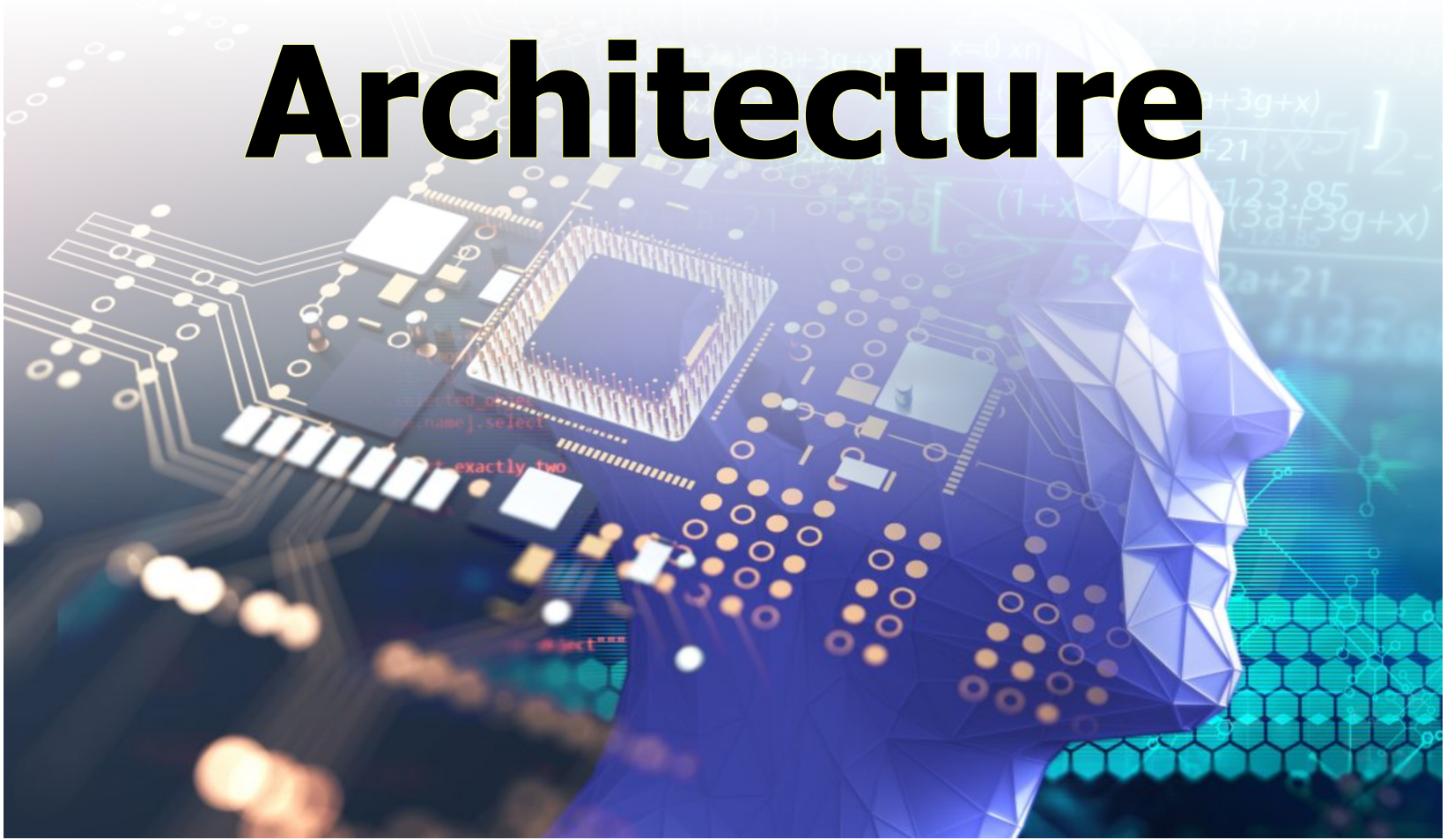
Computer Organization & Architecture



Published By:

Physics Wallah

ISBN: 978-93-94342-39-2

Mobile App: Physics Wallah (Available on Play Store)

Website: www.pw.live

Email: support@pw.live

Rights

Disclaimer

A team of PW experts and faculties with an understanding of the subject has worked hard for the books.

While the author and publisher have used their best efforts in preparing these books. The content has been checked for accuracy. As the book is intended for educational purposes, the author shall not be responsible for any errors contained in the book.

The publication is designed to provide accurate and authoritative information with regard to the subject matter covered.

(This Module shall only be Used for Educational Purpose.)

INDEX

1 MACHINE INSTRUCTIONS

1.1 Introduction

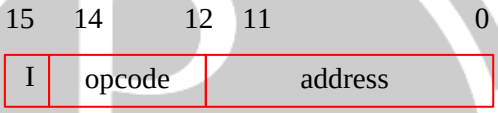
Opcode – Specifies the operation code. The number of bits in opcode depends on total operations.
Address – Specifies the operand address.

- The basic computer has three instruction code formats.

(i) **Memory – Reference instruction**

I = O ⇒ Direct addressing

I = 1 ⇒ Indirect addressing



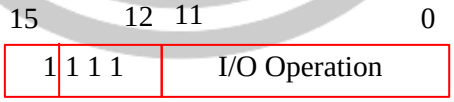
Instruction format

(opcode = 000 to 110)

(ii) **Register – Reference instruction**

(opcode = 111, I = 1)

(iii) **Input – output Instruction**



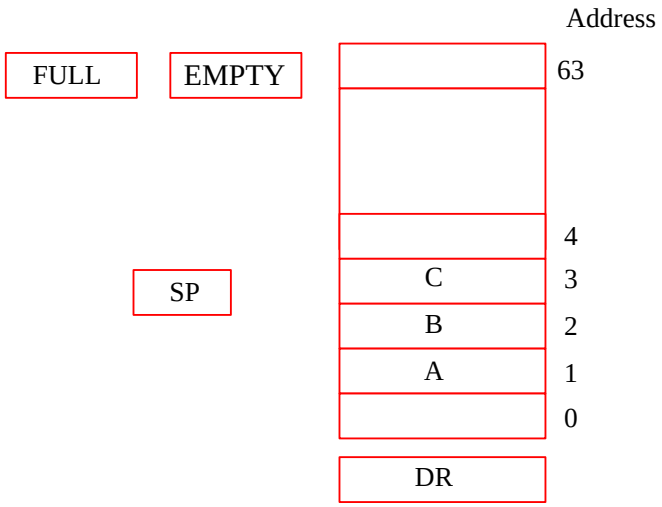
(op code = 111, I = 1)

A stack can be placed in a portion of a large memory or it can be organized as a collection of a memory words or registers. i.e., it may be

(i) Register stack
(ii) Memory stack

1.1.1. Register Stack

- The following figure shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of stack.
- Initially SP is cleared, EMPTY is set to 1, FULL is cleared to 0. If the stack is not full (if FULL = 0), a new item is inserted. With a push operation.



PUSH:

SP ← SP + 1 [increment stack pointer]
M{SP} ← DR [write item on top of stack]
If (SP = 0) then FULL ←1 [check if stack is full]
EMTY ← 0 [mark the stack not empty]

- A item is deleted from the stack if the stack is not empty (if EMTY = 0), called POP operation

DR ← M{SP} [Read item from top of stack]
SP ← Sp – 1 [Decrement stack pointer]
If (SP = 0) then EMTY – 1

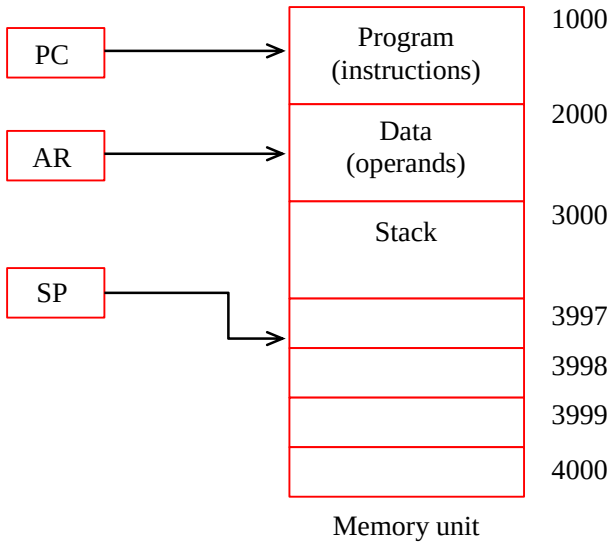
1.1.2. Memory Stack

- A stack can be implemented in a random-access memory. The stack can be implemented by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. i.e.,
- A new item is inserted with the push operation as follows:

PUSH: SP ← SP – 1
M[SP] ← DR

- A new item is deleted with a POP operation as follows

DR ← M[SP]
SP ← SP + 1



- A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. SP is automatically incremented or decremented with every **PUSH** or POP operation.

1.1.3. Reverse Polish Notation

- An expression in post fix form is often called reverse polish notation.

The infix expression (A + B) * [C*(D + E) + F] in reverse polish notation as AB + DE + C * F + *

- A reverse polish expression can be implemented or evaluated using stack for the expression

  X ← (A + B) * (C + D)
  With values A = 2, B = 3, C = 4, d = 2 is
  X ← (2 + 3) * (4 + 2) ← Infix expression
  x ← 23 + 42 + * ← Reverse polish expression

1.2 Instruction Types

- The basic unit of program is the instruction.

  Instructions are classified based on

  (1) Opcode – called functional classification

  (2) Based on number of references.

1.3 Functional Classification

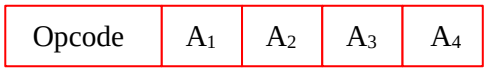
- Data transfer instructions – (LOAD, STOR, MOV, EXCT, IN, OUT, PUSH, POP)
- Arithmetic instructions – (INC, DEC, ADD, SUB, MUL, DIV)
- Logical Instructions & Shift instructions. (CLR, COM, AND, XOR, OR, SHR, SHL, SHRA, SHLA, RO, ROL….)
- Branching instructions. (ABR, JMP, SKP, CALL, RETURN)

1.4 Based on Number of References

Based on number of references, the instructions can be classified as

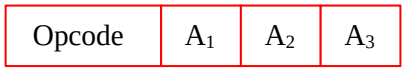
- 4 – address instructions
- 3- address instructions
- 2 – address instructions
- 1 – address instructions
- 0 – address instructions
- RISC Instructions

(i) 4 – Address Instructions:



- A_1, A_2 refers operands
- A_3 refers destination
- A_4 next instruction address.
- Since A_4 is like program counter, the processor which supports 4-address instructions need not use PC.
- The length of instruction is more, hence requires more than one reference.

(ii) 3 – address Instructions



- Each address field specify a register or memory operand.
- It results in short program when evaluating arithmetic expressions.
- Requires too many bits to specify three addresses

Example: To evaluate X = (A + B) * (C + D)

Add R_1, A, B $R_1 \leftarrow M[A] + M[B]$

Add R_2, C, D $R_2 \leftarrow M[C] + M[D]$

MUL X, R_1, R_2 $M[X] \leftarrow R_1 * R_2$

For these instructions program counter must be required.

(iii) 2 – address Instructions:

- A_1 – first operand and destination
- A_2 – second operand
- Uses MOV instruction for data transfer

Example:

$X \leftarrow (A + B) * (C + D)$

MOV R_1, A

ADD R_1, B

MOV R_2, C

ADD R_2, D

MUL R_1, R_2

MOV X R_1

One of the operand permanently lost.

(iv) 1 – address Instructions

- Uses an implied accumulator (AC) register for all data manipulation.
- Easily decoded and processed instructions.

Example: X ← (A + B) * (C + D)

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

(v) Zero – address Instructions:

- The operands are referenced complexity from stack.
- More complex approach than others.
- Any changes in order of operands effect the result.

Example: X = (A + B) * (C + D)

```
PUSH    A
PUSH    B
ADD
PUSH    C
PUSH    D
ADD
MUL
POP     X
```

(vi) RISC Instructions:

- All instructions are executed with in the registers of CPU, without referring to memory (Except LOAD, STOR)
- Uses in RISC processor
- LOAD & STOR are used between data transfer.

Example: X = (A + B) * (C + D)

```
LOAD    R1, A
LOAD    R2, B
LOAD    R3, C
LOAD    R4, D
ADD     R1, R1, R2
ADD     R3, R3, R4
MUL     R1, R1, R3
STOR    X, R1
```

❐❐❐

2 ADDRESSING MODES

2.1 Introduction

The various addressing modes commonly used are:

Implied Mode:

In this mode the operands are specified implicitly in the definition of the instruction.

Example:

- Complement Accumulator (CMA) [All register reference instructions that use an accumulator are implied - mode instructions]
- Zero-address instructions in a stack organized computer. [here the operands are implied to be on top of the stack]

Immediate Mode:

In this mode the operand is specified in the instruction itself.

- It is very faster
- Useful for initializing register to a constant value.

Example:

ADD 10 i.e., AC ← AC + 10

- The range of value limited by the address field

Register Mode:

In this mode the operands are in registers that reside within the CPU.

- A K-bit field can specify any one of 2^k registers.
- Reduces the length of the address field.

Example:

ADD R1

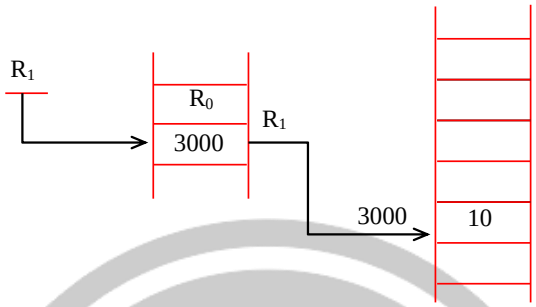
Register Indirect Mode:

In this mode the instruction specifies a register in the CPU whose contents give the address of an operand in memory.

- Before using this mode, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- The advantage is the address field uses fewer bits to select or register.

Example:

ADD (R1)

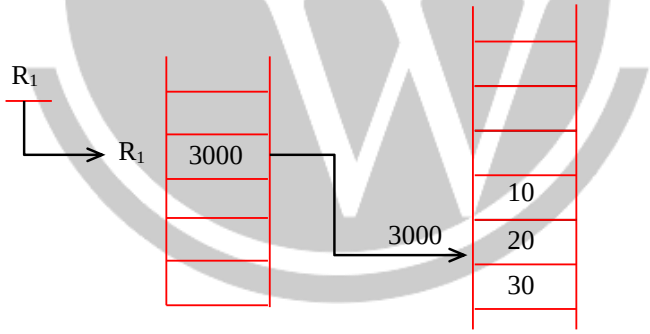


Autoincrement or Autodecrement Mode:

It is similar to register indirect mode except that the register is incremented or decremented after its value is used to access memory.

Example:

ADD (R_1)

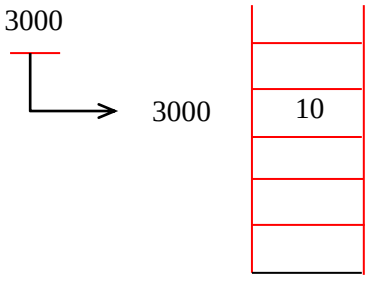


Direct Addressing Mode:

In this mode the effective address is equal to the address part of the instructions.

- The operand resides in memory and its address is given directly by the address field of the instruction.
- Used to represent global variables in a program.
- In a branch type instruction, the address field specifies the actual branch address space.
- Allows to access limited address space.
- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
- The direct addressing mode is also called "Absolute mode".

Example. ADD 3000



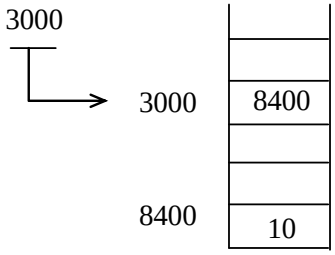
i.e. AC ← AC + M [3000]

Indirect Addressing mode:

In this mode, the address field of the instruction gives the address where the effective address is stored in memory.

- Allows to access larger address space
- Requires 2 memory cycles to read an operand

Example ADD 3000



2.1.1 Displacement addressing modes

The address field of the instruction added to the content of a specific register in the CPU. i.e,
Effective Address = Address part of instruction + content of Register.

