COA Unit - 3 Notes

Minakshi
IT Department.

Instruction:

→ The operation of the processor is determined by the instruction, it executes, referred to as machine instructions or computer instruction

→ The collection of different instructions that processor can execute is referred to as the processor's instruction set.

→ A computer instruction is a binary code that instruct the computer to perform a specific operation.

Elements of a Machine instruction:

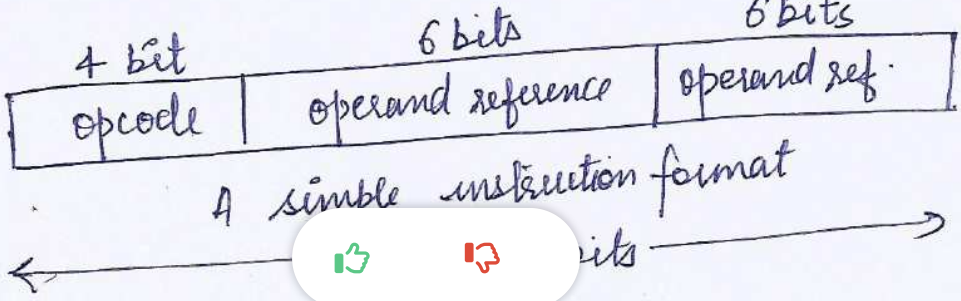
→ operation code
→ source operand reference
→ Result operand reference
→ Next instruction reference.

→ An operation code known as opcode which specifies operation to be performed.

→ Source operand reference: Reference to operands on which the operation is to be performed.

→ Result operand reference: A reference to operand which will store the result produced by the instruction

→ A reference to the next instruction to be executed



A simble instruction format

← bits →

→ source and result can be in one of four areas.
  → main or virtual memory
  → Processor register
  → Immediate (contained in instruction directly)
  → I/O device.

→ opcodes are represented by abbreviations, called mnemonics such as ADD, SUB, LOAD, STOR etc.

→ An instruction may have many fields. A layout of a simple instruction is known as "Instruction format"

Types of Instruction:

Most computer instructions can be classified into three categories:
  → Data Transfer instructions
  → Data Manipulation instructions
  → Program control instruction.

Data transfer instructions cause transfer of data from one location to another without changing the binary information content.

Data manipulation instructions are those that perform arithmetic, logic, and shift operations.

- Program control instruction provide decision making capabilities and change the path taken by the program when executed in the computer.

Data Transfer:

→ Data transfer instructions move data from one location to another without changing the data content.

→ These instructions are used to bring data to and from memory to registers.

Examples:

MOVE: designates a transfer from one register to another.

Store: transfer from a processor register into memory.

Load: a transfer from memory to register

Exchange: swaps information between two registers or a register and memory.

Input: transfer from input terminal to processor register.

output: transfer from process register to output terminal.

Push: transfer from process register to the top of stack.

Pop: transfer from top of stack to process register.

clear: transfer of word 0's (zeros) to the destination

Set: transfer of word 1's (ones) to the destination.

Data Manipulation:

- Data manipulation instructions perform operations on the data.
- These can be divided into three types.

(1) Arithmetic
(2) Logical and bit manipulation instructions
(3) Shift microoperations.

Arithmetic Instructions:

These instructions are used to perform arithmetic operations.

Example:

Add → compute the sum of two operands
Subtract → Compute difference of two operands
Multiply → compute the product of two operands
Divide → Compute quotient of two operands.

Add with carry → compute the sum of two operands with carry
Subtract with borrow → compute the difference with borrow.
Negate (2's complement) → change the sign of the operand.

Increment → Add 1 to operand
Decrement → Subtract 1 from operand.

Logical Instruction:

- Logical and bit manipulation instruction.
- Logical instructions perform binary operations on string of bits stored in registers.

Example:

AND → performs AND operands
OR → performs OR of operands
Complement or NOT → takes complement
X-OR → performs XOR of operands

Clear selected bits ⇒ to clear a bit or selected bits to zero

set selected bits ⇒ to set a bit or selected bits to 1.

complement selected bits ⇒ to complement a bit or selected bits.

Test ⇒ Test specified condition (set flags)

compare ⇒ make logical comparison of two operands.

Enable Interrupt ⇒ } ⇒ set control variables for interrupt handling, timer control etc.

Disable Interrupt ⇒ }

Shift Instructions:

→ shifts are operations in which the bits of the word are moved to the left or right.

Example:

Logical shift Right
Logical shift Left
Arithmetic shift Right
Arithmetic shift Left
Rotate Right
Rotate Left.

Program Control: or Transfer of Control instructions.

→ A program control type of instructions, when executed, may change the address value in the program counter and cause the flow of control to be altered.

→ Program control instructions specify conditions for altering the content of the program counter.

Example Program control instruction

Jump (branch) → Unconditional transfer. Load PC with specified address.

Jump (Conditional) → Test specified condition, either load PC with specified address, or do nothing based on condition

Jump to subroutine → Place current program control information in known location, Jump to specified address.
or CALL

Return → Replace contents of PC and other registers from the known locations.

Skip → Increment to PC to skip next instruction.

Skip (conditional) → Test specified conditions, either skip or do nothing based on condition

Halt → Stop program execution.

Wait (hold) → Stop program execution, test specified condition repeatedly; resume execution when condition is satisfied.

No operation → No operation is performed, but program execution is continued.

Input and output Instructions:

Input (Read) → Transfer data from input terminal to a destination

Output (write) → Transfer data from specified source to output terminal

Start I/O → Transfer instruction to I/O processor to initiate operation

Test I/O → Transfer status information I/O system to the specified destination.

Conversion Instruction:

- convert or translate value or words.

Example

Translate → Translates value in a section of memory based on table of correspondence

Convert → Converts the content of word from one form to another.

Example Conditional Branch instructions.

Branch if zero.
Branch if not zero
Branch if carry
Branch if not carry
Branch if plus
Branch if minus
Branch if overflow
Branch if no overflow.

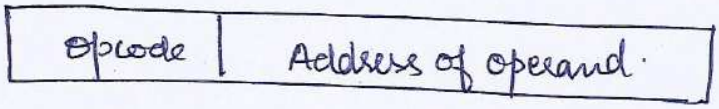
STORED PROGRAM ORGANIZATION : OR Single Accumulator Organization

OR

Processor Organization with Accumulator

→ To organize a computer in this method, Processor has a register called accumulator and an instruction code format with two parts.

→

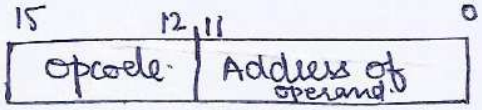


→ The first part specifies the operation to be performed and the second specifies an address of operand in the memory.

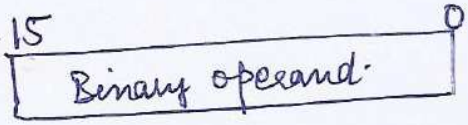
→ The other operand will be stored in the processor register

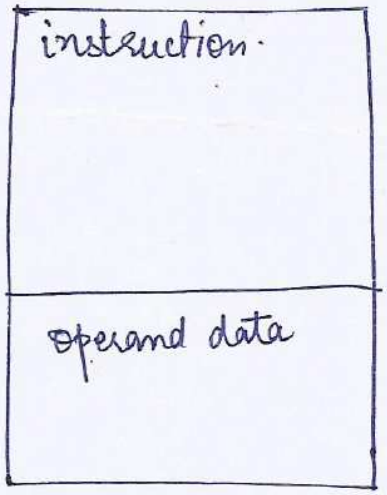
Example

Memory 4096X16



instruction format.





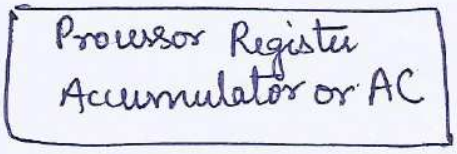
Number of bits in address = 12

Number of memory locations = 2^{12}

= 4096

The numbers of bits of operand or data

= 16.



In this method, processor performs the operation specified by the opcode on the data stored on the memory locatio and content of

Instruction Format :

Most common fields of instructions are:

→ operation field which specifies the operation to be performed

→ Address field which contains the location of opeeand i.e, register or memory location

→ Mode field which specifies how opeeand is to be found.

Generally CPU organizations are

→ Single Accumulator Organization

→ General Register Organization

→ Stack Organization

→ An instruction is of various length depending on the number of addresses, it contains

On the basis of number of addresses instruction can be classified

→ Three address instruction

→ Two address instruction

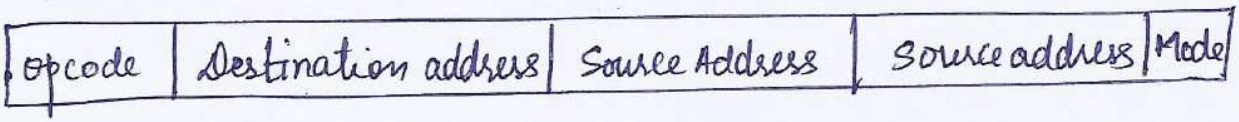
→ One address instruction

→ Zero address instruction

} instruction formats.

Three address instruction :

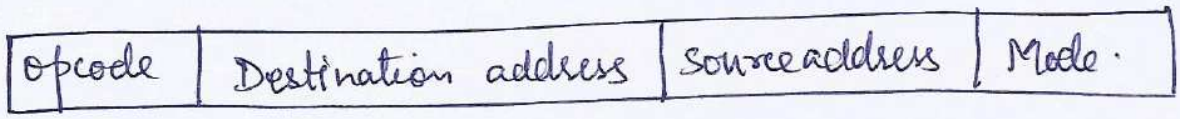
This format of instruction has three addresses to specify a register or memory location.



Examples: ADD R1, R2, R3
OR
ADD R1, A, B

Two address instruction:

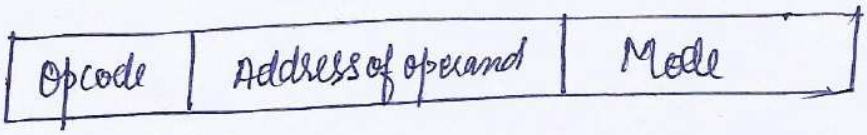
This format has only two address to specify a register or a memory location.



Examples: MOV R1, A
ADD R1, B

One-address instruction :

- It has only one address to specify a register or a memory location
- This uses an implied 'ACCUMULATOR' or AC Register to store one operand / result.



Example LOAD A
or
ADD

Zero Address Instruction:

- This instruction format does not contain any address field.
- A stack based computer do not use address field in instruction to perform operation. Because the operation is performed on the two top items of the stack.

Example ADD

Note In stack-base organization, to evaluate an expression first it is converted to Postfix notation or Reverse Polish Notation

---

Example for instruction formats to evaluate an expression.

Example ① Expression $X = (A+B) * (C+D)$

Three address Format:

	Description
ADD R1, A, B	R1 ← M[A]+M[B]
ADD R2, C, D	R2 ← M[C]+M[D]
MUL X, R1, R2	M[X] ← R1 * R2

Here R1, R2 are 2 registers

M[ ] ⇒ ...tion.

Two address instructions

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

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

One address instruction:

		Description
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

where AC is Accumulator Register

M[T] is temporary memory location to hold the intermediate Result.

M[ ] ← memory location

Zero address Instruction:

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

Postfix notation $X = AB+CD+*$

		Description
PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP ← D
ADD		TOP ← (C+D)
MUL		TOP (A+B) * (C+D)
POP	X	M[X] = TOP

Ques: write a program to evaluate $\frac{(A-B)+C*(D*E-F)}{G+H*K}$

Ans: Three address instruction:

		Description
SUB	R1, A, B	R1 ← M[A] − M[B]
MUL	R2, D, E	R2 ← M[D] * M[E]
SUB	R2, R2, F	R2 ← R2 − M[F]
MUL	R2, R2, C	R2 ← R2 * M[C]
ADD	R1, R1, R2	R1 ← R1 + R2
MUL	R3, H, K	R3 ← M[H] + M[K]
ADD	R3, R3, G	R3 ← R3 + M[G]
DIV	X, R1, R3	M[X] ← R1 / R3

Two address instructions

		Description
MOV	R1, A	R1 ← M[A]
SUB	R1, B	R1 ← R1 − M[B]
MOV	R2, D	R2 ← M[D]
MUL	R2, E	R2 ← R2 * M[E]
SUB	R2, F	R2 ← R2 − M[F]
MUL	R2, C	R2 ← R2 * M[C]
ADD	R1, R2	R1 ← R1 + R2
MOV	R3, H	R3 ← M[H]
ADD	R3, G	R3 ← R3 + M[G]
DIV	R1, R3	R1 ← R1/R3
MOV	X, R1	M[X] ← R1

One address instruction

		Description
LOAD	A	AC ← M[A]
SUB	B	AC ← AC − M[B]
STORE	T	M[T] ← AC
LOAD	D	AC ← M[D]
MUL	E	AC ← AC * M[E]
SUB	F	AC ← AC − M[F]
MUL	C	AC ← AC * M[C]
ADD	T	AC ← AC + M[T]
STORE	T	M[T] ← AC
LOAD	H	AC ← M[H]
MUL	K	AC ← AC * M[K]
ADD	G	AC ← AC + M[G]

STORE	T1	M[T1] ← AC
LOAD	T	AC ← M[T]
DIV	T1	AC ← AC / M[T1]
STORE	X	M[X] ← AC.

