20/11/23

Unit :- 03

Control Unit

Instruction Cycle:- An instruction cycle involves three sub-cycles:-

1. Fetch:- The fetch phase reads the instruction from memory into the CPU.
2. Decode:- The decode phase interprets the opcode by decoding it.
3. Execute:- The execute phase performs the indicated operation.

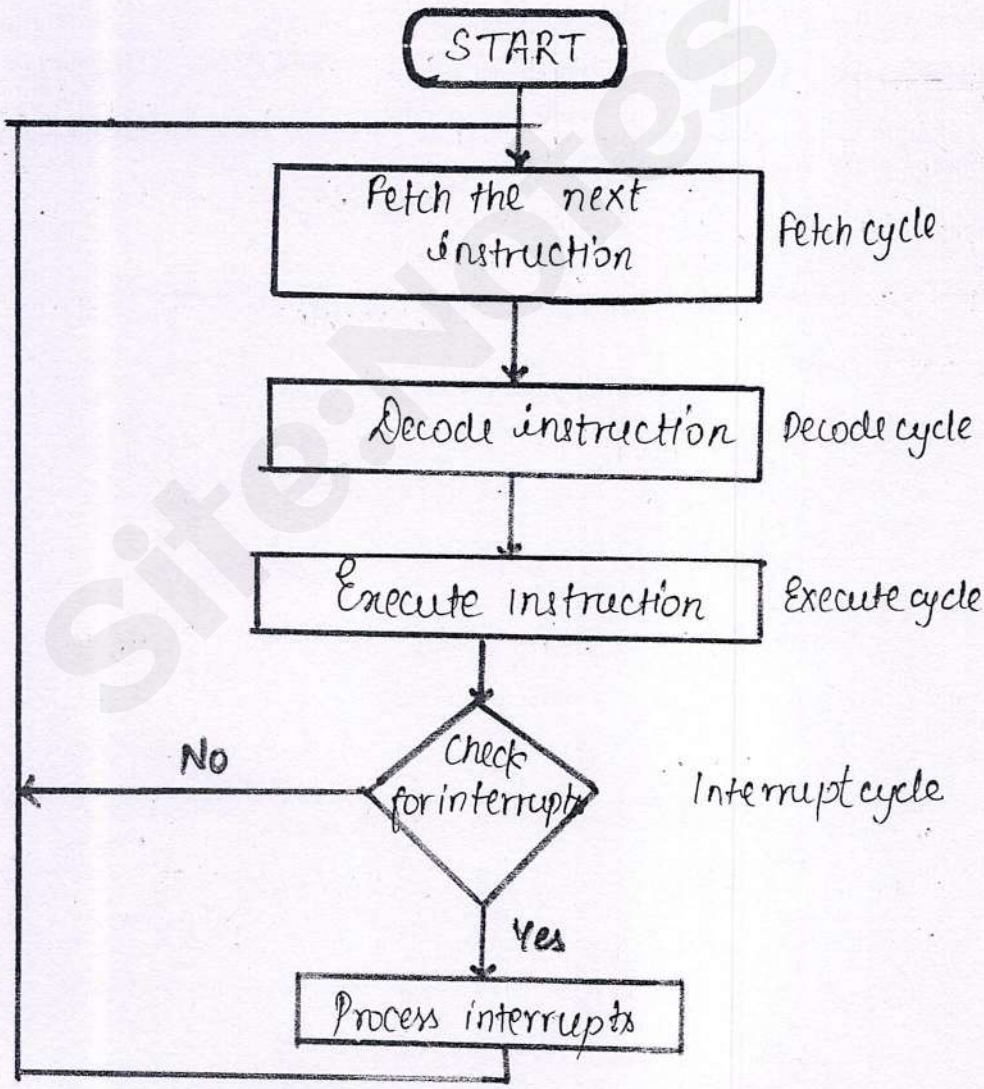


Fig 3.2 Basic instruction cycle with interrupt cycle

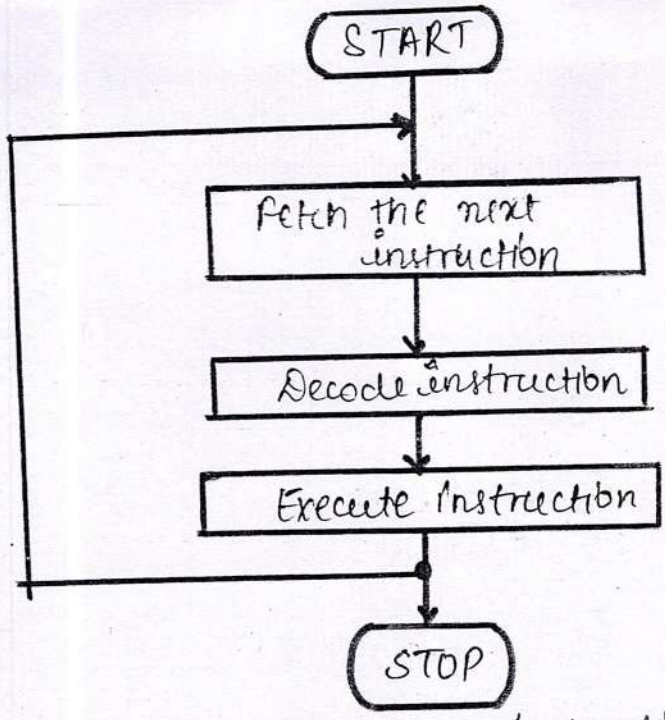
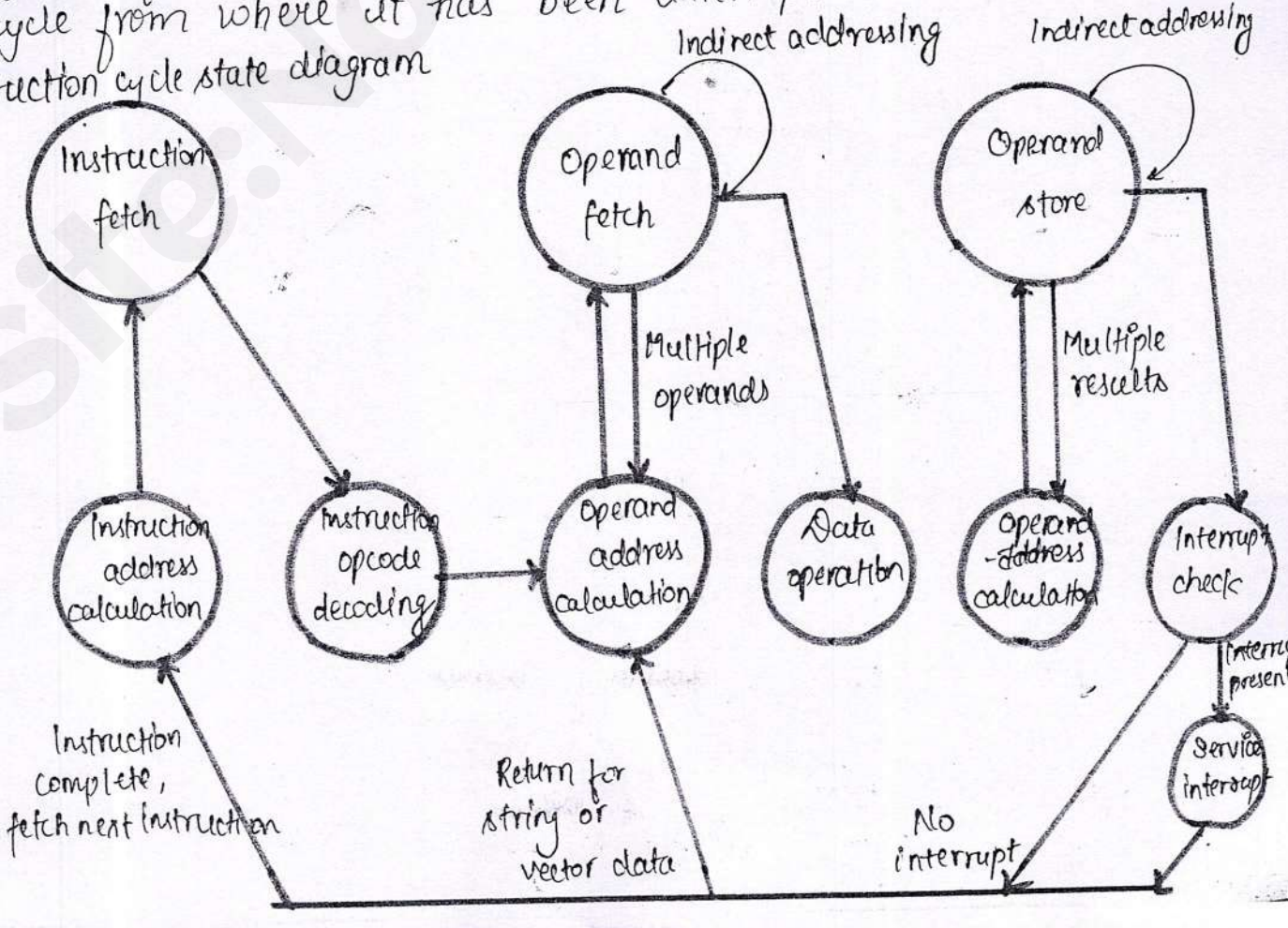


