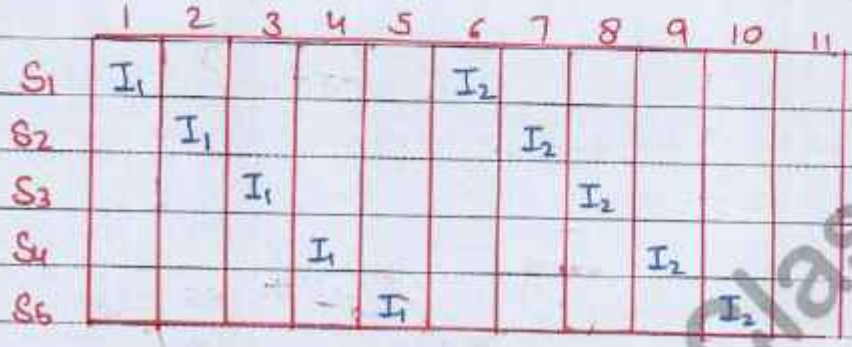
Performance of pipelined processor Consider a 'k' segment pipeline with clock cycle time as 'Tp'. Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n-1' instructions will take only '1' cycle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

$ETpipeline = k + n - 1 \text{ cycles} = (k+n-1)Tp$

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

$ET_{non-pipeline} = n * k * T_p$

n → task / Instruction
k → Stage / Segment
T_p → time



$5 * 2 * 1 = 10$

Speed Up of Pipelining Processor

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 = $ET_{non-pipeline}$ / $ET_{pipeline}$.

$\Rightarrow S = [n * k * T_p] / [(k+n-1) * T_p]$

$S = [n * k] / [k+n-1]$

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

$S = n * k/n$

$S = k.$

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 = $n / (k+n-1) * T_p$

Note: The cycles per instruction (CPI) value of an ideal pipelined processor is 1.

Throughput of Pipelining Processor

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

Throughput:

- It measures 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 measures 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 is influenced by pipeline length, depth, clock cycle time, instruction dependencies and pipeline hazards.

Dependency of Pipelining Processor

There are mainly three types of dependencies possible in a pipelined processor. These are:

1. Structural Dependency
2. Control Dependency
3. Data Dependency

These dependencies may introduce stalls in the pipeline.

Stall : A stall is a cycle in the pipeline without new input.

Structural Dependency

This dependency arises due to the resources conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU. Example:

Instruction / Cycle	1	2	3	4	5
l_1	IF (Mem)	ID	EX	Mem	
l_2		IF (Mem)	ID	EX	
l_3			IF (MEM)	ID	EX
l_4				IF (Mem)	ID

- In above scenario, in cycle 4, instructions l_1 and l_4 are trying to access same resource which introduces a resource conflict. To avoid this problem, we have to keep the instruction on wait until the required resource becomes available. This wait will introduce stalls in the pipeline as shown below:

Cycle	1	2	3	4	5	6	7	8
l_1	IF (Mem)	ID	EX	Mem	WB			
l_2		IF (Mem)	ID	EX	Mem	WB		
l_3			IF (Mem)	ID	EX	Mem	WB	
l_4				-	-	-	IF (Mem)	

Stall

Solution for structural dependency - To minimize ~~set~~ structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming.

Renaming: According to renaming, we divide the memory into two independent modules used to store the instruction and data separat

-ely called Code memory (CM) and Data memory (DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the instructions.

Instruction / Cycle	1	2	3	4	5	6	7
I_1	IF (CM)	ID	EX	DM	WB		
I_2		IF(CM)	ID	EX	DM	WB	
I_3			IF(CM)	ID	EX	DM	WB
I_4				IF(CM)	ID	EX	DM
I_5					IF(CM)	ID	EX
I_6						IF(CM)	ID
I_7							IF(CM)

Control Dependency

Control Dependency (Branch Hazards)

- This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc.

- On many instructions architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline.

- Due to this, unwanted instructions are fed to the pipeline. Consider the following sequence of instructions in the program:

- 100 : I_1
- 101 : I_2 (JMP 250)
- 102 : I_3
- .
- .
- 250 : BI_1
- Expected output : $I_1 \rightarrow I_2 \rightarrow BI_1$

Note: Generally, the target address of the JMP instruction is known after ID stages only.

Instruction / Cycle	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC: 250)	EX	MEM	WB
I_3			IF	ID	EX	Mem
BI_1				IF	ID	EX

- Output Sequence.: $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow BI_1$ So, the output sequence is not equal to the expected output, that means the pipeline is not implemented correctly. To correct the above problem we need to stop the Instruction fetch until we get target address of branch

instruction. This can be implemented by introducing delay slot until we get the target address.

Instruction / Cycle	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID(PC: 250)	EX	Mem	
Delay	–	–	–	–	–	–
BI_1				IF	ID	EX

- Output Sequence: $I_1 \rightarrow I_2 \rightarrow$ Delay (Stall) $\rightarrow BI_1$ As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline.

- Solution for Control dependency Branch Prediction is the method through which stalls due to control dependency can be eliminated. In this at 1st stage prediction is done about which branch will be taken. For branch prediction Branch penalty is zero.

Branch Penalty: The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

Note: As we see that the target address is

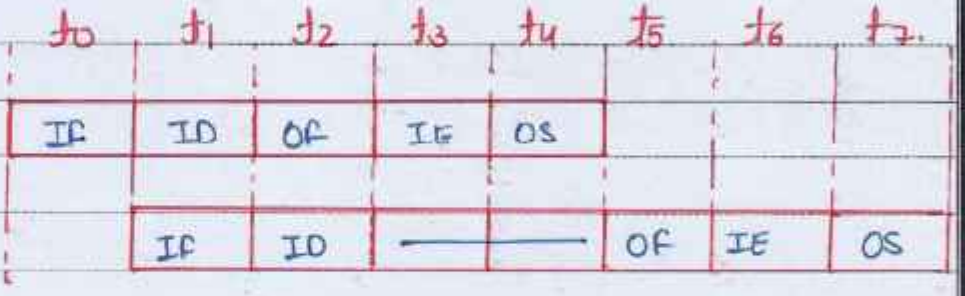
available after the ID stages, so the number of stalls introduced in the pipeline is

1. Suppose, the branch target address would have been present after the ALU stage, there would have been 2 stalls. Generally, if the target address is present after the K^{th} stage, then there will be (K-1) stalls in the pipeline.

- 100 : I_1
- 101 : I_2 (JMP 250)
- 102 I_3 NOP (No operation)
- .
- 250 : BI_1
- Expected output : $I_1 \rightarrow I_2 \rightarrow BI_1$

Data Dependency

- Consider the following two instructions and their pipeline execution.



In the figure above, you can see that result of the ADD instruction is stored in the register R2 and we know that the final result is stored at the end of the execution of the instruction which will happen at the clock cycle t_4. But the Sub instruction need the value of the register R2 at the cycle t_3. So the Sub instruction has to stall two clock cycles. If it doesn't stall it will generate an incorrect result. Thus depending of one instruction on other instruction for data is data dependency. Solution for data dependency is operand forwarding.

	t_0	t_1	t_2	t_3	t_4			
ADD R2, R3, #100	IF	ID	OF	IE	OS			
SUB R9, R2, #30		IF	ID	–	OF	IE	OS	

5 stage pipeline.

	t_0	t_1	t_2	t_3	t_4	t_5
ADD R2, R3, #100	IF	ID	IE	MA	WR	
SUB R9, R2, #30		IF	ID	IE	MA	WB

operand ↑ fwd.

	t_0	t_1	t_2	t_3	t_4	t_5		
ADD R2, R3, #100	IF	ID	IE	MA	WB			
SUB R9, R2, #30		IF	ID	–	–	IE	MA	WB

Data Hazards

Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline.

There are mainly three types of data hazards:

1. RAW (Read after Write) [Flow / True data dependency]
2. WAR (Write after Read) [Anti-Data dependency]
3. WAW (Write after write) [Output data dependency]

Let there be two instructions I and J, such that

- instruction depend on result of prior instruction still in the pipeline.
- It can occur among the operands in the instruction at the pipeline stages.
- It occur when instructions read or write registers that are used by other instructions.
- RAW hazard occurs when instruction J tries to read data before instruction I writes it.
  Eg: I : R2 ← R1 + R3 J : R4 ← R2 + R3.

- WAR hazard occurs when instruction J tries to write data before instruction I reads it.
  Eg: I: $R_2 \leftarrow R_1 + R_3$ J: $R_3 \leftarrow R_4 + R_5$.

- WAW hazard occurs when instruction J tries to write output before instruction I writes it.
  Eg: I: $R_2 \leftarrow R_1 + R_3$ J: $R_2 \leftarrow R_4 + R_5$ J

AKTU Questions

Q1. Explain the concept of pipelining. AKTU 2022-23, 18-19.

Q2. Explain speed up performance metric for pipelined processor. AKTU 2021-22.

Q3. Justify the concept of pipelining using a suitable diagram. AKTU 2021-22.

Q4. What is pipelining in computer architecture? AKTU 2019-20.

Q5. Explain all phases of instruction cycle. AKTU 2018-19.

Q6. How pipeline performance can be measured? Discuss. Give a space time diagram for visualizing the pipeline behaviour for a four stage pipeline. AKTU 2017-18.

Classification of Pipelining

◉ 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 subtract of floating points on a unit of the pipeline here.