Zero address instruction

Postfix Notation AB- CDE*F-*+ GHK-*+/

		Description
PUSH	A	Top ⇒ A
PUSH	B	Top ⇒ B
SUB		Top ⇒ (A-B)
PUSH	C	Top ⇒ C
PUSH	D	Top ⇒ D
PUSH	E	Top ⇒ E
MUL		Top ⇒ (D*E)
PUSH	F	Top ⇒ F
SUB		Top ⇒ ((D*E)-F))
MUL		Top ⇒ C* ((D*E)-F)
ADD		Top ⇒ (A-B)+(C*((D*E)-F)
PUSH	G	Top ⇒ G
PUSH	H	Top ⇒ H
PUSH	K	Top ⇒ K
MUL		Top ⇒ (H*K)
ADD		Top ⇒ G+(H*K)
DIV		Top ⇒ ((A-B)+C*((D*E)-F)/ (G+(H*K))
POP	X	X ← Top.

Tutorial sheet

Ques: ① write an program for expression A+B*C using one address and Two address instruction

Ans: Expression ⇒ D= A+B*C

Program using one address instruction

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

Program using Two address instruction

MOV	R1, B	R1 ← M[B]
MUL	R1, C	R1 ← R1 * M[C]
ADD	R1, A	R1 ← R1 * M[A]
STORE / MOV	D, R1	M[D] ← R1

Program using Zero address instruction

A BC*+=

PUSH	A	Top = A
PUSH	B	Top = B
PUSH	C	Top = C
MUL		Top = B*C
ADD		Top = A + B*C
POP	D	M[D] = A + B*C

New Topic

⇒ How would you find the number of bits ?

Example (Ques1) if a system has memory of 4096 words 16 bits per word then ⇒

Register		Size or Number of bits
Program Counter	PC	12
Address Register	AR	12
Instruction Register	IR	16
Data Register	DR	16
Accumulator	AC	16

Number of locations = 4096

Number of bits to identify 4096 locations ⇒ 12

→ Number of bits in address bus ⇒ 12
Number " " " data bus = 16

Ques2 A computer uses a memory of 256K words of 32 bit each. A binary instruction code is stored in a one word of memory. The instruction has four parts

- An indirection bit
- An operation code
- A register part to specify 64 registers and
- An address part

(i) How many bits are there in the operation code, the register code and address part?

(ii) Draw the instruction format and indicate the number of bits in each part.

(iii) How many bits are there in the data and address inputs/bus of memory

Ans Number of words in memory = 256K ⇒ 256 × 1024 ⇒ $2^8 \times 2^{10}$

Size of the word ⇒ 32 bit

Size of the instruction = 32. { because instruction is stored in one word of memory

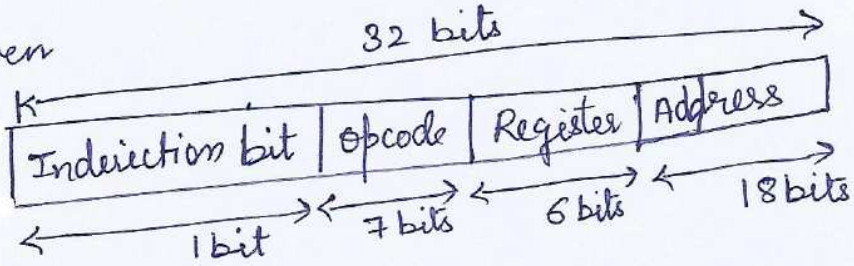
No. of Indirection bits ⇒ 1

Number of register = 64 ⇒ 2^6

bits to identify 64 registers = 6

Number of bits in address = 18

instruction format Given



Number of bits in opcode = 32 − (1+6+18)

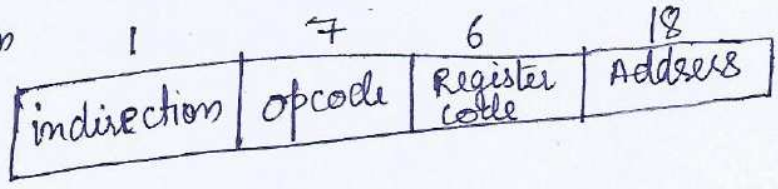
= 7 bits

(i) Number of bits in opcode = 7

Number of bits in register code = 6

Number of bits in address part = 18

(ii) Format of instruction



(iii) Number of bits in the address input = 18

Number of bits in data input = 32.

Ques For 256 instructions and 1024 word memory, find out the number of bits required in each type of instruction.

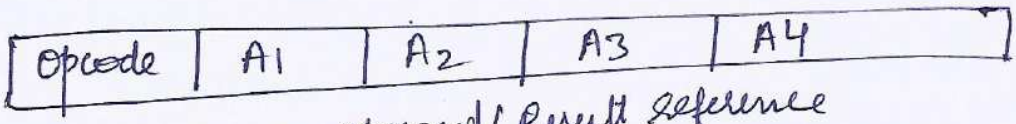
~~where instruction has~~

→ ~~Opcode~~

→ ~~Reference to operands~~

→ ~~Reference of the next instruction~~.

Ans: Four. ~~Three~~ address instruction



$A_1, A_2, A_3 \Rightarrow$ operand/ Result reference

$A_4 \Rightarrow$ Next instruction reference.

- Number of possible instruction = 256

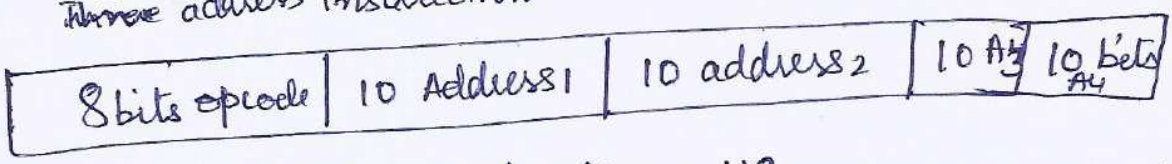
  Number of bits in opcode = 8

  $2^8 = 256$.

Number of locations = 1024

Number of bits in address = 10 $\{\because 2^{10} = 1024\}$

Four. ~~Three~~ address instruction



Number of bits in instruction = 48

instruction

Opcode	operand1	operand2	- - - - - -

⇒ if operands are stored in register, then it is called register reference instruction

→ if operands are stored in memory, then it is called a memory reference instruction.

Qees: if 16-bit fixed length of instruction is stored in 128 word memory, then find out the number bits in opcode and total number of possible operations in 1 address instruction format and 2 address instruction format.

Ans: One address instruction format.

Number of locations in memory = 128

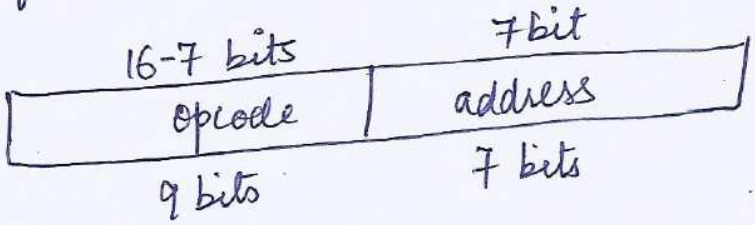
Number of bits in address to identify 128 location = n

$$2^n = 128$$

$$2^n = 2^7$$

$$\boxed{n = 7}$$

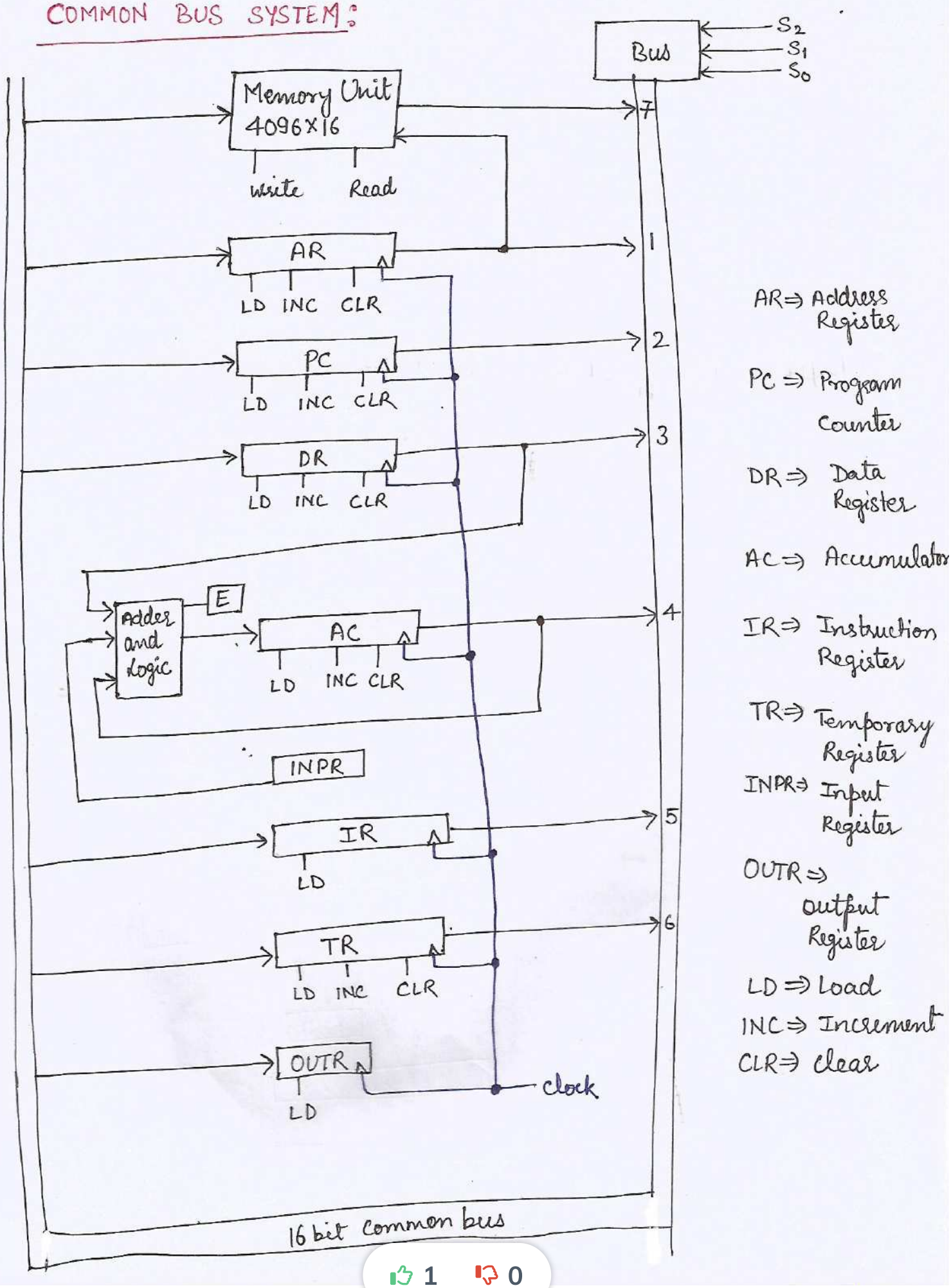
Number of bits in address field = 7



→ Number bits in opcode = 9

Number of possible operations = $2^9 \Rightarrow 512$

COMMON BUS SYSTEM:



AR ⇒ Address Register

PC ⇒ Program Counter

DR ⇒ Data Register

AC ⇒ Accumulator

IR ⇒ Instruction Register

TR ⇒ Temporary Register

INPR ⇒ Input Register

OUTR ⇒ Output Register

LD ⇒ Load

INC ⇒ Increment

CLR ⇒ clear

* Instruction cycle:

An instruction cycle consists of an instruction fetch, followed by Zero or more operands fetches, followed by zero or more operand stores, followed by an interrupt check (if interrupts are enabled).

→ An instruction cycle includes the following stages / subcycles.

- Fetch
- Execute
- Interrupt

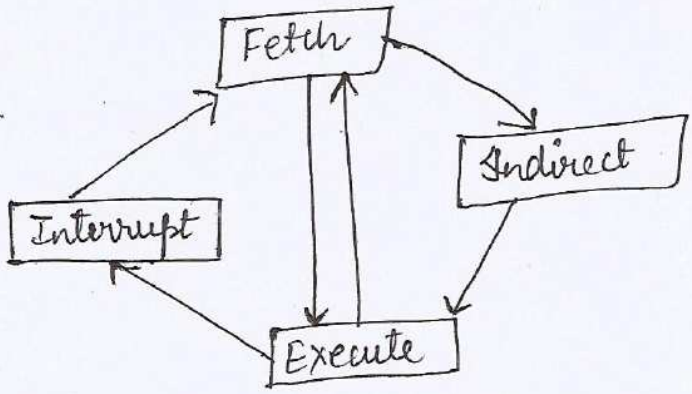
(1) **Fetch:** Read the next instruction from memory into processor.