Fig. 3.1 Basic Instruction cycle

Processor checks for valid interrupt request. After each, each instruction cycle. If any valid interrupt request is present, processor saves the current process state and service the interrupt. Servicing the interrupt means executing interrupt cycle. After completing it, processor starts the new instruction cycle from where it has been interrupted.

*Instruction cycle state diagram



Instruction format: Various fields in instruction format are:-

(i) Opcode:- The operation code field in the instruction specify the operation to be performed. The operation code is specified by binary code, hence the name opcode.

(ii) Source/Destination operands:- The source operand may be in CPU register or in the memory. Many times the instruction specifies the address of the source operand so that operands can be accessed and operated by the CPU according to the instruction.
The operands on which we have to perform opcode are known as source operands.
The operation executed by the CPU may produce result. Most of the times the result is stored in one of the operands. Such operand is known as destination operand.

(iii) Next Instruction Address:- The next instruction address tells the CPU from where to fetch the next instruction after completion of current execution.

Various types of Instruction format

(i) 3- address instruction format.
(ii) 2- address instruction format.
(iii) 1- address instruction format
(iv) 0- address instruction format

Three-Address Instruction: Three address instruction can be represented symbolically as:-

ADD A, B, C.

where A, B, C are the variables. These variable names are assigned to distinct locations in the memory. In this instruction, operand B and C are called source operands and A is called

destination operand and ADD is the operation to be performed on the operands. Thus, the general instruction format for three address instruction is:-

OPCODE, Destination operand, Source operand 1, Source op. 2

If n bits are required to specify one memory address, ~~three~~ 3n bits are required to specify these operands and 1 bit required to specify microoperation.

Example:- WAP to evaluate the arithmetic statement Y=(A+B)*(C+D) by using 3-address instruction format.

ADD R_1 A,B	$R_1 \leftarrow m[A] + m[B]$
ADD R_2 C,D	$R_2 \leftarrow m[C] + m[D]$
MUL Y. R_1, R_2	$m[Y] \leftarrow R_1 * R_2$

2. **Two-Address Instruction:** Two-address instruction can be represented symbolically as:-

ADD A,B.

This instruction adds the content of variable A and B and stores the sum in variable A destroying its previous contents. Here operand B is source operand and operand A it serves as both source and destination operand. The general instruction format as follows:-

opcode, source operand, destination operand.

Example:- Y=(A+B) * (C+D) using 2-add. instruction format.

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 Y R_1	$m[Y] \leftarrow R_1$

3. One-address instruction format

The one address instruction can be represented symbolically as-

ADD B.

This instruction adds the content of variable B into the processor register called accumulator and stores the sum back into the accumulator destroying the previous contents of the accumulator. In this instruction, the second operand is assumed into implicitally in a unique location accumulator. The general instruction format for one address as instruction is:-

operation, source.

In one add. instruction we use load and source instruction type:-

Eg:- $Y = (A+B) * (C+D)$

```
LOAD  A
ADD   B
STORE P
LOAD  C
ADD   D
MUL   P
STORE Y
```

$AC \leftarrow m[A]$

$AC \leftarrow AC + m[B]$

$m[P] \leftarrow AC$

$AC \leftarrow m[C]$

$AC \leftarrow AC + m[D]$

$AC \leftarrow AC * P$

$m[Y] \leftarrow AC$

4. Zero address instruction

In these instruction, the location of all operands are defined implicitly. Such instructions are found in a machine that store operands in a structure called a pushdown stack.

Eg. $Y = (A+B) * C+D$

Postfix:- AB+CD+*

PUSH A	TOS ← m[A]
PUSH B	TOS ← m[B]
ADD	TOS ← m[A] + m[B]
PUSH C	TOS ← m[C]
PUSH D	TOS ← m[D]
ADD	TOS ← m[C] + m[D]
MUL	TOS ← m[A] + m[B] * m[C] + m[D]
POP Y	m[Y] ← TOS

WAP to evaluate using 3,2,1,0 address instruction format

$$Y = \frac{A[B+C(D+E)]}{F(G+H)}$$

3-address Instruction:-

ADD R_1, D, E	$R_1 \leftarrow m[D] + m[E]$
MUL R_2, C, R_1	$R_2 \leftarrow m[C] * R_1$
ADD R_3, R_2, B	$R_3 \leftarrow m[B] + R_2$
MUL R_4, A, R_3	$R_4 \leftarrow m[A] * R_3$
ADD R_5, G, H	$R_5 \leftarrow m[G] + m[H]$
MUL R_6, F, R_5	$R_6 \leftarrow m[F] * R_5$
DIV X, R_4, R_6	$m[X] \leftarrow R_4 / R_5$

2-address Instruction:-

MOV R_1, D	$R_1 \leftarrow m[D]$
ADD R_1, E	$R_1 \leftarrow R_1 + m[E]$
MUL R_1, C	$R_1 \leftarrow R_1 * m[C]$
ADD R_1, B	$R_1 \leftarrow R_1 + m[B]$
MUL R_1, A	$R_1 \leftarrow R_1 * m[A]$
MOV R_2, H	$R_2 \leftarrow m[H]$