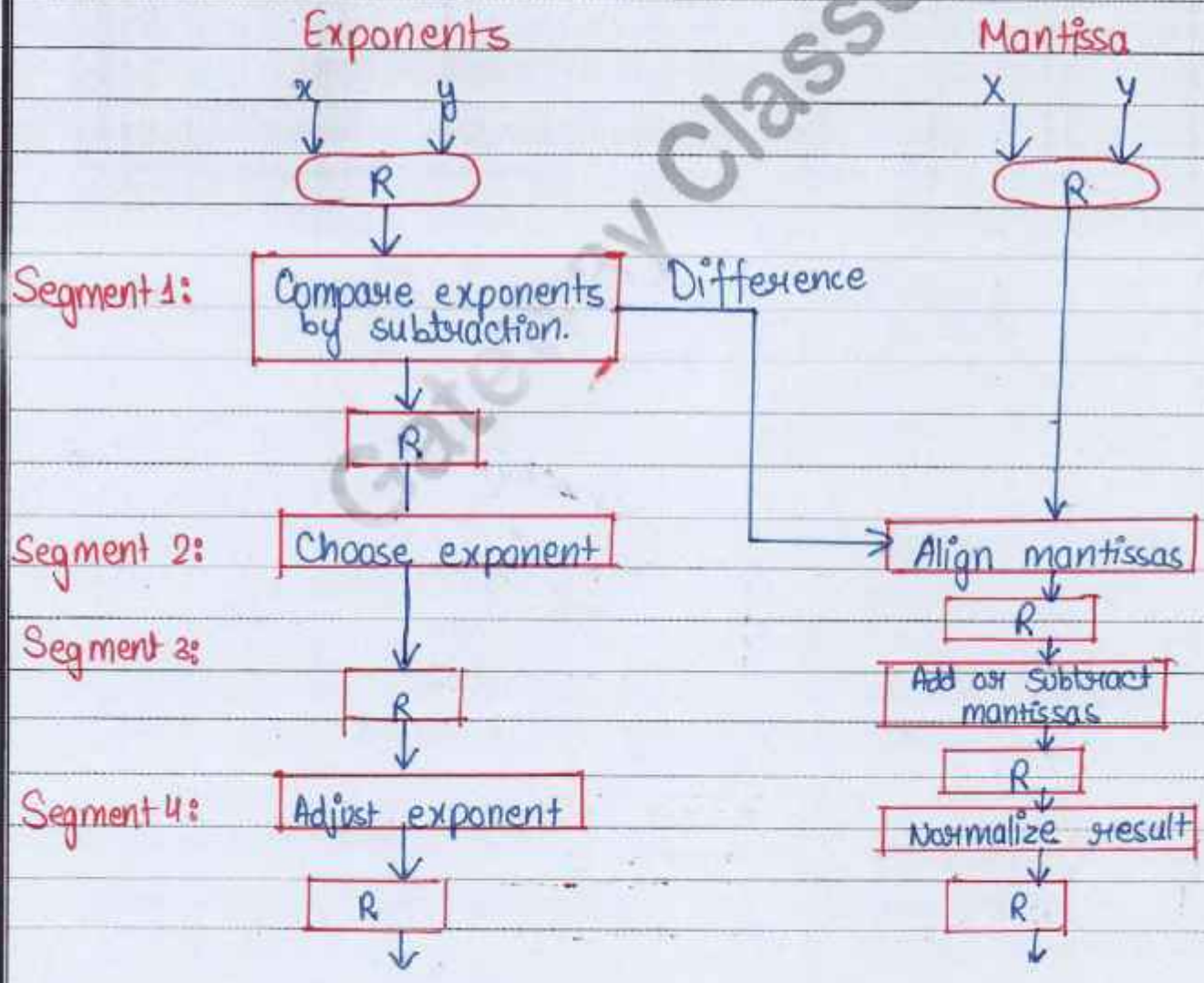
→ The inputs in the floating-point adder pipeline refer to two different normalized floating-point binary numbers. These are defined as follows:

- $A = X * 2^x = 0.9504 * 10^3$
- $B = Y * 2^y = 0.8200 * 10^2$

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.
2. Aligning the mantissa
3. Adding or subtracting the mantissa
4. Normalizing the result

	1	2	3	4	5	6
S_1	I_1	I_2	I_3	I_4	I_5	I_6
S_2		I_1	I_2	I_3	I_4	I_5
S_3			I_1	I_2	I_3	I_4
S_4				I_1	I_2	I_3



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

1. Comparing Exponents by Subtraction - The differen-ce 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.
$A = 0.9504 * 10^3$
$B = 0.08200 * 10^3$

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:

$C = 0.1324 * 10^4$

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.

Instruction Pipelining.

Pipeline processing can happen ~~to~~ 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 instruct-ions 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.

- Each step is carried out in its own segment, and various segments may take different amonts 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.

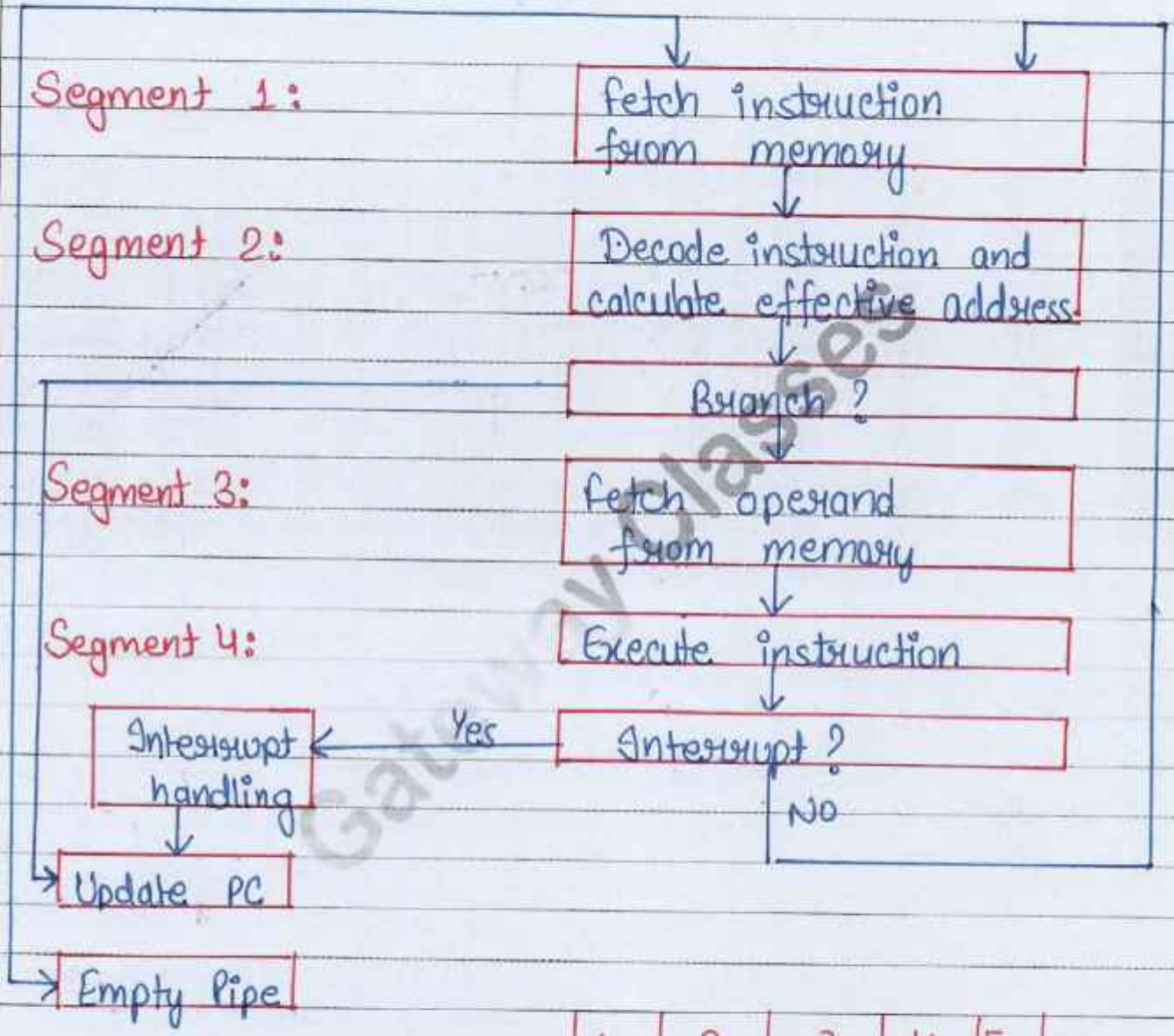
four segment instruction pipeline is illustrated in the block diagram given below. The instruction cycle is divided into four parts:

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 instruction would finally be executed in the very last segment of a pipeline organisation.



	1	2	3	4	5
Fetch → S_1	I_1	I_2	I_3	I_4	
EA + Decode → S_2		I_1	I_2	I_3	
FO → S_3			I_1	I_2	
Execute → S_4				I_1	

Difference

Arithmetic pipelining focuses on parallelizing arithmetic operations, while instruction pipelining optimizes the execution of instructions. Arithmetic pipelines breakdown arithmetic operations into stages, whereas Instruction pipelines breakdown the instruction process into stages.

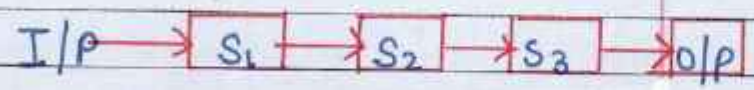
Linear Pipeline	Non-Linear Pipeline
• In linear pipeline a series of processors are connected together in a serial manner.	• In Non-Linear pipeline different pipelines are present at different stages.
• Linear pipeline is also called as static pipeline as it performs fixed functions.	• Non-Linear pipelines 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 feed-back and feed-forward connections.

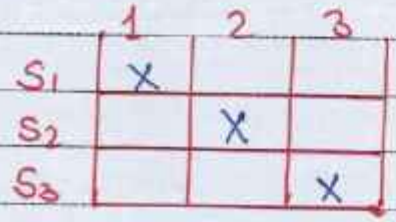
Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

- It generates a single reservation table.
- It allows easy functional partitioning.

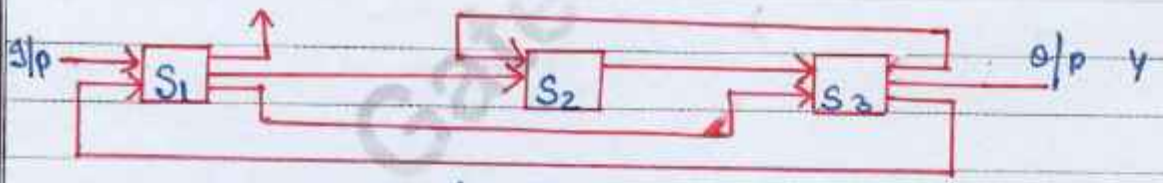
- It can generate more than one reservation table.
- Functional partitioning is difficult in non-linear pipeline.



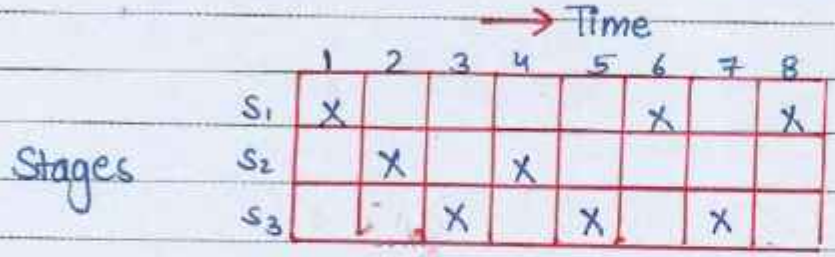


Reservation Table

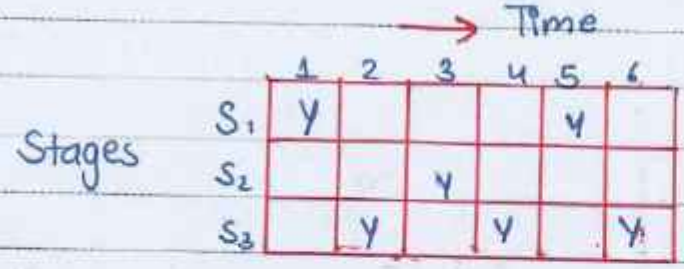
Non-Linear Pipeline



(a) A three-stage pipeline



(b) Reservation table for function X



(c) Reservation table for function Y

Difference

Aspect	Pipelining	Parallelism
Nature of concurrency	Achieves concurrency by breaking down the execut-ion of a single instruction into stages and processing multiple instructions in diff-erent stages simultaneous-ly.	Achieves concurrency by executing multiple instructions or tasks simultaneously, either at the instruction lev-el, thread level, or data level.
Dependency Handling	Dependencies between instru-ctions 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 m-ust also be considered, but different techniques such as out-of-orde-r execution or dyn-amic scheduling, can be employed to handle dependencies.