2.1.2 The Various displacement addressing modes are

Relative addressing mode:

In this mode the content of program counter (PC) is added to the address part of the instruction.

- The address part is a signed number (2's complement) that can be either positive or negative.
- The result produces effective address whose position in memory is relative to the address of the next instruction.

EA = Address Part (off set) + PC value

- Used with branch-type instructions when the branch address is in the area surrounding the instruction word itself.
- The advantage is, address field can be specified with a small number of bits compared to direct address.

Indexed Addressing mode:

In this mode the content of an index register is added to the address part of the instruction.

- Index register contains index value.
- Address part specifies the beginning address of a data array in memory.

EA = Address Part (base address of data array) + Index register value (index value)

- Used for accessing array of data.
- The index register can be special CPU register or any general purpose register.

Base register Addressing mode:

In this mode the content of a base register is added to the address part of the instruction.

- The base register is assumed to hold base address.
- The address field gives the displacement relative to this base address.

EA = Address Part (displacement/offset) + Base register value (Base address)

❐❐❐

3 ALU DATA PATH

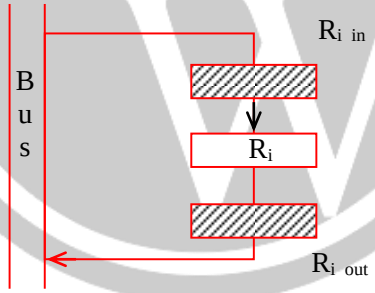
3.1 Introduction

- The sequence of operations involved in processing an instruction constitutes an instruction cycle.
  The instruction cycle consists of phases like.
  (1) Fetch cycle (2) Decode cycle
  (3) Operand fetch cycle (4) Execute cycle
- To perform these, the processor unit has to perform set of operations called “Micro-Operations”.

3.2 The Basic Operations Performed are

3.2.1 Register transfers:

- In general, the input & output of register R_i are connected to the bus via switched controlled by the signals R_i in and $R_i$out



- When Ri in is set to 1, the data on the bus are loaded into Ri.
- When Ri out is set to 1, the contents of register Ri are placed on the bus.

- To transfer the contents of R_1 to register R_4 : $R_4 \leftarrow R_1$.
- Enable the output of register R1 by setting R1 out to 1. This places the contents of R1 on the processor bus.
- Enable the input to register R_4 by setting $R_{4\,in}$ to 1. This loads data from the processor bus into register R_4.

3.2.2 Performing ALU operations

The ALU has two inputs A and B and one output

- A gets operand from output of MUX
- B gets operand from bus
- The result is gets stored in Z-register.

The sequence of steps for ALU operation $R_3 \leftarrow R_1$

(1) The contents of R_1 are loaded in Y.

(2) The contents of Y_1R_2 are applied to A & B inputs of ALU, performs ALU operation & stores the result in Z-register.

(3) The contents of Z are stored in R_3.

- The sequence of operations is

(1) $R_{1\ out}$, Y_{in}

(2) $R_{2\ out}$, select Y, Add, Z_{in}

(3) Z_{out}, $R_{3\ in}$

- The functions performed by ALU depends on the signals applied to its control lines. (Here Add line is set to 1).
- Only one register output can be connected to the bus during any clock cycle.
- The no of steps indicates no of clocks.

3.2.3 Fetching a word from memory (read operation)

- To read a memory word, consider MOV (R_1), R_2. The action needed to execute this instruction are

(1) MAR $\leftarrow$ [R_1]

(2) Start a read operation on the memory bus

(3) Wait for the MFC (Memory function to complete) response from memory.

(4) Load MDR from the memory bus

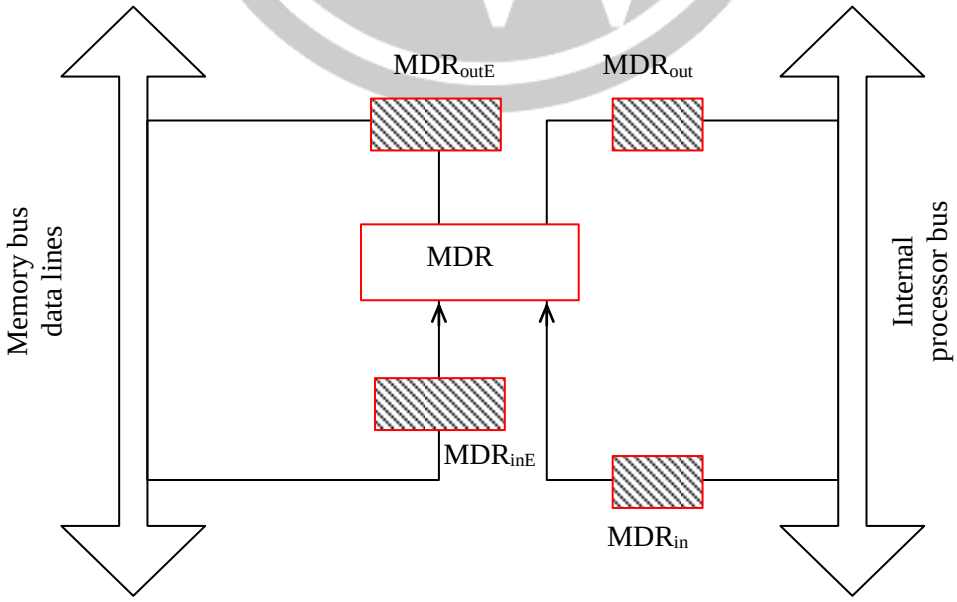
(5) $R_2 \leftarrow$ [MDR]

- Hence the memory read operation sequence of operations is

(1) $R_{1\ out}$, MAR_{in}, Read

(2) WMFC (wait for memory operation to complete), $MDR_{\ inE}$

(3) MDR_{out}, $R_{2\ in}$



3.2.4 Storing a word in Memory (write operation)

- The sequence of steps for write operation MOV R_2,
  (1) The desired address is loaded into MAR
  (2) The data to be written is loaded into MDR
  (3) Write command is issued.
  (4) Wait for memory operation to complete.
- The sequence of operations is
  (1) $R_{1\ out}$, MAR $_{in}$
  (2) R_{2out}, MDR_{in}, write
  (3) MDR_{outE}, WMFC

3.2.5 Branch Instructions

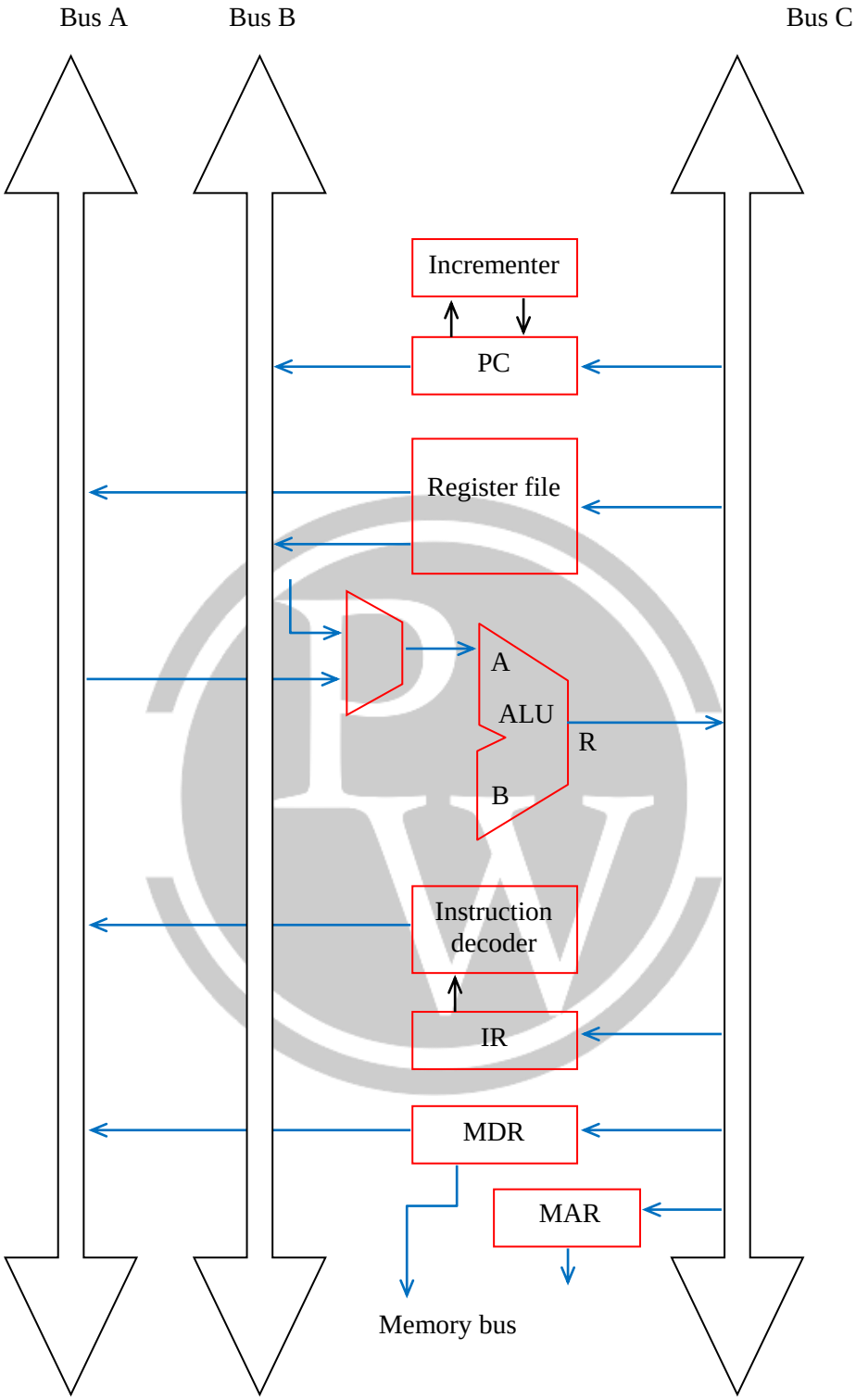
- A branch instruction replaces the content of PC with the branch target address. The address is obtained by adding an offset X, which is given in the branch instruction, to the updated value of PC.
- The control sequence for branching (unconditional is
  (1) PC_{out}, PC_{in}, Y_{in}, WMFC
  (2) Z_{out}, PC_{in}, Y_{in}, WMFC
  (3) MDR_{out}, IR_{in}
  (4) Offset – of – IR_{out}, Add, Z_{in}

3.2.6 Execution of complete Instruction

- Executing an instruction requires (Add (R_3), R_1)
  (1) Fetch the instruction
  (2) Fetch the first operand (memory location specified by R_3)
  (3) Perform the addition
  (4) Load the result into R_1
- The control sequence for the execution of ADD (R_3), R_1 in a single-bus organization is
  (1) PC_{out}, MAR_{in}, Read, Select, Add, Z_{in}
  (2) Z_{out}, PC_{in}, Y_{in}, WMFC
  (3) MDR_{out}, IR_{in}
  4) R_{3out}, MAR_{in}, Read
  (5) $R_{1\ out}$, Yin, WMFC
  (6) MDR_{out}, select Y, Add, Z_{in}
  (7) Z_{out}, $R_{1\ in}$, End

3.3 Multiple – Bus Organization

- With simple single bus structure, the resulting control sequence is quite long because only one data item can be transferred over the bus in a clock cycle. To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.
- The three bus organization of data path is

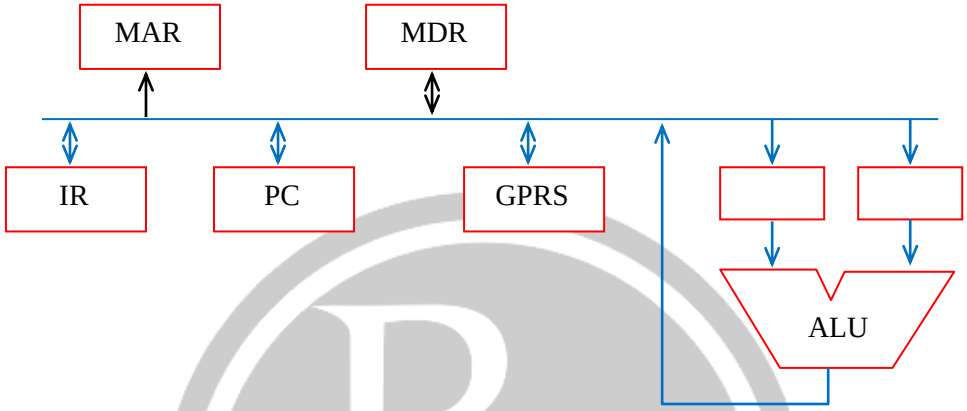


- All general – purpose registers are combined into a single block called register file. It has two – outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The data on bus 'C' to be loaded into third register during same clock.

- Buses A and B are used to transfer the source operands to the A and B inputs of ALU, the result is transferred to the destination over bus C.

- The control sequence for instruction ADD R_4, R_5, R_6
  (1) P_{Cout}, R = B, MAR_{in}, Read, Inc PC
  (2) WMFC
  (3) MDT_{out} B, R = B, IRin
  (4) $R_{4\ outA}$, $R_{5\ outB}$, select A, Add, $R_{6\ in}$, end.

Example (1)



The ALU, bus & registers in data path are of identical size. All operations including incrementing of PC and GPRSs are to be carried out in the ALU. 2-clock cycles are needed for memory read operation. (one for loading address in MAR and loading data from memory into the MDR).

❑❑❑

4 CPU CONTROL UNIT DESIGN

4.1 Introduction

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. The two approaches used for this purpose are

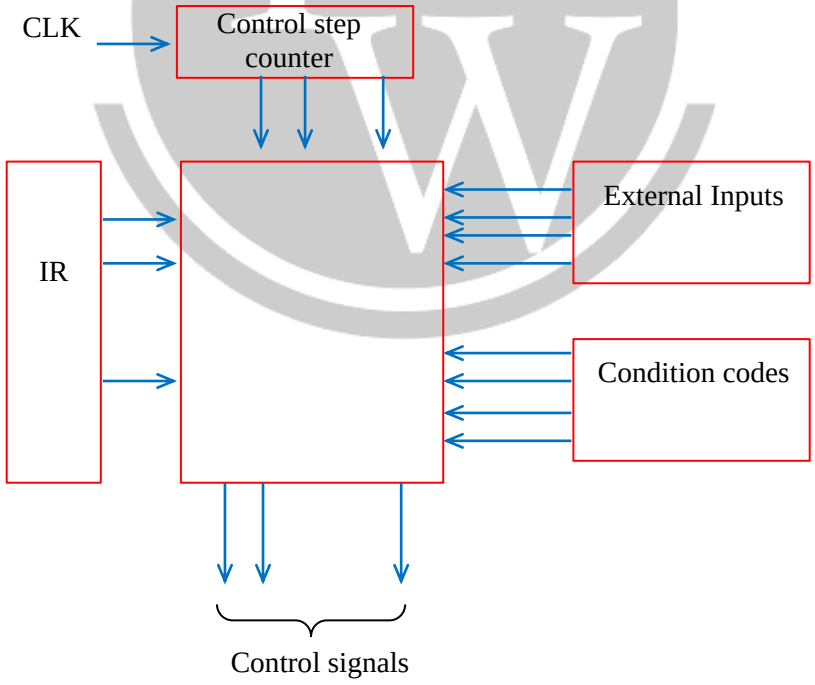
(1) Hardwired control

(2) Micro programmed control

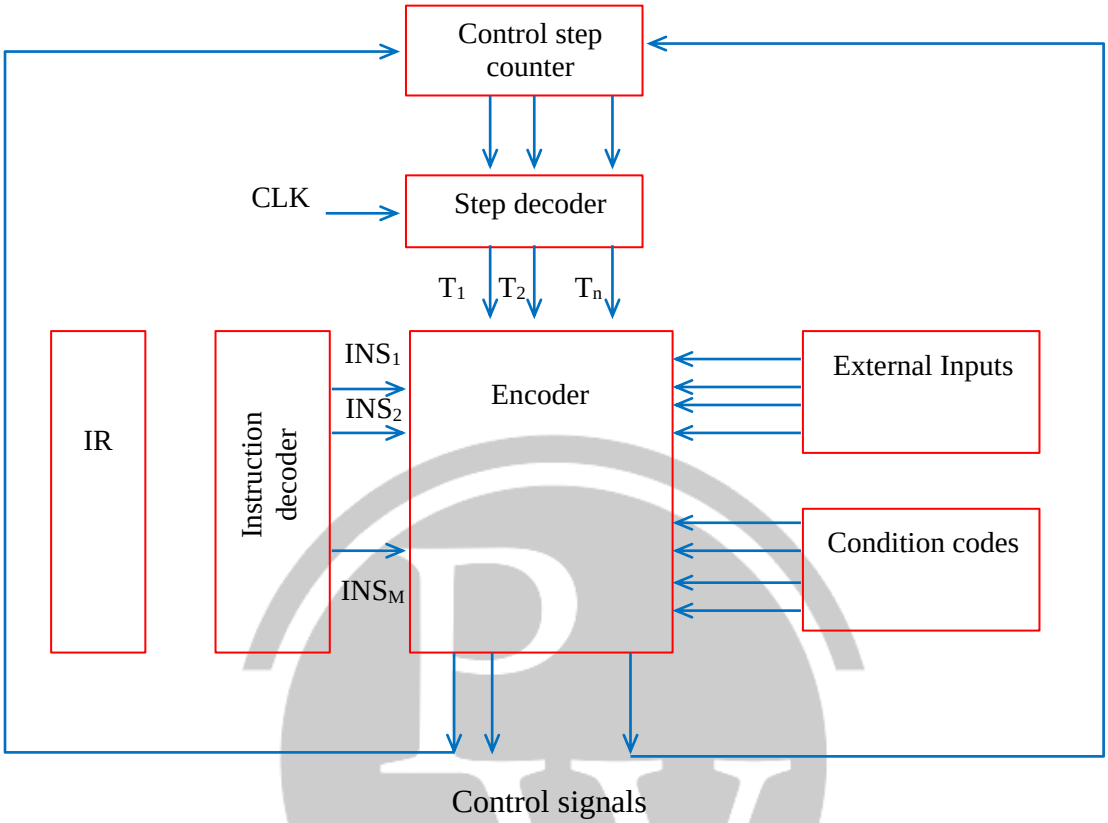
The purpose of control unit is to generate accurate timing & control signals.

