1 Basics of Computer Design

1.1 COMPUTER ARCHITECTURE

- It deals with the system attributes which are visible to the programmer, and these attributes have a direct impact on the execution of the program.
- Few examples of such attributes are I/O mechanism, bits required to represent data types, instruction sets, etc.

Computer organisation:

- In computer organisation we deal with the operational unit of the hardware system and its interconnections
- Examples of organisational attributes are hardware details that are hidden from the programmer.

Von–Neumann architecture:

- It is based on the concept of the same memory holding both data and instructions.
- In this architecture, we have common addresses and data buses for both CPU and Memory.
- John Von Neumann proposed this architecture in 1945.
- In this architecture, single instruction takes 2 clock cycles.
- In this architecture, CPU cannot access instructions and read/write at the same interval of time.
- This architecture is less expensive.
- It is used for personal computers and small-scale computers.

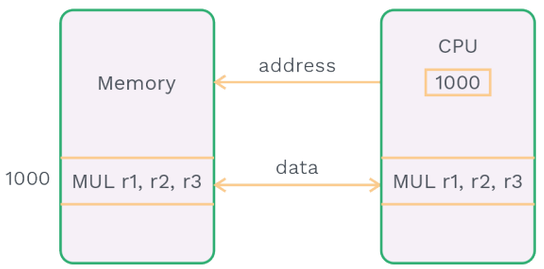


Fig. 1.1 Von–Neumann Architecture

Harvard architecture:

- It is based on the concept of distinct memory models for instructions and data.
- It has two sets of address/data buses shared by CPU and memory.
- It allows two simultaneous memory fetches.
- It allows CPU to fetch the instruction, and it can perform both read and write operations at the same time.
- This architecture is more expensive than Von Neumann's architecture.
- It is used in microcontrollers and digital signal processing (DSP) devices

such as sonar, radar, etc.

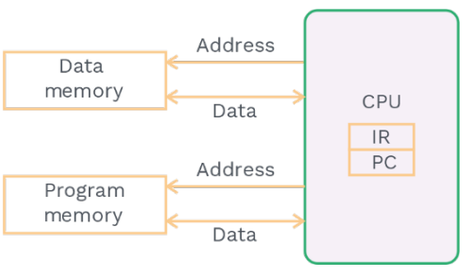


Fig. 1.2 Harvard Architecture

PRACTICE QUESTIONS

Q1 **Among the given options, choose the correct one :**

I) Von neumann architecture shares common memory for instructions and data.

II) Harvard architecture has separate physical memories for instructions and data.

a) Only I is true **b) Only II is true**

c) Both I and II are true **d) Neither I nor II is true**

Sol: **c)** Harvard architecture has two different sets of memory, one set for storing the data elements and another set for storing the code. While Von–Neumann architecture shares common memory for data and instruction.

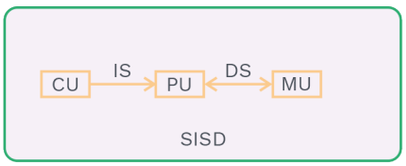
Flynn's classification:

- It is a way of organising multiple processors.
- It was first introduced by Flynn and is the most common approach for categorizing the systems with parallel processing capabilities.
- This classification has four categories of computer systems.

- Single Instruction, Single Data (SISD) stream
- Single Instruction, Multiple Data (SIMD) stream
- Multiple Instruction, Single Data (MISD) Stream
- Multiple Instruction, Multiple Data (MIMD) stream

1) Single Instruction, Single Data (SISD) stream:

- In SISD, one instruction runs on one processor on the data stored on single memory.
- Uniprocessor comes into this category.



CU : Control unit
IS : Instruction stream
PU : Processing unit
DS : Data stream
MU : Memory unit

Fig. 1.3 SISD Stream

- Machine instructions are executed in sequential order.
- Systems based on this architecture are known as sequential computers.
- Most of the conventional computers are based on this model.
- Instructions and the data must be stored in primary memory for processing.
- The rate of transferring information internally determines the speed of processing in SISD.

2) Single Instruction, Multiple Data (SIMD) stream:

- In this architecture, a single machine instruction controls simultaneous execution of a number of processing elements on lock step basis.
- Each processing element has an associated data memory for the execution of instruction on different sets of data by different processors.
- Examples of this model are vectors and arrays.

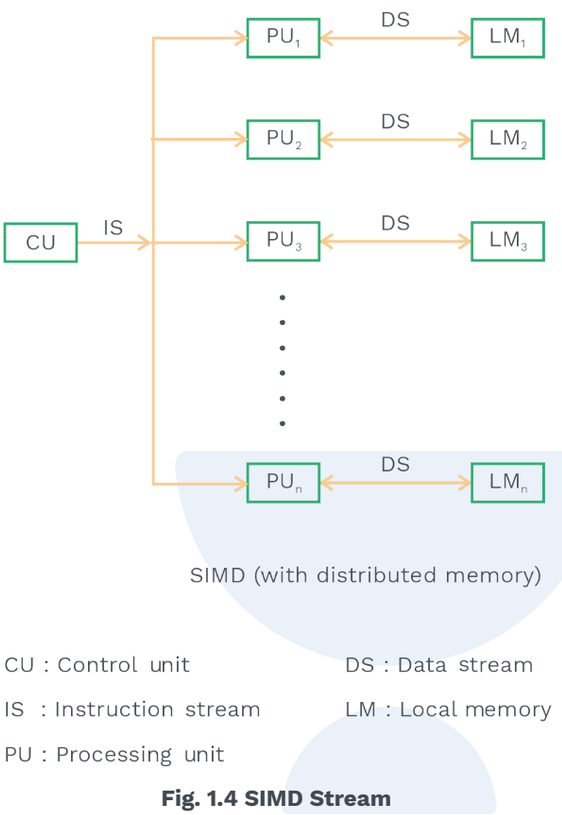


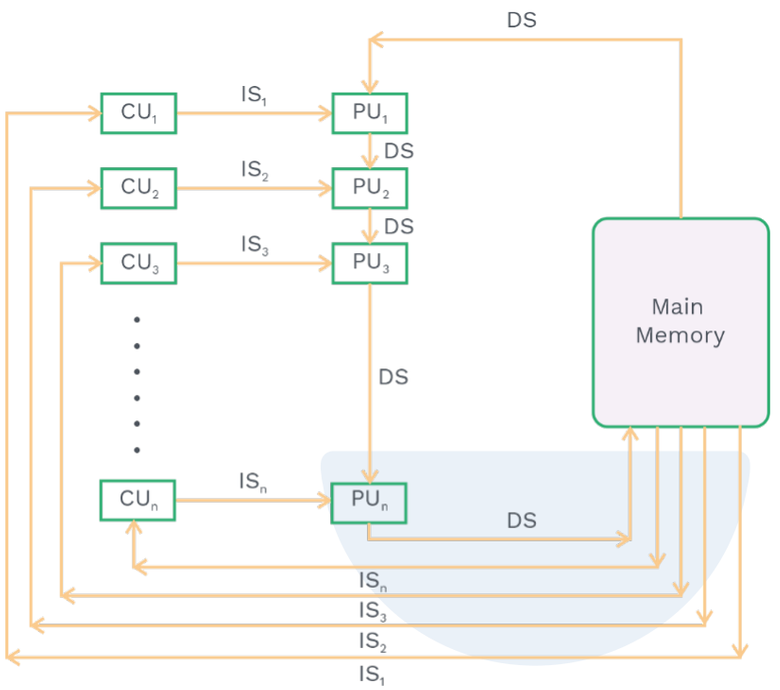
Fig. 1.4 SIMD Stream

- Scientific computing is based on SIMD architecture.
- Cray's vector processing machine is based on this architecture.
- Example of SIMD:

If we want to subtract 10 from each element of array A, it can be done by SIMD model.

3) Multiple Instruction, Single Data (MISD) stream:

- In this architecture, a sequence of data is transmitted to a set of processors, in which every processor executes a different instruction sequence.
- This architecture is not commercially implemented.



MISD architecture

CU : Control unit PU : Processing unit

IS : Instruction set DS : Data set

Fig. 1.5 MISD Stream

- Machines built using the MISD architecture are not very useful in most of the applications.
- Example of MISD is:
  A = tan(a),
  B = sec(a), a: Variable (Single data stream)
  C = cosec(a)
- Here data stream is the same, but the instruction stream is different, i.e. the CPU will first calculate the value of tan(a), second the value of sec(a) and third the value of cosec(a).

4) Multiple Instruction, Multiple Data (MIMD) stream:

- In this architecture, different instructions of different data set execute in different processors.

- Different processors take programs and data from common shared memory, and processors communicate with each other through that memory.
- Example of a shared memory MIMD system is a symmetric multiprocessor (SMP).
- In symmetric multiprocessors, multiple processors are capable of sharing single memory with the help of buses.

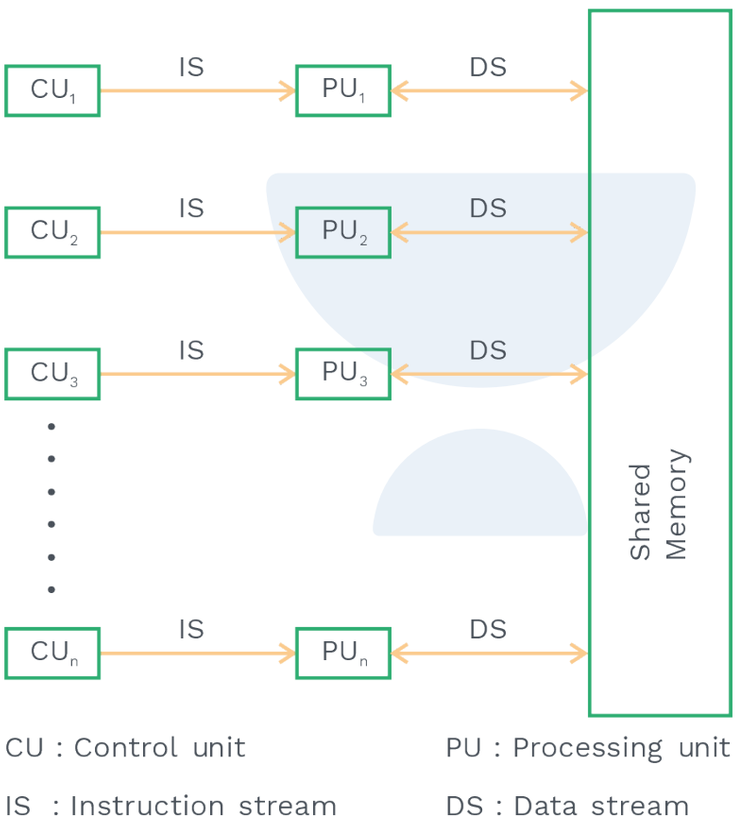
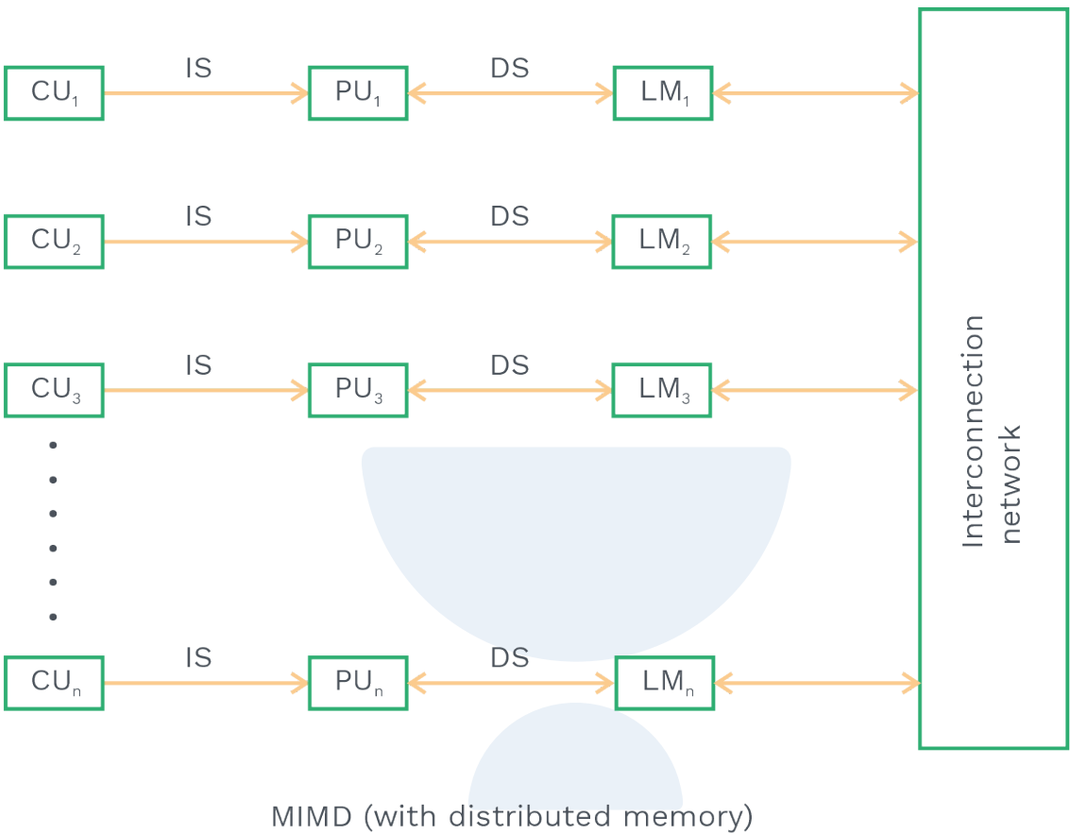


Fig. 1.6 MIMD Stream



MIMD (with distributed memory)

LM : Local memory
CU : Control unit
PU : Processing unit
IS : Instruction set
DS : Data set

Fig. 1.7 MIMD Stream with Distributed Memory

- A more recent development in shared memory MIMD is non–uniform memory access (NUMA) organisation.
- A collection of independent uniprocessors or SMPs maybe interconnected to form a cluster.

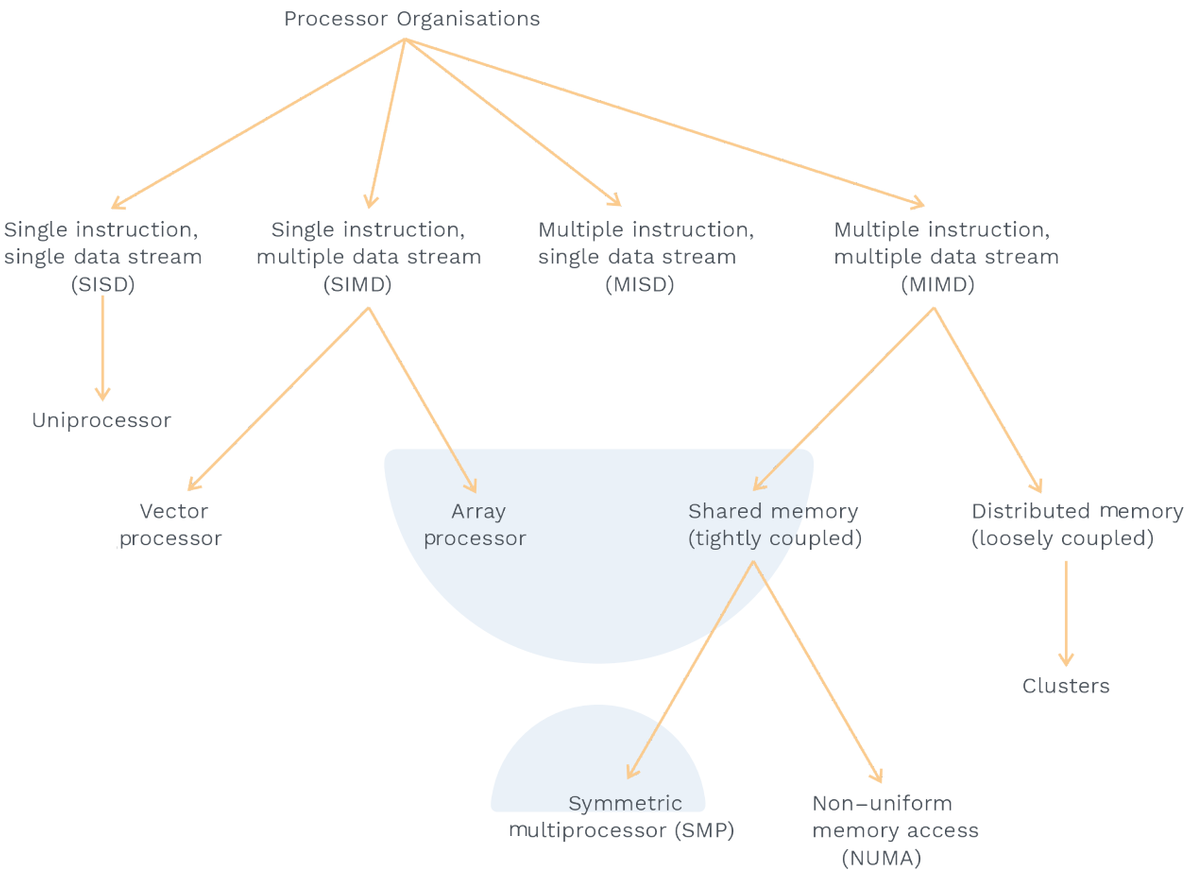


Fig. 1.8 A Taxonomy of Parallel Processor Architectures.

Symmetric multiprocessors:

- With the increase in demand for performance-enhancing and with the dropping in cost of microprocessors, designers have introduced the concept of an SMP organisation.
- Characteristics of an SMP (Symmetric Multiprocessor) are detailed below.
- There are two or more similar processors of comparable capability.
- Memory access time for all the processors will be the same as they are sharing the main memory , as well as input/output, are interconnected.
- All the processors share access to input/output devices through the same channel or through different channels, which provides paths to the same devices.
- All processors can perform the same function (so the term symmetric is used).

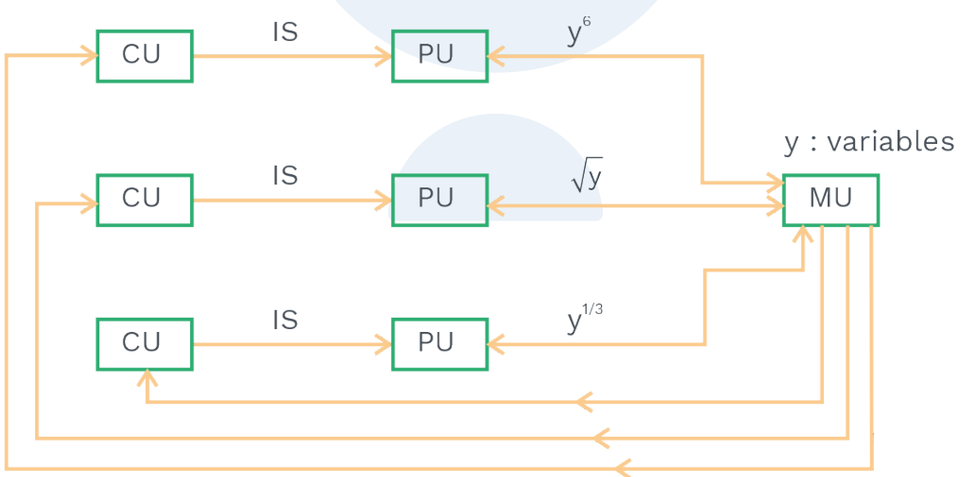
PRACTICE QUESTIONS

Q2 **Suppose we want to compute the value of $y^6, \sqrt{y}, y^{1/3}$ using the given value of y. Which of the following among Flynn's CPU classifications will be best suited for the above calculation?**

a) (SISD) stream.
b) (SIMD) stream.
c) (MISD) stream.
d) (MIMD) stream.

Sol: **c)**

MISD is useful in such operations where different computation is done on the same data set.



CU : Control unit IS : Instruction stream

PU : Processor unit MU : Memory unit

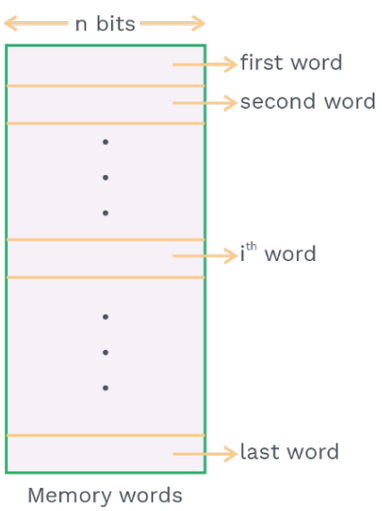
Q3 **The vector and arrays belong to which class of Flynn's classification.**
a) (SIMD) stream.
b) (SISD) stream.
c) (MISD) stream.
d) (MIMD) stream.

Sol: **a)**

The vector and arrays belong to SIMD (Single instruction and multiple data streams).
For example: If we want to decrement all the elements of an array by some constant value, it is usually performed by SIMD type of classification.

Memory locations and addresses:

- The number operands, character operands and instructions are stored in the memory.
- In memory, there are many storage cells, each of which stores the information in binary form.
- A single bit may not be sufficient to provide ample information. So, data is rarely represented by a single bit.
- A pattern of bits is capable of providing valuable information.
- Group of n bits accessed is referred to as word, and n is known as the word length.

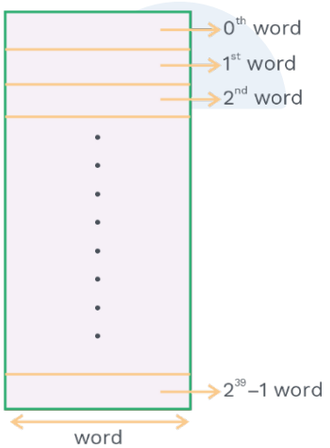


- Most modern computers have word lengths ranging from 16 to 64 bits.
- Word length of 32 bits shows that a single word can store a 32–bit 2's complement number, each occupying 8 bits.
- Size of machine instruction will be equal to one or more words.
- For accessing the information from memory(word or byte), we require unique addresses for each location.
- It is most widely used that numbers from 0 to 2^K–1 are used for representing K values.
- 2^k address space means that there are 2^k address cells, and each cell is represented with k bits number.

PRACTICE QUESTIONS

Q4 **If there are 512 G locations (words) in the memory, then find the number of bits to represent each word uniquely.**

Sol: Total number of locations (words) = 512 G



$1G = 2^{30}$

$512\ G = 512 \times 2^{30} = 2^9 \times 2^{30} = 2^{39}$

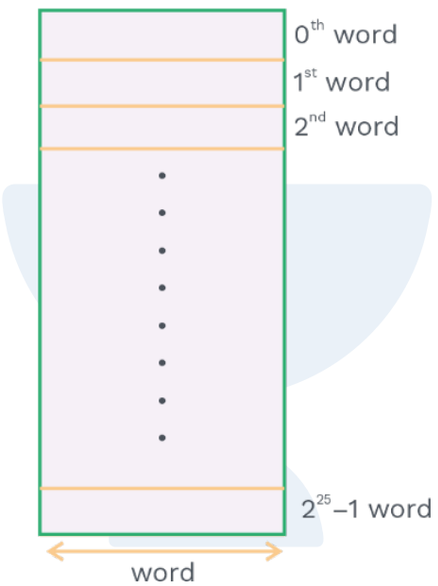
The number of bits required to represent each word uniquely:

$= \lceil \log_2 2^{39} \rceil = 39$ bits.

Q5 **If there are 25 bits available to represent each word of memory, then how many maximum number of locations (words) can be addressed uniquely?**

Sol: n = number of bits = 25

Maximum number of words that can be addressed uniquely



$= 2^{25}$ words $= 2^5 \times 2^{20}$ words

$= 32 \times 1M$ words $= 32$ M words

Q6 **If memory is byte addressable (1 word = 1 Byte) and the total size of memory is 128 KB. Calculate the number of bits required to represent each word and each byte uniquely.**

Sol: Total memory size = 128KB

Number of bits required to represent each byte = $\lceil \log_2 2^7 \times 2^{10} \rceil = \lceil \log_2 2^{17} \rceil$

= 17 bits

Since 1 word = 1 Byte

Number of bits required to represent each word = 17 bits

Q7 **If the size of main memory is 512 GB. The size of the word is 16 bits. Find the number of bits required to represent each bit uniquely if the memory is word addressable.**

Sol: Size of word = 16 bits

8 bits = 1 Byte

16 bits = 2 Byte

1 word = 2 Byte

Total number of words = $\frac{512\,GB}{2B}$ = 256 G words

1 word = 16 bits

256 G words = 256 G × 16 bits = $2^8 \times 2^{30} \times 2^4 = 2^{42}$ bits

So, number of bits required to represent each bit uniquely inside memory

= $\lceil \log_2 2^{42} \rceil$ = 42 bits

Q8 **Memory size = 128 Gb**
word size = 16 bits
Give the memory capacity in words, bytes and bits and also give the number of bits required for addressing, respectively.

Sol:

Memory capacity	Addressability
Bit = 128 Gb	$2^7 \times 2^{30} = 2^{37}$ $\boxed{n = 37}$
Byte = 16 GB	$2^4 \times 2^{30} = 2^{34}$ $\boxed{n = 34}$
Word = 8 GW	$2^3 \times 2^{30} = 2^{33}$ $\boxed{n = 33}$

n = number of bits required for addressing respectively.

RAM chips and organization:

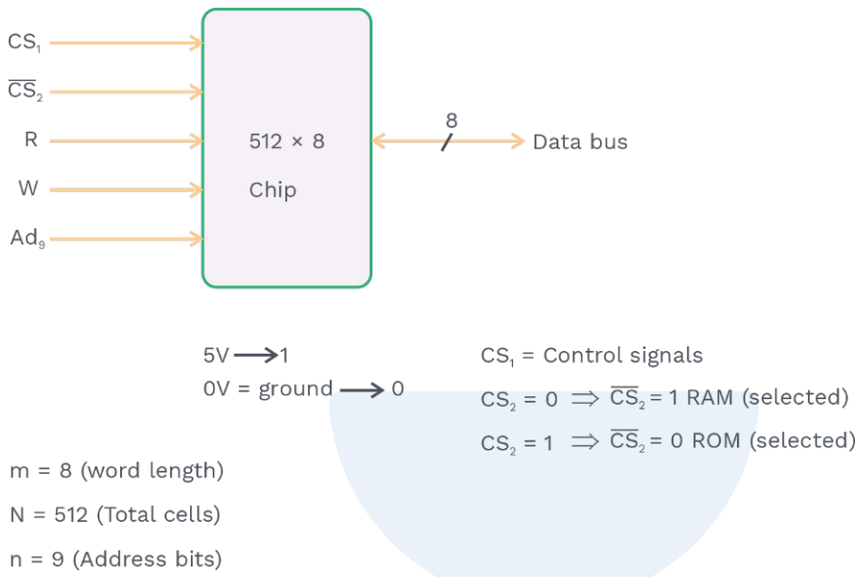


Fig. 1.9

Building 1024 B using 256 B chips:

$$\frac{1024\,B}{256\,B} = 4$$

N = 1024 W (1W = 1B)

n = $\lceil \log_2 1024 \rceil$ = 10-bit address

00 – select first RAM chip
01 – select second RAM chip
10 – select third RAM chip
11 – select fourth RAM chip
R = Read signal (it is active high)
W = Write signal (it is active high)
Ad = Address (8 bits) = Each chip has 256 words.
CS_1 = Chip select, it will be done by decoder as shown above. It is active high.
$\overline{CS_2}$ = It is active low and is connected with G (ground), which is always 0. Hence 0 input is given to CS_2, so $\overline{CS_2}$ =1. Hence all the chips will be working as RAM and not ROM.

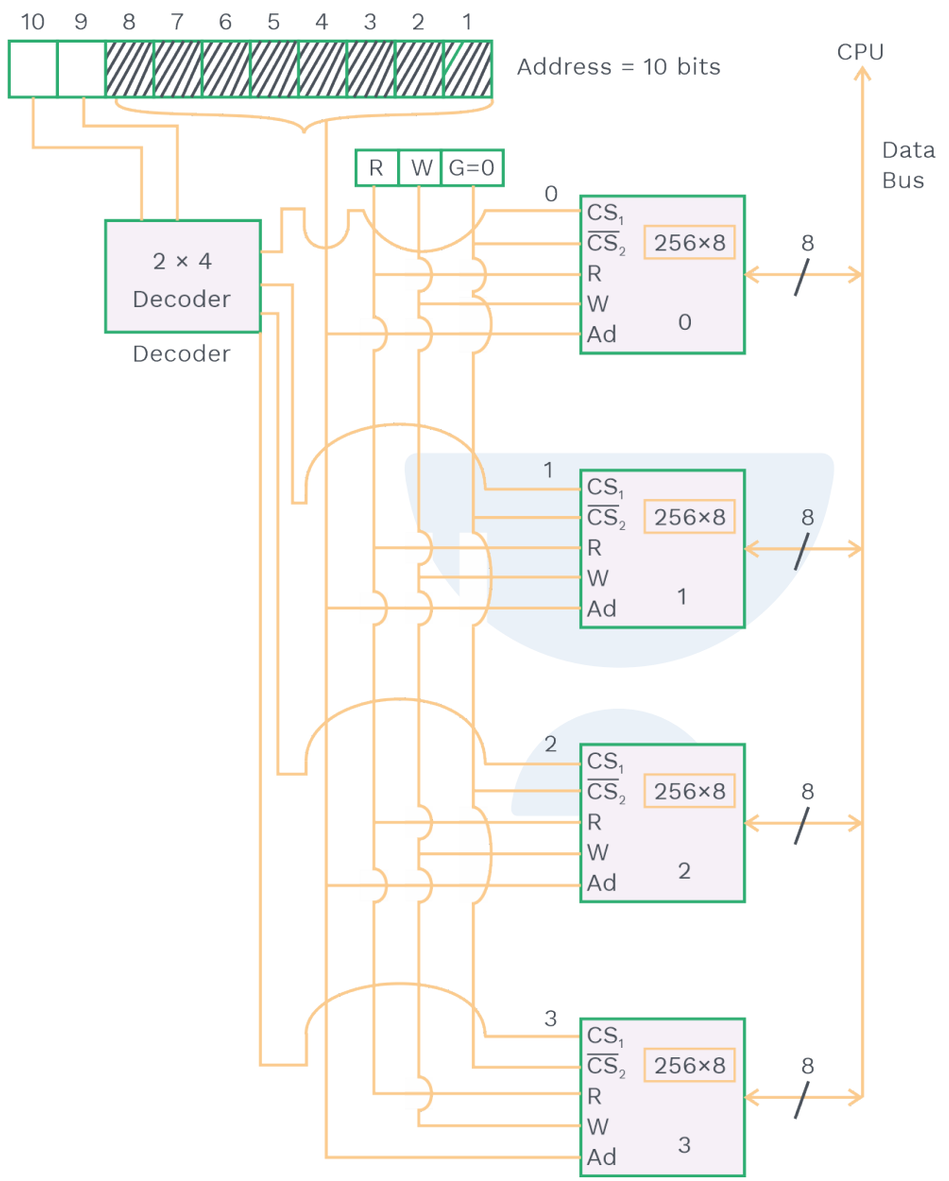


Fig. 1.10

For example, Address=1000101011, R=1, W=0, G=0.
It means CPU wants to read the data from memory. If the two MSB bits are 10, they will select the 2nd RAM chip. G = 0, which means the chip will act like RAM. 8 LSB bits will signify the word number (43rd word from 2nd chip). 43rd word (8 bit) will be kept on the data bus, and CPU will read it.

PRACTICE QUESTIONS

Q9 **How many 256 MB RAM chips are needed to build 2GB RAM?**

Sol: Number of RAM chips required $= \frac{2\,GB}{256\,MB} = \frac{2 \times 2^{30}}{2^8 \times 2^{20}} = \frac{2^{31}}{2^{28}}$

$= 2^3 = \boxed{8\,chips}$

Q10 **How many 128 KB RAM chips are required to build 256 MB of RAM?**

Sol: Total capacity RAM = 256 MB

Size of RAM chip available = 128 KB

Number of RAM chips required

$= \frac{256\,MB}{128\,KB} = \frac{2^8 \times 2^{20}}{2^7 \times 2^{10}}$

$= 2^{11}$

= 2048 chips

Building larger RAMs using smaller RAM chips:

⇒ There are broadly two types of strategies:

1) Vertical increasing (constant word size):

64 MB RAM using 128 × 8 chips

Each word = 8 bits

(Here 1W = 8 bits = 1 Byte)

Number of chips required

$= \frac{64\,MB}{128\,W} = \frac{64\,MW}{128\,W} = \frac{2^{26}}{2^7} = 2^{19}$

$= 2^{19}$ chips

= 512 K chips

Decoder size = $\boxed{19 \times 2^{19}}$

Address = 19 + 7 = 26 bit

19 → To decoder

7 → To each chip from 2^7

PRACTICE QUESTIONS

Q11 **How many 64K × 8 RAM chips are required to build 4 MB of RAM chip? Also, calculate the size of the decoder required. Assume memory is Byte addressable.**

Sol: Number of chips required

$$= \frac{\text{Total Required Size of RAM}}{\text{Available RAM Size}}$$

$$= \frac{4\,\text{MB}}{64\,\text{K} \times 8} \quad \text{(Here, we assume 1 word = 1 Byte)}$$

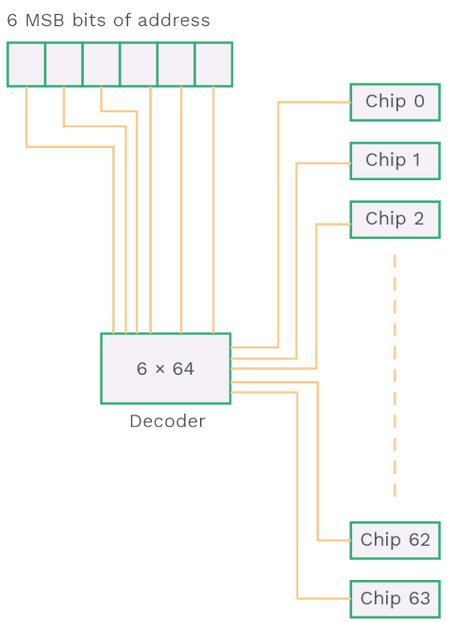
$$= \frac{4\,\text{MB}}{64\,\text{KW}} = \frac{4\,\text{MW}}{64\,\text{KW}} = \frac{2^2 \times 2^{20}}{2^6 \times 2^{10}}$$

$$= = \frac{2^{22}}{2^{16}} = 2^6 = 64\,\text{chips are used}$$

Here we are using the concept of vertical increasing, i.e., word size will remain constant.

Hence $6 \times 2^6 =$ $\boxed{6 \times 64}$ size decoder will be required.

* At a single time, only 1 chip will be activated by the decoder, the rest all will remain deactivated.



Each chip has an enable line which activates the chip. We need to select one of 64 chips so a $\boxed{6 \times 64}$ size decoder will be required for this purpose.

Q12 **Calculate the size of the decoder and the number of chips required to build a RAM of 32 Gb from 256 K × 8 RAM chip. Also, give the diagrammatic representation of the decoder and RAM chips. Assume Byte addressable memory.**

Sol: Given, the size of RAM chips = 256 K × 8

Required size of larger RAM chip = 32 Gb

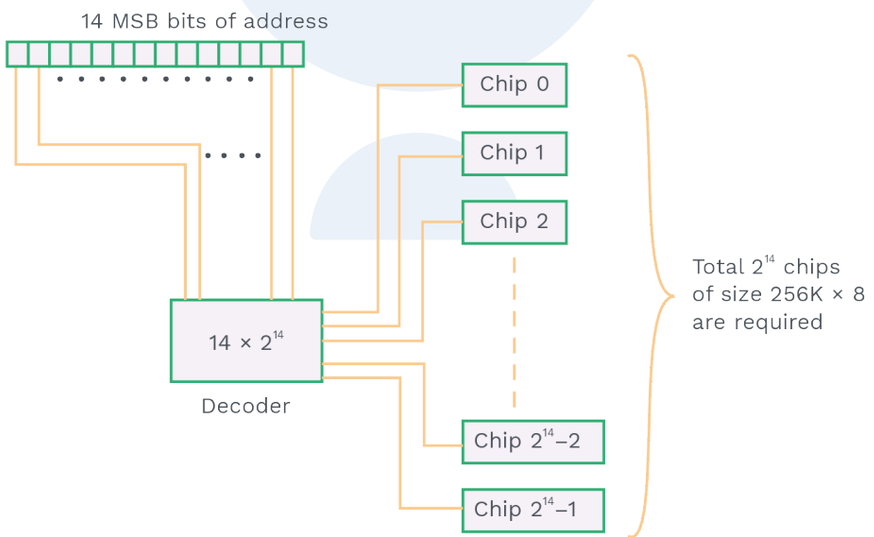
Number of smaller chips required

$$= \frac{32\,\text{Gb}}{256\,\text{K} \times 8} = \frac{2^5 \times 2^{30}}{2^8 \times 2^{10} \times 2^3} = \frac{2^{35}}{2^{21}}$$

$= 2^{14}$ chips

Here, again we are using the concept of vertical increasing, i.e., word size will remain constant.

So $\boxed{14 \times 2^{14}}$ size decoder will be required.



- Each chip has an enable line which activates the chip. We need to select one of 2^{14} chips. Hence a decoder of size $\boxed{14 \times 2^{14}}$ will be required for this purpose.
- As we are using a decoder, at a single time, only 1 chip will be activated by the decoder, the rest will remain deactivated.

2) Horizontal Increasing (change in word size):

- Here, the word size will be changed

For example: we want to build a 256 B RAM using 256 × 1 RAM.

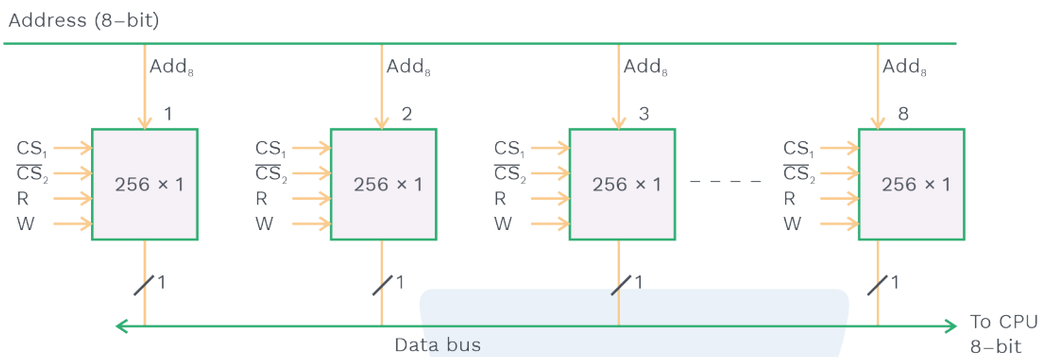


Fig. 1.11

- Here, all the chips are activated simultaneously.
- We are assuming that our required chip has a word size of 1 byte and the given chip has a word size of 1 bit.

Required chip size = 256 B (1W = 1B)
Given chip size = 256 × 1 (1W = 1bit)
Number of chips (given) required

$$= \frac{256\,B}{256 \times 1} = 8\,\text{chips}$$

- Each smaller chip has 256 memory locations of 1 bit each. But we are required to give 1 Byte of data (8 bits) to the CPU through the data bus.
- Each chip will give 1 bit each, so we have to concatenate all the bits from 8 chips, i.e., 1 bit each from 8 chips and concatenated 8 bits (1 Byte) will be given to the data bus.
- All 8 chips will be activated simultaneously to serve the purpose of sending 8 bits (1 Byte).
- As there are 256 locations in each smaller chip, an address line of 8 bits is required and sent to each smaller chip.
- Each chip has CS_1, $\overline{CS_2}$, R(Read), W(Write) pins which do their functions the same as in vertical increasing case.

PRACTICE QUESTIONS

Q13 **We are given a 128 × 1 RAM chip. Designers want to build a bigger RAM chip of size 128 B. How many RAM chips of size 128 × 1 will be required to get the larger RAM chip? Also, give a diagrammatical representation of arrangements of chips.**

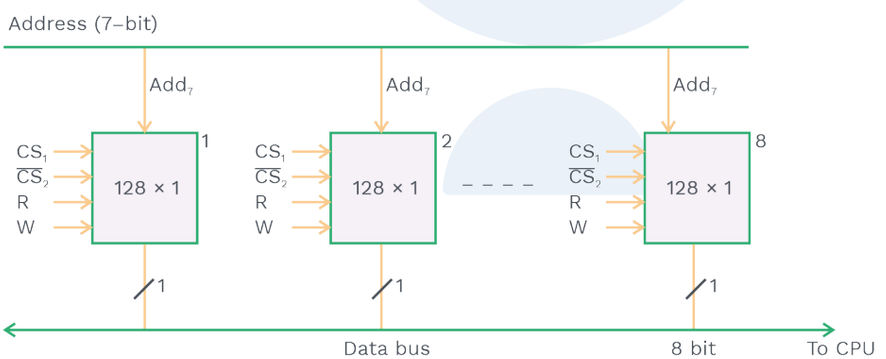
Sol: Required RAM size = 128 B

Given RAM size = 128 × 1

In given RAM size, 1 word = 1 Bit

In required RAM chip, 1 word = 8 Bit

$$\text{Number of chips required} = \frac{128\,\text{B}}{128 \times 1} = 8 \text{ chips}$$



- Here again, we are using horizontal increasing, where word size will be changed.
- All 8 bits (1 bit from each chip) will be merged together and sent to the data bus.
- Each smaller chip has 128 locations (cells), so a 7-bit address is required to uniquely address each location of the respective chips.
- Address pin Add_7 will be activated simultaneously for all the chips so that each chip can give 1 bit each.

3) Vertical + Horizontal increasing:

- In the real world, we end up using vertical + horizontal increasing to serve the purpose of practical needs.

For example: We are required to build 64 KB RAM using 512 × 1 RAM chips.