Gateway Classes

COA | AKTU | CS / IT / CS Allied / MCA

Aspect	Pipelining	Parallelism
Resource Usage	Use a single set of hard-ware components that are shared among diffe-rent stages of the pipeli-ne.	Involves multiple sets of hardware componen-ts that can operate concurrently on differen-t instructions or tasks.
Workflow	Each stage of the pipeli-ne is responsible for a specific task; and as soon as one stage completes its task, it passes the re-sult to the next stage.	Simutaneous execution of multiple instructions or tasks, either at the ins-truction level, thread level or data level.
Advantages	Improved throughput and overall system performance. Increased instruction thro-ughput by overlapping stages. Utilizes resources more efficiently.	Faster execution of tasks by performing them sim-ultaneously. Better resour-ce utilization. Enhanced overall system throughp-ut.

AKTU Questions

Q. Explain different type of pipelining. AKTU 2021-22

OR

What is instruction pipelining 2018-19

Q2. Difference between linear and non-linear pipeline. AKTU 2019-20

Q3. What is parallelism and pipelining in computer architecture AKTU 2019-20

AKTU Questions

Q1. Explain hardwired control unit in detail. AKTU 2022-23, 2021-22

Q2. Explain hardwired control unit with its block diagram. AKTU 2021-22

Q3. What is hardwired control unit? Discuss the method of designing hard wired control unit. AKTU 2016-17, 2018-19

Q4. What is hardwired control unit? List various design method for hardwired control. Discuss in detail using diagram any one of the method for designing GCD processor. AKTU 2017-18.

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.
- The hardwired and microprogrammed control unit generates the control signal to fetch and execute instructions. The fundamental difference between hardwired and microprog-rammed control unit is the hardwired circ-uitry approach where as the microprogram control unit is implemented by programming.

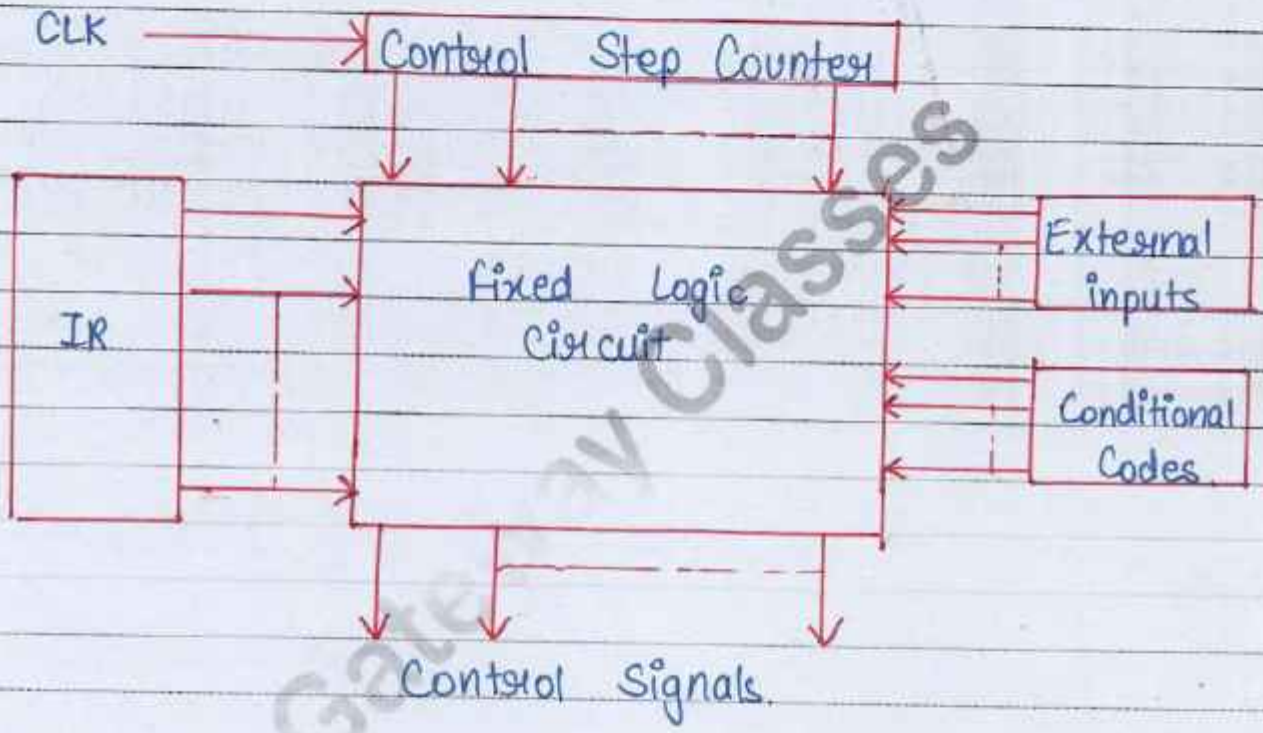
→ Hardwired Control Unit:

- The name suggests, these control units are desig-ned 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 figure below shows the architecture view of the Hardwired control unit as follows.

Block diagram of Hardwired Control unit



- 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.

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.

Disadvantages of Hardwired Control Unit:

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

- The complexity of the design increases as we require more control signals to be generate -d (need of more encoders & decoders)
- Modifications in the control signals are very diff -icult 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)

- Here, the behaviour 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.

Advantage -

- It is the simplest method.
- This method is mainly used for small instruction set processors (i.e. in RISC processors).

Disadvantage -

- 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 represented for N number of instruction. So the cost of circuitry design may increase.

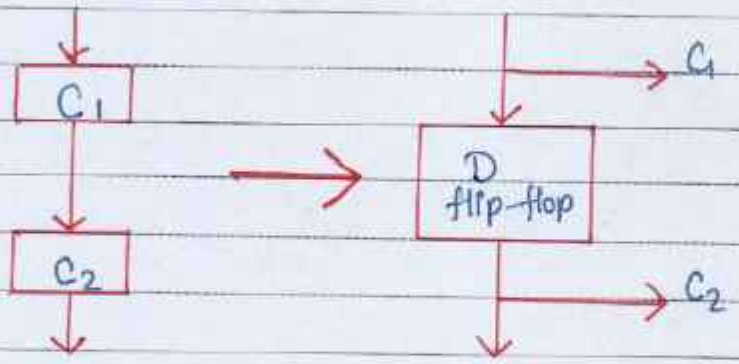
Delay Element Method.

- Here the control unit behaviour 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 perform-ed, 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 differe-nt T-state.

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

- Therefore, D flip-flops are inserted between every two consecutive control signals.

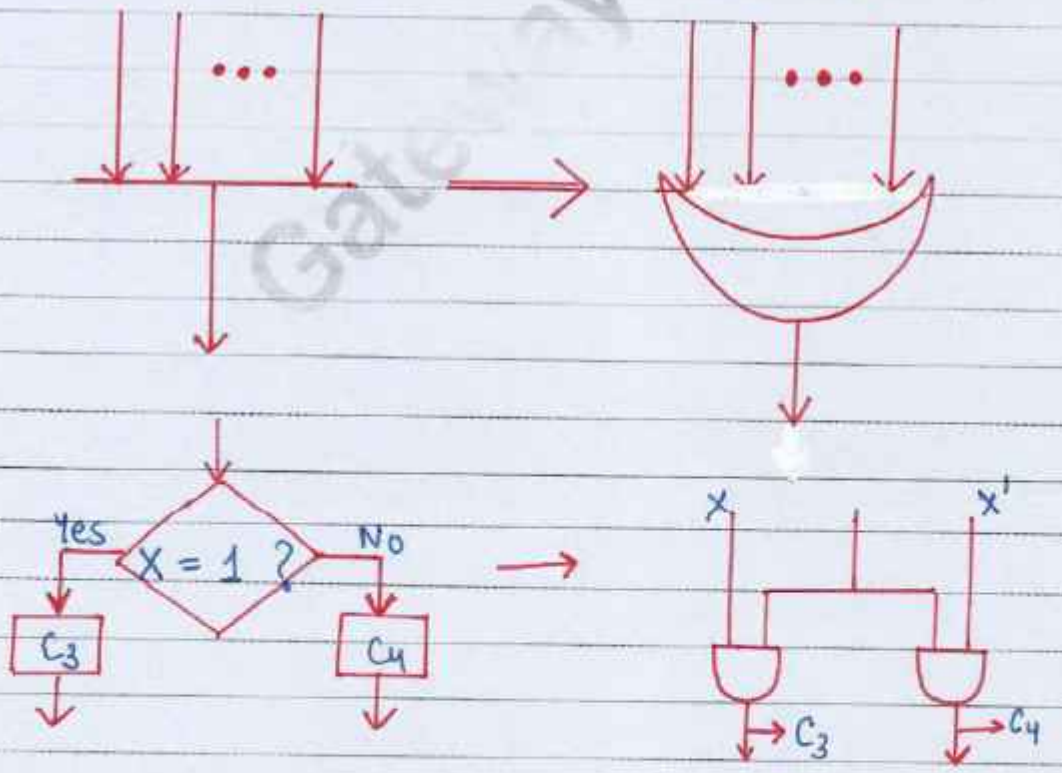


Gateway Classes

- 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.



Example -

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.