(2) **Execute:** Interpret the opcode and perform the indicated operation

(3) **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

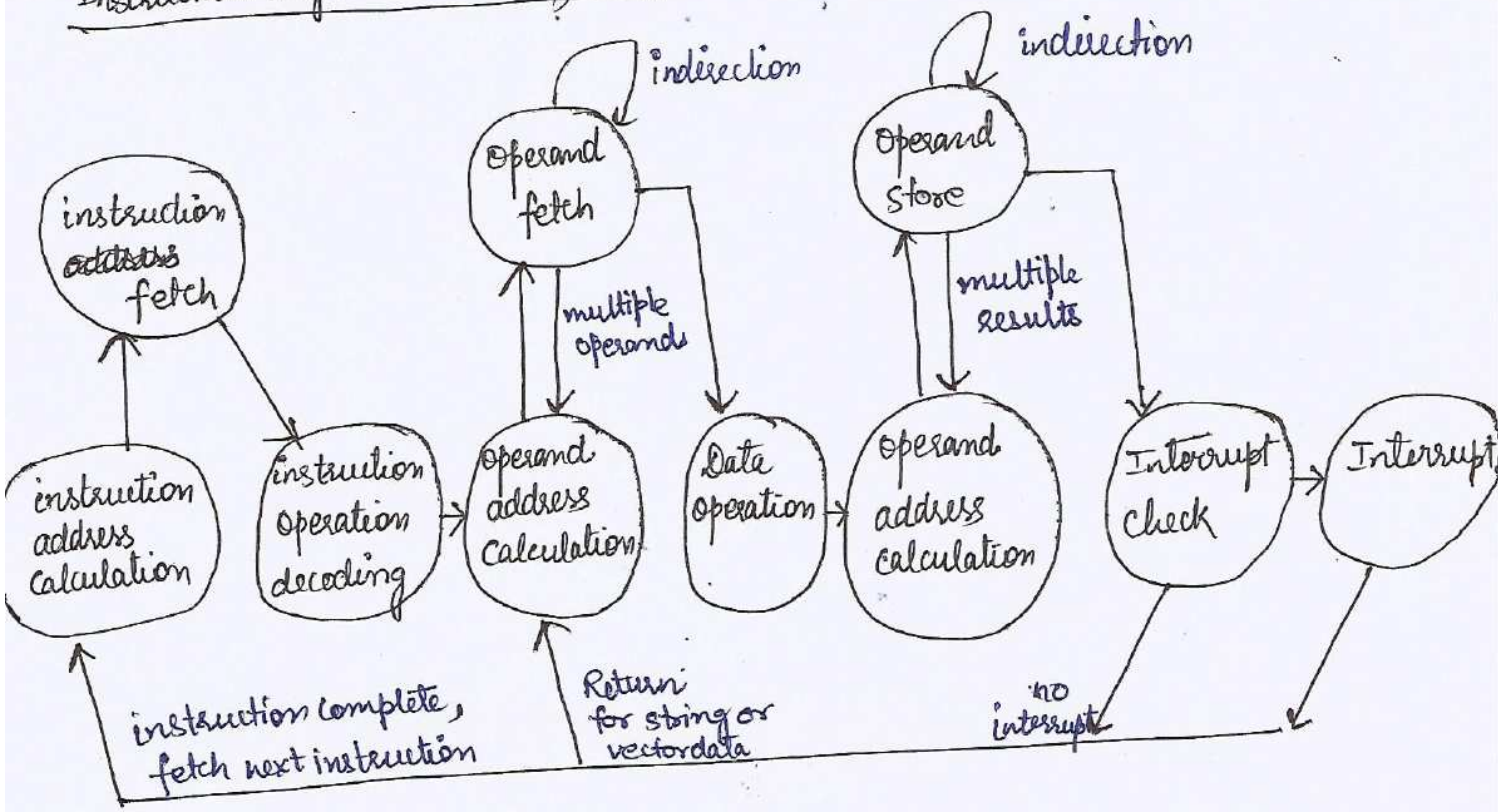
(4) **Indirect** (additional) only when indirect addressing involved.

After an instruction is fetched, it is examined if any indirect addressing is involved or not, if involved, the required operands are fetched using indirect addressing. Following execution, an interrupt may be processed before next instruct



Instruction cycle (indirect addressing involved)

Instruction cycle state diagram:

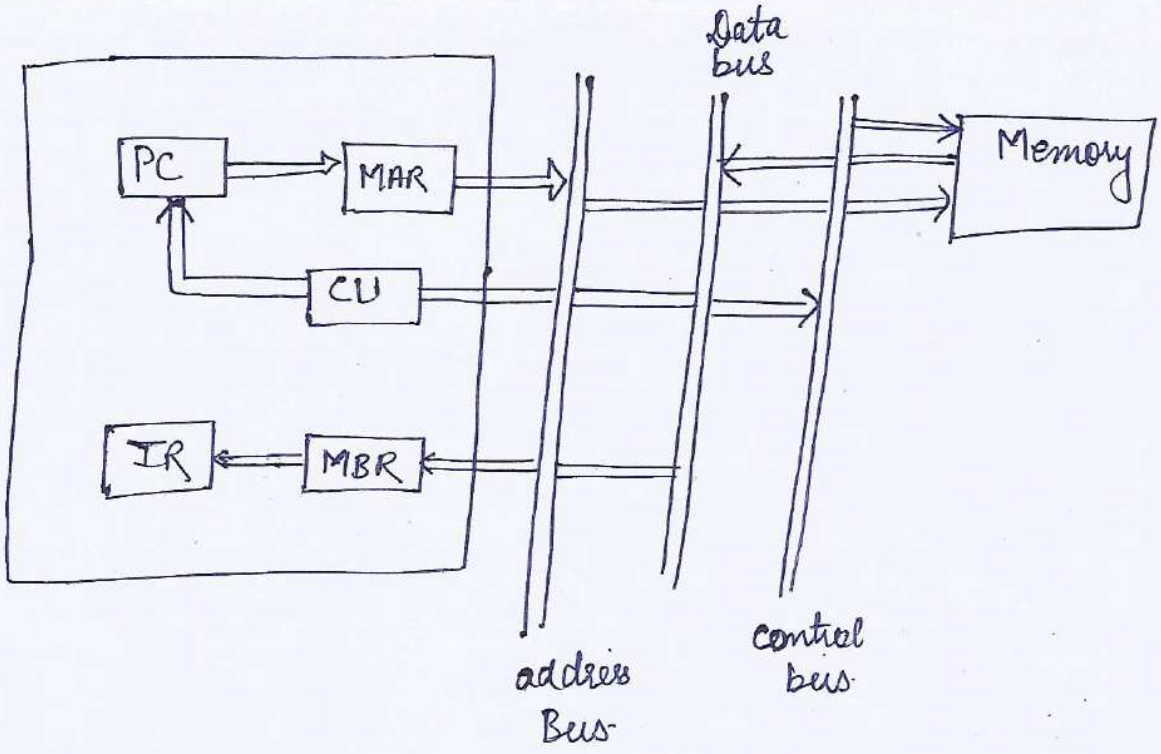


The exact sequence of events during an instruction cycle depends on the design of the processor.

Instruction Sub-cycle's Data flow:

Fetch cycle: During fetch cycle, an instruction is read from memory. PC (program counter) contains the address of next instruction to be fetched. This address is moved to MAR and placed on the address bus.

The control unit requests a memory read and the result is placed on the data bus and copied to MBR and moved to IR. And then PC is incremented by 1.

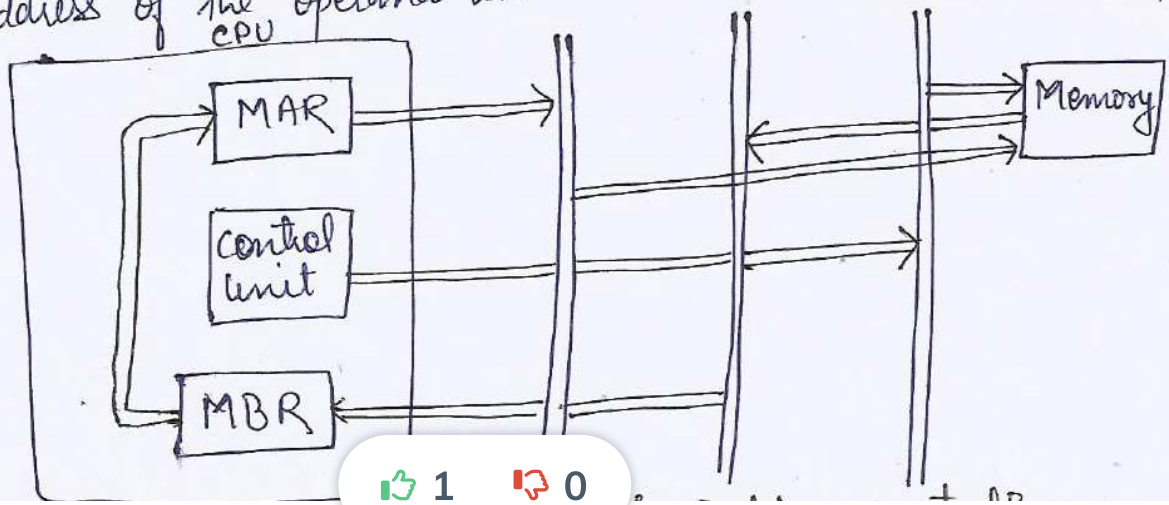


Data flow fetch cycle

Indirect Cycle:

After fetch cycle, control unit checks IR if it contains an operand specifier using indirect addressing, if so an indirect cycle is performed.

→ The right most N bits of MBR, which contains the address reference are transferred to the MAR.
Then control unit requests a memory read, to get the desired address of the operand into the MBR.



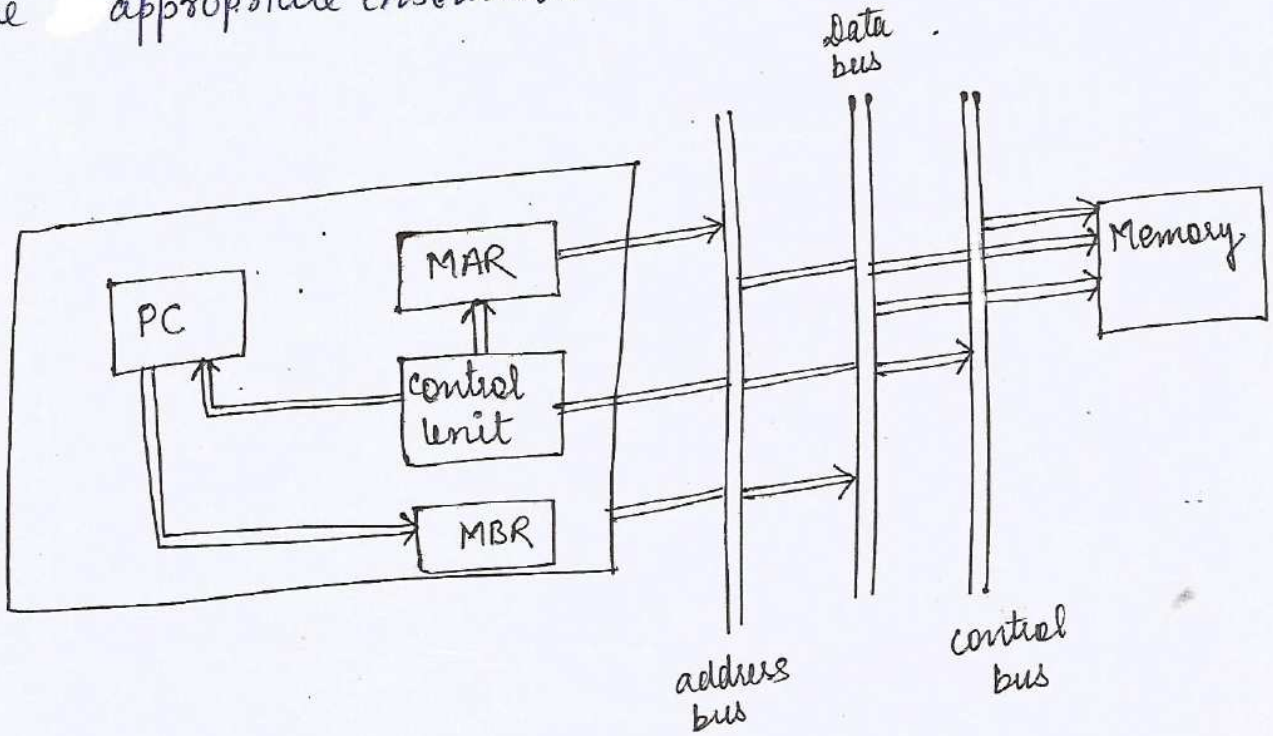
Dataflow indirect cycle

Interrupt cycle:

If an interrupt has occurred, the current content of PC must be saved so that the processor can resume normal activity after the interrupt.

Thus the content of PC are transferred to the MBR to be written into the memory. The special memory location reserved for this purpose is loaded into MAR from the control unit.

The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.