4.1 Hardwired Control

- The control unit uses fixed logic circuits to interpret instructions and generate control signals from them. Every control signal is expressed as SOP (sum of products) expression and realized using digital hardware.



- The below figure shows the detailed hardwired control unit design.



- For executing an instruction completely each step is completed in one clock period. A counter is used to keep track of the control steps.
- The required control signals are determined with the following information.
  - (1) Contents of control step counter.
  - (2) Contents of instruction register
  - (3) Contents of condition code flags
  - (4) External input signals, such as MFC and interrupt requests.
  - The instruction decoder decodes the instruction loaded in IR
  - The step decoder provides a separate signal line for each step or time slot in a control sequence.
  - The encoder gets input from instruction decoder step decoder, external inputs and condition codes, thus uses to generate the individual control signals.
  - After execution of each instruction, the end signal is generated which resets control step counter and makes it ready for generation of control step for next instruction.

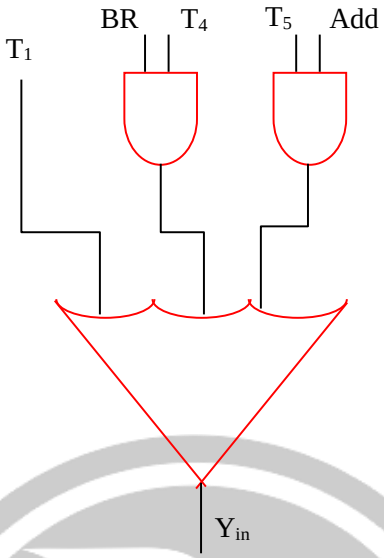
For example:

(1) The encoder circuit implements the following logic function to generate Y_{in}.

$Y_{in} = T_1 + T_5$ Add $+ T_4$ BR $+ \ldots..$

i.e. Here Y_{in} signal is asserted during time interval T_1 for all instructions, during T_5 for add instruction, during T_4 for branch instruction.

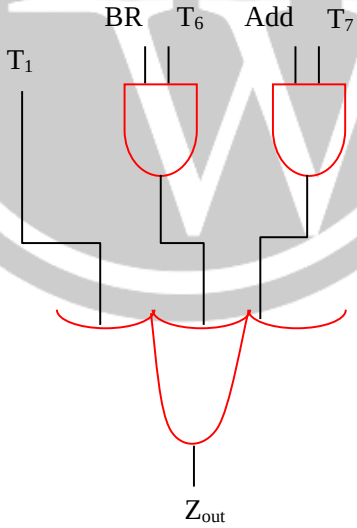
The generation of Y_{in} control signal is



(2) The logic function to generate Z_{out} signal can be given by

$Z_{out} = T_2 + T_7$ add $+ T_6$ BR + ……

i.e., the Z_{out} signal is asserted during time interval T_2 for all instructions, during T_7 for add instruction, during T_6 for unconditional branch …….. etc.

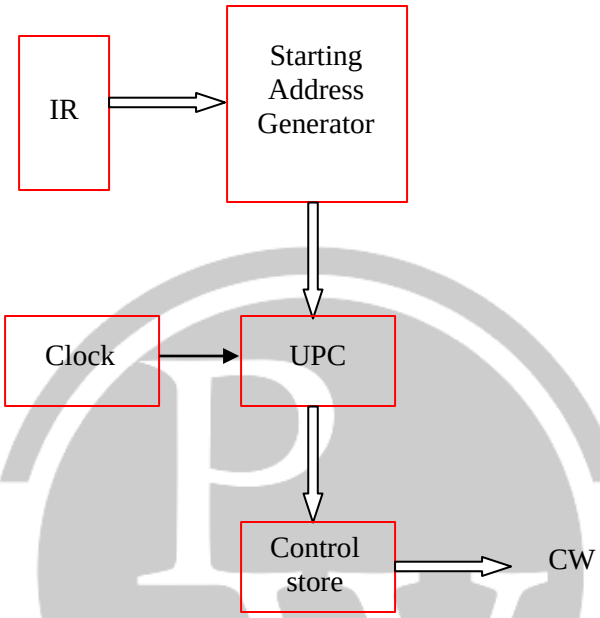


4.2 Microprogrammed Control

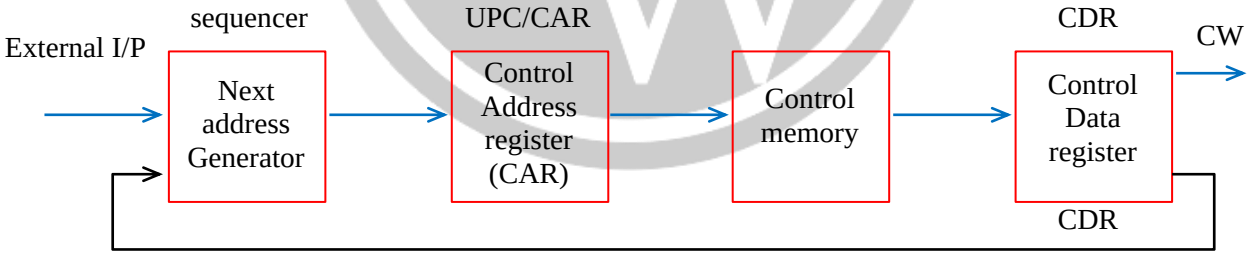
- Every instruction in a processor is implemented by a sequence of one or more sets of micro operations. Each micro operation is associated with a specific set of control lines which when activated, causes that micro operation to take place.
- Using micro programmed control, control signals are generated by a program. Using this the control signal selection and sequencing information is stored in ROM or RAM called control memory (CM).

- The control memory is part of control unit; it contains fixed programs (micro programs) that cannot be altered by occasional user.
- A control word (CW) is a word whose individual bits represent the various control signals.
- A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro routine for that instruction, (micro program)
- The individual control words in the microprogram are referred as micro instruction.

- The basic organization of a microprogrammed control unit is



- To read the control words sequentially from the control memory a "micro program counter" (μPC) is used.



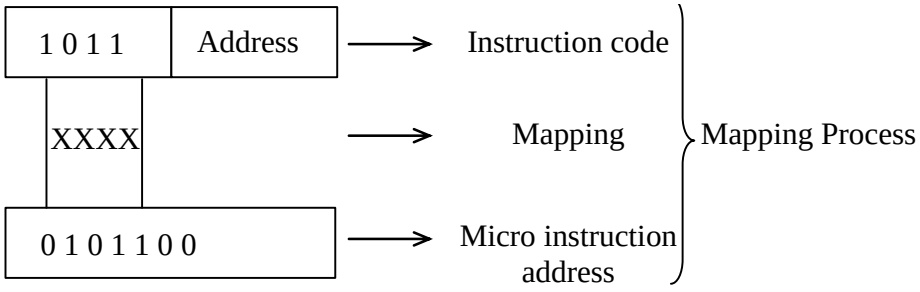
- The sequence of operations are:
  (1) CAR holds the address of next micro instruction to be read.
  (2) When address is available in UPC, the sequencer issues the READ command to CM.
  (3) The word from addressed location is read into CDR/UIR.
  (4) The UPC is incremented automatically by the clock, causing successive micro instructions to be read from CM.
  (5) The contents of UIR generates control signals which are delivered to various parts of the processor in correct sequence.
- The μPC or CAR can be updated with various options using the address sequencer circuit as:

(1) When a new instruction is loaded in IR, the μPC is loaded with the starting address of the micro routine for that instruction (called mapping process).

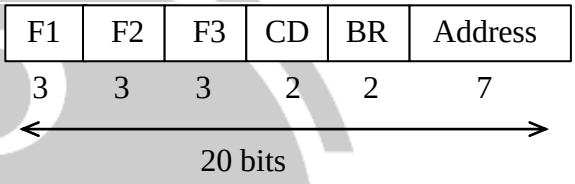
(2) When a branch instruction is (micro) encountered and branch condition satisfied, the μPC is loaded with the branch address.

(3) When END instruction is encountered, the μPC is loaded with address of first word.

(4) In any other situation, the μPC is incremented every time a new micro instruction is fetched from CM.



- The transformation from the instruction code bits to an address in control memory where the routine is located is called Mapping process.
- The basic microinstruction format is
- The function fields (F1, F2, F3,), condition field (CD) & Branch field (BR) may be optional.



Comparison between Hardwired & Microprogramed

	Hardwired	**Micro programmed**
1. Speed	Fast	Slow
2. Control functions	Implemented in Hardwire	Implemented in software
3. Ability to handle complex instructions	Complex	Easier
4. Design process	Complicated	Orderly and systematic
5. Instruction set size	Under 100	Over 100
6. ROM size	NIL	2k to 10k (20–400 bit micro instruction)
7. Applications	RISC processors	CISC, Main frames

4.2.1. The Micro Programmed Control Unit Can be

(1) Horizontal Microprogramming:

- One bit per control signal.
- Maximum parallelism i.e., more than one signal can be simultaneously.
- No extra decoders are required for decoding.
- The length of control word is large, need to access more than once from control memory.

(2) Vertical Microprogramming:

- The control signals are encoded in form for k-bits 2^k signals.
- Maximum degree of parallelism is 1.
- The length of micro instruction is small.
- Response is relatively slower.
- Requires a decoder additionally.

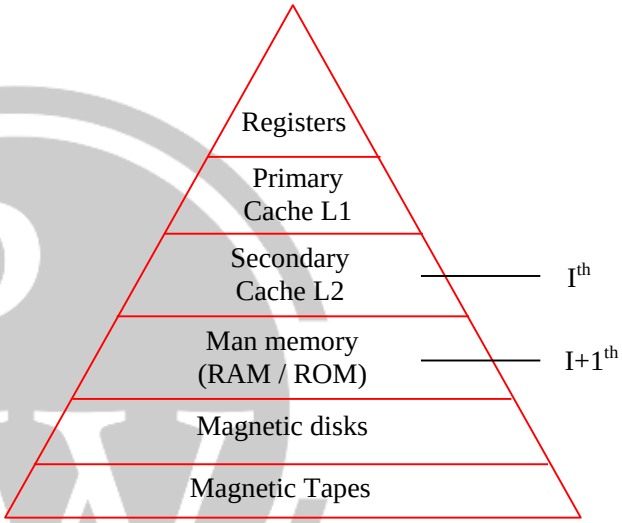
- To increase the degree of parallelism soft vertical microprogramming is used which divides the control signals into mutually exclusive groups. Each group is associated with associated number of control bits. (i.e., combination of both vertical and horizontal microprogramming).

❑❑❑

5 MEMORY INTERFACING

5.1 Memory Hierarchy

- The memory hierarchy system consists of all storage devices.
  The purpose of memory hierarchy is to bridge the speed mismatch between the fastest processor to the slowest memory component at reasonable cost.



- In the structure of memory hierarchy I^{th} level memory is physically positioned higher than $(I + 1)^H$ level memory.
- Let Ti, Si, Ci, & Fi are the access time, size, cost per bit and frequency of references to the I^{th} level memory. Therefore

$$Ti < Ti + 1$$

$$Si < Si + 1$$

$$Ci > Ci + 1$$

$$fi > fi + 1$$

- Since same data presents at different levels. I^{th} level memory is the subset of $I\ i + 1^{th}$ level.

  i.e. $I\ i \subset I\ i + 1$

5.2 Memory Characteristics

Location: Memory can be placed in 3 locations like registers, main memory, Auxiliary (or) secondary storage.

Capacity: Word size, number of words i.e. capacity = number of words * word size.

Unit of transfer: Maximum number of bits that can be read or written (blocks, bytes…)

Access method: Random or sequential

Performance: Access time, memory cycle time, transfer rate, physical type.

Physical: Volatile / non volatile

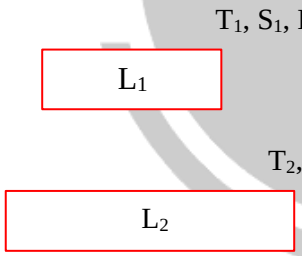
erasable / non erasable

	Serial Access		Random Access
(1)	Memory is organized into units of data called records/blocks accessed sequentially	**(1)**	Each storage location has an address uniquely
(2)	Access time depends on position of storage location	**(2)**	Access time is independent of storage location
(3)	Slower to Access	**(3)**	Faster to access
(4)	Cheaper	**(4)**	Costly relatively
(5)	Nonvolatile memories	**(5)**	May be relative or non
(6)	Example: Magnetic tapes	**(6)**	Example: Magnetic disks.

- When processor reads I^{th} level memory, if it is found in that level, his will occur otherwise it will be fault.

5.2.1. In a Two – Level Memory System

Case – I: When fault occurs, in L1 reads from L2 (Proxy Hierarchy)



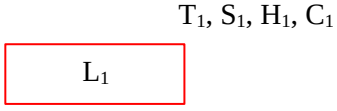
$T_{avg} = H_1 * T_1 + (1–H_1) * T_2$

Case – II: When fault occurs in L_1, must be brought from L2 to L1 memory. (**strict hierarchy**)

$T_{avg} = H_1 * T_1 + (1–H_1) * (T_2 + T_1)$

5.2.2. In a Three – Level Memory System

Case – I: When fault occurs in one level, then reads from its down level.



T_2, S_2, H_2, C_2

L_2

T_2, S_2, H_2, C_2

L_3

$T_{avg} = H_1 * T_1 + (1 – H_1) * H_2 * T_2 + (1 – H_1) * (1 – H_2) * T_3$

Case – II: When fault occurs, must access from L1.