Circuit design - Between two consecutive control signals which are independent, one delay element is introduced between them to produce a delay of 1T state. The decision box is converted into and complemented AND gate circuit (ie. if $x=0$ then $x'=1$, so, a C_4 control signal is generated)

Advantage -

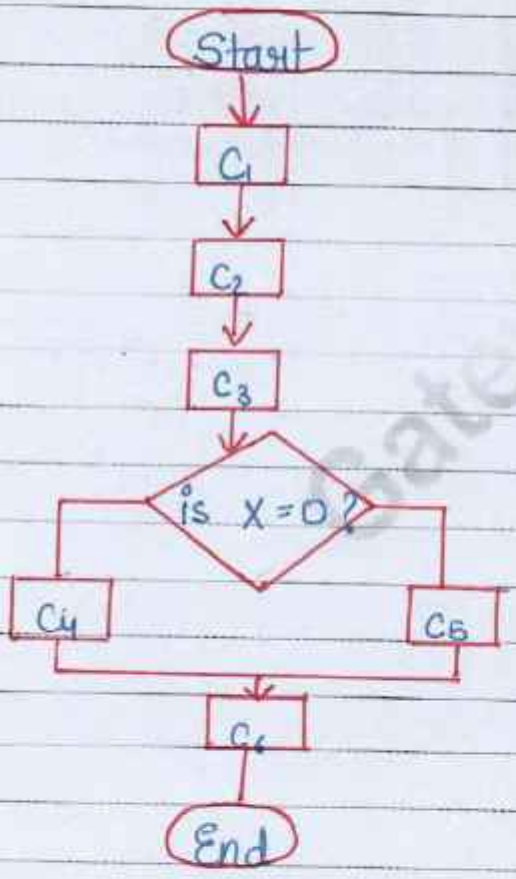
- This method has a logical approach, therefore it helps to

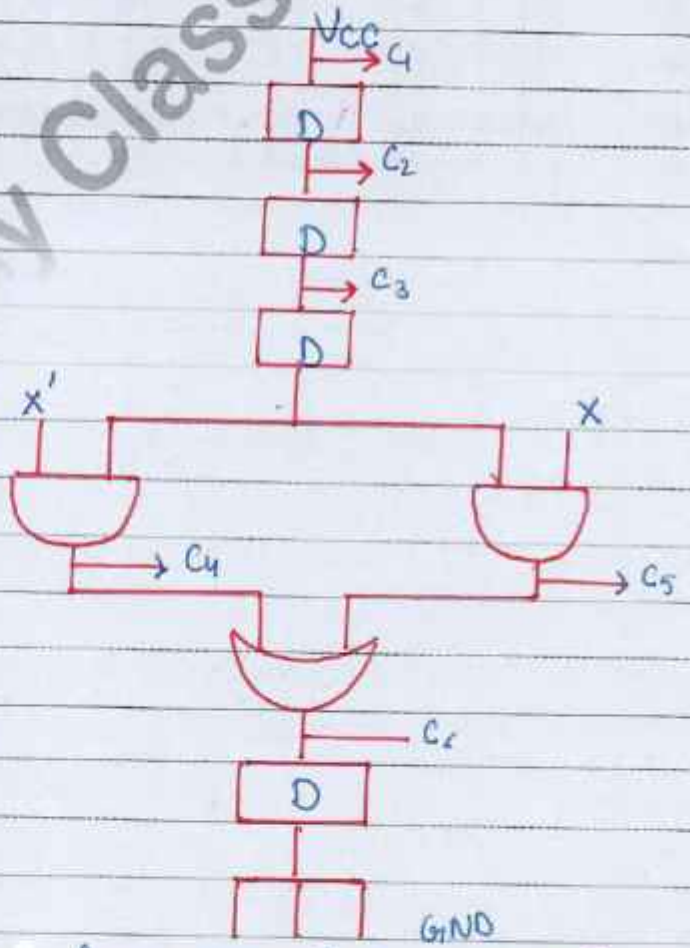
reduce the circuit complexity.

- for the control signals which need to be generated in every instruction, for them only one circuit -ry can be designed.

Drawback -

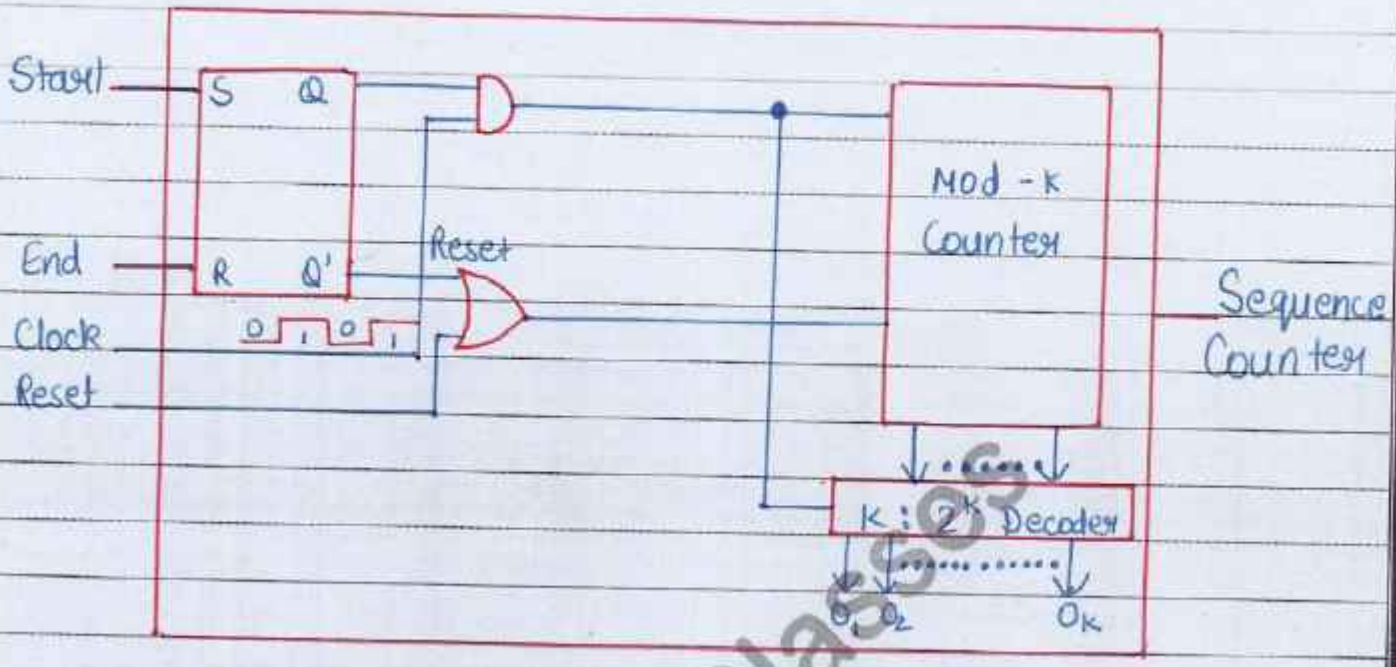
- As the number of instructions increases, the number of DFF for generating delay is increased, so overall circ -uit complexity and cost increases.





Delay element method for generating control signals.

Sequence Counter Method



- This is the most popular and most commonly used method for generating delays between every consecutive control signal.
- It's main advantage is that it uses the logical approach of flowchart and doesn't use the unnecessary number of D ff.
- First, a flowchart is designed to represent the behaviour of a control unit.
- It is then converted into a circuit using the same method of AND & OR gates (as seen above in the Delay element method).

Gateway Classes

- It is similar to the delay element method, but the only difference is that instead of unnecessary D flip flops there are triggering point in the circuit. They are activated after a gap of one-one T-state.

- When the instruction cycle starts, then start = 1.

- As we know, when start = 1, because S is connected to start, therefore Q becomes 1 and and Q' becomes 0.

- Here the level triggering clock is used. Therefore when clock = 1 or high and start = 1, as both outputs are connected to AND gate, so if the resultant of both is 1 that will enable the counter and counter starts counting from 0 0 0 state. So the 0 0 0 state is decoded by a decoder and produces output O_1, which will trigger the triggering point in the control circuit.

- As the clock becomes high again after 1 T-state. Therefore, when clock = 0, then the counter state is preserved (Q and Q') remains the same until the clock becomes high again. This makes sure that the counter changes its states after a gap of one-one T state.

- Suppose the counter is 3 bits, it generates $2^3=8$ states (000 001 ... 111). The first count 000 is given to 3:8 decoder. It will active output number 1. This output is not a control signal but this will trigger the triggering point in the control unit circuit.

- As the clock becomes high again after a gap of one T-state, therefore clock = 1 and Start = 1, then the counter is enabled and changes it state to 001 and the counter decodes the count and makes O_2 output high. And this will trigger a second triggering point in the circuit.

- All counting states are decoded in the same manner. the counter is of K bits then $K:2^K$ decoder is required this can produce 2K outputs and that will trigger the 2K triggering points after a gap of 1-1 T-states in the circuit.

- When the instruction ends, the control signal is generated to make End pin = 1, and the counter is reset, so the next time, it begins from the first count (000).

- If reset pin = 1, then the counter will reset and after that it will again start counting from 000 states.

Advantages :

- Less number of flip-flops are used.

Disadvantages of hardwired control unit :

- 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 then a large part of the circuit needs to be changed. Therefore, it is suited for RISC processors.

Programmable logic array (PLA) method.

- The external sequence register establishes the present state of the control circuit.
- The PLA output determines which micro-operations should be initiated depending on the external input conditions and the present state of the sequence register.

At the same time other PLA outputs determines the next stage of the sequence register.

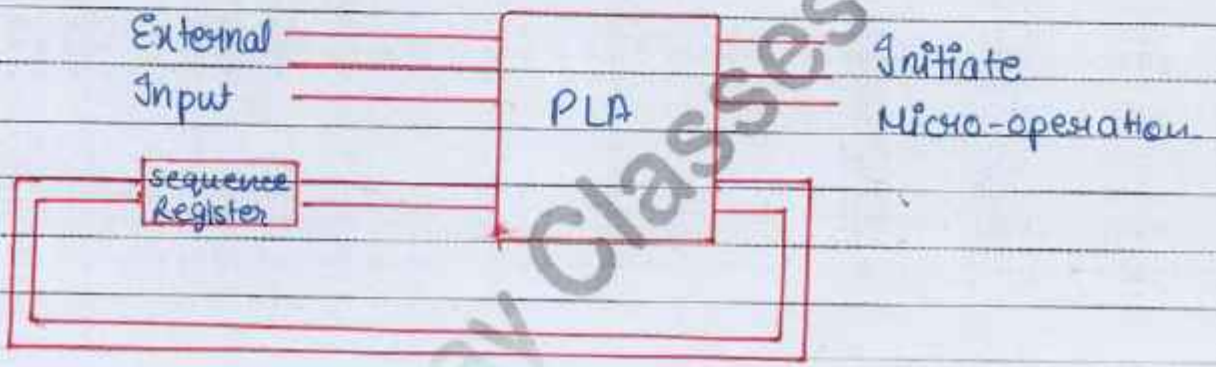
The sequence register is external to PLA.

Some PLAs include not only gate but also flip-flop

within the unit.

Note -

- A programmable Logic Array (PLA) is a type of digital logic device that can be programmed to implement custom combinational logic circuits. It consists of a fixed AND array and a programmable OR array.



- A flip-flop is a fundamental building block of sequential logic circuits. Sequential logic circuits are those in which the output depends not only on the present inputs but also on the past sequence of inputs. Flip-flops are used to store binary information and are the basic storage elements in digital systems.

AKTU Questions

Q1. Compare microinstruction and micro-program.
AKTU 2021-22

Q2. Explain the organisation of micro-programmed control unit. AKTU 2019-20.

Q3. What is address sequencing. AKTU 2019-20.

Q4. Briefly define the following terms :- Micro-operation, micro-instruction, micro-code, micro-program, control word. AKTU 2015-16, 20-21, 18-19.

