



COA Unit 3 Notes

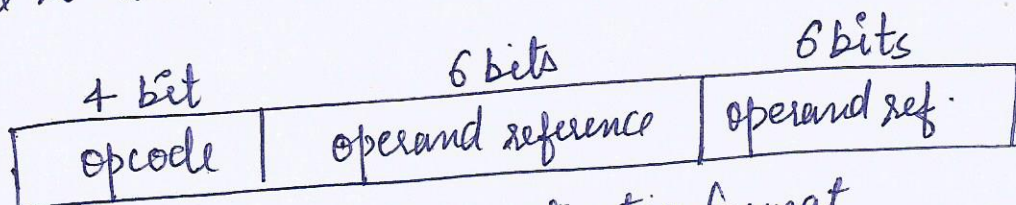
Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

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 processors 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 simple instruction format

→ source ~~and~~ 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 mneumonics 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 ~~are~~ 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 ~~word~~ from a processor register into memory.
- Load: a transfer from memory to register.
- Exchange: swaps information ~~is~~ 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 - $\rightarrow$ compute the sum of two operands
- Subtract $\rightarrow$ Compute difference of two operands
- Multiply $\rightarrow$ compute the product of two operands
- Divide $\rightarrow$ Compute quotient of two operands.
- Add with carry $\rightarrow$ compute the sum of two operands with carry
- Subtract with borrow $\rightarrow$ compute the difference with borrow.
- Negate (2's complement) $\rightarrow$ change the sign of the operand.
- Increment $\rightarrow$ Add 1 to operand
- Decrement $\rightarrow$ 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 $\rightarrow$ performs AND operands
- OR $\rightarrow$ performs OR of operands
- Complement $\rightarrow$ takes complement
or NOT
- X-OR $\rightarrow$ performs XOR of operands

clear selected $\Rightarrow$ to clear a bit or selected bits to zero bits

set selected bits $\Rightarrow$ to set a bit or selected bits to 1.

complement selected bits $\Rightarrow$ to complement a bit or selected bits.

Test $\Rightarrow$ Test specified condition (set flags)

compare $\Rightarrow$ make logical comparison of two operands.

Enable Interrupt $\Rightarrow$ } $\Rightarrow$ set control variables for interrupt handling, timer control etc.
Disable Interrupt $\Rightarrow$ }

Shift Instructions:

$\rightarrow$ 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, or CALL
Jump to specified address.

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

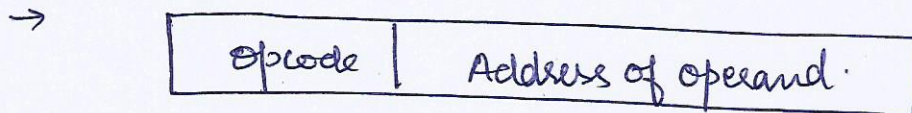
Branch if no overflow.

STORED PROGRAM ORGANIZATION: OR Single Accumulator Organization

OR

Processor Organization with Accumulator

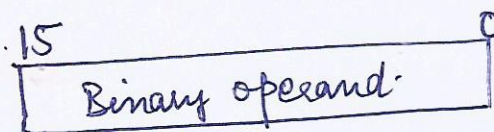
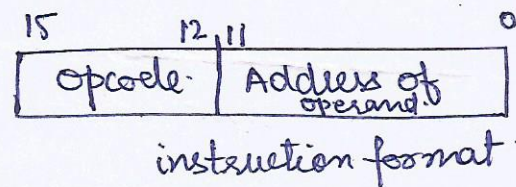
→ To organize a computer in this method, Processor has a register called accumulator and an instruction code format with two parts.



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

→ ~~Another~~ The other operand will be stored in the processor register

Example



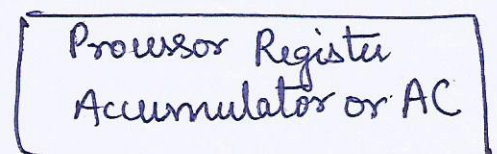
Memory 4096X16



Number of bits in address = ~~12~~ 12

Number of memory locations = 2^{12}
= 4096

The number of bits of operand or data = 16



In this method, processor performs the operation specified by the opcode on the data specified by ~~stored~~ stored on the memory location and content of AC register.

Instruction Format :

Most common fields of instructions are-

- operation field which specifies the operation to be performed
- Address field which contains the location of operand i.e., register or memory location
- Mode field which specifies how operand 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.

opcode	Destination address	Source Address	Source address	Mode
--------	---------------------	----------------	----------------	------

Examples:

ADD R1, R2, R3

OR

ADD R1, A, B

Two address instruction:

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

opcode	Destination address	Source address	Mode.
--------	---------------------	----------------	-------

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.

Opcode	Address of operand	Mode
--------	--------------------	------

Example

LOAD A

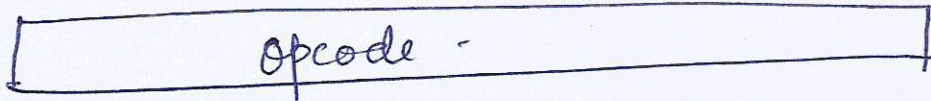
OR

ADD B

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.

Ex



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

$R1 \leftarrow M[A] + M[B]$

ADD R2, C, D

$R2 \leftarrow M[C] + M[D]$

MUL ~~R1~~ X ~~R2~~, R1, R2

$M[X] \leftarrow R1 * R2$

Here R1, R2 are registers

$M[] \Rightarrow$ memory location

Two address instructions

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

MOV R1, A

ADD R1, B

MOV R2, C

ADD R2, D

MUL R1, R2

MOV X, R1

Description

$R1 \leftarrow M[A]$

$R1 \leftarrow R1 + M[B]$

$R2 \leftarrow M[C]$

$R2 \leftarrow R2 + M[D]$

$R1 \leftarrow R1 * R2$

$M[X] \leftarrow R1$

One address instruction:

LOAD A

ADD B

STORE T

LOAD C

ADD D

MUL T

STORE X

Description

$AC \leftarrow M[A]$

$AC \leftarrow AC + M[B]$

$M[T] \leftarrow AC$

$AC \leftarrow M[C]$

$AC \leftarrow AC + M[D]$

$AC \leftarrow AC * M[T]$

$M[X] \leftarrow AC$

Where AC ^{is} ~~are~~ Accumulator Register

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

$M[T] \leftarrow$ memory location

Zero address Instruction:

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

Postfix notation $X = AB + CD + *$

	Description
PUSH A	TOP = MSA 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	$R1 \leftarrow M[A] - M[B]$
MUL R2, D, E	$R2 \leftarrow M[D] * M[E]$
SUB R2, R2, F	$R2 \leftarrow R2 - M[F]$
MUL R2, R2, C	$R2 \leftarrow R2 * M[C]$
ADD R1, R1, R2	$R1 \leftarrow R1 + R2$
MUL R3, H, K	$R3 \leftarrow M[H] * M[K]$
ADD R3, R3, G	$R3 \leftarrow R3 + M[G]$
DIV X, R1, R3	$M[X] \leftarrow R1 / R3$

Two address instructions

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

One address instruction

		Description
LOAD	A	$AC \leftarrow M[A]$
SUB	B	$AC \leftarrow AC - M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	D	$AC \leftarrow M[D]$
MUL	E	$AC \leftarrow AC * M[E]$
SUB	F	$AC \leftarrow AC - M[F]$
MUL	C	$AC \leftarrow AC * M[C]$
ADD	T	$AC \leftarrow AC + M[T]$
STORE	T	$M[T] \leftarrow AC$
LOAD	H	$AC \leftarrow M[H]$
MUL	K	$AC \leftarrow AC * M[K]$
ADD	G	$AC \leftarrow AC + M[G]$

STORE TI $M[TI] \leftarrow AC$
 LOAD T $AC \leftarrow M[T]$
 DIV TI $AC \leftarrow AC / M[TI]$
 STORE X $M[X] \leftarrow AC$

Zero address instruction

Postfix Notation $AB - CDE * F - * + G HK * + /$

Description

PUSH A	Top $\Rightarrow A$
PUSH B	Top $\Rightarrow B$
SUB	Top $\Rightarrow (A - B)$
PUSH C	Top $\Rightarrow C$
PUSH D	Top $\Rightarrow D$
PUSH E	Top $\Rightarrow E$
MUL	Top $\Rightarrow (D * E)$
PUSH F	Top $\Rightarrow F$
SUB	Top $\Rightarrow ((D * E) - F)$
MUL	Top $\Rightarrow C * ((D * E) - F)$
ADD	Top $\Rightarrow (A - B) + (C * ((D * E) - F))$
PUSH G	Top $\Rightarrow G$
PUSH H	Top $\Rightarrow H$
PUSH K	Top $\Rightarrow K$
MUL	Top $\Rightarrow (H * K)$
ADD	Top $\Rightarrow G + (H * K)$
DIV	Top $\Rightarrow ((A - B) + C * ((D * E) - F) / (G + (H * K)))$
POP X	$X \leftarrow \text{Top}$

Tutorial sheet

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

Ans: Expression $\Rightarrow D = A + B * C$

Program using one address instruction

LOAD	B	$AC \leftarrow M[B]$
MUL	C	$AC \leftarrow AC * M[C]$
ADD	A	$AC \leftarrow AC + M[A]$
STORE	D	$M[D] \leftarrow AC$

Program using Two address instruction

MOV	RI, B	$RI \leftarrow M[B]$
MUL	RI, C	$RI \leftarrow RI * M[C]$
ADD	RI, A	$RI \leftarrow RI * M[A]$
STORE MOV	D, RI	$M[D] \leftarrow RI$

Program using Zero address instruction

~~ABC~~ $A B C * + =$

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

- How many bits are there in the operation code, the register code and address part?
- Draw the instruction format and indicate the number of bits in each part.
- How many bits are there in the data and address inputs/buses of memory

Ans

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

Size of the word $\Rightarrow 32$ bit

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

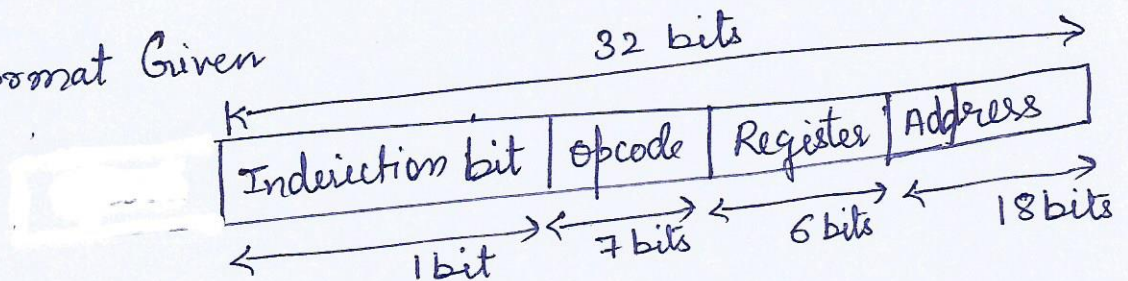
No. of Indirection bits $\Rightarrow 1$

Number of register = 64 $\Rightarrow 2^6$

bits to identify 64 registers = 6

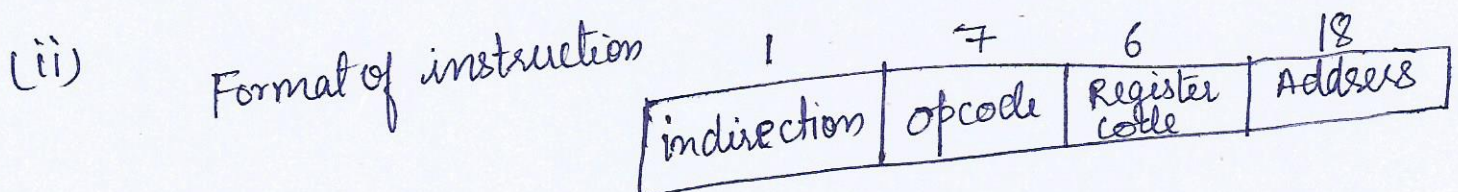
Number of bits in address = 18

instruction format Given



$$\begin{aligned}\text{Number of bits in opcode} &= 32 - (1 + 6 + 18) \\ &= 7 \text{ bits}\end{aligned}$$

- (i) Number of bits in opcode = 7
Number of bits in register code = 6
Number of bits in address part = 18



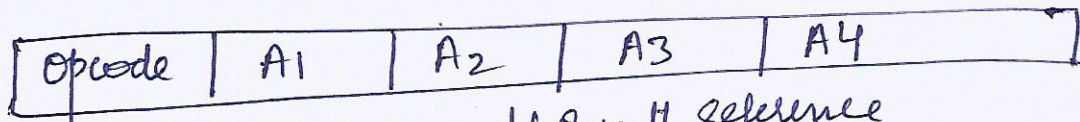
- (iii) Number of bits in the address input = 18
Number of bits in data input = 32.