$T_{avg} = H_1 * T_1 + (1–H_1)\,(H_2)\,(T_2 + T_1) + (1–H_1)\,(1–H_2)\,(T_3 + T_2 + T_1)$

* Average cost per bit

$$C_{Avg} = \frac{C_1S_1 + C_2S_2}{S_1 + S_2} \quad \text{(Two - level)}$$

$$C_{Avg} = \frac{C_1S_1 + C_2S_2 + C_3S_3}{S_1 + S_2 + S_3} \quad \text{(Three - level)}$$

❑❑❑

6 INPUT OUTPUT INTERFACING

6.1 Input – Output Organization

- The input – output subsystem (I/O) provides an efficient mode of communication b/n system and outside environment. The Commonly used peripheral devices are. Keyboard, monitors, printer & magnetic tape, magnetic disc…..
- The input & output devices that communicates with computer & people usually with alphanumeric character from ASCII – 128-character set, 7 – bits are used represent each character. It contains 26 upper case letters, 26 lower case letters, 10 numerals, 32 special chars such as %, *, $ …… In general ASCII chars are stored in a single unit called 1 byte (8-bits). The 8^{th} bit may be used for parity bit for error detection.

6.1.1. Input – Output Interface

- The I/o interface provides a method of transferring information b/n internal storage (main memory) & external I/o devices. The devices/ peripherals connected to a computer need special communication links for interfacing with CPU because.
  - Peripherals are electromechanical & electromagnetic devices, whereas CPU & memory are electronic devices hence the operations are different.
  - The data transfer rate is shown then the transfer rate of CPU.
  - The data codes & formats are different.
  - The operating modes of peripherals is different and must be controlled.
- The four types of I/o command an interface will receive are

6.1.2. Control Command

Issued to activate the peripheral & and to inform it what to do. Depending on mode of operation a control command sequence is issued.

6.1.3. Status Command

Used to test various status conditions in the interface & the peripheral. Eg. Checking status of device, errors detected by interface. The status information is maintained in status register.

6.1.4. Data Output

Causes the interface to respond by transferring data from the bus into one of its registers buffer.

6.1.5. Data Input

With this command, interface receives an item of data from peripheral & places it in its buffer register. Then transfers to data bus of processor.

The I/o read & I/o write are enabled during I/o transfer, memory ready & memory write are enabled during memory transfer. The two configurations possible for communication are.

(a) Isolated I/o:

The CPU has distinct input & output instructions, each instruction is associated with the address of an interface register. When CPU fetches & decodes the opcode, it places address associated with the instruction into the common address lines, at the same time, it enables the I/o read or I/o write control line.

An isolated I/o method isolates memory & I/o addresses each has its own address space, hence memory address values are not affected by interface address. Uses one common bus for memory & I/o, with common control lines.

(b) Memory – Mapped I/o:

In this configuration, only one set of read & write signals and do not distinguish b/n memory & I/o addresses. and I/o.

6.2 Modes of Data Transfer

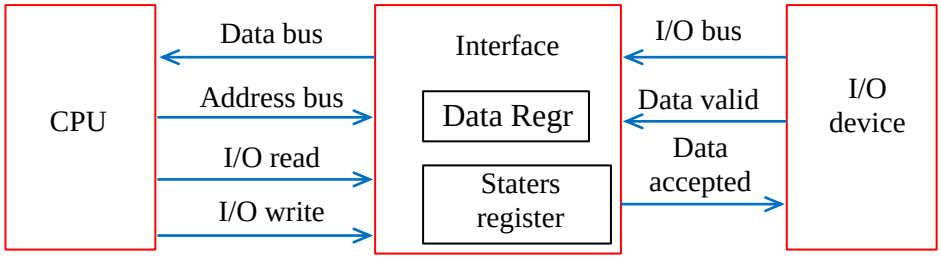
Data transfer b/n the central computer (main memory) and I/o device may be handled in a variety of modes like programmed I/o, Interrupt – Initiated I/o & Direct memory access.

6.2.1. Programmed I/O

- Programmed I/o operations are the result of I/o instruction written in computer program:

Example:

In programmed I/o, the I/o device doesn't have direct access to memory. A transfer from I/o device to memory requires the CPU to extents several instruction.



F – flag bit

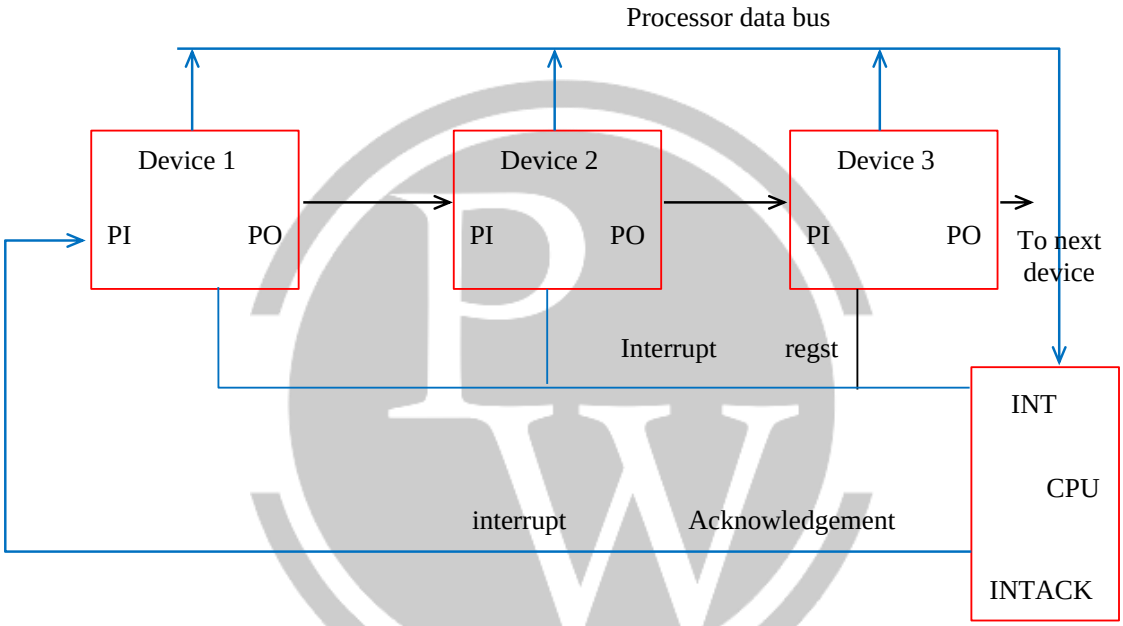
- (Data transfer from I/o device to CPU) The device transfers bytes of data one at a time as they are available. When a byte of data is available the device places it on I/o bus and enables its data valid line. The Interface accepts the byte into its data register and enables the data accepted line. The interface sets the Flag bit. Then the device disables the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface.

6.2.2. Interrupt – Initiated I/O

- In programmed I/O, the CPU stays in a program loop until the I/o unit indicates that it is ready to transfer. Hence this process keeps processor busy needlessly. meanwhile keeps monitoring the device. When the interface determining that the device is ready, it generates an interrupt request
- **Priority Interrupt:** It is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. It also determines which conditions are permitted during processing of an

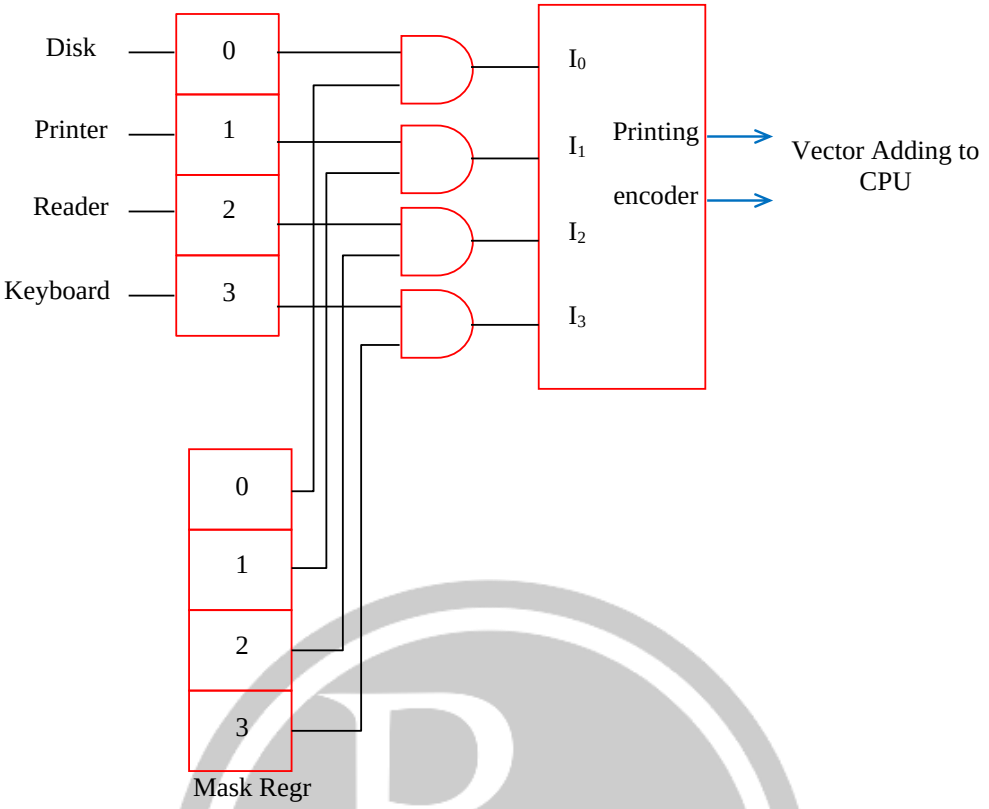
6.2.3. Daisy – Chaining Priority (Serial – Priority Interrupt)

The system consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in first position followed by lower – priority devices. The lowest priority will be placed last in the chain.



6.2.4. Parallel Priority Interrupt

- This method uses a register whose bits are set separately by the interrupt signal from each device Priority is established according to the position of bits in the register. The CKT will also include a mask register to control the starting of each interrupt request. The mask register can be programmed to disable low-priority interrupts while a high – priority device is being revised. If will also allow a high-priority device to interrupt the CPU while low – priority device is being serviced.



6.2.5. Direct Memory Access (DMA)

- The speed of data transfer can be increased by removing CPU from the path and the peripheral device to manage the memory buses directly. This kind of transfer technique is called DMA transfer during DMA transfer the CPU is idle and has no control of the memory buses. The DMA controller takes control of buses to manage the transfer directly b/n i/o device & memory.
- To keep CPU idle to use bus, the "bus request (BR) input is used by the DMA controller to request the bus to relinquish control of the buses. When BR is active, CPU terminates current execution and places data, control & address lines in high impedance state. Then the bus behaves like an open circuit. The CPU then activates. "Bus grant" (BG) output to DMA, then DMA takes control of the bus to transfer without CPU intervention, when the transfer completes DMA disables the Bus request & CPU disables the bus grant. Then the CPU gets the control of the buses.

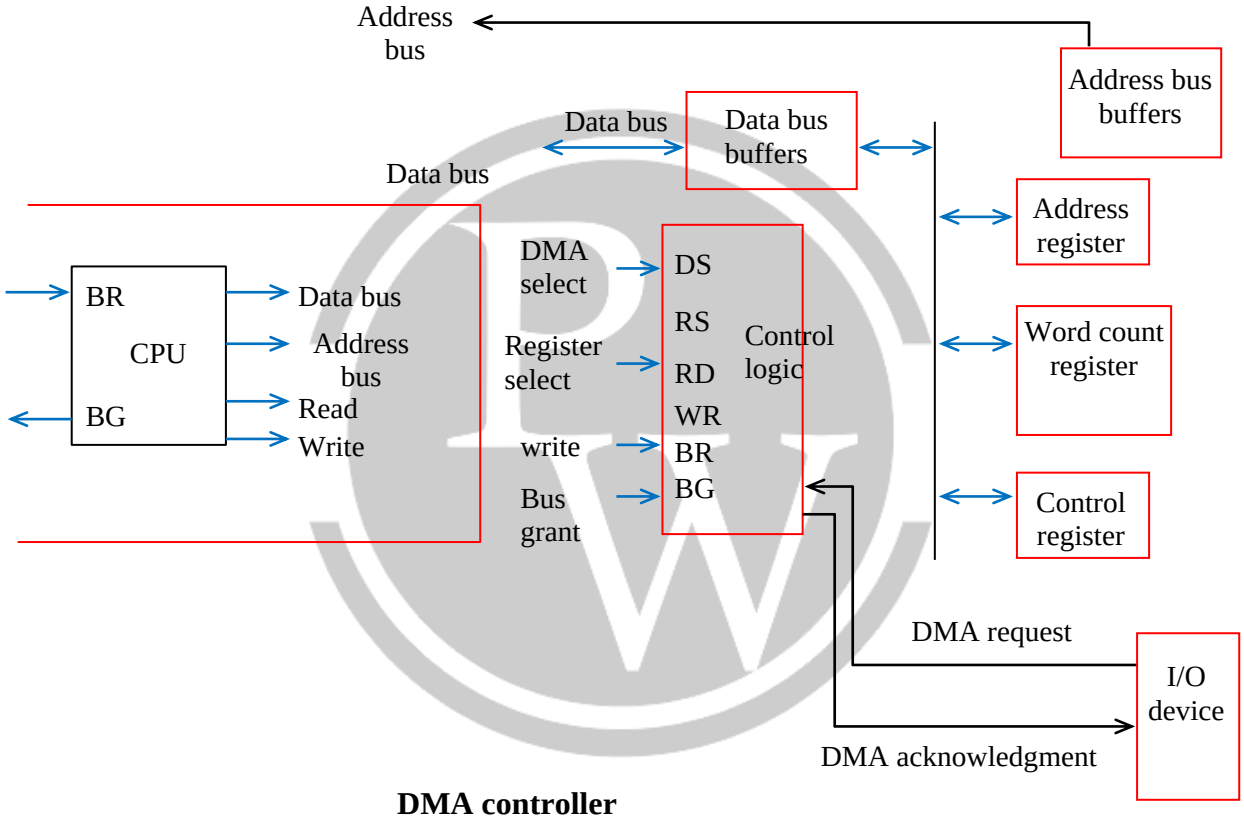
6.3. The DMA Transfer Take Place in Two Modes

(1) Burst transfer:

A block sequence consisting of a number of memory words is transferred in a continuous burst while DMA controller is master of the memory buses. Used to transfer fast devices like magnetic disks for large volumes. In this mode the data transfer will not be stopped, until an entire block is transferred.

(2) Cycle Stealing:

In this mode, DMA controller transfers one word at a time, after which it must return control of the buses to the CPU, later it will 'steal' memory cycle when CPU is idle.



DMA controller

- The DMA controller communicates with the CPU the data bus and control lines. The register in DMA are selected by CPU through address bus by enabling DS & RS inputs. When BG = 1 (bus grant) the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying address in address bus & activating the RD or WR control. The DMA communicates with the external device through the request and acknowledge lines.

6.4. The CPU Initialize the DMA by Sending

(1) The starting address of memory block where data are available (for read) or where data are to be stored (for write)

(2) Word count, the no of words in memory block.