ADD R_2, G	$R_2 \leftarrow R_2 + m[G]$
MUL R_2, F	$R_2 \leftarrow R_2 * m[F]$
DIV R_1, R_2	$R_1 \leftarrow R_1 / R_2$
MOV X, R_1	$m[X] \leftarrow R_1$

1-address Instruction

LOAD H	$AC \leftarrow m[G]$
ADD G	$AC \leftarrow AC + m[H]$
MUL F	$AC \leftarrow AC * m[F]$
STORE P	$m[P] \leftarrow AC$
LOAD E	$AC \leftarrow m[D]$
ADD D	$AC \leftarrow AC + m[E]$
MUL C	$AC \leftarrow AC * m[C]$
ADD B	$AC \leftarrow AC + m[B]$
MUL A	$AC \leftarrow AC * m[A]$
DIV P	$AC \leftarrow AC / m[P]$
STORE X	$X \leftarrow AC$

0-address Instruction

Postfix :- ABCDE+*+*FGH+*/

PUSH A	$TOS \leftarrow m[A]$
PUSH B	$TOS \leftarrow m[B]$
PUSH C	$TOS \leftarrow m[C]$
PUSH D	$TOS \leftarrow m[D]$
PUSH E	$TOS \leftarrow m[E]$
ADD	$TOS \leftarrow m[D] + m[E]$
MUL	$TOS \leftarrow m[D] + m[E] * m[C]$
ADD	$TOS \leftarrow m[D] + m[E] * m[C] + m[B]$
MUL	$TOS \leftarrow m[D] + m[E] * m[C] + m[B] * m[A]$
PUSH F	$TOS \leftarrow m[F]$
PUSH G	$TOS \leftarrow m[G]$
PUSH H	$TOS \leftarrow m[H]$

ADD	TOS ← m[G] + m[H]
MUL	TOS ← m[G] + m[H] * m[F]
DIV	TOS ← m[D] + m[E] * m[C] + m[B] * m[A] / m[G] + m[H] * m[E]
POPX	m[X] ← TOS

WAP to evaluate $X = (A+B*C)/(D-E/F)$ using all address instructions.

3-address instruction:-

```
MUL  R1, B, C
ADD  R2, R1, A
DIV  R3, E, F
SUB  R4, D, R3
DIV  X, R2, R4
```

2-address instruction:-

```
MOV  R1, C
MUL  R1, B
ADD  R1, A
MOV  R2, E
DIV  R2, F
MOV  R3, D
SUB  R3, R2
DIV  R1, R3
MOV  X, R1
```

1-address instruction:-

```
LOAD E
DIV F
STORE R
LOAD D
SUB R
LOAD C
MUL B
ADD A
DIV R
STORE X
```

0-address instruction:-

Postfix:- ABC * + DEF / - /

```
PUSH A
PUSH B
PUSH C
MUL
ADD
PUSH D
PUSH E
PUSH F
DIV
SUB
DIV
POP X.
```

Instructions type:- A computer has a set of instructions which can be classified as:-

1. Data Transfer Instruction
2. Arithmetic Instructions
3. Logical Instructions
4. Shift and rotate instructions
5. Program Control Instruction

1. Data Transfer Instruction:- Data transfer instructions perform the following data transfer operations:-

- Data transfer between memory and CPU register.
- Data transfer blw CPU and registers.
- Data transfer blw processor and input/output devices.

Instruction	Mnemonic	Description
Load	LD	It transfers data from specified memory location to the processor register, usually accumulator.
Store	ST	It transfers data from processor register (usually accumulator) to the specified memory location.
Move	MOV	It transfers data between processor registers and memory or between two memory words
Exchange	XCH	It swaps data blw two registers or a register and a memory word.
POP	POP	It transfers data from stack memory (Top of stack) to the processor registers.

2. Arithmetic Instructions :-

Instruction	Mnemonic	Description
Add	ADD	It adds the contents of two operands.
subtract	SUB	It subtracts the contents of two operand
Increment	INC	It adds 1 to the value stored in a register or memory word.
Decrement	DEC	It subtracts 1 from the value stored in a register or memory word.
Multiply	MUL	It multiplies the contents of two operands.

3. Logical Instructions :-

Instruction	Mnemonic	Description
AND	AND	It logically ANDs the individual bits of the operands.
OR	OR	It logically ORs the individual bits of the operands.
Clear	CLR	It causes the specified operand to be replaced by 0's.
Complement	COM	It gives 1's complement of the specified operand.
Set carry	STC	It sets the carry bit to 1.
Clear carry	CLC	It resets the carry bit to 0.

4. Shift and Rotate Instructions

Instruction	Mnemonic	Description
Logical shift left	SHL	It shifts the contents of the specified register 1-bit position towards left and fills vacant bit (rightmost) with 0.
Logical shift right	SHR	It shifts the contents of the specified register 1-bit position towards right and fills vacant bit (leftmost) with 0.
Rotate right	ROR	It rotates (circular-shifts) right the contents of specified register
Rotate left	ROL	It rotates (circular-shifts) left the contents of specified register.
Arithmetic Shift left ~	ASHL	It shifts the contents of specified register 1-bit position towards left and fills vacant bit (rightmost) with zero.

5. Program Control Instructions

Instruction	Mnemonic	Description
Branch	BR	It transfers program control to the specified address
Jump	JMP	It transfers program control
Skip	SKP	It skips the next instruction.
call	CALL	It saves the address of the next instruction in the stack and transfers the program control to the specified address.
Return	RET	It reads the address from the top of stack, and transfer the program counter to the read address.

Micro-operation:- To fetch, decode and execute cycles, the processor unit has to perform set of operations called micro-operation. These operations include -

1. Transfer a word of data from one CPU register to the another.
2. Perform the arithmetic or logic operations on the data from the CPU registers and store the result in a CPU register.
3. Fetch a word of data from specified memory location and load them into a CPU register.
4. Store a word of data from a CPU register into specified memory location.

Micro-code:- The translation of symbolic micro-operation to binary produces a binary micro-program called micro-code.

Microprogram:- A sequence of one or more micro-operations designed to perform specific operation is called microprogram.

Control memory:- A memory that is part of control unit is referred to as a control memory. It stores the sequence of micro-operations to be performed to execute micro-instruction.

Microprogram Sequencer:- It is a sub-unit of microprogram control unit which presents an address to the control memory is called microprogram sequencer. The next address logic of the sequencer determines the specific address source to be loaded into the control address register. The block diagram of microprogram sequencer that selects an address from four sources and routes it into control address

register (CAR).

The output from CAR provides the address for the control memory. The contents of CAR are incremented and applied to the multiplexer and to the stack register file. The register selected in the stack is determined by the stack pointer.

Inputs I_2, I_1 and I_0 & T derived from the CD (condition code) and BR fields of microinstruction specify the operation for the sequencer.

They specify the input source to the multiplexer also generate push and pop signals required for stack pointer operation.

Stack pointer is a 3-bit register and it gives the address of the stack file consist of ($2^3 = 8$) eight registers.

Initially, the stack pointer is cleared and point address zero in the stack. Using PUSH operation, it is possible to write data into the stack at the address specified by the stack pointer. After data is written, stack pointer is incremented by 1 to get ready for the next PUSH operation.