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

which instruction has

- opcode
- Reference to operands
- ~~Reference of the next instruction~~

Ans: ~~Three~~ Four address instruction



A₁, A₂, A₃ ⇒ operand/Result reference

A₄ ⇒ Next instruction reference.

Number of possible instruction = 256

Number of bits in opcode = 8

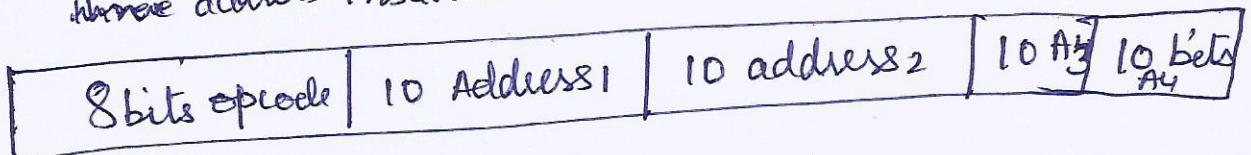
$$2^8 = 256.$$

Number of locations = 1024

Number of bits in address = 10

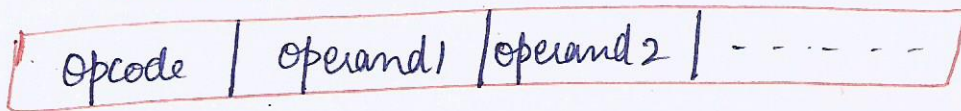
$$\{ \because 2^{10} = 1024 \}$$

~~Three~~ Four address instruction



Number of bits in instruction = 48

instruction



⇒ 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.

Ques: 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

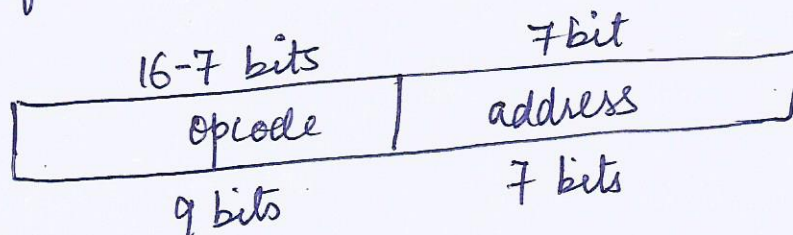
Number of bits in address to identify 128 location = n

$$2^n = 128$$

$$2^n = 2^7$$

$$\boxed{n = 7}$$

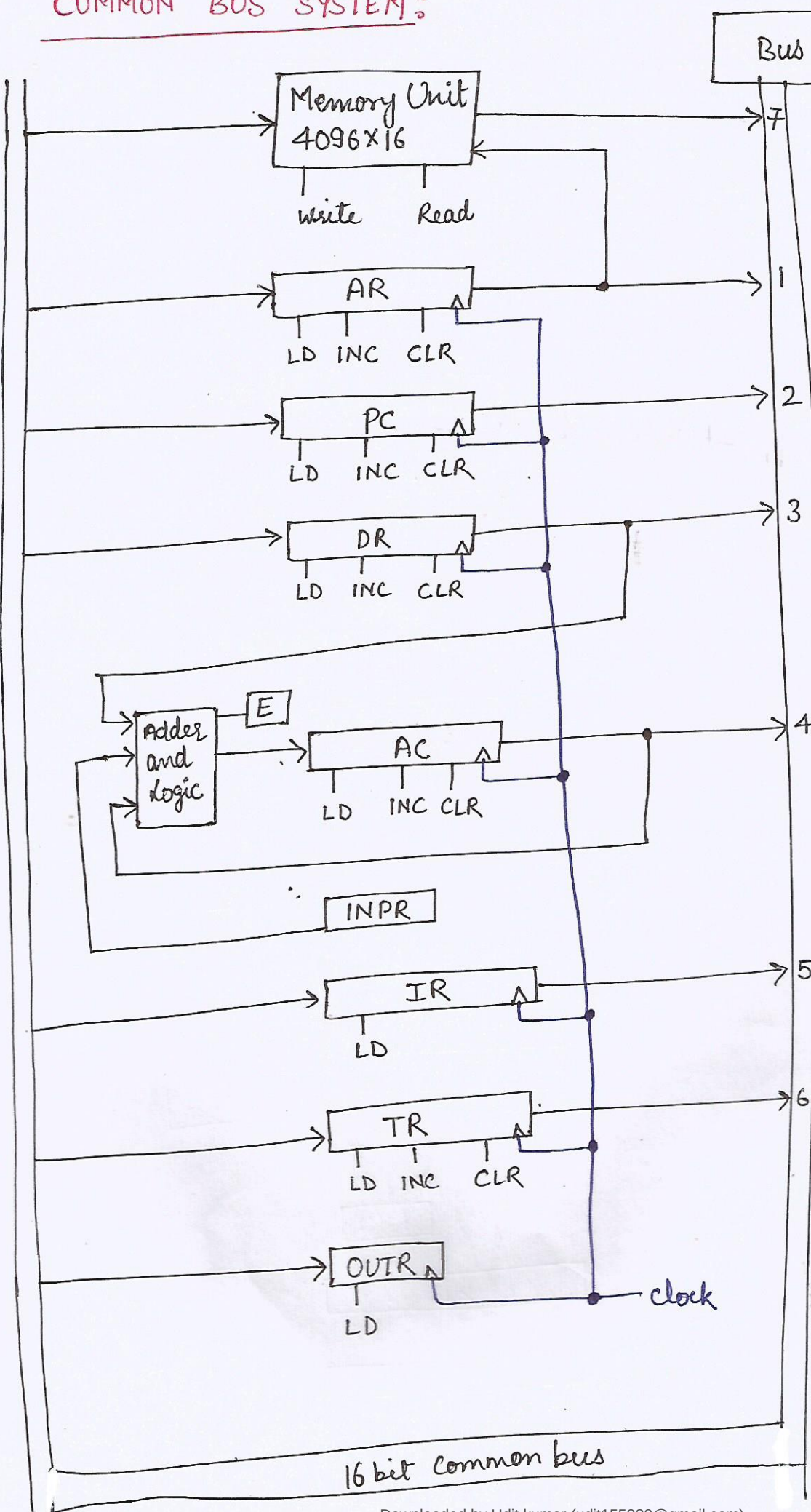
Number of bits in address field = 7



→ number bits in opcode = 9

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

COMMON BUS SYSTEM:



AR $\Rightarrow$ Address Register

PC $\Rightarrow$ Program Counter

DR $\Rightarrow$ Data Register

AC $\Rightarrow$ Accumulator

IR $\Rightarrow$ Instruction Register

TR $\Rightarrow$ Temporary Register

INPR $\Rightarrow$ Input Register

OTR $\Rightarrow$ Output Register

LD $\Rightarrow$ Load

INC $\Rightarrow$ Increment

CLR $\Rightarrow$ Clear

* Instruction cycle:

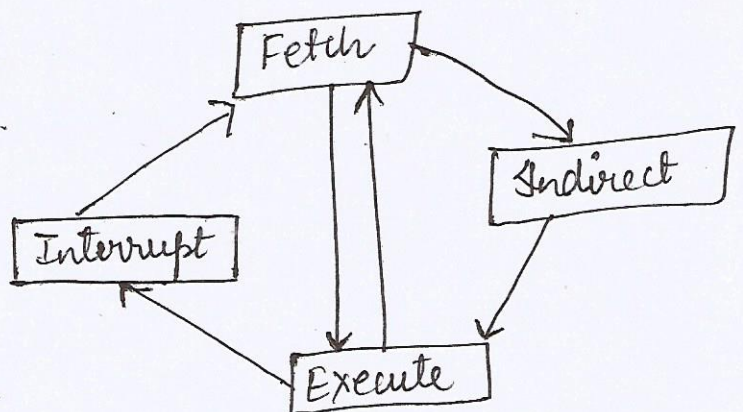
An instruction cycle consists of an instruction fetch, followed by zero or more operands fetches, followed by zero or more operand stores, followed by an interrupt check (if interrupts are enabled).

→ An instruction cycle includes the following stages/subcycles.

- Fetch
- Execute
- Interrupt

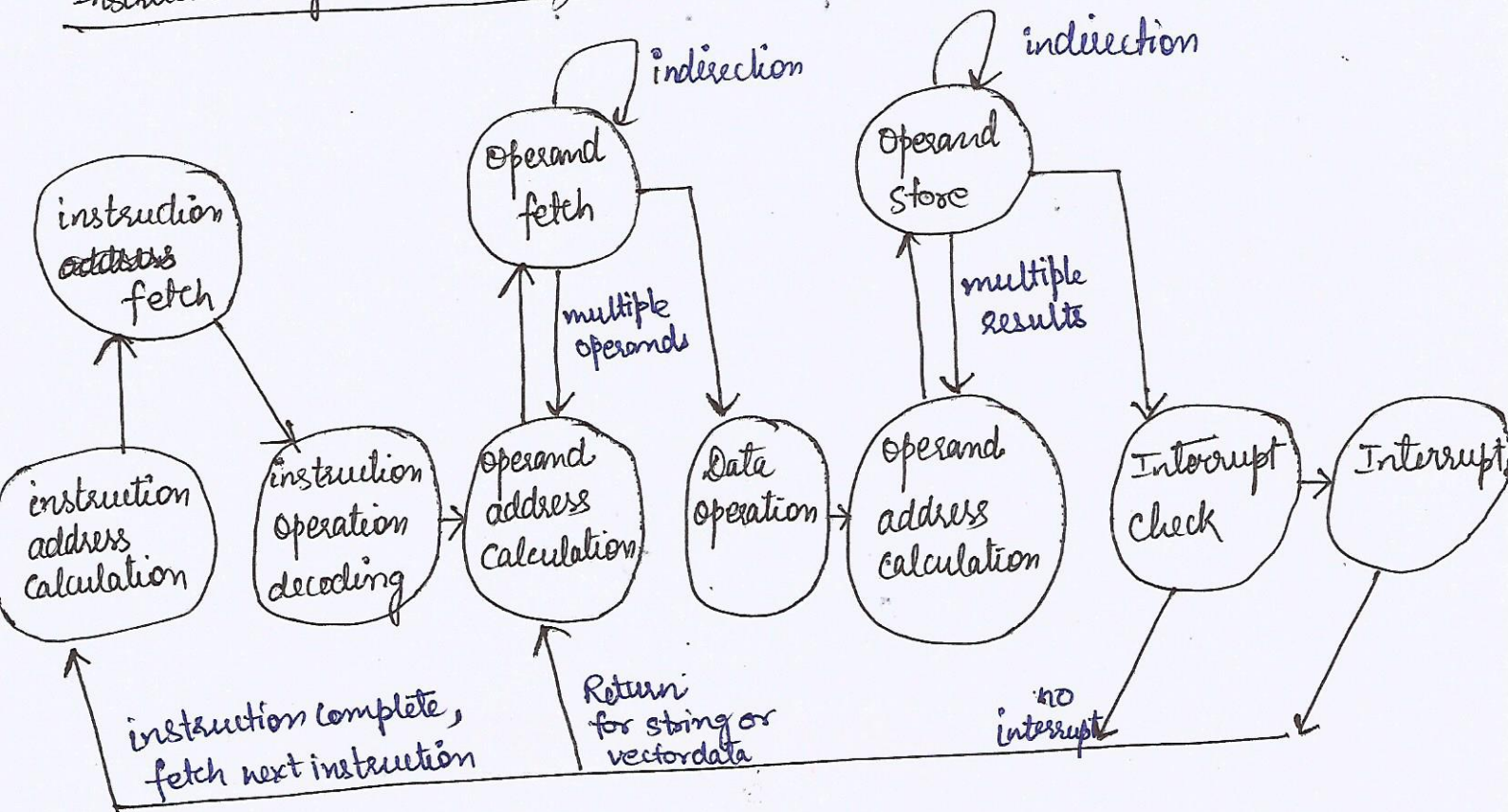
- ① Fetch: Read the next instruction from memory into processor.
- ② Execute: Interpret the opcode and perform the indicated operation.
- ③ Interrupt: If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.
- ④ Indirect (additional) only when indirect addressing is 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 raised before next instruction fetch.