Data flow, interrupt cycle

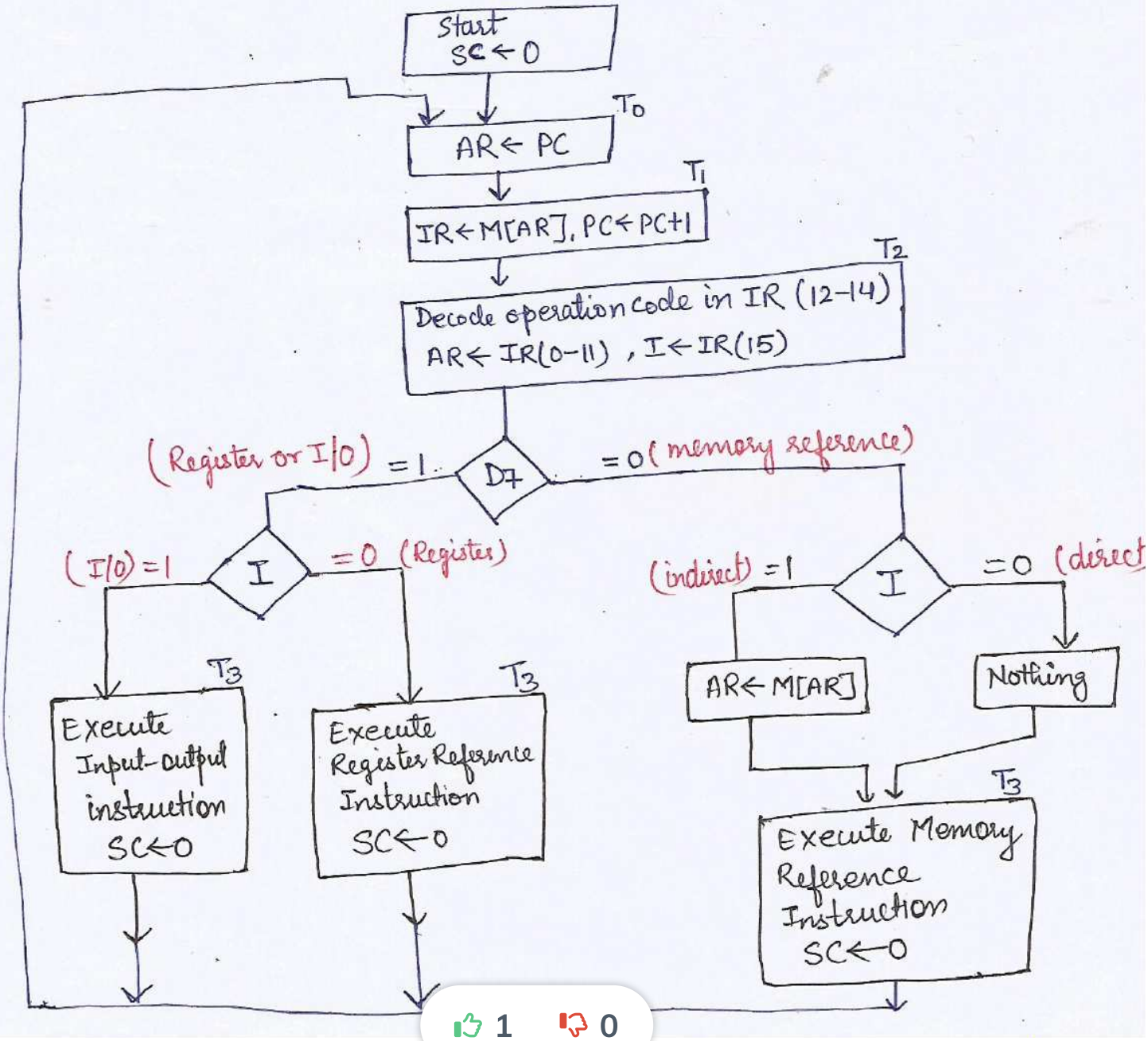
Flow chart of instruction cycle:

- In basic computer, each instruction cycle consists of the following:

1- Fetch the instruction from memory
2- Decode the instruction
3- Read the effective address from the memory if indirect address
4- Execute the instruction.

After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.

This process continues unless a HALT instruction is encountered

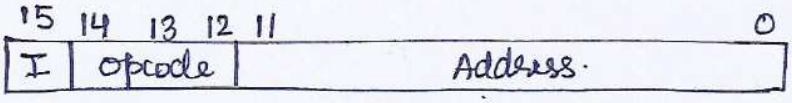


The flowchart represents initial configuration of instruction cycle. and shows how the control determines the instruction type after the decoding.

Types of instructions:

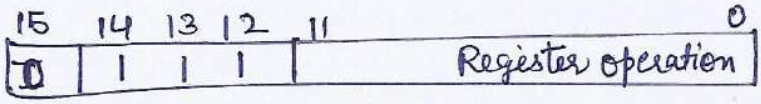
Memory Reference Instructions

opcode from 000 to 110 i.e $D_7=0$



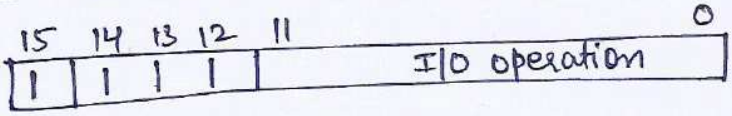
Register Reference Instruction

opcode 111 i.e $D_7=1$, $I=0$



Input-output Instruction

opcode 111 i.e $D_7=1$, $I=1$



Register Reference Instructions

(1)	CLA	$AC \leftarrow 0$	clear AC
(2)	CLE	$E \leftarrow 0$	clear E
(3)	CMA	$AC \leftarrow \overline{AC}$	complement AC
(4)	CME	$E \leftarrow \overline{E}$	complement E
(5)	CIR	$AC \leftarrow shr\ AC$ $AC(15) \leftarrow E$ $E \leftarrow AC(0)$	circulate AC right

E ⇒ End carry

(6)	CIL	AC ← shl AC AC(0) ← E E ← AC(15)	circulate left.
(7)	INC	AC ← AC+1	increment AC
(8)	SPA	if (AC(15)==0) then PC ← PC+1	skip if positive
(9)	SNA	if (AC(15)==1) then PC ← PC+1	skip if negative
(10)	SZA	if (AC==0) then PC ← PC+1	skip if AC Zero
(11)	SZE	if (E==0) then PC ← PC+1	skip if E zero
(12)	HLT	S ← 0 (S is a start-stop flip flop)	Halt computer

Memory Reference Instructions:

operation Decoder.

(1) AND — D_0

D_0T_4: DR ← M[AR]

D_0T_5: AC ← AC ∧ DR, SC ← 0

(2) ADD — D_1

D_1T_4: DR ← M[AR]

D_1T_5: AC ← AC + DR, E ← Cout, SC ← 0

(3) LDA — D_2 — Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

(4)	STA ↑ store AC	D_3	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$.
(5)	BUN Branch unconditionally	D_4	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
(6)	BSA ↑ Branch and save Return address	D_5	D_5T_4: D_5T_5:	$M[AR] \leftarrow PC$, $AR \leftarrow AR+1$ $PC \leftarrow AR$, $SC \leftarrow 0$
(7)	ISZ Increment & and skip if zero	D_6	D_6T_4: D_6T_5: D_6T_6:	$DR \leftarrow M[AR]$ $DR \leftarrow DR+1$ $M[AR] \leftarrow DR$, if $(DR=0)$ then $PC \leftarrow PC+1$ $SC \leftarrow 0$

Note that we need seven time signals (T_0 to T_6) to execute the longest instruction ISZ

Input - Output Instructions:

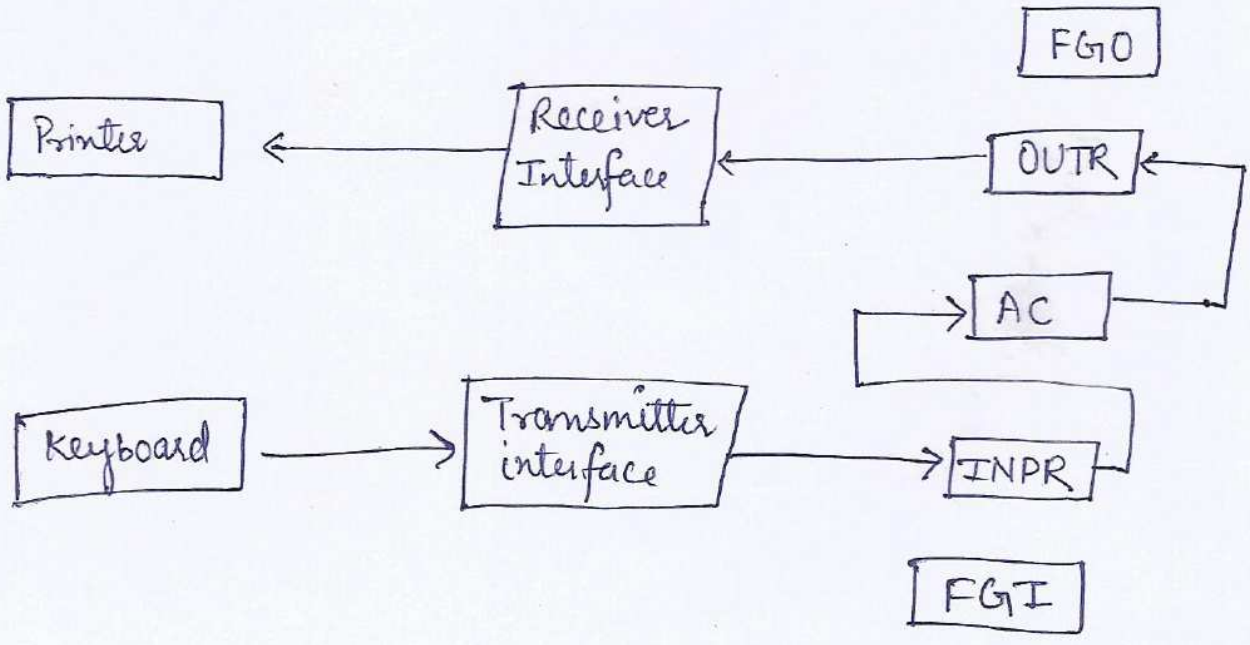
INP	AC(0-7) ← INPR , FGI←0	input character
OUT	OUTR ← AC(0-7) FGO←0	output character
SKI	if(FGI=1) then PC←PC+1	skip on input flag
SKO	If(FGO=1) then PC←PC+1	skip on output flag
ION	IEN←1	interrupt enable ON
IOF	IEN←0	interrupt enable Off

input - output configuration

input - output Terminal

Serial communication interface

Computer Register and flip flops.



INPR ⇒ 8 bit input Register

AC ⇒ Accumulator

FGI ⇒ 1 bit control flip flop. This flag is set to 1 when new information is available. And, this flag is clear to zero when information is accepted by the computer.

OUTR ⇒ output Register (8 bits)

FGO ⇒ is 1 bit control flip flop. If FGO is 1 then information is transferred from AC to to OUTR. And then FGO is clear to zero.

Instruction codes:-

* The organization of a computer can be defined by its internal registers, the timing and control structure, and the set of instructions it uses.
* The internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers.

→ A computer instruction is a binary code that specifies a sequence of microoperations for the computer.

→ Instruction codes together with the data are stored in memory. (called stored program concept)
The computer reads each instruction from memory. The control then interprets the binary code of instruction and proceeds to execute it by using a sequence of microoperations. Every computer has its own unique instruction set.

→ Instruction code is a group of bits that instructs the computer to perform a specific operation. It is usually divided into parts

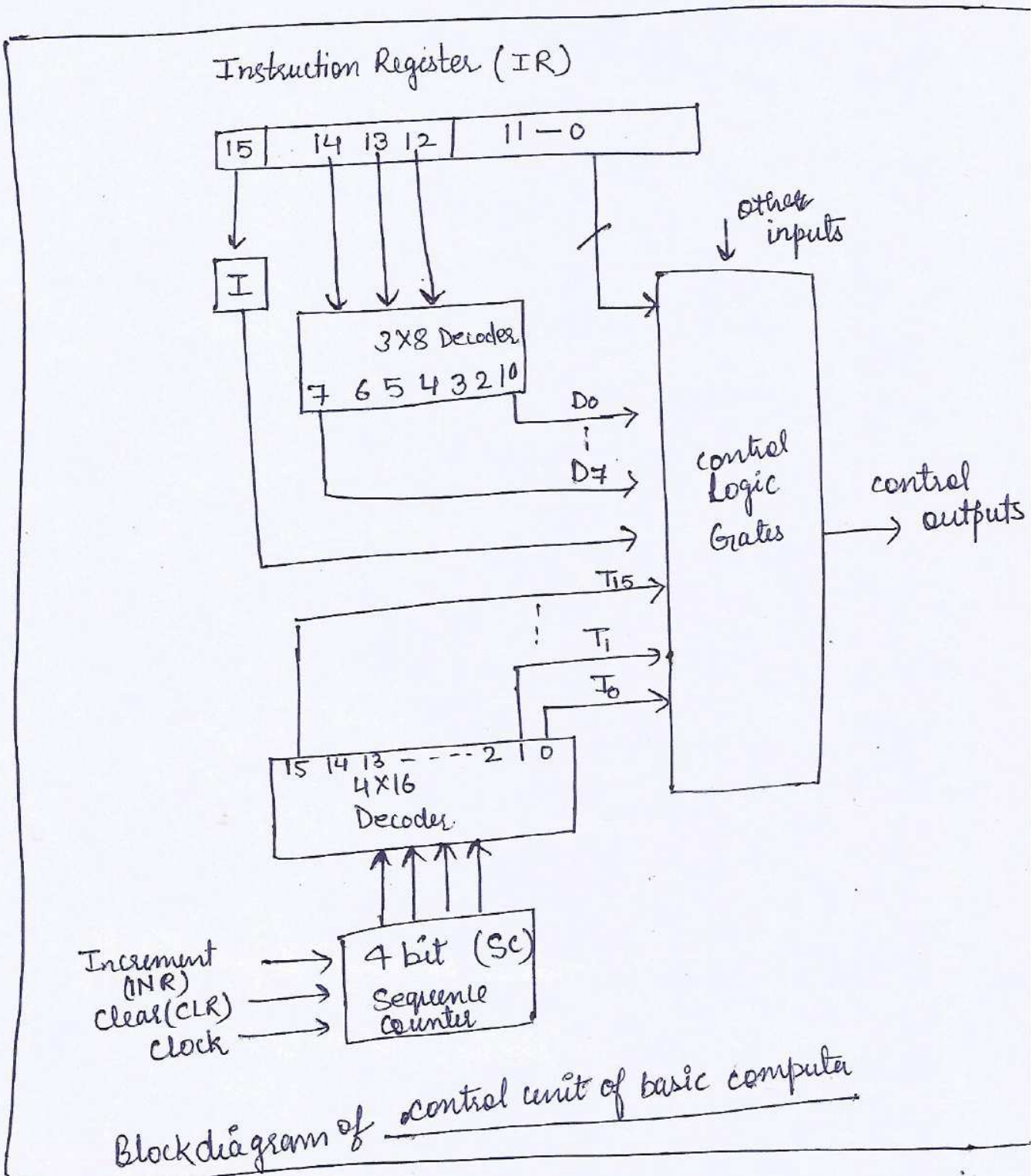
→ **Operation code:** operation code part of an instruction is a group of bits that define operations such as add, subtract, shift, complement or increment etc.