I_2	I_1	I_0	T	S_1	S_0	Operation	Description
X	0	0	X	0	0	CAR ← EXA	Transfer external address.
1	0	1	1	0	1	CAR ← BRA, SR ← CAR+1	Branch to subroutine and save the next instruction address in stack (Push operation)
0	0	1	1	0	1	CAR ← BRA	Transfer branch address
X	1	0	0	1	0	CAR ← SR	Transfer from stack register.
0	1	1	1	1	1	CAR ← CAR+1	Increment address.

∴ Typical microoperation sequencer organization

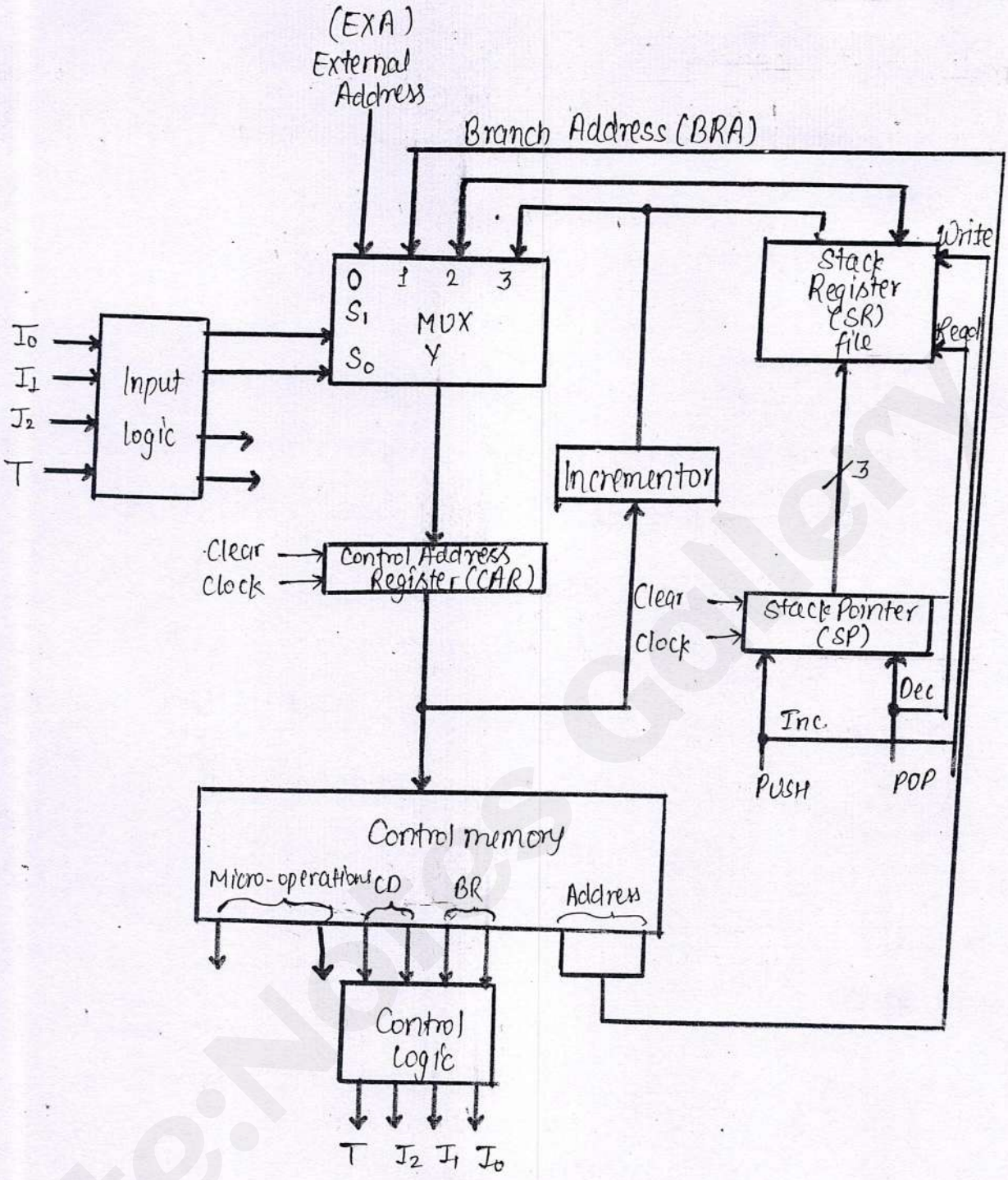


Fig. Typical microprogram sequencer organization

Address selection in Control Memory

Pre-fetching micro-instruction:- The disadvantage of micro-program control is that it results slower operating speed because of the time it takes to fetch micro instruction from the control memory. This problem can be solved by pre-fetching the next micro instruction while the current one is being executed. In this technique, the execution time can be overlapped with the fetch time.

Sometimes, the address of the next micro instruction is determined from the status flag and from the result of currently executed micro-instruction. In such cases, pre fetching of micro-instructions sometimes fetches a wrong micro instruction. So, in such case, the fetch must be repeated with correct address which require more complex hardware.

The pre-fetching technique is used to increase execution speed.

Comparison b/w hardwired control and micro-programmed control

S.No.	Characteristics	Hardwired control	Micro-programmed control
1.	Speed	Fast	Slow
2.	Implementation	Hardware	software
3.	Flexibility	Not flexible	Flexible
4.	Ability to handle large/ complex instruction set	Somewhat difficult	Easier
5.	Ability to support operating system and diagnostic features.	Very difficult	Easy
6.	Design process	Difficult for more operation.	Easy
7.	Memory	Not used	Control memory used (RAM or ROM)
8.	Chip area efficiency	useless area	uses more area
9.	Used in	RISC processor	CISC processor

10.	Output generation	On the basis of input signal.	On the basis of control line
11.	Control functions	Implemented in hardware	Implemented in software
12.	Instruction set size	usually under 100 instructions	usually over 100 instructions.
13	ROM size	—	2K to 10K by 20-400 bit micro instructions

Hardwired Control Unit :- In the hardwired control unit, uses fixed logic circuit to interpret instructions and generate control signals from them. In a hardwired control unit, all the control signals are generated by the hardware device.

The fixed logic circuits use contents of the control step counter, contents of instruction register, contents of the condition code flag and the external input signal. The fixed logic circuit block includes combinational circuit (decoder and encoder) that generates the required control outputs, depending on the state of all its inputs.

The instruction decoder decodes the instruction loaded in the IR. If IR is an 8-bit register then decoder generates $2^8 = 256$, lines, one for each instruction.