Instruction cycle
(indirect addressing involved)

Instruction cycle state diagram:

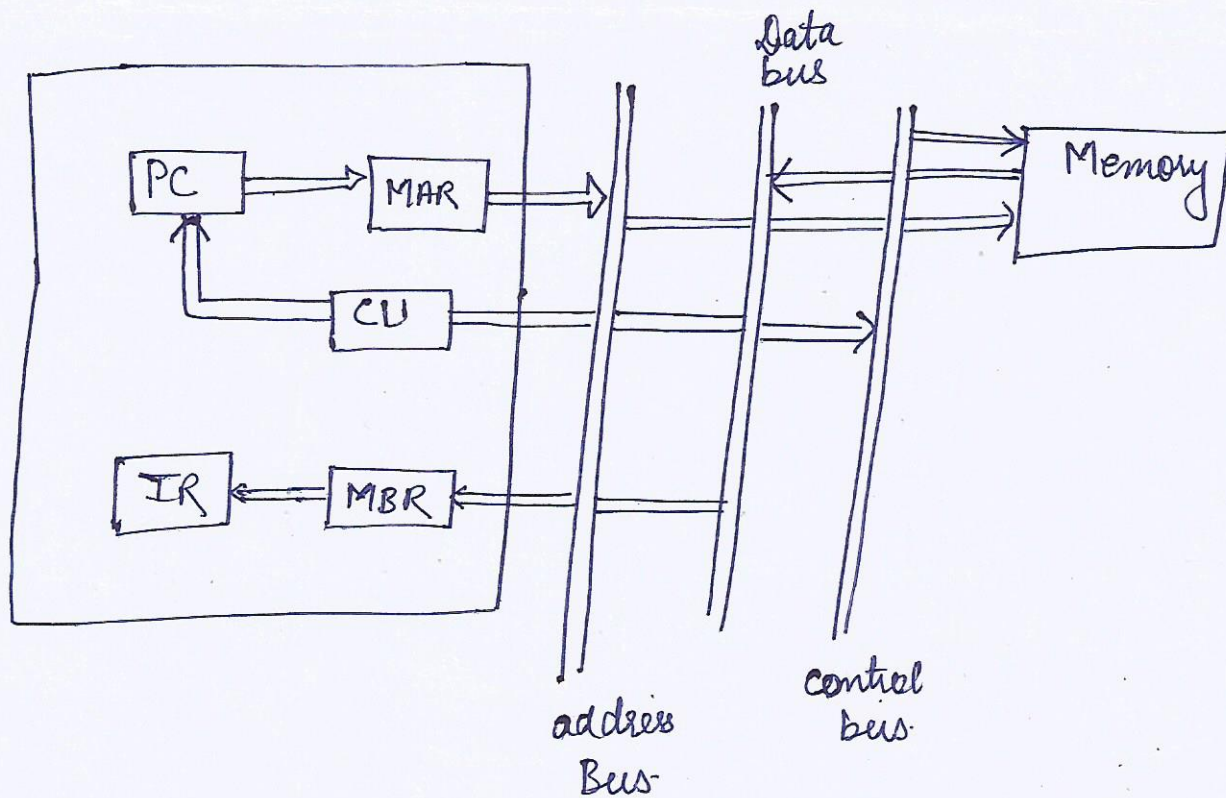


The exact sequence of events during an instruction cycle depends on the design of the processor.

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.

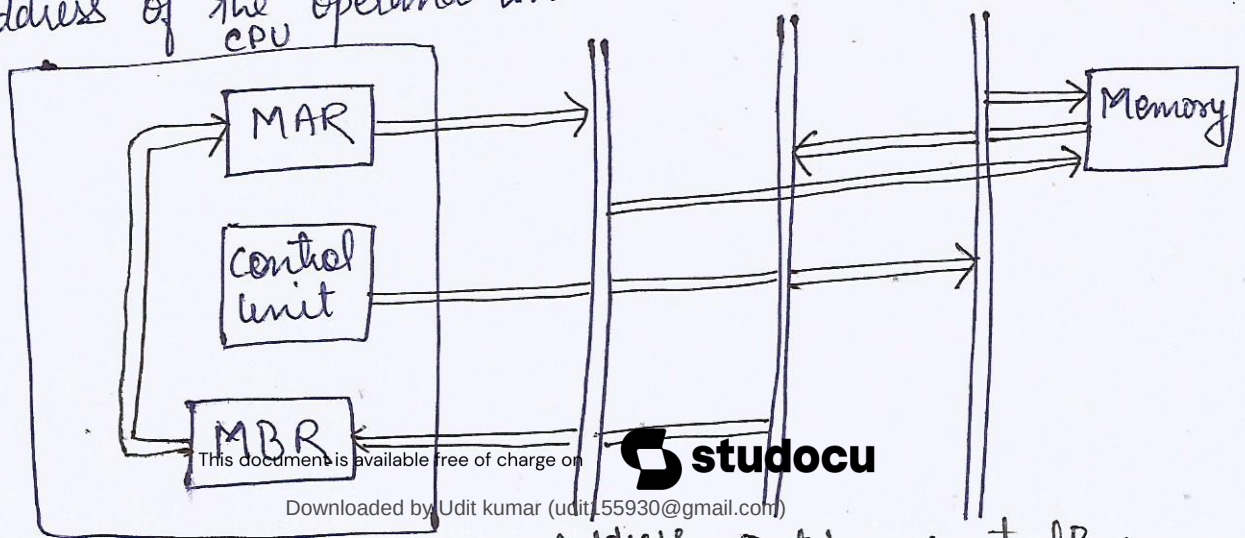


Data flow fetch cycle

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.



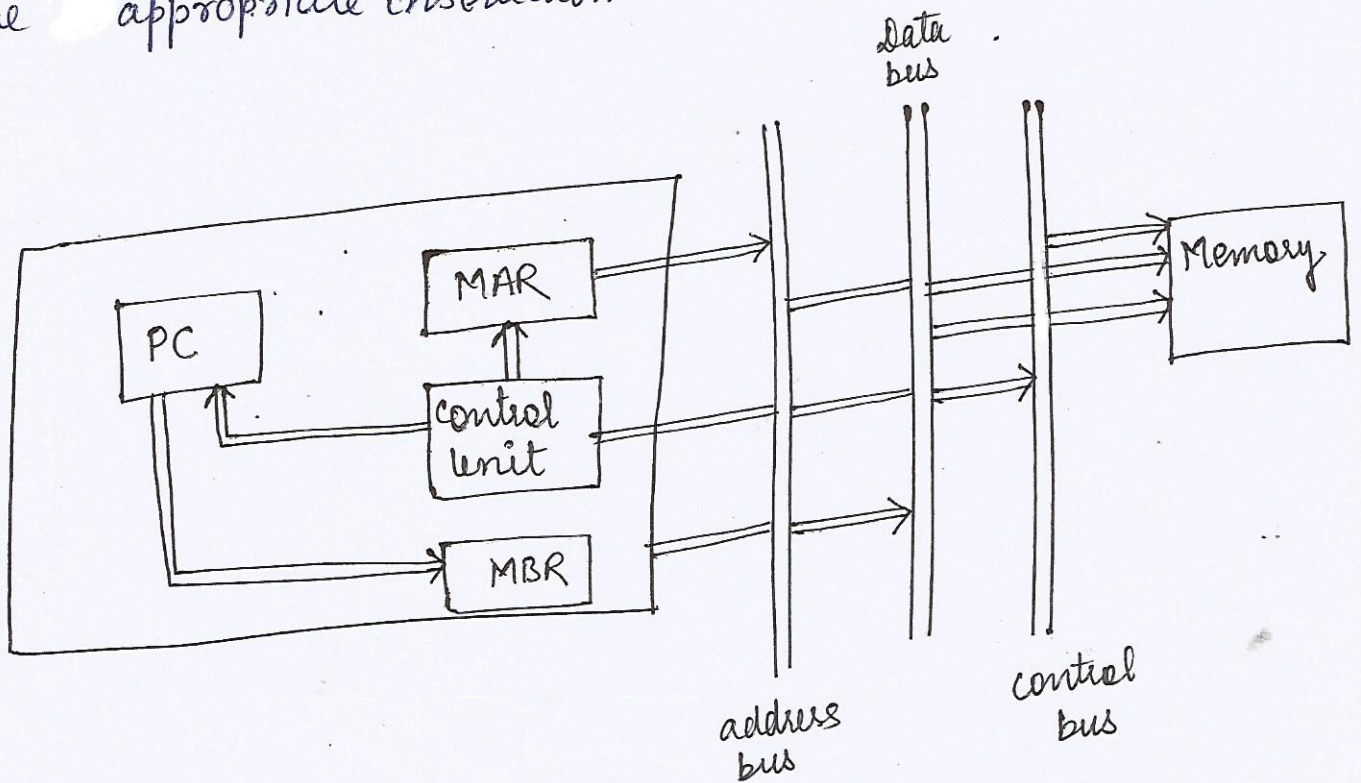
Data flow indirect cycle

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:

- A programme In basic computer, each instruction cycle consists of the following.