Imp Q5. Explain micro-instruction format. AKTU 2017-18

Imp Q6. Difference between hardwired and microprogram-med control unit. AKTU 2014-15, 15-16, 18-19, 21-22, 22-23

Some Basic Terms

Micro programmed control unit:

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

Control Memory:

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

Writeable Control Memory:

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.
- Microinstructions are low-level instructions used in the microprogramming process of a computer's control unit. These instructions are part of the control unit's firmware and are responsible for directing the operations of the processor at a very granular level. Microprogramming is a technique that allows the design and implementation of control units to be more flexible.

Features

Control Unit Operation: Microinstructions control the internal operations of the control unit, specify how data is moved between registers, how arithmetic and logic operations are performed, and how control signals are generated.

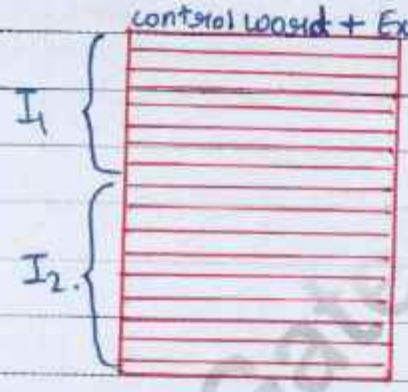
Translation from Macroinstructions: Microinstructions are generated as part of the translation process from higher-level instructions (macroinstructions) to lower-

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.

I_1 → 6 Microprogrammed
I_2 → 7 Microprogrammed



Control Memory (ROM)

control Variable :

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

level operations that the control unit can execute. This translation enables the processor to execute complex instructions efficiently.

Stored in Control Memory: Microinstructions are typically stored in a special type of memory called control memory or control store. This memory is part of the control unit.

Sequential Execution: A sequence of microinstructions is executed sequentially to perform a specific operation. Each microinstruction corresponds to a small and specific action that contributes to the overall execution of a macroinstruction.

Control Signals: Microinstructions include control signals that activate or deactivate various components of the processor, such as ALU (Arithmetic logic Unit), registers, and input / output devices.

Microprogram:

- A sequence of microinstructions constitutes a micro-program.
- Since alternations of the microprogram are not needed once the control unit is in operation.
- The control memory can be a read-only memory (Rom).

Gateway Classes

- ROM words are made permanent during the hardware production of the unit.
- The use of a micro-program involves placing all control variables in words of ROM for use by the control unit through successive read operations.
- The content of the word in ROM at a given address specifies a microinstruction.

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 programmed control unit :

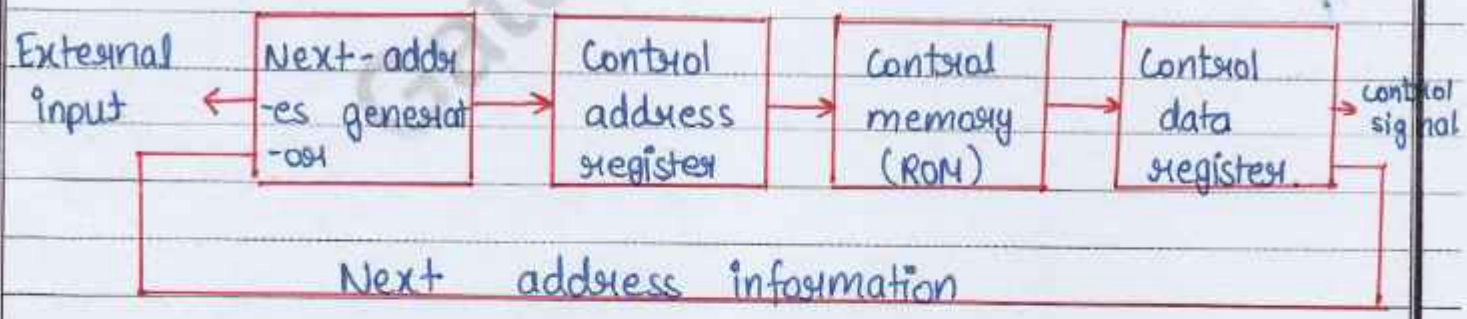
- A control unit whose binary control variables are stored in memory is called a micro programmed control unit.
- Control information is stored in control memory.

Gateway Classes

- Control memory is programmed to initiate the requir-ed sequence of micro-operations.
- The key characteristics are
- Speed of operation is low when compared with hardw-ired.
  - Less complex
  - Less expensive
  - Flexibility to add new instructions.
- Examples of CPU with microprogrammed control unit are Intel 8080, Motorola 68000 and any CISC CPUs.

Organization of micro programmed Control Unit.

Block Diagram:



- The general configuration of a micro-programmed control unit is demonstrated in the block diagram of figure.

Gateway Classes

- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control memory address (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 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 micro-instruction.
- 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, loading into the control address register an address from control memory, transf-erring an external address, or loading an initial address to start the control operations.
- The control data register holds the present microin-struction while the next address is computed and read from memory.
- The data register is sometimes called a pipeline register.
- It allows the execution of the micro-operations speci-fied by the control word simultaneously with the generation of the next microinstruction.
- This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.
- The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

- 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 controls signals for execution of register micro-operations.
- 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

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 incremen ting 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.
- The effective address computation routine in contr -ol 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 micro operation 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:

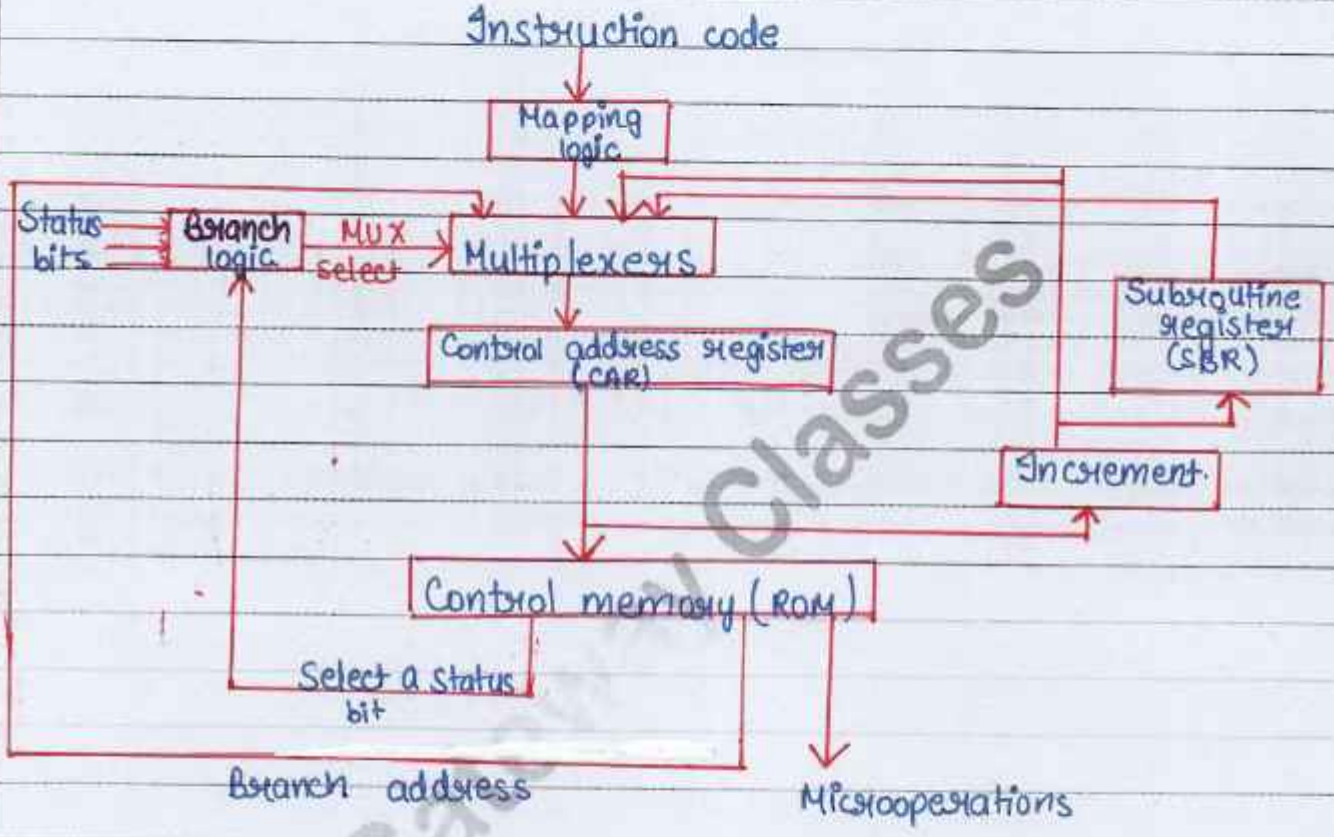
- Once the required routine is reached, the micro-instructions that execute the instruction may be sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing the return address.
- Return addresses cannot be stored in 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.



Next Microinstruction Selection

- Above figure shows a block diagram of a control memory and the associated hardware needed for selection the next microinstruction address.
- The microinstruction in control memory contains a set of bits to initiate microoperations 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 microprogram wishes to return from the subroutine.
- The branch logic of figure 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.

- A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.

- A 0 output in the multiplexer causes the address register to be incremented.

Microinstruction format –

→ The microinstruction format for the control memory is shown in the figure. The 20 bits of the microinstruction are divided into four functional parts as follows :

1. The three fields F1, F2 and F3 specify microoperations for the computer. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded

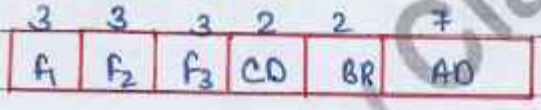
to specify seven distinct micro-operations. This gives a total of 21 micro-operations.

2. The CD field selects status bit conditions.

3. The BR field specifies the type of branch to be used.

4. The AD field contains a branch address. The address field is seven bits wide.

Microinstruction format



F1, F2, F3 : Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field.

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

Gateway Classes

COA | AKTU | CS / IT / CS Allied / MCA

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC − DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTOR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTOR

F3	Microoperation	Symbol
000	None	NOR
001	AC ← AC ⊕ DR	XOR
010	AC ← $\overline{AC}$	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

F1	F2	F3	Ex1
001	110	101	
000		000	
000	110	101	

AC ← AC + DR − Nope

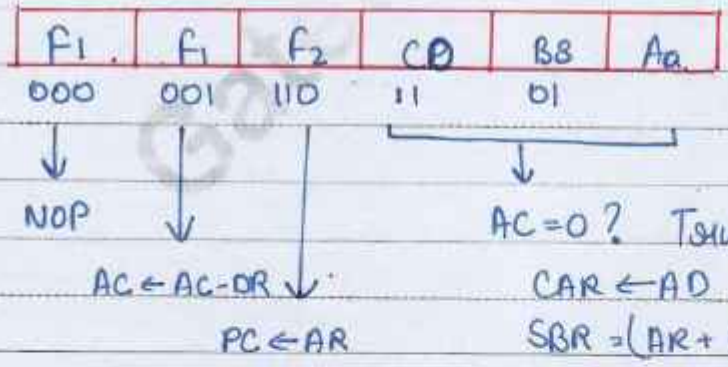
DR ← DR + 1 DR ← DR+1

P_1 ← PC + 1 → NoP

Gateway Classes

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



- The condition field (CD) is two bits to specify four status bit conditions.