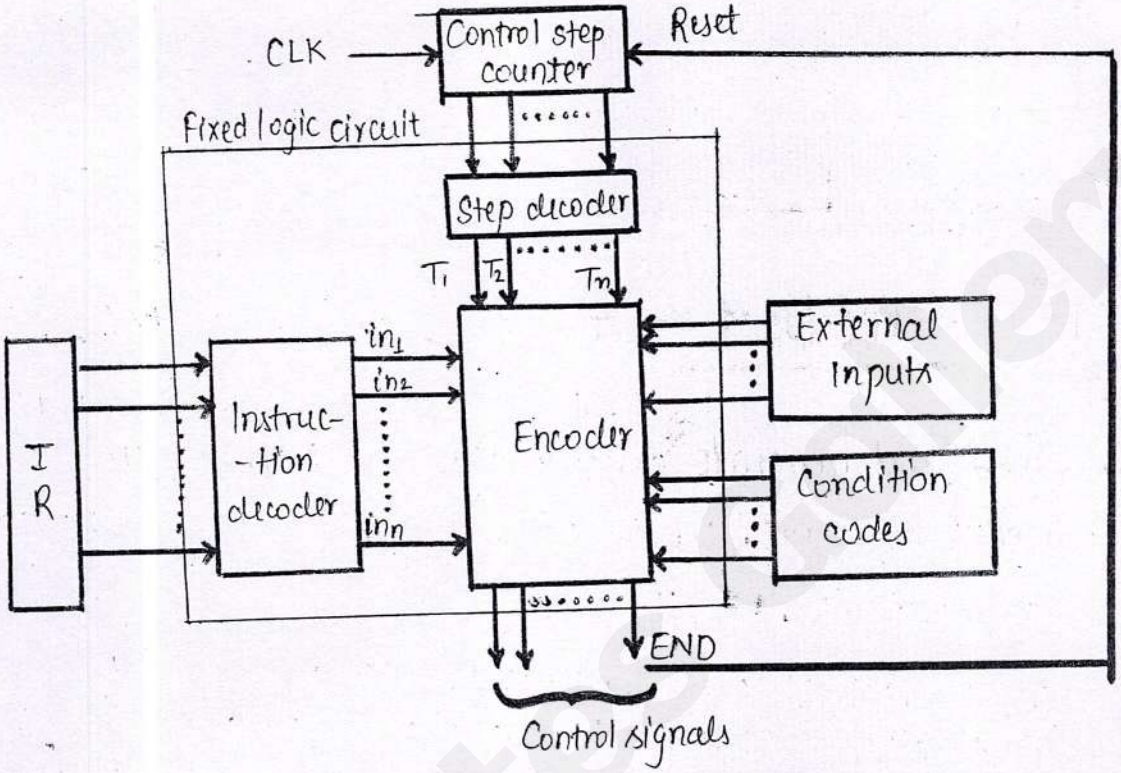
According to the code in the IR, only one line among all output lines of decoder goes high, that is, 1 and remaining lines are set to 0.

The step decoder provides a separate signal line for each step, time in a control sequence.

The encoder gets the input from the instruction decoder, step decoder, external inputs and condition codes. It uses all these

inputs to generate the individual control signals.

After execution of each instruction, end signal is generated which reset control step counter and make it ready for generation of next instruction.



Advantages of Hardwired Control Unit

1. Hardwired control unit is fast because control signals are generated by combinational circuits.
2. The delay in generation of control signals depends upon the no. of gates.
3. It has greater chip area efficiency since its uses less area on-chip.

Disadvantages of Hardwired Control Unit

1. More the control signals required by CPU, more complex will be the design of control unit.
2. Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit.

~~It has greater chip area effi~~

3. It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

Microprogrammed Control Unit / Software Based Control Unit

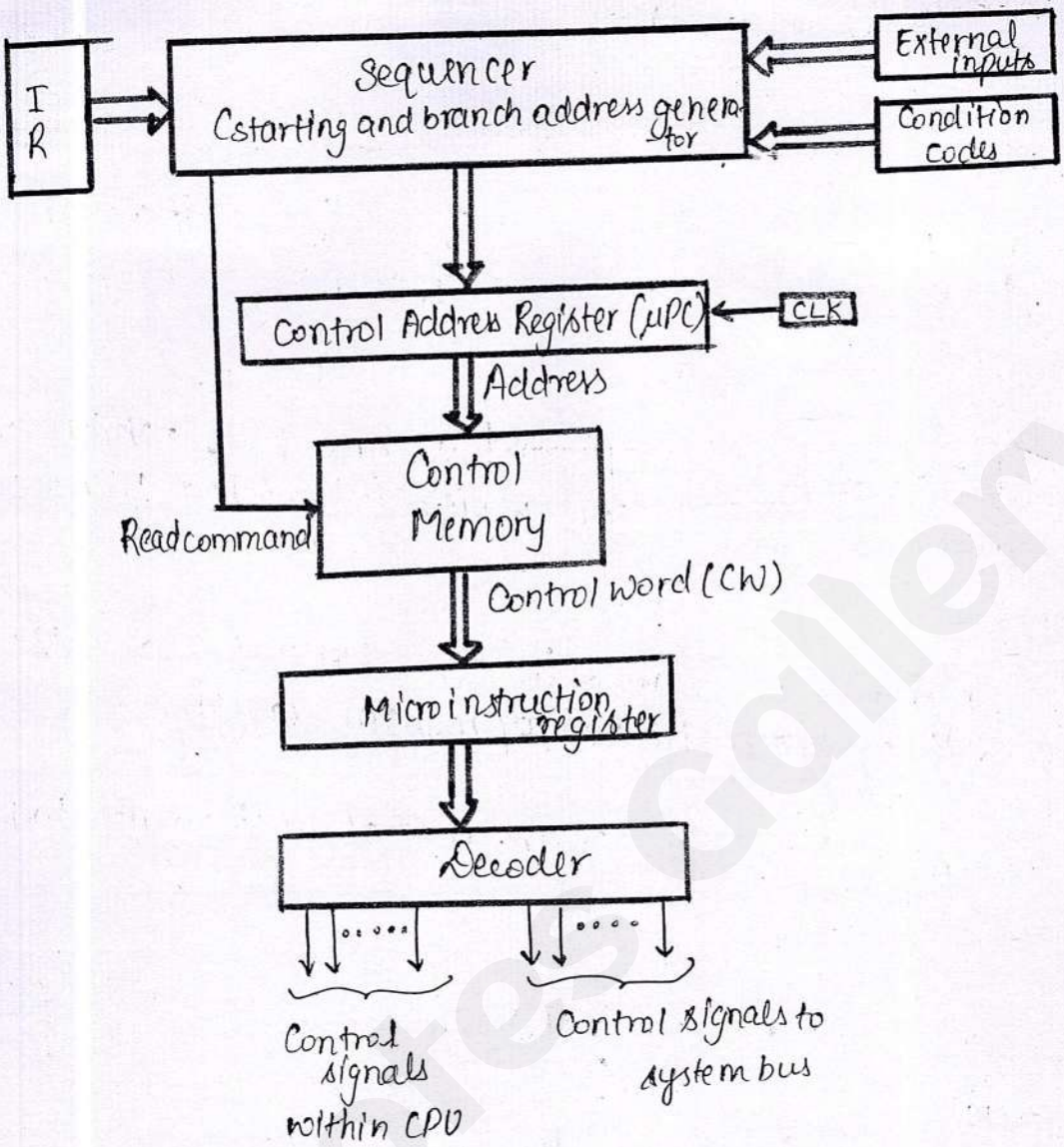
Microprogramming is a method of control unit design in which the control signal selection and sequencing information is stored in control memory.

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

The micro-programs are stored in control memory and their addresses are generated by microprogram sequencer.