- Here 1 word = 1 Byte (in our required chip)

Number of smaller chips required

$$= \frac{64\,KB}{512 \times 1} = \frac{2^6 \times 2^{10} \times 2^3}{2^9}$$

$$= \frac{2^{19}}{2^9} = 2^{10} = 1024 \text{ chips}$$

- Total, we need 1024 chips, but here these chips will be arranged both in a horizontal and vertical manner.
- Given smaller chip gives only 1 bit, but we are required 8 bits (1 Byte) in larger RAM chip, so here horizontal increasing is required.

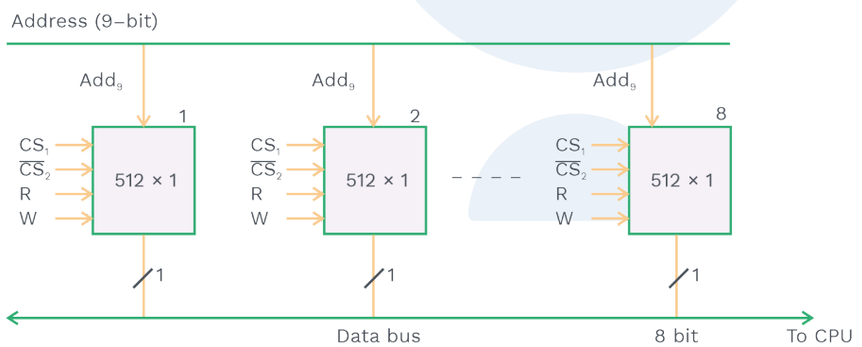


Fig. 1.12 Horizontal Part

- Now, this is only 1 chunk of horizontal arrangement of chips; it will give 8 bits in total by merging, as we have seen earlier in the horizontal case.
- Total, we require 1024 (2^{10}) chips, but out of these, 8 chips per horizontal block are required to get the desired word length.
- $\frac{1024}{8} = \frac{2^{10}}{2^3} = \boxed{2^7}$, this will be the total number of horizontal block sets which will be arranged vertically.

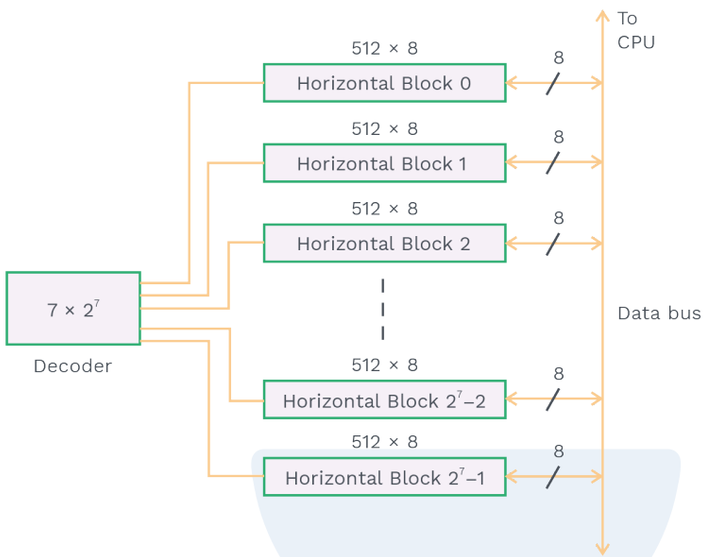


Fig. 1.13 Vertical Arrangement of Horizontal Blocks

- Here, the address will be divided into 2 parts, the first one is the decoder and the second one is the address bits for each horizontal chunk of chips.

$$\underbrace{7 \text{ bits}}_{\text{To decoder}} + \underbrace{9 \text{ bits}}_{\text{To Horizontal Block}} = 16 \text{ bits (Address)}$$

- In total, we have 16 bits, which will give a total RAM size of 2^{16} = 64 KB, where 1 word = 1 Byte.
- As there are a total of 2^7 horizontal blocks, we need to select one of 2^7 blocks; it will be done by the decoder of size $\boxed{7 \times 2^7}$.
- After selecting one of 2^7 blocks, the horizontal arrangement comes into play; a 9 bit address will be given to each chip inside the selected horizontal block.
- As there are 512 locations, each of size 1 bit, one location of each chip will be selected, so total 8 bits will be put on the data bus.
- In this way, this hybridisation (vertical + horizontal) concept will be implemented, which is used in most practical applications.

PRACTICE QUESTIONS

Q14 **We are given 64 K × 1 chips of RAM. Our aim is to build a larger RAM chip of size 512 KB. In larger RAM chip 1 word = 1 Byte. Calculate the total number of chips required and the size of the decoder. Also, give the diagrammatic arrangements of chips.**

Sol: Here 1 word = 1 Byte (in a larger RAM chip)

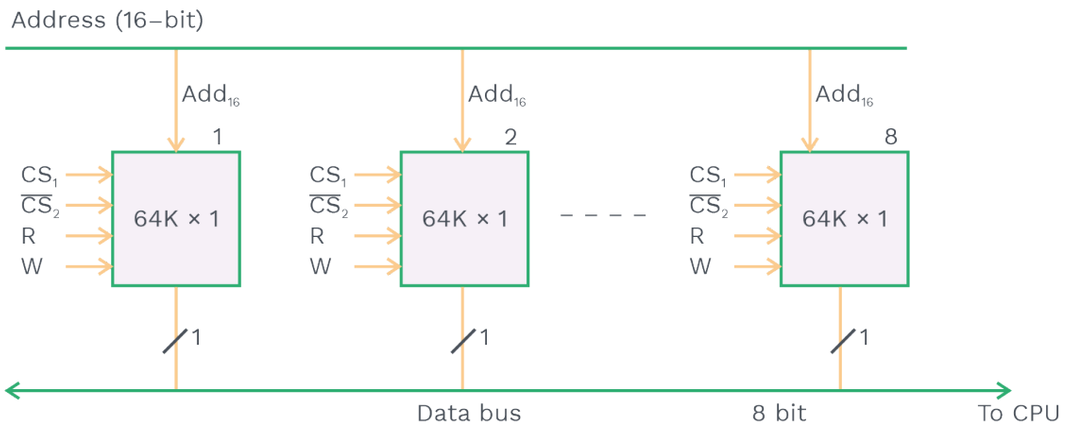
Number of smaller chips required

$$= \frac{512\,KB}{64\,K \times 1}$$

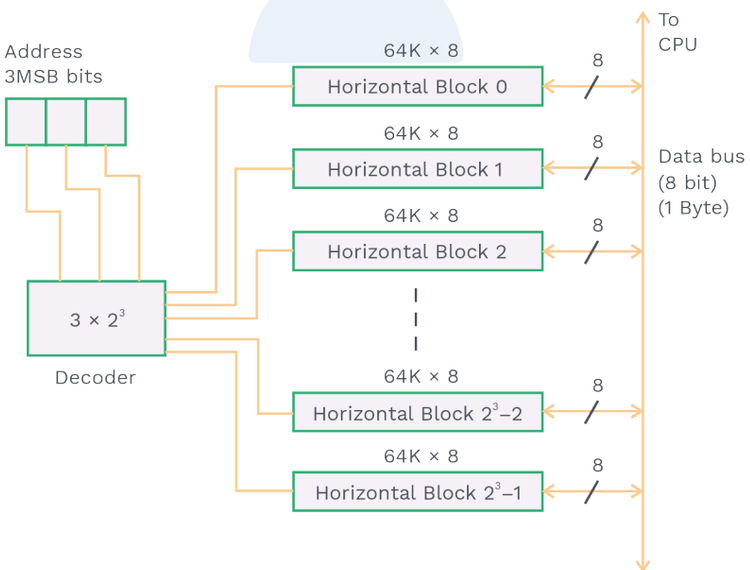
$$= \frac{2^9 \times 2^{10} \times 2^3}{2^6 \times 2^{10}}$$

$$= \frac{2^{22}}{2^{16}} = 2^6 = 64 \text{ chips}$$

- Total, we need 64 chips, but these chips will be arranged both in the horizontal and vertical ways.
- Smaller chips give only 1 bit, but we are required to get 8 bits (1 Byte) in our target chip. So, both horizontal and vertical increasing is required.
- In total, we require 64 (2^6) chips, but out of these 64 chips, 8 chips will be required per horizontal block.
- $\frac{64}{8} = 8 = \boxed{2^3}$ horizontal blocks will be placed vertically, one after the other and each horizontal block will contain 8 chips.
- As there are 2^3 horizontal blocks, we require a decoder of size $\boxed{3 \times 8}$ to select one of these horizontal blocks.
- Address will be divided into 2 parts; the first part goes to the decoder and the second part goes to each smaller chip inside the horizontal block.
- 3 bits = To decoder
  16 bits = To each chip in the horizontal block.



- This is only 1 chunk (Block) of horizontal arrangement of chips; it will give 8 bits (1 Byte) in total by concatenating each bit.
- There is a total of 2^3 horizontal blocks; out of these, we need to select one; this task will be accomplished by the decoder of size $\boxed{3 \times 2^3}$.
- After selection of one of 2^3 blocks, horizontally arranged chips will be given an address of 16 bits (to each chip).



Diagrammatical representation
of vertical + horizontal scheme

- Total, we have 19 bits, so a total of 2^{19} = 512 KB memory will be available.

Previous Years' Question

How many 32 K × 1 RAM chips are needed to provide a memory capacity of 256 K bytes?

a) 8 **b)** 32 **c)** 64 **d)** 128

Sol: c) **(GATE CS-2009)**

Previous Years' Question

A RAM chip has a capacity of 1024 words of eight bits each (1K × 8). The number of 2 × 4 decoders with enable input line needed to construct a 16K × 16 RAM from 1K × 8 RAM is

a) 4 **b) 5** **c)** 6 **d)** 7

Sol: b) **(GATE CSE-2013)**

Big Endian and Little Endian:

- Byte addresses can be allocated in two different ways across the words.
- The first one is the big endian in which the most significant byte(leftmost side) of the word is stored at lower byte address.
- Another one is the little endian, which has opposite ordering. In this the least significant byte(rightmost side)of the word is stored at lower byte address.
- The bit present in the position of the highest and lowest powers of 2 are used as the most significant and the least significant bits, respectively.
- Given below is the diagrammatic representation of big–endian.

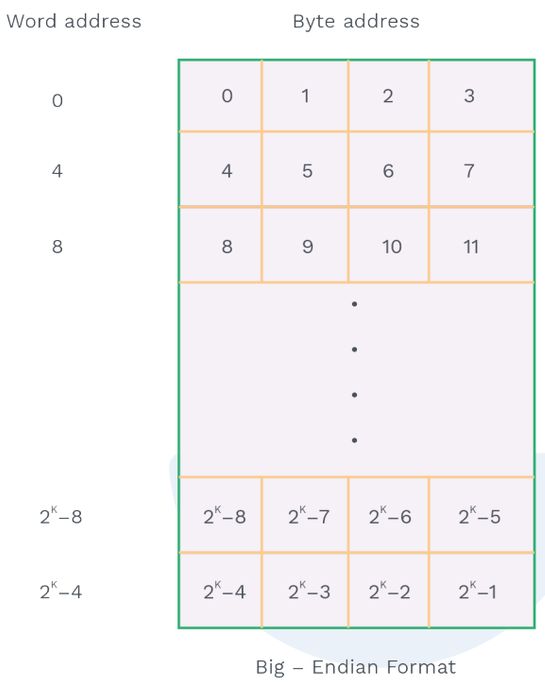


Fig. 1.14

- Both the methods little endian and big endian is used in many commercial machines.
- The byte addresses 0,4,8,... are considered as the address of the words in succession. These byte addresses are used when a specific word is read or written from memory.
- Given below is a signed integer.

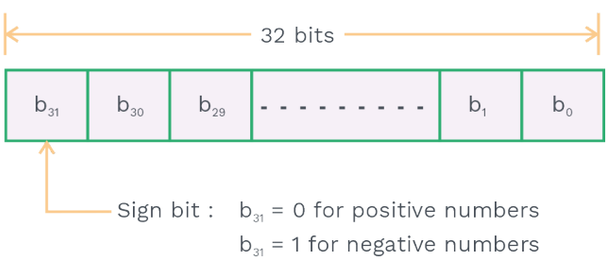


Fig. 1.15

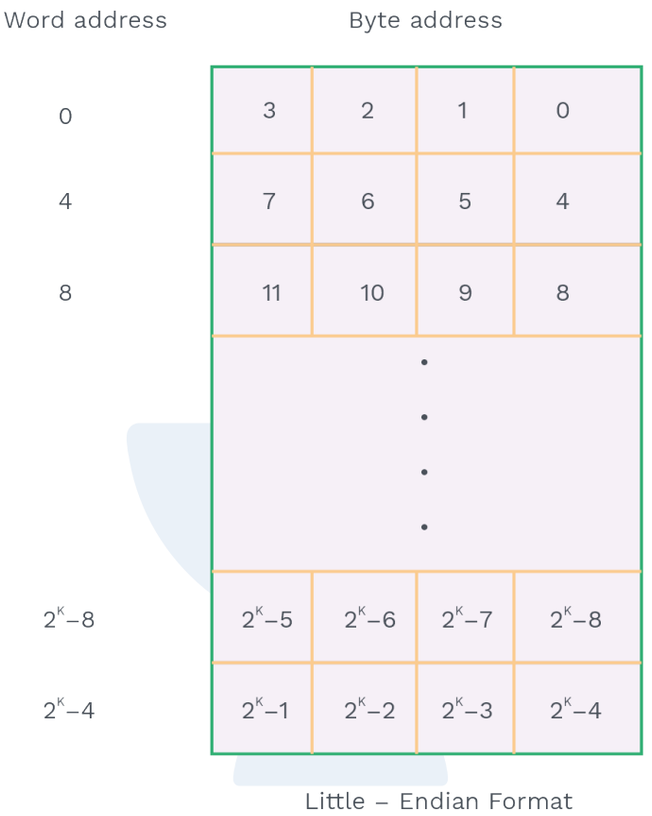


Fig. 1.16

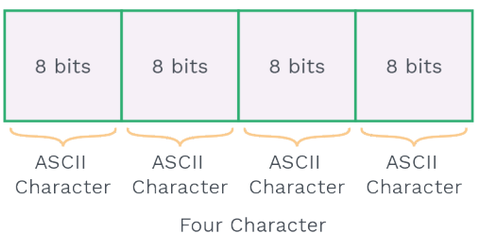


Fig. 1.17

- This is the example of encoded information in a 32-bit word.

PRACTICE QUESTIONS

Q15 **A CPU write the 'register content' into the memory in a big–endian mode. When the same content is read back from memory into a register, the CPU reads it in little–endian mode.**
Now choose the correct statement considering the above scenario.

a) The new content in the register will be different from what was written.
b) The contents will be the same
c) The content may be the same or different
d) Endianness need not be the same while reading and writing the contents into/from memory.

Sol: **a)**

In little–endian representation, the last byte of the binary number of multiple bytes data types is stored first.
In Big–endian representation, the first byte of the binary number of multiple data types is stored first.
Hence the new content in the register will be different from what was written.

Q16 **We are given two bytes of data as 10101100 00110011. How it will be stored in little–endian format?**

a)

0×100	0×101
00110011	10101100

b)

0×100	0×101
10101100	00110011

c)

0×100	0×101
10100011	11000011

d)

0×100	0×101
11000011	10100011

Sol: **a)**

In little–endian format, last byte of binary representation is stored first.

0×100	0×101
00110011	10101100

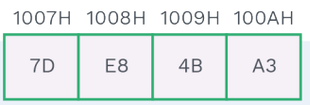
Q17 **A hexadecimal number A34BE87D is stored from memory address 1007H in little endian format. What will be the byte stored in memory location 1007H, 1008H, 1009H and 100AH?**

a) A3, 4B, E8, 7D **b) D7, 8E, B4, 3A**

c) A3, 4B, D7, 8E **d) 7D, E8, 4B, A3**

Sol: **d)** As we know in little–endian representation, the last byte of the binary number of multiple byte data type is stored first.

So hexadecimal number A34BE87D will be stored as:



Computer registers:

- Registers are required to store the operands, the result of the intermediate computations and also the addresses of the operands.
- The data register (DR) has the operand read from memory.
- The accumulator (AC) register is a general-purpose register.
- After fetching an instruction from memory, it is stored in the instruction register (IR).
- For storing the temporary data during the execution of an instruction, we use temporary registers(TR).

Special registers Symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds current instruction code
PC	12	Program counter	Holds address of next instruction

TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	holds output character

List of registers for the basic computer:

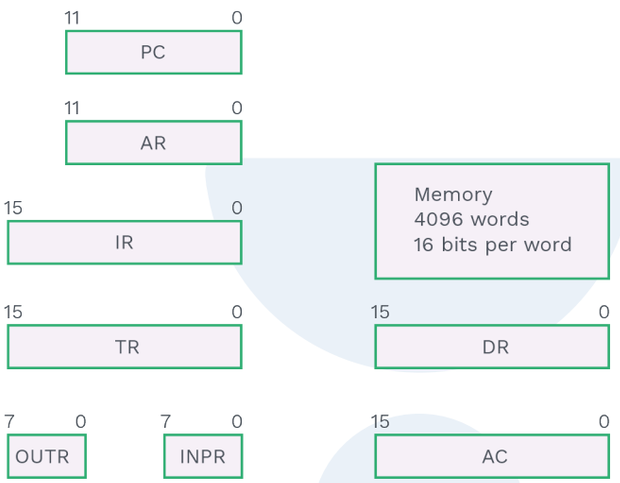


Fig. 1.18 Basic Computer Registers and Memory

- As there are 4096 words in the memory, therefore, the memory address register(AR) has 12 bits.
- Program counter is the register that holds the address of the instruction that is to be executed next.
- The PC goes through a sequence of counts, and it reads the instruction from memory sequentially.
- Instructions are read and executed sequentially unless any branch instruction comes across.
- A branch instruction jumps to a non-consecutive instruction in the program.
- The target address is held by the program counter (PC).
- There are mainly two registers : Input registers hold 8-bit data supplied by the input device. Output registers hold 8-bit data to output to an output device.

PRACTICE QUESTIONS

Q18 **Which of the following register is connected to system bus?**

a) PC (Program Counter) **b) MAR (Memory Address Register)**

c) IR (Instruction Register) **d) Accumulator**

Sol: **b)** MAR is connected to the system bus to access the memory. As we can see in the diagram given below.

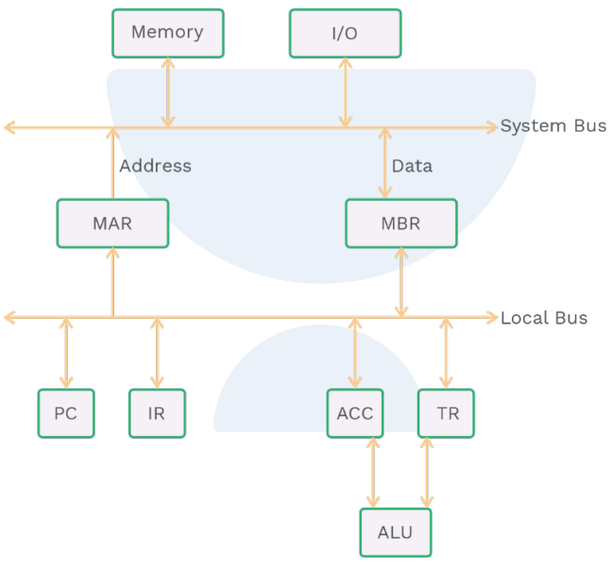


Fig. 1.20

Q19 **Which register always points to the next instruction to be executed?**

a) PC (Program Counter) **b) MAR (Memory Address Register)**

c) MDR (Memory Data Register) **d) IR (Instruction Register)**

Sol: **a)**

Program counter is the register that contains the address of the instruction which is to be executed next.

Q20 **Which register contains the most recently fetched instruction?**
a) MDR (Memory Data Register) **b) IR (Instruction Register)**
c) TR (Temporary Register) **d) MAR (Memory Address Register)**

Sol: **b)**

IR (Instruction Register) contains the instruction which is most recently fetched.

Q21 **Which register contains the address of the main memory location from where instruction has to be fetched?**
a) PC (Program Counter) **b) MAR (Memory Address Register)**
c) MDR (Memory Data Register) **d) IR (Instruction Register)**

Sol: **b)**

MAR signifies memory address register. It holds the address of the memory data to be fetched. It is connected to the address bus.

Q22 **Which register contains the contents found at the address held in the MAR?**
a) MDR **b) TR** **c) PC** **d) MAR**

Sol: **a)**

Memory Data Register (MDR) holds the contents found at the address held in the MAR. MDR content is directly connected to the data bus.

Common bus system:

- A basic computer design comprises a memory, a control unit and 8 special registers.
- The common bus acts as the connecting link among these modules. It is responsible for the transmission of information among these components.
- The connection of the registers and memory in a basic computer to a common bus system is explained in the diagram on the next page.
- The outputs of 7 registers and the memory are connected to the common bus.
- To select the specific output for bus lines at any time is determined from binary value of the selection variables S_0, S_1 and S_2.
- Each output refers to a number that represents a decimal number for given binary number.
- For example: DR holds S_2 S_1 S_0 = 011.
- The common bus lines are linked with the inputs of the memory and the registers.
- That register will only receive the data whose LD(Load) input enabled from the bus architecture in the next clock cycle.
- The bus content is transmitted to the memory when the Write Signal is set/Active.
- The 16-bit output content is placed from the memory to the bus when the Read Signal is set, and S_2 S_1 S_0=111.
- The registers-DR, IR, AC and TR are 16 bits each.
- The registers- PC and AR store 12-bit memory addresses.
- When the common bus caries memory addresses(12-bits), the initial four significant bits are set to 0's.
- After receiving the information from the bus, address register and the program counter transfers 12 least significant bits to register.

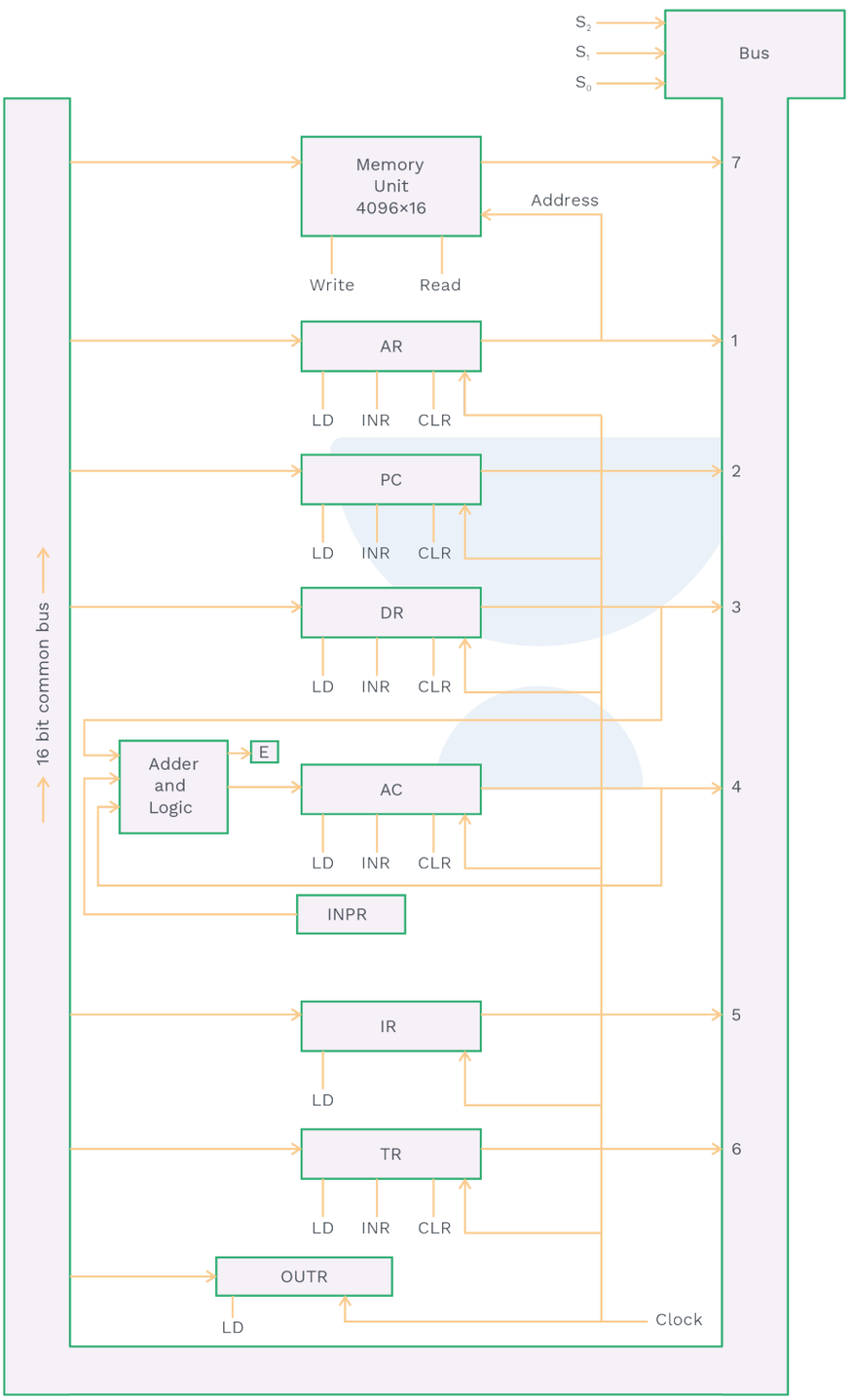


Fig. 1.20 Diagram of Basic Computer Registers Connected to a Common Bus

- INPR and OUTR refer to input and output registers.
- INPR is responsible for loading information to the bus whereas OUTR is responsible for taking data from bus.
- INPR takes data from bus and transfers it to the accumulator, and OUTR takes data from the accumulator and transfers it to other output devices.
- The bus lines are connected to the inputs of six registers and the memory.
- There are basic signals which are: LD (Load), INR (Increment) and CLR (Clear).

Chapter Summary

- Computer architecture refers to those attributes of a system which are visible to a programmer.
- Computer organization deals with the functionality of the hardware units and their connections.
- Types of Architecture
  1) **Von–Neumann architecture:**
     Based on the concept of the same memory, holding both data and instructions.
  2) **Harvard architecture:**
     Based on the concept of separate memories for data and instructions.
- Flynn's Classification
  1) Single Instruction, Single Data (SISD) stream
  2) Single Instruction, Multiple Data (SIMD) stream
  3) Multiple Instruction, Single Data (MISD) stream
  4) Multiple Instruction, Multiple Data (MIMD) stream

1) **Shared memory:**
   i) Symmetric multiprocessor (SMP)
   ii) Non–uniform memory access (NUMA)

2) **Distributed memory:**
   i) Clusters

- RAM chip organization:
  1) Vertical increasing (constant word size)
  2) Horizontal increasing (change in word size)
  3) Vertical + Horizontal increasing
- Memory locations and addresses

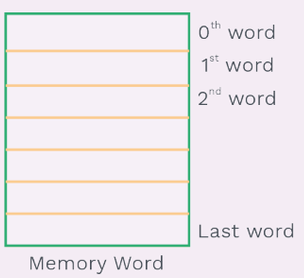


Fig. 1.21

- Byte ordering conventions:
  1) Big Endian
  2) Little Endian
- Computer Registers
  1) Data Register
  2) Address Register
  3) Accumulator
  4) Instruction Register
  5) Program Counter
  6) Temporary Register
  7) Input Register
  8) Output Register

2 Machine Instruction & Measuring CPU Performance

Stack CPU:

- In this organisation, ALU operations are performed only on stack data, which means both of the operands are always required to be in the stack.
  After the data processing, the result is also placed in the stack.
- In the stack, insertion and deletion operation takes place at the same end called as top of the stack (TOS), so TOS becomes a default location. Therefore, the address is not required in the instruction.
- Compatible instruction format is:

Opcode – – – 0 address instruction format

Example:

(A * B) + C ; variables are in the memory.

Code:

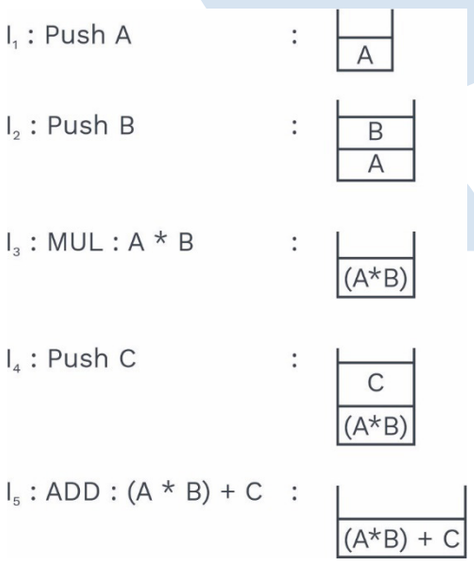
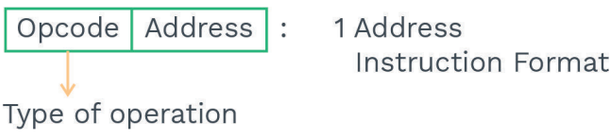


Fig. 2.1 Operations in Stack

Accumulator CPU:

- In this organisation, for an ALU operation, the first operand is always required in the accumulator and the second operand is present either in the register or in the memory.
  After the data processing, the result is placed into the accumulator.
- In the CPU design, only 1 accumulator is present so it becomes a default location. Its address is not required in the instruction.

- Compatible instruction format is



Example: (A * B) + C, variables are in the memory

Code:

I_1: Load A ; Accumulator ← M[A]

I_2: MUL B ; Accumulator ← Accumulator * M[B]

I_3: Add C ; Accumulator ← Accumulator + M[C]

General register CPU:

- Based on the number of registers possible in the CPU, architecture is of two types:
  - **i)** Register to memory reference CPU (CPU with less registers)
  - **ii)** Register to Register reference CPU (CPU with more registers)

Register to memory reference CPU:

- In this organisation, ALU first operand is always required in the register and the second operand is present either in the register or in the memory. After the data processing, the result will be placed into source$_1$ (first operand) location.
- Compatible instruction format is:

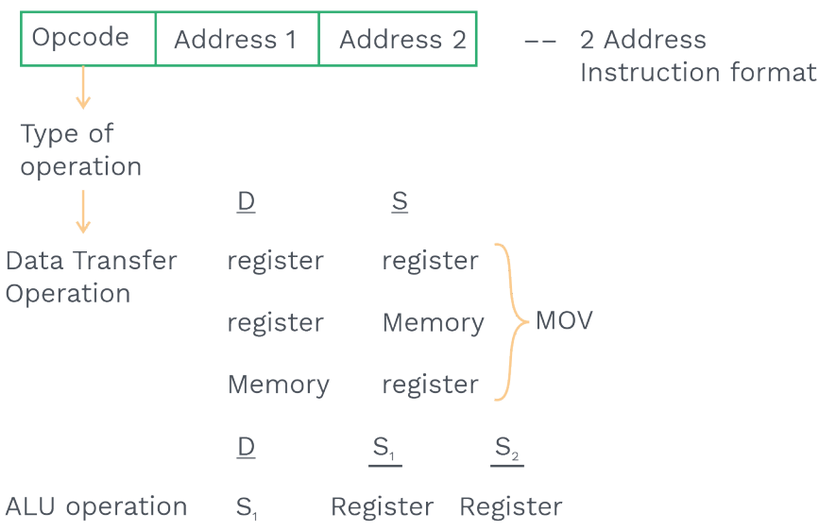


Fig. 2.2 2-Address Instruction Format

D: Destination
S= source
S_1= source 1
S_2 = source 2

Example: (A * B) + C ; variables are in the memory.

Codes:

		Action
I_1 : MOV r_0 , A	:	$r_0 \longleftarrow$ M[A]
I_2 : MUL r_0 , B	:	$r_0 \longleftarrow r_0$ * M[B]
I_3 : ADD r_0 , C	:	$r_0 \longleftarrow r_0$ + M[C]

Register to register reference CPU:

- In this organisation, ALU operations are performed only on register data, so both the operands are always present in the register. After the data processing result is also placed in a register.
- Compatible instruction format is:

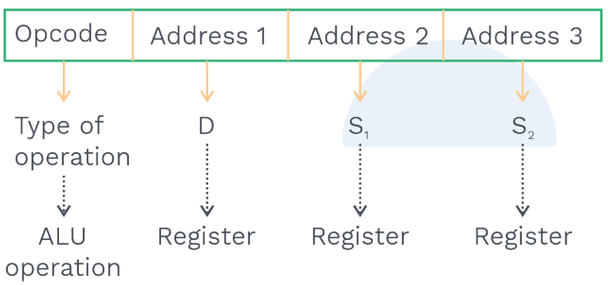


Fig. 2.3 3 Address Instruction Format

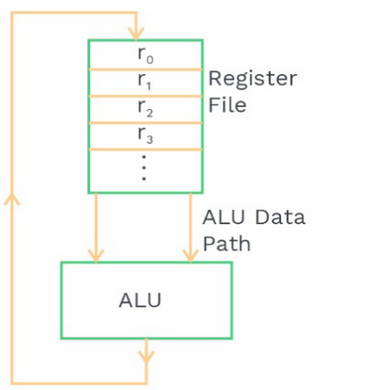


Fig. 2.4 Register-Register Reference CPU

Example: (A * B) + C: Variables are in the memory

Code:

I_1: Load r_0, A
I_2: Load r_1, B
I_3: MUL r_2, r_0, r_1
I_4: Load r_3, C
I_5: Add r_4, r_2, r_3

4 address instruction format:

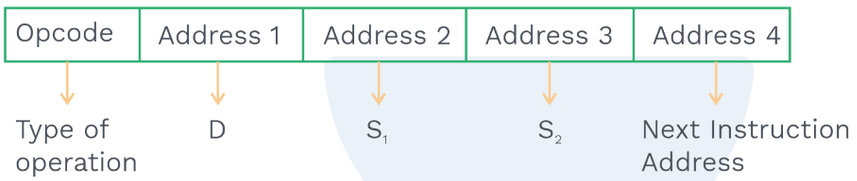


Fig. 2.5 4-Address Instruction Format

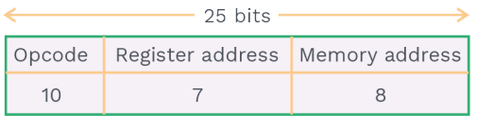
Note:
Program Counter (PC) is the mandatory register utilised by the CPU, which holds the address of the next instruction during the accomplishment of the current instruction. Therefore, this format (4 address instruction format) is not in use.

PRACTICE QUESTIONS

Q1 **A processor has 700 distinct instructions and 80 general purpose registers. A 25-bit instruction word has an opcode, one register operand and a memory operand. How many bits are reserved for the memory operand field?**
a) 5 **b) 10** **c) 8** **d) 6**
Sol: c)

Sol: Given, that we have 700 distinct instructions, so it requires $\lceil \log_2 700 \rceil$ bits = 10 bits.

- 80 general purpose registers requires $\lceil \log_2 80 \rceil$ bits = 7 bits.
- Size of the instruction = 25 bits



- Number of bits available for the memory operand field = 25 – (10+7)
  = 25 – 17
  = 8

Previous Years' Question

A processor has 40 distinct instructions and 24 general purpose registers. A 32–bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is _____ .

a) 16 bits
b) 17 bits
c) 18 bits
d) 19 bits

Sol: a) **(GATE-2016 (Set-2)**

ADDRESSING MODES

- Addressing mode shows the location of a required object in the computer.
- Object may be a data or instruction.
- Output of an addressing mode is the effective address (EA).
- EA is the actual address of an object,

  Object = [EA] content of

- Addressing mode can be implemented in the instruction in 2 ways:

1) Implicit: Here, opcode itself specifies the type of addressing mode used.
Instruction:

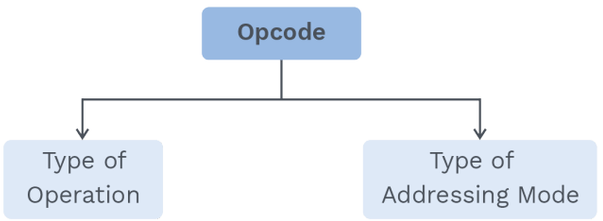


Table 2.1 Implicit Addressing Mode

2) Explicit: Mode field is used in the instruction to specify the type of addressing mode used.
Instruction:

- Symbols used for specifying the different addressing modes (AMs) in the user program are as follows:

Type of addressing mode	Symbol
1) Immediate AM	#
2) Register AM	Register name
3) Direct AM	[ ]
4) Indirect AM	@ , ( )
5) Indexed AM	Index register name
6) Auto indexed AM	+ , –

Table 2.2 Symbolic Representation of Addressing Modes

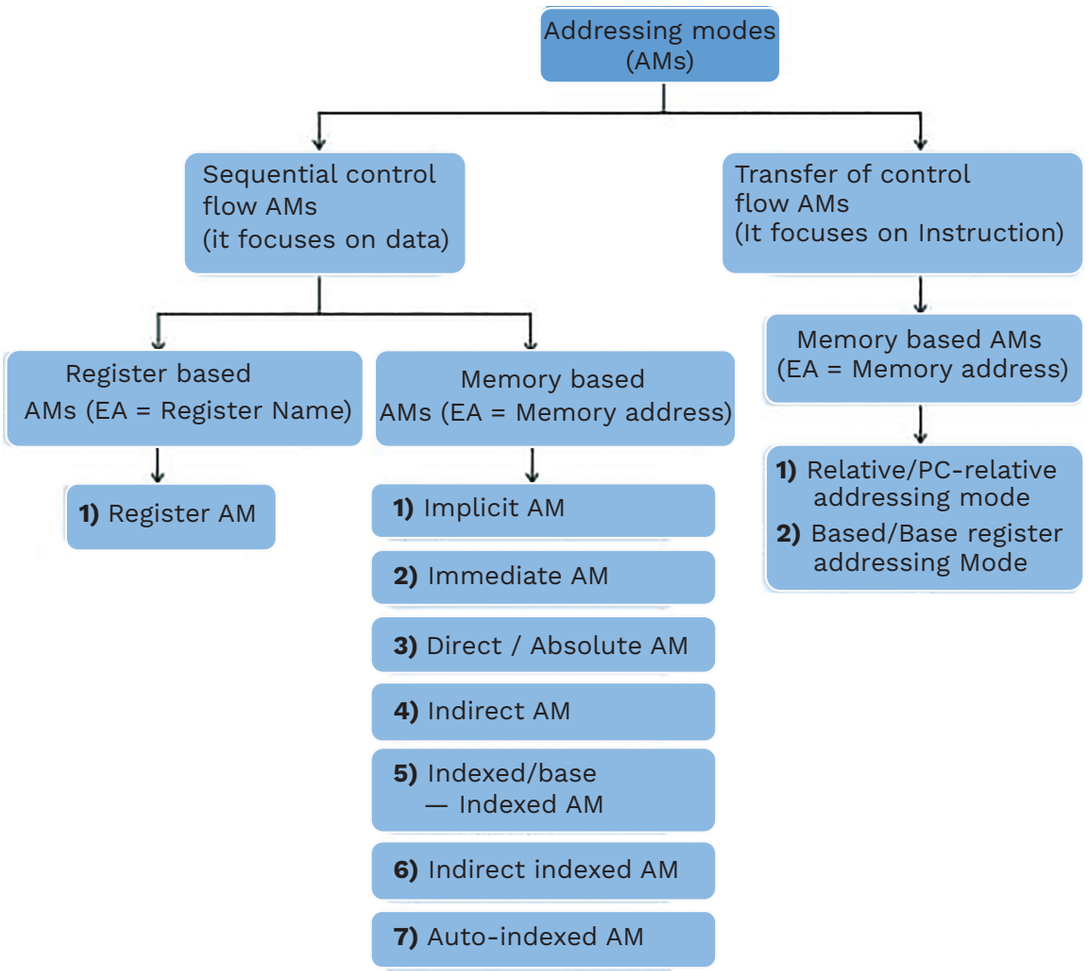


Table 2.3 Classification of Addressing Modes

Memory based:

Sequential control flow AM:

- These types of modes concentrate on the data location, so data transfer and data manipulation instructions are designed with this addressing modes.

  1) Implied addressing mode:

  In the implied addressing mode, the opcode itself specifies the data information along with the operation.

  Example: Set Carry Clear Carry

  STC CLC

  Carry = 1 Carry = 0

Example: ADD on Stack CPU

i) POP
ii) POP
iii) +
iv) Push

Note:
All zero address instructions are implied instructions, but all implied instructions are not zero address instructions.

2) Immediate addressing mode:
This mode is used for accessing the constants. In this mode, data is present in the address field of an instruction.
Instruction:

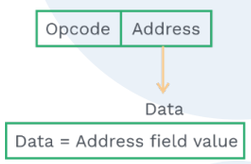


Fig. 2.6 Immediate Addressing Mode

Example:

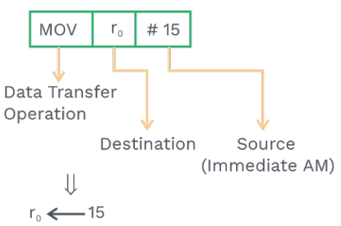


Fig. 2.7 Valid Data Transfer using Immediate Addressing Mode

Example:

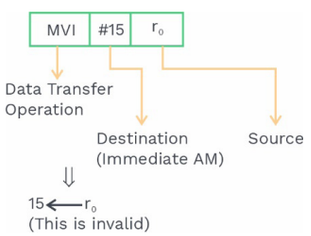


Fig. 2.8 Invalid Data Transfer using Immediate Addressing Mode

Limitation:

Range of constants is restricted by the size of the address–field, i.e., if k–bit address field, then

Range = {0 to (2^k – 1)} unsigned constants.

Range = {– (2^{k-1}) to + (2^{k-1} – 1)} signed constants.

3) Direct addressing mode:

- This mode is best utilised to refer to the static variables.
- In this mode, the address field of the instruction contains the effective address of data which is present in the memory.