The number of bits in opcode = n will depend on the number of operations in computer.

2^n ⇒ number of operations

Control Unit:

control unit generates control signals. control signals provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.



Block diagram of control unit of basic computer

→ components of control unit are

① → Two decoders
② → One sequence
③ → control logic gates.

- An instruction is read from memory is placed in the IR register.
- In control unit the IR is divided into three parts
  - I bit (Indirect bit)
  - the operation code bit (12-14)
  - bit 0 to 11

- The operation code bits are decoded by a 3×8 decoder as outputs D_0 to D_7.
- Bit 15 is transferred to a flip flop I.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit Sequence counter can count in binary from 0 to 15. The outputs of counter are decoded into 16 timing signals T0 to T15

- The sequence counter SC can be incremented or cleared. Most of the time, the counter is incremented to provide the sequence of timing signals, Once in awhile, the counter is cleared to zero. causing the next active timing signal to be T0.

Example consider a case where SC is incremented to provide the signals T0, T1, T2, T3 and T4 in sequence. At the time of T4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically.

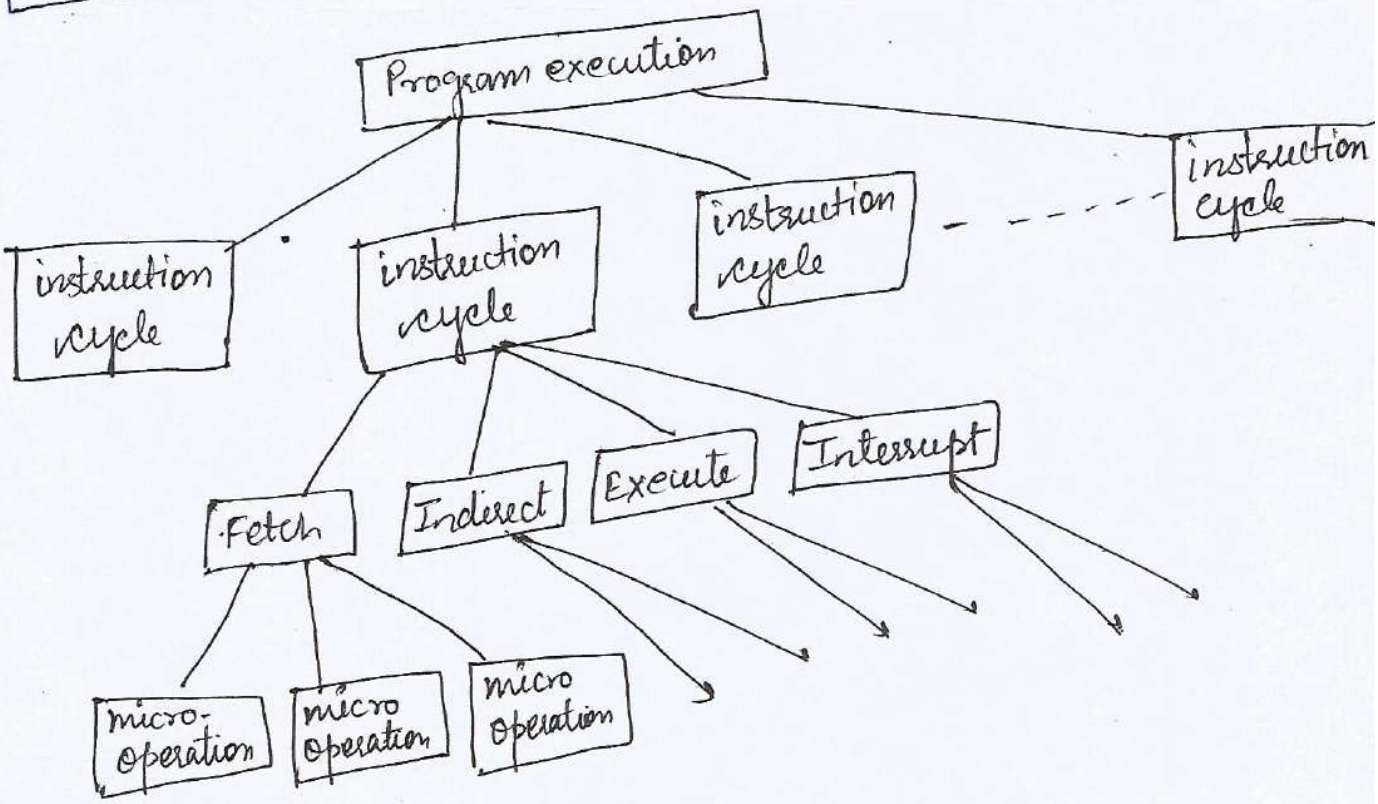
$$D_3T_4: SC \leftarrow 0$$

Micro operation

→ The execution of an instruction involves the execution of a sequence of substeps, generally called cycles.

For example, an execution may consist of fetch, indirect, execute and interrupt cycles. Each cycle is in turn made up of a sequence of more fundamental operations called microoperations.

> A single microoperation generally involves a transfer between registers, transfer between a register and an external bus or a simple ALU operation



Ques: what is difference between operation (macro operation) and micro-operation.

Ans: An operation is the part of instruction. An operation is a binary code that tells the computer to perform the specific task or operation.

The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro operations in internal computer registers.

For every operation code, the control issues a sequence of micro operations. For this reason, an operation code is sometimes called a macro operation because it specifies a set of micro operations.

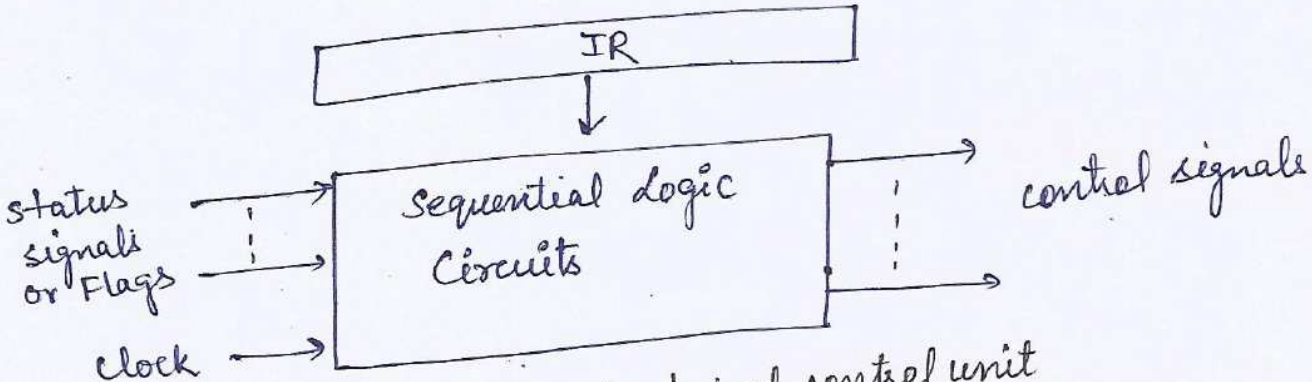
Hardwired and Microprogrammed control:

→ There are two types of control organization. A control unit can be implemented using two techniques:

- → Hardwired control unit
- → Microprogrammed control unit

Hardwired control:

- In hardwired organization, the control unit is implemented with gates, flip flops, decoders and other digital circuits.
- When the control signals are generated by the hardware using conventional logic design techniques, the control unit said to be hardwired.



Structure of a hardwired control unit

→ Hardwired control unit uses a fixed logic to interpret an instruction and generate appropriate signals.

Micro programmed control

A micro-programmed control unit is implemented using programming approach.

→ The control unit initiates a series of sequential microoperations. The control variables can be represented by a string of 1's and 0's called a control word.

A microprogrammed control unit is a control unit whose binary control variables are stored in memory

Control memory: A memory that is part of control unit is called control memory. control memory is the storage in the microprogrammed control unit to store the micro program.

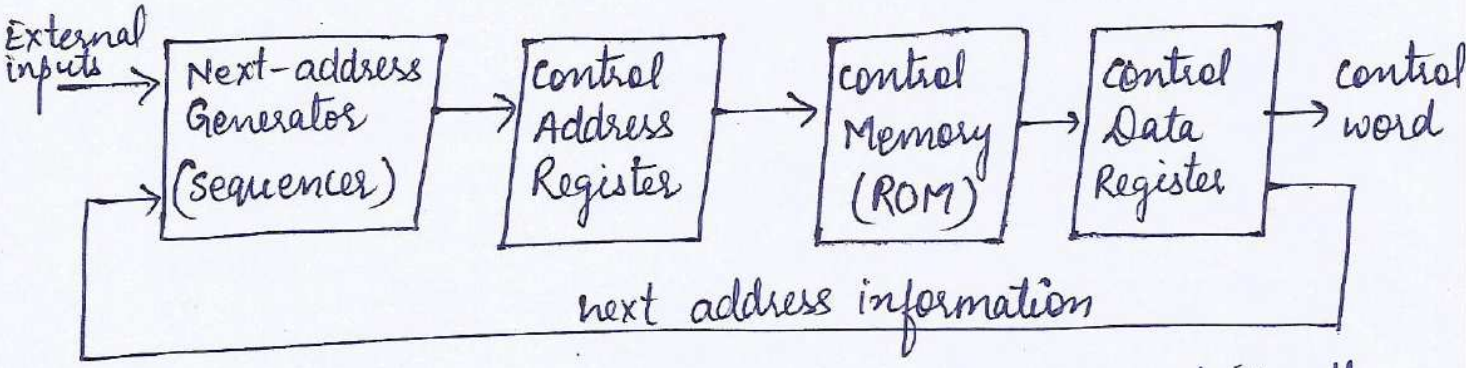
micro-instruction: Each word in control memory contains a micro instruction. A micro instruction specifies one or more micro operations for the system.

micro program: A sequence of micro instructions constitutes a ~~micro opera~~ microprogram.

Microcode: A logically coherent portion of micro program is called micro-code.

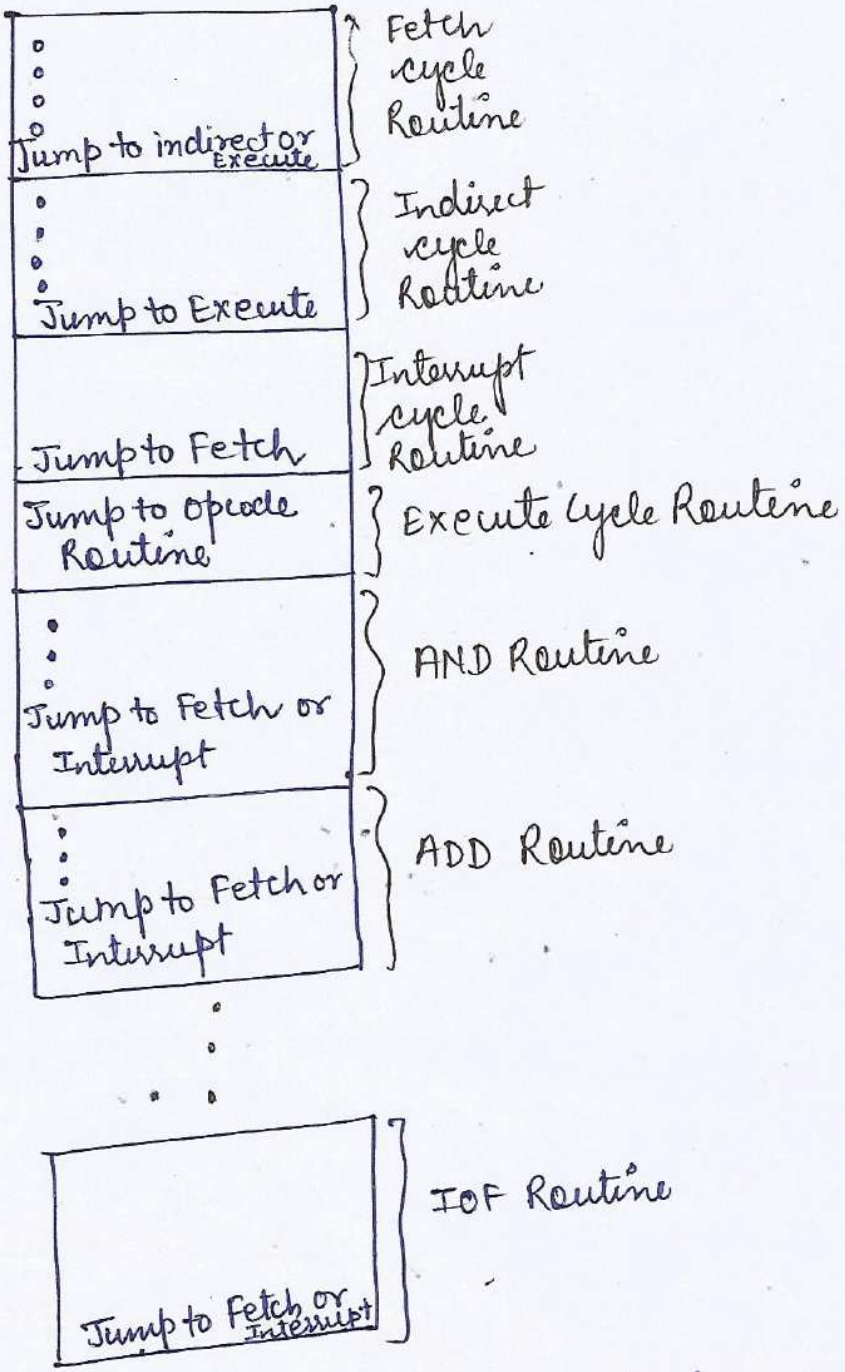
A micro instruction can cause execution of one or more micro operations and a sequence of micro instruction (micro program) can cause the execution of an instruction.