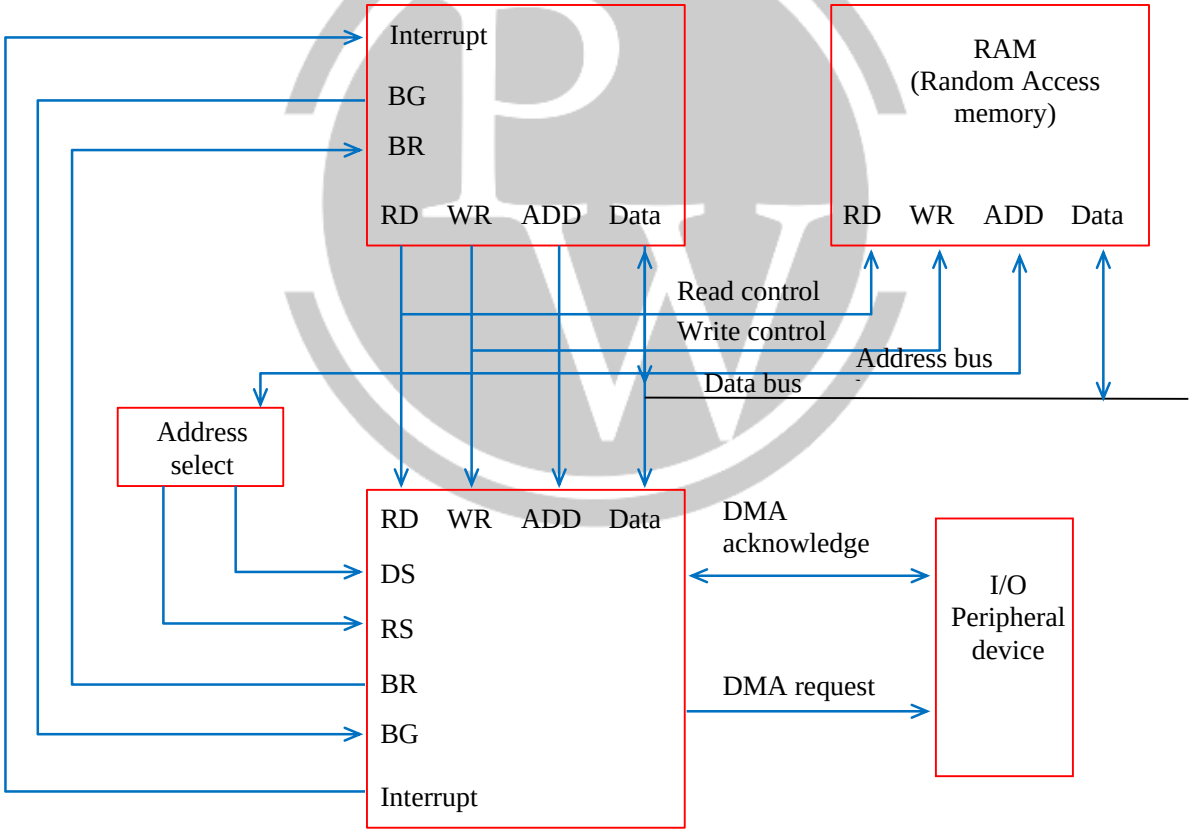
(3) Control of specify mode of transfer such as read or write.

(4) A control to start the DMA transfer.

- Once DMA is initialized, the CPU stops communicating with the DMA unless it receivers an interrupt signal.

6.5. DMA Transfer

When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing DMA that its buses are disabled. DMA then puts the current value of its address register into the adder bus and initiates RD or WR signal. It then sends DMA acknowledge to the peripheral device. When BG = 0, then CPU communicates with the internal DMA registers, when BG = 1 then RD & WR lines are used from DMA controller to RAM to specify read or write operation.



When the device receives DMA acknowledge, it puts a word in the data bus (write) or receives on word from the data bus (read). In this way the peripheral communicates with memory without any involvement of CPU. For each word transfer DMA increments the address register and decrements its word count register. When count reaches to zero, the DMA disables bus request line so that the CPU can continue to execute its program DMA transfer is very useful for fast transfer of data.

❑❑❑

7 PIPELINING

7.1 Introduction

- "Parallel processing" denotes a large class of techniques that are used to provide simultaneous data–processing tasks for the purpose of increasing the computational speed of a computer system.
- The purpose of parallel processing is to speed up the computer processing capability and increases its throughput. "Through put" is the amount of processing that can be accomplished during a given interval of time".
- Parallel processing can be viewed from various levels as

  (i) At registers level, Registers with parallel load operate with all the bits of the word simultaneously. (Instead of shift registers). This is at lowest level.

  (ii) Higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

  (iii) By distributing the data among the functional units in multiple. Eg: The arithmetic logical & shift operations can be separated.

  (iv) Using multiple processors. (Flynn's classification) (SISD, SIMD, MISD, MIMD).

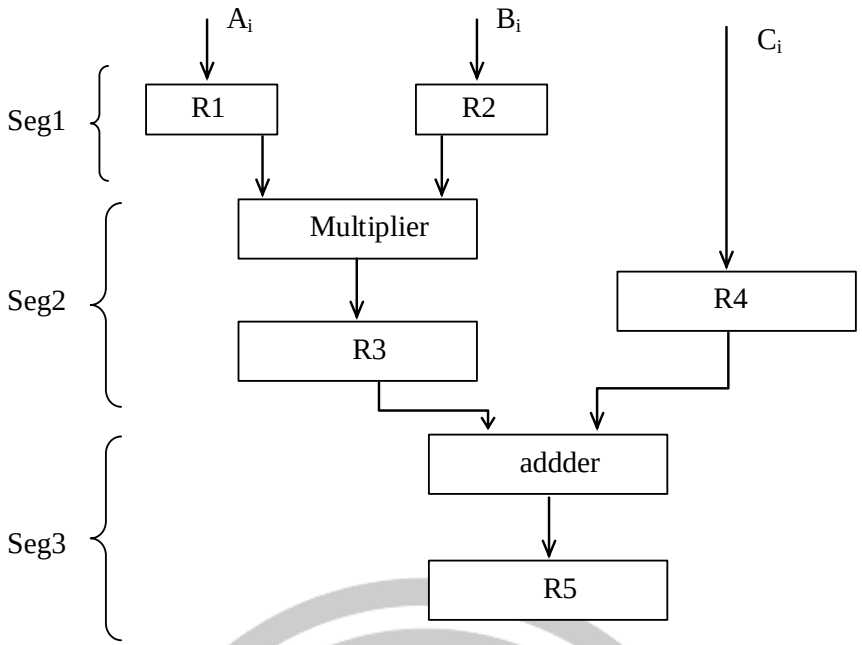
  (v) Using pipelining in unit processor systems.

7.2 Pipelining

- Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- The result obtained from one segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

Example:

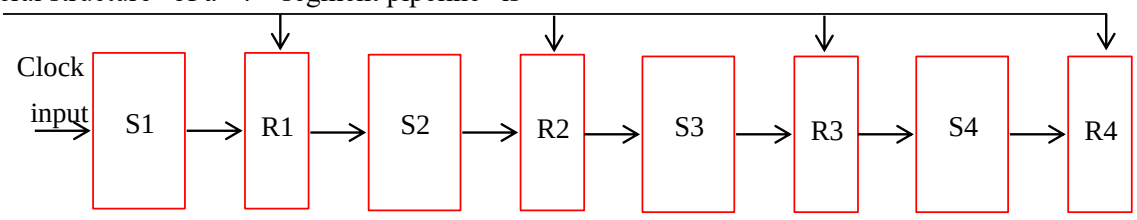
The perform Ai * Bi + Ci , for i = 1 to 6. Each sub operation multiply & add implemented in a segment with in a pipeline. Each segment has one or more registers.



- R1 through R5 are registers that receive new data with every clock pulse.
- The effect of each clock pulse can be shown as

Clock pulse Number	Segment – 1		Segment - 2		Segment - 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	---
2	A2	B2	$A_1 * B_1$	C1	---
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	---	---	A6 * B6	C6	A5 * B5 + C5
8	---	---	---	---	A6 * B6 + C6

- The "General structure" of a "4 – segment pipeline" is



- The operands pass through all four segments in a fixed sequence.
- Each segment consists of combinational circuit Si that performs a sub operation.
- The segments are separated by registers Ri that holds the intermediate results between stages.
- Information flows between adjacent stages under the control of common clock applied to all registers simultaneously.
- A task as the total operation performed going through all the segments in the pipeline.
- The behaviour of a pipeline can be illustrated with "Space – time diagram". It shows the segment utilization as a function of time. The following diagram is for tasks T1 to T6 executed in 4 – segment pipeline.

Segments	1	2	3	4	5	6	7	8	9
S – 1	T1	T2	T3	T4	T5	T6			
S – 2		T1	T2	T3	T4	T5	T6		
S – 3			T1	T2	T3	T4	T5	T6	
S – 4				T1	T2	T3	T4	T5	T6

Clock cycles

- "An Arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments."

7.2.1 Pipeline performance evaluation

- Consider a K – segment pipeline with a clock cycle time t_p is used to execute n tasks.
- To complete n – tasks using a K – segment pipeline requires K + (n–1) clock cycles.

(1) The number of clock cycles needed in a pipeline to execute 100 tasks in 6 segments.

Ans. K = 6

n = 100

$\Rightarrow$ 6 + (100-1) = 105 clocks

- The processing time in each stage is called as "stage delay", In a pipeline. (t_p).
- The time delay due to interstage transfer of data is "interstage delay" (td)
- The interstage delays can be same between the stages, stage delay very from stage to stage based on segment operation.

The time period for clock cycle, 't'$_p$ is

$t_p = \max\{t_i\} + t_d$

$t_p = t_m + t_d$

Since $t_m >> t_d$, maximum stage delay denotes the clock period.

i.e. $t_p = t_m$

- The total time required in pipeline execution is

$T_P = [K + (n–1)] * t_p$.

7.2.2 Speed up Ratio

- The speed up of a K – stage pipeline over an equivalent non-pipelined processor is defined as

$$S = \frac{\text{time without pipeline}}{\text{time with pipeline}}$$

- Consider a non-pipeline processor that performs the same operation as pipelined and takes a time equal to "t_n" to complete each task.

$$S = \frac{n * t_n}{(K + (n-1)) * t_p}$$

- As the number of tasks increases, k + n - 1 approaches to n

Since $t_n \simeq K * t_p$

$$S = \frac{n * K * t_p}{(K + (n-1)) * t_p} \qquad \therefore t_n \simeq nt_p$$

$$S = \frac{nK}{K + (n-1)}$$

$$S = \frac{n\,t_n}{n * t_p} \qquad \therefore K + n - 1 \simeq n$$

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

For large number of tasks

$$S = \frac{nk}{K + n - 1} \qquad S = \frac{t_n}{t_p}$$

Or

$$= \frac{nK}{n} \qquad S = \frac{K \times t_p}{t_p}$$

$$\boxed{S = K} \qquad \boxed{S = K}$$

- The maximum speed up that can be achieved in a pipelined processor is equal to "number of stages". [as n is large, K + n –1 is K]. (K).

$$S_{ideal} = S_{max} = K$$

7.2.3 Efficiency

- The efficiency of a pipeline is defined as the ratio of speed up factor and the number of stages in the pipeline.

$$E_k = \frac{S}{K} = \left(\frac{nk}{K + (n-1)}\right) / K$$

$$E_K = \frac{n}{K+n-1}$$

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

7.2.4 Throughput

- It is the number of tasks that can be completed by a pipeline per unit time.

$$H_K = \frac{n}{(K+(n-1))\times t_p}$$

7.2.5 Stall cycle

- The performance of a pipeline is influence by

  Number of instructions

  Uneven stage delays

  Buffer overhead (Interstage delay)

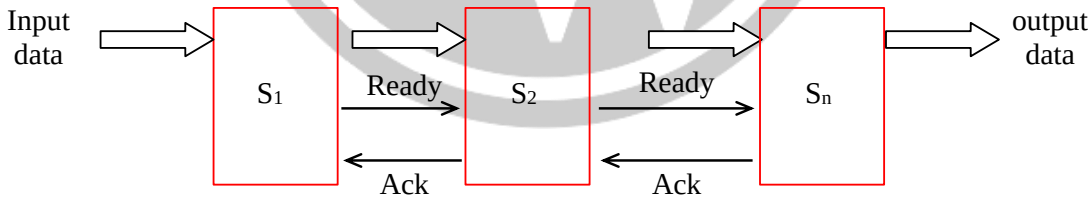
  Dependencies.
- The objective of pipelines is $CPI_{avg} = 1$.

  (i.e. Clocks per instruction – CPI)
- Depending on control mechanism used, the pipelines can be categorized as

7.2.6 I Asynchronous pipeline

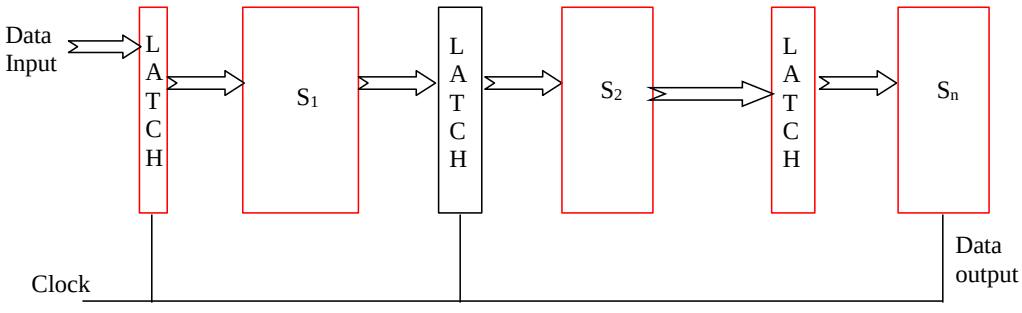
- Data flow along the pipeline stages is controlled by handshaking protector i.e. when S_i is ready to send/transmit, it sends ready signal to S_{i+1} and S_{i+1} send A.

  ck to Si after receiving.



7.2.7 II Synchronous pipeline

- Clocked high speed latches are used to interface between stages. At the falling edge of the clock pulse, all latches transfer data to the next stages simultaneously.



7.2.8 Instruction pipelining

- An instruction pipeline operates on a stream of instructions by overlapping the Fetch, Decode, Execute and other phases of instruction cycle.
- The instruction cycle with 4 – segments is

Instructions	1	2	3	4	5	6	7	8	9	Clock
I 1	FI	DA	FO	FX						
I 2		FI	DA	FO	EX					
I 3			FI	DA	FO	EX				
I 4				FI	DA	FO	EX			
I 5					FI	DA	FO	EX		

Here FI - fetches an instruction

DA - Decodes the instruction & calculates effective address

FO - fetches the operand

EX - Executes the instruction

- "In general, each stage in a pipeline is expected to complete its operation in one clock cycle, hence the clock period must allow the longest task to be completed.
- "The performance of a pipeline is high if different stages require about same amount of time."
- The use of cache solves the memory access problem.

(5) Consider a 4–stage pipeline, where different instructions require different number of clock cycles, at different stages. The total number of clocks required for execution of 4 instructions is _____

Ans.:

	1	2	3	4	
I 1	2	1	2	2	**7**
I 2	1	3	3	2	**9**
I 3	2	2	2	2	**8**
I 4	1	3	1	1	**6**
					30

Through sequential process 30 clocks. But using pipeline the number of clock cycles required is 14. That is,

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
I1	S1	S1	S2	S3	S3	S4	S4							
I2			S1	S2	S2	S2	S3	S3	S3	S4	S4			
I3				S1	S1	-	S2	S2	-	S3	S3	S4	S4	
I4						S1	-	-	S2	S2	S2	S3	-	S4

7.2.9 Data Hazards

- A data hazard is situation in which the pipeline is stalled because the data to be operated on are delayed for some reason, as illustrated in Figure. We will now examine the issue of availability of data in some detail.
- Consider a program that contains two instructions, I_1 followed by I_2. When this program is executed in a pipeline, the execution of I_2 can begin before the execution of I_1 is completed. This means that the results generated by I_1 may not be available for use by I_2. We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially. The potential for obtaining incorrect results when operations are performed concurrently can be demonstrated by a simple example. Assume that A = 5, and consider the following two operations:

$$A \leftarrow 3 + A$$

$$B \leftarrow 4 \times A$$

- When these operations are performed in the order given, the result is B = 32. But if they are performed concurrently, the value of A used in computing B would be the original value, 5, leading to an incorrect result. If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in the second instruction depend on the result of the first instruction. On the other hand, the two operations.

$$A \leftarrow 5 \times C$$

$$B \leftarrow 20 + C$$

- Can be performed concurrently, because these operations are independent.