1- Fetch the instruction from memory

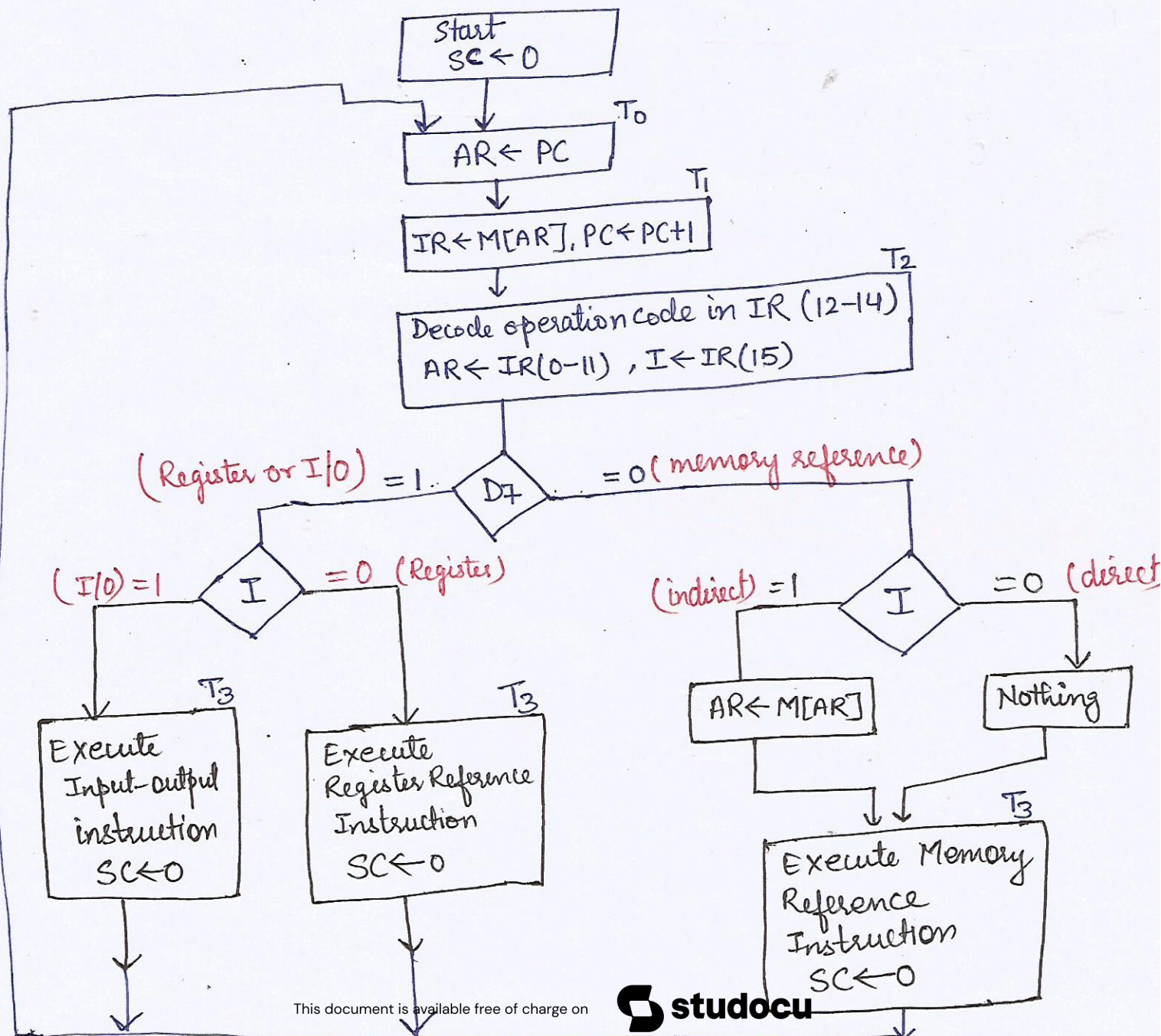
2- Decode the instruction

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:

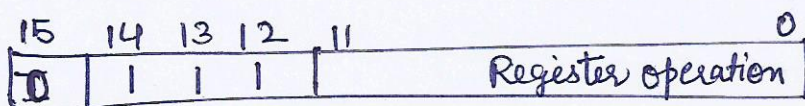
Memory Reference Instructions

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



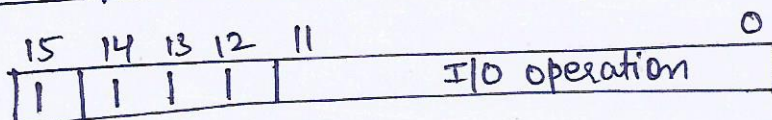
Register Reference Instruction

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 | $AC \leftarrow \text{shr } AC$ | circulate AC right |
| | | $AC(15) \leftarrow E$ | |
| | | $E \leftarrow AC(0)$ | |
- $E \Rightarrow \text{End carry}$

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

Memory Reference Instructions:

operation Decoder.

①	AND	Do	$AC \leftarrow AC \wedge M[AR]$ $Do T_4: DR \leftarrow M[AR]$ $Do T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$
②	ADD	D_1	$D_1 T_4: DR \leftarrow M[AR]$ $D_1 T_5: \begin{cases} AC \leftarrow AC + DR \\ E \leftarrow \text{Count}, SC \leftarrow 0 \end{cases}$
③	LDA	D_2	$D_2 T_4: DR \leftarrow M[AR]$ $D_2 T_5: AC \leftarrow DR, SC \leftarrow 0$

↑ load to AC

④ STA D_3 $D_3 T_4$: $M[AR] \leftarrow AC, SC \leftarrow 0$.
 $\uparrow$ store AC

⑤ BUN D_4 $D_4 T_4$: $PC \leftarrow AR, SC \leftarrow 0$
 Branch unconditionally

⑥ BSA D_5 $D_5 T_4$: $M[AR] \leftarrow PC, AR \leftarrow AR + 1$
 $\uparrow$ $D_5 T_5$: $PC \leftarrow AR, SC \leftarrow 0$
 Branch and Save Return address

⑦ ISZ D_6 $D_6 T_4$: $DR \leftarrow M[AR]$
 $D_6 T_5$: $DR \leftarrow DR + 1$
 $D_6 T_6$: $M[AR] \leftarrow DR,$
 Increment if and skip if zero
 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:

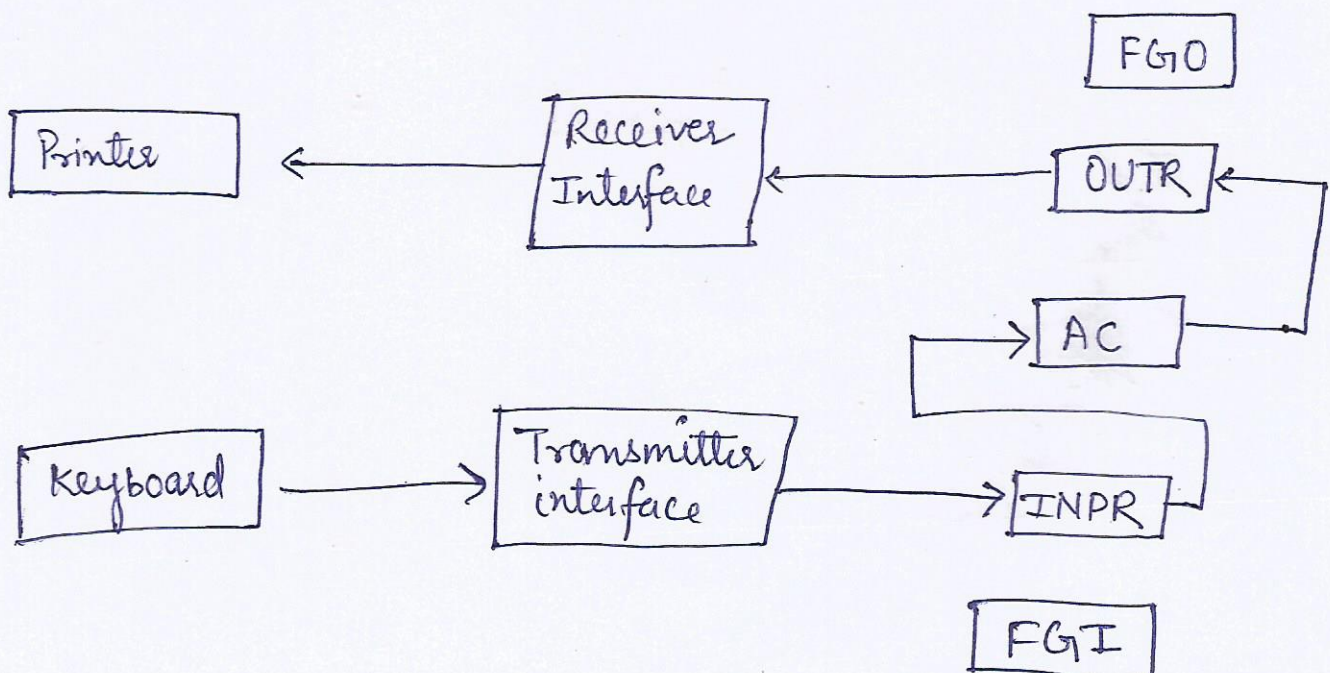
INP	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	input character
OUT	$OUTR \leftarrow AC(0-7) \quad FGO \leftarrow 0$	output character
SKI	if $(FGI = 1)$ then $PC \leftarrow PC + 1$	skip on input flag
SKO	If $(FGO = 1)$ then $PC \leftarrow PC + 1$	skip on output flag
IDN	$IEN \leftarrow 1$	interrupt enable ON
IOF	$IEN \leftarrow 0$	interrupt enable off

input-output configuration

input-output Terminal

Serial communication interface

computer Register and flip flops.



INPR $\Rightarrow$ 8 bit input Register

AC $\Rightarrow$ Accumulator

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

OUTR $\Rightarrow$ output Register (8 bits)

FGO $\Rightarrow$ is 1 bit control flip flop. If FGO is 1 then information ~~from~~ is transferred from AC 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 microoperations 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 and 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 ~~two~~ parts

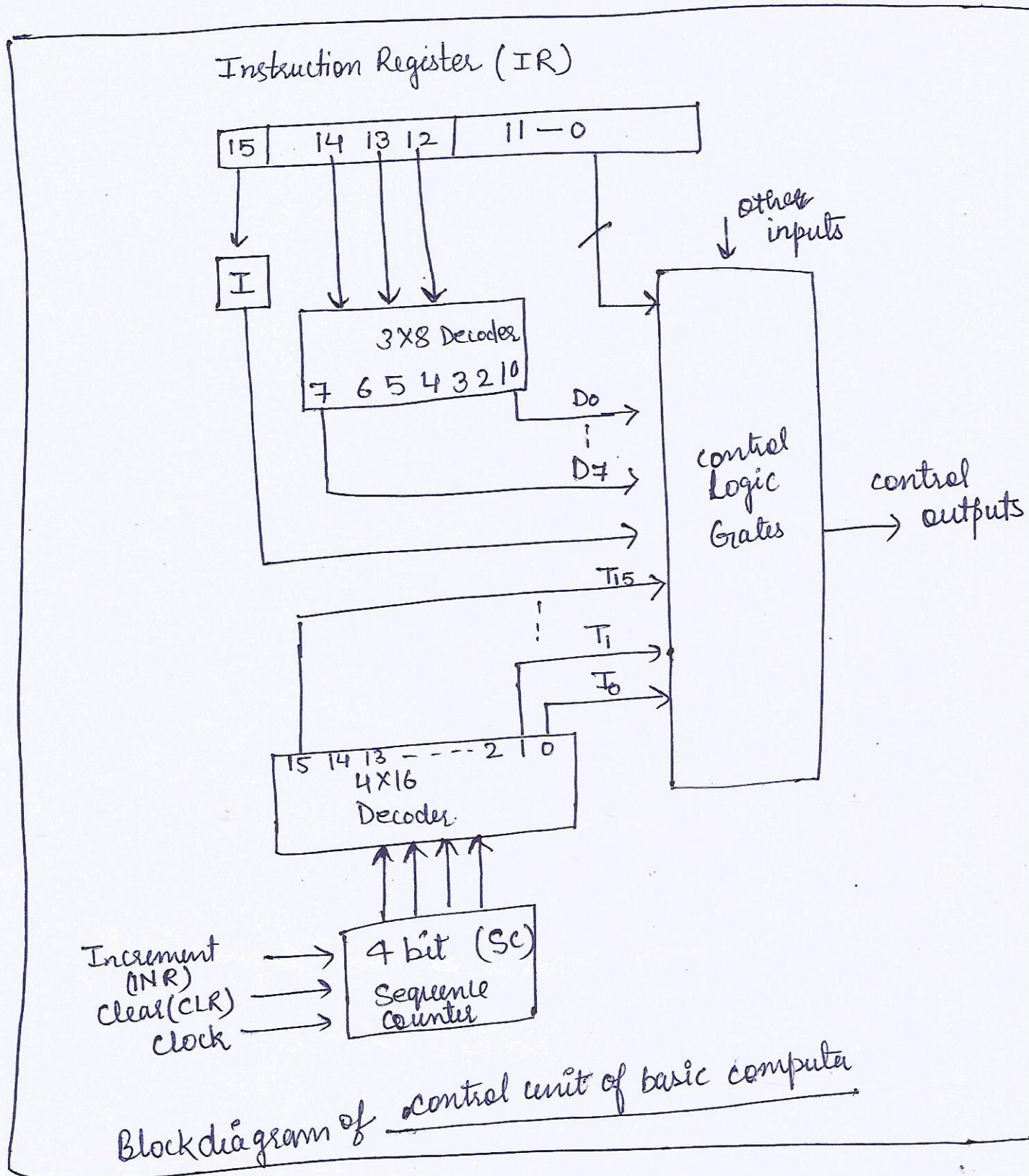
→ operation code: operation code part of an instruction is a group of bits that define operations such as add, subtract, shift, complement or increment etc.