Micro-programmed control organization



- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control address register specifies the address of the microinstruction and control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more micro operations for the processor. Once these operations are executed, the control must determine the next address.
- The location of the next micro instruction may be the one next in the sequence or it may be located somewhere else in the control memory.
- The next address generator is sometimes called a micro program sequencer as it determines the address sequence that is read from the control memory.

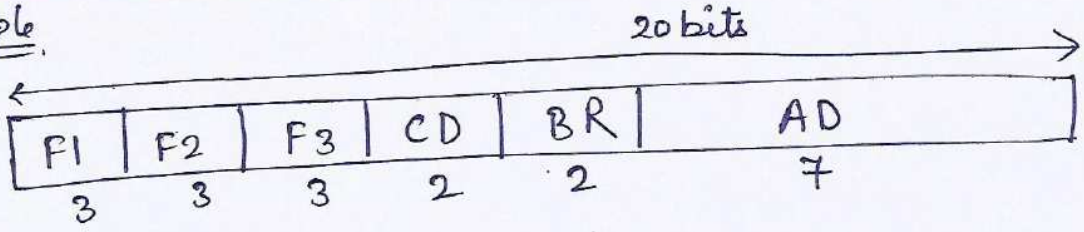
Organization of Control Memory:



This shows control words or microinstruction arranged in control memory. The microinstruction in each routine are to be executed sequentially. Each Routine ends with a branch or jump instruction indicating where to go next.

Micro instruction Format :

Example.



→ F1, F2, F3 ⇒ specify micro operations
each F defines seven distinct micro operations
None for zero (000)
Different operations for (1 to 6)

→ CD ⇒ Field selects status bit conditions

→ BR ⇒ field specifies the type of branch to be used

→ AD ⇒ This field contains a branch address. The size of this field depends on the control memory size.
Here AD has 7 bits that means ⇒ 2^7 words in control memory.

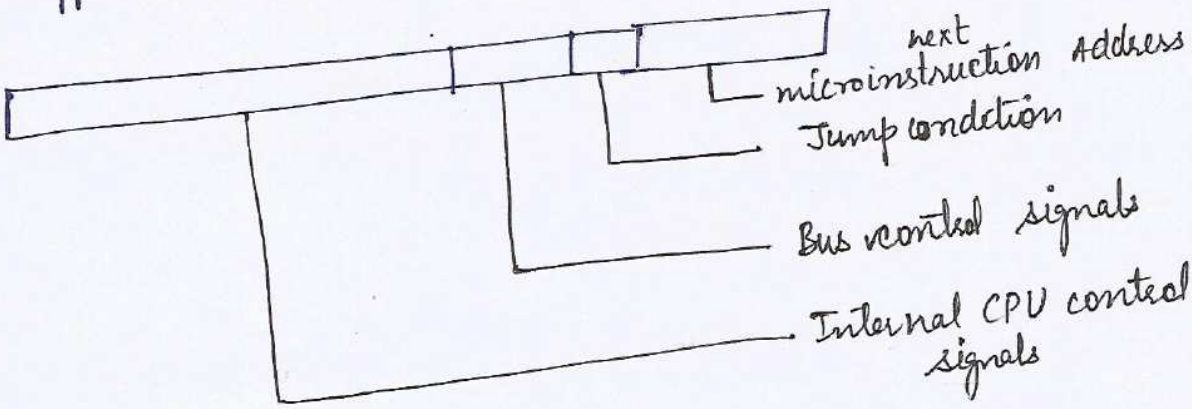
Horizontal Microprogrammed Control unit:

→ The control signals are represented in decoded binary format that is 1 bit per one control signal.

→ If 53 control signals are present in the processor then 53 bits are required

→ More than 1 control signal can be enabled at a time.

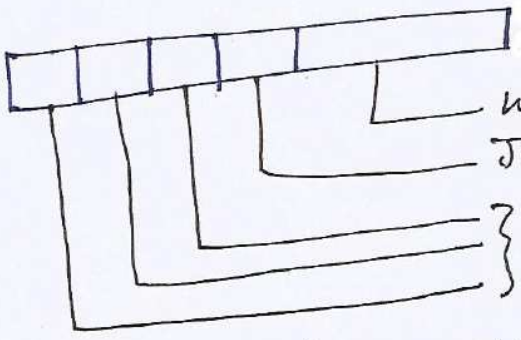
→ It supports longer control word.



→ It requires no additional hardware (decoders).

→ It is faster than Vertical microprogrammed.

Vertical Microprogrammed control unit:



→ The control signals are represented in the encoded binary format.

- For N control signals $\log_2 N$ bits are required.
- It supports shorter control words.
- It requires an additional hardware (decoders) to generate control signals.
- It is slower than horizontal microprogrammed.

Ques: suppose there are 64 signal control signals in the system and control 1024 word control memory then find out the size of microinstructions horizontal and vertical both.

Horizontal:
- 64 bits for 64 control signals
- Address bits ⇒ 10 because words = $1024 = 2^{10}$

size of microinstruction

control signal bits	Address bits
64	10

⇒ 74 bits

Vertical:

control signals = $64 \Rightarrow 2^6$

Number of bits for control signals = 6.

Address bits ⇒ 10. because $1024 \Rightarrow 2^{10}$.

size of vertical micro instruction ⇒

control signal bits	Address bits
6	10

⇒ 16 bits

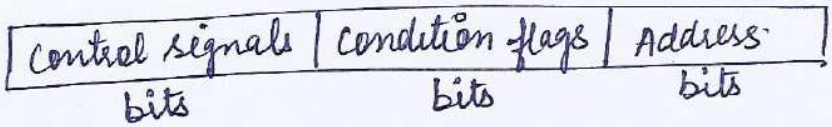
Ques: consider a Microprogrammed CU, where 1024 word control memory is used. The CU has to support 50 control signals and 16 flag conditions.

(i) How many bits required in control word.

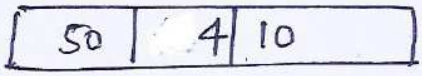
(ii) what is the size of control memory

Ans:

Format of control word or microinstruction:



Horizontal microprogramming:

size of control word or microinstruction ⇒ 

⇒ 64

Number of bits for control signals ⇒ 50 (bit per control signal)

Number of bits for flags ⇒ 4 because $16 \Rightarrow 2^4$

Number of address bits ⇒ 10 because $1024 = 2^{10}$.

The size of each control word ⇒ 64 bit.

control memory has words ⇒ 1024

Size of control memory = 1024 × 64 bits

⇒ 1KB × 8B

⇒ 8KB

Vertical Microprogramming:

control signal bits ⇒ 6 because Control signals ⇒ 50

flag bits ⇒ 4

Address bits ⇒ 10

size of control word ⇒ 20 bits ⇒ (6+4+10)

size of control memory ⇒ 20 × 1024

→ Micro program Sequencer:

→ The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.

→ Typical functions of a microprogram sequencer are:
- → Incrementing the control address register by one.
- → Loading into the control register an address from control memory
- → Transferring an external address
- → Loading an initial address to start the control operations.

→ Address Sequencing:

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

→ Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction

→ Steps that the control must undergo during the executing of a single computer instruction.
- → An initial address is loaded into the control register address register when power is turned on in the computer
- → This address is usually the address of the first micro instruction that activates the instruction fetch routine.

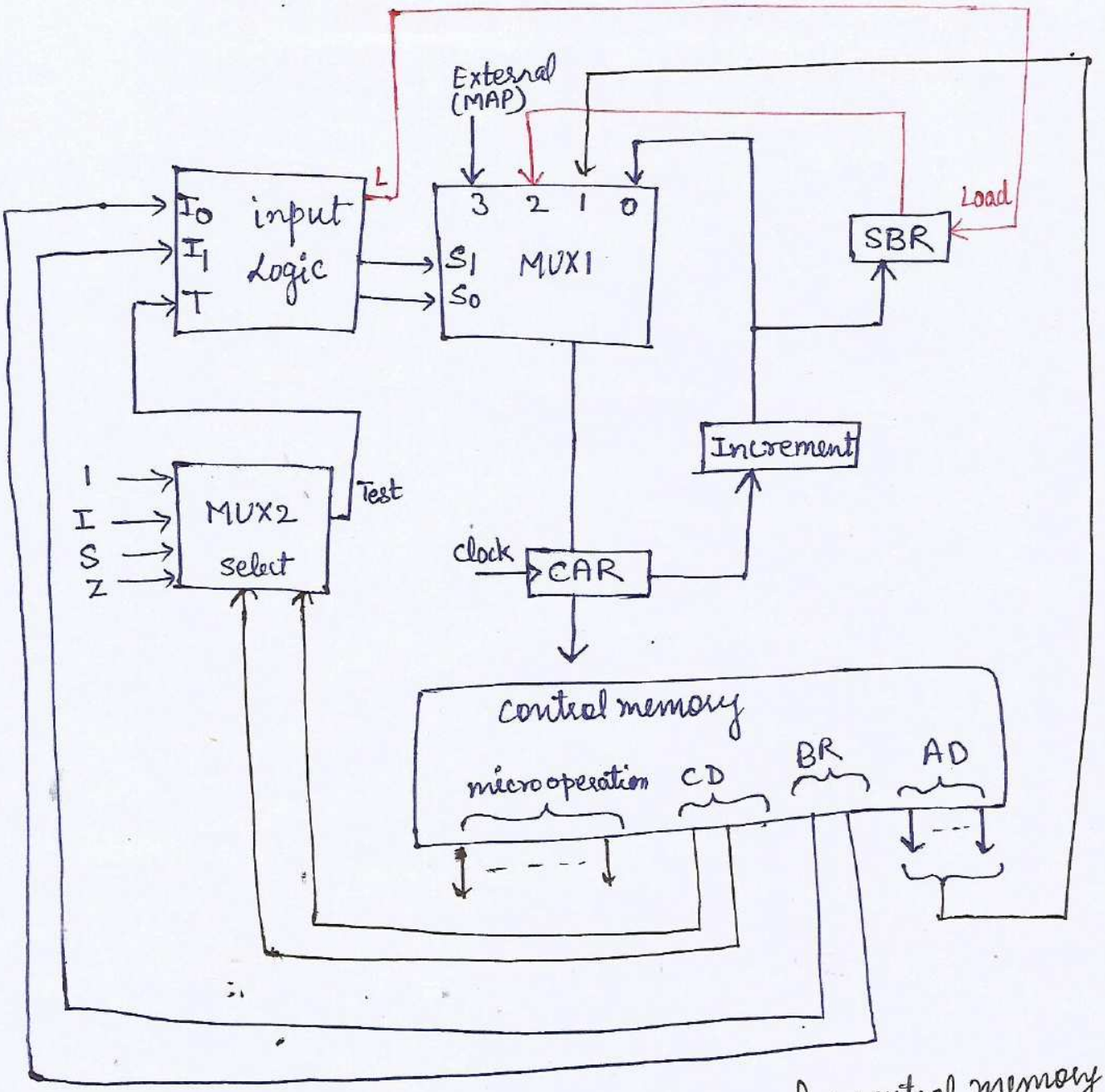
→ The control unit must go through the routine that determines the effective address of the operand.

→ The next step is to generate the micro operations that execute the instruction fetched from the memory.

Transformation from the instruction code bits to an address in control memory where the routine is located is referred to as mapping process

→ Address sequencing capabilities required in a control memory are:
  → Incrementing the CAR
  → Unconditional branch or conditional branch, depending on status bit conditions
  → Mapping process from the bits of the instruction to an address for control memory.
  → A facility for subroutine call or return.

→ The micro-instruction contains
  → a set of bits to initiate microoperations in control registers.
  → other bits to specify the method by which the next address is obtained.



micro program sequencer for control memory

Explanation:

- The first multiplexer selects an address from one of four sources (incremented External, from micro instruction or SBR) and routes it into CAR.
- The second multiplexer tests the value of a selected status bit and the result of test is applied to input logic.
- SBR ⇒ Sub Routine Register.

CD field of microinstruction selects one of the status bits in the second multiplexer.

CD	condition	Symbol	
00	always = 1	U	unconditional branch
01		I	Indirection bit
10		S	Signbit
11		Z	Zero value in AC

BR ⇒ Branch type.

00	JMP
01	CALL
10	RET
11	MAP

Input logic design ⇒

$S_1 = I_1$

$S_0 = I_1 . I_0 + \overline{I_1} T$

$L = \overline{I_1} I_0 T$

Reduced Instruction Set Computer (RISC)

The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set.

Characteristics of RISC Processors:

→ Relatively few instructions
→ Relatively few addressing modes
→ Memory access limited to load and store instructions
→ All operations done within the register of CPU
→ Fixed length, easily decoded instruction format
→ Single cycle instruction execution
→ Hardwired rather than microprogrammed.
→ Simple load and store operations for memory access.

Complex Instruction Set Computers (CISC)

The essential goal of CISC architecture is to provide a single machine instruction for each statement that is written in high level language.

Characteristics of CISC architecture:

- A large number of instructions (from 100 to 250 typically)
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes — typically from 5 to 20 different modes.
- Variable length instruction formats.
- Instructions that manipulate operands in memory.