The branch field (BR) consists of two bits and is used

with the address field to choose the address of the next microinstruction

Difference Between

Hardwired Control Unit	Microprogramed Control Unit
• Hardwired control unit gene-rates the control signals needed for the processor us-ing logic circuits.	• Microprogrammed control unit generates the control signals with the help of microinstru-ctions stored in control memory
• Hardwired CU is faster when compared to microprogrammed CU as the required control signals are generated with the help of hardwares.	• 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 wir-ed.	• Easy to modify as the modifi cation need to be done only at the instruction level.
• More costlier as everything has to be realized in terms of logic gates.	• less costlier than hardwired control as only micro instructio-ns are used for generating control signals.
• It cannot handle complex	• It can handle complex instructi-ons

instructions as the circuit design for it becomes complex.

- Only limited number of instruction are used due to the hardware implementation.
- Used in computers that makes use of Reduced Instruction Set Computers (RISC)
- It can use limited instructions
- Control signals for many instructions can be generated.
- Used in computers that makes use of complex Instruction Set Computers (CISC)
- It can generate control signal for many instructions.

AKTU Questions

Q1. Compare microinstruction and microprogram. AKTU 2021-22

Q2. Explain the organization of micro-programmed control unit AKTU 2019-20

Q3 What is address sequencing. AKTU 2016-17

Q4. Difference between horizontal and vertical microprogramming. AKTU 2021-22, 20-21, 19-20, 17-18.

Q5 There is an instruction pipeline with four stages. The stage delay for each stage in 5nsec, 6nsec, and 8nsec respectively. Consider the delay of

interstage register in the pipeline in 1nsec. Determine the approximate speedup of the pipeline in the steady state under ideal condition as compared to the corresponding non-pipelined implementation. AKTU 2021-22

Q6. A vertical microprogrammed control unit support 512 instructions. The system is using 8 conditio-nal 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

Types of Microprogrammed Control Unit

- The micro-programmed control unit can be classi-fied into two types based on the type of control word stored in the control Memory i.e. 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.

Horizontal 4 control signal – 4 bit 1 for each.

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$

8 control Signal.

2^3 — 3 bit needed to represent

If 31 control signal – 5 bits.

Vertical 4 control signal – $\log_2 4$

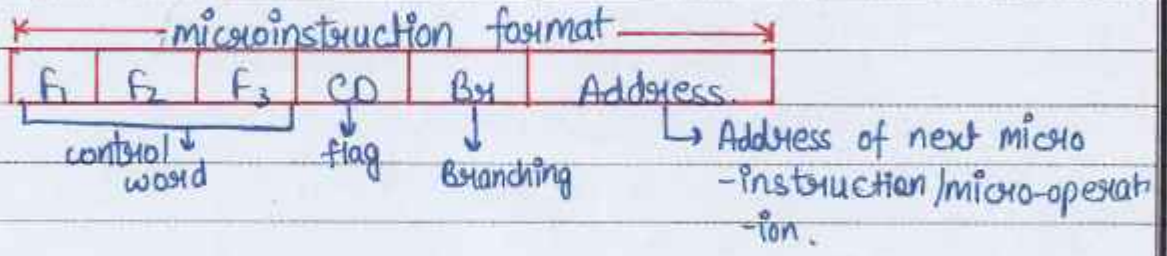
Types of Microprogrammed Control Unit

Horizontal μ-programmed CU	Vertical μ-programmed CU
• It supports longer control word.	• It supports the shorter control word.
• It allows a higher degree of parallelism. If degree is n, then n control signals are enabled at time.	• It allows a low degree of parallelism i.e, the degree of parallelism is either 0 or 1.
• No additional hardware is required.	• Additional hardware in the form of decoders is required to generate control signals.

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA

• It is faster than a vertical microprogrammed control unit.	• It is slower than Horizontal microprogrammed control unit.
• It is more flexible than a vertical microprogrammed control unit.	• It is less flexible than horizontal but more flexible than that of a hardwired control unit.
• 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.
• 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.



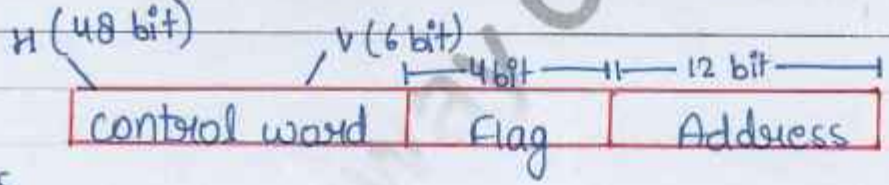
Numericals On Micro-programmed Control Unit.

Q1. Consider a micro-programmed control unit design which support 256 instruction consumes 16 micro operations on an average. The system is using 48 control signals, 16 flags must be supported for taking branching decision. What is the size of control memory (in bytes) for horizontal and vertical micro programmed control unit design respectively ?

Sol. Total instruction = 256

Each instruction consumes = 16 microoperation / 16 μ ope.

Total microoperation = $256 * 16 = 2^8 * 2^4 = 2^{12}$ μ operation



16 flags

No. of bit needed to represent 16 flags = 2^4 = 4 bit.

Horizontal

One μ-instruction size = 48 + 4 + 12 = 64 bit

Total bit needed for 2^{12} μ operation = $\frac{64 * 2^{12}}{8}$

$= 2^3 * 2^{12} = 2^{15}$ byte

Vertical

Size of Microinstruction = 6 + 4 + 12 = 22 bits

2^{12} microinstruction need bit = $2^{12} * 22$ bit

$= \frac{2^{12} * 22}{8} = \frac{2^{12} * 2 * 11}{2^3} = \frac{2^{10} * 2^3 * 11}{2^3} = 11$ KB

Q2. Consider a hypothetical control unit that support 10 mutually exclusive group of control signals show below how many number of bits are saved using vertical micro-programming over the horizontal microprogramming.

Sol.

group	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
control signals	3	7	11	14	6	12	5	3

Horizontal

No of bit = 3 + 7 + 11 + 14 + 6 + 12 + 5 + 3 + 3 + 7 = 71 bits

Vertical

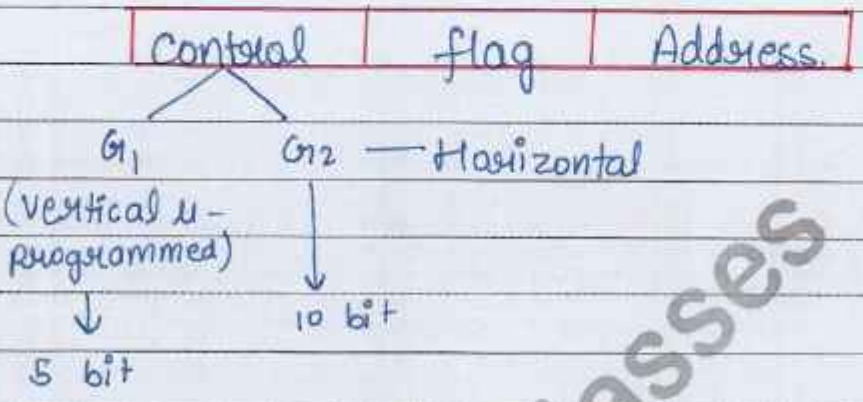
group	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
control signals	3	7	11	14	6	12	5	3	3	7

No of bit = 2 + 3 + 4 + 4 + 3 + 4 + 3 + 2 + 2 + 3 = 30 bit

No of bit saved using vertical μ programmed unit over horizontal one = 71 - 30 = 41 bits.

Q3. Consider a hypothetical control unit that supports 2 groups of control signal group 1 indicate none or one 30 control signal group 2 indicate 10 almost control signal what is size of control word or signal.

Sol.



Control word size = $G_1 + G_{12}$ = 5 + 10 = 15 bit

Q4. A vertical micro programmed control unit suppo-rt 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 approx. size of memory in bytes. (AKTU 2021-22)

Sol. Vertical

No. of instruction = 512 8 conditional flag

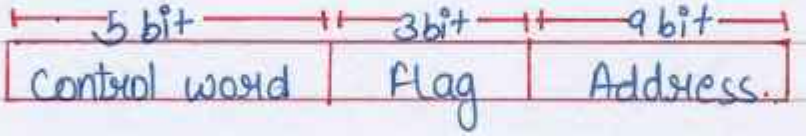
Each instruction = 1 uoperation 31 control signal

No. of bit needed to represent 8 conditional flag = 2^3 = 3 bit.

Total u operation = 512 * 1 = 512 u-operation = 2^9 u-ope.

No. of bit needed to represent 512 μ-operation
$= 2^9 = 9$ bits.

31 control signal need bits = 5 bits



No of bits to represent one μ-instruction = 5+3+9 = 17bit
No. of bits to represent 2^9 μ-operation = $2^9 * 17$ bit.
$= \frac{2^9 \times 17}{8}$ byte $= \frac{2^9 * 17}{2^3} = 2^6 * 17 = 64 * 17$.
= 1088 Byte (Size of control memory)

Pipeline formulas and Concepts

Performance of Pipelined Execution -

The following parameters serve as criterion to estimate the performance of pipelined execution- speed Up., Efficiency and throughput.

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

Efficiency (η) = $\frac{\text{Speed Up}}{\text{No. of stages in Pipelined Architecture}}$

Throughput = $\frac{\text{No. of instructions executed}}{\text{Total time taken}}$

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.

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. Thus,
- 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 parallelly.

- No. of clock cycles taken by the first instruction = k clock cycles.
- After first instruction has compeletely executed, one instruction comes out per clock cycle.
- So, number of clock cycles taken by each remaining instruction.
- = 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.

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, $n \to \infty$.
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.

Numericals in Pipeline

Q1. 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.

Sol. (i) Pipeline cycletime = 1 clock time.
Max of any stage + Register or latch delay.
Max {60, 50, 90, 80} + 10
90 ns + 10 ns = 100 ns.
Pipeline cycle = 100 ns

(ii) Non pipeline Execution time for 1 instruction.
= time to go through All stages.
= 60 + 50 + 90 + 80 = 280 ns.

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

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

efficiency = $\frac{\text{Speed Up}}{\text{No. of stages in pipelined}} = \frac{2.8}{4} = 0.7.$

(iv) Pipeline time for 1000 task.

first instruction + Remaining
K * cycletime + n-1 * cycletime
$\boxed{(K+n-1) * \text{cycletime.}}$
$(4+1000-1) * 100$
$1003 * 1000 = 100300$

- Non-pipelined Execution

Execution time = one instruction execution time * no. of instruction
$= 280 * 1000 = 280000$ ns.