Instruction:

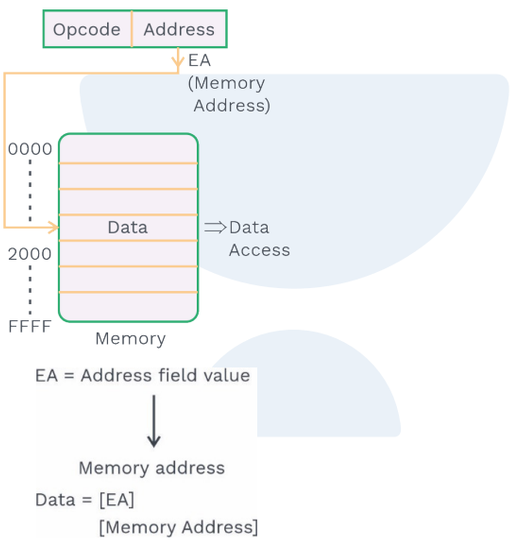


Fig. 2.9 Direct Addressing Mode

- Here, 1 memory reference is required to access the data.

Example: Consider 3 word Instruction

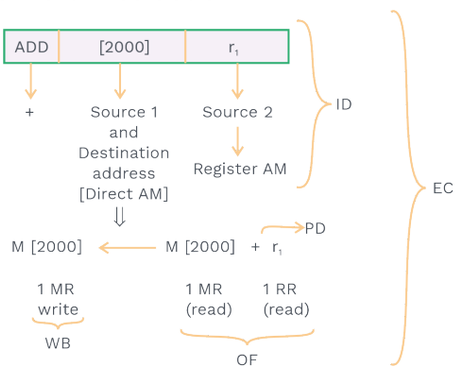


Fig. 2.10 Data Transfer using Direct Addressing Mode

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	2 MR	S_1	S_2	1ALU	1MR
		1MR	1RR		

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data (Execute)
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference

Fig. 2.11 Instruction Cycle

4) Indirect addressing mode:

The idea of pointers is best realised using indirect addressing mode.

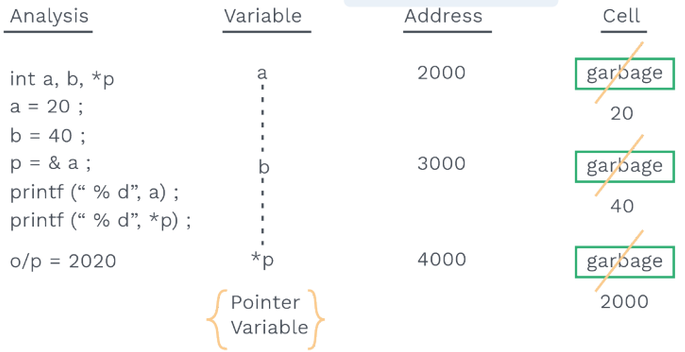


Fig. 2.12 Pointers using Indirect Addressing Mode

- The default storage class is auto, so the default value is garbage.

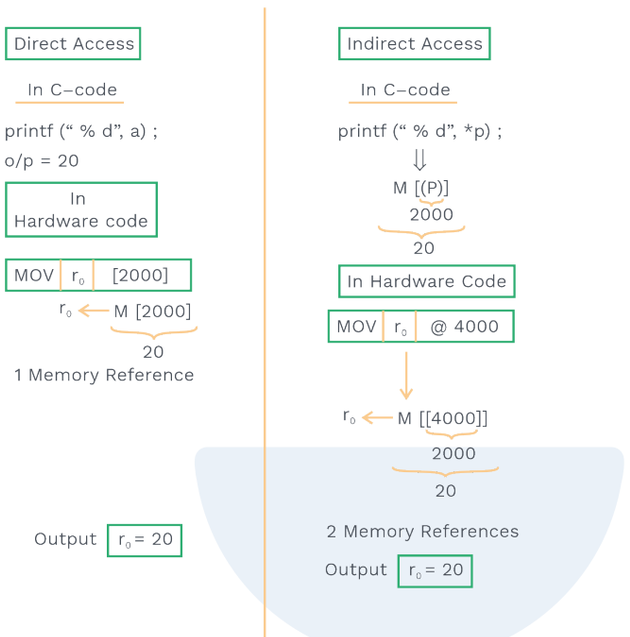


Fig. 2.13 Program Analysis using Direct and Indirect Addressing Modes

4) a) Memory indirect addressing mode:

In this mode, the effective address [EA] is present in the memory; the corresponding memory address will be maintained in the address field of instruction as an address of the Effective Address.

Instruction:

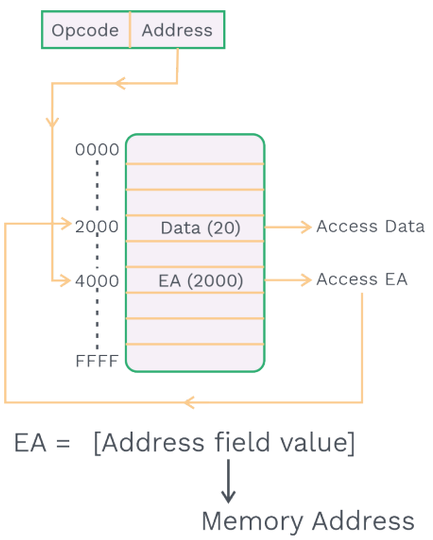


Fig. 2.14 Memory Indirect Addressing Mode

Data = [EA]
[[Memory Address]]
* 1 memory reference to access the EA.
* 1 memory reference to access the data.

Example: Consider the 4 word Instruction.

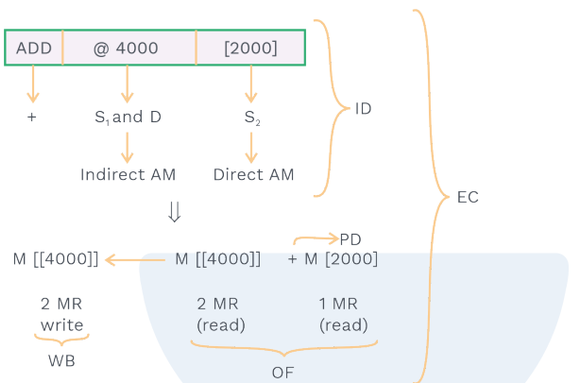


Fig. 2.15 4-Word ALU Instruction

Instruction cycle:

Fetch Cycle (FC)		Execution Cycle (EC)			
IF	ID	OF		PD	WB
1 MR	3 MR	S_1	S_2	1 ALU	D
		2MR	1MR		2 MR

4) b) Register indirect AM:
The address bits of the instruction format store the name of the register containing the effective address of the operand in the memory.
Instruction:

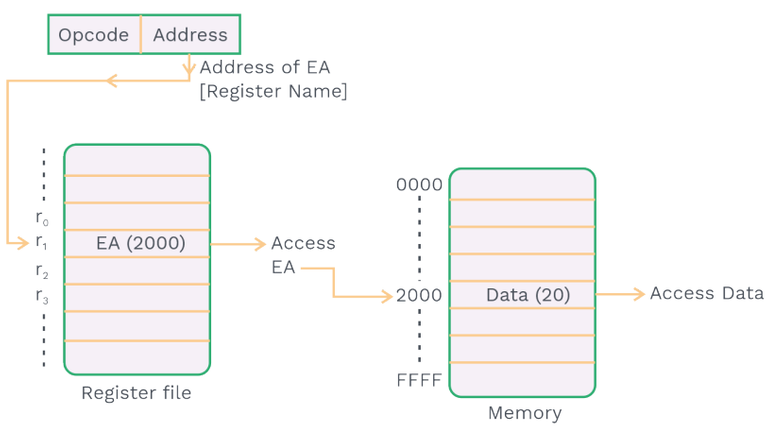


Fig. 2.16 Register Indirect Addressing Mode

Data = [EA]
[Register Name]
* 1 register reference required to access the Effective Address.
* 1 memory reference to access the data.
Example: Consider the 4 word instruction.

Instruction:

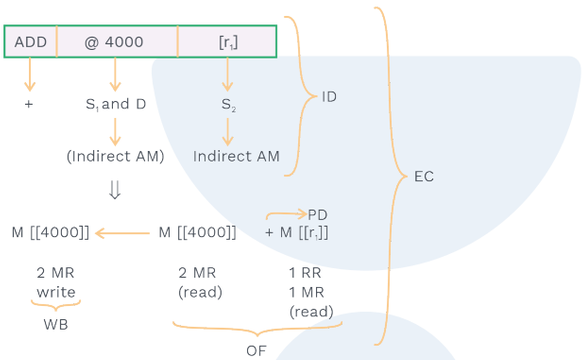


Fig. 2.17 4-Word ALU Instruction using Memory and Register Indirect Addressing Modes

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	3 MR	S_1	S_2	1ALU	D
		2MR	1RR 1MR		2 MR

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference
RR = Register reference

Fig. 2.18 Description of Instruction Cycle

5) Indexed addressing mode:

- This mode is used to access the arrays.

Analysis:

char a [10] ;

Storage:

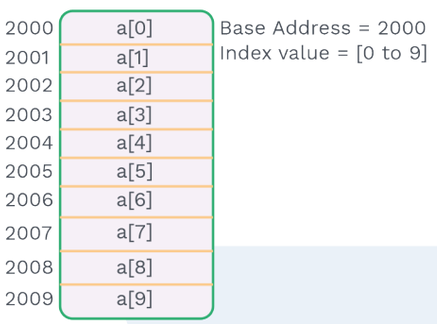


Fig. 2.19 Indexed Addressing Mode

Example: a[4]

I_1: MOV r_0, # 4 Assume r_0 as an index register

I_2:

MOV	r_1	2000 (r_0)

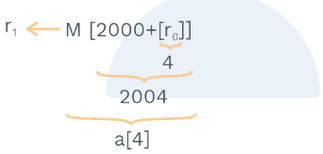


Fig. 2.20 Random Accessing of Array Elements I

r_1 = a[4]

Example: a[10]

I_1: MOV r_0, # 10

I_2:

MOV	r_1	2000 (r_0)

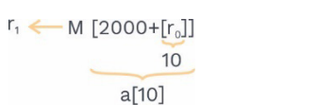


Fig. 2.21 Random Accessing of Array Elements II

r_1 = a[10]

- The above 2 examples are in the category of randomly accessing the array.

i) Pre increment:

I_1: MOV r_0, # 1 FFF

I_2: | MOV | r_1 | + (r_0) |

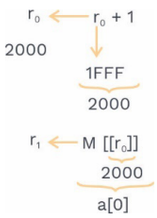


Fig. 2.22 Pre-Increment Indexed Addressing Mode

ii) Post increment:

I_1: MOV r_0, # 2000

I_2: | MOV | r_1 | (r_0) + |

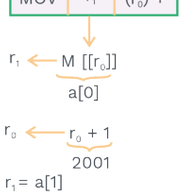


Fig. 2.23 Post-Increment Indexed Addressing Mode

iii) Pre decrement:

I_1: MOV r_0, # 200 A

I_2: | MOV | r_1 | – (r_0) |

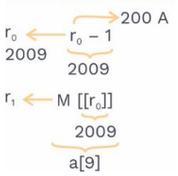


Fig. 2.24 Pre-Decrement Indexed Addressing Mode

r_1 = a[9]

iv) Post decrement:

I_1: MOV r_0, # 2009

I_2: | MOV | r_1 | (r_0) – |

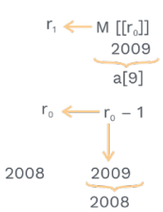


Fig. 2.25 Post-Increment Indexed Addressing Mode

r_1 = a[9]

- Above all are examples of linearly accessing the memory.
- Indexed AM is used to access the random array elements from the memory.
- Here 2 parameters are required.

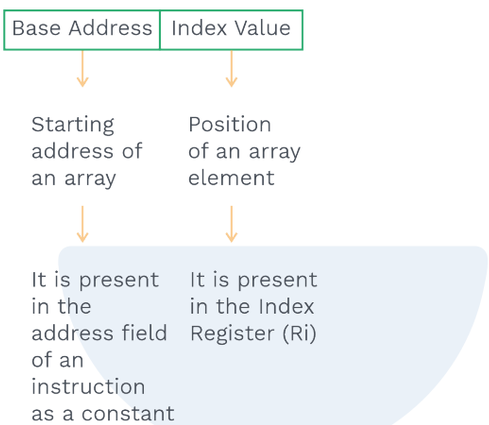


Fig. 2.26 Array Accessing Parameters

- The Effective address (EA) is obtained by adding the constant to the contents of the (Ri).

EA = Address field value + [Ri]

Data = [EA]

- Here 1 register reference, 1 memory reference and 1 ALU operation are required.

Instruction design:

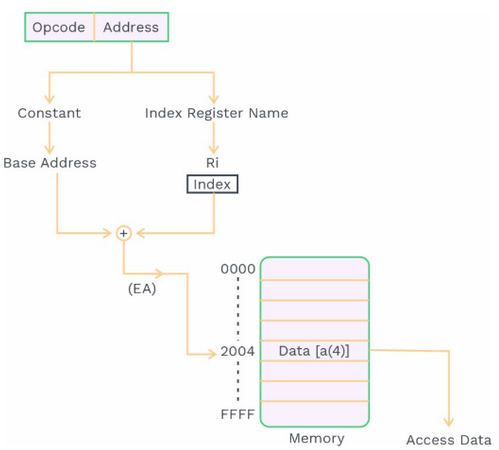
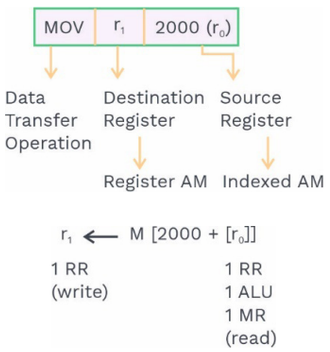


Fig. 2.27 Instruction Design

Example: Consider 3 word instruction:



Instruction fetch = 1 MR
Instruction decode = 2 MR
Operand fetch = S = 1 RR
1 ALU
1 MR
Write Back to D = 1 RR

S = Source Register
D = Destination Register

Fig. 2.28 3-Word Data Transfer Instruction using Register and Indexed Addressing Modes

6) Indirect indexed AM:

- EA is present in the memory in this addressing mode.
- In this mode, the address of the EA is calculated by adding the index value to base address.
  EA = [Address field value + (Ri)]
  Data = [EA]
- 1 RR is required to get the index
  1 ALU is required to calculate the address of EA
  1 MR is required to get the EA
  1 MR is required to access the data

Instruction design:

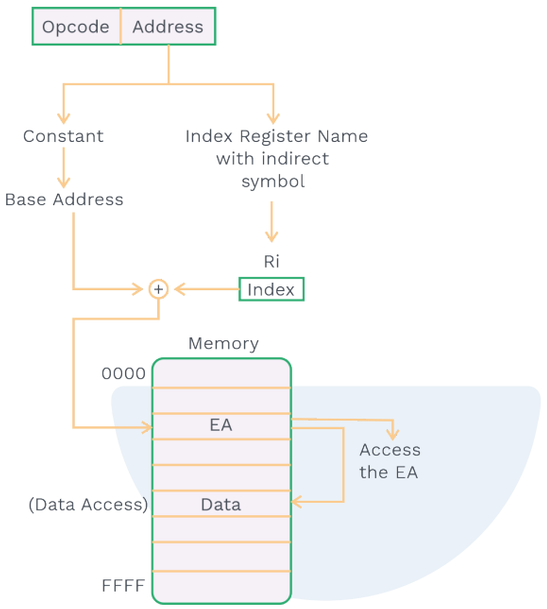


Fig. 2.29 Instruction Design using Indirect Indexed Addressing Mode

- This mode takes more time to access the data because 2 MR are required to access the EA and data, respectively.

7) Auto indexed AM:

- It is used to access the linear array of elements from memory.

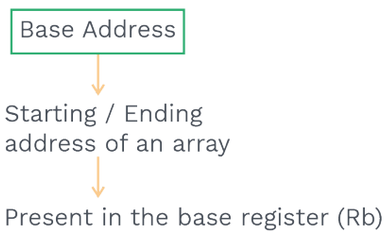


Fig. 2.30 Linear Accessing of Array Elements

- Effective address is calculated either by incrementing or decrementing the contents of Rb with some given step size.
- Step size is fixed constant, depending on the word length of the CPU.
  EA = [Rb] (+ / –) step size
  Data = [EA]

1) 1 RR is required to get the base address (Rb).
2) 1 ALU operation is required to calculate the EA.
3) 1 MR is required to access the data.

Instruction design:

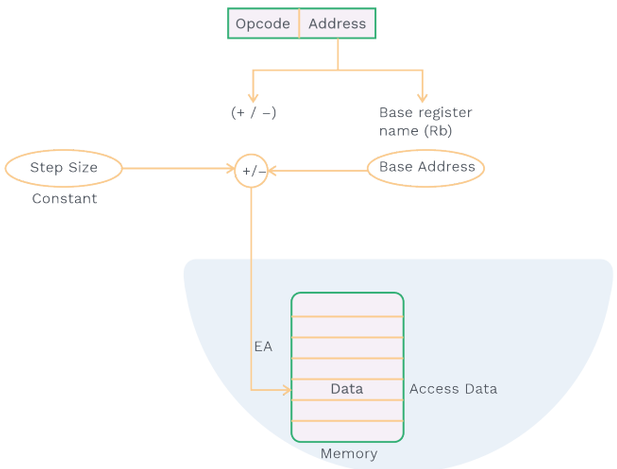


Fig. 2.31 Instruction Design using Pre-Increment/ Decrement Auto Indexed Addressing Mode

- This is pre-increment/pre decrement Auto indexed AM.

Instruction design:

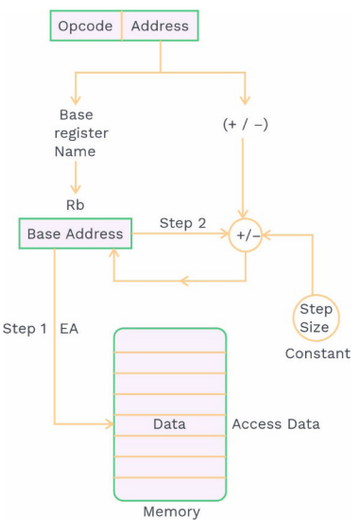


Fig. 2.32 Instruction Design using Post-Increment/ Decrement Auto Indexed Addressing Mode

- This is post-increment/post-decrement auto indexed AM.

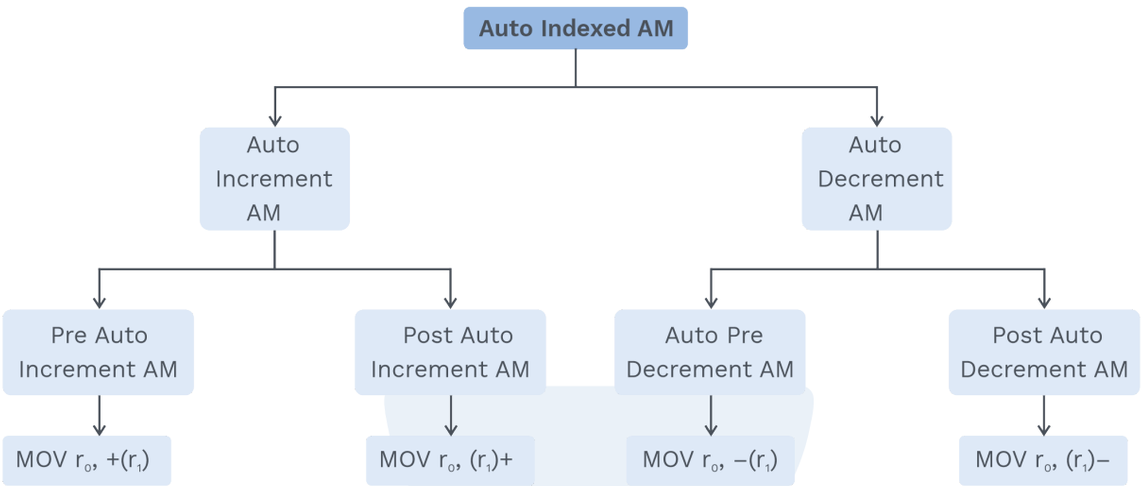


Table 2.4 Classification of Auto Indexed Addressing Modes

- Here "r_1" is the base register.
- By default, auto increment AM uses post auto increment AM.
- And by default, auto decrement AM uses pre auto decrement AM.

Register based:

Register addressing mode:

The local variables are referred to on utilising this mode. In this mode, data is present in the register, and the corresponding register name will be maintained in the address field of instruction as effective address.

Instruction:

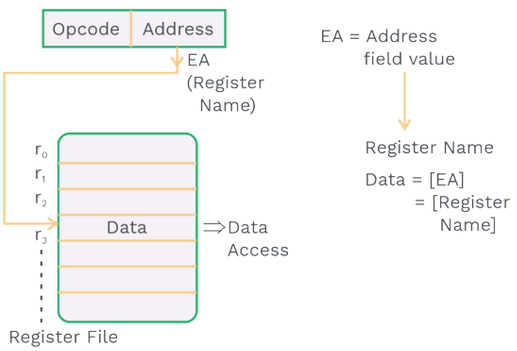


Fig. 2.33 Register Addressing Mode

- Here, 1 register reference is there to access the data.

Example: Consider 2 word instruction

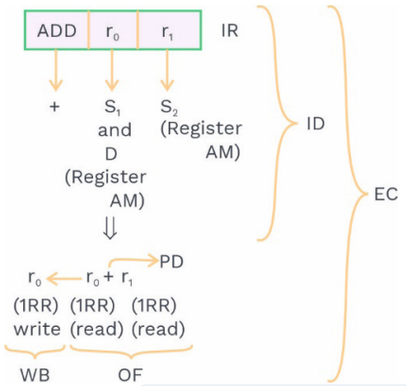


Fig. 2.34 2-Word ALU Instruction using Register Addressing Mode

Instruction cycle:

Fetch Cycle (FC)	Execution Cycle (EC)				
IF	ID	OF		PD	WB
1 MR	1 MR	S_1	S_2	1ALU	1RR
		1RR	1RR		

IF = Instruction Fetch
ID = Instruction Decode
OF = Operand Fetch
PD = Process Data (Execute)
WB = Write Back

S_1 = Source 1 register
S_2 = Source 2 register
D = Destination register
MR = Memory reference

Fig. 2.35 Description of Instruction Cycle

Transfer of control flow addressing modes:

- These addressing modes focus on the location of the next instruction.
- When the program contains control structures, then during the program execution, control will be transferred from the current location to the target location.

Code:

```
        1000 : I1
        1001 : I2
        1002 : I3   (goto l1)
        1003 : I4
          :
          :
[l1]    2000 : Branch Instruction 1 (BI1)
        2001 : Branch Instruction 2 (BI2)
```

Output sequence: $I_1 - I_2 - I_3 - BI_1 - BI_2 - - -$

- To implement the control structures in the base hardware, transfer of control (TOC) instructions are used, named JUMP (JMP) / branch/ skip.
- Transfer of control instructions are designed with the following addressing modes, used to calculate the effective address or target address or branch address.

Types of transfer of control addressing modes (AMs):

1) PC relative AM

2) Base register AM

- For analysing the above AMs, we will consider the 8086-memory organisation as a reference model.
- 8086 supports 1 MB physical memory, organized into 16 logical segments with each segment size of 64 KB.

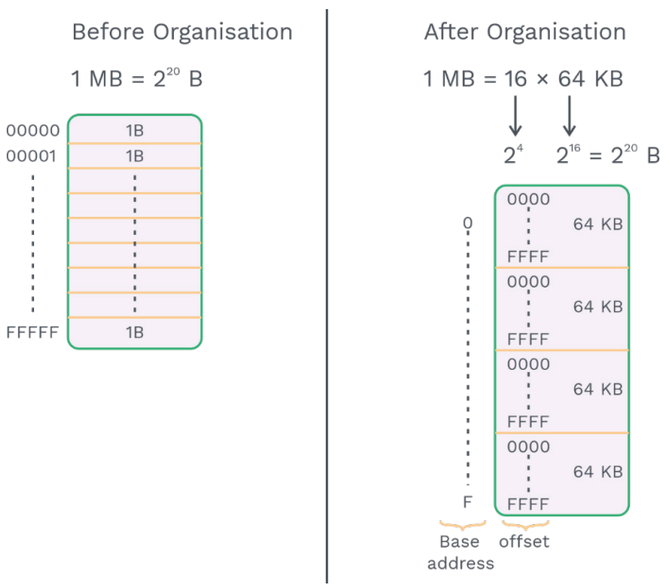


Fig. 2.36 Memory Organization of 8086 Microprocessor Chip

Intra-segment TOC:

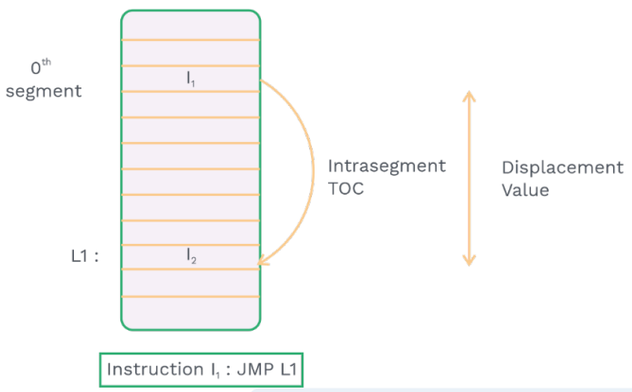


Fig. 2.37 Intra-segment TOC

Inter-segment TOC:

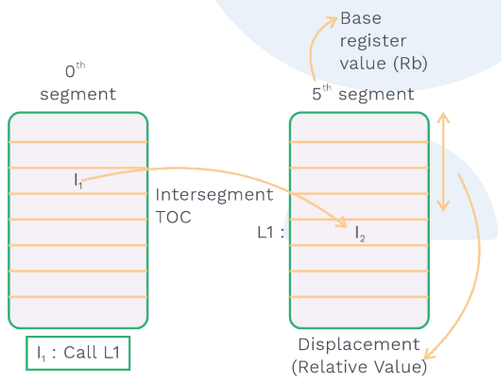


Fig. 2.38 Inter-segment TOC

1) PC relative AM:

- The targeted instruction, when available in the same segment, control is relocated within the segment while machine code execution. This procedure is termed Intra-segment TOC.
- This Addressing Mode (AM) is used to implement the above operation.
- Effective Address is obtained by adding the relative value to PC.
- Relative value means the distance between the current location and target location. It is a signed constant present in the address field of instruction.

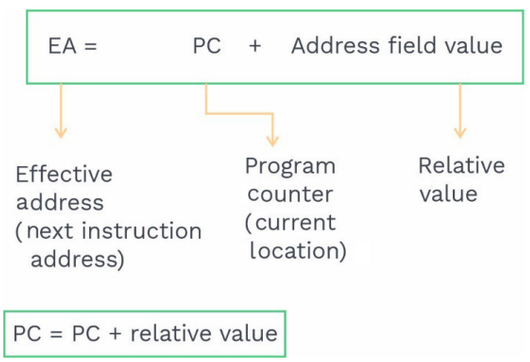


Fig. 2.39 PC Relative Addressing Mode

2) Base register AM:

- The targeted instruction, when available in a distinct segment, control is relocated from one segment to another segment while machine code execution. This procedure is termed Inter-segment TOC.
- Base Register AM is best suitable for the implementation of the above concept.
- Effective Address (EA) is obtained by adding the relative value to the content of the base register.

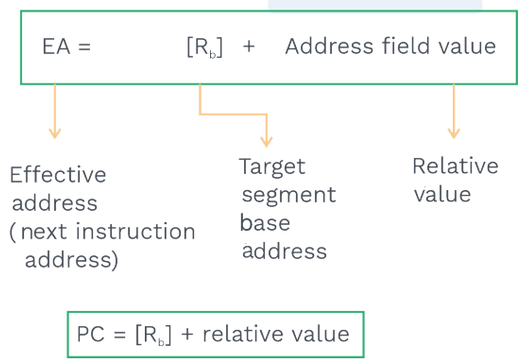


Fig. 2.40 Base Register Addressing Mode

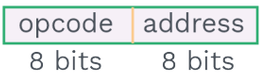
Note:

i) Runtime program relocation is best supported by both PC relative and base register addressing modes.

ii) Position independent codes framing is best supported by a base register addressing mode.

Consider the following data for the examples (2 to 8):

- A 2 byte instruction is present in memory starting at $(550)_{10}$, and the instruction format is given below:

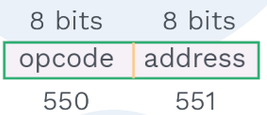


- Address field contains the value $(400)_{10}$
- M[400] = 600
- Register R1 contains the value $(320)_{10}$
- R1 has an address of 800.

PRACTICE QUESTIONS

Q2 **Calculate the effective address if immediate addressing mode is utilised.**
Sol: 551

Sol:



Value of address field = 400

- In immediate addressing mode, the value in the address field is used as data.
- 400 is used as data to opcode in immediate addressing mode.

400 is stored at address 551.
so, effective address = 551

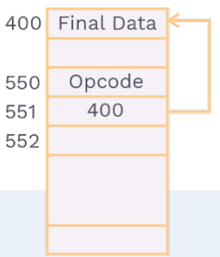
Q3 **Calculate the effective address if register addressing mode is used.**
Sol: 800

Sol: Register R1 contains the value of 320, and it is stored at address 800. So, effective address = 800

Note:
Here we are assuming only one register R1 is given.

Q4 **What will be the effective address using direct addressing mode?**
Sol: 400

Sol: In direct addressing mode, the value at the address field is considered the actual address of the data. So, here effective address = 400
Final data resides at address location 400.



Q5 **What is the effective address using register indirect addressing mode?**
Sol: 320

Sol: In register indirect addressing mode, whatever the data contained in register will be considered as the final address.
Here R1 contains the value of 320.

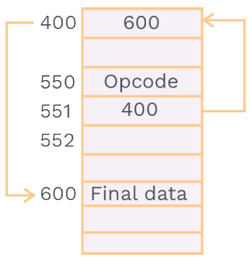
R1 | 320 |
800

so, effective address = 320

- At memory location 320, the final data resides.

Q6 **Calculate the effective address using memory indirect addressing mode.**
Sol: 600

Sol: This addressing mode works as a pointer
M[400] = 600
In memory location 600, the final data resides.



so the effective address = 600

Note:
The effective address is the final address where data resides.

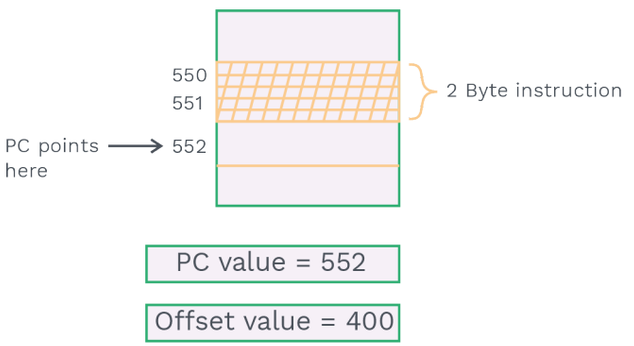
Q7 **Calculate the effective address using indexed–addressing mode (R1 =index register).**
Sol: 720

Sol: In indexed–addressing mode, the value in the address field will be used as the base address.

- Here address field value = 400
- Index value = Value in R1 = 320
- Effective address = Base register + Index value
  = 400 + 320 = 720

Q8 **Calculate the effective address using the PC–relative addressing mode.**
Sol: 952

Sol: In PC–relative addressing mode, the value in the address field will be used as an offset.



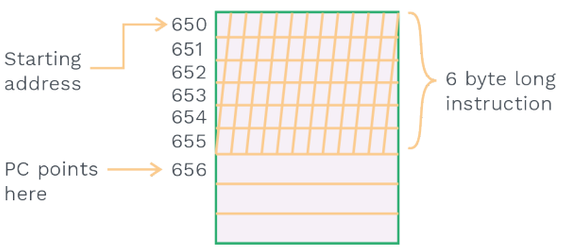
Effective address = PC value + Offset value
= 552 + 400
= 952

Note:
For PC–relative addressing, we are assuming that we have JUMP like instruction.

- **Consider the following data for examples (9 to 11):**
A 6 Byte long PC–relative jump instruction started at $(650)_{10}$ onwards, and (–88) is the signed displacement value which is present in the address field of the instruction.

Q9 **What is the branch address to which it will branch (or the effective address)?**
Sol: 568

Sol:



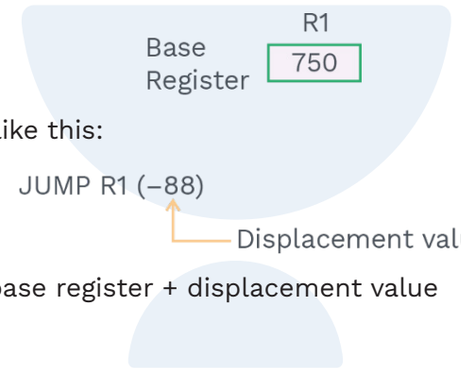
- Relative address is also known as displacement value. This field has a value of (–88).

PC value = 656

effective (Branch) address = PC value + Relative (displacement value)
= 656 + (−88)
= 568

Q10 **If base−register R1 contains $(750)_{10}$, what will be the branch address (or the effective address) if base−register addressing mode is used?**
Sol: 662

Sol: Suppose the base register is R1



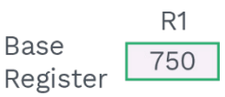
Instruction will look like this:

Effective address = base register + displacement value
= 750 + (−88)
= 662

Note:
The base register always contains the first address of the segment. The displacement field contains displacement within the segment.

Q11 **Calculate the branch address (the effective address) if based−indexed addressing mode is used, if index−value = 92.**
Sol: 754

Sol:

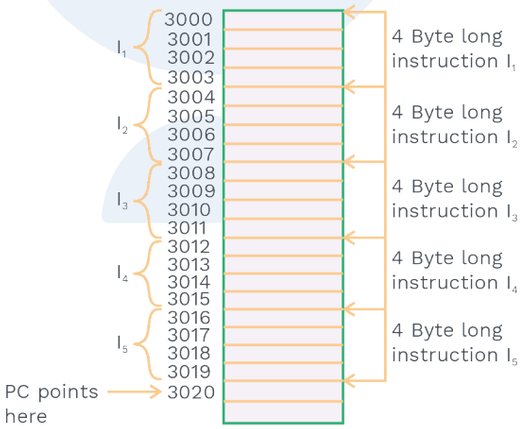


index value = 92
Displacement value = (−88)
Effective address = Base register value

+
Index–value
+
Displacement value
= 750 + 92 – 88
= 754

Q12 **Assume that there are 5 instructions in the loop body; starting memory locations are 3000, 3004, 3008, 3012 and 3016, respectively. Assume each instruction takes 4 bytes of memory and memory is byte addressable. Calculate the offset value needed to return to the loop (starting of the loop). Sol: 20**

Sol: Suppose five instructions are I_1, I_2, I_3, I_4 and I_5.



- PC value at the end of loop (or after instruction I_5) = 3020.
- To return to starting of the loop, the effective address should be equal to starting address.

Effective address = PC value + Offset value
Effective address = starting address of the loop
Effective address = 3000
3000 = PC value + Offset value
3000 = 3020 + Offset value
Offset value = –20

PC = Program Counter

Q13 **In a certain processor, a 4 Byte jump instruction is encountered at memory address $(3603)_{10}$; the jump instruction is in PC relative mode. The instruction is JMP(–13), where (–13) is a signed byte. Determine the branch target address. (Assume byte addressable memory).**

a) 3594 **b) 3590** **c) 3616** **d) 3620**

Sol: a)

Sol: The jump instruction is at address $(3603)_{10}$, and the instruction is 4 byte long.

Hence ; PC (Program Counter) points to 3603 + 4 = 3607

Effective (branch) address = PC value + Displacement (relative) value

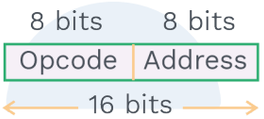
Here relative (displacement) value = –13

Effective address = 3607 + (–13)

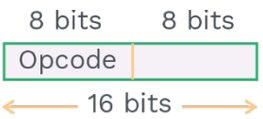
= 3594

Expand opcode–technique:

- Assume a microprocessor with fixed–length instructions (16 bits). This microprocessor has both 1–address instructions and 0–address instructions.

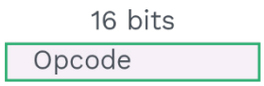


- This is a 1–address instruction format; here, opcode = 8 bits and address field = 8 bits. So, 1–address instruction perfectly fits inside fixed length instruction size.



0–address instruction format

- Here, opcode is 8 bits, but the issue is with the remaining 8 bits because the microprocessor has a fixed length instruction of size 16 bits. So, the question arises, what do we have to do with these 8 bits which are remaining.
- So, now we will use this method of expanding the opcode. In the remaining 8 bits, we have the scope of filling 00000000 to 11111111. So, instead of wasting these 8 bits, we will expand the opcode to use all 16 bits.

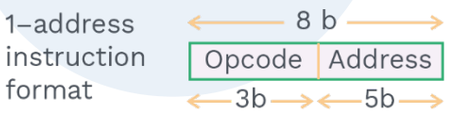


0–address instruction format

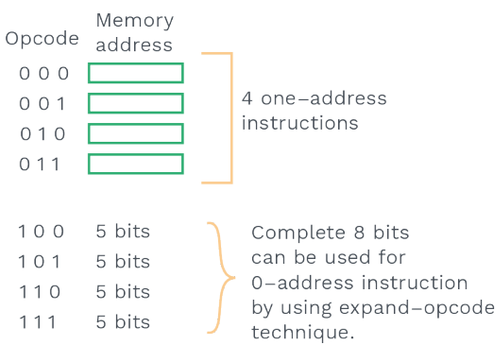
- In this way, the size of the opcode field is increased.

Q14 **A microprocessor supports both 0–address instructions and 1–address instructions. The memory size is 32 words. This microprocessor has fixed–sized instructions of 8 bits. Suppose there are 4 one–address instructions. Calculate the number of possible zero–address instructions.**
Sol: 128

Sol: Given fixed instruction size of 8 bits.



There are 32 words in memory, hence address field of 1–address instruction will take $\lceil \log_2 32 \rceil = 5$ bits. So the remaining 3 bits are there for the opcode.
Total number of 1–address instructions possible = $2^3 = 8$.
But we are using only 4 one–address instructions, the remaining 8 – 4 = 4 can be used for 0–address instructions.



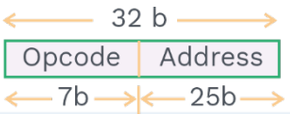
So total, we can have 4×2^5 zero–address instructions.
$2^2 \times 2^5 = 2^7 = 128$ zero–address instructions

Q15 **Consider a microprocessor having both 0–address and 1–address instruction and a fixed size instruction of 32 bits. The main memory is 32 MW in size. If there are 'x' 1–address instructions, calculate the number of 0–address instructions.**

Sol: Memory size = 32 MW

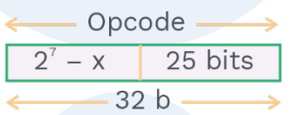
$= 2^5 \times 2^{20} = 2^{25}$ words

so 25 bits will be used for the address field.



1–address instruction format

Total opcodes possible 2^7, 'x' is used for 1–address instructions, remaining 2^7–x can be used for 0–address instructions.



0–address instruction format

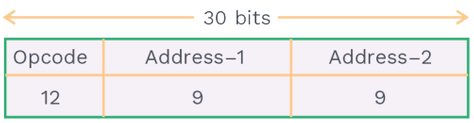
- Number of 0–address instructions possible $= \boxed{(2^7 - x) \times 2^{25}}$ by using expand opcode technique.

Q16 **A computer system supports two–address, and one–address instructions, and the word size are 30 bits. The main memory is word addressable, and it supports 512 words. If there are 160 two address instructions, then how many one address instructions are there in the system?**
a) 2,015,232 b) 1, 007, 616 c) 503, 808 d) 4, 030, 464
Sol: a)

Sol: Given instruction size = 30 bits

address size = 512 words, it requires $\lceil \log_2 512 \rceil$ = 9 bits

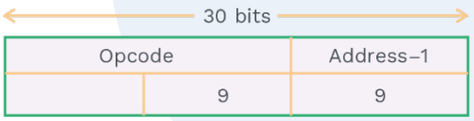
2–address instruction format:



2–Address Instruction Format

Number of 2–address instructions possible = 2^{12} = 4096
But we are using only 160, 2–address instructions,
so remaining opcodes = 2^{12} – 160
= 4096 – 160
= 3936

- We can use these opcodes in 1–address instructions by using expand opcode technique.



One–Address Instruction Format

- So total number of one address instruction = 3936 × 2^9
= 3936 × 512
= 2,015,232

Performance measuring of CPU:

- Performance can be measured on the basis of two criterias.

i) Execution time (ET):

Performance is inversely proportional to the execution time.

$$P(\text{Performance}) \propto \frac{1}{ET}$$

If there are two microprocessors A and B. If A is taking 5 ns and B is taking 10 ns. As $P \propto \frac{1}{ET}$, we can say that A has higher performance than B.

ii) Throughput:

It is defined as the number of jobs completed (executed) per unit time.

Speed–up–factor (S):

If there are two microprocessors X and Y, we want to evaluate how fast is X as compared to Y. As we know $P(\text{Performance}) \propto \frac{1}{ET}$.

$$(\text{Speedup})\, S = \frac{P_X}{P_Y} = \frac{\frac{1}{ET_X}}{\frac{1}{ET_Y}} = \frac{ET_Y}{ET_X}$$

- If X executes a process in 5 ns and Y executes the same process in 20 ns. Then

  $$\text{speedup}(S) = \frac{ET_Y}{ET_X} = \frac{20\,\text{ns}}{5\,\text{ns}}$$

  $$\text{speedup}(S) = 4.$$

  X is 4 times faster than microprocessor Y.