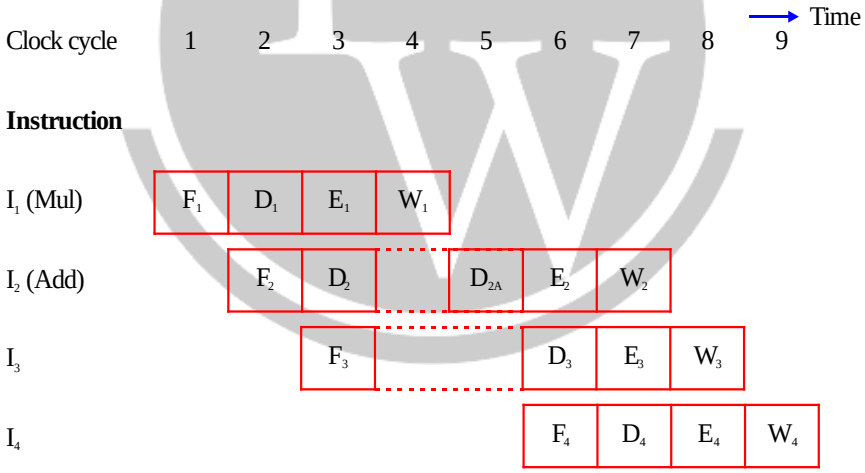


Fig: Pipeline stalled by data dependency between D_2 and W_1

- This example illustrates a basic constraint that must be enforced to guarantee correct results. When two operations depend on each other, they must be performed sequentially in the correct order. This rather obvious condition has far-reaching consequences. Understanding its implications is the key to understanding the variety of design alternatives and trade-offs encountered in pipelined computers.
- Consider the pipeline in Figure. The data dependency just described arises when the destination of one instruction is used as a source in the next instruction. For example, the two instructions

Mul R2.R3.R4

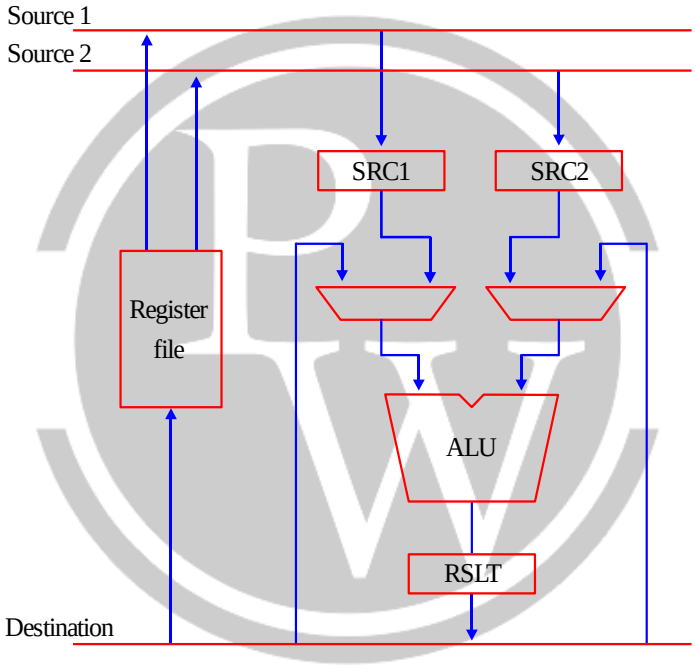
Mul R5.R4.R6

- Give rise to a data dependency. The result of the multiply instruction is placed into register R4. which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete; execution would proceed as shown in Figure. As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed unit the W step of multiply instruction has been completed. Completion of step D_2 must be delayed to clock cycle 5, and is shown as step D_{2A} in figure. Instruction I_3 is fetched in cycle 3, but its decoding must be delayed because step D_3 cannot precede D_2. Hence, pipelined execution is stalled for two cycles.

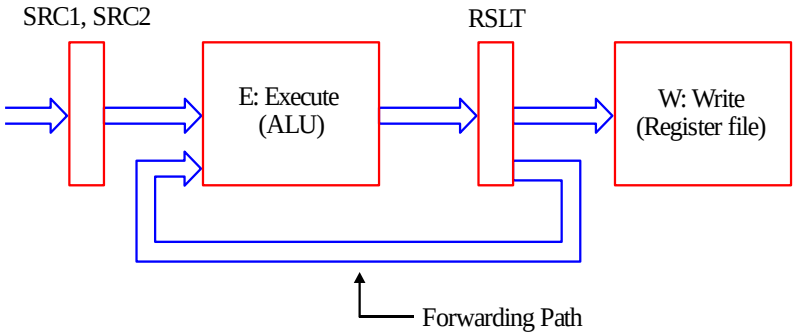
7.2.10 Operand Forwarding

- The data hazard just described arises because one instruction, instruction I_2 in Figure, is waiting for data to be written in the register file. However, these data are available at the output of the ALU once the Execute stage completes step E_1. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I_1 to be forwarded directly for use in step E_2.

Figure shows a part of the processor data path involving the ALU and the register file.



(a) Datapath



(b) Position of the source and result registers in the processor pipeline

- Interstage buffers needed for pipelined operation, as illustrated in Figure. With reference to Figure, registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3. The data forwarding mechanism is provided by the blue connection

lines. The two multiplexes connected at the inputs to the ALU allow the data on the destination bust to be selected instead of the contents of either the SRC1 or SRC2 register.

- When the instructions in Figure 8.6 are executed in the data path of Figure, the operations performed in each clock cycle are as follows. After decoding instruction I_2 and detecting the data dependency, a decision is made to use data forwarding. The operand not involved in the dependency, register R2, is read and loaded in register SRC1 in clock cycle 3. In the next clock cycle, the product produced by instruction I_1 is available in register RSLT, and because of the forwarding connection, it can be used in step E_2. Hence, execution of I_2 proceeds without interruption.

7.2.11 Handling Data Hazards in Software

- In Figure, we assumed the data dependency is discovered by the hardware while the instruction is being decoded. The control hardware delays reading register R4 unit cycle 5, thus introducing a 2-cycle stall unless operand forwarding is used. An alternative approach is to leave the task of detecting data dependence and dealing with them to the software. In this case, the compiler can introduce the two-cycle delay needed between instructions I_1 and I_2 by inserting NOP (No-operation) instructions, as follows:

I_1 : Mul R2, R3, R4
NOP
NOP
I_2 : Add R5, R4, R6

- If the responsibility for detecting such dependencies is left entirely to the software, the compiler must insert the NOP instructions to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware. A particular feature can be either implemented in hardware or left to the compiler. Leaving tasks such as inserting NOP instructions to the compiler leads to simpler hardware. Being aware of the need for a delay, the compiler can attempt to reorder instructions to perform useful tasks in the NOP slots, and thus achieve better performance. On the other hand, the insertion of NOP instructions leads to larger code size. Also, it is often the case that a given processor architecture has several hardware implementations, offering different features.

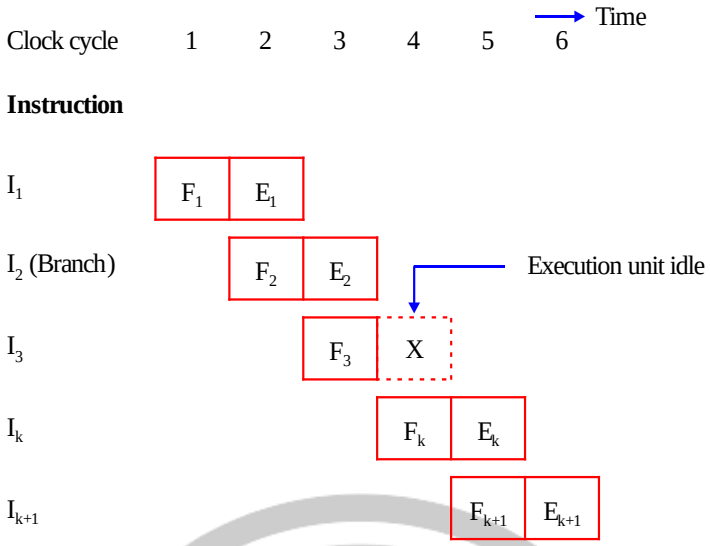
7.3 INSTRUCTION HAZARDS

- The purpose of the instruction fetch unit is to supply the execution units with a steady stream of instructions. Whenever this stream is interrupted, the pipeline stalls, as Figure illustrates for the case of cache miss. A branch instruction may also cause the pipeline to stall. We will now examine the effect of branch instructions and the techniques that can be used for mitigating their impact. We start with unconditional branches.

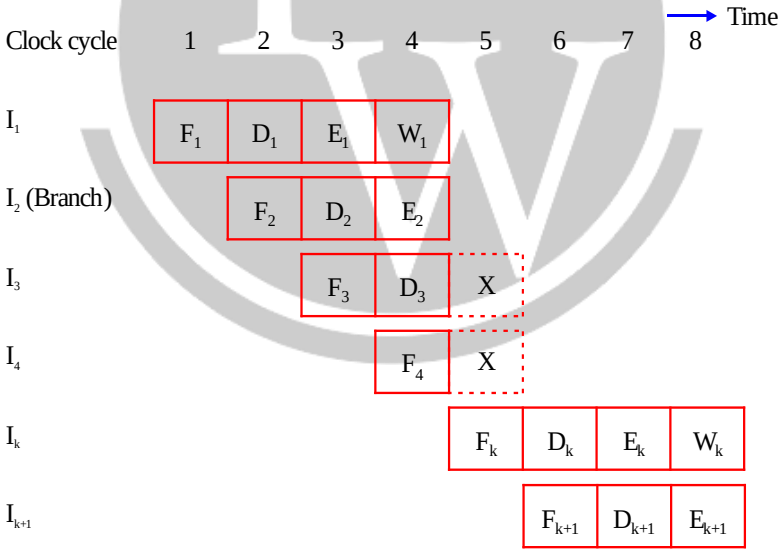
7.4 UNCONDITIONAL BRANCHES

- Figure shows a sequence of instructions being executed in a two-stage pipeline. Instructions I_1 to I_3 are stored at successive memory addresses, and I_2 is a branch instruction. Let the branch target be instruction I_h. In clock cycle 3, the fetch operation for instruction I_3 is in progress at the same time that the branch instruction is being decoded and the target address computed. In clock cycle 4, the processor must discard I_3, which has been incorrectly fetched, and fetch instruction I_h. In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.
- The time lost as a result of a branch instruction is often referred to as the *branch penalty*. In Figure, the branch penalty is one clock cycle. For a longer pipeline, the branch penalty may be higher. For example, Figure shows the effect of a branch

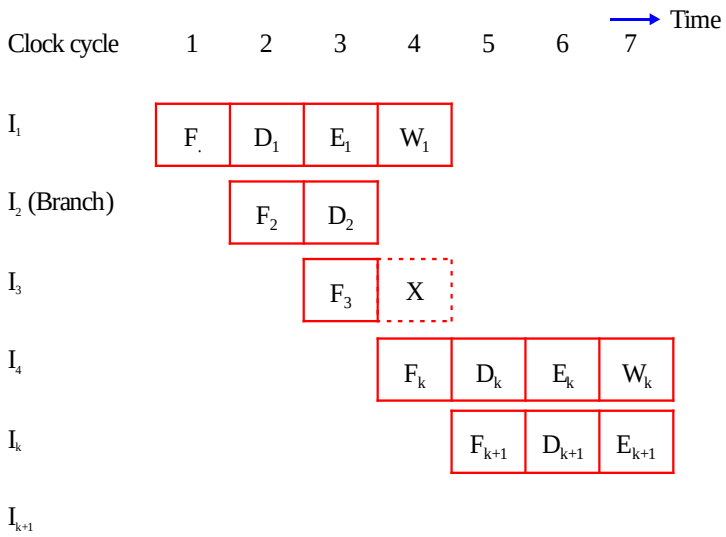
instruction on a four-stage pipeline. We have assumed that the branch address is computed in step E_2. Instructions I_3 and I_4 must be discarded, and the target instruction, I_h, is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.



- Reducing the branch penalty requires the branch address to be computed earlier in the pipeline. Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched. With this additional hardware, both of these tasks can be performed in step D_2, leading to the sequence of events shown in Figure. In this case, the branch penalty is only one clock cycle.



(a) Branch address computed in Execute stage



(b) Branch address computed in Decode stage

7.4.1 Instruction Queue and Prefetching

- Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles. To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue. Typically, the instruction queue can store several instructions. A separate unit, which we call the *dispatch unit*, takes instructions from the front of the queue and

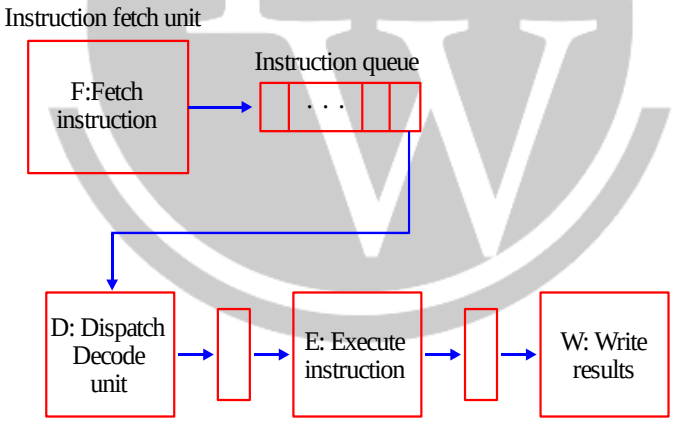


Fig: Use of an instruction queue in the hardware organization of fig (b)

- Sends them to the execution unit. This leads to the organization shown in Figure 8.10. The dispatch unit also performs the decoding function.
- To be effective, the fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions. It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions. When the pipeline stalls because of a data hazard, for example, the dispatch unit is not able to issue instructions from the instruction queue. However, the fetch unit continues to fetch instructions and add them to the queue. Conversely, if there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue.

7.5 CONDITIONAL BRANCHES AND BRANCH PREDICTION

- A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction. The decision to branch cannot be made until the execution of that instruction has been completed.
- Branch instructions occur frequently. In fact, they represent about 20 percent of the dynamic instruction count of most programs. (The dynamic count is the number of instruction executions, taking into account the fact that some program instructions are executed many times because of loops.) Because of the branch penalty, this large percentage would reduce the gain in performance expected from pipelining. Fortunately, branch instructions can be handled in several ways to reduce their negative impact on the rate of execution of instructions.

7.5.1 Delayed Branch

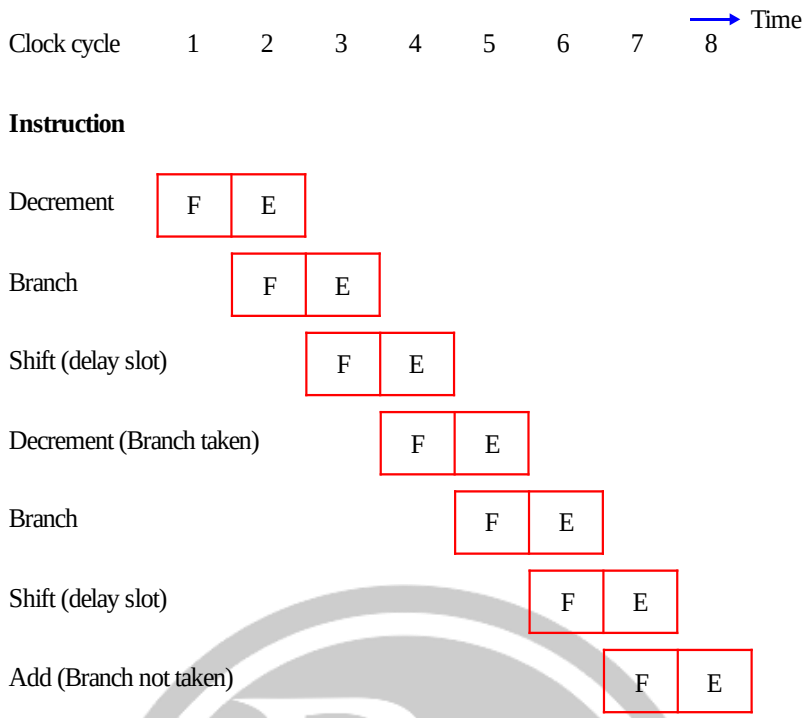
- In Figure the processor fetches instruction I_3 before it determines whether the current instruction, I_2, is a branch instruction. When execution of I_2 is completed and a branch is to be made, the processor must discard I_3 and fetch the instruction at the branch target. The location following a branch instruction is called a *branch delay slot.* There may be more than one branch delay slot, depending on the time it takes to execute a branch instruction. For example, there are two branch delay slots in Figure and one delay slot in Figure. The instructions in the delay slots are always fetched and atleast partially executed before the branch decision is made and the branch target address is computed.
- A technique called *delayed branching* can minimize the penalty incurred as a result of conditional branch instructions. The idea is simple. The instructions in the delay slots are always fetched. Therefore, we would like to arrange for them to be fully executed whether or not the branch is taken. The objective is to be able to place useful instructions in these slots. If no useful instructions can be placed in the delay slots, these slots must be filled with NOP instructions.