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

$2^n \Rightarrow$ 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.



→ components of control unit are

① → Two decoders

② → One sequence counter

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)
 - ~~the~~ bit 0 to 11
- The operation code bits are decoded by a 3x8 decoder as outputs D₀ to D₇.
- 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 ~~to~~ 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 ~~to be~~ T₀, T₁, T₂, T₃ and T₄ in sequence. At the time of T₄, SC is cleared to 0 if decoder output D₃ is active. This is expressed symbolically.

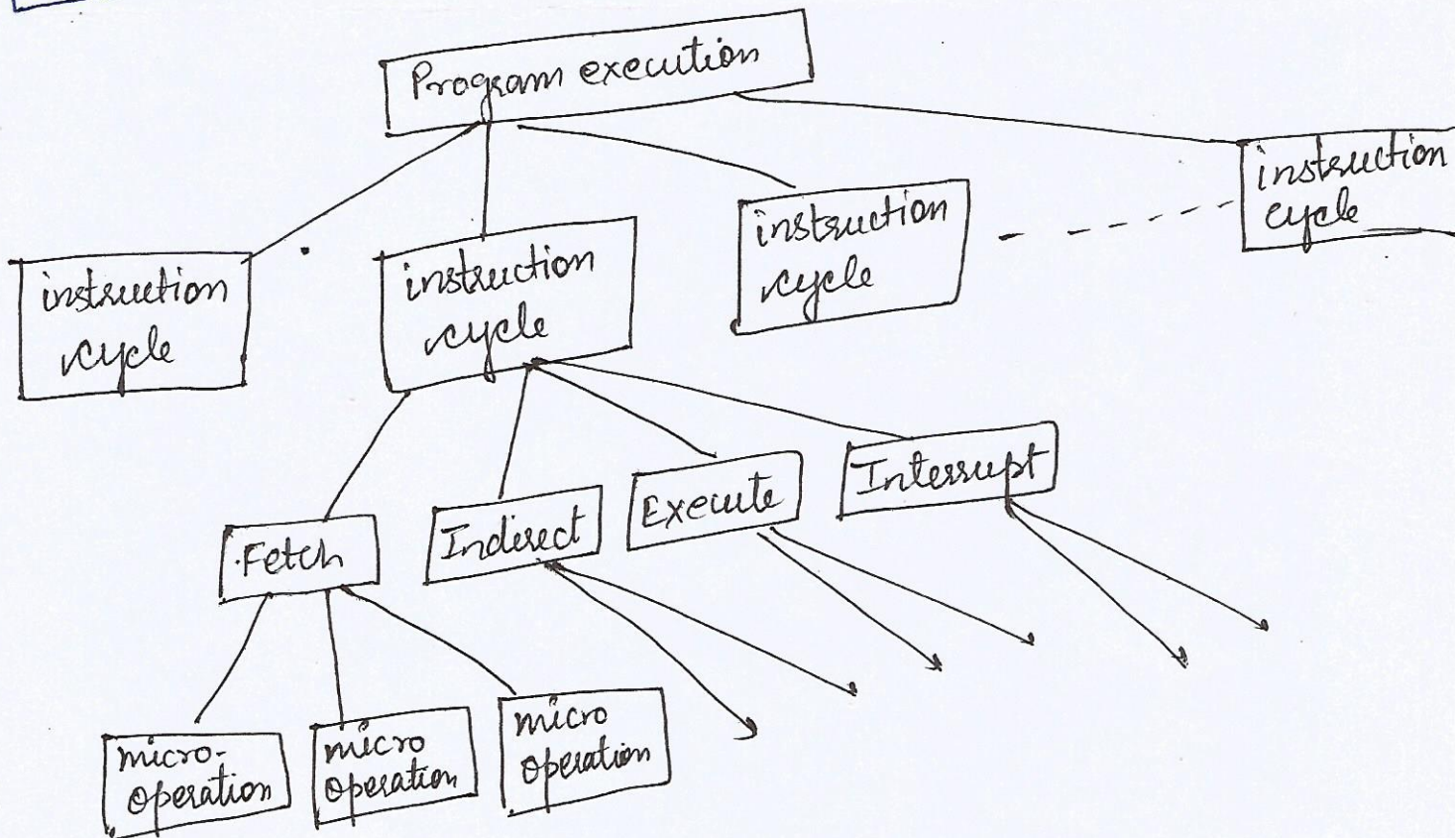
$$D_3 T_4 : SC \leftarrow 0$$

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 microoperations in internal computer registers.

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

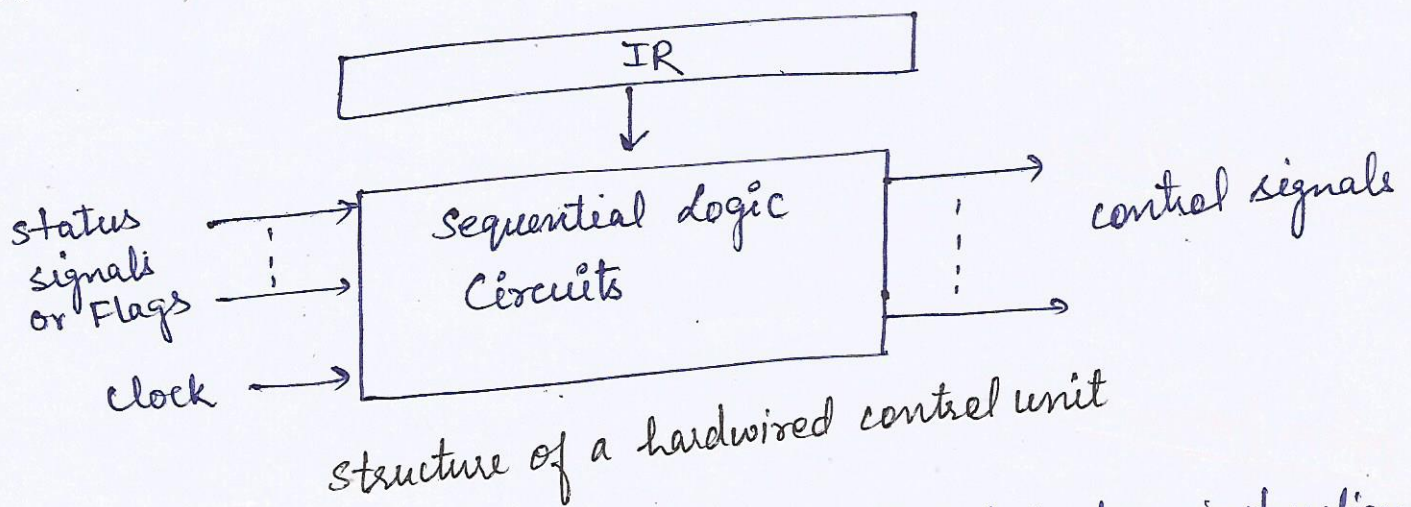
Hardwired and Microprogrammed control:

→ 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.



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

Microprogrammed control

A microprogrammed 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 controlword.

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 ~~mem~~ memory contains a microinstruction. A microinstruction specifies one or more microoperations for the system.

Micro program:

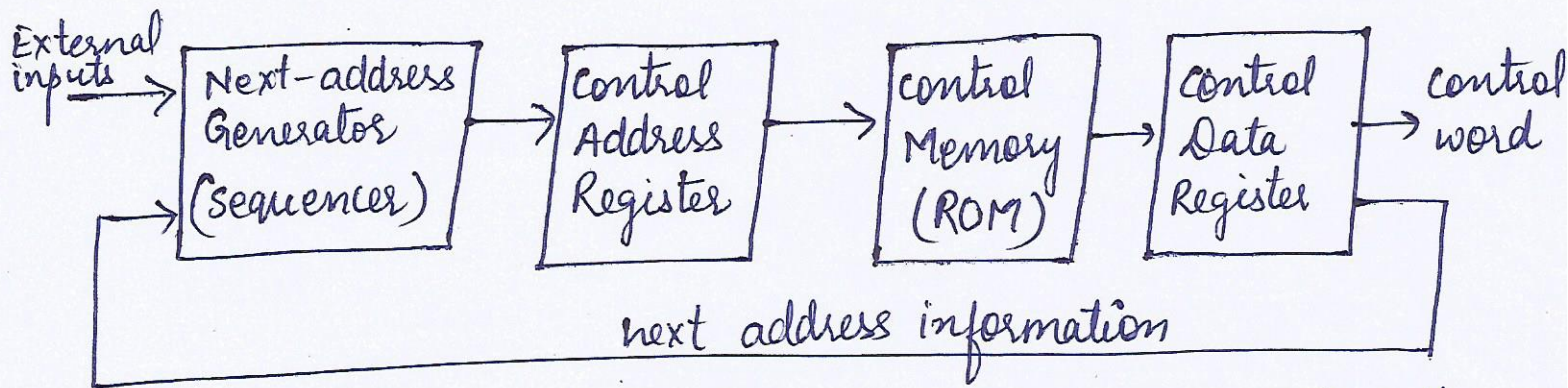
A sequence of microinstructions constitutes a ~~microop~~ microprogram.