Amdahl's law:

It is a formula which gives the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected from the system whose resources are improved.

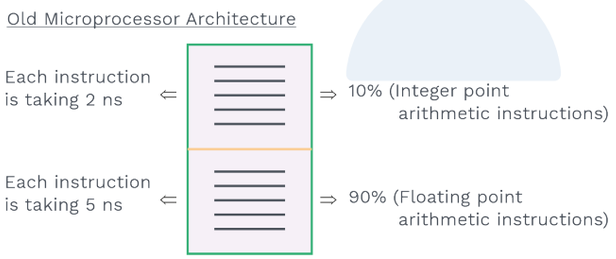


Fig. 2.41 Old Microprocessor Architecture

Expected execution time (old architecture)

ET_{old} = 0.1 × 2 + 0.9 × 5

= 0.2 + 4.5 = 4.7 ns

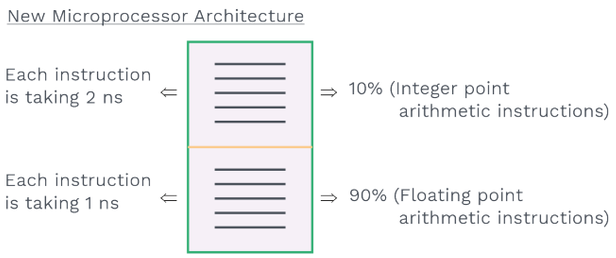


Fig. 2.42 New Microprocessor Architecture

Expected execution time (New architecture)

$ET_{new} = 0.1 \times 2 + 0.9 \times 1$

$= 1.1$ ns

- In Amdahl's law, common case (more frequently occurring instructions) is made fast. For example, floating point instructions are made fast in above scenario.

$$\text{overall speedup} = \frac{ET_{old}}{ET_{new}} = \frac{4.7\,ns}{1.1ns}$$

$$= 4.27$$

Generalised case:

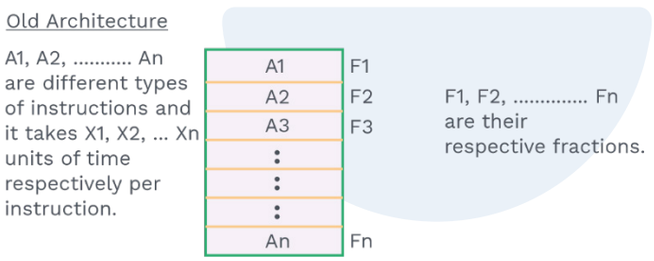


Fig. 2.43 Generalised Case of Old Architecture

Expected/average execution time

$$ET_{old} = \sum_{i=1}^{n} XiFi$$

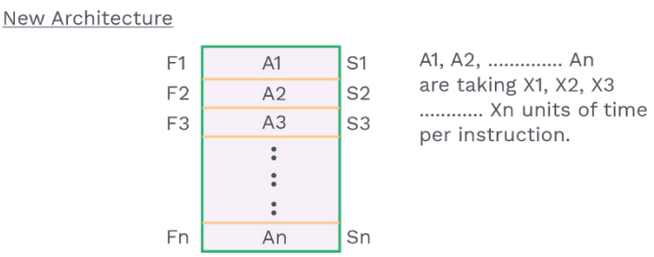


Fig. 2.44 Generalised Case of New Architecture

- S1, S2, S3 ..... Sn are the speedup of A1, A2, A3 ....., An types of instructions respectively, where each S1, S2 ..... Sn are greater than or equal to 1 ($Si \geq 1$) ($i = 1$ to $i = n$)

Expected/Average execution time

$$ET_{new} = \sum_{i=1}^{n} Fi\left(\frac{Xi}{Si}\right)$$

$$\text{Overall speedup} = \frac{ET_{old}}{ET_{new}}$$

Deriving the Amdahl's formula:

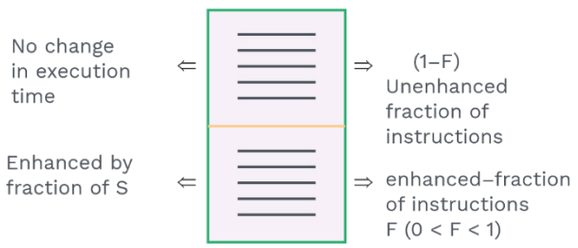


Fig. 2.45 Derivation of Amdahl's Formula

- Assume each instruction initially takes the same time; let's take 1 ns for simplicity.
  ET_{old} = 1 ns

$$\text{Overall speedup } (S_{overall}) = \frac{ET_{old}}{ET_{new}}$$

$$= \frac{1}{1\times(1-F)+\frac{1}{S}\times F}$$

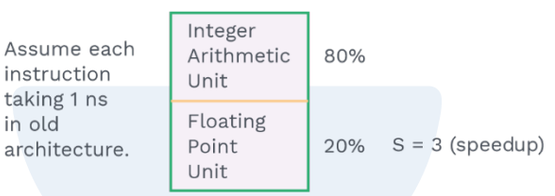
$$= \frac{1}{(1-F)+\frac{F}{S}}$$

$$= \frac{1}{1-(\text{Fraction})_{enhanced}+\frac{(\text{Fraction})_{enhanced}}{(\text{Speedup})_{enhanced}}}$$

PRACTICE QUESTIONS

Q17 **Assume a microprocessor given which is widely used for scientific applications. It has both integer and floating point instructions. Now floating point unit is enhanced and made 3 times faster than before, the integer point unit is still unenhanced. If there are 20% of floating point instructions in the program. Calculate the overall speedup achieved.**

Sol:



$ET_{old} = 1$ ns

$$ET_{new} = 0.8 \times 1 + \frac{0.2}{3}$$

$$= \frac{3 \times 0.8 + 0.2}{3}$$

$$= 0.867$$

$$S_{overall} = \frac{ET_{old}}{ET_{new}} = \frac{1}{0.867} = \boxed{1.153}$$

- Now by using formula–based approach:

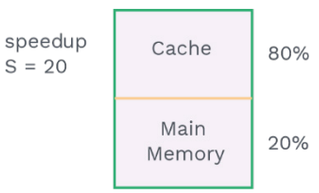
$$= \frac{1}{(1-F) + \frac{F}{S}} \quad \text{Here F = 20\%}$$

$$= \frac{1}{(1-0.2) + \frac{0.2}{3}}$$

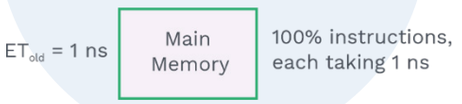
$$= \frac{3}{3 \times 0.8 + 0.2} = \frac{3}{2.6} = \boxed{1.153}$$

Q18 **Assume in a hypothetical system cache is 20 times faster than main memory and CPU refers to the cache 80% of the time. Calculate the overall speedup of this system.**

Sol:



- If we assume that in an old system, only the main memory was there, and each instruction takes 1 ns.



$$ET_{new} = (1-F) + \frac{F}{S}$$

F = 80%

$$S_{overall} = \frac{1}{(1-F) + \frac{F}{S}}$$

$$= \frac{1}{(1-0.8) + \frac{0.8}{20}}$$

$$= \frac{1}{0.2 + 0.04} = \frac{1}{0.24}$$

$$= \boxed{4.167}$$

Q19 **In a hypothetical system, floating point instructions are improved to run five times as fast, but only 40% of the time was spent on these instructions originally. How much speedup is achieved by the new machine?**
a) 1.85 b) 1.47 c) 1.39 d) 1.22
Sol: b)

Sol: By using Amdahl's law:

$$\text{Overall speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{1 - (\text{Fraction})_{\text{enhanced}} + \frac{(\text{Fraction})_{\text{enhanced}}}{(\text{Speedup})_{\text{enhanced}}}}$$

$$= \frac{1}{\left((1 - 0.4) + \frac{0.4}{5}\right)}$$

$$= \frac{1}{(0.6 + 0.08)}$$

$$\frac{1}{0.68} = \boxed{1.47}$$

Processor–clock cycle time and frequency

- **t = clock cycle time,** it is a time interval between two consecutive rising edges or 2 consecutive falling edges.
- **Frequency (f) =** $\frac{1}{t}$ (It is measured in cycles per unit time). Its SI unit is Hertz (Hz).
- $\boxed{\text{CPU time}}$ = Instruction count × CPI × Clock cycle time

CPI = Clocks per instruction
Here, we are assuming each instruction takes the same amount/number of cycles.

- For different instructions, CPU takes a different number of instructions in the real world, e.g., data—transfer instruction, data–manipulation instructions and transfer of control instructions take a different number of cycles.

$$\text{CPU time} = \left[\sum_{i=1}^{K} IC_i \times CPI_i\right] \times t$$

IC_i = Instruction count for i^{th} type of instruction
CPI_i = Clocks per instruction for each i^{th} type of instruction
t = clock cycle time.

PRACTICE QUESTIONS

Q20 **Assume the frequency (clock rate) of CPU is given as 4.5 GHz. Assume we have different type of instructions which have different number of instruction count (IC) and different CPI. Now calculate i) average instruction execution time ii) MIPS–rate iii) Program execution time (ET).**

Instruction type	Instruction count	CPI
Load	**100**	**10**
Store	**200**	**8**
Arithmetic	**300**	**5**
Logical	**150**	**4**
Shift	**250**	**3**
Branch	**400**	**2**

Sol: Clock rate (frequency) = 4.5 GHz

$$t = \text{clock cycle time} = \frac{1}{4.5 \times 10^9}\,\text{sec}$$
$$= 0.22\ \text{ns}$$

a) Average–instruction Execution time (ET) $= \dfrac{\left[\sum_{i=1}^{K} Ic_i \times CPI_i\right] \times t}{\text{Total instruction count}}$

$$= \frac{(100 \times 10 + 200 \times 8 + 300 \times 5 + 150 \times 4 + 250 \times 3 + 400 \times 2) \times 0.22\,\text{ns}}{100 + 200 + 300 + 150 + 250 + 400}$$

$$= \frac{(6250) \times 0.22\,ns}{1400} = 0.9821ns$$

b) MIPS–rate
MIPS = Millions of instructions per second
On average, 1 instruction is taking = 0.9821 ns
In 1 sec, we can execute,

$$\frac{1}{0.9821 \times 10^{-9}}$$

$$= \frac{10^9}{0.9821} = 1.018 \times 10^9 \text{ instructions}$$

$= 1018 \times 10^6$

$= \boxed{1018\,MIPS}$

c) Program execution time
= Total instruction × Average instruction execution time
= 1400 × 0.9821 ns
= 1374.94 ns

Q21 **In a certain processor, we have a program which takes 1 million instructions to execute with processor whose speed is given as 2GHz. Suppose 40% of the instruction takes 2 clock cycles, 30% of the instructions take 3 clock cycles and the remaining 30% takes 5 clock cycles for their respective execution. What is the total execution time for the program (in milliseconds)?**
a) 3.2 **b) 4.8** **c) 1.6** **d) 2.4**
Sol: c)

Sol: We have a total instructions = 10^6
Frequency = 2GHz

$$\text{Clock cycle time} = \frac{1}{\text{Frequency}}$$

$$= \frac{1}{2 \times 10^9} \text{ sec}$$

$$= \boxed{0.5 \times 10^{-9} \text{ sec}}$$

Total execution time = Total CPI × Number of instructions × Clock cycle time

$\boxed{\text{CPI} = \text{Clocks per instruction}}$

Number of clock cycles	Percentage of instructions	Multiplication
2	40% = 0.40	0.40 × 2 = 0.8
3	30% = 0.30	0.30 × 3 = 0.9
5	30% = 0.30	0.30 × 5 = 1.5

Total CPI = 0.8 + 0.9 + 1.5
= 3.2
Total execution time = $3.2 \times 10^{6} \times 0.5 \times 10^{-9}$
= 1.6×10^{-3}
= 1.6 milliseconds
= 1.6 ms

Q22 **Consider a 2 GHz processor which is used to execute a benchmark program with the following instruction and clock cycle count:**

Instuction	Clock cycle count
80000	2
40000	3
5000	3
10000	4

What is the effective CPI of the program and total execution time?
a) CPI = 2.48 **b) Execution time = 2.5 ns**
c) CPI = 3.20 **d) Execution time = 1.24 ns**
(ns = Nanoseconds) **(CPI = Clocks per instruction)**
Sol: a), d)

Sol: Total instruction counts
= 80000 + 40000 + 5000 + 10000
= 135000

$$\text{effective CPI} = \frac{\Sigma\left(\underset{I_i}{\text{Instruction count}}\right) \times \left(\underset{C_i}{\text{Clock cycle count}}\right)}{\text{Total instruction count}}$$

I_i = Instruction count of i^{th} instruction
C_i = Clock cycle of i^{th} instruction

$$= \frac{(80000 \times 2) + (40000 \times 3) + (5000 \times 3) + (10000 \times 4)}{135000}$$

$$= \frac{335000}{135000} = \boxed{2.48}$$

Program execution time = Effective CPI × Clock cycle time

$$\text{Clock cycle time} = \frac{1}{\text{Frequency}} = \frac{1}{2 \times 10^9} \text{ sec}$$

$$\text{Program execution time} = 2.48 \times \frac{1}{2 \times 10^9} \text{ sec}$$

$= 1.24 \times 10^{-9}$ sec = $\boxed{1.24\,\text{ns}}$

CISC and RISC processors:

- CISC = Complex Instruction Set Computer
- It has a large number of instructions

e.g.:

i) MUL [1080] [1050]
ii) MUL r_0, r_1
iii) MUL r_0, [1080]

} Different types of multiplication instruction

- For multiplication, large number of instructions are present:
  - **i)** Multiplication between 2 memory location data.
  - **ii)** Multiplication between 2 register data.
  - **iii)** Multiplication between register and memory location data.
- It has more complex circuitry because of a large number of instructions, so it will consume more power.
- It has fewer registers because most of the space is occupied by the circuitry on the physical chip; hence less space is available for registers.
- Instructions in this architecture have a variable number of cycles to execute.

For example: Some instructions have a long execution time; these are the instructions that copy an entire block from one part of the memory to another part of the memory. Others copy multiple registers to and from memory.

- **i)** MUL [2080] [1050]
- **ii)** MUL r_0, [1080]

Both are multiplication instructions only but take a different number of cycles for execution because

- **i) instruction Involves only memory operands**
- **ii) instruction involves both register and memory operand.**

- Examples of CISC are: Pentium i3, i5, i7.

RISC: Reduced instruction set computer:

- It has fewer instructions.
  MUL R_0, R_1

It can only multiply data in 2 registers.

MUL[1080][1050] will be translated into these four instructions.

LOAD R_0, [1050]
LOAD R_1, [1080]
MUL R_0, R_1
STORE [1080], R_0

- It has less number of instructions, so it has simple circuitry.
- As it has simple circuitry, it will consume less power; for example smart phones and i–pads prefer this (RISC) architecture so that they have longer battery life.
- As it has simpler circuitry, it can have more number registers on the same chip as compared to CISC based architecture.
- It has a fixed number of cycles per instruction (mostly 1 cycle per instruction).
  LOAD R_0, [1050] $\rightarrow$ 1 cycle
  LOAD R_1, [1080] $\rightarrow$ 1 cycle
  MUL R_0, R_1 $\rightarrow$ 1 cycle
  STORE [1080], R_0 $\rightarrow$ 1 cycle
- All these can be executed in 1 cycle.
- Examples of RISC architecture-based devices are Mobile, i–pads and ARM processors for laptops.
- Key differences between RISC and CISC

S. No.	RISC	CISC
1)	Simple instructions taking one cycle	Complex instructions taking multiple cycles
2)	Supports fixed format instructions	Supports variable format instructions
3)	Contains multiple register sets	Contains single register set
4)	Few instructions	Many instructions
5)	Highly pipelined	Not pipelined or less pipelined
6)	Few addressing modes, and most instructions have register to register addressing mode	Many addressing modes

S. No.	RISC	CISC
7)	Consumes less power	Consumes more power
8)	Uses hardwired control unit	Uses microprogrammed control unit
9)	Very few instructions refer to memory	Most of the instructions may refer to memory

Table 1.5 Comparison between RISC and CISC Architectures

Register–organisation in RISC:

$\boxed{G(g)}$ G = Global register file

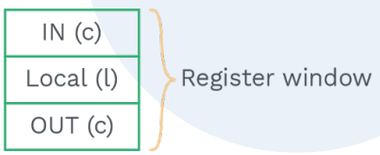


Fig. 2.46 Layout of Register Window

Register window = Collection of registers, having IN, OUT (Common registers) and local registers (l).

g = number of global registers
l = number of local registers per window
c = number of common registers per window
w = number of windows

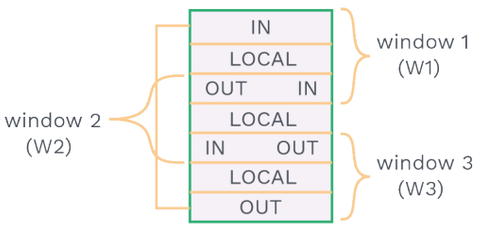


Fig. 2.47 Overlapped Register Window

- OUT of W1 is shared with IN of W2; similarly OUT of W2 is shared with IN of W3 and OUT of W3 is shared with IN of W1.

Total number of registers:

$$g + w \times l + \frac{w(2 \times c)}{2}$$

$$= \boxed{g + w(l + c)}$$

PRACTICE QUESTIONS

Q23 **A RISC processor has 400 registers. Each window has 4 inputs, 20 local and 4 output registers. The number of register windows is 10. What is the total number of global registers?**

a) 180 **b) 140** **c) 160** **d) 200**

Sol: c)

Sol: Given,

L = Local register = 20

G = Global register = G

C = Common register = 4

W = Number of windows = 10

- In RISC, overlapping of registers is there, 'Input' register of one window overlaps with 'output' register of other. So, only one among the 'input' and 'output' registers is counted.

Total registers = 400

Number of registers in RISC processor

$400 = G + W (L + C)$

$400 = G + 10 (20 + 4)$

$400 = G + 10 (24)$

$400 = G + 240$

$\boxed{G = 160}$

Previous Years' Question

Consider the following processor design characteristics.

1) Register–to–register arithmetic operations only

2) Fixed–length instruction format

3) Hardwired control unit

Which of the characteristics above are used in the design of a RISC processor?

a) 1 and 2 only

b) 2 and 3 only

c) 1 and 3 only

d) 1, 2 and 3

Sol: d) **(GATE-2018)**

Instruction set:

i) Arithmetic Instructions:

a) INC (Increment) instruction:

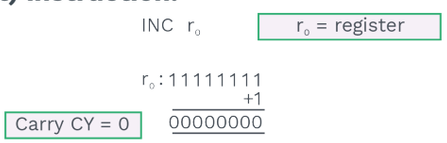


Fig. 2.48 Increment Instruction

b) DEC r_0 (Decrement) instruction:

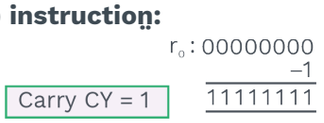


Fig. 2.49 Decrement Instruction

- If the initial value of the carry flag is 1, then after the above operation, it will still remain the same, it will not be 0 as happens in a typical SUB operation. It would not change.
- So, in increment and decrement instructions carry value will not change; it will remain the same as before in most microprocessors.

ii) Logical Instructions:

1) AND, OR and EX–OR:

Binary number	AND	OR	EX–OR
0 0	0	0	0
0 1	0	1	1
1 0	0	1	1
1 1	1	1	0

Table 1.6 AND, OR and EX-OR Operations

- Every microprocessor provides these logical instructions.

2) CMP (Comparison) instruction:

CMP r_0 r_1 (Comparison of register r_0 and r_1)

- Internally ALU is used and ($r_0 - r_1$) is performed through subtraction, the and result is stored in accumulator.

CMP	CY (Carry flag)	Z (Zero flag)
$r_0 = r_1$	0	1
$r_0 > r_1$	0	0
$r_0 < r_1$	1	0

Table 1.7 Status of CY and Z Flags in CMP Instruction

- We can compare r_0 and r_1 on the basis of carry (CY) and zero flags (Z), which are stored in the program status word (PSW).

iii) Shift operations:

1) Logical shift right:

Least Significant Bit (LSB) will fall off, and MSB will be filled with '0'.

Example:

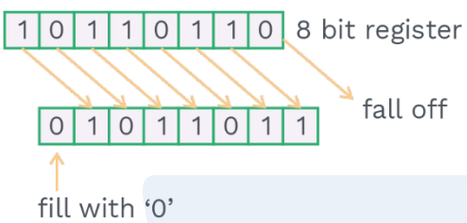


Fig. 2.50 Logical Shift Right Operation

- We can shift many times in the same manner.

2) Logical shift left:

It is opposite to the right shift. The most significant bit (MSB) will fall off, and LSB (Least significant bit) will be filled with '0'.

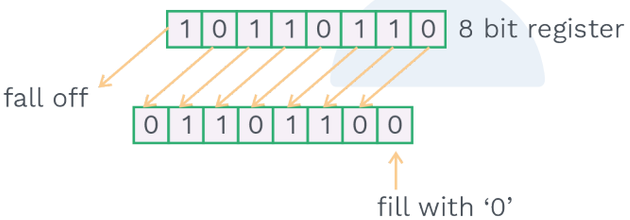


Fig. 2.51 Logical Shift Left Operation

- We can shift as many times as asked to shift.

3) Arithmetic right shift:

(LSB) Least Significant Bit will fall off, and MSB (Most Significant Bit) will be filled with the same bit value of MSB as before.

Example: 1

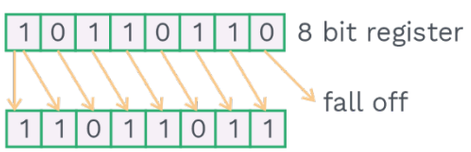


Fig. 2.52 Arithmetic Shift Right Operation I

Example: 2

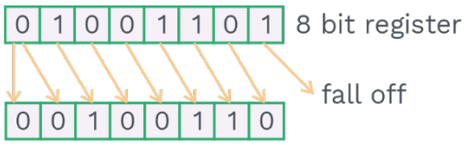


Fig. 2.53 Arithmetic Shift Right Operations II

If we consider these numbers like 2's complement number, then a negative number will remain negative, and a positive number will remain positive in the arithmetic right shift because MSB bit is not changed; hence the sign remains the same.

4) Arithmetic left shift:

It is equivalent to a logical left shift. So we don't need two different circuits and two different instructions for this operation.

5) Rotate right/left:

Here bit will not fall, rather it will enter into the register from another side , i.e. rotation.

i)

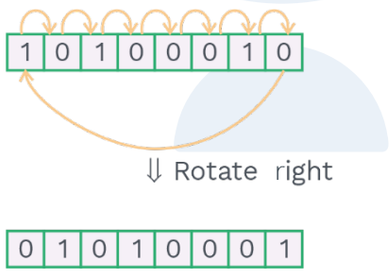


Fig. 1.54 ROR Operation

ii)

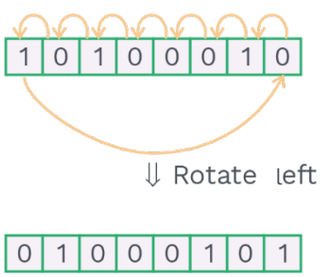


Fig. 2.55 ROL Instruction

6) Rotate right/left through carry:

In this operation, one more bit is appended, i.e. carry flag.

(Think it as ⑨ bit register)

c) Carry flag

Fig. 2.56 Register With CY Flag

i)

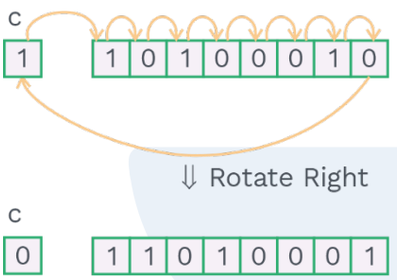


Fig. 2.57 RCR Operation

ii)

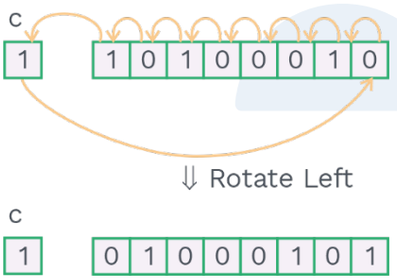


Fig. 2.58 RCL Operation

Chapter Summary

Types of CPU organisation:

1) **Stack CPU:** ALU operations are performed only on stack data.
2) **Accumulator CPU:** ALU first operand required in an accumulator.
3) **General register CPU:** Architecture is of two kinds:
   a) Register to memory reference: ALU first operand is always required in register, and the second operand can be in memory/register.
   b) Register to register: Both operands are required in a register.

Instruction format:

- An instruction format defines layout of bits of an instruction in terms of its constituent parts.

Elements of machine instruction:

1) Operation Code
2) Source Operand Reference
3) Result Operand Reference
4) Next Instruction Reference

Number of addresses:

1) Zero address instruction
2) One address instruction
3) Two address instruction
4) Three address instruction

Addressing modes: Shows the location of a required object on computer.

Sequential control flow AMs:

1) Register AM
2) Implied/Implicit AM
3) Immediate AM
4) Direct/Absolute AM
5) Indirect AM
6) Indexed/Base–Indexed AM
7) Indirect Indexed AM
8) Auto–Indexed AM

Transfer Flow of Control flow AMs

1) Relative/PC – relative Addressing Mode
2) Based/Base Register Addressing Mode

Chapter Summary

Expand–Opcode technique:
The address field of primitive instruction is joined with the free opcodes to generate derived opcodes.

Performance measuring of CPU:
1) Execution Time
2) Throughput
3) speedup
Amdahl's law:

$$= \frac{1}{1 - (\text{Fraction})_{\text{enhanced}} + \frac{(\text{Fraction})_{\text{enhanced}}}{(\text{Speedup})_{\text{enhanced}}}}$$

Processor clock cycle time and frequency:
Clock cycle time: Interval between 2 consecutive rising or falling edges.

$$\textbf{Frequency} = \frac{1}{\text{clock cycle time}}$$

CISC architecture and RISC architecture:
Key difference:

S. No.	RISC	CISC
1)	Few instructions	Many instructions
2)	Highly pipelined	Less pipelined or not pipelined
3)	Fixed format instruction	Variable format instruction
4)	Uses hardwired control unit	Uses microprogrammed control unit

Register–organisation in RISC:

Total number of registers = $\boxed{g + w(l + c)}$

3 ALU, Data-path & Control Unit

Introduction:

- A computer is used for the execution of the program, which consists of a set of instructions stored in the memory.
- The actual work of executing the instructions of the program is done by the processor(CPU).
- To process an instruction, CPU uses three internal components named as:
  1) Registers
  2) ALU (arithmetic and logic unit)
  3) Control unit (CU)

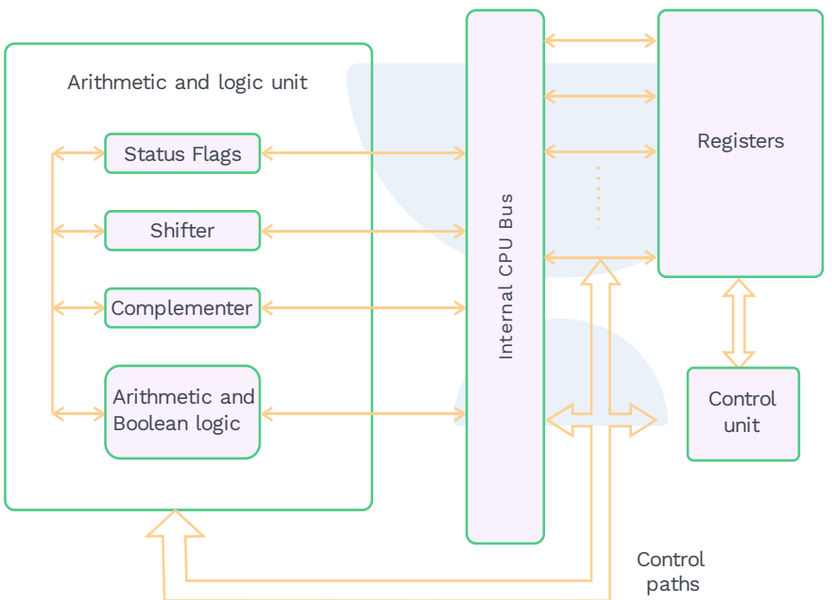


Fig. 3.1 Internal Structure of CPU

1) Register:

- A register is a storage component present inside the CPU which is used to accept, store and transfer the data and instructions used by the CPU.
- Registers are very fast computer memory units because they are made up of a group of flip flops and gates.
- Flip Flops are used to store the binary information, and gates control the transfer of information in and out of registers.

2) ALU (arithmetic and logic unit):

- The arithmetic and logical operations on data items inside the computer are performed by ALU.

- All the other components like control unit, registers, memory, and I/O bring the data for processing into the ALU and then take the result back out.
- Data to be processed is forwarded to registers and ALU, and the result of ALU operation is stored in specified registers.
- Based on the computation result, the ALU can also set flags.
  For example, if the result of an ALU operation which is to be stored in a register exceeds the length of that register, then overflow is set to 1.
- The control unit sends the signals that control the operation of the ALU and also controls the movement of the data into and out of the ALU.

Fig. 3.2 ALU Inputs and Outputs

3) Control unit (CU):

- Control unit is that component of the processor that causes things to happen, i.e. it directs the operation of the processor.
- Control unit converts the information that is being received into the control signals and then transfers the control signals to the central processor.
- This processor then tells the hardware what operations are to be performed based on the control signals.
- Control unit consists of inputs such as instruction registers, flags and control signals from external sources (e.g. interrupt signals).

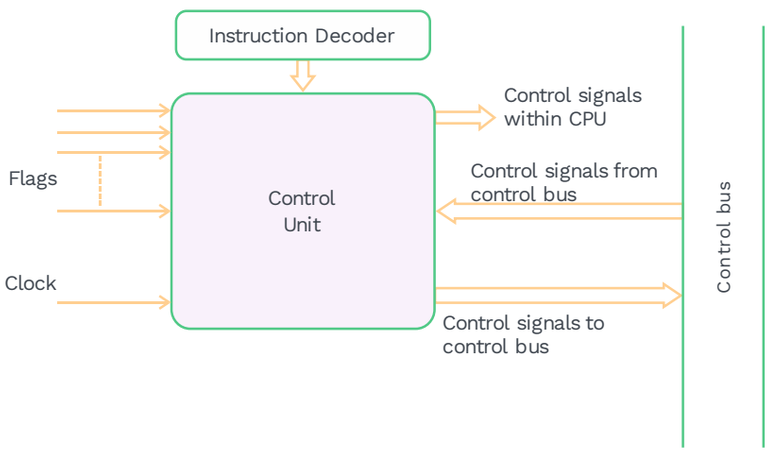


Fig. 3.3 Block Diagram of Control Unit

Control unit functions:

- In order to exchange data between memory and I/O modules, control signals are being issued by the control unit external to the processor.
- Control signals internal to the processor are being issued by the control unit for ALU to perform the specified operation, move the data between the registers and to regulate other internal operations.
- Control unit handles tasks such as fetching, decoding, executing and storage of results of the micro-operations.
- It regulates that the data inside the processor will be executed in what sequence and will be stored in which register, i.e. it directs the data flow sequence inside the processor as well as outside the processor.
- The two basic tasks performed by the control unit are:
  1) **Sequencing:** The control unit makes the processor move through a series of micro-operations in the proper sequence based on which the program is being executed.
  2) **Execution:** The control unit is also responsible for each micro-operation to be performed.

Micro-operations:

- Execution of a program comprises sequential execution of instructions.
- Execution of each instruction is done during an instruction cycle which is made up of shorter sub cycles (e.g. fetch cycle, execute cycle, interrupt cycle).
- Execution of each sub cycle consists of one or more shorter operations, called micro-operations.
- Micro-operations are the functional or atomic operations of a processor that finishes its execution in 1 clock cycle.

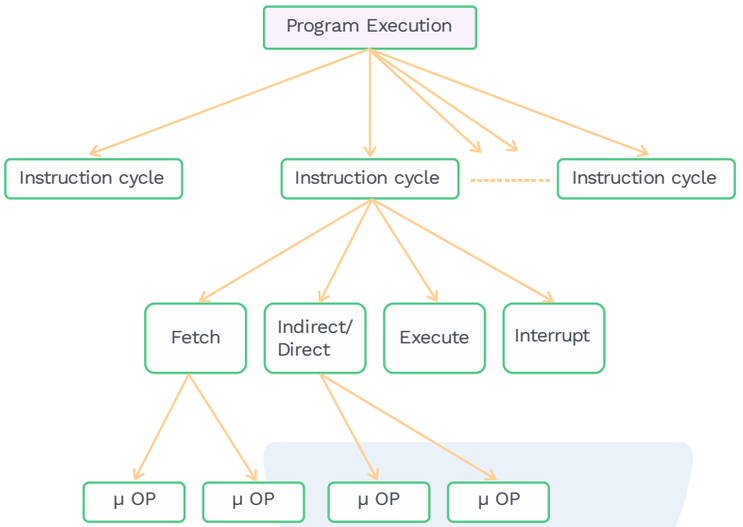
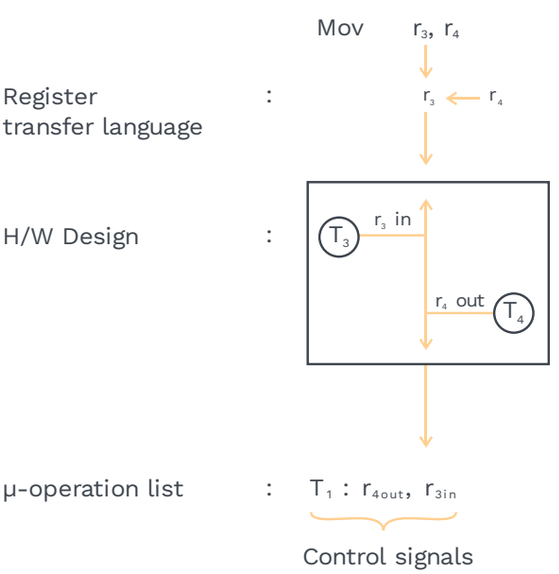


Fig. 3.4 Constituent Elements of Execution of Program

Note:

Register to Register transfer operation is a μ-operation from the user's point of view.

Example: Following is the micro-operation to move the content from one register to another which will be completed in one clock cycle.



Micro-instruction:

Note:

Micro-instruction designed with one micro-operation or more than 1 μ-operation, will always takes 1 cycle to complete.

Definitions

Micro-instruction is a collection of micro-operations.

Micro-program:

- Sequence of μ-instructions which are used to execute a task in the hardware is known as μ-program.

Example: In order to execute an instruction, the instruction must be brought to the processor from the memory. To complete this task following micro-program is used:

t_1: MAR ← (PC)

t_2: MBR ← Memory
PC ← (PC) + step size

t_3: IR ← (MBR)

The above micro-program consists of four micro-instructions, which takes three clock cycles.

The fetch cycle:

- Every instruction is placed in the main memory.
- In order to execute an instruction, it must be brought into the CPU from the main memory.
- Thus, the fetch cycle appears at the starting of every instruction cycle and causes an instruction to be fetched from memory.
- Following are the sequence of events for the fetch cycle:

1) Program Counter(PC) consist of the address of the next instruction which is to be executed, and this address is moved to the memory address register (MAR).

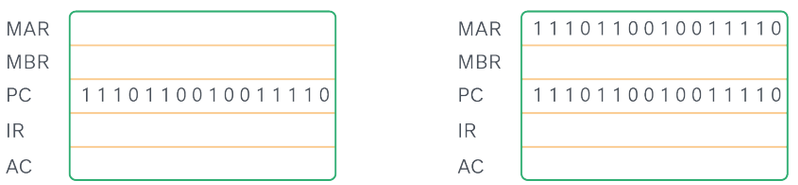


Fig. 3.5 Orientation of PC and MAR

2) Next step is to bring the instruction from the memory.
The MAR contains the instruction address, which is placed on the address bus and control unit sends the READ command on the control

bus, and the fetched instruction appears on the data bus, which is then copied to memory buffer register (MBR)

MAR	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0
MBR	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0
PC	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1
IR	
AC	

Fig. 3.6 Orientation of MAR and MBR

Since read from memory and incrementing PC to the instruction size does not affect each other, so in this step, PC (program counter) can also be incremented.

3) Now, the data is placed from the MBR to IR.

MAR	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0
MBR	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0
PC	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1
IR	1 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0
AC	

Fig. 3.7 Orientation of MBR and IR

Thus, the fetch cycle has four micro operations, which can be completed in three clock cycles where each micro operation involves movement of data in and out of registers.

Hardware design:

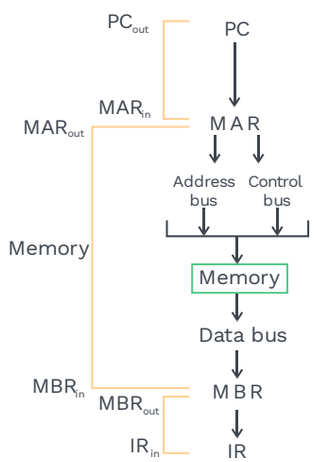


Fig. 3.8 Hardware Design of IF Cycle

IF μ-program:

t_1: MAR ← (PC)

t_2: MBR ← Memory

PC ← (PC) + I

t_3: IR ← (MBR)

Where I is the instruction length and t_1, t_2, and t_3 are the time units.

First time unit (t_1): MAR_{in}, PC_{out}

Second time unit (t_2): MAR_{out}, Memory, MBR_{in}

PC_{out}, Increment, PC_{in}

Third time unit (t_3): MBR_{out}, IR_{in}

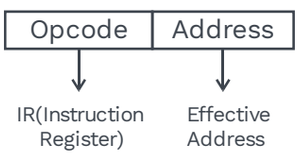
The operand fetch cycle:

- After the instruction is fetched, the source operands must also be fetched.
- If the instruction specifies a direct/indirect address, then a direct/indirect cycle must precede accordingly before the execute cycle.

a) Direct addressing mode:

Let us assume a one-address instruction format with direct addressing mode.

Instruction:



- During the instruction fetch, the instruction is placed in the IR(Instruction Register).
- The address field of the instruction is moved to the memory address register (MAR).
- Then, this address is used to fetch the operand from the memory that is stored in the corresponding address.
- The operand which is fetched from memory is kept on the data bus and is then placed in the memory buffer register (MBR).
- The memory buffer register (MBR) places the operand in the accumulator register for further processing.

Hardware design:

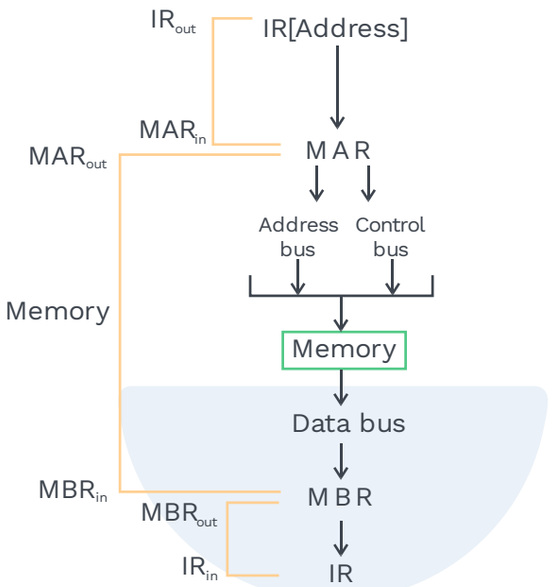


Fig. 3.9 Hardware Design of Direct Operand Fetch Cycle

Micro-program of direct operand fetch cycle:

t_1: MAR ← IR[Address]

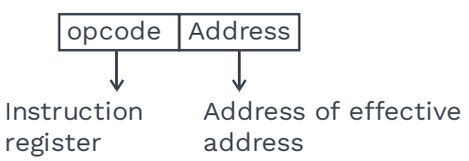
t_2: MBR ← M[MAR]

t_3: AC ← MBR

b) Indirect addressing mode:

An example of a one-address instruction format that uses indirect addressing mode is considered for the detailed analysis below.

Instruction:



- The address field of the instruction placed in the instruction register (IR) is transferred to the memory address register (MAR).
- The operand's effective address is achieved by accessing the memory address mentioned in the address field of the given instruction format.
- The effective address(EA) which is being fetched is placed in the MBR and the address field of the instruction register is replaced by

the effective address(EA), so now, instead of an indirect address, the address field contains a direct address.

- Then again, the address field of IR is transferred to MAR, and this address is used to fetch the operand from the corresponding memory location.
- The fetched operand appears on the data bus and is then placed in MBR.
- Then, from the memory buffer register (MBR), the operand is placed in accumulator register(AC) for further processing / execution.

Hardware design:

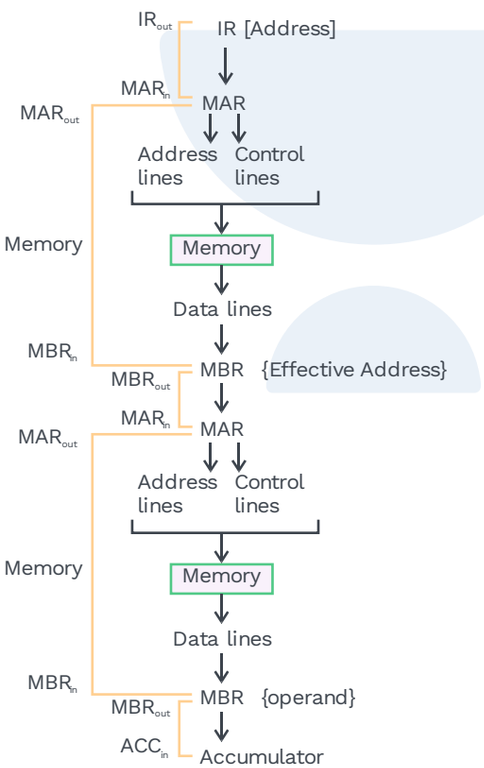


Fig. 3.10 Hardware Design of Indirect Operand Fetch Cycle

Microprogram of indirect operand fetch cycle:

t_1: MAR ← (IR (address))

t_2: MBR ← Memory

t_3: MAR ← MBR

t_4: MBR ← Memory

t_5: Accumulator ← MBR

The execute cycle:

- The micro-operations in the execution cycle vary as they depend on the type of the opcode.
- Consider the following MUL instruction using direct addressing mode:

MUL R1, A

- It multiplies the contents of location A to the contents of register R1 and store the result in R1.