- throughput (pipeline) = $\frac{\text{No. of instruction}}{\text{Total time}} = \frac{1000}{100300}$

Q2. A four stage pipeline has the stage delays as 150, 120, 160 and 140 ns respectively. Registers are used between the stages and have a delay of

5 ns each. Assuming constant clocking rate, the total time taken to process 1000 data items on the pipeline will be.

Sol. cycle time of pipeline execution time = Maximum of any stage + Register delay
= Max { 150, 120, 160, 140 } + 5
= 160 + 5 = 165 ns

Total time taken to process 1000 instruction.
= (K + n - 1) * cycle time = (4 + 1000 - 1) * 165
= 1003 * 165 = 165495 ns
= 165.495 x 10^{-3} ns = 165.495 μs.

Q3. Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycle per instruction of 4. The same processor is upgraded to a pipeline processor with five stages but due to internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume there is no stalls in the pipeline. The speed up achieved in this pipelined processor is -
Note since there is no stall in the pipeline, so ideally one instruction is executed per clock cycle.

Sol. Non-pipelined cycle time = $\frac{1}{f}$ = $\frac{1}{2.5}$ gigahertz = $\frac{1}{2.5 * 10^9}$ sec

$= \frac{1}{2.5} * 10^{-9}$ sec $= 0.4$ ns.

Non-pipeline time for executing on c instruction.
$= 0.4 * 4 = 1.6$ ns

Pipeline cycle $= \frac{1}{f} = \frac{1}{2 \text{ gigahertz}} = \frac{1}{2 * 10^9 \text{ Hertz}}$

$= 0.5 * 10^{-9}$ s $= 0.5$ ns

time for execution one instruction = 1 cycle time
$= 1 * 0.5$ ns $= 0.5$ ns.

Speed Up $= \frac{\text{Non-pipeline Execution}}{\text{Pipeline Execution}} = \frac{1.6 \text{ ns}}{0.5 \text{ ns}} = 3.2.$

Qy. Non-pipelined single cycle processor operating at 100 MHz is converted into a synchronous pipelined processor with five stages requiring 2.5 ns, 1.5 ns, 2 ns, 1.5 ns and 2.5 ns respectively. The delay of the latches is 0.5 sec.
The speed up of the pipeline processor for a large number of instructions is.

Sol. Non-pipeline,
frequency = 100 MHz.
Cycle $= \frac{1}{f} = \frac{1}{100 \text{ MHz}} = \frac{1}{100\ 10^6} \text{ s} = \frac{1}{100} 10^{-6} \text{ s} =$

$= \frac{1}{100} * 10^{-6} * 10^{6}$ μs $= 0.01$ μs

1 sec $= 10^6$ μs

0.01 μs $= \frac{0.01 \times 10^{-6} * 10}{10^2 * 10} = 10^{-9} * 10 = 10$ ns

Execution time for 1 instruction

$\frac{1}{100} * 10^{-6} = \frac{10^{-8} * 10}{10} = 10$ ns

Pipeline

Max { 2.5, 1.5, 1.5, 2.5, 2 } + 0.5

2.5 + 0.5 = 3ns

Execution time = 1 cycle time = 3ns.

speed up = $\frac{\text{Non-pipeline Execution}}{\text{Pipeline execution time}} = \frac{10 \text{ ns}}{3 \text{ ns}} = 3.33$

Q5. We have 2 designs D1 and D2 for a synchronous pipeline processor. D1 has 5 stage pipeline with execution time of 3ns, 2ns, 4ns, 2ns and 3ns. While the design D2 has 8 pipeline stages each with 2ns execution time. How much time can be saved using design D2 over design D1 for executing 100 instructions?

214 ns, 202 ns, 86ns, 200ns.

Sol. For D1 - cycle time = Max { 3, 4, 2, 2, 3 } + 0 = 4ns

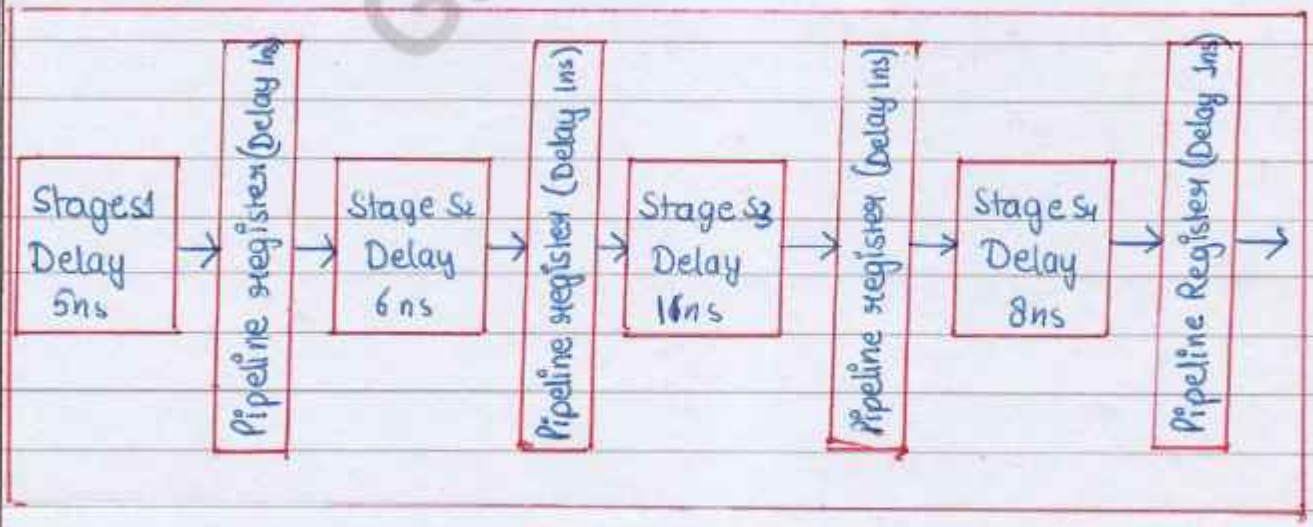
Executing time = (k + n - 1) * cycle time

$= (5+100-1) * 4 = 104 * 4 = 416$ ns
$(k=5, n=100)$

For D2 - cycle time = 2ns + 0 = 2ns [k=8]
Execution time for Executing 100 instruction [n = 100]
$= (k+n-1) *$ cycle time $= (8+100-1) * 2$
$= 107 * 2 = 214$ ns

Total time save using design D_2 over D_1 = 416 - 214 = 202 ns

Q4 Consider an instruction pipeline with four stages (S_1, S_2, S_3 and S_4) each with combinational circuit only. The pipeline registers are required between each stage and at the end of the last stage. Delays for the stages and for the pipeline registers are as given in figure. What is the approx. speed up of the pipeline in steady state under ideal conditions when compared to the corresponding non-pipeline implementation?



Sol. Pipeline - Cycle time = Max of any stage + Register
= Max {5, 6, 11, 8} + 1ns
= 11ns + 1ns = 12ns
Pipeline execution time = 1 clock cycle = 12ns

Non-Pipeline - For execution one instruction
(Time taken) = 5 + 6 + 11 + 8 = 30ns

Speed Up - $\frac{\text{Non-pipeline execution}}{\text{pipeline execution}} = \frac{30 \text{ ns}}{12 \text{ ns}} = \frac{5}{4} = 1.25$

Qz. The stage delays in a 4 stage pipeline are 800, 500, 400 and 300 picoseconds. The first stage is replaced with a functionally equivalent design involving two stages with respective delays 600 and 350 picoseconds. The throughput increase of the pipeline is.

Sol. 4 stage -
Cycle time = Maximum of any stage + Register delay
= {800, 500, 400, 300} + 0 = 800 picoseconds
Execution time one instruction = $\frac{\text{No of instruction}}{\text{Time taken}} = \frac{1}{100 \text{ ps}}$

2 stage -
Cycle = Max of any stage + Register delay
= Max {600 + 350} + 0 = 600 ps.
Execution time of 1 instruction = 1 clock cycle = 600 ps
Throughput = $\frac{1}{600}$ ps

Throughput = $\frac{\text{Final} - \text{Initial}}{\text{Initial}} * 100$

$= \frac{\frac{1}{600} - \frac{1}{800}}{\frac{1}{800}} * 100 = \left(\frac{800}{600} - 1\right) * 100$

$= (1.333 - 1) * 100 = 0.3333 * 100 = 33.33\%$

Q8. There is an instruction pipeline with four stages. The stage delay for each stage is 5nsec, 6nsec, 11nsec, and 8nsec respectively. Consider the delay of inter stage register in the pipeline is 5 nsec. Determine the approx speed up of the pipeline in the steady state under ideal condition as compared to the corresponding non-pipelined implementation.

Sol. Speed Up = $\frac{\text{Non pipeline execution}}{\text{Pipeline execution}} = \frac{30}{15} = 2.$

where, Non-pipeline -
for executing one-instruction = 5+6+11+8 = 30ns

Pipeline cycle = 11+5 = 15ns.

AKTU Questions

Q1. Difference between horizontal and vertical microprogramming. AKTU 2021-22, 20-21, 19-20, 17-18

Q2. There is an instruction pipeline with four stages. The stage delay for each stage is 5nsec, 6nsec, 11 nsec, and 8nsec respectively. Consider the delay of inter stage register in the pipeline is 1 nsec. Determine the approx speedup of the pipeline in the steady state under ideal condition as compared to the corresponding non-pipelined implementation. AKTU 2021-22