Microcode:

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

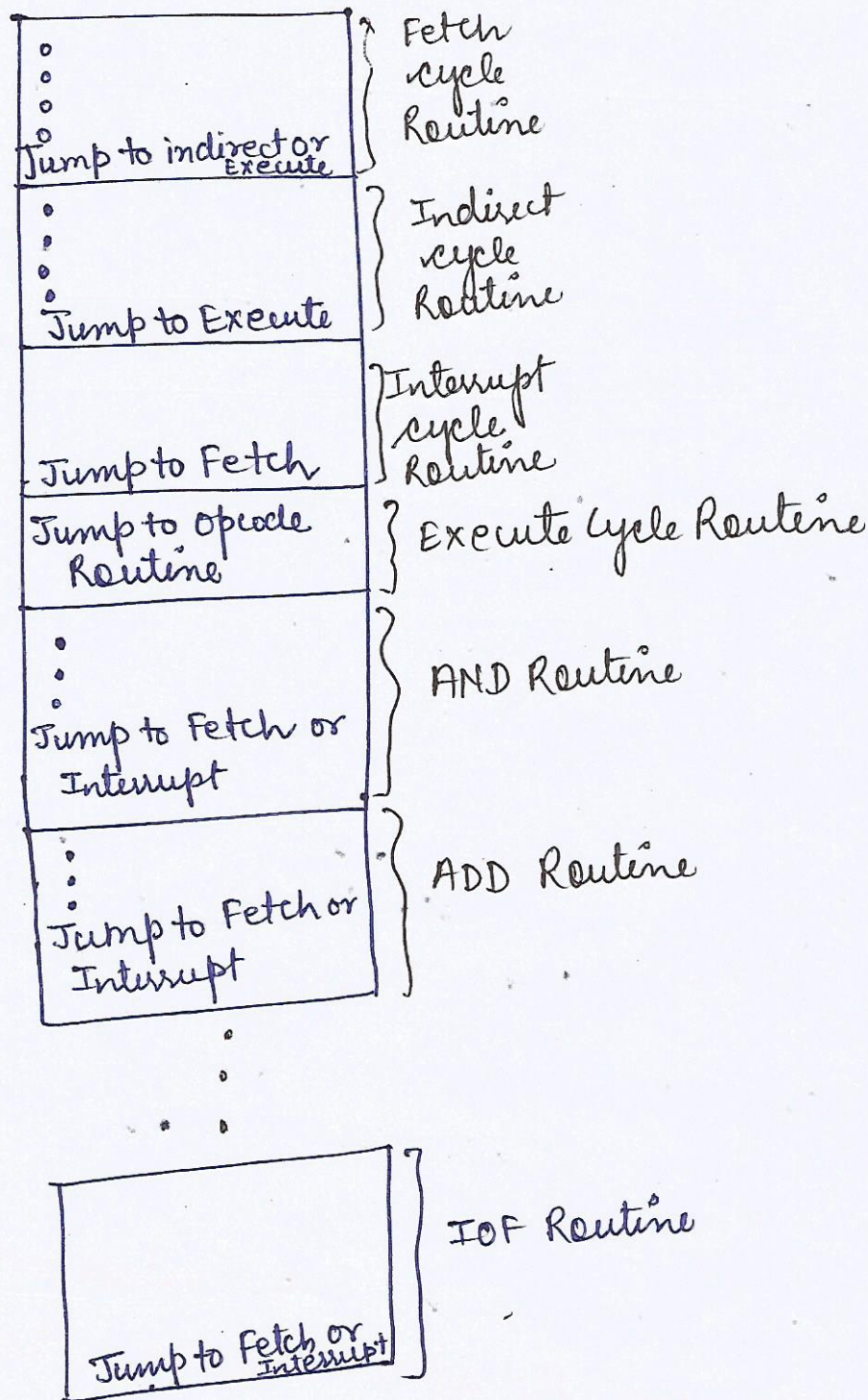
A microinstruction can cause execution of one or more micro operation and a sequence of microinstruction (micro program) can cause the execution of an instruction.

Microprogrammed 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 microoperations for the processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in 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.

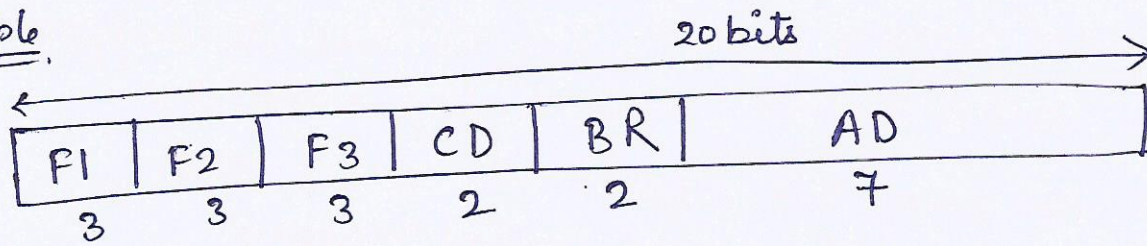
Organization of Control Memory:



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.

Microinstruction Format :

Example



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

→ CD ⇒ Field selects status bit conditions

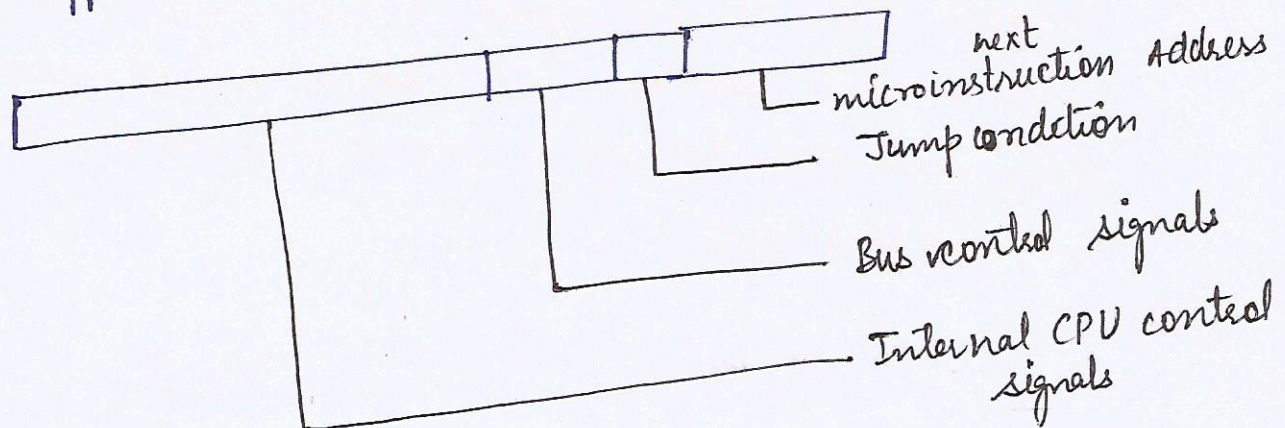
→ BR ⇒ field specifies the type of branch to be used

→ AD ⇒ This field contains a branch address. ~~The~~ The size of this field depends on the control memory size.

Here AD has 7 bits that means ⇒ 2^7 words in control memory.

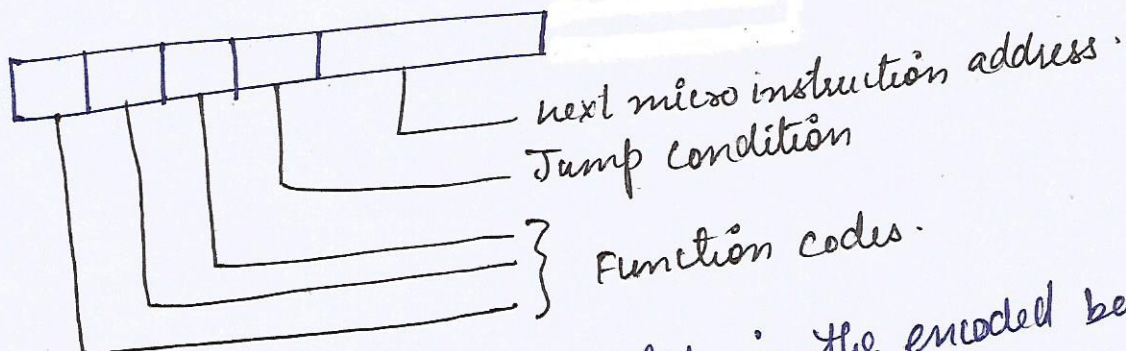
Horizontal Microprogrammed Control unit:

- 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 $\Rightarrow 10$ because words = $1024 = 2^{10}$
 size of microinstruction

control signal bits	Address bits
64	10

 $\Rightarrow 74$ bits

vertical: control signals = 64 $\Rightarrow 2^6$
 Number of bits for control signals = 6.
 Address bits $\Rightarrow 10$ because $1024 \Rightarrow 2^{10}$.
 size of vertical micro instruction $\Rightarrow$

control signal bits	Address bits
6	10

$\Rightarrow 16$ 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.

Control signals	Condition flags	Address
bits	bits	bits

Horizontal microprogramming:

Size of control word or microinstruction $\Rightarrow$

50	4	10
----	---	----

 $\Rightarrow 64$

Number of bits for control signals $\Rightarrow 50$ (bit per control signal)

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

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

The size of each control word $\Rightarrow 64$ bit.

Control memory has words $\Rightarrow 1024$

Size of control memory = 1024×64 bits

$\Rightarrow 1 \text{ KB} \times 8 \text{ B}$

$\Rightarrow 8 \text{ KB}$

Vertical Microprogramming:

Control signal bits $\Rightarrow 6$ because

Control signals $\Rightarrow 50$

flag bits $\Rightarrow 4$

Address bits $\Rightarrow 10$

Size of control word $\Rightarrow 20$ bits $\Rightarrow (6+4+10)$

Size of control memory $\Rightarrow 20 \times 1024$

→ 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.

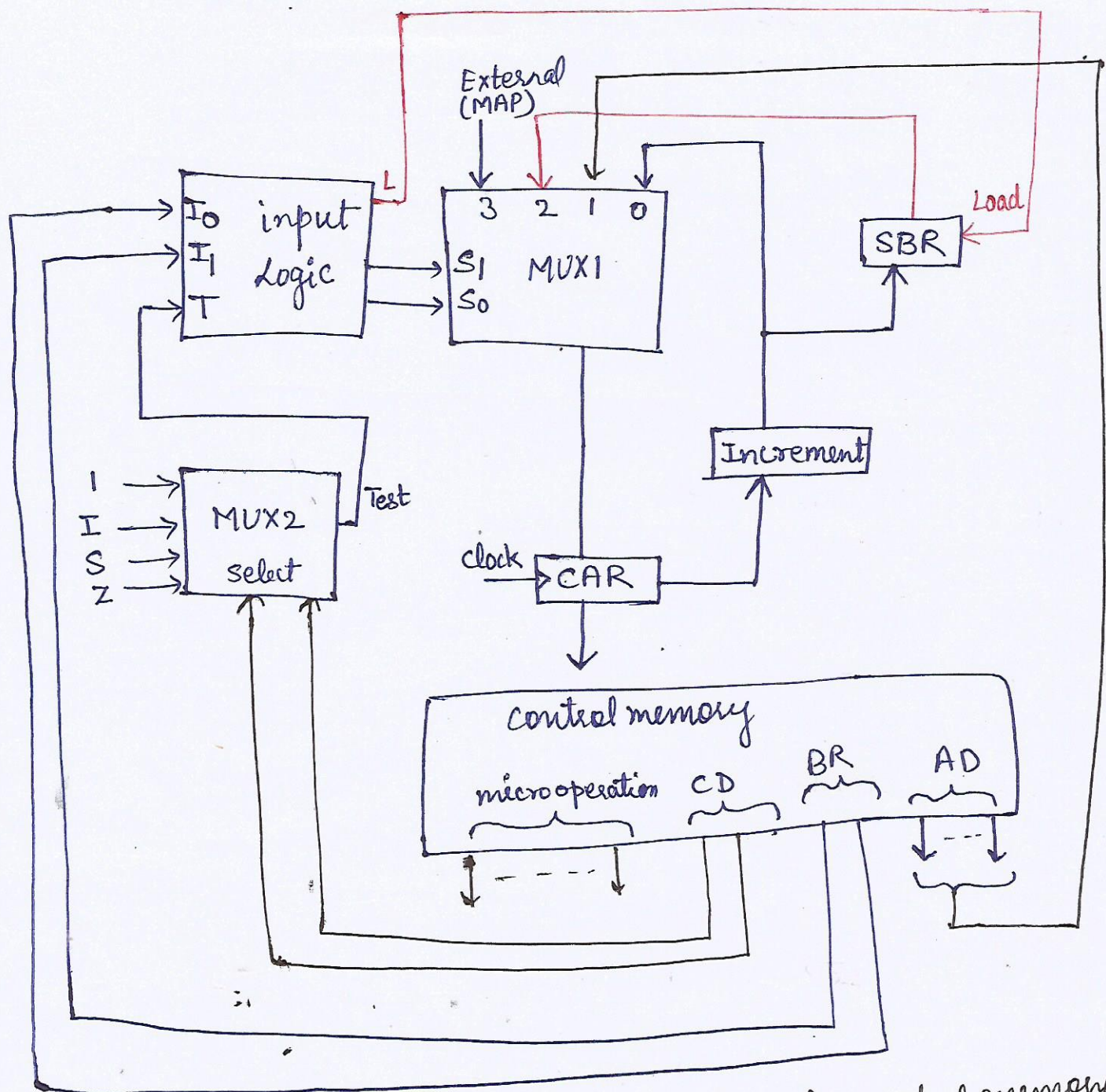
→ steps that the control must undergo during the executing of a single computer instruction.