The following sequence of micro-operations occur:

1) The instruction register (IR) contains the MUL instruction. The address part of IR is loaded into the MAR.

2) Then, the memory location corresponding to the address in MAR is read.

3) Then, the data present in location A is transferred to MBR (memory buffer register).

4) The ALU then multiplies the contents of register R1 and MBR.

Hardware design:

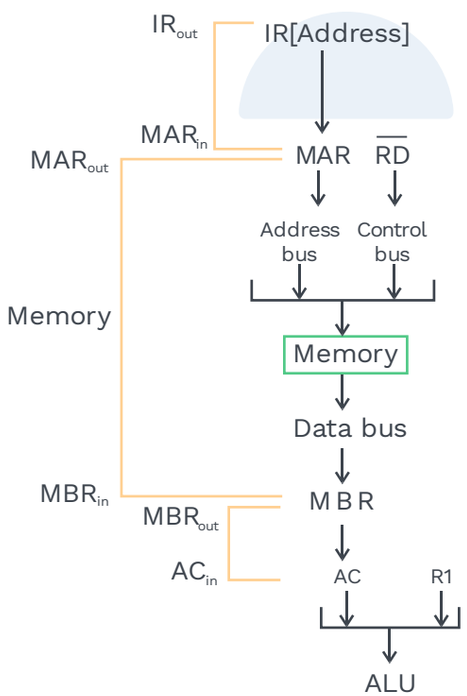


Fig. 3.11 Hardware Design of Execute Cycle

Micro-program:

t_1: MAR ← (IR Address)

t_2: MBR ← Memory

t_3: Accumulator ← MBR

t_4: R1 ← (R1) * (AC)

Write back micro-program:

- After the execution of the instruction, the result might be required to be stored in memory location.

The following sequence of micro-operation occurs:

1) After the execution, the result is stored in the register (accumulator), this result is transferred to the memory buffer register (MBR).

2) IR contains the instruction and the address field of the instruction, which contains the address of the memory location where the result needs to be stored is transferred to MAR.

3) The data present in MBR is stored in the memory location whose address is present in MAR.

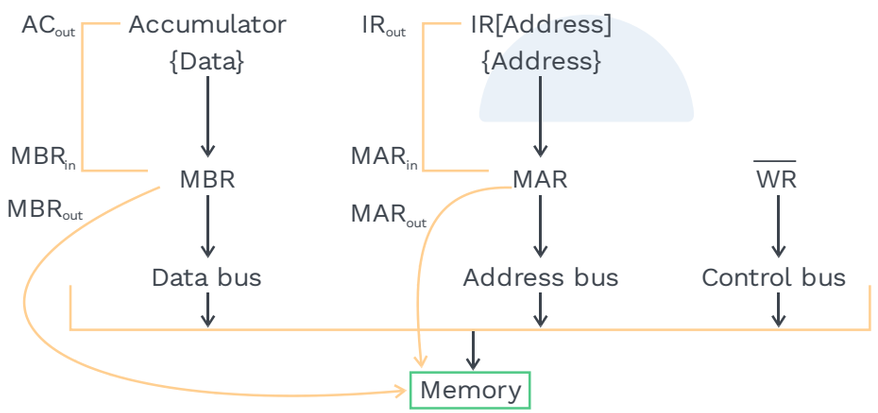


Fig. 3.12 Hardware Design of Write Back

Micro program:

t_1: MBR ← Accumulator

t_2: MAR ← IR [Address]

t_3: Memory ← MBR

The interrupt cycle:

- After the completion of execute cycle, it is checked whether any enabled interrupts have occurred or not.

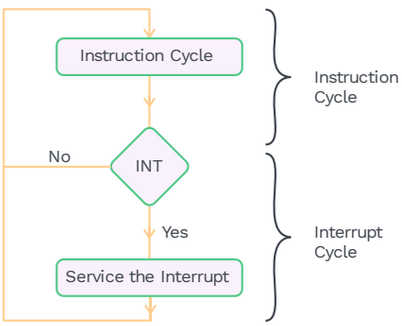


Fig. 3.13 Interrupt Servicing Mechanism

- If interrupts occur, the following sequence of micro-operations takes place:
  1) The next instruction address which is to be executed is present inside the PC. When an interrupt occurs, the address present inside the PC is sent to the memory buffer register(MBR) so that it can be saved for return from the interrupt.
  2) The memory address at which the contents of the PC are to be stored is loaded in MAR.
  3) The starting address of the interrupt processing routine is loaded on the PC.
  4) The last step is to load the contents of MBR, which contains the old value of PC, into the memory whose address is present in the MAR.

Hardware design:

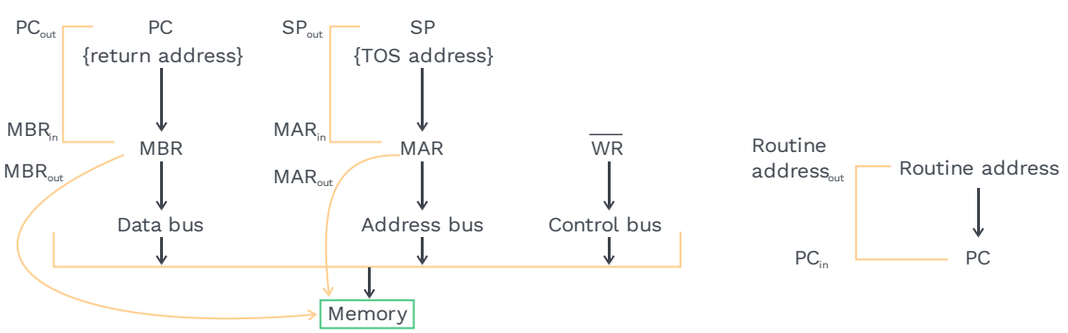


Fig. 3.14 Hardware Design of Interrupt Cycle

Micro-program:

t_1: MBR ← (PC)

t_2: MAR ← Save-address

PC ← Routine address

t_3: Memory ← (MBR)

PRACTICE QUESTIONS

Q1 **Consider a one-word long instruction:**
MOV [2000], @3000
If the processor fetches, decodes, operand fetches, executes and writes back the result of the instruction to the memory. Then the number of times the memory is referred by the processor?

Sol: Fetch: 1 memory reference will be required for fetching the instruction.
Decode: Since the instruction is 1 word long, no memory reference is required during decode.
Operand fetch: Since operand fetch, indirect addressing mode is used, so two memory references will be required.
1 MR for getting the effective address, i.e. address of the operand.
1 MR for getting the operand.
Execute: No memory reference for execute cycle because given instruction is MOV instruction only.
Write Back: One memory reference to write the operand at memory location [2000]
Total memory references = 1 + 2 + 1 = 4

Q2 **For the operand fetch using indirect addressing mode, how many number of cycles are required to bring the operand to the CPU?**

Sol: **Ans. 4**
The microprogram for operand fetch using indirect addressing mode is as follows:

t_1: MAR ← IR [Address]

t_2: MBR ← Memory

Thus, in these two cycles effective address, i.e., the address of the operand, is brought to the CPU.

t_3: MAR ← MBR

t_4: MBR ← Memory

These two cycles are required to bring the operand from memory to the CPU. So, total of 4 cycles is required.

Q3 **The fetch cycle consists of four micro-operations, and these four micro-operations can be completed in three units of time_____.**
Which among the four micro-operations can be completed in a single time unit in the fetch cycle?

I) MAR ← (PC)

II) MBR ← Memory

III) IR ← (MBR)

IV) PC ← (PC) + Instruction size (I)

a) I and II can be completed in a single time unit.
b) II and IV can be completed in a single time unit.
c) III and IV can be completed in a single time unit.
d) I and IV can be completed in a single time unit.

Sol: **Ans. b), c)**

MBR ← Memory

PC ← (PC) + instruction size (I)

Both these micro-operations can be completed in a single cycle because these two actions, read a word from memory and add instruction size (I) to PC, do not interfere with each other.

IR ← (MBR)

PC ← (PC) + instruction size (I)

Both these micro-operations can be completed in a single cycle because they both do not interfere with each other.
Conflicts occur when reading and writing are done on the same register in single time unit.

Bus architecture:

- Two or more devices in a computer are connected through a common transmission medium known as a bus.
- There are many devices that are connected to the bus, and if any device sends the signal, it can be received by all the other devices connected to the bus.
- Signals that represent binary 1 and binary 0 are transmitted via bus, which contains multiple communication lines.

Previous Years' Question

Consider the following sequence of micro-operations, **(GATE CS 2013)**

MBR ← PC
MAR ← X
PC ← Y
Memory ← MBR

Which one of the following is a possible operation performed by this sequence?

a) Instruction fetch
b) Operand fetch
c) Conditional branch
d) Initiation of interrupt service

Sol: d)

System bus:

- The major units of a system, like memory, CPU and I/O are connected by a system bus.
- Typically, the system bus contains the following lines:
  - **a) Data lines:** Data is transferred between various system modules through data lines.
    Data lines altogether are known as the data bus.
  - **b) Address lines:** The address of the data which is present on the data bus is contained by address lines.
    Address lines altogether are known as address bus.
  - **c) Control lines:** The control lines are bearers of control signals that effectively establish the communication of control commands and timing signals among different system components.
- The validity of address and data information is specified by timing signals.
- Control signals describe the operation to be performed like memory write, memory read, I/O write, I/O read etc.

Note:

1) The word length of the CPU can be described based on the width of the data bus.
2) The memory capacity of the system can be described based on the width of the address bus and data bus.

Bus architecture:

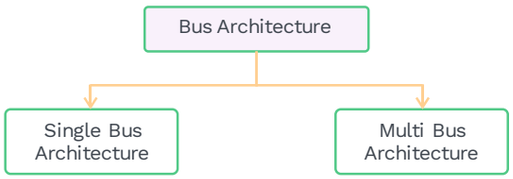


Fig. 3.15 Bus Architecture

1) Single bus architecture:

- To form an operational system, the functional individual units must be connected in some organized way.
- The single bus architecture is the easiest way to interconnect all the functional units.

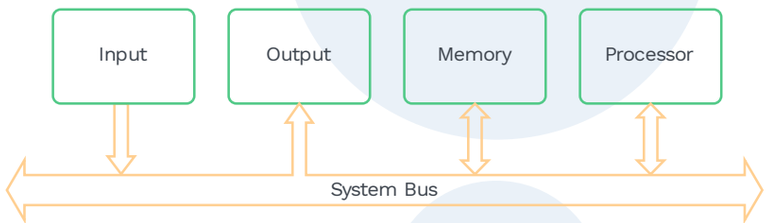


Fig. 3.16 Single-Bus Structure

- Both the instructions and data are transferred through this single bus.

Advantages:

- The single bus architecture is not an expensive design.
- Less number of registers are associated.
- Flexible for attaching new peripheral devices.

Disadvantages:

- Only single transmission can be done by the bus at a given point in time, i.e. only two units can actively use the bus at any given time.
- Less concurrency in operations.
- Requires a greater number of cycles for execution, thus making execution of process slow.

Multi bus architecture:

- The performance of single bus architecture is affected if many devices are connected to the bus.
- With the increase in the count of the components connected to the bus architecture, propagation delay increases. Therefore, devices will take a large amount of time to synchronize the use of the bus.

- Also, when the aggregate data transfer demand becomes so large that it reaches the capacity of the bus and thus bus may become a bottleneck.
- The above problems may not always be overcome by single bus architecture.
- This problem can be solved by using multiple buses, generally laid out in a hierarchy.
- Multiple buses allow the devices to work simultaneously, thus improving the CPU's speed.

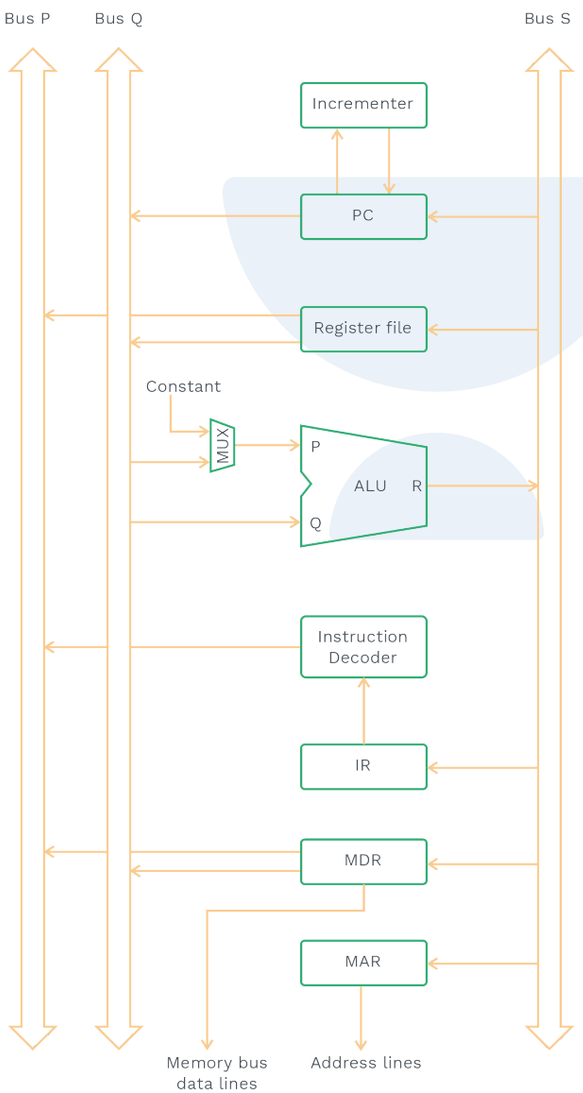


Fig. 3.17 Three Bus Organisation of Data Path

- Buses P and Q are used to transmit the source operands to the P and Q inputs of ALU, where an arithmetic or logic operation can be performed.
- The result is transmitted to the destination over bus S.
- Whenever required, the ALU can forward one of its two input operands, not modified, to bus S.

Advantages:

- The number of cycles required for the execution of the process is less.
- More concurrency in operations.

Disadvantage:

- Multibus architecture is an expensive design.

Control signals:

- Every micro-operation is performed by activating some control signals.
- Control signals are put in directly as binary inputs to the independent logic gates.
- Control signals can be used to activate an ALU function, a data path or other external interfaces.
- The CPU consists of some general purpose and some special purpose registers.
- All the registers are connected to a common data path known as BUS.
- For transferring the information from one register to another, each register has two control signals R_{in} and R_{out}.
- When the content of the bus is loaded into the register R, then R_{in} must be set to 1.
- When the content from the register R is loaded into the bus, then R_{out} must be set to 1.

- Let us consider Fetch Cycle:

Following is the micro-program for the fetch cycle:

t_1: **MAR ← (PC)**

The control unit turns on the control signals that unlock the gates between the PC and MAR permitting the bits in PC onto MAR.

t_2: **MBR ← Memory**

PC ← (PC) + 1

For this micro-operation control unit activates the following control signals:

1) A control signal that unlocks the gates and permits the contents present in MAR onto the address bus.

2) A control signal to read the memory location and transfer the data onto the data bus.

3) A control signal that unlocks the gates and permits the contents on the data bus to be transferred onto the MBR.

4) A control signal that increments the PC value by the instruction length and stores it on PC.

t_3: IR ← (MBR)

A control signal that allows the contents of MBR to be transferred to IR.

Fig. 3.18 Fetch Cycle

- These control signals can be activated by:
  - **a)** Hardwired control unit design
  - **b)** Micro-programmed control unit design.

Control unit design:

- The implementation of the control unit requires the following information:
  - **1)** Number of instructions implemented in base hardware.
  - **2)** Number of micro-operations required for each instruction.
  - **3)** The control signals which are required for micro-operation.
- The control unit in the CPU is responsible for decoding the instructions and generating the control signals corresponding to the sequence of micro-instructions involved in the instruction.
- There are two approaches to implement control unit:
  - **1)** Hardwired control unit
  - **2)** Micro-programmed control unit

1) Hardwired control unit:

- The hardwired control unit is a sequential state machine to generate the specific fixed sequences of control signals.
- In this design, control signals are expressed in the sum of product expression format.
- Gates ,decoders, flip flops and other digital circuits are used to implement control logic.
- To generate the control signals, fixed logic circuits that directly correspond to Boolean expressions are used.

Advantages:

- Since logic gates and flip flops are used to design control units, so it is the fastest control unit.

Disadvantage:

- It is not flexible because even very small changes in the Boolean expression requires redesign and reconnection of a logic circuit.

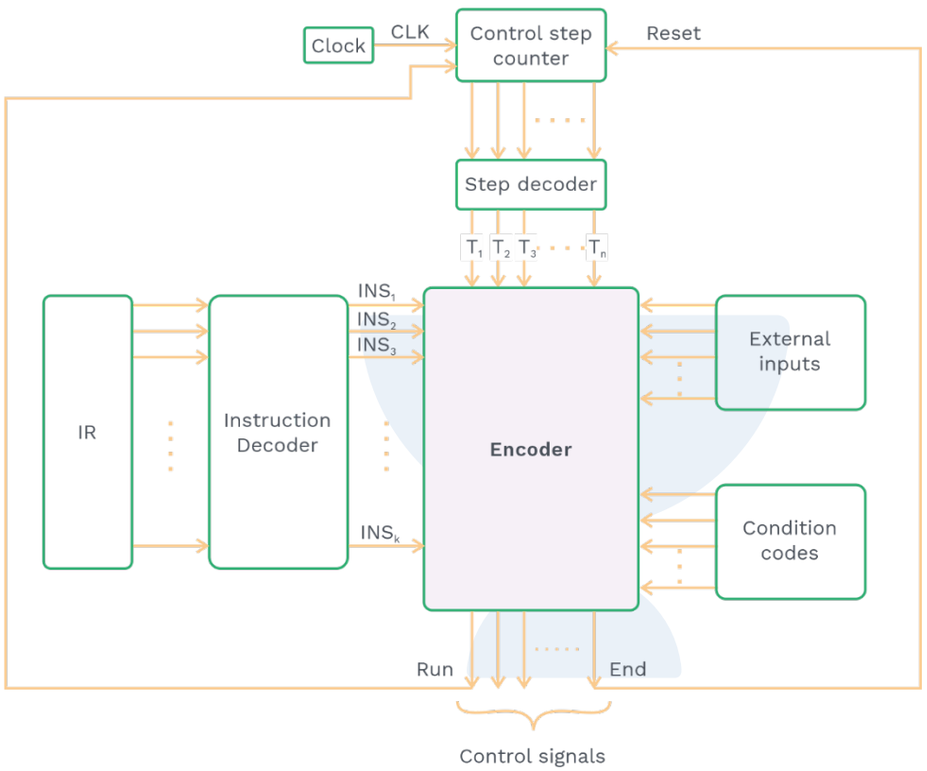


Fig. 3.19 Control Unit Organisation

- The blocks of decoder and encoder are combinational circuits that are used to produce the desired control outputs.
- In control sequence, a different signal line for every step is provided by the step decoder.
- While loading any instruction in IR (Instruction Register), one of the output lines INS_1 through INS_k are set to 1, and all other lines are set to 0.
- The input signals to the encoder block are combined to generate the individual control signals Y_{in}, PC_{out}, Add, End etc.

  eg. $Z_{in} = T_1 + T_6 \cdot ADD + T_4 \cdot BR + ...$

- The signal is asserted during time slot T_1 for all instructions, during T_6 for an ADD instruction during T_4 for an unconditional branch instruction and so on.

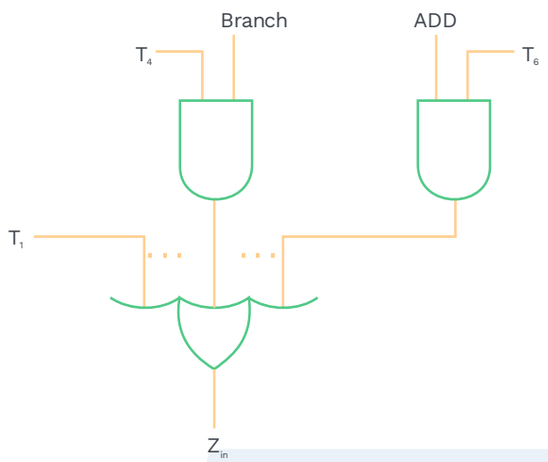


Fig. 3.20 Generation of Z_{in} Control Signal

Note:

Hardwired control unit is used in designing RISC architecture.

PRACTICE QUESTIONS

Q4 **Consider a system in which we have 4 control signals S_0, S_1, S_2, S_3 and 3 different instruction I_1, I_2, I_3 and 4 μ-operations T_1, T_2, T_3, T_4. Following are the control signals which are required for a particular instruction at different μ-operation timings:**

μ-operation/Instruction	I_1	I_2	I_3
T_1	S_0	S_2, S_3	S_0, S_1, S_2, S_3
T_2	S_0, S_2	S_3	S_1, S_3
T_3	S_3, S_2, S_0	S_0, S_3	S_1
T_4	S_0, S_1, S_2	S_3, S_2, S_1	S_2

The expressions for the control signals will be?

Sol: The expression for control signal S_0 will be,

$$S_0 = T_1(I_1 + I_3) + T_2(I_1) + T_3(I_1 + I_2) + T_4(I_1)$$

The expression for control signal S_1 will be,

$$S_1 = T_1(I_3) + T_2(I_3) + T_3(I_3) + T_4(I_1 + I_2)$$

The expression for control signal S_2 will be,

$$S_2 = T_1(I_2 + I_3) + T_2(I_1) + T_3(I_1) + T_4(I_1 + I_2 + I_3)$$

The expression for control signal S_3 will be:

$$S_3 = T_1(I_2 + I_3) + T_2(I_2 + I_3) + T_3(I_1 + I_2) + T_4(I_2)$$

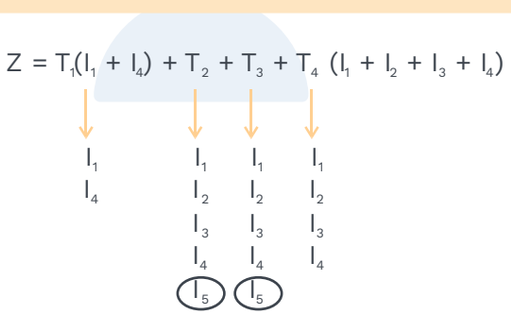
Q5 **Following is the expression for control signal Z, present in a hardware:**

$$Z = T_1(I_1 + I_4) + T_2 + T_3 + T_4\ (I_1 + I_2 + I_3 + I_4)$$

Given that the system has 5 different instructions I_1, I_2, I_3, I_4, I_5 and each instruction has 4 micro-operations T_1, T_2, T_3, T_4. During the execution of instruction I_5, in which micro-operation of the control signal is enabled in the hardware?

a) T_1 and T_2
b) T_2 and T_3
c) T_1, T_2, T_3 and T_4
d) None Sol.

Sol:



So, option 'b' is correct.

Previous Years' Question

A hardwired CPU uses 10 control signals S_1 to S_{10}, in various time steps T_1 to T_5, to implement 4 instructions I_1 to I_4 as shown below: **(GATE CS 2005)**

	T_1	T_2	T_3	T_4	T_5
I_1	S_1, S_3, S_5	S_2, S_4, S_6	S_1, S_7	S_{10}	S_3, S_8
I_2	S_1, S_3, S_5	S_8, S_9, S_{10}	S_5, S_6, S_7	S_6	S_{10}
I_3	S_1, S_3, S_5	S_7, S_8, S_{10}	S_2, S_6, S_9	S_{10}	S_1, S_3
I_4	S_1, S_3, S_5	S_2, S_6, S_7	S_5, S_{10}	S_6, S_9	S_{10}

Which of the following pairs of expressions represent the circuit for generating control signals S_5 and S_{10}, respectively?
($(I_j + I_k)T_n$ indicates that the control signal should be generated in time step T_n if the instruction being executed is I_j or I_k).

a) $S_5 = T_1 + I_2 \cdot T_3$ and

$S_{10} = (I_1 + I_3) \cdot T_4 + (I_2 + I_4) \cdot T_5$

b) $S_5 = T_1 + (I_2 + I_4) \cdot T_3$ and

$S_{10} = (I_1 + I_3) \cdot T_4 + (I_2 + I_4) \cdot T_5$

c) $S_5 = T_1 + (I_2 + I_4)T_3$ and

$S_{10} = (I_2 + I_3 + I_4) \cdot T_2 + (I_1 + I_3) \cdot T_4 + (I_2 + I_4) \cdot T_5$

d) $S_5 = T_1 + (I_2 + I_4) \cdot T_3$ and

$S_{10} = (I_2 + I_3) \cdot T_2 + I_4 \cdot T_3 + (I_1 + I_3) \cdot T_4 + (I_2 + I_4) \cdot T_5$

Sol: d)

Micro-programmed control unit:

- Control Memory is a permanent memory, i.e ROM.
- In this design, micro-programs are used to program the control memory.
- The behaviour of the control unit is depicted by the program present in the control memory.
- Thus, by simply executing that program, the control unit can be implemented.
- The set of micro-instructions is stored in the control memory.

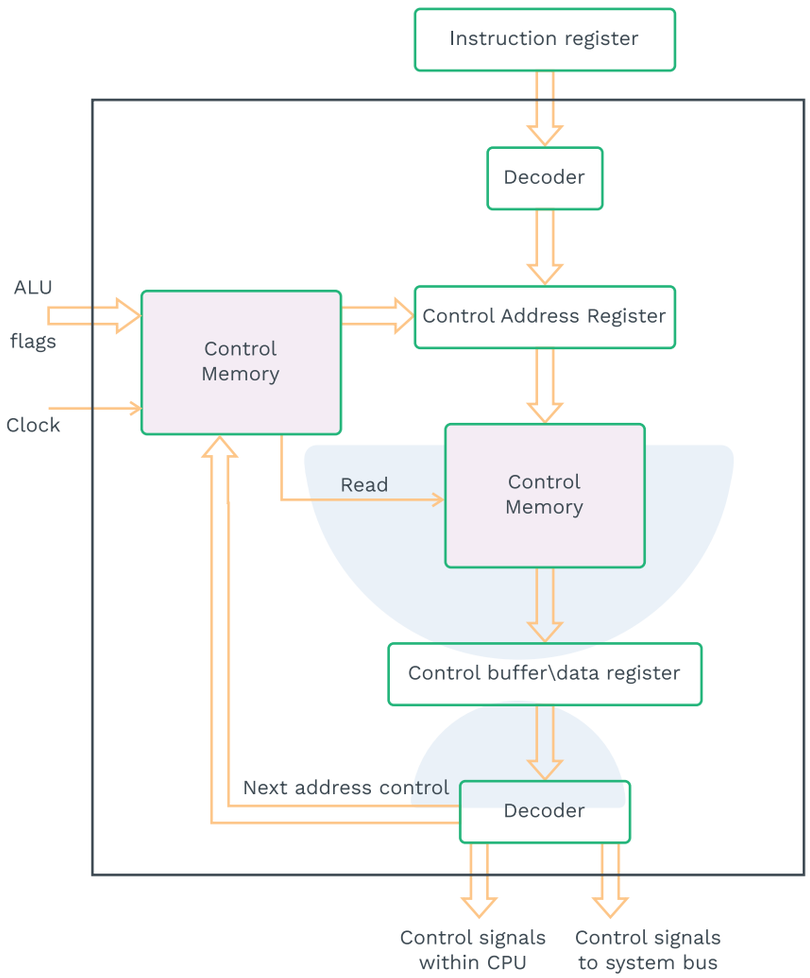


Fig. 3.21 Functioning of Microprogrammed Control Unit

- The address of the next micro instruction to be read is present inside the control address register (CAR).
- After reading micro instruction from the control memory, the micro instruction is transmitted to the buffer register known as a control data register(CDR).
- The sequencing unit loads the control address register and issues a read command.
- Following are the functions of the control unit:
  1) The READ command to the control memory is being issued by the sequencing logic in order to execute an instruction.
  2) The word whose address is given in the control register is read into the control buffer register.

3) Control signals and next address information are generated by the content of the control buffer register.
4) Based on the next address information present inside the CDR and ALU flags, a new address is being loaded by the sequencing the logic unit in the CAR.

- The upper decoder converts the opcode of IR into the control memory address.
- The lower decoder is not used for horizontal micro instructions, but for vertical micro-instructions.
- Control Memory is associated with CAR (Control Memory address register) and CDR (Control Memory data register) registers which are used to hold the control memory address and control memory content, respectively.

Hardware design:

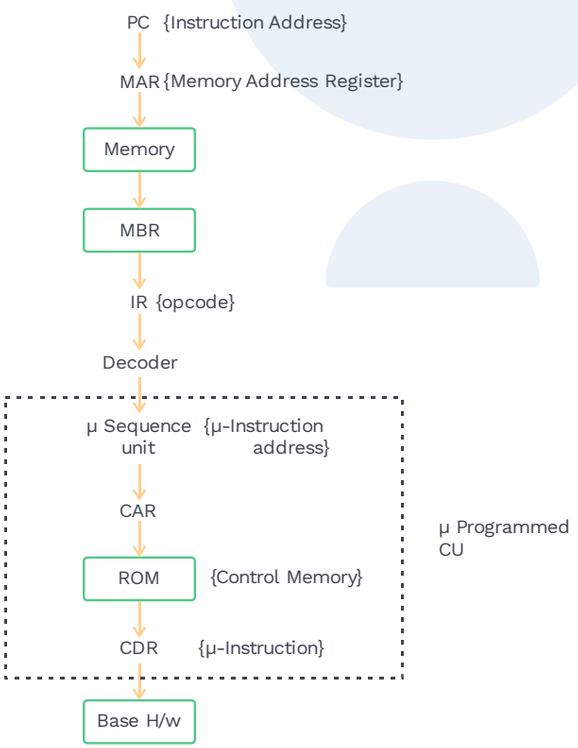


Fig. 3.22 Hardware Design implementing Microprogrammed Control Unit

Advantages:

- The use of micro-programming for implementing the control unit simplifies the design of control unit.

- It's implementation is cheaper and is less error-prone.

Disadvantages:

- Micro-programmed control unit is slower than hard-wired control unit of comparable technology.

Note:

The Micro–programmed control unit is used for designing CISC architectures.

Instruction format:

The μ-Instruction in control memory is stored in the following format:

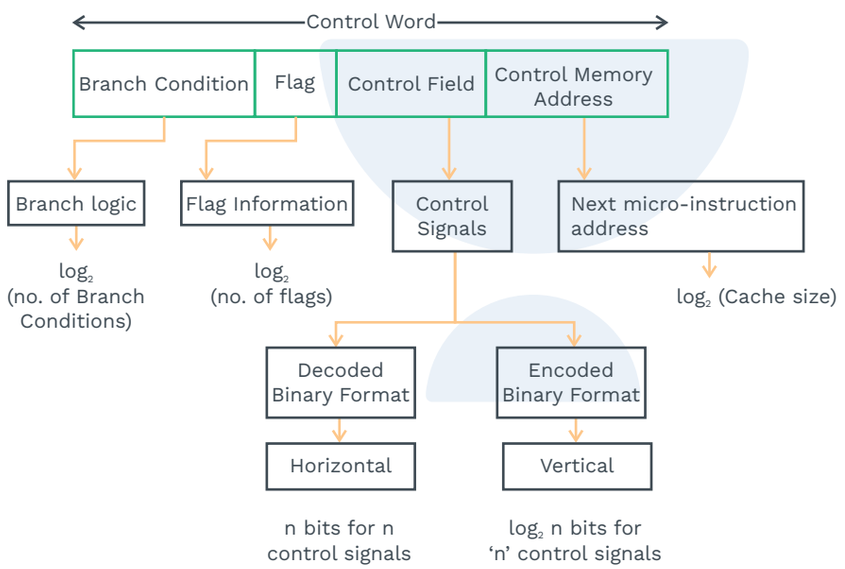


Fig. 3.23 Control Word Design

Note:

Fields (branch condition, flag, control memory address) are used for jumping conditions.

Example: JNZ4004

In this we require three things

1) Condition: Jump if non zero

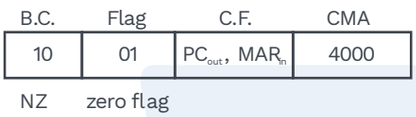
2) Flag: Zero flag

3) Address: 4004

Micro-instruction sequencing:
The micro-programmed control unit carries out the two important operations listed below as:

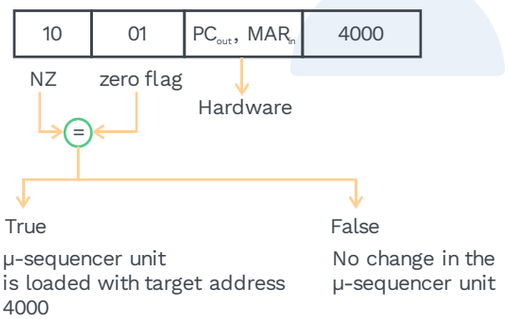
1) **Microinstruction sequencing:** It fetches the subsequent micro-instruction from the control memory.
2) **Microinstruction execution:** It is the procedure of generating the control signals to carry out the execution of the pre-fetched micro-instruction.

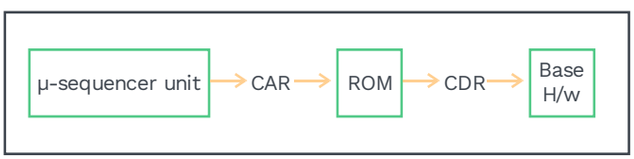
Example: Consider the following micro instruction with "Nonzero" branch logic and having target address of 4000.



Where {PC_{out}, MAR_{in}} are control signals.
Sol: This micro instruction will be executed as follows:

1) **Micro instruction sequencing:**
   Fetch the µ-instruction from the ROM
2) **Micro-instruction execution:**





µ-Programmed CU

Fig. 3.24 Execution of Micro-Instruction

PRACTICE QUESTIONS

Q6 **Vertical micro-programming, horizontal micro-programming and hardwired control are used for control unit design. Which among them has the highest and lowest operational speed, respectively?**

a) Highest: Hardwired control
b) Highest: Vertical micro-programming
c) Lowest: Vertical micro-programming
d) Lowest: Horizontal micro-programming

Sol: **a), c)**

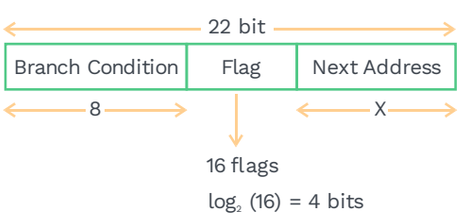
Highest: hardwired control
lowest: vertical micro-programming.

- In hardwired, the control logic is implemented using gates and flipflops. So, it is the fastest.
- In horizontal microprogramming, for each control signal, a separate bit is used.
- In vertical microprogramming, control signals are expressed in the form of encoded binary format.

Thus, a decoder is used.
So, for 'n' control signals, $\log_2 n$ bits are used, so it has the slowest operational speed.

Q7 **Consider a control memory unit in which a 22-bit micro-instruction is stored.**
The micro-instruction has three fields, the "next address" field of size 'X' bits, the "branch condition" field of 8 bits and the hardware containing 16 flags.
Then what is the size of the control memory?

Sol: **Ans.** 2816 bytes
μ-Instruction format:



Number of bits for next address field (X)
= 22 – (8 + 4)
= 10 bits
Size of control Memory = 2^{10} * 22 bits = (2^{10}) * 22/8 = 2816 bytes.

Horizontal micro-programmed control unit:

- Control signals in horizontal micro programmed control units are expressed in the decoded binary format.
- In this control unit design, each control signal is associated with 1 bit.
- Hence, 'n' number of control signals will require decoding of 'n' bits.

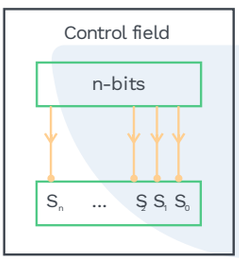


Fig. 3.25 Hardware Design of Microprogrammed Control Unit

- This micro-programmed control unit is fast, as all the control signals are enabled altogether.

Vertical micro-programmed control unit:

- Control signals in vertical micro programmed control units are expressed in encoded binary format.
- Hence, 'n' number of control signals will require encoding of '$\log_2 n$' bits.

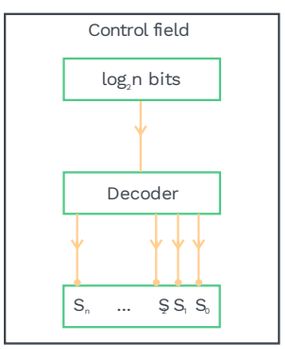


Fig. 3.26 Hardware Design of Vertical Microprogrammed Control Unit

- This micro-programmed control unit is slow since any one control signal can be enabled altogether.

Micro-instruction design using horizontal micro-programming:

- If for instruction 'I', T is one of the micro instructions having 4 control signals (S_0, S_1, S_2, S_3).
- Suppose the following control signals are enabled.
  T: {S_0, S_1, S_3}

µ-instruction design:

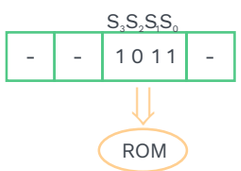


Fig. 3.27 Microinstruction Design Example

Execution:

- Fetch T from the control memory (ROM)

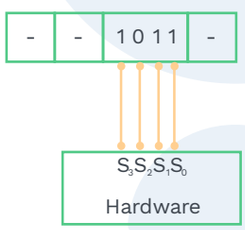


Fig. 3.28 Hardware Execution

Since no branch condition is given, so,
µ-sequencer unit → CAR (control address register)

Micro-instruction design using vertical microprogramming:

- If for instruction I, T is one of the micro-instruction having 4 control signals (S_0, S_1, S_2, S_3).
- If the following control signals are enabled

  T {S_0, S_1, S_3}

- µ-Instruction format:

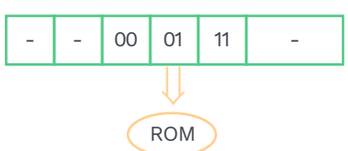


Fig. 3.29 Vertical Microinstruction Design Example

Execution:

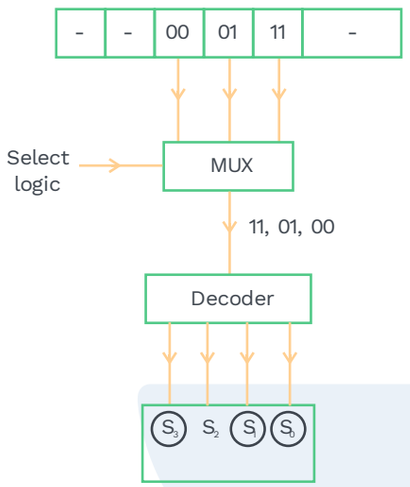


Fig. 3.30 Hardware Execution

Note:

1) Control field in the horizontal micro programming control unit is fixed.
2) Control field in the vertical micro programming control unit is variable, i.e. it depends on the number of function code.

	Horizontal micro-programming		Vertical micro-programming
1)	In horizontal micro-programming, the control signals are expressed in decoded binary format.	**1)**	In vertical micro-programming, the control signals are expressed in encoded binary format.
2)	High degree of parallelism is allowed, as in this none or more than one signal can be enabled at any instance of time.	**2)**	It allows a low degree of parallelism, as in this control unit none or only one signal can be enabled at any instance of time.
3)	It is faster as each control signal is denoted by a bit. So, an external decoder is not needed.	**3)**	It is slower as 'n' control signals require $\log_2 n$ bits. So, a external decoder is required to generate control signals.

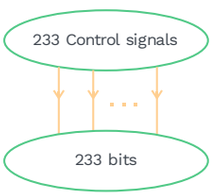
	Horizontal micro-programming		Vertical micro-programming
4)	It is less flexible	4)	It is more flexible.
5)	The control unit, using horizontal micro-programming is very complex and expensive to design.	5)	The control unit, using vertical micro-programming is comparatively easy and cheaper to design.
6)	It supports longer control word.	6)	It supports shorter control word.

Table 3.1 Comparison between Horizontal and Vertical Microprogramming

PRACTICE QUESTIONS

Q8 **Consider a hypothetical control unit, which has a total of 233 control signals. How many bits are required in control memory word for horizontal micro-programming and vertical microprogramming, respectively?**
a) 233, 233 b) 233, 1 c) 8, 8 d) 233, 8

Sol: **Ans. d)**
Horizontal micro-programming:
In horizontal micro-programming, for each control signal one bit is used.



Vertical micro-programming:
In vertical micro-programming, for n control signals, $\log_2 n$ control bits are required because in this design, control signals are expressed in an encoded binary format.

Therefore, $\log_2 233 = 8$ bits are required.

Q9 **Consider a micro-programmed control unit design which supports 7 groups of mutually exclusive control signals.**

Groups	Gr1	Gr2	Gr3	Gr4	Gr5	Gr6	Gr7
Control signals	2	10	4	1	18	23	6

How many more control bits are required using horizontal micro-programming over vertical micro-programming?