The components of microprogram control unit works together as follows:-

1. The control address register (μPC) holds the address of next microinstruction to be read.
2. Every time a new instruction is loaded into IR,
3. When address is available in control address register, the sequencer issues READ command to the control memory.
4. After issue of READ command, the word from the addressed location is read into the micro-instruction register.
5. The μPC is then automatically incremented by the clock.
6. The content of microinstruction register generates control signals which are delivered to various parts of processor in the correct sequence



Micro Instruction:- A Each word in the control memory is a micro instruction which specify the control signals to be activated to perform one or more micro-operation And instruction that control the data flow and sequencing in a processor and a more fundamental level of machine instruction is known as micro instruction.

x A micro instruction is usually consist of 4 parts:- x

A micro instruction is usually consists of four parts:-

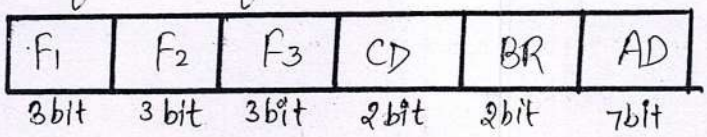
(1.) Microinstruction field denoted as F_1, F_2, F_3

(2.) Condition branching (CD)

(3.) Branch Field (BR)

(4.) Address Field (AD)

The general format of micro-instruction is as follows:-



Techniques of grouping control signals / List 2-techniques / Difference blw Horizontal & Vertical Micro / Types of Microinstruction

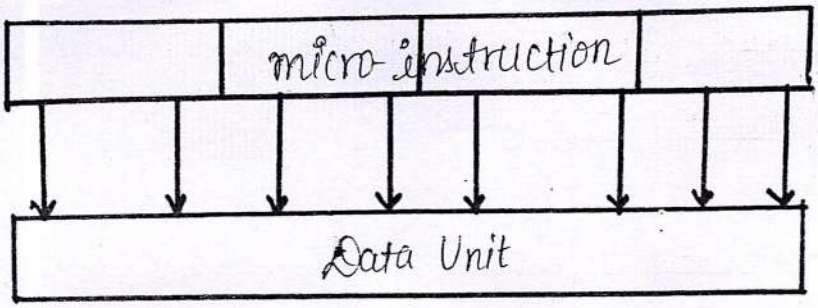
The grouping of control signals can be done using two techniques.

1. Vertical Micro Instruction Organization
2. Horizontal Micro Instruction organization

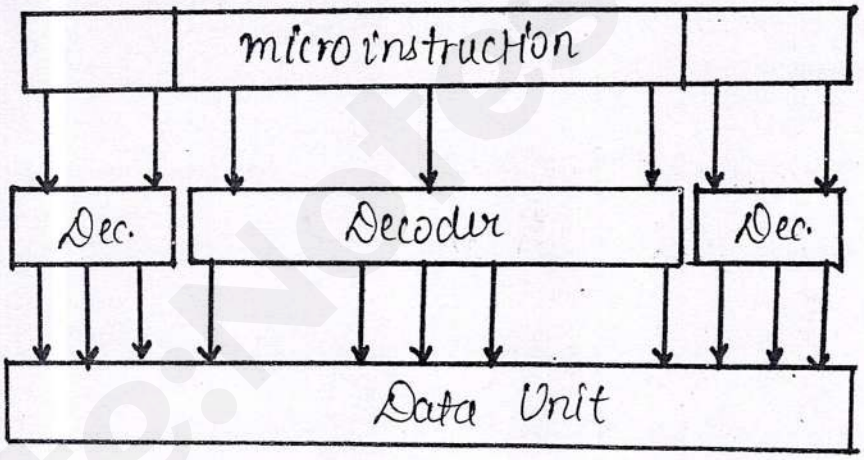
1. Horizontal Micro Instruction Organization:- With the help of decoded binary format we can represent the control signals in the horizontal micro-program. Here n- bit encoding is needed n-control signals.

A higher degree of parallelism is provided by the horizontal organization.

With the help of single control point, each bit is identified in the horizontal microprogramming.



Vertical Micro Instruction Orgⁿ:- In contrast to, the horizontal micro-programming a higher degree of encoding and variable format can be applied in the variable microprogramming organization. With the help of vertical organization, we can minimize the length of micro instruction as well as prevent the length of micro instruction from being directly affected by the increasing memory capacity



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

4.	It is faster than a vertical micro-programmed control unit.	4.	It is slower than a horizontal micro-programmed control unit.
5.	It is less flexible than a vertical micro-programmed control unit.	5.	It is more flexible than a horizontal micro-programmed control unit.

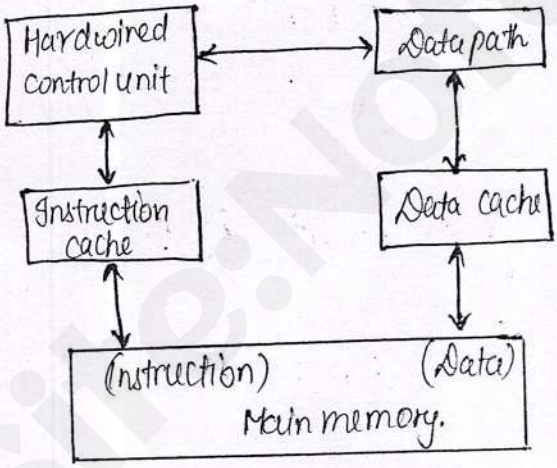
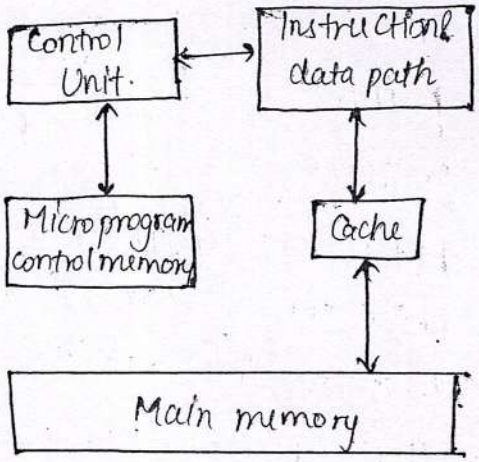
Execution of a complete instruction

As we know that for performing any micro-operation, we have to follow instruction cycle. Execution is a sub-cycle of Instruction cycle.

Let us find the complete control sequence for execution of the instruction ADD R_3, R_1. This instruction adds the contents of register R_1 and the contents of memory location specified by the register R_3 and store result in the memory location pointed by R_3. To execute this instruction, it is necessary to perform following actions:-

1. Fetch the instruction.
2. Fetch the operand from memory location pointed by R_3.
3. Perform the addition.
4. Store the result memory location pointed to the R_3.

S.No.	RISC (Reduced Instruction Set Computer)	CISC [Complex Instruction Set computer]
1.	Multiple register sets, often consisting of more than 256 registers.	Single register set, often consisting 6 to 16 registers total.
2.	Three registers operands allowed per instruction (for example, add R_1, R_2, R_3	One or two register operands allowed per instruction (for example, add R_1, R_2)
	Register based	Transistor based.

	RISC		CISC
3.	Parameter passing through efficient on-chip register windows.	*	Parameter passing through inefficient off-chip memory.
4.	Single-cycle instructions (except for load and store).	*	Multiple-cycle instructions.
5.	Hardwired control.	*	Micro-programmed control.
6.	Highly pipelined / power supply requirement	*	Less pipelined / power supply requirement
7.	Simple instructions are few in number.	*	There are many complex instructions
8.	Fixed length instructions.	*	Variable length instructions
9.	Complexity in compiler.	*	Complexity in microcode.
10.	Only load and store instructions can access memory.	*	Many instructions can access memory.
11.	Few addressing modes.	*	Many addressing modes.
*		*	
	RISC Architecture		CISC Architecture

Register to register data transfer occurs.