- 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 microoperations 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.



microprogram sequences 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 $\Rightarrow$ 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	Sign bit
11		Z	Zero value in AC

BR $\Rightarrow$ Branch type.

00	JMP
01	CALL
10	RET
11	MAP

Input logic design $\Rightarrow$

$$S_1 = I_1$$

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

$$L = \overline{I_1} I_0 T$$

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)

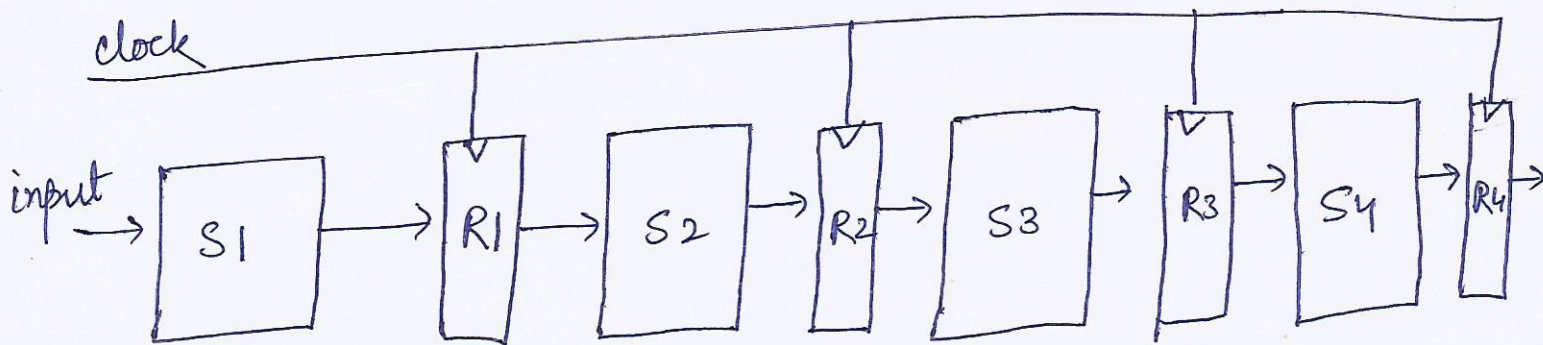
The essential goal of CISC architecture is to provide a single machine instruction for each statement that is written in high level language.

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:

- Pipelining 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 register 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 and etc.

Instruction pipeline:

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

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \dots, 7.$$

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

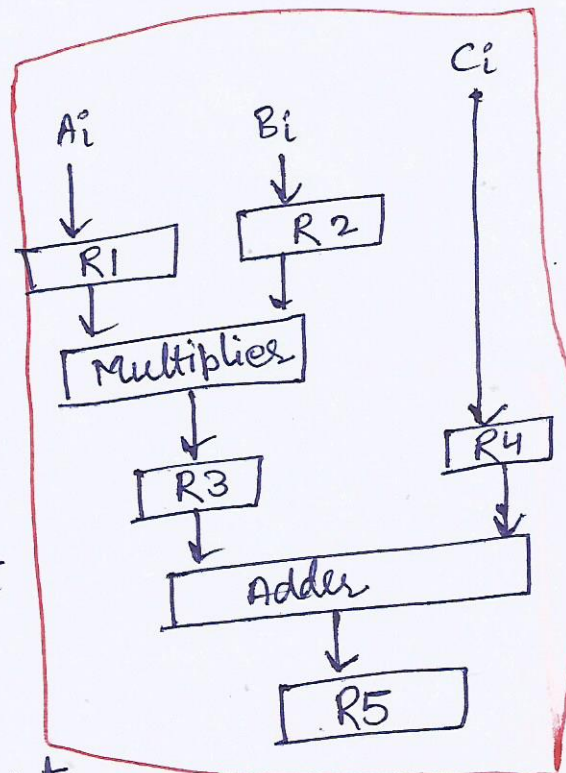
• suboperations

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

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

$$R5 \leftarrow R3 + R4$$

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
	⋮				
					soon -

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.

	1	2	3	4	5	6	7	8	9	→ clock cycle
Segment 1	T_1	T_2	T_3	T_4	T_5	T_6				
Segment 2		T_1	T_2	T_3	T_4	T_5	T_6			
Segment 3			T_1	T_2	T_3	T_4	T_5	T_6		
Segment 4				T_1	T_2	T_3	T_4	T_5	T_6	

Assumed Number of task = 6

Instruction Pipeline

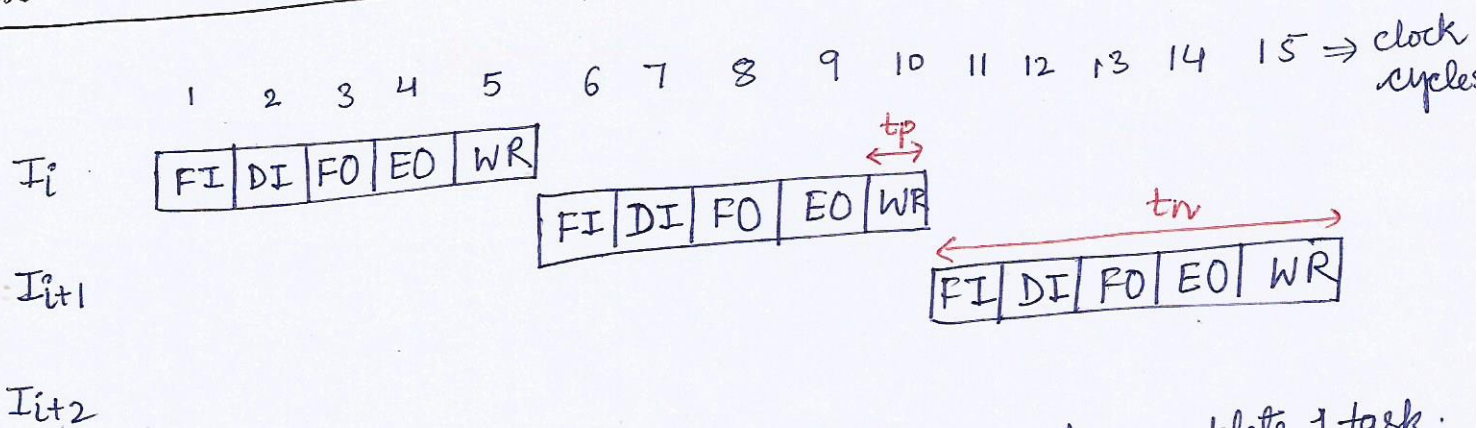
⇒ The phases of a computer instruction cycle overlap in the execution.

Instruction cycle ⇒ 5 phases.

~~FO~~ 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 ⇒ 3 × cycle to complete 1 task.
⇒ 3×5
= ~~3~~ 15

Time to execute 3 tasks ⇒ $3 \times t_n$
= 3 × time to execute or complete one task

Instruction execution in pipeline architecture

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

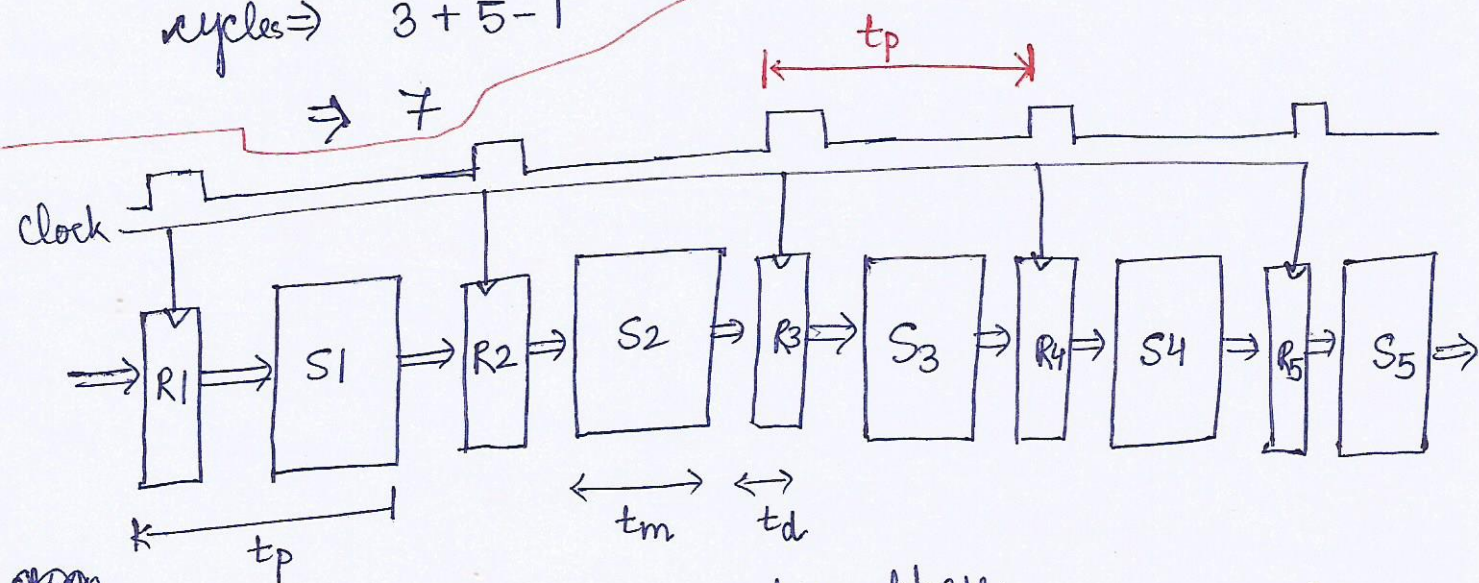
Number of cycle required to complete n tasks in k -stage.
 pipeline = $k + n - 1$

$n = 3$ (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.

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

$$t_p = \text{max time taken by any segment} + \text{Register delay}$$

5-Segment pipeline

Formulas :

considers k -segment pipeline $\Rightarrow$ means Number of segments $\Rightarrow k$ (or phases).