Sol: **Ans. 43**

For vertical micro programming:

Micro-Instruction format:

-	-	Gr1	Gr2	Gr3	Gr4	Gr5	Gr6	Gr7	-
		$\log_2 2$	$\log_2 10$	$\log_2 4$	$\log_2 1$	$\log_2 18$	$\log_2 23$	$\log_2 6$	

Total control bits required = 1 + 4 + 2 + 1 + 5 + 5 + 3.
= 21 bits

[For horizontal micro-programming]

Micro instruction format:

-	-	Gr1	Gr2	Gr3	Gr4	Gr5	Gr6	Gr7	-
		2	10	4	1	18	23	6	

Total bits required = 2 + 10 + 4 +1 + 18 + 23 + 6
= 64 bits

So, horizontal micro programming requires 64-21 = 43, more bits.

Previous Years' Question

Consider a CPU where all the instructions require 7 clock cycles to complete execution. There are 140 instructions in the instruction set. It is found that 125 control signals are needed to be generated by the control unit. While designing the horizontal micro-programmed control unit, a single address field format is used for branch control logic. What is the minimum size of the control word and control address register?

a) 125, 7 **b) 125, 10** **c) 135, 9** **d) 135, 10**

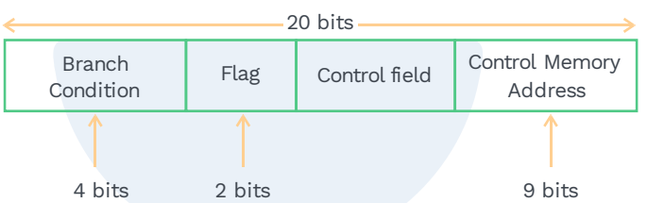
Sol: d) **(GATE CS 2008 (IT))**

Q10 **Consider a hypothetical control unit, which supports 512 bytes of the control word memory. If the control word is 20 bits long and hardware contains 4 flags and supports 16 branch conditions. What is the maximum number of control signals that can be generated at a time using horizontal micro-programming and vertical micro-programming, respectively?**

a) 1, 1 **b) 32, 32** **c) 32, 1** **d) 1, 32**

Sol: **Ans. c)**

μ-instruction design:



Control word memory = 512 bytes
So, control memory address field requires = 9 bits
Number of flags = 4
So, number of flag bits required = 2
Number of branch conditions = 16
So, number of branch condition bits required = 4
Number of bits required for control field = 20 – (4 + 2 + 9) = 5
Therefore, the number of control signals that can be generated = 2^5 = 32
In horizontal micro-programming, for each control signal, 1 bit is there,
So, all control signals can be activated at a time.
∴ Maximum degree of parallelism for horizontal microprogramming = 32
In vertical micro programming only 1 control signal can be activated at a time.
∴ Maximum degree of parallelism for vertical micro-programming = 1
∴ Option c) is correct.

Chapter summary

- **Introduction:** To process the instructions, the CPU uses three components:
- Registers
- ALU (arithmetic and logic unit)
- Control unit (CU)
- **Control unit functions:** Two basic tasks performed by the control unit are:
- Sequencing
- Execution
- **Micro-operations:** Micro-operations are functional or atomic operations of a processor that completes its execution in 1 cycle.
- **Micro-program:** Sequence of micro-instructions used to execute a task.

It includes the following cycles:

1) Fetch cycle
- Instruction Fetch
- Operand fetch

2) Execute cycle

3) Write back

4) Interrupt cycle

- **Bus architecture:** To form an operational system, functional individual units must be converted in some organized way:
  1) Single bus architecture
  2) Multi bus architecture
- **Control unit Design**
  1) Hard wired control unit
  2) Micro programmed control unit
- **Hard wired control unit:** Control logic is implemented with logic gates, flip-flops and other digital circuits
- **Micro-programmed control unit:** In this micro-program is used to program control memory.
  1) Horizontal micro-programmed control unit.
  2) Vertical micro-programmed control unit.

4 Pipelining

4.1 FUNCTION OF THE PIPELINE

Definition

Accepts the new input at one end before the previous accepted input appears as an output at the other end.

- Pipeline uses decomposition technique, it means problem statement is divided into independent subproblems, assign them the independent hardware, later connect the hardware in a pipelining sequence, i.e. first hardware unit output is connected as an input to second hardware unit, and so on.
- The definition states that, insert the new input into the pipeline before computation of an old input, so new input is executed along with the old input called as overlapping execution. Here, we can mention input is referring to as an instruction because use of input is not that convincing.
- Overlapping execution sequence is described using the space-time diagram.
- Successful characteristic of pipelining is, in every new cycle, new instruction is inserted into the pipeline.

So, CPI = 1 (clocks per instruction)

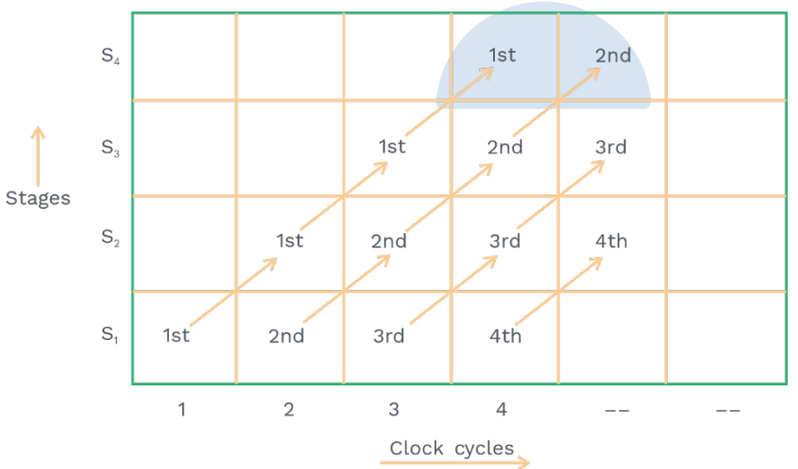


Fig. 4.1

Note:

In the non-pipeline system, new input is inserted after the completion of old input.

Pipeline layout:

- Every pipeline has two ends, known as input and output end. Multiple pipes linking the two terminals are interconnected to satisfy the functionality of the pipeline. Each pipe in the pipeline is called as stage or segment.
- Interface register (buffer) is used between the stages to hold the intermediate output. It is also called as buffer, or latch or pipeline register.
- The respective stages and the interface latches are connected to a common clock. So, clock adjustment is very important in the pipeline design.

a) Uniform delay pipeline:

$$\text{Cycle time}\left(t_p\right) = \text{Stage delay}$$

b) Non–uniform delay pipeline:

$$\text{Cycle time}\left(t_p\right) = \text{Maximum}\left(\text{stage delay}\right)$$

c) If buffer delay present in the pipeline:

$$\text{Cycle time}\left(t_p\right) = \text{Maximum}\left(\text{stage delay} + \text{buffer delay}\right)$$

d) If skew time/setup time overhead is present in the pipeline design

$$t_p = t_p + \text{Overhead}$$

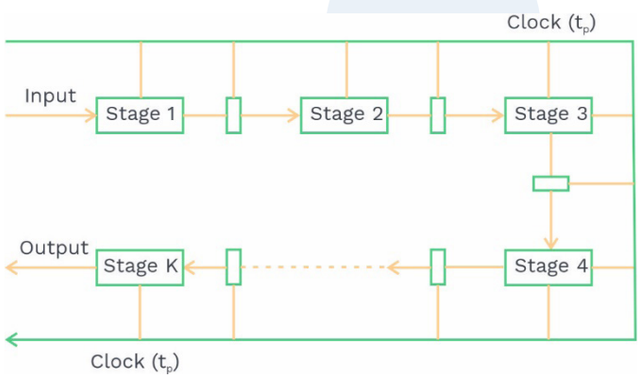


Fig. 4.2 Pipeline Layout

Performance analysis of pipeline:

- Consider a 'K' segment pipeline with the cycle time of 't_p' used to execute 'n' tasks.
- The first task in the pipeline is executed in a non–overlapping order, so it takes 'K' cycles to complete. The remaining 'n–1' tasks emerge from the pipeline at the rate of 1 task per cycle. So 'n–1' tasks take 'n–1' cycles to complete.

- So, total time required to complete 'n' tasks in a 'K' segment pipeline is:

$$ET_{pipe} = (K + n - 1)\,\text{cycles} = (K + n - 1)\,t_p$$

where ET_{pipe} = Execution time of pipeline.

- Now, consider a non-pipeline processor used to execute 'n' tasks in which each task takes 't_n' time to complete, so total time required to complete 'n' tasks in the non–pipeline is:

$$ET_{nonpipe} = n \cdot t_n$$

where $ET_{nonpipe}$ = Execution time of non–pipeline
t_n = One task execution time in non–pipeline

Performance gain of a pipeline is:

$$\text{Speedup}(S) = \frac{\text{Execution time of non-pipeline}}{\text{Execution time of pipeline}}$$

$$S = \frac{ET_{non-pipe}}{ET_{pipe}}$$

$$S = \frac{n \cdot t_n}{(K + n - 1)\,t_p} \quad (\text{when n is finite})$$

- With the increase in the count of the tasks, n attains much larger value than 'K–1', so 'n + K –1' approaches to 'n'.

Under this condition

$$S = \frac{t_n}{t_p}$$, when n approaches to infinity (∞)

- When the pipeline stages are perfectly balanced (uniform delay) then one task execution time in the non–pipelining system is equal to number of stages in the pipeline system, so t_n = K cycles:

$$t_n = K \cdot t_p$$

- Considering this condition:

$$\text{Speedup}(S) = \frac{K \cdot t_p}{t_p}$$

$$S = K$$

- If efficiency ($\eta = 100\%$), then speedup (S) is equal to number of stages in the pipeline (depth of pipeline).
- When the system is operating with 100% efficiency, then maximum speed up is possible. Maximum speed-up is equivalent to the count of the pipeline stages.

$$\eta_{pipe}\left(\text{Efficiency}\right) = \frac{\text{Speed–up}\ \left(S\right)}{\text{Number of stages}\left(K\right)}$$

$$\eta_{pipeline} = \frac{S}{K}$$

- $$\text{Throughput of pipeline} = \frac{\text{Number of tasks executed}}{\text{Total time taken to execute all the tasks}}$$

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

PRACTICE QUESTIONS

Q1 **Suppose there are 6 stages of a pipeline as given below:**

Stage number	Stage execution time (ns)
S1	10
S2	20
S3	15
S4	35
S5	10
S6	10

If the buffer delay is 30 ns, then calculate the execution time for 1000 instructions using pipelining.

a) 65.325 ns **b) 65.325 μs**

c) 48.975 μs **d) 58.965 ns**

Sol: **b)**

When various stages of pipeline have non–uniform clock time, we take longest clock time or maximum clock time.

Clock time = Maximum of (10 ns, 20 ns, 15 ns, 35 ns, 10 ns, 10 ns) + buffer delay

= 35 ns + 30 ns

= 65 ns

$T_{pipeline}$ = 65 ns

Execution time of pipeline ($ET_{pipeline}$) = (K + n–1) × $T_{pipeline}$

Where K = Number of stages = 6

n = Number of instructions = 1000

$ET_{pipeline}$ = (6 + 1000 – 1) × 65 ns

= $1005 \times 65 \times 10^{-9}$ sec

= $65,325 \times 10^{-9}$ sec

= 65.325×10^{-6} sec

= 65.325 μs (micro–second)

Q2 **Assume a pipeline P which operates at 3 GHz clock rate. It has a speedup factor of 10 and efficiency of 40%. Calculate the number of stages in the above pipeline.**

a) 10 **b) 25** **c) 15** **d) 30**

Sol: **b)**

$$\text{Efficiency of pipeline} = \frac{\text{speed up factor}}{\text{Number of stages}}$$

$$\text{Number of stages} = \frac{\text{speed up factor}}{\text{Efficiency}}$$

$$\text{Number of stages} = \frac{10}{0.4}$$

$$= \frac{10}{4} \times 10 = 25$$

Q3 **Consider a 3–stage pipeline having delays of 100 ns, 200 ns and 500 ns, respectively. The third stage is splitted into two stages of 250 ns and 250 ns respectively. Calculate the throughput increase/decrease in percentage. Assume ideal pipelined processor.**

a) 100% increase **b) 60% decrease**
c) 40% increase **d) 50% decrease**

Sol: c)

Ist Case:

Delays = 100 ns, 200 ns, 500 ns
Clock cycle time = Maximum (100, 200, 500) ns = 500 ns
Execution time for 1 instruction = CPI × clock cycle time
CPI = 1 (for ideal pipeline)
= 1 × 500 ns
= 500 ns
For execution of 1 instruction, it takes 500 ns.
1 instruction = 500 ns
1 instruction = 500×10^{-9} s

$$1 \text{ s} = \frac{1}{500 \times 10^{-9}} \text{instructions}$$

$$1 \text{ s} = \frac{10^9}{500} \text{instructions}$$

$$1 \text{ s} = \frac{1000}{500} \times 10^6 \text{ instructions}$$

Old throughput = 2 MIPS (million instructions per second)

2nd Case:

3rd stage is partitioned into two stages of delay 250 ns and 250 ns.
Delays = 100 ns, 200 ns, 250 ns, 250 ns
Clock cycle time = Maximum (100, 200, 250, 250) ns
= 250 ns
CPI = 1 (for ideal pipeline)
Execution time for 1 instructions
= CPI × clock cycle time
= 1 × 250 ns
= 250 ns
For execution of 1 instruction, it takes 250 ns.
250 ns = 1 instruction
250×10^{-9} s = 1 instruction

$$1\ s = \frac{1}{250 \times 10^{-9}} \text{instructions}$$

$$1\ s = \frac{1000}{250} \times 10^{6} \text{ instructions}$$

$$= 4 \times 10^{6} \text{ instructions}$$

New-throughput = 4 MIPS (million Instructions per second)

Percentage increase in throughput

$$= \frac{\text{New} - \text{Old}}{\text{Old}} \times 100\%$$

$$= \frac{4 - 2}{2} \times 100\%$$

$$= \frac{2}{2} \times 100\%$$

$$= 100\%$$

Q4 **Consider a pipeline having 5 stages with duration 10 ns, 30 ns, 45 ns, 80 ns and 35 ns. If latch (buffer) delay is 20 ns. Calculate the speedup of pipelined over non–pipelined processor.**

a) 2 **b) 2.5** **c) 1.5** **d) 3**

Sol: a)

Pipeline execution time

$ET_{pipeline} = (K + n - 1) \times T_{pipeline}$

$T_{pipeline}$ = Maximum delay among all stages + delay due to register (latch delay)

$T_{pipeline}$ = Maximum of (10, 30, 45, 80, 35) ns + 20 ns

= 80 ns + 20 ns = 100 ns

Non–pipeline execution time = (10 + 30 + 45 + 80 + 35) ns

= 200 ns

Note:

In non–pipeline, we don't consider buffer delays.

$T_{non-pipeline}$ = 200 ns

$$\text{Speed up ratio} = \frac{ET_{non-pipeline}}{ET_{non-pipeline}}$$

Here, we assume n (number of instructions) is very large, i.e. $n \to \infty$

$$\text{Speed-up ratio} = \frac{n \times T_{\text{non–pipeline}}}{(K + n - 1) \times T_{\text{pipeline}}}$$

$$= \lim_{n \to \infty} \frac{n \times 200}{(K + n - 1) \times 100}$$

$$= \lim_{n \to \infty} \frac{n \times 200}{n\left(\frac{K}{n} + 1 - \frac{1}{n}\right) \times 100}$$

$$= \frac{200}{100} = 2$$

Here K = number of stages in pipeline.

Q5 **Assume we have two pipelines P1 and P2, respectively. P1 has 6 stages, having execution time of 12 ns, 14 ns, 19 ns, 20 ns, 22 ns and 25 ns. P2 has 4 stages each having execution time of 10 ns. Calculate the time (in ns) that can be saved while using P2 pipeline over P1 pipeline, if 2000 instructions are executed.**

a) 15250 ns **b) 8765 ns** **c) 30095 ns** **d) 20070 ns**

Sol: c)

Consider pipeline P1

K = number of stages = 6

n = number of instructions = 2000

Clock cycle time for P1

T_{P1} = Maximum of (12, 14, 19, 20, 22, 25) ns

= 25 ns

Execution time for P1 = $(K + n - 1) \times T_{P1}$

$= (6 + 2000 - 1) \times 25$ ns

E_{P1} = 50, 125 ns

Now consider pipeline P2

K = Number of stages = 4

n = Number of instructions = 2000

Clock cycle time for P2 = 10 ns

T_{P2} = 10 ns

Execution time (E_{P2}) = $(K + n - 1) \times T_{P2}$

$E_{P2} = (4 + 2000 - 1) \times 10$ ns

= 20030 ns

Time saved = $E_{P1} - E_{P2}$

= 50, 125 – 20030
= 30095 ns

Q6 **If we have a non–pipeline processor which has cycle time of 40 ns and average CPI of 5.4. Assuming a 5-stage pipeline model, calculate the speedup of pipeline over non-pipeline processor.**
a) 5 **b) 4** **c) 4.5** **d) 6.5**

Sol: a)

We have 5 stages of equal delays in pipelined processors. Each stage will have clock cycle time $= \frac{40}{5}\text{ns} = 8\text{ns}$.

In the best case of pipeline, we have CPI = 1.
So in every cycle we will have one instruction as output, i.e. in every 8 ns.

$$\text{Speed up} = \frac{\text{Old cycle time}}{\text{New cycle time}}$$

$$= \frac{40\text{ns}}{8\text{ns}} = 5$$

Alternate solution:
Maximum speed up = number of stages = 5

Types of pipelines:
1) Linear pipeline:

- This type of pipeline contains forward connections only.
- Linear pipelines are synchronous pipelines.
- Linear pipeline latency is always 1.
- Latency means the clock cycle difference between two successive initiations in the pipeline.

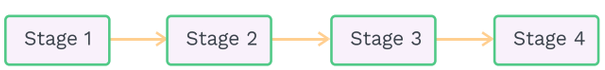


Fig. 4.3

2) Non-linear pipeline:

- This pipeline contains forward and backward connections, so the reservation table is used to process the inputs in the non-linear pipeline.
- Non-linear pipelines are asynchronous pipelines.

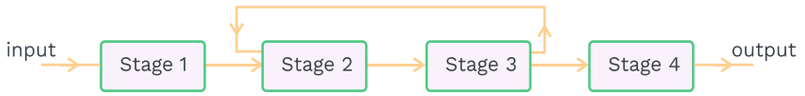


Fig. 4.4 Non-linear Pipeline

Note:

1) If there exist uniform delays among the pipeline stages, same amount of time will be required to execute one job in pipeline as well as non-pipeline layouts.
2) If there persist non-uniform delays among the stages, the time taken to accomplish the first job in the pipeline processor will be always greater than that in a non-pipelined architecture.

PRACTICE QUESTIONS

Q7 **A four-stage pipelining is given below:**

	S1	S2	S3	S4
I_1	2	1	1	3
I_2	1	3	2	1
I_3	1	1	3	1
I_4	2	1	1	1

What is the performance gain achieved by the above pipeline over non–pipeline execution?

a) 1.84 **b) 1.72** **c) 1.92** **d) 2.08**

Sol: c)

Pipeline execution:

	1	2	3	4	5	6	7	8	9	10	11	12	13
S4					I_1	I_1	I_1		I_2			I_3	I_4
S3				I_1			I_2	I_2	I_3	I_3	I_3	I_4	
S2			I_2	I_2	I_2	I_2	I_3	–	I_4	–	–		
S1	I_2	I_2	I_2	I_3	–	–	I_4	I_4					

$T_{pipe} \Rightarrow 13$ cycles

Non-pipeline execution:

$$T_{nonpipe} = \overset{I_1 \quad I_2 \quad I_3 \quad I_4}{(7 + 7 + 6 + 5)} \text{ cycles}$$

$T_{nonpipe} \Rightarrow 25$ cycles

$$\text{Performance gain} = \frac{T_{nonpipe}}{T_{pipe}}$$

$$= \frac{25}{13} = 1.92$$

Dependencies in the pipeline:

- Dependency is a major problem in the pipeline, it causes extra cycles.
- In a pipeline structure, any clock cycle deprived of new input initiation is called as extra cycle, also named as stall or hazard.
- When stall is present in the pipeline then CPI $\neq$ 1

1. 10 cycles $\rightarrow$ 10 inputs (CPI = 1)
2. 10 cycles $\rightarrow$ 7 inputs (CPI $\neq$ 1) (with 3 stalls)
3. 10 cycles $\rightarrow$ 9 inputs (CPI $\neq$ 1) (with 1 stall)

- There are three kinds of dependencies which are possible in the pipeline:
  1) Structural dependency
  2) Data dependency
  3) Control dependency

1) Structural dependency

- It occurs in the pipeline due to a resource conflict. Resource may be a register, memory or function unit (ALU).

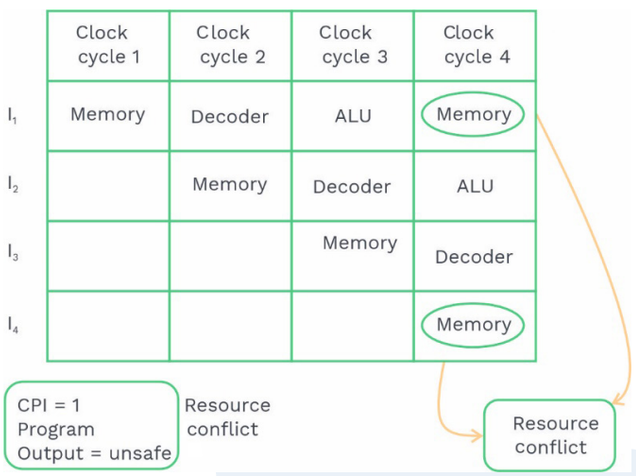


Fig. 4.5 Resource Conflict

- Conflict is an unsuccessful operation. To handle this condition, keep the instruction I_4 as waiting until the resource becomes available.
- The instruction stand by is referred to as stalls which is elaborated below:

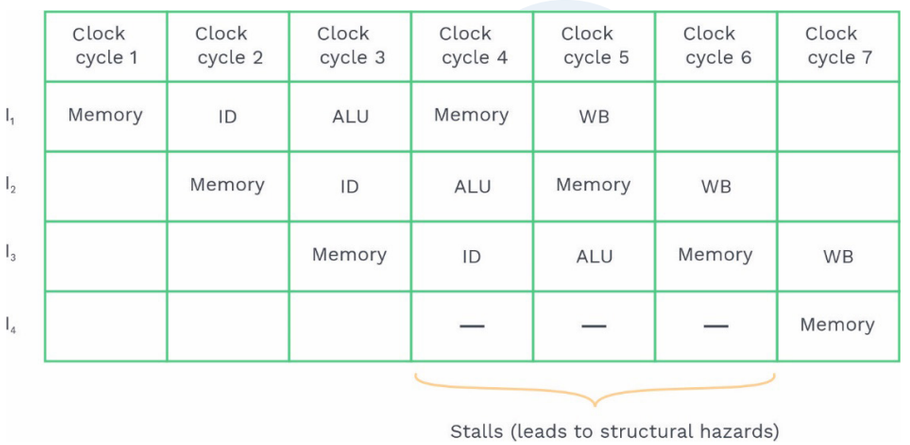


Fig. 4.6 Stalls created by Structural Dependency'

7 clock cycles – 4 inputs
= 3 stalls

$$\left\{\begin{matrix}\text{CPI} \neq 1 \\ \text{Program output = safe}\end{matrix}\right\}$$

- To minimize the structural hazards, hardware technique is used known as re-naming.
- In Fig. 4.5, instruction I_1 refers the memory in fourth stage of the pipeline to access the data.

- Simultaneously, instruction I_4 refers the memory in the first stage of the pipeline to access the instruction in the same clock cycle CC4.
- When the instruction and data, both are present in the same memory, then the above situation creates conflict.
- 'Renaming' mechanism isolates the memory into two independent sub-memory blocks: code memory (stores instructions) and data memory (stores data).
- Refer the code memory (CM) in first stage and refer the DM (data memory) in fourth stage of the pipeline, so that accessing of these two memories in the same cycle does not create the conflict as described below:

CC = Clock cycles
I_i = Instruction number i

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1	CM	ID	ALU	DM	WB		
I_2		CM	ID	ALU	DM	WB	
I_3			CM	ID	ALU	DM	WB
I_4				CM	ID	ALU	DM
I_5					CM	ID	ALU
I_6						CM	ID
I_7							CM

Fig. 4.7 Resolution of Structural Hazards through Renaming

{CM = Code memory
DM = Data memory
ID = Instruction decode
WB = Write back}

7 cycles = 7 inputs = 0 stalls

{CPI = 1
Program output = safe}

2) Data dependency:

- Consider the program segment where instruction J follows instruction I in a program order.

I	:	Instruction
J	:	Instruction
⋮		⋮
⋮		⋮
⋮		⋮

- Data dependency condition will be occurred in the pipeline when the instruction J tries to read the data before instruction I write it.

Read – Before – write

Example:

Note:

A non-pipelined manner of execution does not suffer from any data dependency. Instructions are executed successively, i.e If there exists a register A which is modified by instruction Y and read by instruction X, then instruction X cannot start before instruction Y completes execution.

- If the above code is running on a pipelined processor, then data dependency condition will occur because I_2 is executed along with I_1. So I_2 tries to read the register r_0 data before I_1 writes it. Therefore, I_2 incorrectly accesses the old value from the register r_0 (data loss) as described below:

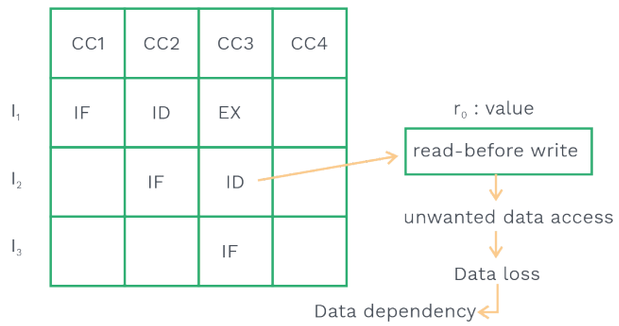


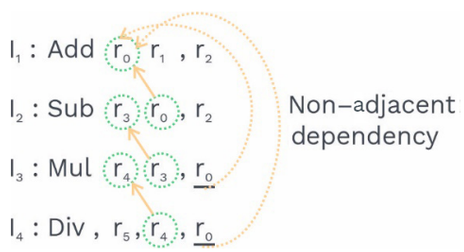
Fig. 4.8 Data Hazards

- To minimize the data hazards, hardware mechanism is used, i.e. operand forwarding, also called as bypassing or short-circuiting.
- This technique states that, use the buffer between the stages to hold the intermediate operation result, so that dependent instruction will be accessing the new value from the buffer before updating the register file.
- In this technique, last stage operation is modified to perform:

WRITE – READ operation

Means the updated data is immediately available to read, therefore buffer is not required at the end of last stage.

- Consider the following program segment, executed on a RISC pipeline using operand forwarding technique.



- Adjacent data dependency is known as 'true data dependency'.

  That is, $I_2 \rightarrow I_1 (r_0)$

  $I_3 \rightarrow I_2 (r_3)$

  $I_4 \rightarrow I_3 (r_4)$

- Non-adjacent data dependency is known as, data dependency only. Because it is eliminated via true data dependency.

  That is, $I_3 \rightarrow I_1 (r_0)$

  $I_4 \rightarrow I_1 (r_0)$

3) Control dependency:

- Control dependency will occur in the pipeline when transfer of control (TOC) instructions are executed in the pipeline.

Code:

1000 : I_1 ↓ Falling/ falling through path
1001 : I_2 ↓
1002 : I_3 (JMP 2000)
1003 : I_4
1004 : I_5
⋮ ⋮
2000 : BI_1 ↓ Taken path
2001 : BI_2 ↓
⋮ ⋮

expected output sequence : $I_1 - I_2 - I_3 - BI_1 - BI_2$

Instruction fetch micro-program (μ) program:

T_1 : PC → MAR
T_2: M[MAR] → MBR
PC + 1 → PC
T_3: MBR → IR

PC = 1000

	CC1	CC2	CC3	CC4	CC5	CC6
I_1	IF PC : 1001	ID	EX	MA	WB	
I_2		IF PC : 1002	ID	EX	MA	WB
I_3			IF PC : 1003	ID uncond : TOC PC : ~~1004~~ 2000	EX	MA
I_4				IF PC : ~~1004~~	ID	EX
BI_1					IF PC : 2001	ID
BI_2						IF PC : 2002

uncond : TOC ⟶ Unconditional transfer of control

Actual Output seq. is –

$I_1 - I_2 - I_3 - I_4 - BI_1 - BI_2$

⋮
Unwanted instruction

Fig. 4.9 Control Hazards

- In the above execution sequence, unwanted instruction is executed in the pipeline, so it brings the unwanted behaviour in the program output.

This kind of disturbance in the pipeline is called as control dependency.

- To handle the above problem, "Flush" operation is used, also called as freeze operation.

This operation states that, insert the NOP instruction after the JMP instruction to suspend the unwanted instruction fetch.

Code with flush operation:

1000 : I_1
1001 : I_2
1002 : I_3 (JMP 2000)
1003 : I_4 = (NOP)
1004 : I_5
. .
. .
. .
2000 : BI_1
2001 : BI_2
. .
. .

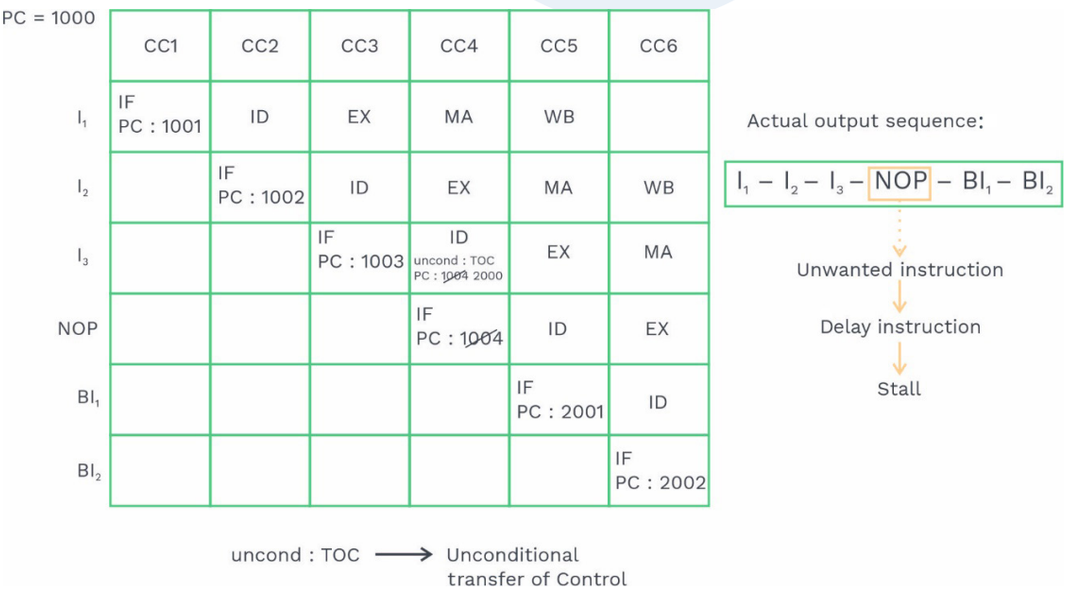


Fig. 4.10 Delayed Branch Technique

- Number of stalls created in the pipeline due to a branch instruction is called as branch penalty.

It depends on the availability of the TA (target address) in the pipeline, i.e.

$$\text{Branch penalty} = \text{At what stage, Target address is available in the pipeline} - 1$$

- In the RISC pipeline, branch penalty is always 1 because TA is available in the second stage.

Note:

In the hypothetical pipelines, TA availability is given as:

a) Stage number given, then

$$\text{Penalty} = \text{Stage number} - 1$$

b) Stage name given, then

$$\text{Penalty} = \text{Corresponding stage number} - 1$$

c) Until the instruction is completed (or) all instructions are proceed through all the stages, then

$$\text{Penalty} = \text{Last stage number} - 1$$

Total number of stalls created from the branches/program:

$$= \text{Branch frequency} * \text{Branch penalty}$$

Number of Branch instructions per program

Number of stalls/ branch

BRANCH PREDICTION BUFFER

- To minimize the control hazards, hardware mechanism is used, i.e. 'branch prediction buffer', also called as 'branch target buffer' or 'loop buffer'.
- It is a high-speed buffer, present in the first stage of the pipeline used to hold the predicted target addresses.
  If the TA is present in the first stage, then no stall is present. (Prediction is possible when the program contains loops.)

Note:
If the question states that the pipeline is with branch prediction, then we can assume that TA is present in the first stage. Otherwise (without branch prediction) assume that TA is not present in the first stage.

Note:
To minimize the control hazards, software mechanism is also used, i.e. 'delayed branch'.

Delayed branch:
It is a compiler technique, so compiler re–arranges the code to avoid the stall, if possible. Otherwise substitute the NOP instruction after the JMP instruction to preserve the execution path, if not possible to rearrange.

User code:

1000 : I_1
1001 : I_2
1002 : I_3 (JMP 2000)
1003 : I_4
⋮ ⋮
2000 : BI_1
2001 : BI_2
⋮ ⋮

Expected output sequence : $I_1 - I_2 - I_3 - BI_1 - BI_2$

Actual output sequence : $I_1 - I_2 - I_3 - \textcircled{I_4} - BI_1 - BI_2$

Unwanted

Delayed branch:

a) Re–arrangement

I_1
I_3 (JMP BI_1)
I_2
I_4
⋮
BI_1
BI_2
⋮

PC = I_1	CC1	CC2	CC3	CC4	CC5
I_1	IF PC : I_3	ID	EX	MA	WB
I_3		IF PC : I_2	ID unconditional : TOC PC : ↙ BI_1	EX	MA
I_2			IF PC : ~~↙~~ BI_1	ID	EX
BI_1				IF PC : BI_2	ID
BI_2					IF PC : BI_3

uncond : TOC ⟶ Unconditional transfer of control

Actual output sequence is:

$I_1 - I_3 - I_2 - BI_1 - BI_2$

Execution sequence is different but program output is same (without stall)

Fig. 4.11 Re-arrangement

b) NOP substitution:

I_1
I_2
I_3 (JMP BI_1)
NOP
I_4
⋮
BI_1
BI_2

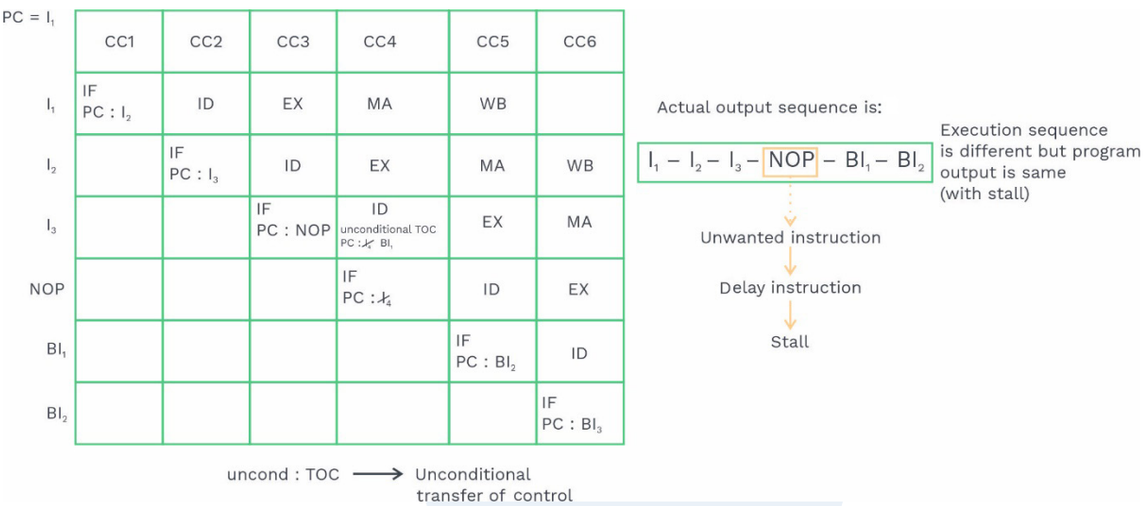
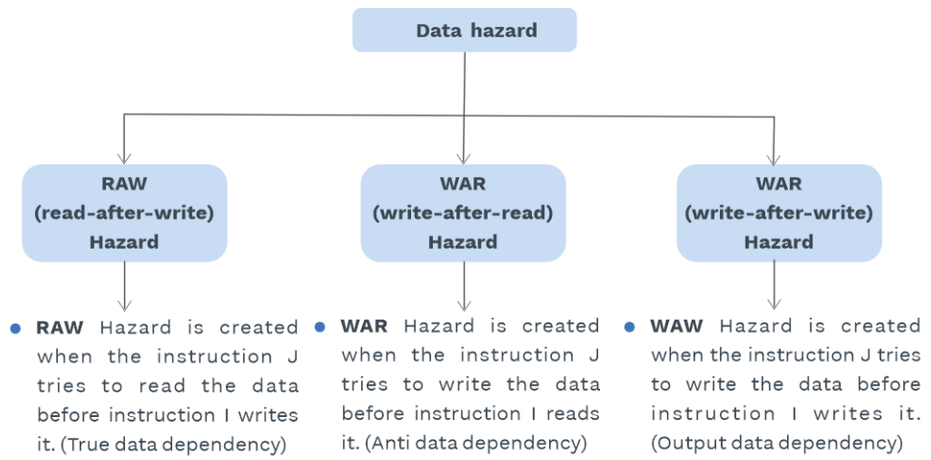


Fig. 4.12 NOP Substitution

Hazards:

- Hazard is a delay.
- Delay present in the pipeline is due to a dependency conditions.
- Three kinds of a hazards are possible in the pipeline:
  1) Structural hazard
  2) Data hazard
  3) Control hazard
- Data hazard is further classified into three types, based on the order of the read and write operations.



PRACTICE QUESTIONS

Q8 **Consider these two instructions:**
ADD R1, R2, R3
SUB R2, R4, R5
Which of the following is true?
a) It is an anti-dependency.
b) It is an output dependency
c) It is true dependency
d) Instructions are free from all dependencies

Sol: **a)**

ADD R1, R2, R3 is equivalent to R1 ← R2 + R3
SUB R2, R4, R5 is equivalent to R2 ← R4 – R5
Clearly, it is write-after-read (WAR) dependency which is also known as anti–dependency.

Q9 **Consider the two instructions given below:**
DIV R1, R2, R3
ADD R1, R4, R3
Which of the following dependencies can be noticed in these instructions?
a) Data dependency
b) Anti dependency
c) Output dependency
d) Instructions are free from any kind of dependency

Sol: **c)**

DIV R1, R2, R3 is equivalent to $R1 \leftarrow \frac{R2}{R3}$

ADD R1, R4, R3 can be converted to R1 ← R4 + R3
In both the instructions, result is written in R1, hence it is write-after-write (WAW) hazard which is also known as output dependency.

Q10 **Which of the following technique cannot be used to handle the control hazards?**

a) Delayed branch **b) Operand forwarding**
c) Branch prediction **d) Multiple pipelines**

Sol: b)

a) Delayed branch is the software solution provided by the compiler for control dependencies.
b) Operand forwarding is the solution for data dependencies.
c) Branch prediction is the hardware solution for control dependencies.
d) Multiple pipelines are the solutions for control hazard. In one pipeline, the execution takes place if the next instruction is sequential instruction after branch, i.e. branch is not taken. In another pipeline, execution of target in struction (not the sequential instruction after branch instruction) takes place.

Q11 **Consider the following code having a sequence of instructions:**

Instruction	Meaning
I_1: MUL R1, R2, R3	R1 ← R2 × R3
I_2: ADD R5, R4, R1	R5 ← R4 + R1
I_3: SUB R1, R5, R2	R1 ← R5 – R2
I_4: DIV R3, R4, R5	$R3 \leftarrow \frac{R4}{R5}$

Calculate the total number of dependencies including RAW, WAR and WAW. Assume instructions have the same execution sequence as given above.
a) 4 **b) 5** **c) 6** **d) 7**

Sol: c)

Consider the following RAW dependencies:

I_1 : MUL R1, R2, R3
R1 ← R2 × R3
I_2 : ADD R5, R4, R1
R5 ← R4 + R1

I_3 : SUB R1, R5, R2

$R1 \leftarrow R5 - R2$

I_4 : DIV R3, R4, R5

$R3 \leftarrow \frac{R4}{R5}$

$I_1 \rightarrow I_2$, $I_2 \rightarrow I_3$, $I_2 \rightarrow I_4$ } RAW (read-after-write) dependencies

$I_1 \rightarrow I_4$, $I_2 \rightarrow I_3$ } WAR (write-after-write) dependencies

$I_1 \rightarrow I_3$ } WAW (write-after-write) dependency

RAW = 3 dependencies
WAR = 2 dependencies
WAW = 1 dependency
Total = 3 + 2 + 1 = 6 dependencies

Q12 **Consider the following two instructions:**

MUL R3, R1, R2

DIV R5, R4, R3

Which of the following dependency can be observed in these instructions?

a) True dependency

b) Anti dependency

c) Output dependency

d) Instructions are free from any kind of dependencies

Sol: **a)**

1) MUL R3, R1, R2 is equivalent to $R3 \leftarrow R1 \times R2$

2) DIV R5, R4, R3 is equivalent to $R5 \leftarrow \frac{R4}{R3}$

Here result is written on R3 in multiplication instruction and then contents in R3 is read in division instruction. Clearly it is a case of RAW hazard which is also known as true dependency.

Instruction scheduling (when 'operand forwarding is not supported')

- CPU always executes the program in a sequence called as in-order execution.
- In this execution sequence, if instruction is dependent, then the remaining instructions are also sharing the stall cycles, even if they are independent.