Register to register/ memory to register memory to memory data transfer

for Hardwired only

for microprogrammed + hardwired

Pipelining:- The term pipelining refers to the temporary over lapping of processing.

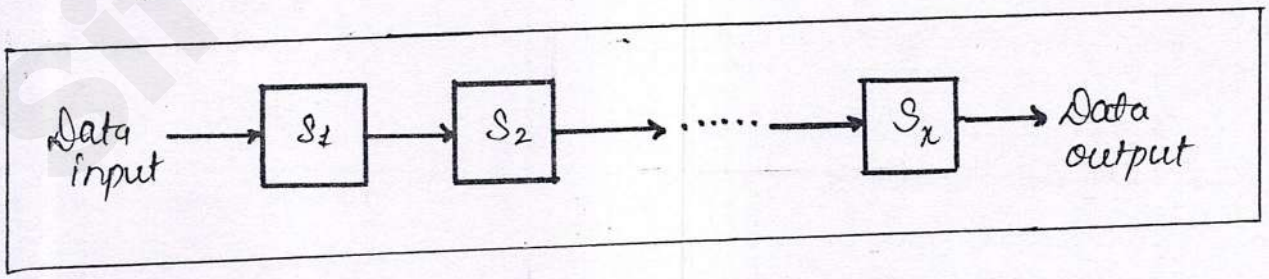
To achieve pipelining, one task must be divided into the sequence of sub-task, each of which can be executed by specialised hardware.

Pipelining is an implementation technique where multiple instructions are overlapped in execution.

The computer pipeline is divided in stages, each stage completes a part of an instruction in parallel.

Pipelining is a technique of decomposing a sequential process into sub-operations, each sub-process being executed in a special dedicated segment that operates currently with the all other segments.

Pipelining is a process of arrangement of hardware components of the CPU. such that its overall performance. is increased. Many instructions will be executed at a time in a pipeline processor i.e. more than one instruction can execute at a time in a pipeline processor.



Basic structure of pipeline processor

Eg:- Perform the arithmetic operation $(A_i * B_i) + (C_i * D_i)$. specify a pipeline configuration to carry out the task. list the contents of all registers in a pipeline for $i = 1$ through 6.

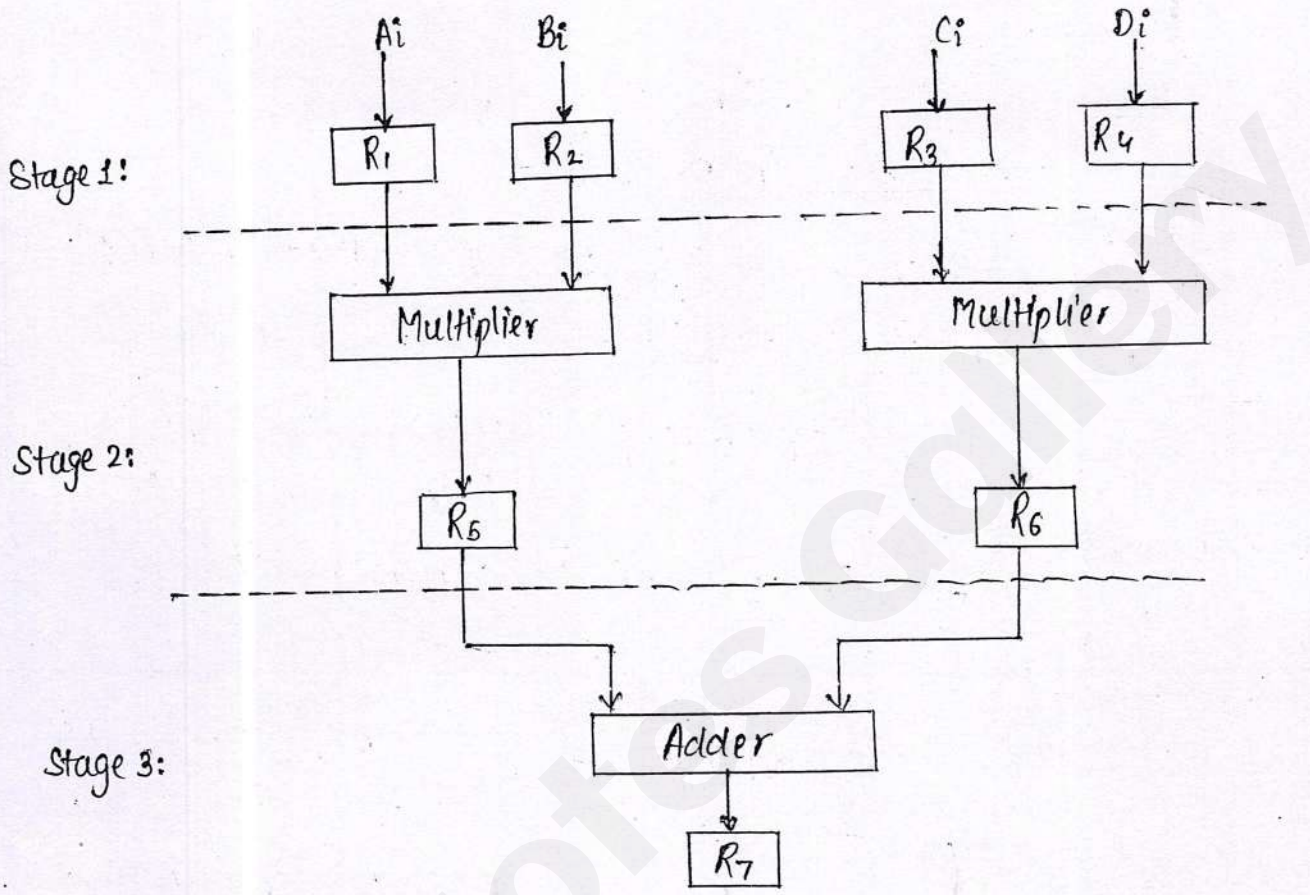


Fig. 6.3.2 Pipelining processing

Stage 1: $R_1 \leftarrow A_i$ $R_2 \leftarrow B_i$ $R_3 \leftarrow C_i$ $R_4 \leftarrow D_i$

Stage 2: $R_5 \leftarrow R_1 * R_2$, $R_6 \leftarrow R_3 * R_4$

Stage 3: $R_7 \leftarrow R_5 + R_6$

Clock Pulse Number (t)	Stage 1				Stage 2		Stage 3
	R_1	R_2	R_3	R_4	R_5	R_6	R_7
1.	A_1	B_1	C_1	D_1	–	–	–
2.	A_2	B_2	C_2	D_2	$A_1 * B_1$	$C_1 * D_1$	–
3.	A_3	B_3	C_3	D_3	$A_2 * B_2$	$C_2 * D_2$	$A_1 * B_1 + C_1 * D_1$
4.	A_4	B_4	C_4	D_4	$A_3 * B_3$	$C_3 * D_3$	$A_2 * B_2 + C_2 * D_2$
5.	A_5	B_5	C_5	D_5	$A_4 * B_4$	$C_4 * D_4$	$A_3 * B_3 + C_3 * D_3$
6.	A_6	B_6	C_6	D_6	$A_5 * B_5$	$C_5 * D_5$	$A_4 * B_4 + C_4 * D_4$
7.	–	–	–	–	$A_6 * B_6$	$C_6 * D_6$	$A_5 * B_5 + C_5 * D_5$
8	–	–	–	–	–	–	$A_6 * B_6 + C_6 * D_6$

Table: Contents of all registers in the pipeline $i = 1$ through 6

Types of pipelining

1. **Arithmetic pipelining:-** The ALU can be segmentized for pipeline operation in a various data formats.
2. **Instruction pipelining:-** In this, a stream of instructions can be executed by over lapping fetch, decode and execute phase of an instruction cycle.
3. **Processor pipelining:-** This refers to pipeline processing of the same data stream by cascade of processors each of which processes a specific task

Models of Pipelining

1. **Asynchronous Pipeline model**:- (→ own clock) In asynchronous pipeline models, data flow along the pipeline stages is controlled by handshaping protocol. Every segment uses different - different clock pulse.