Number of tasks to be executed $= n$

clock cycle time $= t_p$ ~~or time~~

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

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

$$\begin{aligned}\text{The time taken by } n \text{ tasks using } k\text{-segment pipeline} \\ &= k t_p + (n-1) t_p \\ &= [k + (n-1)] \cdot t_p\end{aligned}$$

$$\begin{aligned}\text{Number of cycle by } n \text{ tasks using } k\text{-segment pipeline} \\ \Rightarrow k + (n-1)\end{aligned}$$

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. $\Rightarrow n t_n$

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

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 $\Rightarrow$

$$S = \frac{t_n}{t_p}$$

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

$$t_n = K t_p \Rightarrow$$

Including this assumption the speedup reduces to

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

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

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

$$\text{Throughput } H_k = \frac{n}{(K+n-1) t_p} \quad \text{or} \quad \frac{n f}{(K+n-1)}$$

Throughput : The number of tasks performed per time unit

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

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 $\Rightarrow$ pipeline stage or processing element
Column $\Rightarrow$ 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.

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

	1	2	3	4	5	6		
S_1	X					X	$\Rightarrow 1, 6$	$6-1 \Rightarrow 5$
S_2		X		X			$\Rightarrow 2, 4$	$4-2 \Rightarrow 2$
S_3			X		X		$\Rightarrow 3, 5$	$5-3 = 2$

Forbidden latencies = $\{2, 5\}$

Permissible latencies $\Rightarrow \{1, 3, 4\}$

- 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
 $m = \text{maximum forbidden latency}$

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

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

$C_i = 0$ otherwise

Previous Example

Forbidden latencies = $\{2, 5\}$

$$C = C_5 C_4 C_3 C_2 C_1$$

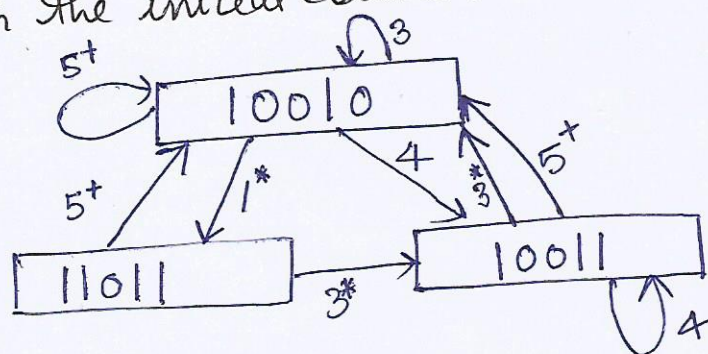
$$1 \ 0 \ 0 \ 1 \ 0$$

state diagram:

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

$\Rightarrow$ collision vector forms the initial state

$\Rightarrow$ 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.

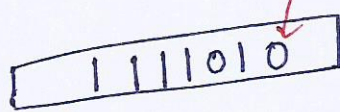
Latency Cycle: A latency cycle is a latency sequence which repeats the same subsequence (cycle) indefinitely.

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

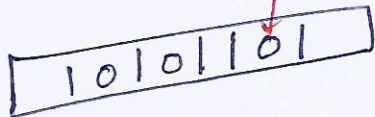
Simple cycle: A simple cycle is a latency cycle in which each state appear only once.

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.

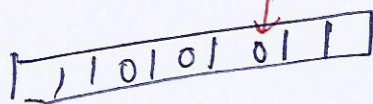
Suppose
state



minimum latency (permissible) = 1



min latency $\Rightarrow$ 2.



min latency = 3

Note

Minimum latency respect 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 $\Rightarrow$ forms initial state.
it has permissible latency $\Rightarrow 1, 3, 4$

initial state 10010

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

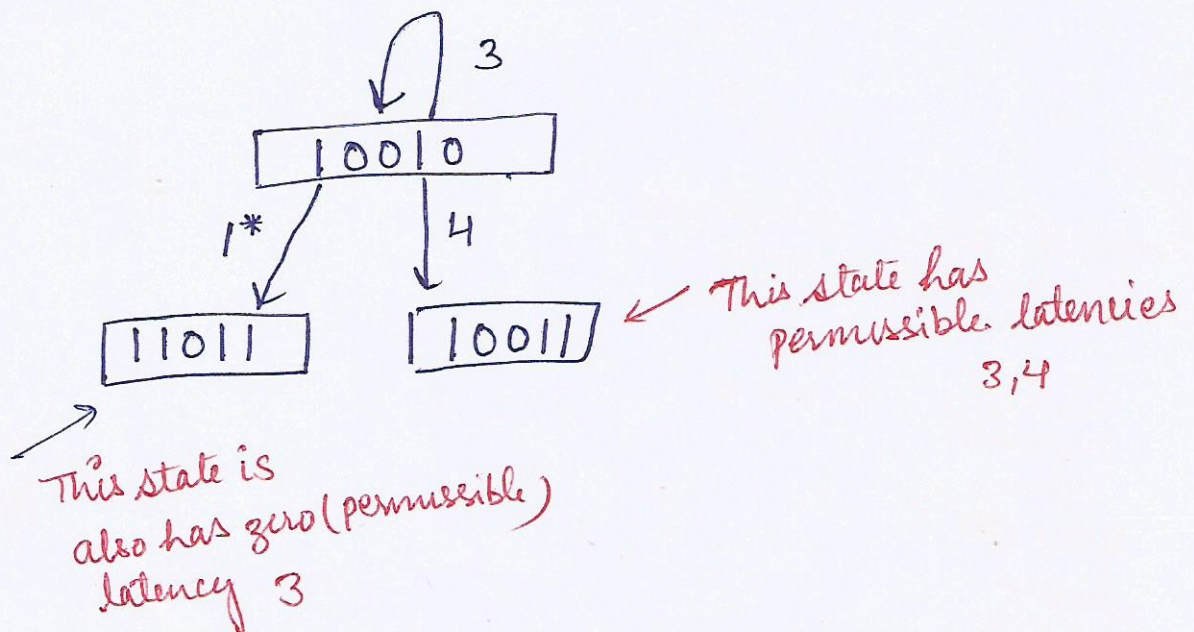
case 1 $\Rightarrow$

$$\begin{array}{r} 10010 \\ \downarrow \text{1 shift Right} \\ 01001 \\ \underline{10010} \text{ ORing} \\ \underline{11011} \text{ next state} \end{array}$$

case 3 $\Rightarrow$

$$\begin{array}{r} 10010 \text{ — CV} \\ 00010 \text{ — 3 shift Right} \\ \underline{10010} \text{ — CV (ORing)} \\ \underline{10010} \text{ — next state} \end{array}$$

case 4 $\Rightarrow$

$$\begin{array}{r} 10010 \\ 00001 \text{ — 4 shift Right} \\ \underline{10010} \\ \underline{10011} \text{ — next state} \end{array}$$


State $\Rightarrow$ 11011

Case 3 shift of 3.

$$\begin{array}{r} 00011 \Rightarrow \text{shift of 3.} \\ 10010 \Rightarrow \text{CV} \\ \hline 10011 \Rightarrow \text{ORing} \Rightarrow \text{next state (already present state)} \end{array}$$

State : 10011

~~Shift~~ Permissible latency $\Rightarrow$ 3, 4.

Case 3:- shift of 3

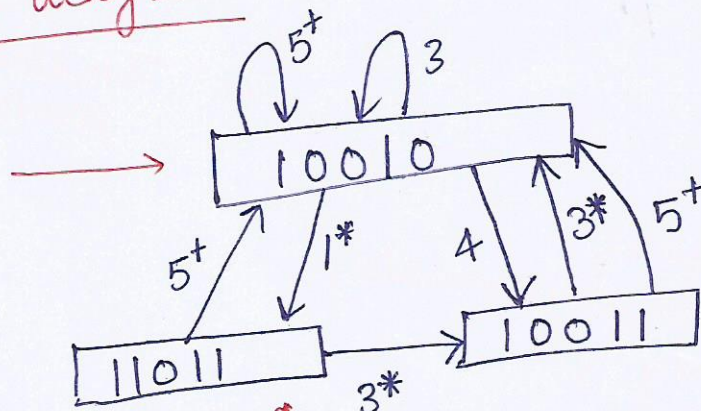
$$\begin{array}{r} 10011 \rightarrow 3 \text{ shifts Right} \\ 00010 \\ \hline 10010 \quad \text{ORing} \\ \hline 10010 \end{array}$$

Case 4: shift of 4.

$$\begin{array}{r} 10011 \rightarrow 4 \text{ shifts Right} \\ 00001 \\ \hline 10010 \quad \text{ORing} \\ \hline 10011 \end{array}$$

complete state diagram

minimum permissible latency of this state = 1
So marked with star.



min permissible latency = 3

min permissible latency $\Rightarrow$ 3

Simple cycle:

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

Constant cycle:

(3), (5)

Greedy cycles:- Examples $\Rightarrow$ (1,5), (3,4)

MAL Minimum Average Latency

• Simple cycles.

Average latency

3
5
(1,5)

$$\frac{3+5}{2} = 3$$

$$\text{So } MAL = 3$$

(1,3,5)

$$\frac{1+3+5}{3} = 3$$

(4,3)

$$\frac{4+3}{2} = 3.5$$

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

Throughput = if minimum constant cycle used.
minimum constant cycle = 3.

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

Reservation Table for Function X:

	1	2	3	4	5	6	7	8
S ₁	X					X		X
S ₂		X		X				
S ₃			X		X		X	

Question

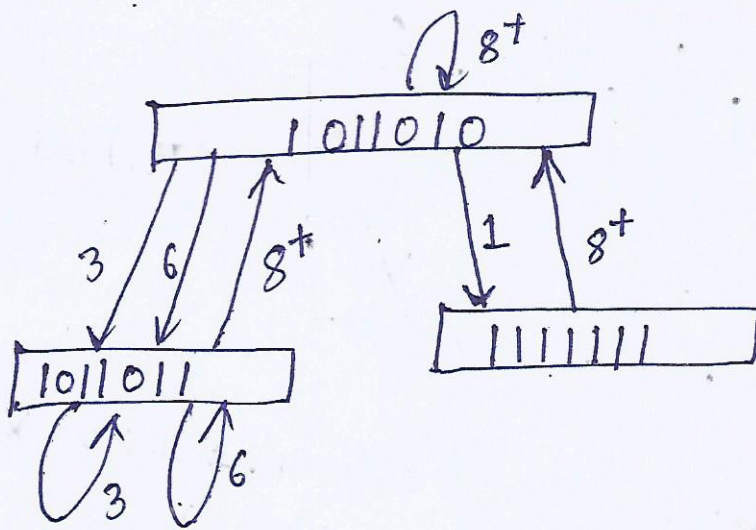
Forbidden latencies
 $\Rightarrow 2, 4, 5, 7$

Reservation Table for Function Y:

	1	2	3	4	5	6
S ₁	Y				Y	
S ₂			Y			
S ₃		Y		Y		Y

Forbidden latencies
 $\Rightarrow 2, 4$

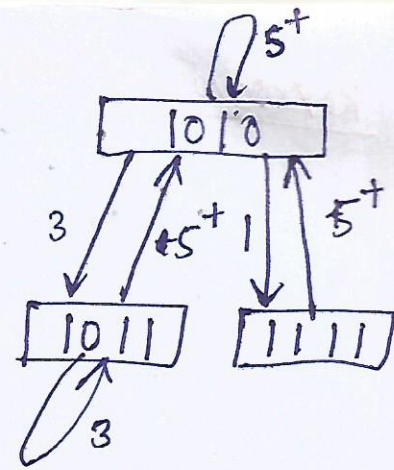
- $\Rightarrow$ If reservation table has n columns, the maximum forbidden latency is $m \leq (n-1)$
- $\Rightarrow$ permissible latency p will satisfy $1 \leq p \leq m-1$



possible Cycles $\Rightarrow$ (1,8)
 (3)
 (6)

$$MAL = 3.$$

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



(1,5) $\Rightarrow$ 3.0
 3 3.0'
 (3,5) $\Rightarrow$ 4.0
 MAL $\Rightarrow$ 3.