Code:

I_1 : ADD r_0 r_1 , r_2

I_2 : SUB r_3 , r_0 , r_4

I_3 : MUL r_4 , r_5 , r_6

I_4 : DIV r_3 , r_7 , r_8

In-order execution sequence is:

I_1 – I_2 – I_3 – I_4

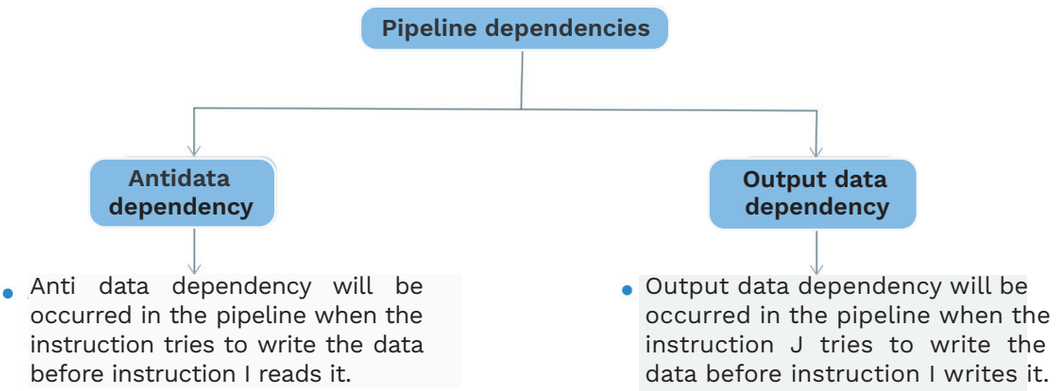
Stalls

- In the above code, I_2 is data dependent on I_1, so I_2 will be waiting until the I_1 execution is completed. This waiting creates stalls in the pipeline. These stalls are also shared by the I_3 and I_4 instructions, even they are independent.
- To handle this problem, 'operand forwarding' technique is used.
- If the CPU doesn't support this technique, then instruction scheduling concept is used to optimize the stalls.
- Instruction scheduling executes the independent instructions first, called as out of execution order (re–order).

That is,

I_1 – I_3 – I_4 – I_2

- Out of order execution creates two more dependencies in the pipeline:



Note:

To handle the above dependency conditions, hardware mechanism is used. That is 'register renaming'.

Performance evaluation with stalls:

$$S(\text{speed up}) = \frac{\text{Average instruction ET}_{\text{non pipeline}}}{\text{Average instruction ET}_{\text{pipeline}}}$$

$$S(\text{speed up}) = \frac{CPI_{nonpipe} * \text{Cycle time}_{nonpipe}}{CPI_{pipe} * \text{Cycle time}_{pipe}}$$

- Ideal CPI of pipeline is always 1.
- But due to the dependency problem, extra cycles are created in the pipeline. So,

$$S(\text{speed up}) = \frac{CPI_{nonpipe} * \text{Cycle time}_{nonpipe}}{(1 + \text{number of stalls per instruction}) * \text{Cycle time}_{pipe}}$$

PRACTICE QUESTIONS

Q13 **Consider a 5-stage pipeline having stages as instruction: Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Execute (Ex) and Write Back (WB). Here we are given 4 instructions. IF, ID, OF and WB stages takes 1 clock cycle each but the EX stage takes 1 cycle for ADD and SUB, 2 cycles for MUL and 3 cycles for DIV operation. If the operand forwarding technique is used from EX stage to OF stage then calculate the total execution time of below program. Assume the clock rate of pipeline processor as 5 GHz.**

Instruction number	Instruction	Meaning
I_1	DIV A, B, C	$A \leftarrow \frac{B}{C}$
I_2	MUL F, E, D	$F \leftarrow E \times D$
I_3	SUB Y, A, F	$Y \leftarrow A - F$
I_4	ADD H, Y, G	$H \leftarrow Y + G$

Sol: **2.2**

	1	2	3	4	5	6	7	8	9	10	11
I_1	IF	ID	OF	EX	EX	EX	WB				
I_2		IF	ID	OF	–	–	EX	EX	WB		
I_3			IF	ID	–	–	OF	–	EX	WB	
I_4				IF	ID	–	–	OF	–	EX	WB

Here operand forwarding is used from EX to OF stage.

I_1 DIV A, B, C
I_2 MUL F, E, D
I_3 SUB Y, A, F
I_4 ADD H, Y, G

Instruction I_3 is dependent on both I_1 and I_2, so the operands for I_3 are fetched after the execution of instruction I_1 and I_2.

When execution of I_2 gets completed, we can execute the instruction I_3.
Similarly, I_4 is dependent on I_3. We can only execute I_4 after the execution of I_3.
Total execution time = Total number of clock cycles × clock cycle time

$$\text{Clock cycle time} = \frac{1}{\text{clock rate}}$$
$$= \frac{1}{5 \times 10^9}\text{ s}$$
$$= 0.2 \times 10^{-9}\text{ s}$$
$$= 0.2\text{ ns}$$
$$\text{Total execution time} = 11 \times 0.2\text{ ns}$$
$$= 2.2\text{ ns}$$

Q14 **Given below is a set of instructions:**

I_0: A ← B + C
I_1: B ← A – C
I_2: C ← B × A
I_3: A ← C/B
I_4: B ← A

Where A, B and C are registers.
How many true data dependency, anti-data dependency and output data dependency exist?

a) 3, 8, 3 **b) 3, 6, 2** **c) 4, 6, 2** **d) 4, 8, 2**

Sol: **d)**

True data dependency → RAW hazards

$I_1 - I_0$ over A
$I_2 - I_1$ over B
$I_3 - I_2$ over C
$I_4 - I_3$ over A
} 4

Anti data dependency → WAR hazards

$I_1 - I_0$ over B
$I_4 - I_0$ over B
$I_2 - I_0$ over C
$I_2 - I_1$ over C
$I_3 - I_1$ over A
$I_4 - I_2$ over B
$I_4 - I_3$ over B
$I_3 - I_2$ over A
} 8

Output data dependency → WAW

$I_0 - I_3$ over A
$I_1 - I_4$ over B
} 2

Q15 **Consider a pipeline architecture having 5 stages. Except the third stage, all instructions spend one cycle in the other stages. The third stage takes 3 clock cycles for instruction LOAD. What is the count of clock cycles required for the execution of the following machine code using optimization?**

I_0: LOAD R_0, 3(R_1); $R_0 \leftarrow [3 + [R_1]]$
I_1: ADD R_2, R_0, R_1; $R_2 \leftarrow R_0 + R_1$
I_2: LOAD R_3, 4(R_4); $R_3 \leftarrow [4 + [R_4]$
I_3: SUB R_5, R_3, R_4 ; $R_5 \leftarrow R_3 - R_4$

Sol: **14**

operand forwarding through interstage buffer register

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I_0	IF	ID	EX	MA	MA	MA	WB							
I_1		IF	ID	ID	ID	ID	EX	MA	WB					
I_2			IF	IF	IF	IF	ID	EX	MA	MA	MA	WB		
I_3							IF	ID	ID	ID	ID	EX	MA	WB

operand forwarding through interstage buffer register

I_1 depends on I_0 for value in R_0. Since I_0 is LOAD instruction that requires memory access, R_0 data will be available after MA stage in the buffer. (Assuming no stage delays.)

[** Similar explanation for I_3 and I_4 **]

∴ Number of clock cycles required = 14

Q16 **A 6-stage pipelined processor is uniformly balanced with a stage propagation delay of 10 ns. A program segment comprising of 40% unconditional branch instructions is executed on this architecture. The target address for branch operation is not available until the last stage. Estimate the average execution time of the processor.**

a) 30 ns **b) 20 ns** **c) 15 ns** **d) 35 ns**

Sol: a)

Given that the instruction after branch is not fetched till the branch instruction is completed. So CPI for branch instruction = 6, because we come to known about the target address after the last stage (6th stage) of pipeline.

For normal (non–branch) instructions, CPI = 1.

Given that we have only unconditional branch, so definitely we will have a branch during these unconditional branch instructions.

Let the total instructions = n

Branch instructions = 40% of n

$$\text{Average CPI} = \frac{\begin{pmatrix}\text{Non-Branch Instructions}\\ \text{Percentage} \times \text{CPI of}\\ \text{Non-Branch Instructions}\end{pmatrix} + \begin{pmatrix}\text{Branch Instructions}\\ \text{Percentage} \times \text{CPI of}\\ \text{Branch Instructions}\end{pmatrix}}{\text{Total Instructions}}$$

$$= \frac{(60\% \text{ of } n) \times 1 + (40\% \text{ of } n) \times 6}{n}$$

$$= \frac{0.6n \times 1 + 0.4n \times 6}{n}$$

$$= \frac{0.6n + 2.4n}{n} = 3$$

Average-instruction execution time = Average CPI × $T_{pipeline}$

= 3 × 10 ns (Clock cycle time $T_{pipeline}$ = 10 ns)

= 30 ns

Q17 **Consider a non-pipelined processor running at 4 GHz. it takes 10 cycles to finish an instruction. Designers have converted it into 10-stage pipeline processor but the hardware overhead makes the frequency of new design as 2 GHz. Due to instruction cache miss, 20% of the instructions cause a stall of 15 cycles during the instruction fetch (IF) stage. There are no other hazards in the system. Calculate the speed up of pipelined processor over non–pipelined processor.**

a) 1.75 **b) 1.25** **c) 2.25** **d) 2.75**

Sol: **b)**

Execution time of non–pipelined processor = Number of clock cycles × clock cycle time.

$$\text{Clock cycle time} = \frac{1}{\text{Frequency}}$$

$$\text{Execution time for non–pipelined processor} = 10 \times \frac{1}{4 \times 10^9}\ \text{s}$$

$$= 2.5 \times 10^{-9}\ \text{s}$$

$$= 2.5\ \text{ns}$$

- Effective CPI for pipelined processor = Ideal CPI + (Percentage of Instruction cache miss) × (Stalls due to instructions cache miss)
  = 1 + (20%) × 15
  = 1 + 0.2 × 15
  = 1 + 3 = 4

Execution time of pipelined processor = Effective CPI × Clock cycle time

$$\text{clock cycle time} = \frac{1}{\text{Frequency}}$$

$$= \frac{1}{2 \times 10^9}\ \text{s}$$

Execution time of pipeline processor

$$= 4 \times \frac{1}{2 \times 10^9}\ \text{s}$$

$$= 2 \times 10^{-9}\ \text{s}$$

$$= 2\ \text{ns}$$

$$\text{speedup} = \frac{\text{Non – pipelined execution time}}{\text{Pipelined execution time}}$$

$$= \frac{2.5\,\text{ns}}{2\,\text{ns}} = \boxed{1.25}$$

Q18 **Consider a pipeline of 5 stages. In this pipeline we have 50% of the unconditional branch instructions whose target address is known after third stage only, rest of the instructions are normal instructions. There is no penalty for normal instructions. The clock cycle time for this pipelined processor is 100 ns. Calculate the throughput in MIPS (million instructions per second).**
a) 4 MIPS **b) 8 MIPS** **c) 5 MIPS** **d) 7 MIPS**

Sol: **c)**

Let total instructions = x
Unconditional branched instructions = 50% of x
= 0.5 x
CPI for unconditional branched instructions = 3 (target address is known after third stage only).
CPI for normal instructions = 1

$$\text{Average CPI} = \frac{0.5x \times 3 + 0.5x \times 1}{x} = \frac{1.5x + 0.5x}{x} = \frac{2x}{x} = 2$$

Note:

$$\text{Average CPI} = \frac{\left(\begin{array}{l}\text{Percentage of}\\ \text{unconditional}\\ \text{branch instructions}\\ \times\text{CPI of unconditional}\\ \text{branch instructions}\end{array}\right) + \left(\begin{array}{l}\text{Percentage of}\\ \text{normal instructions}\\ \times\text{CPI of normal}\\ \text{instructions}\end{array}\right)}{\text{Total instructions}}$$

- Execution time for 1 instruction = Average CPI × Clock cycle time
  = 2 × 100 ns = 200 ns
  1 instruction takes = 200 ns
  1 instruction = 200×10^{-9} s

$$1 \text{ s} = \frac{1}{200 \times 10^{-9}} \text{ instructions}$$

$$1 \text{ s} = \frac{10^9}{200} \text{ instructions}$$

$$1 \text{ s} = \frac{10^3}{200} \times 10^6 \text{ instructions}$$

$= 5 \times 10^6$ instructions
= 5 MIPS

Q19 **Consider a 5-stage pipeline without branch prediction buffer. It is operated with a 2ns clock and allows all instructions except JMP instruction. The target address is not available until fourth stage. A program segment is to be executed on this pipeline. The program segment contains 40% JMP instructions. Among them 30% are conditional branch instructions, since 40% of the conditional JMP instructions fails to satisfy the condition, no stalls are present for them. What is the performance gain achieved by the system?**

a) 2.43 **b) 2.56** **c) 3.43** **d) 3.08**

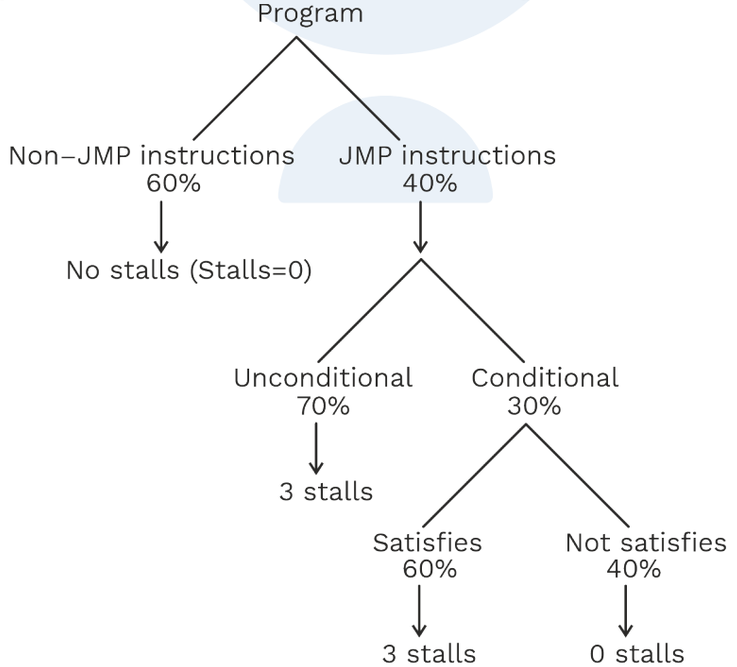
Sol: a)

Pipeline is without branch prediction buffer

⇓

Stalls are present

Since, target address is available at the fourth stage, number of stalls = 4 – 1 = 3



Number of stalls/instruction = (0.6 × 0) + (0.4 × 0.7 × 3)
+(0.4 × 0.3 × 0.6 × 3)
+(0.4 × 0.3 × 0.4 × 0)
= 1.056

$$\text{Performance gain} = \frac{K}{1 + \text{Number of stalls / instructions}} = \frac{5}{1 + 1.056}$$

$$= 2.43$$

Q20 **Consider a 8–stage pipeline operated with 3ns clock pulse is used to execute a program segment. There are 20 instructions in that program numbered from I_1 to I_{20}. I_5 is an unconditional JMP instruction that transfers the control to I_{17}. Assuming that the target address is available at the last stage of the pipeline. What is the execution time of the program (in nanoseconds)?**

a) 72 **b) 69** **c) 75** **d) 66**

Sol: **b)**

Since target address is available at the last stage of the pipeline, number of stalls = 8 – 1 = 7.

Successful execution:

$I_1 – I_2 – I_3 – I_4 – I_5 – NOP – NOP – NOP – NOP – NOP – NOP – NOP – I_{17} – I_{18} – I_{19} – I_{20}$

n = 16

$\therefore$ Execution time$_{pipe}$ = (K + n – 1) × tp

= (8 + 16 – 1) × 3ns

= 69 ns

Chapter Summary

- **Pipeline definition:**

The definition states that, insert the new input into the pipeline before computation of an old input, so new input is executing along with the old input called as overlapping execution.

- **Design of pipeline:**
  - **a)** Uniform delay pipeline:

$$\boxed{\text{Cycle time}\left(t_p\right) = \text{Stage delay}}$$

  - **b)** Non-uniform delay pipeline:

$$\boxed{\text{Cycle time}\left(t_p\right) = \text{Maximum}\left(\text{Stage delay}\right)}$$

  - **c)** If buffer delay present in the pipeline:

$$\boxed{\text{Cycle time}\left(t_p\right) = \text{Maximum}\left(\text{Stage delay} + \text{Buffer delay}\right)}$$

- **Performance analysis of pipeline:**

$$\boxed{S = \frac{n.t_n}{(K+n-1)\,t_p}} \left(\text{When n is finite}\right)$$

$\boxed{S = \frac{t_n}{t_p}}$, when n approaches to infinity (∞)

- **Types of pipelines:**
  1) **Linear pipeline:** This type of pipeline contains forward connections only.
  2) **Non-linear pipeline:** This pipeline contains forward and backward connection.
- **Dependencies in the pipeline:**
  1) **Structural dependency:** It occurs in the pipeline due to a resource conflict
  2) **Data dependency:** Data dependency condition will occur in the pipeline when the instruction J tries to read the data before instruction I write it.
  3) **Control dependency:** Control dependency will occur in the pipeline when transfer of control (TOC) instructions are executed in the pipeline.
- **Delayed branch:** It is a compiler technique, so compiler rearranges the code to avoid the stall.
- **Hazards:** Hazard is a delay which is present in the pipeline due to a dependency condition.

Chapter Summary

Three kinds of a hazards are possible in the pipeline:

1) Structural hazard
2) Data hazard
3) Control hazard

- Data hazard is further classified into three types:
  - i) RAW (read-after-write) hazard
  - ii) WAR (write-after-write) hazard
  - iii) WAW (write-after-write) hazard
- **Operand forwarding:** It is a hardware mechanism used to minimize the data hazards.
- **Instruction scheduling:** It is used when operand forwarding is not supported.
- **Performance evaluation with stalls:**

$$S(\text{speed up}) = \frac{CPI_{nonpipe} * \text{cycle time}_{nonpipe}}{(1 + \text{number of stalls per instruction}) * \text{cycle time}_{pipe}}$$

5 Memory Hierarchy Design and Cache Memory

5.1 MEMORY ORGANISATION

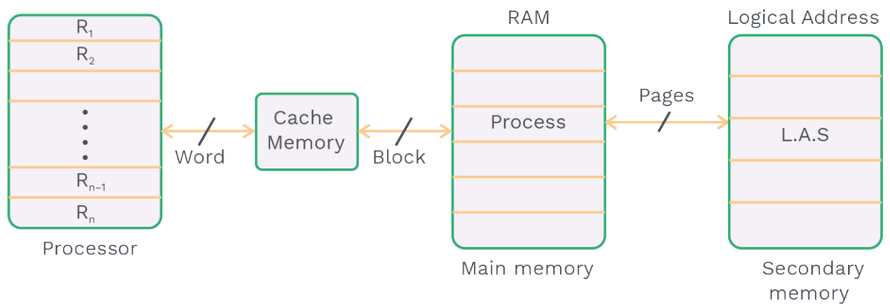


Fig. 5.1 Memory Organisation

Memory hierarchy:

Why is memory essential?

- Memory is an essential component of a computer since it's used for storing programs and necessary data.

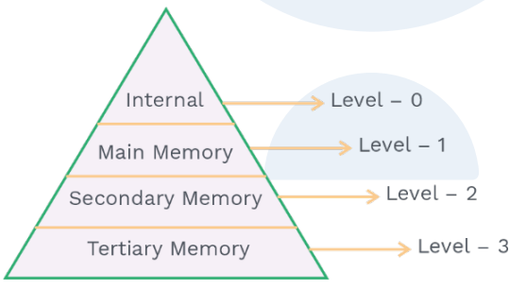


Fig. 5.2 Memory Heirarchy

- Internal memory consists of registers and cache.
- Main memory consists of RAM and ROM.
- Secondary memory consists of a Hard disk.
- Tertiary memory consists of tape.

Representation of memory:

1) In terms of storage:

Storage capacity of memory will increase from level 0 to level 3.

2) In terms of access time:

Access time of the memory is increased from level 0 to level 3. The access time of internal memory is very less negligible as compared to tertiary memory.

3) In terms of speed:
Access speed of memory decreases from level 0 to level 3. Access speed of internal memory is very fast as compared to others. Level 0 memory is very close to the CPU.

4) In terms of frequency access and per bit storage cost:
Frequency decreases from level 0 to level 3. The frequency of access level–0 is more as compared to other levels.
Per bit storage cost of level 0 to level 3 is decreased. Level–0 cost is high and level–3 cost is low.

$$\text{Average memory access time} = \frac{(\text{Fetch time} + \text{Write time}) \text{ of all instructions}}{\text{Total instructions}}$$

$$\text{Performance}(P) \propto \frac{1}{\text{Access Time}(A_T)}$$

Total time = Hit/miss latency + Access time (A_T) + Transfer Time
Hit/miss latency is negligible, and transfer time depends upon B.W. (Bandwidth)

Types of memory organisation:
Memory organisation is divided into two parts based on accessing order.
a) Hierarchical access memory organisation
b) Simultaneous access memory organisation

To understand the concept of hierarchical/simultaneous, we need to understand the concept of Hit and Miss.

Hit:
The successful access of data from cache/main memory is called hit.

Hit ratio (H):
It is the ratio of the number of successful memory access to the total number of attempts to access memory.

$$\text{Hit ratio} = \frac{\text{Number of hits}}{\text{Total number of access}} = \frac{\text{Number of hits}}{\text{Number of hits} + \text{number of misses}}$$

Miss:
The unsuccessful access of data from cache/main memory is called miss.

Miss ratio (M):

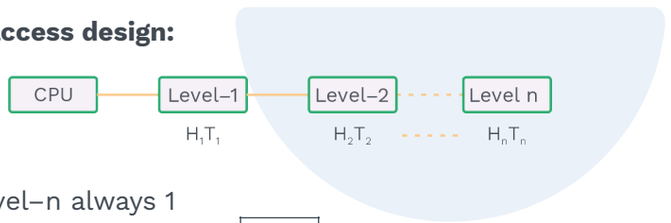
$$\boxed{1 - \text{Hit ratio}(H)}$$

a) Hierarchical access:

In a hierarchical organisation, the CPU always accesses the data from level 1 cache only.

If data is not present in level–1 cache memory, then the data is copied from level–2 to level–1. If data is not present in the level–2 cache, then data is copied from level–3 to level–2 and level–2 to level–1, then the CPU will access the data from level–1 cache, or we can say data is copied from higher (higher number of levels) level to level–1 memory.

Hierarchical access design:



Hit ratio at level–n always 1

$$\boxed{H_n = 1}$$

$$\text{Average access time } (T_{avg}) = H_1T_1 + (1-H_1)H_2\left(\overbrace{T_2}^{\text{Search Time}} + T_1\right) + (1-H_1)(1-H_2)H_3$$

$$\left(\underbrace{T_1 + T_2}_{\text{Search time}} + T_3\right) + \ldots\ldots\ldots\ldots\ldots\ldots\ldots + \ldots\ldots\ldots\ldots + (1-H_1)(1-H_2)\ldots\ldots(1-H_{n-1})H_n(T_1 + T_2 + \ldots T_n)$$

H_1 = Hit ratio at level–1

H_2 = Hit ratio at level–2

.

.

H_n = Hit ratio at level–n

T_1 = Access time at level–1

T_2 = Access time at level–2

.

.

T_n = Access time at level–n

T_{avg} = Total time to access 1 word data from memory.

PRACTICE QUESTIONS

Q1 **Consider a system with two level cache access time of level 1, level 2, and main memory are 10 ns, 70 ns and 600 ns, respectively. If there is miss in level–1 cache then CPU copied the data from level–2 to level–1 and access it. Hit ratio of level 1 and level 2 is 0.6 and 0.8, respectively. What is the average memory access time?**

Sol: **86**

Hit rate level 1 = 0.6

Hit rate level 2 = 0.8

Access time L_1 = 10 ns

Access time L_2 = 70 ns

Access time of main memory T_m = 600 ns

$T_{avg} = H_1T_1 + (1–H_1)H_2\ (T_2+T_1) + (1–H_1)\ (1–H_2)H_3 \quad (T_m+T_2+T_1)$

$T_{avg} = 0.6 * 10 + 0.4 * 0.8 * 80 + 0.4 * 0.2 * 680$

$T_{avg} = 6 + 25.6 + 54.4$

$T_{avg} = 86$ ns

Q2 **Consider a three-level memory architecture specified below:**

Level	Access time / word	Block Size	Hit Ratio
L_1	30 ns	–	0.75
L_2	50 ns	2	0.7
L_3	100 ns	4	–

CPU accesses a block from L_1 memory. If it is a miss the block is copied from L_2 to L_1. If the block is not found in L_2, the sound block is copied from L_3 to L_2 and then L_2 to L_1. What is the effective memory access time?

a) 43 ns **b) 47 ns** **c) 45 ns** **d) 41 ns**

Sol: **d)**

Clearly, the above architecture is hierarchical one.

Effective Access Time(EAT) = $H_1T_1 + (1 – H_1)H_2(T_1 + T_2) + (1 – H_1)(1 – H_2)H_3(T_1 + T_2 + T_3)$

Now, T_1 = 30 ns (Default block size = 1 word)

T_2 = 100 ns (Block size = 2 words)

T_3 = 400 ns (Block size = 4 words)

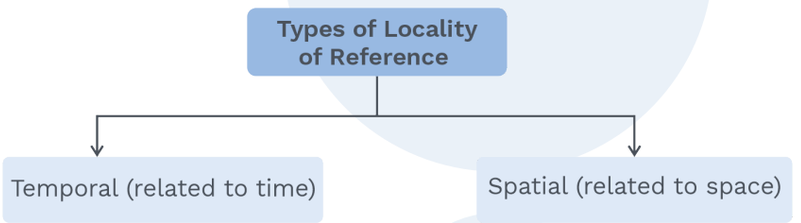
EAT = 0.95 × 30 + 0.05 × 0.7 × 130 + 0.05 × 0.3 × 1 × 530

[∵ There is always hit in L_1]

= 41 ns

Locality of reference:

When the CPU has requested one address for memory access, then that particular address or nearby address will be accessed soon, known as the locality of reference. Based on locality of reference, average memory access time decreases because programs tend to reuse data and instructions they have used recently.



Temporal locality:

If an item is referenced in memory, it will tend to be referenced again soon because of loops. LRU takes care of temporal locality.

Spatial locality:

If an item is referenced in memory, nearby items will tend to be referenced soon because, during data access, their adjacent data words are accessed in a sequence. Block size takes care of spatial locality.

Simultaneous access memory organisation:

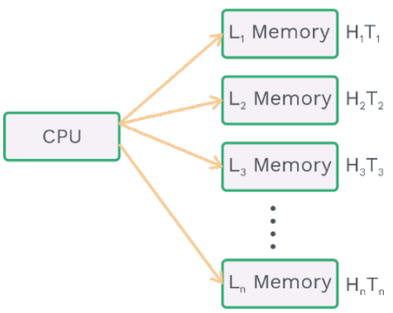


Fig. 5.3 Simultaneous Access Memory Organization

In this organisation, a CPU is directly connected to all levels of memory, CPU can directly access the data from L_i memory, L_i means the level where the data is present. Here we can't copy the data from level–n to level–1. If there is a miss in level 1 the then CPU can directly access the data from level–2 if not in level–2 then the CPU can directly access the data from level–3 and so on. Search time is ignored.

Hit ratio at level–n is 1.

$$\text{Hit ratio} = \frac{\text{Number of hits}}{\text{Total no. of access}}$$

Assume "T" represents access time and "H" is hit ratio then,

Relation:

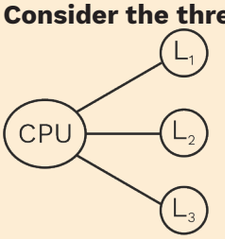
$$\text{Average access time} = T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)H_3T_3 + ... \\ ... + (1-H_1)(1-H_2)....(1-H_{n-1})H_nT_n$$

Throughput of memory in WPS (Words Per Second)

$$\eta_{memory} = \frac{1}{T_{avg}} \text{WPS}$$

PRACTICE QUESTIONS

Q3 **Consider the three-level memory configuration as:**



Hit ratio = 0.8, T_{avg}/block = 30 ns

Hit ratio = 0.7, T_{avg}/block = 40 ns

Hit ratio = 1, T_{avg}/block = 50 ns

If there is a miss in L_1 memory, CPU accesses the block from the higher levels without copying the block from higher to lower levels of memory. What is the estimated effective memory access time for this configuration _____ns.

Sol: **32.6**

$T_{avg} = H_1T_1 + (1 - H_1)\ H_2\ T_2 + (1 - H_1)\ (1 - H_2)\ H_3\ T_3$

$= 0.8 \times 30 + 0.2 \times 0.7 \times 40 + 0.2 \times 0.3 \times 1 \times 50$

$= 32.6$ ns

Q4 **In a two-level simultaneous memory organization the speed of level-1 is 8 times more than level-2 memory. CPU always accesses the requested block from level-1 memory first. If the access time of level-1 memory is 30 ns less than the effective memory access time. What is the hit ratio of level-1 memory? (Assume, T_1 = 40 ns, where T_1 = Access time of level-1 memory).**

a) 0.64 **b) 0.89** **c) 0.95**

d) 0.75

Sol: **b)**

$\Rightarrow T_1 = 40$ ns

$\Rightarrow T_{avg} = (40 + 30)$ ns $= 70$ ns

Again, level-1 memory is 8 times faster than level-2 memory.

$\Rightarrow T_2 = 40 \times 8 = 320$ ns

Now, since it's a simultaneous memory model-

$\Rightarrow T_{avg} = HT_1 + (1 - H_1)\ H_2\ T_2$

$\Rightarrow 70 = H_1 \times 40 + (1 - H_1)\ 1 \times 320$

$\Rightarrow 70 + 40H_1 + 320 - 320H_1$

$\Rightarrow 280H_1 = 250$

$\Rightarrow H_1 = 25/28 = 0.89$

Cache memory:

- The speed of the main memory is very slow compared to modern processors. Cache memory is very fast, and the smallest component between CPU and main memory.
- Cache memory plays a very important role in reducing average memory access time and increase the overall throughput of the system.
- The potency of the working of the cache mechanism is established upon the principle of locality of reference. The locality of reference we have discussed previously in hierarchical access design.

Another way of defining cache memory: Based on the locality of reference concept, all the current demanded localities are kept in a smaller and faster memory called as cache.

Previous Years' Question

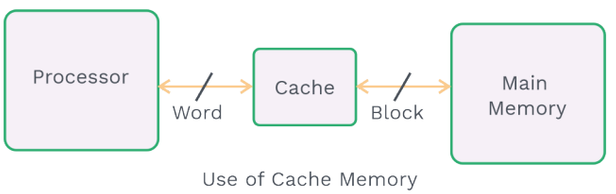
Consider a system with 2 level caches. Access times of level 1 cache, level 2 cache and main memory are 1 ns, 10 ns, and 500 ns, respectively. The hit rates of level 1 and level 2 caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache? (Use Hierarchy memory access time)

a) 13.0 ns **b)** 12.8 ns
c) 12.6 ns **d)** 12.4 ns

Sol: c) **(GATE-2004–1 Mark)**

Performance of cache memory:
Performance of cache memory depends on several segments:

- Cache size
- Cache block size
- Number of levels of cache
- Cache mapping technique
- Cache replacement policy
- Cache update policy

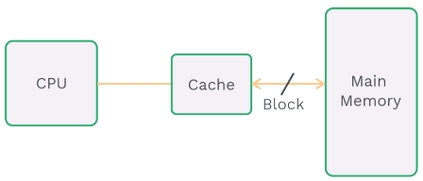


Use of Cache Memory

Fig. 5.4

Cache size:
Cache size is the amount of main memory data it can add.

Cache block size:
The number of data words/bytes in each cache block is defined as cache block size. It is always in power of 2.



(Main memory transferring the data in terms of Block)

Fig. 5.5

5.2 MULTI–LEVEL CACHE

- Multilevel cache is used to decrease/reduce the miss penalty and improve the performance in the system.
- With the help of a two–level or three–level cache (i.e., multilevel cache organization) we can reduce the miss penalty. If data is not present in the lower level cache, then the time required to transfer the data from the higher level to lower-level memory is called a miss penalty.

Definitions

Time taken to service a miss is known as miss penalty.

- Some designs are important to understand the topic.
- Suppose the CPU wants 15 memory references (MR), then calculates the access time.

System design without cache:

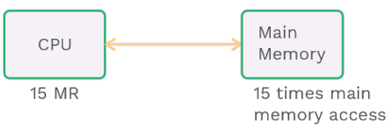


Fig. 5.6

System design With Cache:

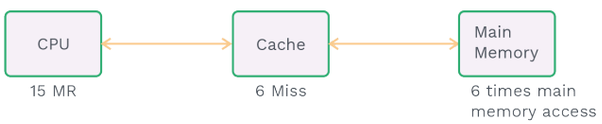


Fig. 5.7

System design with multilevel cache:

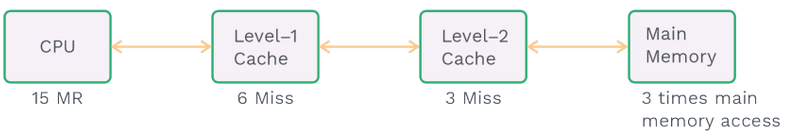


Fig. 5.8

- This system design with a multilevel cache accesses the main memory 3 times. So miss the penalty is less here comparatively.

Important points:

- Level–1 cache memory size is always a subset of level–2 cache memory size.
- Level–1 cache performance is always better than level–2 because level–1 the cache memory is very close to the CPU.
- Level–1 memory access time is less than the level–2 memory access time because level–1 cache is close to the CPU, so the CPU can access the data in the level–1 cache in no time.
- In the multilevel cache organisation, two types of miss rates we calculate:
  a) Global Miss Rate (GMR)
  b) Local Miss Rate (LMR)

Global miss rate (GMR):
Global miss rate is the ratio of total misses in cache memory (CM) and total number of CPU generated memory references at each level.

$$\text{Global Miss Rate (GMR)} = \frac{\text{Number of Miss Operation in CM}}{\text{Total Number of Memory References generated by CPU}}$$

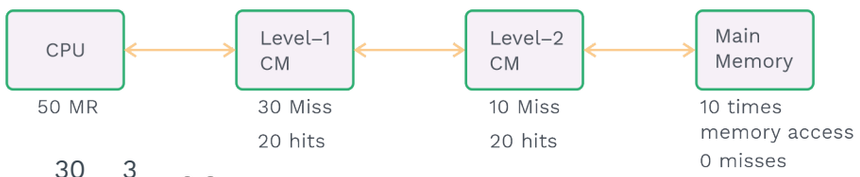
Local miss rate (LMR):
Local miss rate is the ratio of total miss operation in cache memory (CM) to total no. of access the cache at each level.

$$\text{Local Miss Rate (LMR)} = \frac{\text{Number of Miss Operation in CM}}{\text{Total Number of Access to the CM}}$$

PRACTICE QUESTIONS

Q5 **Consider two–level cache CPU generated 50 memory references. 30 miss operation present in level–1 cache and 10 miss operation, present in level–2 cache. Then calculate:**
a) Global miss rate at level–1 (GMR L_1)
b) Local miss rate at level–1 (LMR L_1)
c) Global miss rate at level–2 (GMR L_2)
d) Local miss rate at level–2 (LMR L_2)

Sol: $$\text{GMR } L_1 = \frac{\text{Number of Misses Operation at level} - 1}{\text{Total number of Memory References generated by CPU}}$$



$$GMRL_1 = \frac{30}{50} = \frac{3}{5} = 0.6$$

$$LMRL_1 = \frac{\text{Numbeer of Misses Operation at level} - 1 \text{ cache memory}}{\text{Total Number of access to the CM}}$$

$$LMRL_1 = \frac{30}{50} = \frac{3}{5} = 0.6$$

$$GMRL_2 = \frac{\text{Number of Misses Operation at level} - 2}{\text{Total number of Memory Generated by CPU}}$$

$$= \frac{10}{50} = \frac{1}{5} = 0.2$$

$$LMRL_2 = \frac{\text{Number of Misses Operation at level} - 2 \text{ cache memory}}{\text{Total number of Access to the CM}}$$

$$= \frac{10}{30} = \frac{1}{3}$$

$$= 0.33$$

Previous Years' Question

Consider a two–level cache hierarchy L1 and L2 caches. An application incurs 1.4 memory accesses per instruction on average. For this application, the miss rate of L1 cache 0.1, the L2 cache experience on average, 7 misses per 1000 instructions. The miss rate of L2 expressed correct to two decimal places is _____.

Sol: [0.05] **(GATE-2017 (Set–1) 2 Marks)**

Formula:

How to calculate the average memory access time in terms of the hit time, miss penalty and the local miss rate for 2 level cache memory.

$$\text{Average Memory Access Time}\left(T_{avg}\right) = \text{Hit time level} - 1 + \left(\begin{matrix}\text{Miss rate} \\ \text{level} - 1\end{matrix} * \begin{matrix}\text{Miss Penalty} \\ \text{level} - 1\end{matrix}\right)$$

$$\text{Miss Penalty level} - 1 = \text{Hit time level} - 2 + \left(\frac{\text{Miss rate}}{\text{level} - 2} * \frac{\text{Miss Penalty}}{\text{level} - 2}\right)$$

$$\text{Miss Penalty level} - 2 = \text{Main memory access time}$$

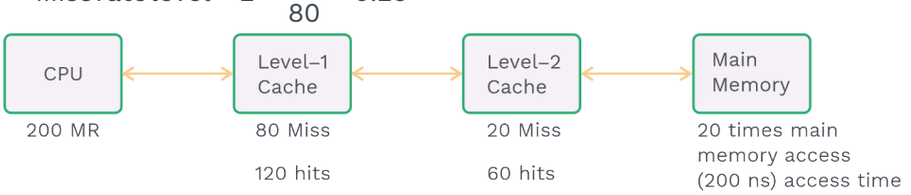
- Miss penalty level–2 is same as the main memory access time, because in the level–2, total number of miss operations is same as the number of time main memory access.

PRACTICE QUESTIONS

Q6 **Consider two level cache CPU generates 200 memory reference (MR). 80 miss operation present at level–1 cache and 20 miss operation present at level–2 cache. Hit time at level–1 and level–2 cache is 40 and 90 ns, respectively and main memory access time is 200 ns. What is the average memory access time? (Use the concept of local miss rate)**

Sol: **96**

- Level–1 miss operation = 80
- level–2 miss operation = 20
- CPU generated = 200 MR
- Hit time level–1 = 40 ns
- Hit time level–2 = 90 ns
- Main memory access time = 200 ns
- $\text{Using local miss rate concept, Miss rate level} - 1 = \frac{80}{200} = 0.4$
- $\text{Miss rate level} - 2 = \frac{20}{80} = 0.25$



$$T_{avg} = \text{Hit time level} - 1 + \left(\frac{\text{Miss rate}}{\text{level} - 1} * \frac{\text{Miss penalty}}{\text{level} - 1}\right)$$

$$= 40 + \left[0.4 * \left[\begin{matrix}\text{Hit time} \\ \text{level} - 2\end{matrix} + \begin{pmatrix}\text{Miss rate} * \text{Miss penalty} \\ \text{level} - 2 \quad \text{level} - 2\end{pmatrix}\right]\right]$$

As we know,

$$(\text{Miss penalty})_{L2} = (\text{Hit time})_{L2} + \left((\text{Miss rate})_{L2} * (\text{Miss penalty})_{L2}\right)$$

$$= 40 + \left[0.4 * \left[90 + (0.25 * \text{Main memory accessy time})\right]\right]$$

$$= 40 + \left[0.4 * \left[90 + (0.25 * 200)\right]\right]$$

$$= 40 + \left[0.4 * [90 + 50]\right]$$

$$= 40 + 56$$

$$= 96 \text{ ns}$$

Q7 **Consider a two-level cache system. CPU generates 1500 block request out of which 900 requests can be satisfied through level-1 cache. The time taken to access a block from level-1 cache is 4 ns. 400 out of the remaining requests are available in level-2 cache. Time taken by CPU to access a block from level-2 is 12 ns. If miss occurs at both level caches, CPU accesses the block from main memory in 18 ns. Then, the effective memory access time is _____ns.**

Sol: **8**

$$\text{EMAT} = \text{Hit time}_{L_1} + \text{Miss rate}_{L_1} * \text{Miss Penalty}_{L_1}$$

$$= \frac{900}{1500} * 4\text{ns} + \left(\frac{600}{1500} * \left(\text{Hit time}_{L_2} + \text{Miss rate}_{L_2} * \text{Miss Memory access time}\right)\right)$$

$$= \frac{900}{1500} * 4\text{ns} + \left(\frac{600}{1500} * \left(\frac{400}{600} * 12 + \frac{200}{600} * 18\right)\right)\text{ns}$$

$$= \frac{120}{15}\text{ns}$$

$$= 8\text{ns}$$

5.3 MAPPING TECHNIQUE:

Transferring the data (block) from the main memory to the cache memory is called mapping. There are three types of mapping:

a) Direct Mapping
b) Set Associative Mapping
c) Fully Set Associative Mapping

Note:

To denote cache memory, we will use shorthand notation as CM.