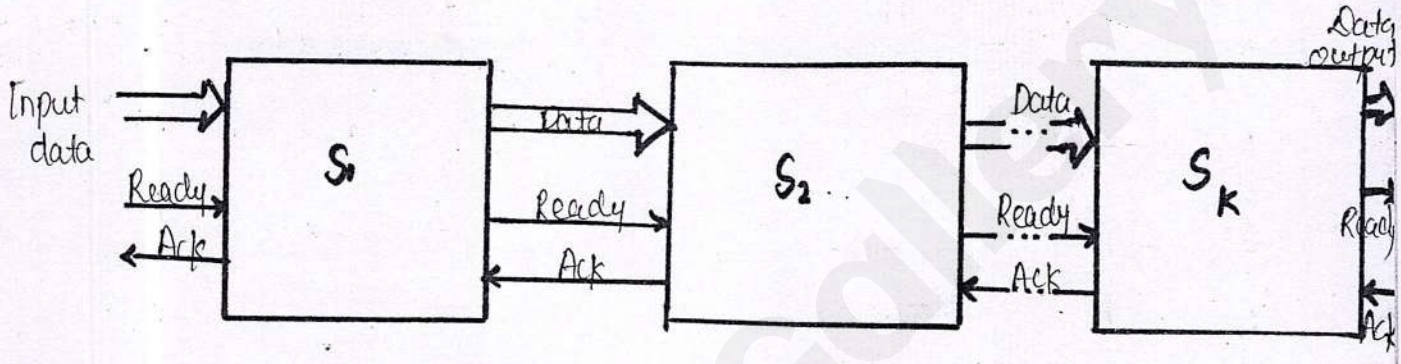
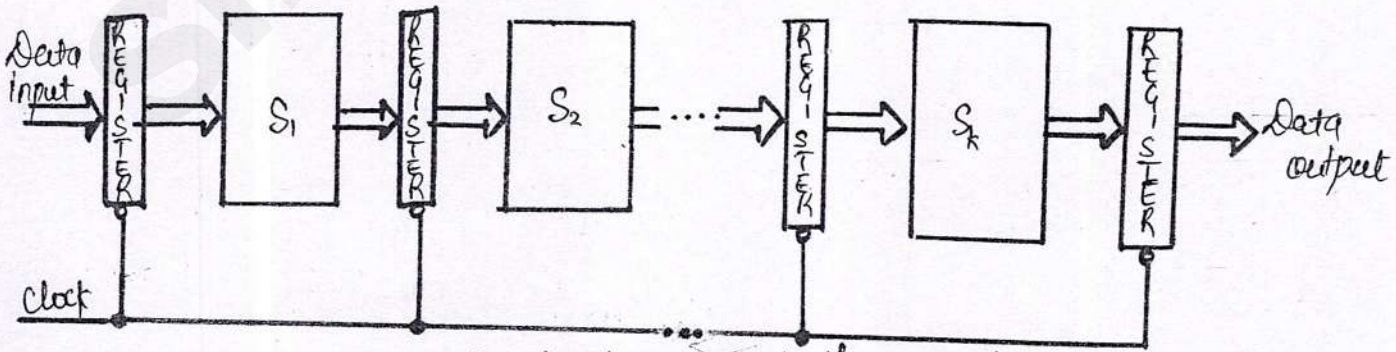


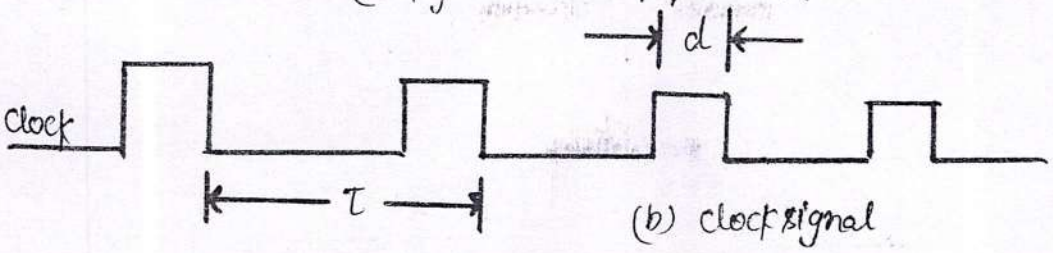
Fig 6.3.3:- Asynchronous pipeline model

(→ same) Global clock (same clock)

2. **Synchronous Pipeline model**:- In synchronous pipeline model clock high speed registers are used to interface b/w stages. At the falling edge of clock pulse on registers transfer data to the next stage simultaneously.



(a) Synchronous pipeline model



(b) Clock signal

Ques:- Draw a space time diagram for 6 segments, 8 stages, pipeline representing the time. Each stage has eight tasks.

Soln:-

Time → / stages	1	2	3	4	5	6	7	8	9	10	11	12	13
Stage 1	T_1^1	T_1^2	T_1^3	T_1^4	T_1^5	T_1^6	T_1^7	T_1^8					
stage 2		T_2^1	T_2^2	T_2^3	T_2^4	T_2^5	T_2^6	T_2^7	T_2^8				
stage 3			T_3^1	T_3^2	T_3^3	T_3^4	T_3^5	T_3^6	T_3^7	T_3^8			
stage 4				T_4^1	T_4^2	T_4^3	T_4^4	T_4^5	T_4^6	T_4^7	T_4^8		
Stage 5					T_5^1	T_5^2	T_5^3	T_5^4	T_5^5	T_5^6	T_5^7	T_5^8	
stage 6						T_6^1	T_6^2	T_6^3	T_6^4	T_6^5	T_6^6	T_6^7	T_6^8

T_j^i j:- segment (process)

i:- subprocess of segment.

Ques:- Draw a time space diagram for no. of processes = 4 and it can be decomposed in 5 sub-process.

Time → / Stages	1	2	3	4	5	6	7	8
Stage 1	T_1^1	T_1^2	T_1^3	T_1^4	T_1^5			
Stage 2		T_2^1	T_2^2	T_2^3	T_2^4	T_2^5		
Stage 3			T_3^1	T_3^2	T_3^3	T_3^4	T_3^5	
Stage 4				T_4^1	T_4^2	T_4^3	T_4^4	T_4^5
~~Stage 5~~					~~T_5^1~~	~~T_5^2~~	~~T_5^3~~	~~T_5^4~~