Pipelining:

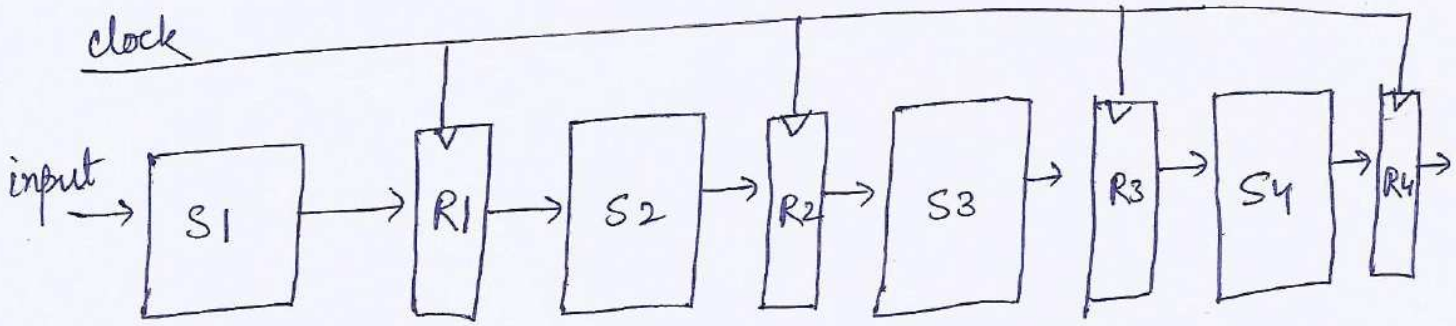
→ Pipeling is an implementation technique whereby the multiple instructions are overlapped in execution.

→ Pipeline processing is an implementation technique where arithmetic suboperations or the phases of a computer instruction cycle overlap in the execution.

→ Pipeline is divided into stages or segments and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

→ Pipeline increases the overall instruction throughput.

→ In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of the next segment combinational circuit is applied to the input registers of the next segment.



Four segment pipeline

Types of pipeline:

→ Arithmetic Pipeline
→ Instruction Pipeline

Arithmetic pipeline:

→ found in very high speed computers
→ Arithmetic suboperations overlapping.
→ used to implement floating point operations, multiplication of fixed point numbers ~~etc~~ etc.

Instruction pipeline:

- In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle.
- ~~increase~~ increases throughput of the system.

Advantages of pipelining:

- The cycle time of processor is reduced.
- It increases throughput of the system.
- It makes system reliable.

Disadvantages of pipelining

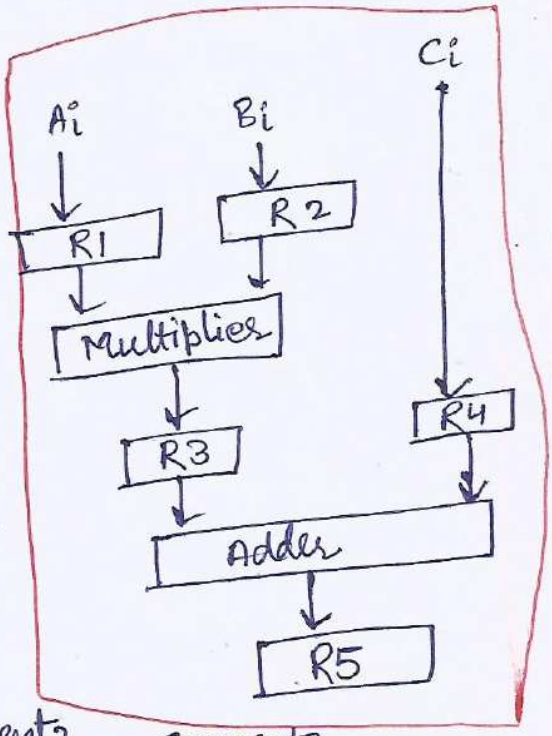
→ The design of a pipelined processor is complex and costly to manufacture.
- Instruction latency is more.

Example for pipeline organization

- Example of performing combined multiply and add operations with a stream of numbers.

$A_i * B_i + C_i$ for $i \leftarrow 1, 2, 3 - - - 7$.

Each suboperation is to be implemented in a segment with a pipeline.



- suboperations

$R1 \leftarrow A_i \quad R2 \leftarrow B_i$

$R3 \leftarrow A_i * B_i \quad R4 \leftarrow C_i$

$R5 \leftarrow R3 + R4$

The five registers are loaded with new data every clock pulse.

Clock pulse	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1			—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$

⋮ so on.

General Considerations :

⇒ some applications need to repeat the same task many times with different sets of data.

⇒ The operands pass through all the segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a suboperation over the data stream flowing through the pipe.

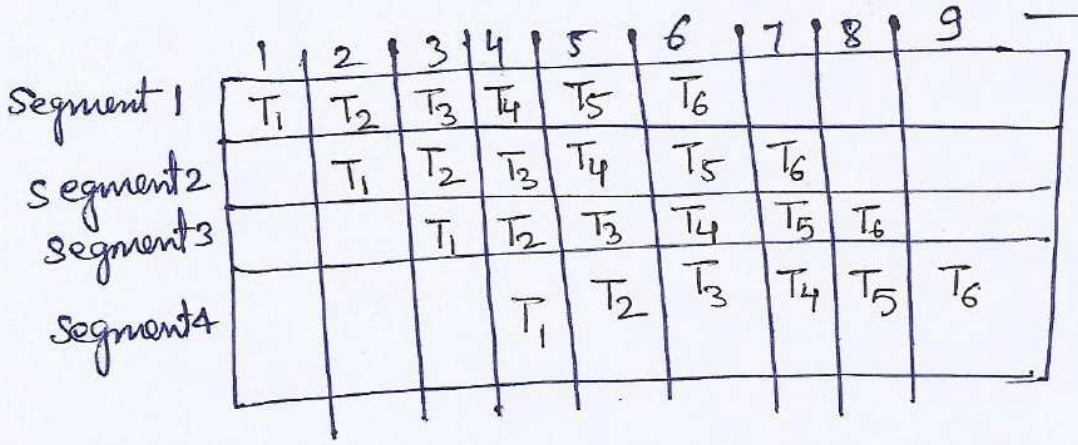
⇒ The segments are separated by registers R_i that hold the intermediate results between the stages.

⇒ Information flows between adjacent stages under the control of a common clock pulse applied to all registers simultaneously.

Task: a task as the total operation performed going through all the segments in the pipeline.

Space-time diagram: (For four pipeline stages)

⇒ shows the segment utilization as a function of time.



Assumed Number of task = 6

Instruction Pipeline

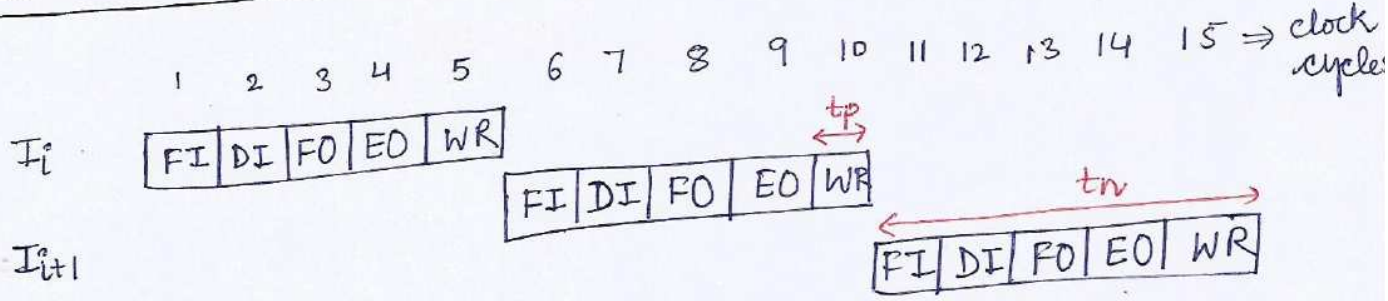
⇒ The phases of a computer instruction cycle overlap in the execution.

Instruction cycle ⇒ 5 phases.

FI — Fetch instruction
DI — Decode instruction
FO — Fetch operands
EO — Execute operation
WR — Write Result.

Suppose there are three instructions to execute I_i , I_{i+1} , I_{i+2}

Instruction execution in nonpipeline architecture



Number of cycles to complete 3 tasks ⇒ 3X cycle to complete 1 task.
⇒ 3X5
= 15

Time to execute 3 tasks ⇒ 3X tn
= 3X time to execute or complete one task

Instruction execution in pipeline architecture

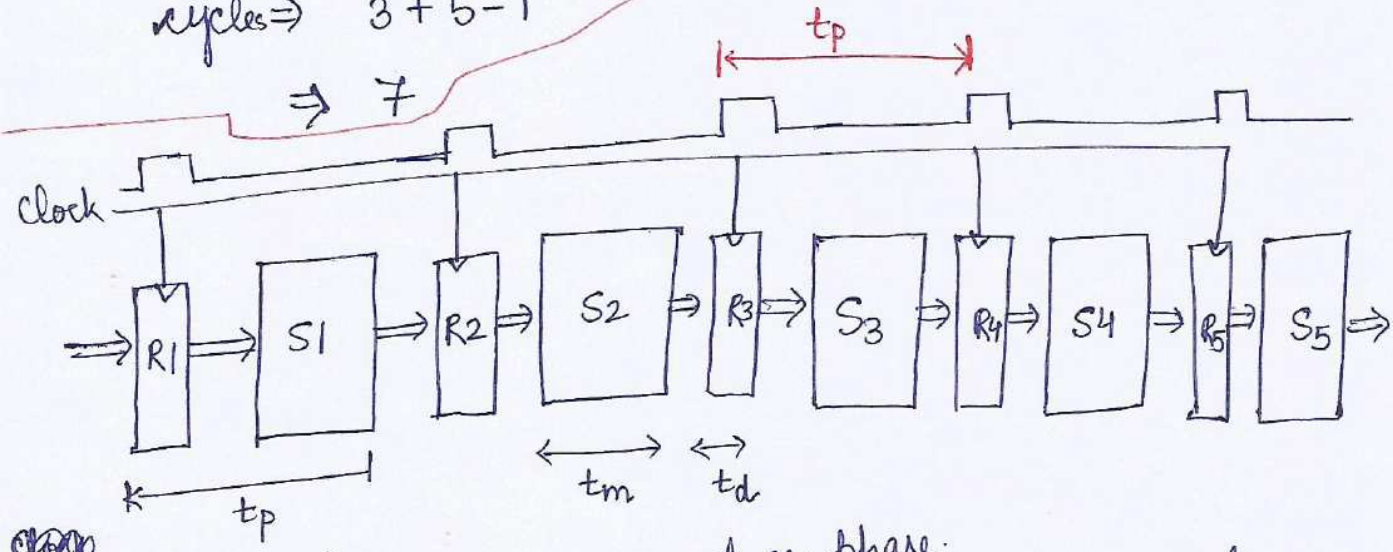
	1	2	3	4	5	6	7
I_i	FI	DI	FO	EO	WR		
I_{i+1}		FI	DI	FO	EO	WR	
I_{i+2}			FI	DI	FO	EO	WR

⇒ clock cycles.

Number of cycle required to complete n tasks in K-stage pipeline = K+n-1

n = 3 (3 instructions)
K = 5 (5 stages of instruction cycle)

cycles ⇒ 3 + 5 - 1
⇒ 7



t_p ⇒ time taken by a segment or phase.
t_m ⇒ time taken by combination circuit of segment.
t_d ⇒ propagation delay of register.

$$t_p = \max_{i=1}^{k} (t_i) + t_d$$

t_p = max time taken by any segment + Register delay

5-Segment pipeline

Formulas:

consider k-segment pipeline (or phases) ⇒ means Number of segments ⇒ k

Number of tasks to be executed = n

clock cycle time = t_p

Time taken by task 1 to complete ⇒ $k \cdot t_p$

The remaining (n-1) tasks, time equal to $(n-1) \cdot t_p$

The time taken by n tasks using k-segment pipeline

$$= k t_p + (n-1) t_p$$

$$= [k + (n-1)] \cdot t_p$$

Number of cycle by n tasks using k-segment pipeline

⇒ $K + (n-1)$

consider a non-pipeline unit that performs the same task and takes a time equal to t_n to complete each task.

then time required to complete n tasks. ⇒ $n t_n$

$$\text{Speedup ratio } S = \frac{\text{time taken by non-pipeline}}{\text{time taken by pipeline system}}$$

As the number of tasks increases n becomes much larger than k-1, and k+n-1 approaches the value of n

under this condition the speedup ⇒ $S = \frac{t_n}{t_p}$

we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuit, we will have

$$t_n = Kt_p \Rightarrow$$

Including this assumption the speedup reduces to

$$S = \frac{kt_p}{t_p} = K$$

This shows that theoritical maximum speedup that a pipeline can provide is k (number of segment in pipeline)

Frequency $= \frac{1}{t_p}$

Throughput $H_k = \frac{n}{(k+n-1)t_p}$ or $\frac{nf}{(k+n-1)}$

Throughput: The number of tasks performed per time unit

Efficiency $= \frac{S_k}{K}$

Latency: The number of time units (clock cycle) between two initiations of a pipeline is the latency between them.

Collision: Any attempt by two or more initiations to use the same pipeline stage at the same time will cause a collision

Forbidden Latencies:- that will cause collision

Permissible latencies:- that will not cause collision

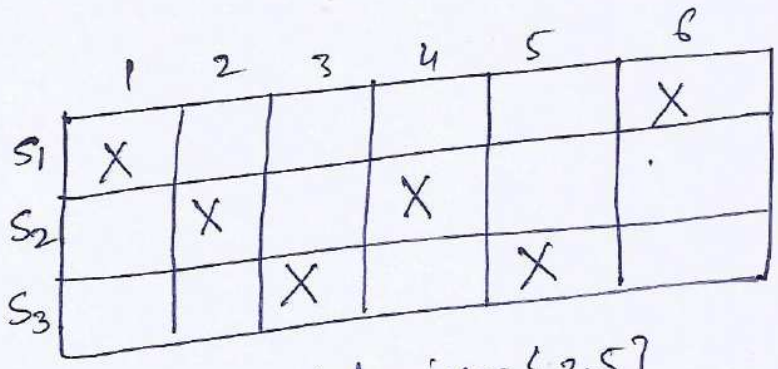
Reservation Table: A reservation table is a two-dimensional table

Row ⇒ pipeline stage or processing element

Column ⇒ represents cycles in the execution of a task

An X is placed in the entries corresponding to the stages required by the task during that cycle.