LOOP	Shift left	R1
	Decrement	R2
	Branch=0	LOOP
NEXT	Add	R1.R3

(a) Original program loop

LOOP	Decrement	R2
	Branch=0	LOOP
	Shift left	R1
NEXT	Add	R1.R3

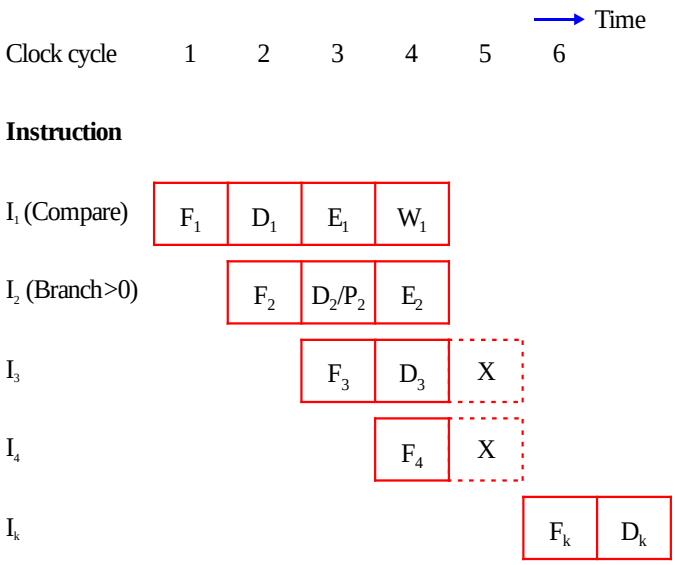
(b) Reordered instructions



- The effectiveness of the delayed branch approach depends on how often it is possible to reorder instructions as in Figure. Experimental data collected from many programs indicate that sophisticated compilation techniques can use one branch delay slot in as many as 85 percent of the cases. For a processor with two branch delay slots, the compiler attempts o find two instructions preceding the branch instruction that it can move into the delay slots without introducing a logical error. The changes of finding two such instructions are considerably less than the changes of finding one. Thus, if increasing the number of pipeline stages involves an increase in the number of branch delay slots, the potential gain in performance may not be fully realized.

7.5.2 Branch Prediction

- Another technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken. The simplest form of branch prediction is to assume that the branch will not take place and to continue to fetch instructions in sequential address order.
- Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis. Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.
- Hence, care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed by executed. If the branch decision indicates otherwise, the instructions and all their associated data in the execution units must be purged, and the correct instructions fetched and executed.



7.5.3 Dynamic Branch Prediction

- The objective of branch prediction algorithms is to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded. In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decision every time that instruction is executed.
- In its simplest form, the execution history used in predicting the outcome of a given branch instruction is the result of the most recent execution of that instruction. The processor assumes that the nest time the instruction is executed, the result is likely to be the same.

7.6 INFLUENCE ON INSTRUCTION SETS

- We have seen that some instructions are much better suited to pipeline execution than others. For example, instruction side effects can lead to undesirable data dependencies. In this section, we examine the relationship between pipelined execution and machine instruction features. We discuss two key aspects of machine instructions – addressing modes and condition code flags.

7.6.1 Condition Codes

- In many processors, the condition code flags are stored in the processor status register. They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution. An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur. In doing so, the compiler must ensure that reordering does not cause a change in the outcome of a computation. The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions.

- Consider the sequence of instructions in Figure, and assume that the execution of the Compare and Branch = 0 instructions proceeds as in Figure. The branch decision takes place in step E_2 rather than D_2 because it must await the result of the Compare instruction. The execution time of the Branch instruction can be reduced

```
Add          R1, R2
Compare      R3, R4
Branch = 0   . . .
```

(a) A program fragment

```
Compare      R3, R4
Add          R1, R2
Branch = 0   . . .
```

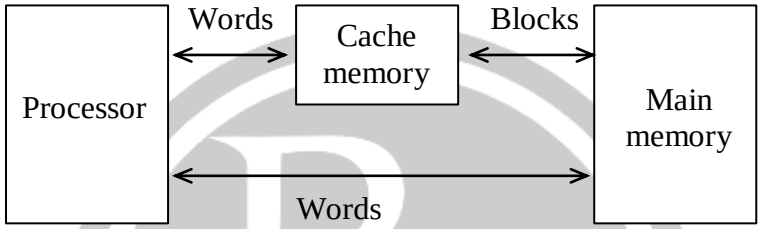
(b) Instructions reordered

❐❐❐

8 CACHE MEMORY

8.1. Introduction

- The speed of the main memory is very low in comparison with the speed of modern processors. For good performance an efficient solution is to use a fast cache memory.



  - It is the smallest and fastest memory component in the hierarchy.
  - By placing active portions of the program and data in a fast small memory, the average access time can be reduced.
  - The effectiveness of cache mechanism is based on principle called **"locality of reference"**
  - The Locality of reference states that, "The references to memory at any given interval of time tend to be confined within a few localized areas in memory. i.e. many instructions in localized areas of the program are executed repeatedly. It can be
    1. **Temporal:** It means that recently executed instruction is likely to be executed again very soon.
    2. **Spatial:** It means that instructions in close proximity to Cl recently executed instruction,

       Example: loops, nested loops, procedure calls. Etc.

- The performance of cache is measured using Hit ratio

$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{Total CPU references}} \times 100$$

$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{no.of hits} + \text{no.of misses}} \times 100$$

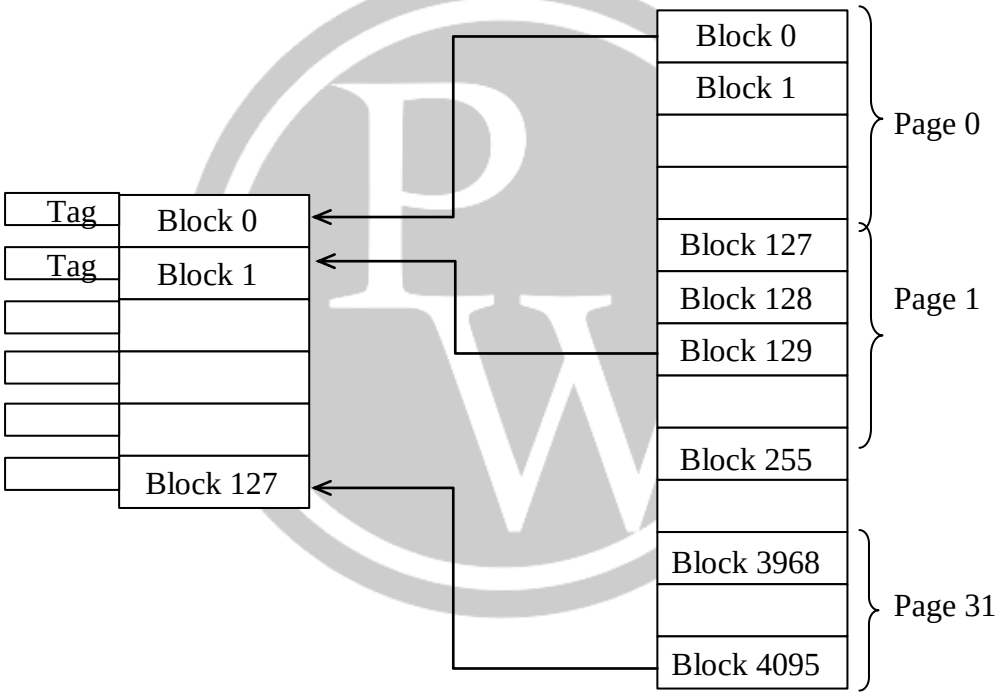
- The performance of cache can be analysed with the following characteristics.
  (1) Cache size (Small in KB's)
  (2) Block or line size
  (3) No. of levels of cache
  (4) Cache mapping process.
  (5) Cache replacement algorithm
  (6) Cache updating scheme.

8.2. Cache Mapping Functions

- Consider a cache consisting of 128 blocks of 16 words cache, assume the main memory is addressable by a 16-bit address. The main memory is addressable by a 16 –bit address. The main memory has 64K words viewed as 4K blocks of 16 words each.
- The various mapping functions are
  (i) Direct mapping
  (ii) Associative mapping
  (iii) Set – Associative mapping

8.2.1 Direct Mapping

- The simplest way to determine cache locations in which to store memory blocks.
- Each block from the main memory has only one possible location in cache.
- Block j of the main memory maps to block j mod n. Where n is the no. of blocks in the cache.



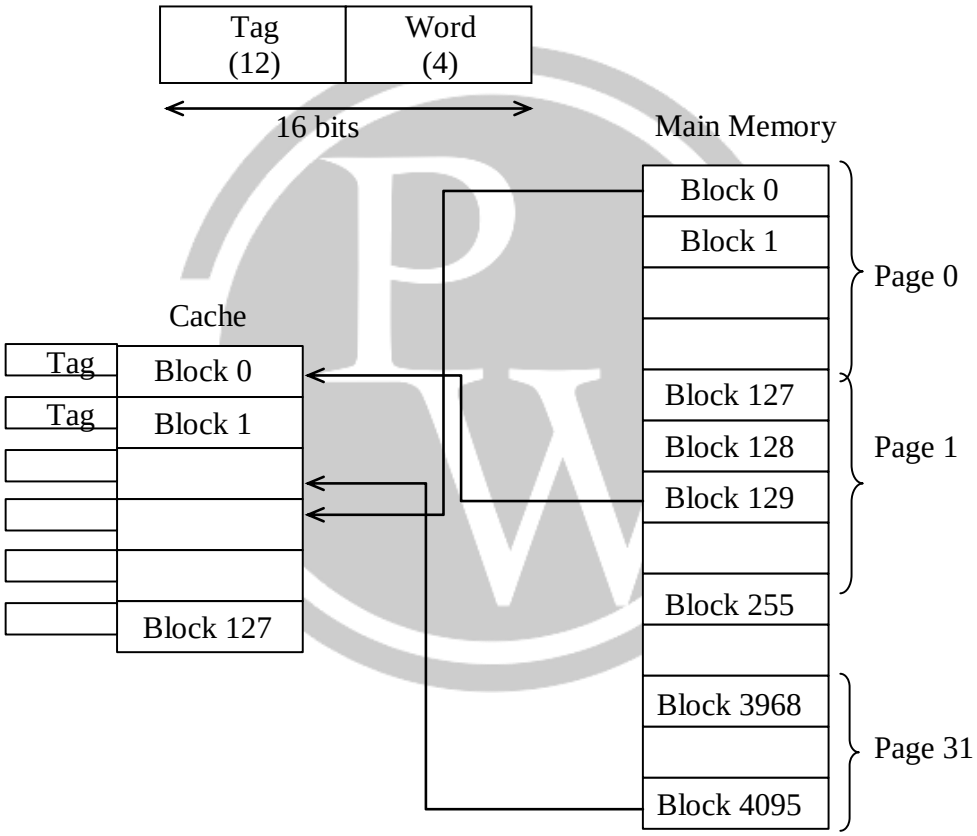
- If the processor needs to access same memory locations from two different pages of the main memory frequently, hen miss & will be more.
- Easy to implement but hit ratio is less.
- To implement this the address is divided into Three fields.
- Example :

Tag	Block	Word
5	7	4

- As execution proceeds, the higher order tag bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of cache. If there is no match, then miss will occur.
- Requires only one comparison.

8.2.2 Associative Mapping

- In which a main memory block can be placed into any cache block position.
- It gives complete freedom in choosing the cache location in which to place the memory block. Thus the space in the cache can be used effectively.
- A new block that has to be brought into the cache has to replace an existing block if the cache is full.
- The cost is high, because of need to search all tag patterns to determine whether a given block is in the cache, i.e. comparison circuit is more complex.
- Requires maximum of n comparisons. Where n is the number of cache blocks.
- The no. of tag comparators = no. of cache blocks.
- The address is divided into 2 fields.

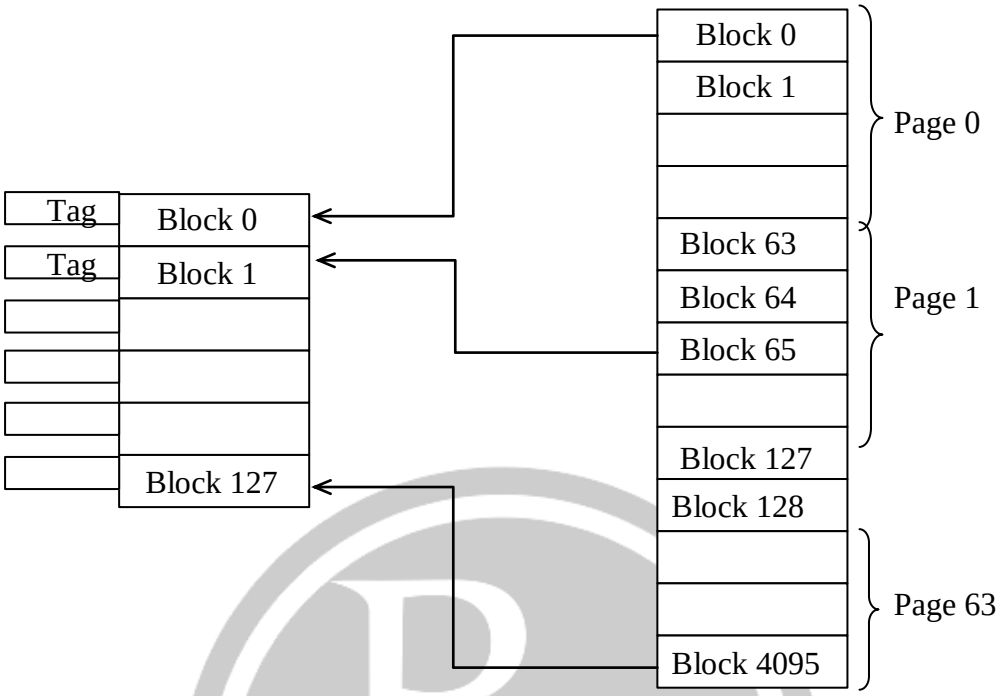


- We need an algorithm to select the block to be replaced.

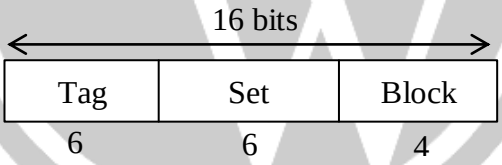
8.2.3 Set – Associative Mapping

- For efficiency, a combination of direct and associative mapping techniques can be used.
- Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. i.e. contains several groups of directed mapped caches in parallel.
- The contention problem of the direct mapped method is eased by having a few choices for block placement.

- The hardware cost is reduced for comparing tags unlike associative mapping.
- A block can occupy either of blocks in the set.



- A cache that has K blocks per set, then it is referred as K-way set-associative cache.
- The address is divided into 3- fields.



- Number of tag comparisons is equal to number of blocks in the set.

8.3. Cache Replacement Policy

- In direct mapped cache, the position of each block is predetermined hence no replacement strategy exists.
- In Associative and set – Associative caches there exists some flexibility, when a new block is to be brought into the cache and all positions that it may occupy are full. Hence replacement strategy exists.
- The replacement policies are aimed for reducing the penalty (miss) to feature references.

The various block replacement policies are

(1) FIFO: The block which enters first will be the candidate for replacement. i.e. Assumes that, since it has spent long time in cache all the references to it are slightly exhausted. Implemented using queue data structure.

(2) LRV: The block in the se t which has been in the cache longest with no references to it is selected for the replacement. (Least recently used).

(3) LFU: The block in the set which has the fewest references is selected for replacement. (Least frequently used).3

(4) Random: No specific criteria for replacement of any block. The existing blocks are replaced randomly.

8.4. Cache Updation Policy

- The two cache updation schemes employed are

➢ **Write – through:** The simplest and commonly used approach. Updation of cache and main memory are done simultaneously.

  - Main memory contains the same data as the cache. Which is important for DMA transfers.

➢ **Write–back:** In this method, only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.

  - i.e. updation of main memory is postponed until the cache block selected for replacement.

8.5. Cache Memory & Arrays

An array is a homogeneous collection of data items stored in contiguous memory locations either in row major order or column major order.