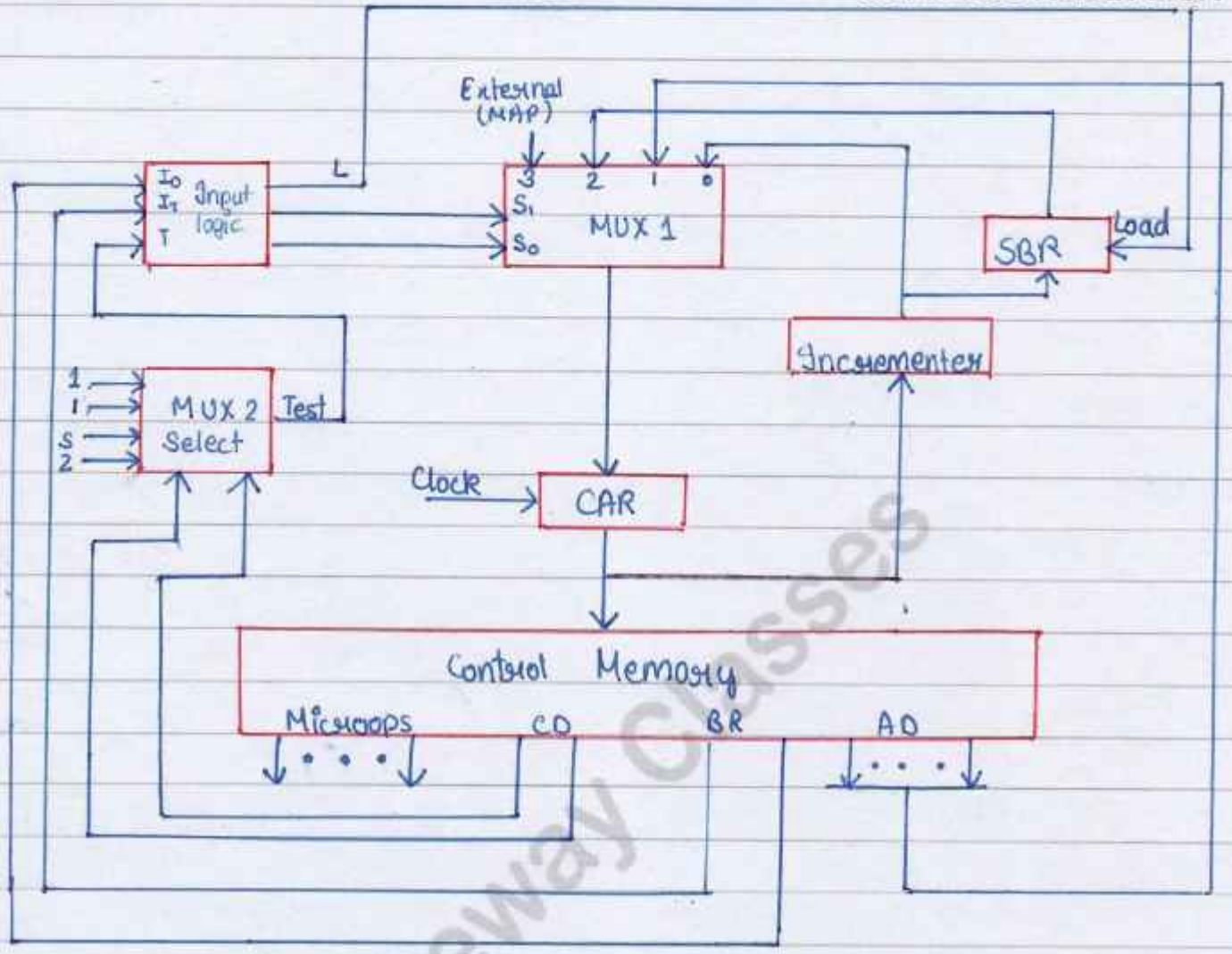
Q3. A vertical micro programmed control unit support 512 instructions. The system is using 8 conditional flag and contain 31 control signal. Each instruction on an average has 1 micro operation. Calculate the approx. size of memory in bytes. AKTU 2021-22

Q4. Explain the working of microprogram sequencer with suitable diagram and tables. AKTU 2021-22, 20-21, 18-19

Q5. Explain instruction format + previous all numericals. AKTU 2021-22, 15-16, 20-21, 2

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA



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

Microprogram Sequencer

BR FIELD		Input I_0	Input I_1	Input T	MUX 1 S_0	MUX 1 S_1	Load SBR I
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

BR TABLE

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

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.
- 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 second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit. The first multiplexer selects an address from one of four

sources and routes it into control address register CAR.

- 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.
- 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 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 determines 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 6.
- The type of operations that are available in the unit.
- 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 micro-instruction (BR = 01) provided that the status bit condition is satisfied (T = 1).

Infix and Postfix

(a+b+c) * (c+d)) infix expression

(ab+c+) * (cd+)

ab+c+cd+* postfix expression

= (a+b+c) * (c+d)
= ((ab+)+c) * (c+d)
= (ab+c+) * (c+d)
= (ab+c+) * (cd+)
ab+c+cd+*

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.

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 it into reverse polish notation.

Mode	Opcode

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

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.

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.

Mode	Opcode	Operand

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

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

- All operations are done between the accumulator (AC) register and a memory operand.
- M[ ] is any memory location.
- M[T] addresses a temporary memory location for storing the intermediate result.
- This instruction format has only one operand field. This addresse field uses two special instructions to perform data transfer, namely:
- LOAD: This is used to transfer the data to the accumulator.
- STORE: This is used to move the data from the

accumulator.

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

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.

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	Operand 1	Operand 2.

- This is common in commercial computers.

Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address.

Mode	Opcode	Destination address	Source address

- Example: The program to evaluate $X = (A+B) * (C+D)$ is as follows.

MOV R_1, A	$R_1 \leftarrow M[A]$
ADD R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV R_2, C	$R_2 \leftarrow M[C]$
ADD R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV X, R_1	$M[X] \leftarrow R_1$

The MOV instruction transfers the operands to the memory from the processor registers. R_1, R_2.

Advantages:

They allow for more complex operations and can be more efficient than 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

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	Operand 1	operand 2	Operand 3

- This has three address field to specify a register or a memory location.
- Program created are much short in size but number of bits per instruction increases.
- These instructions make creation of program much

easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation will be performed in one cycle only

Mode	Opcode	Destination address	source address	Source address

ADD R_1, A, B	$R_1 = M[A] + M[B]$
ADD R_2, C, D	$R_2 = M[C] + M[D]$
MUL X, R_1, R_2	$M[X] = R_1 * R_2$

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, executio

-on time, and flexibility.

Complete Execution of Instruction.

ADD R_1, (R_3) $R_1 \leftarrow R_1 + M[(R_3)]$

1. Fetch the instruction.
2. Fetch the operand.
3. Perform the addition.
4. Load the result in R_1

Step	Action
1	PCout, MARin, Read, Select 4, Add, Zin
2	Zout, PCin, Yin, WMF C
3	MDRout, IRin
4	R3out, MARin, Read
5	R1out, Yin, WMF C
6	MDRout, Select Y, Add, Zin.
7	Zout, Rin, End

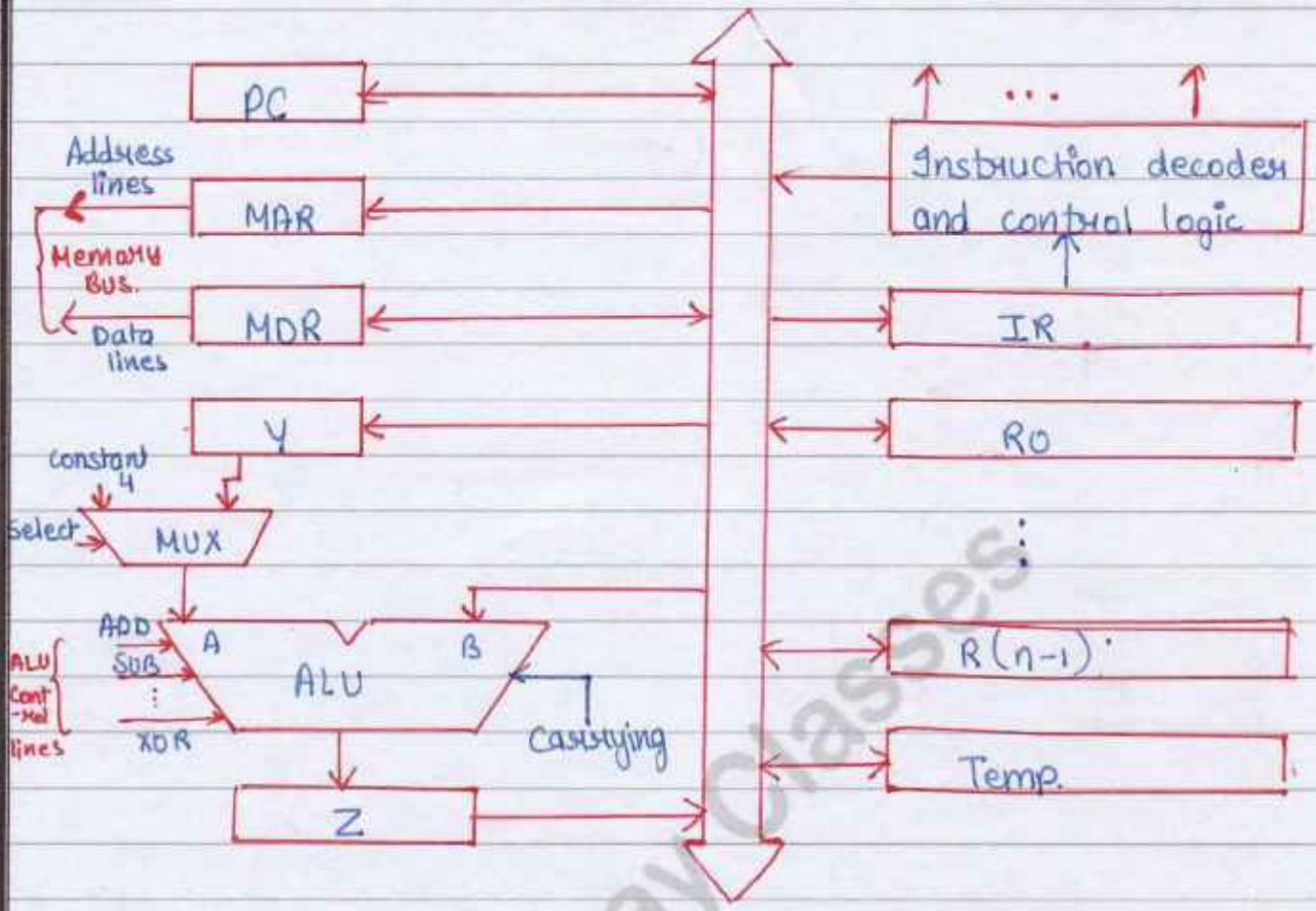
Pcout, MARin, Read, Select4, ADD, Zin

- Load the content of the PC into MAR and send a rea -d request.
- Pcout, MARin, Read
- While waiting for a response. Increment PC

- Select constant 4 in MUX
- ALU input B is receiving the current value in PC
- Specify add operation
- In step 2, move updated value back into PC and wait MFC (Z_{out}, P_{cin}, WMFC)
- In step 3, the word fetch from memory is loaded in IR (MDR_{OUT}, IR_{IN}).
- Step 4 And 5
- Fetch the operand : the content of the memory location pointed to by R3.
- R3out, MARin, Read.
- R1out, Yin, WMFC
- Step 6
- Perform the Addition
- MDRin, Select Y Add Zin
- Step 7: Load the result in R1
- Zout, R1in, End.

Gateway Classes

COA AKTU CS / IT / CS Allied / MCA



AKTU Questions

Q1. Explain the working of microprogram sequencer with suitable diagram and tables. AKTU 2021-22, 20-21, 18-19.

Q2. Explain instruction format + numerical AKTU 21-22, 15-16 20-21.

AKTU

B. Tech II-Year

Gateway Classes

Full Courses Available in App

AKTU B.Tech I- Year : All Branches

AKTU B.Tech II- Year

Branches : 1. CS IT & CS Allied

2. EC & EC Allied

3. ME & ME Allied

4. EE & EE Allied

Downlod App Now

Download App

Full Courses

- V. Lectures
- Pdf Notes
- AKTU PYQs
- DPP