⇒ Forbidden latencies ⇒ distance between two X (check marks) in the same row of the reservation table.



⇒ 1, 6 ⇒ 6-1 ⇒ 5

⇒ 2, 4 ⇒ 4-2 ⇒ 2

⇒ 3, 5 ⇒ 5-3 = 2.

Forbidden latencies = {2, 5}

Permissible latencies ⇒ {1, 3, 4}

- if reservation table has n columns, the maximum forbidden latency is $m \leq (n-1)$
- permissible latency p will satisfy $1 \leq p \leq m-1$

collision vector collision vector is an m-bit vector
m = maximum forbidden latency.

$$C = C_m C_{m-1} - - - - C_2 C_1$$

$C_i = 1$ if latency i is a forbidden latency.

$C_i = 0$ otherwise

Previous Example: Forbidden latencies = {2, 5}

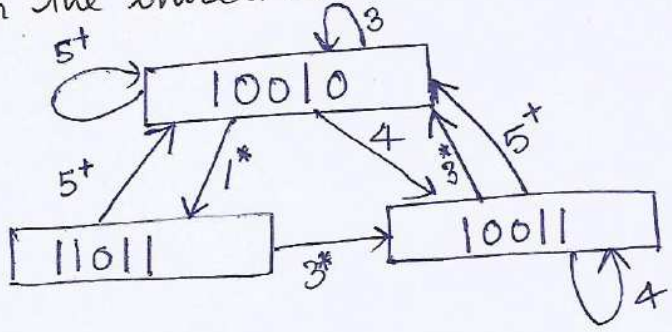
$$C = C_5 C_4 C_3 C_2 C_1$$
$$1\ 0\ 0\ 1\ 0$$

State Diagram:

From collision vector, we can construct a specifying the permissible state transitions among the successive iterations

⇒ collision vector forms the initial state

⇒ The next state at time t+p is obtained by shifting the present state p-bits to the right and ORing with the initial collision vector.



5+ means 5 or more than 5 bits shift

Latency Sequence: A latency sequence is a sequence of permissible non forbidden latencies between successive task initiations.

Latency Cycle: A latency cycle is a latency sequence which repeats the same subsequence (cycle) indefinitely.

constant cycle:- A constant cycle is a latency cycle which contains only one latency value.

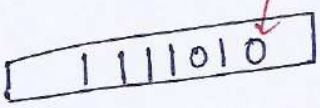
Simple cycle: A simple cycle is a latency cycle in which each state appear only once.

Greedy cycle: Some of the simple cycles are greedy cycles.

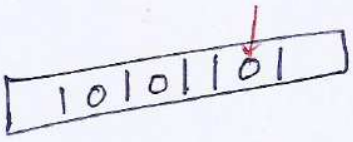
A greedy cycle is one whose edges are all made with the minimum latencies from their respective starting states.

Greedy cycles are formed only using outgoing edges with minimum latencies.

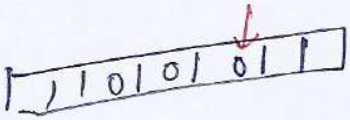
Suppose state



minimum latency (permissible) = 1



min latency ⇒ 2.



min latency = 3

Note

Minimum latency respect respective to state is marked with * in state diagram.

- when number of shifts is m+1 or greater, all transitions are redirected back to the initial state.

Ques:

How to draw state diagram

collision vector 10010 ⇒ forms initial state.
it has permissible latency ⇒ 1, 3, 4

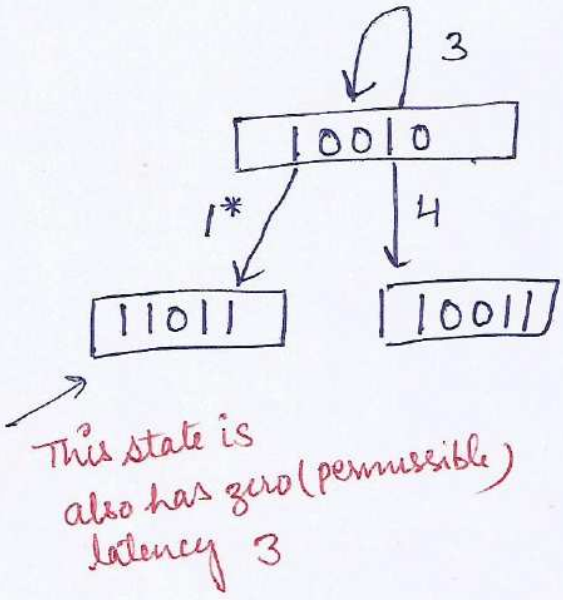
① do shifts 1, 3, or 4
② ORing with CV

initial state 10010

case 1 ⇒ 10010
↓ 1 shift Right
01001
10010 ORing
11011 next state

case 3 ⇒ 10010 — CV.
00010 — 3 shift Right
10010 — CV (ORing)
10010 — next state.

case 4 ⇒ 10010
00001 — 4-shift Right
10010
10011 — next state.



This state has permissible latencies 3, 4

This state is also has zero (permissible) latency 3

State ⇒ 11011

Case 3 shift of 3.

00011 ⇒ shift of 3.
10010 ⇒ CV
10011 ⇒ ORing ⇒ next state (already present state)

State : 10011 Permissible latency ⇒ 3, 4.

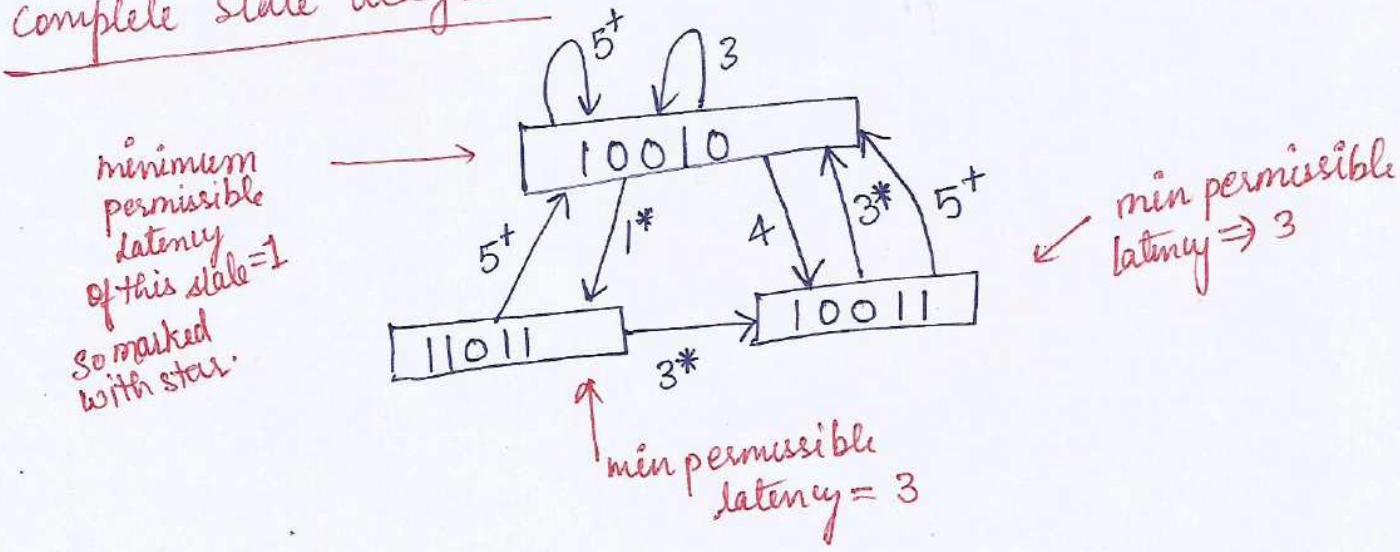
Case 3:- shift of 3

10011 → 3 shifts Right.
00010
10010 ORing.
10010

Case 4: shift of 4.

10011 → 4 shifts Right
00001
10010 — ORing
10011

Complete state diagram



Simple cycle:

(3), (5), (1,5), (1,3,5), (4,3), (4,5) and so on.

Constant cycle:

(3), (5)

Greedy cycles:- Examples ⇒ (1,5), (3,4)

MAL Minimum Average Latency

• Simple cycles	Average latency
3	3
5	5
(1,5)	$\frac{1+5}{2} = 3$
(1,3,5)	$\frac{1+3+5}{3} = 3$
(4,3)	$\frac{4+3}{2} \Rightarrow 3.5$

So MAL = 3

$$\text{Throughput} = \frac{1}{\text{MAL}} = \frac{1}{3} \Rightarrow 16.67 \text{ million operations per second.}$$

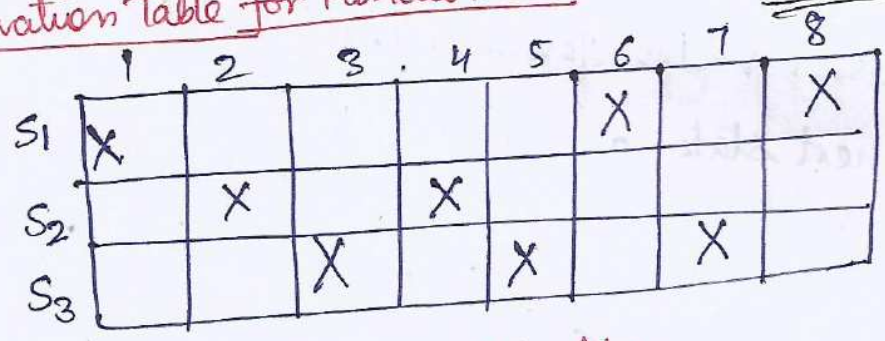
Throughput = "if minimum constant cycle used.

minimum constant cycle = 3.

$$\text{Throughput} = \frac{1}{3} \text{ MOPS.}$$

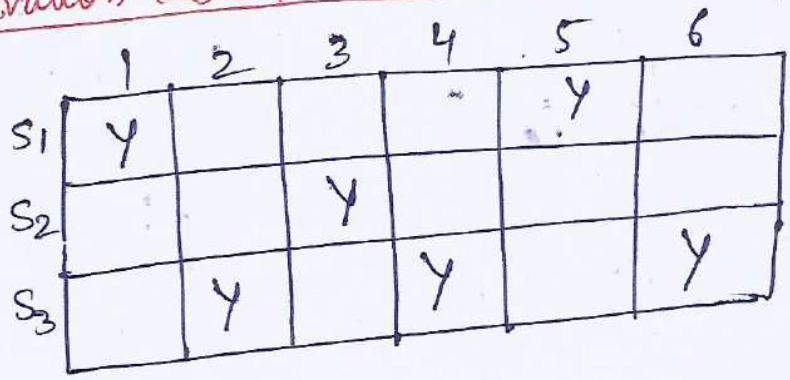
Question

Reservation Table for Function X:



Forbidden latencies
⇒ 2, 4, 5, 7

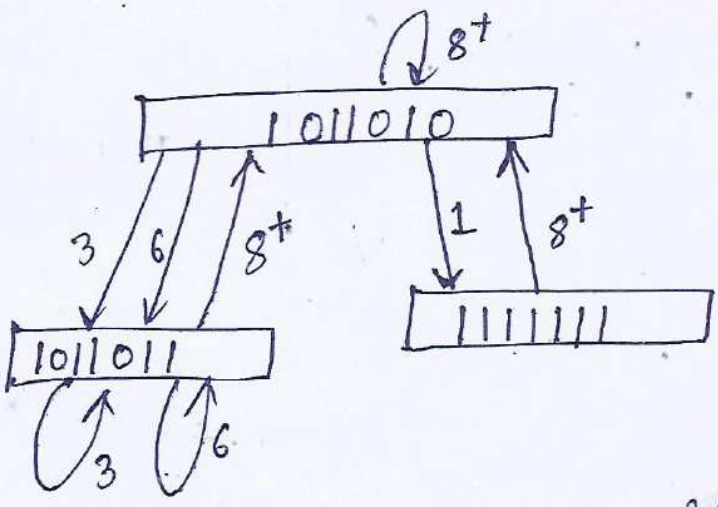
Reservation Table for function Y:

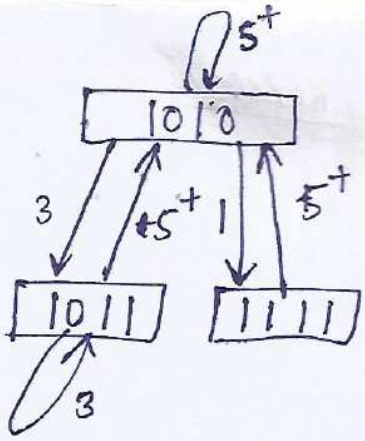


Forbidden latencies
⇒ 2, 4.

⇒ if reservation table has n columns & the maximum forbidden latency is $m \leq (n-1)$

⇒ permissible latency p will satisfy $1 \leq p \leq m-1$





possible cycles ⇒ (1,8)
(3)
(6)

MAL = 3.

Average latency
$\frac{1+8}{2} \Rightarrow 4.5$
⇒ 3
⇒ 6

(1,5) ⇒ 3.0
3 3.0
(3,5) ⇒ 4.0

MAL ⇒ 3.