Example:

$$\begin{bmatrix} 3 & 1 & 2 \\ 5 & 6 & 9 \\ 8 & 4 & 7 \end{bmatrix}$$

Row major:

00	01	02	10	11	12	20	21	22
3	1	2	5	6	9	8	4	7

Col major:

00	10	20	01	11	21	02	12	22
3	5	8	1	6	4	2	9	7

Q.1. Consider an array is A[100] and each element occupies 4 – words, 32 – word cache is used and it is divided into 8 – word blocks (a) what is the hit ratio for the following instruction.

(a) For (i = 0 ; i < 100 ; i ++)
A [i] = A [i] + 10 ;

	R	W
A[0]	M	H
A[1]	H	H
A[2]	M	H
A[3]	H	H

A(0)
A(1)
A(2)
A(3)

Ans.

(b) How many times block '0' is modified in the cache memory.

Ans. 0, 8, 16, - - - - - 80, 88, 96. ⇒ 13 times.

(c) How many times block '0' is replaced.

Ans. 12 times.

❑❑❑

9 MAIN MEMORY

9.1. Introduction

- The main memory is the central storage unit in a computer system. The principal technology used for the main memory is based on semiconductor integrated circuits. Main memory is made up of RAM and ROM chips. Different types of integrated circuit memories are given in Table 1.
- Integrated circuit RAM chips are available in two possible operating modes, static and dynamic. The static RAM consists of interval flip flops that store the binary information. The stored information remains valid as long as power is applied to the unit i.e. ROM is volatile. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.

Memory	Access	Volatile	Features
Read/Write	Random or serial	Yes	Data can be read or written with equal case. Used whenever data is needed fast and often, and is frequently changed.
Read-only (ROM)	Random	No	(a) Data is stored permanently by a masking step during manufacture. (b) Used whenever data is not to be changed as in fixed tables, constants or computer instruction sets.
Programmable Read-only (PROM)	Random	No	(a) Can be programmed after IC is manufactured. (b) Data is written by opening fusible metal links or P-N junction's wit high currents. (c) Can only be written into once. Same use as a ROM, but is cheaper in small quantities.
Re-programmable read-only	Random	No	Are ROMs that can be written into many times. Are really "read-mostly" rather than "read-only" memories Two main types: (a) Those in which writing is by voltage application and erasing (all data at once) by exposing chip to ultra-violet radiation through a window on the IC (b) Those in which reading and writing is electrical. Data remains stored even with no power applied through the use of MNOS transistors. Used where frequent or at least more than one change in a program is required as in debugging a program.
Content-addressable (CAM)	Random	Yes	This extracts all data in an address when a part of the contents of that address match a specified number. Used in associative memories to obtain stored data related in some way to the input data.

Charge-coupled device (CCD)	Serial	Yes	(a) Digital input is converted to charge and stepped through a shift register. (b) Requires refresh circuitry to present data loss.
Programmable logic array (PLA)	Random	No	(a) Is a memory structure that is mask or field (FPLA) programmed as is a ROM. However, it implements logic function. (b) Inputs are functions of the input variables and the logic operation stored in the address. (c) Is more flexible than random (hard-wired separate IC) circuits because PLAs and FPLAs are programmable.

Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. The Table 2 shows the summary of various types of integrated circuit memories.

(a) Types of Memory Access	
Random access	Any address can be accessed at random, that is, without going through other address first. Data retrieval time is relatively fixed.
Serial access	Typified by shift register or charge-coupled-device (CCD) memories. Data retrieval is serial in a fixed order. All data ahead of required data must be read first.
(b) Static verses Dynamic Storage	
Static storage	Data is stored in flip-flops or other memory cells in which the data does not deteriorate with time.
Dynamic storage	Data is stored in leaky capacitors so that refresh circuitry is required to prevent data loss.

9.2. Memory Interface

A computer uses memory capacity of 512 bytes of RAM and 512 bytes of ROM. The IC sizes of RAM and ROM are 128 x 8 and 512 × 8 bits respectively.

(a) Find the number of RAM ICs.

(b) Find the number of ROM ICs.

(c) Give the memory map of the system

(d) Mention the size of decoder.

(e) Give the diagram of memory connection to the CPU.

Solution:

(a) The number of RAM ICs $= \frac{\text{Total'RAM size}}{\text{RAM IC size}}$

$= \frac{512\times 8}{128\times 8}$

$= 4$

(b) The number of ROM ICs $= \frac{\text{Total 'ROM size}}{\text{ROM IC size}}$

$= \frac{512 \times 8}{512 \times 8}$

$= 1$

(c) The memory map of this system is given by Table. 3

Component	Hexadecimal address	Address bus											
		10	9		8	7	6	5		4	3	2	1
RAM 1	0000–007F	0	0		0	×	×	×		×	×	×	×
RAM 2	0030–00FF	0	0		1	×	×	×		×	×	×	×
RAM 3	0100–017F	0	1		0	×	×	×		×	×	×	×
RAM 4	0180–01FF	0	1		1	×	×	×		×	×	×	×
ROM	0200–03FF	1	×		×	×	×	×		×	×	×	×

(d) 2 × 4 decoder

Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 × 4 decoder. The outputs of decoder are given to the CSI inputs in each RAM chip. Thus when address lines 8 and 9 are equal to 00, the first RAM chip is selected when 01, the second RAM chip is selected and so on.

(e) The connection of memory chips to the CPU is shown in Fig. RAM and ROM chips are connected to a CPU through the data and address buses. The low order lines in the address bus selected the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.

❐❐❐

10 SECONDARY STORAGE

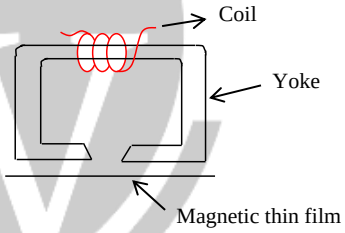
10.1 Introduction

- The cost per bit of stored information is high in c semi-conductor memories (main memory). This limits the use of these memories for large storage. Large storage requirements of most of computers are economically fulfilled by magnetic disks, magnetic tapes and optical disk memories. These memories are referred as secondary storage devices.

10.2 Magnetic Disks

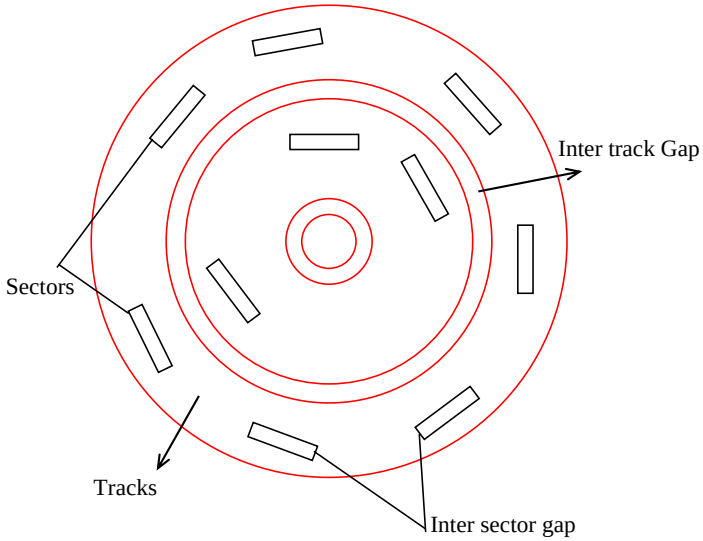
- A magnetic disk is a thin, circular metal plate. It is coated with a thin magnetic film usually on both sides.
- Digital information is stored on the magnetic disk by magnetizing the magnetic surface in a particular direction.
- Magnetic disks are semi random access memories.

The head consists of a magnetic yoke and the magnetizing coil. The Digital information can be stored on the film by applying current pulses of suitable polarity.



10.2.1. Data organization & Formatting

- The data on the disk is organized in a concentration set of rings. These concentric set of rings are called tracks.
  - ✓ Each track has same width as head and adjacent tracks are separated by gaps.
  - ✓ Each track is divided into manageable units called sectors.
  - ✓ Each sector stores a block of data which can be transferred to or from the disk.
  - ✓ The universal size of sector is 512 bytes.
  - ✓ The R/W head is capable of reading a sector from disk and writing a sector to disk.
  - ✓ Placing R/W head on desired track is random access and reading concerned sector is serial. Hence disks are semi random access devices.



10.2.2. Physical Characteristics

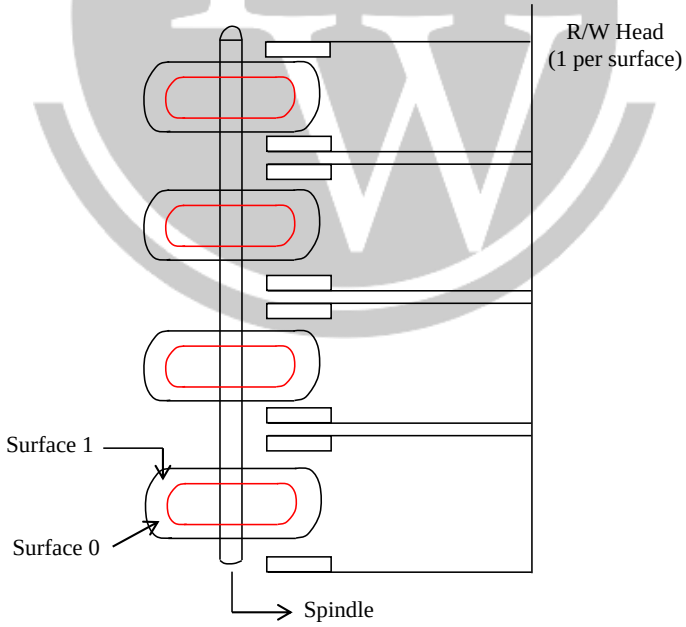
(i) Head Motion - Fixed head (one per track)
Movable head (one per surface)

(ii) Disk portability - Non removable or Removable

(iii) Sides - Single sided, Double – sided.

(iv) Disk/surface - Single surface, Multiple – surfaces

(v) Head Mechanism - Contact, fixed Gap, Aerodynamic Gap.



A vertical set of all of the tracks with the same number on each surface of a diskette or hard disk is called "cylinder."

(vi) Platters - Single platter,
Multiple platters – some disk
Accommodate multiple platters
Stacked vertically a fraction of an inch apart.

(1) If all the tracks are having constant capacity then the disk moves with "constant angular capacity" and "recording density" varying from track to track.

(2) If each track is having variable capacity then the disk is moving with "constant linear velocity"

- Placing the control information in the sector is called as formatting.
- The disk controller is the hardware interface to control the operation of a disk drive. Used to control more than one drive. The major functions are seek, read, write and error checking.
- The start and end of each sector is determined by the control data stored in the sector.
- Disk performance parameters:

Seek Time: It is the time required to more the disk arm to the required track.

Rotational Delay: The time taken for the beginning of sector to reach. (latency)

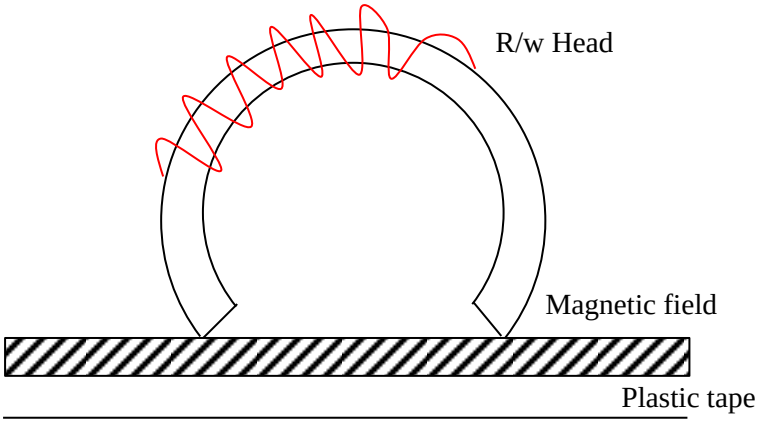
Access time: The sum of seek time and the rotational delay.

- The average time to disk access is
  $t_{avg} = t_{seektime} + t_{rotational\ delay} + t_{data\ transfer} + t_{over\ head}$
- Maximum Recording Density (P) = No. of Bytes/cm
- Data transfer rate (D) = Number of Bytes/Sec.

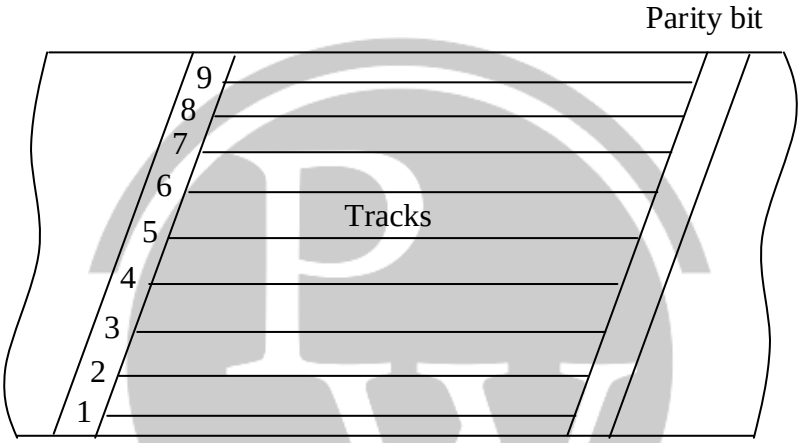
10.3 Secondary Storage Devices

10.3.1. Magnetic Tapes

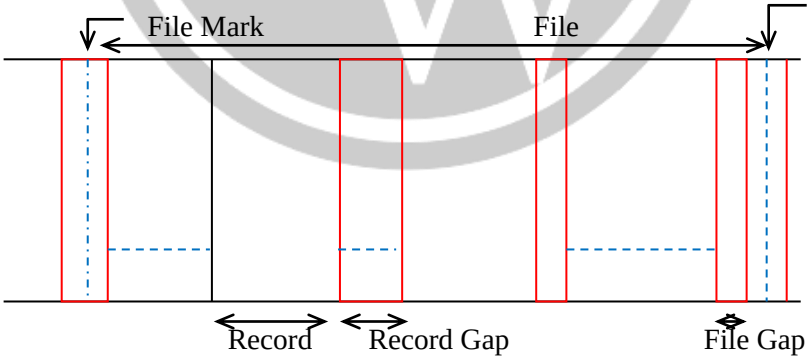
- Magnetic tape is one of the most popular storage medium for large data that are sequentially accessed and processed.
- The tape is formed by depositing magnetic film on a very thin and ½ or ¼ inch wide plastic tape.
- Usually, iron oxide is used as a magnetizing material.
- The information is recorded on the tape with the help of read/write head. It magnetizes or non magnetizes tiny invisible spots (1's & 0's).
- Usually 7 or a bits are recorded (a character) in parallel across the width of the tape, perpendicular to the direction of motion. A separate R/Wo head is provided for each bit position on the tape.



- Data on the tape are organized in the form of records separated by gaps.
- A set of related records constitutes a file.



- The data on a 9^{th} track (a) tape is stored in parallel consists of data byte and a parity bit.



- Data transfer takes place when the tape is moving at constant velocity relative to a read/write head.
- Hence the maximum data transfer rate depends largely on the storage density along the tape and speed of the tape.
- The file mark is used as header or identifier.
- The information on the magnetic tape is organized into blocks. These blocks have a fixed length and separated by gap between them.
- If the block length is B_L and inter block gap length is G_L, then the utilization factor of the tape (u) is given by

$$u = \frac{B_L}{B_L + G_L}$$

- The unit of data transfer is a record.
- The tape is moving with a linear velocity and Read/write head is constant.
- Let L is the length of the tape,
  - N is the number of parallel tracks,
  - P is the constant recording density.

(i) Capacity of the tape

$$C = L \times N \times P$$

(ii) Let V is the linear velocity of the tape, then the data transfer rate

$$D = V * N * P$$

✓ With utilization factor, the effective data transfer rate is

$$D_{eff} = D * \frac{B_L}{B_L + G_L}$$

✓ Due to inter block gap and time needed to start and stop and tape between accesses, the effective data transfer rate d_{eff} is actually less than the maximum rate d.

$$d_{eff} = \frac{t_D \times d}{t_D + t_G + t_{ss}}$$

t_D = time to scan a data block

t_G = time to scan the inter block gap.

T_{ss} = time to start and stop the tape.

.

❐❐❐