Direct mapping:

Basics:

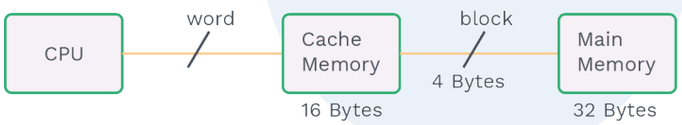


Fig. 5.9

Cache memory organization:

Let's say, cache memory size = 16 B
Block size = 4 B

$$\text{No. of blocks (lines) in CM} = \frac{\text{CM size}}{\text{Block size}} = \frac{16}{4} = 4 \text{ lines}$$

Total 16 Bytes of memory so cache index = $\log_2 16 = 4$ bits

Main memory organisation:

$$\text{No. of blocks in main memory} = \frac{\text{Main memory size}}{\text{Block size}} = \frac{32 \text{ Bytes}}{4 \text{ Bytes}} = 8$$

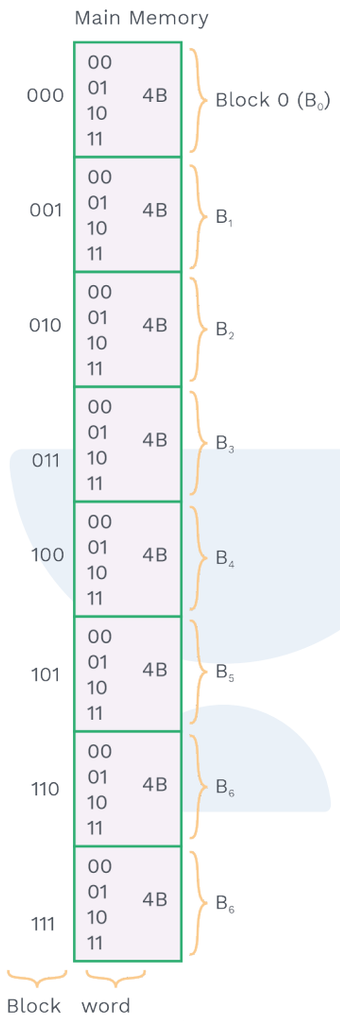
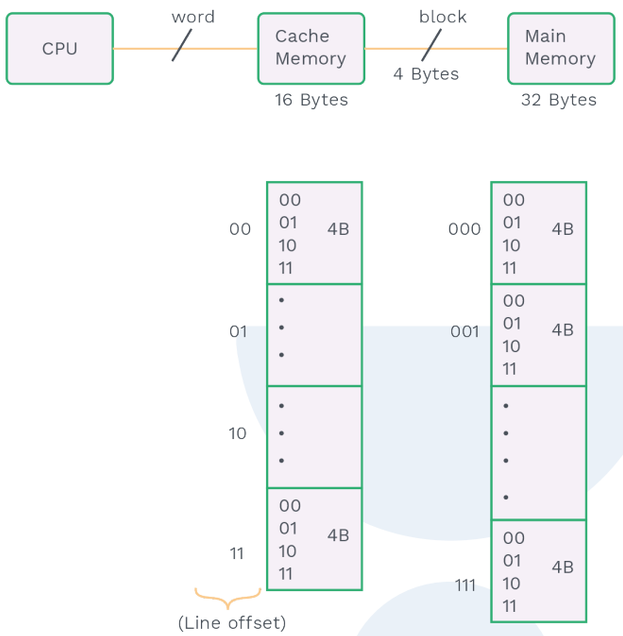


Fig. 5.10 Example of Block Organization of Main Memory

Physical address space = 32 bytes
Physical address = $\log_2 32$
= $\log_2 2^5$
= 5 bits

Note:

Mapping



"Copy" the data from MM to CM

Definition of direct mapping:

To map the data block from main memory to cache memory "Modulo (MOD)" function is used.

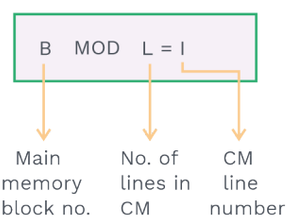
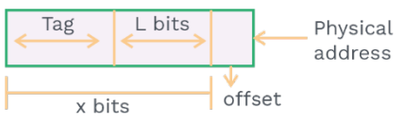


Fig. 5.11 Direct Mapping Function

Example:

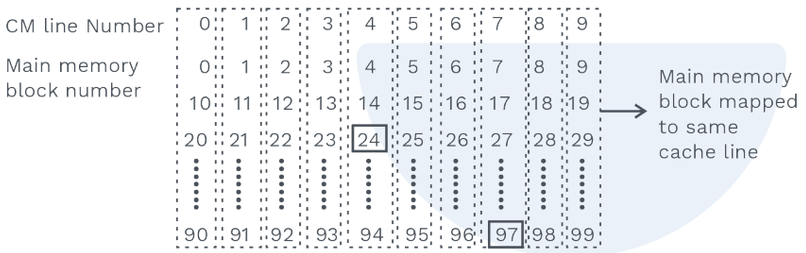
- No. of blocks in Main Memory = 2^x
- No. of cache lines = 2^L

2^x % 2^L will give least significant L bits from x bits which is nothing but the cache line number.



Example:

- CM line = 10 (0 to 9)
- Main memory blocks = 100 (0 to 99)



- Block 24 is mapped to cache line no. 4

24 mod 10 = 4

- Block 97 is mapped to cache line no. 7

97 mod 10 = 7

Calculate Word Offset, Line Offset and Tag

Block size = 4B

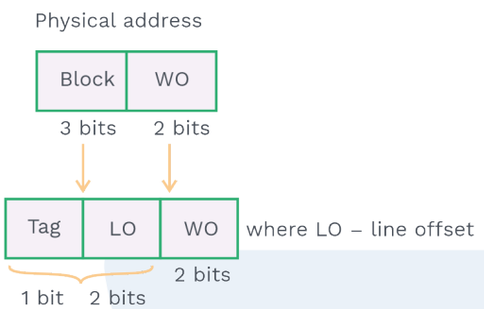
Number of blocks = 8 (0 to 7)

Number of lines = 4 (0 to 3)

Physical address = Block address + Word offset

= 3 + 2

= 5 bit



Note:

By default, we consider memory as byte addressable.

Note:

If some additional bit is given like valid, invalid, dirty, write back (updation) bits.
All these bits we add in tag space

Important formula:

$$\text{Cache memory size} = \text{No. of lines in cache memory} \times \text{Block size}$$

$$\text{Tag memory size} = \text{No. of lines in CM} \times \text{Tag space in the line}$$

$$\text{Cache index} = \text{Line offset} + \text{Word offset}$$

PRACTICE QUESTIONS

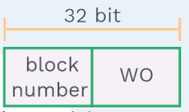
Q8 **Consider 16KB direct mapped cache organized into a 2KB data blocks. Physical address of memory is 32 bits. What is the tag size in bits and line offset(LO)? [Assume memory is byte addressable]**

a) 18, 3 **b) 21, 3** **c) 18, 11** **d) 3, 11**

Sol: **a)**

- Direct mapped cache
- Physical address = 32 bits
- Block size = 2KB = 2×2^{10} B
- Line offset = ?

Physical address format:

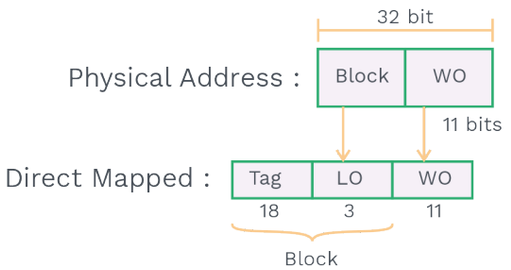


word offset (WO) = $\log_2 2^{11}$ = 11 bits

No. of lines (L) $= \frac{\text{Cache size}}{\text{Block size}}$

$= \frac{16\,\text{KB}}{2\,\text{KB}} = 8$

Line offset (LO)) = $\log_2 8$ = 3



Tag = 18 bits

LO = 3 bits

Hence option (A) is correct

Q9 **An 64KB direct mapped cache organize into 32 word blocks, word length of the CPU is 16 bits. CM data is subset of main memory of 8GB each tag is comprising with 1 valid bit and 1 update bit. What is size of tag directory and cache memory size in cache controller?**

a) 19456 bits, 32 MB **b) 19456 bits, 64 KB**

c) 9728 bits, 64 KB **d) None of these**

Sol: **b)**

- Direct mapped cache
- Cache memory size = 64 KB
- Block size = 32 Words
- Word length = 16 bit

So, Block size $= \frac{32 * 16}{8} = 64\,\text{Bytes}$

- Main memory size = 8 GB = $(2^3 \times 2^{30})$ Bytes
- Physical address = $\log_2 2^{33}$ = 33 bits
- Valid bit = 1
- Update bit = 1

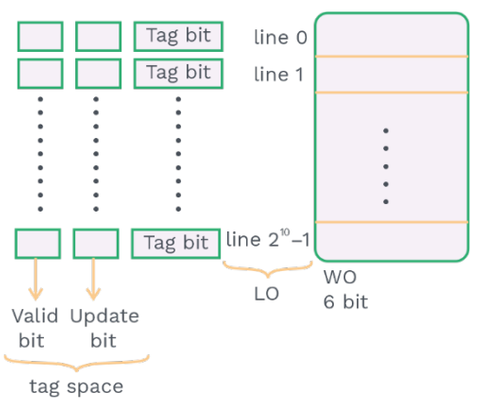
No. of lines in cache memory (L) $= \frac{\text{CM size}}{\text{block size}}$

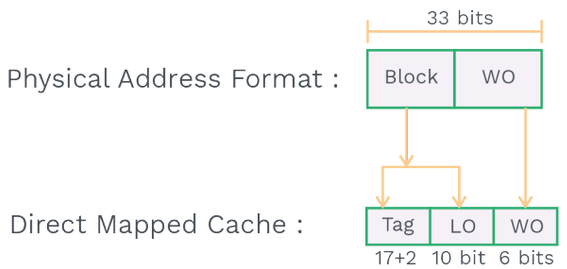
$= \frac{64\,\text{KB}}{64\,\text{bytes}}$

$= 1\text{K}$

$= 2^{10}$

Cache memory:





Tag directory size = Number of lines in cache memory * Tag space
= 2^{10} * 19 = 1024 * 19
= 19456 bits

Cache memory size = Number of lines in Cache Memory * block size
= 2^{10} * 2^{6}
= 64 KB

Hence option (B) is correct

Previous Years' Question

Consider a direct mapped cache of size 32 KB with block size 32 bytes. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively

a) 10, 17 **b)** 10, 22 **c)** 15, 17 **d)** 5, 17

Sol: c) **(GATE-2005 (2 Marks))**

Previous Years' Question

An 8KB direct–mapped write–back cache is organized as multiple blocks, each of size 32–bytes. The processor generates 32–bit address. The cache controlled maintains the tag information for each cache block comprising of the following.

1 Valid bit

1 Modified bit

As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controlled to store meta–data (tags) for the cache?

a) 4864 bits **b)** 6144 bits

c) 6656 bits **d)** 5376 bits

Sol: d) **(GATE-2011 (2 Marks))**

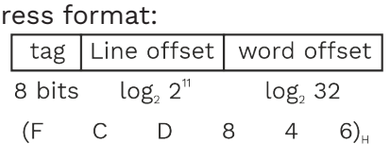
Q10 **Consider a 32-bit CPU which supports 64KB direct mapped cache. The cache is organized into 8 word blocks. The memory address $(FCD846)_H$ is mapped to cache line number-**

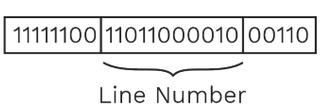
a) $(6C2)_H$ **b)** $(7C5)_H$ **c)** $(7C2)_H$ **d)** $(6C5)_H$

Sol: **a)**

Number of blocks in cache memory = $\frac{64\,KB}{8 \times 4\,B}$ = 216–5 = 211

Address format:





Line Number = 110 1100 0010

6 C 2

= $(6C2)_H$

5.4 HARDWARE DESIGN OF DIRECT MAPPING

Hardware implementation in direct mapping:

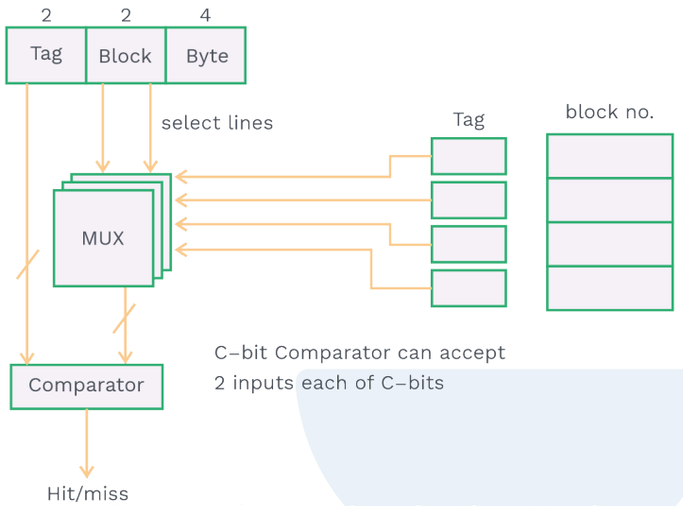


Fig. 5.12 Hardware Design using Direct Mapping

Number of select lines = Number of bits required for block number
Number of input lines = Number of blocks in CM
"C" bits comparator required
C → tag bits
here, C = 2
Number of MUX required for selecting the tag = tag bits
Size of MUX = Number of blocks in cache: 1

Previous Years' Question

Consider a machine with byte addressable memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume a direct mapped cache having 512 cache lines are used with this machine. The size of the tag field in bits is
Sol: 18 **(GATE-2005 (2 Marks))**

Hit latency in direct mapping:
Total time to check whether the block hit or miss is comparator time and MUX time, is known as hit latency.

$$\text{Hit latency} = \text{Multiplexer delay} + \text{Comparator delay}$$

Set associative mapping:
To check hit/miss in set associative mapping, three things are required:

a) Multiplexer:
This mapping requires R–way sets of MUX. In each set number of MUX based on the number of sets.

b) Comparator:
In this mapping R(R–way) comparator is needed.

c) OR gate:
To check hit/miss 1 or gate, it is needed to add comparator bits.

Set associative mapping
Set associative mapping is made with the combination of direct and associative mapping. Associative mapping is a much more flexible mapping method because the main memory block has 'S' choice to place in cache memory (S means S–way set associative).

Design of 2–way set associative:

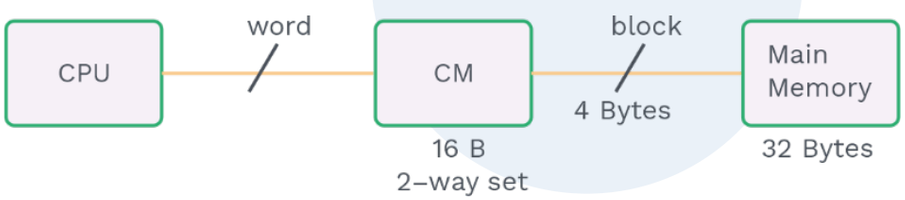


Fig. 5.13

- No. of blocks in main memory = $\frac{\text{Main memory size}}{\text{Block size}}$

$$= \frac{32\,B}{4\,B}$$

$$= 8$$

- No. of lines in cache memory (L) = $\frac{\text{CM size}}{\text{Block size}}$

$$= \frac{16\,B}{4\,B}$$

$$= 4$$

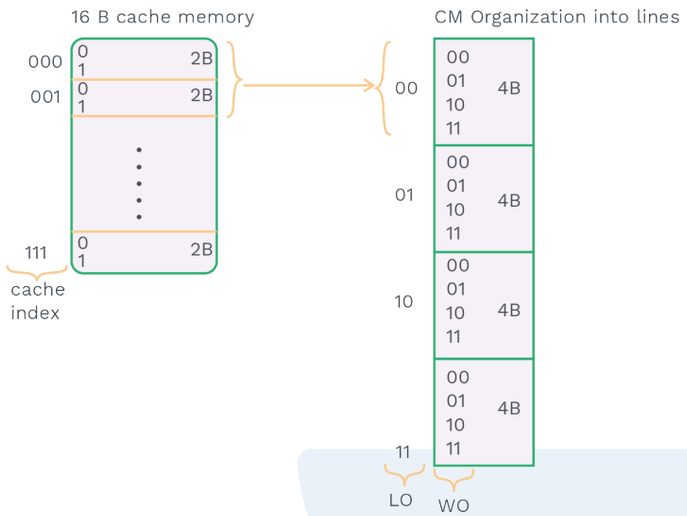


Fig. 5.14

CM organized into sets:

$$\text{No. of sets}(S) = \frac{L}{R - \text{way}} = \frac{\text{No. of lines}}{R}$$

$$= \frac{4}{2}$$

$$= 2$$

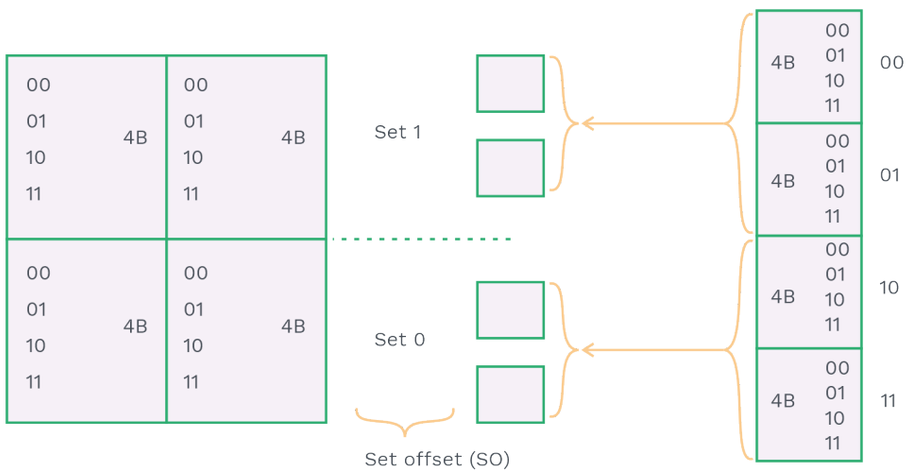


Fig. 5.15

Mapping function:

We use the mod function to map the main memory (MM) block number.

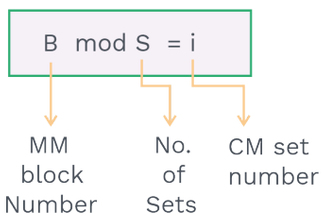
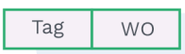


Fig. 5.16 Set Associative Mapping Function

- Main memory address: 

- Main memory = 32 Bytes, main memory address in bits = 5 bits
- Block size = 4B, Word offset = 2 bits
- No. of sets = 2, set offset = $\log_2 2 = 1$

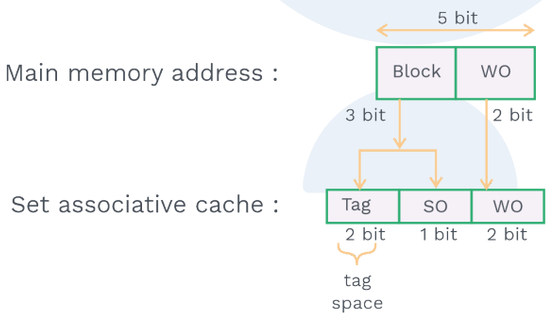


Fig. 5.17

Formulas:

$$\text{Tag directory size} = \text{Number of blocks in CM} * (\text{Tag bits} + \text{extra bit if present})$$

PRACTICE QUESTIONS

Q11 **Consider a cache with capacity of 64 KB, 8–way set associative with block size of 32 bytes. The processor sends 32 bit address to the cache controller, each tag directory entry contains 2 valid bits and 1 replacement bit. Then the size of the cache tag directory is _____ K bits.**

Sol: **44**

- 8–way set associative mapping
- CM size = 64 KB
- Block size = 32 bytes
- Physical address = 32 bits
- Number of extra bits = 2 + 1 = 3 bits

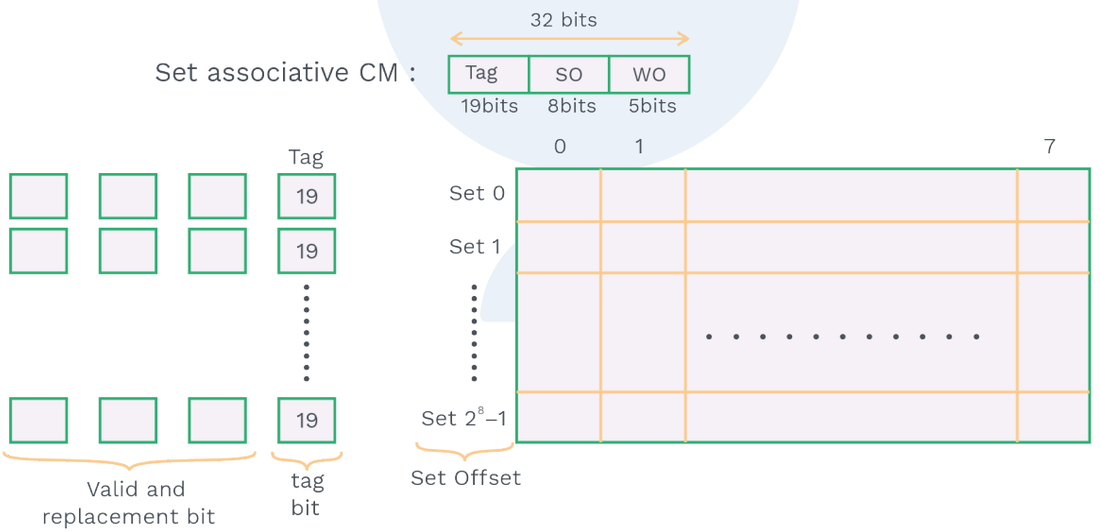
Number of lines (L) = $\frac{\text{CM size}}{\text{Block size}} = \frac{64\,\text{KB}}{32\,\text{B}} = 2^{11}$

Number of sets (S) = $\frac{L}{R - \text{way}}$

$= \frac{2^{11}}{8}$

$= 2^8$

Set associative CM :



Tag directory size = No. of blocks in CM * (Tag + extra) bits
= No. of lines * (19 + 3) bits
= 2^{11} * 22
= 44 K bits

Q12 **Consider 4–way set associative cache with the capacity of 512 byte. CPU generates 32 bits physical address and block size is 16 bytes then calculate the following term.**
a) No. of bits in the tag field

b) Size of tag directory

Sol: **800**

4–way set associative

- CM size = 512 bytes
- Block size = 16 bytes
- Physical address = 32 bits
- No. of lines (L) in cache = $\frac{\text{Cache size}}{\text{Block size}} = \frac{512\,B}{16\,B} = 32$

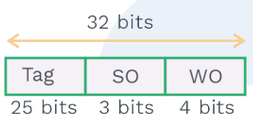
No. of sets (S) = $\frac{L}{R - way}$

$= \frac{32}{4}$

$= 8$

Set offset = $\log_2 8$
= 3 bits

Set associative cache :



a) tag space = 25 bits
b) Tag directory size = No. of lines (in CM) * Tag space
= 32 * 25
= 800 bits

Q13 **Consider a system implementing a 2-way set associative cache with 27 blocks. It uses least recently used policy for replacement. Conflict misses occur when two or more blocks contend for the same cache set. First refer-**

ence miss occurs when a block is accessed for the very first time. The memory access sequence (0, 64, 128, 64, 0, 64, 128, 64, 165, 129, 65, 1, 65, 129, 65) is repeated 2 times. The number of cache conflict misses and miss ratio in the cache are_____, _____.

a) 16, 50.00% **b) 10, 62.50%**

c) 14, 47.25% **d) 18, 56.25%**

Sol: **c)**

Number of sets = $\frac{128}{2} = 64$.

0	~~0~~ ~~128~~ ~~0~~ 128	64
1	~~1~~ ~~129~~ ~~1~~ 129	65
2		
.	.	.
.	.	.
.	.	.
63		

→

0	~~0~~ ~~128~~ ~~0~~ ~~128~~ ~~0~~ ~~128~~ ~~0~~ 128	64
1	~~1~~ ~~129~~ ~~1~~ ~~129~~ ~~1~~ ~~129~~ ~~1~~ 129	65
2		
.	.	.
.	.	.
.	.	.
63		

0 mod 64 = 0 → First ref Miss
64 mod 64 = 0 → First ref Miss
128 mod 64 = 0 → conflict miss, LRU is used, so 128 replaces 0
64 → Hit
0 mod 64 = 0 → conflict miss, LRU is used, so 0 replaces 128
64 → Hit
128 mod 64 = 0 → conflict miss, LRU is used, so 128 replaces 0.
64 → Hit
1 mod 64 = 1 → First ref miss
65 mod 64 = 1 → First ref miss
129 mod 64 = 1 → conflict miss, LRU is used, so 129 replaces 1.
65 → Hit
1 mod 64 = 1 → conflict miss, LRU is used, so 1 replaces 129.
65 → Hit
129 mod 64 = 1 → conflict miss, LRU is used, so 129 replaces 1.
65 → Hit
0 mod 64 = 0 → conflict miss, LRU is used, so 0 replaces 128.
64 → Hit
128 mod 64 = 0 → conflict miss, LRU is used, so 128 replaces 0.
64 → Hit

0 mod 64 = 0 → conflict miss
64 → Hit
128 mod 64 = 0 → conflict miss
64 → Hit
1 mod 64 = 1 → conflict miss
65 → Hit
129 mod 64 = 1 → conflict miss
65 → Hit
1 mod 64 = 1 → conflict miss
65 → Hit
129 mod 64 = 1 → conflict miss
65 → Hit

∴ Total number of references = 32
Number of misses = 18

∴ Miss ratio = $\frac{18}{32} = \frac{9}{16} = 47.25\%$

Number of conflict misses = 14

Previous Years' Question

In a k–way set associative cache, the cache is divided into v sets, each of which consists of k lines. The lines of a set are placed in the sequence of one after another. The lines in the set s are sequenced before the lines in the set (s+1). The main memory blocks are numbered 0 onwards. The main memory block numbered j must be mapped to any one of the cache lines from

a) (j mod v) * k to (j mod v) * k + (k–1)

b) (j mod v) to (j mod v) + (k–1)

c) (j mod k) to (j mod k) + (v–1)

d) (j mod k) * v to (j mod k) * v + (v–1)

Sol: a) **(GATE-2013 (1 Mark))**

5.5 HARDWARE IMPLEMENTATION OF SET ASSOCIATIVE MAPPING

Basic diagram to check hit/miss in set associative mapping:

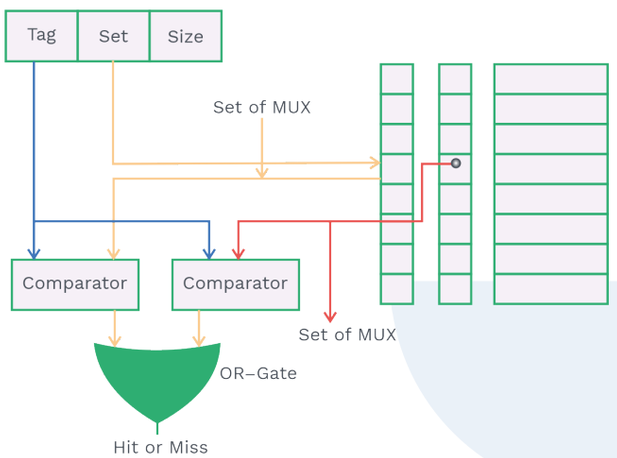


Fig. 5.18 Checking Hit/Miss in Associative Mapping

Previous Years' Question

The width of the physical address on a machine is 40 bits. The width of the tag field in a 512 KB 8–way set associative cache is _____ bits.

Sol: Range 24 to 24

(GATE-2016 (Set–2) (2 Marks))

Hardware implementation in set associative

No. of blocks in CM = 8

and 2–way set associative

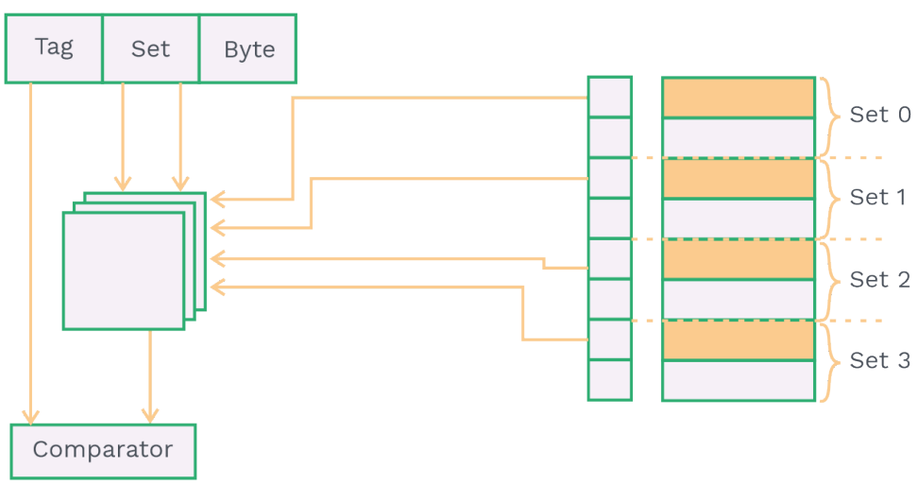


Fig. 5.19 Hardware Design used in Associative Mapping

Hardware implementation in set associative mapping

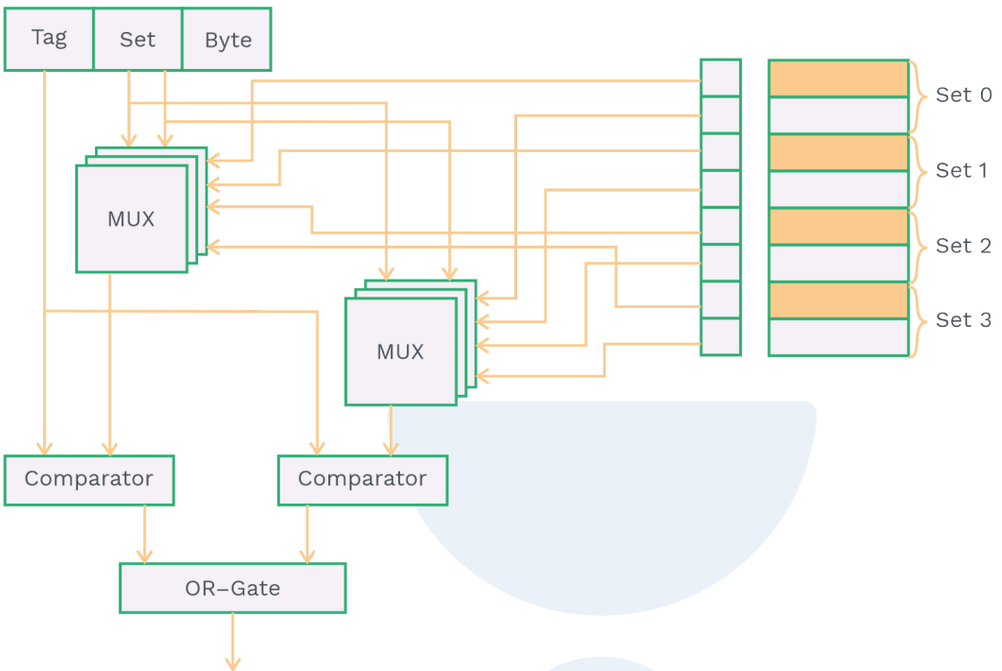


Fig. 5.20 Hardware Design using Set-Associative Mapping

Hardware Size for R–way Set Associative

a) Multiplexer (MUX):
No. of MUX = R * Number of Tag bits
Size of MUX = No. of sets in cache: 1

b) Comparator:
No. of comparator = R
Size of comparator = Number of Tag bits

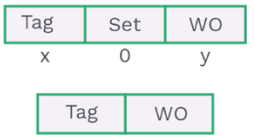
c) OR gate:
No. of gates = 1
No. of inputs = R

Hit latency in set associative mapping:

$$\text{Total time to check hit / miss (Hit latency)} = \text{MUX delay} + \text{Comparator delay} + \text{OR gate delay}$$

Fully associative mapping:
In fully associative, all the blocks of the main memory mapped with a single set. So there is no need of set index. i.e. set index bit is 0 (2^0 = 1 set)

Eg:



* Index bit comparison in all mapping techniques.

Technique	Index bit
Direct mapping	Cache line number bit
Set associative mapping	set offset bit
Fully associative mapping	0–bit

Table 5.1 Index Bits Reserved in Different Mapping Techniques

Formula:

Tag directory size = No. of cache blocks * Tag bits

PRACTICE QUESTIONS

Q14 **Consider the following parameters**
Cache memory size = 64 KB
Block size = 32 bytes
Physical address = 34 bits
Calculate the tag directory size in
a) Direct mapping
b) Fully associative mapping
c) 8–way set associative mapping

Sol: No. of blocks in CM (L) = $\frac{\text{CM size}}{\text{Block size}} = \frac{64\,\text{KB}}{32\,\text{B}}$

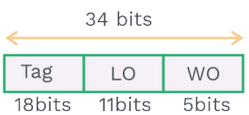
= 2K

L = 2^{11}

Line offset(LO) = 11 bits

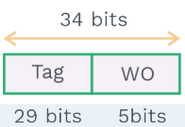
Word offset (WO) = 5 bits

a) Direct mapping:



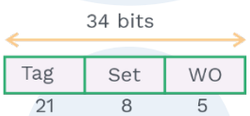
Tag directory size = No. of lines in CM * Tag space
= 2^{11} * 18
= 36 K bits

b) Fully associative cache:



Tag directory size = No. of lines in CM * Tag bits
= 2^{11} * 29
= 58 K bits

c) 8–way set associative cache:



No. of sets = $\frac{\text{No. of lines (L)}}{\text{R – way set}} = \frac{2^{11}}{8}$

= 2^8

Set offset = 8 bit

Tag directory size = No. of lines in CM * Tag space
= 2^{11} * 21
= 42K bits

Q15 **Consider fully associative cache memory size of 16KB having 4 KB data blocks. Consider the following references blocks 8, 13, 16, 6, 8, 15, 20, 23, 29, 27, 23, 16, 20 What is the hit ratio if the cache is initially empty and designed with LRU (Least recently used)?**

Sol: **15.38**

Given: Fully associative cache

CM size = 16 KB

Block size = 4 KB

No. of lines in CM (L) = $\frac{\text{CM size}}{\text{Block size}} = \frac{16\,\text{KB}}{4\,\text{KB}} = 4$

CM

~~8~~	23		
~~16~~	~~20~~	16	
~~13~~	~~15~~	27	
~~8~~	~~29~~	20	

8 – Miss
13 – Miss
16 – Miss
6 – Miss
8 – Hit
15 – Miss
20 – Miss
23 – Miss
29 – Miss
27 – Miss
23 – Hit
16 – Miss
20 – Miss

Hit ratio = $\frac{\text{No. Hits}}{\text{Total block reference}} = \frac{2}{13} * 100 = 15.38$

Previous Years' Question

A certain processor uses a fully associative cache of size 16 KB, the cache block size is 16 bytes. Assume that the main memory is byte addressable and uses a 32–bit address. How many bits are required for the tag and the index fields respectively in the addresses generated by the processor?

a) 24 bits and 0 bits **b)** 28 bits and 4 bits
c) 24 bits and 4 bits **d)** 28 bits and 0 bits

Sol: d) **(GATE-2016 (Set–2) (2 Marks))**

5.6 HARDWARE IMPLEMENTATION OF FULLY ASSOCIATIVE

Fully associative mapping:

Checking hit/miss in fully associative mapping:

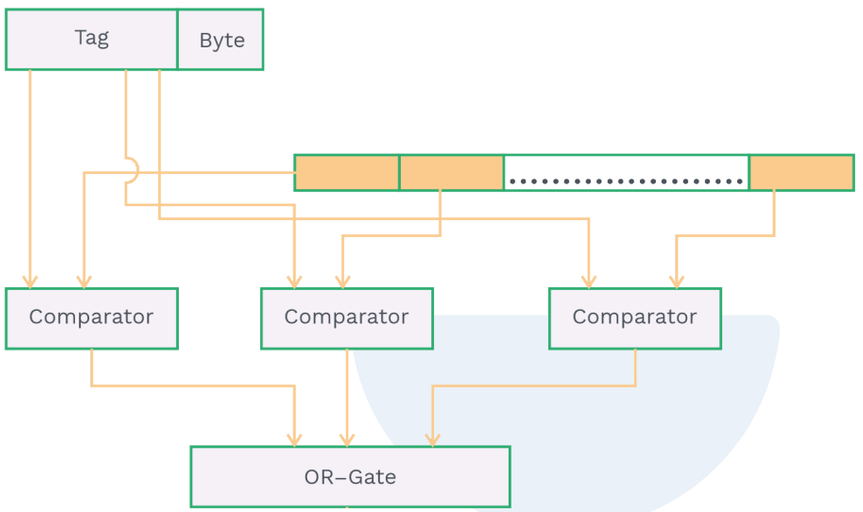


Fig. 5.21 Hardware Design used in Fully Associative Mapping

Note:

We don't need MUX here because we don't have any concept of choosing sets/lines.

Hardware size for fully associative mapping:

a) Comparator:

No. of comparator = No. of blocks in CM

Size of comparator = Tag bits

b) OR gate:

No. of OR gates = 1

Size of OR gates = No. of blocks in cache

Hit latency in fully associative mapping:

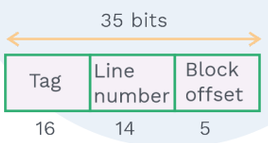
$$\text{Total time to check Hit / miss (Hit latency)} = \text{Comparator delay} + \text{OR gate delay}$$

PRACTICE QUESTIONS

Q16 **Consider 512 KB cache in direct mapping. Block size of the cache is 32 bytes and CPU generates 35 bit physical address then calculate**
No. of MUX and size of MUX in direct and also calculate same parameters in 4–way set associative cache.
No. of comparator and size of comparator in all three mapping technique.
No. of OR gate and input size of OR gate in 4–way set associative cache and fully associative cache.

Sol: Number of lines = $\frac{512\,KB}{32\,B} = \frac{2^9 \times 2^{10}}{2^5} = 2^{14}$

1) Direct mapping:



a) No. of multiplexer (MUX) = No. of bits in tag
No. of MUX = 16
b) Size of MUX = No. of blocks in CM: 1
$= 2^{14}: 1$
c) No. of comparator = 1
d) Size of comparator = 16 bits

2) 4–way set associative:

Number of sets = $\frac{2^{14}}{2^2} = 2^{12}$

a) Number of MUX = R * Number of bits in tag space
$= 4 * 18 = 72$
b) Size of MUX = No. of sets: 1 = $2^{12}: 1$
c) Number of comparator = 4
d) Size of comparator = 18 bits
e) Number of OR gate = 1
f) Number of input for OR gate = 4

3) Fully associative:

a) No. of comparator = No. of blocks = 2^{14}
b) Size of comparator = 30 bits
c) No. of OR gate = 1
d) Size of OR gate = 2^{14} inputs

Previous Years' Question

Consider two cache organizations: The first one is 32 KB 2–way set associative with 32–byte block size. The second one is of the same size but direct mapped. The size of an address is 32 bits in both cases. A 2–to–1 multiplexer has a latency of 0.6 ns while a k bit comparator has a latency of k/10 ns. The hit latency of the set associative organization is h1 while that of the direct mapped one is h2.
The value of h1 is:

a) 2.4 ns **b)** 2.3 ns
c) 1.8 ns **d)** 1.7 ns

Sol: a) **(GATE-2006 (2 Marks))**

5.7 TYPES OF CACHE MISS

1) Compulsory miss (cold start miss):

First time access of a block will always cause a miss means every first time reference block is not present in Cache Memory during program execution.

- Compulsory misses can be reduced by increasing the cache block size.

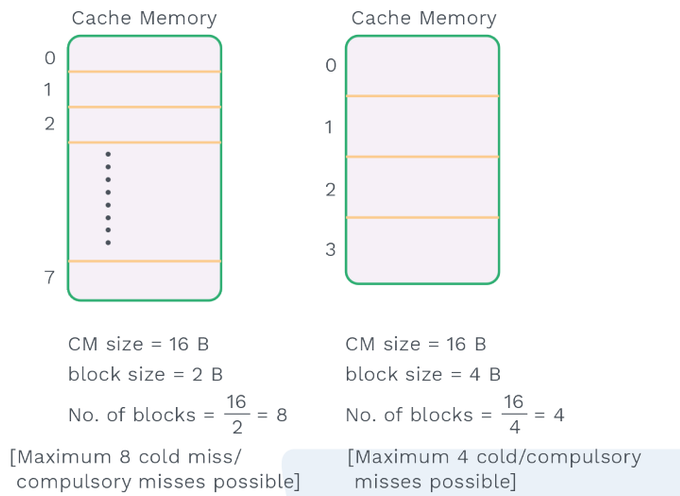


Fig. 5.22

Note:

If block size increases, miss penalty time will increase because transferring time of block will increase; so time will increase.

2) Capacity miss:

Capacity miss occurs when the cache is full, and its' not a compulsory miss. Capacity miss occurs high in a fully associative cache.

- Capacity misses can be reduced by increasing the cache memory size because the cache block will increase.

3) Conflict miss:

This miss occurs when the cache set is full, and many blocks are mapped in the same cache line/block.

- Conflict misses can be reduced by increasing (doubling) the set associativity in set associative cache.

Suppose reference blocks are: 4, 8, 12, 16, 4, 8
In direct mapping, every block in the main memory is mapped to a particular cache line. So, conflict misses are more.
It is medium in set associative mapping.

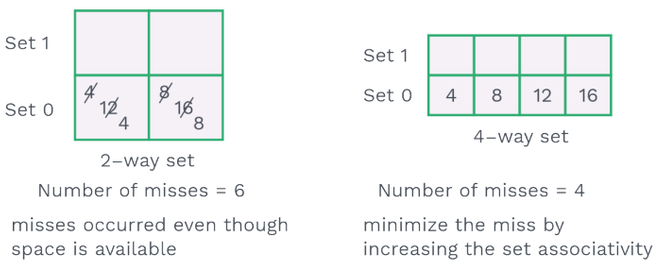


Fig. 5.23

Note:

Conflict miss does not occur in fully set–associative cache.

Previous Years' Question

Consider a 2–way set associative cache with 256 blocks and uses LRU replacement, Initially, the cache is empty. Conflict misses are those misses which occur due to the contention of multiple blocks for the same cache set. Compulsory misses occur due to first time access to the block. The following sequence of accesses to memory blocks
(0, 128, 256, 128, 0, 128, 256, 128, 1, 129, 257, 129, 1, 129, 257, 129)
is repeated 10 times. The number of conflict misses experienced by the cache is ___

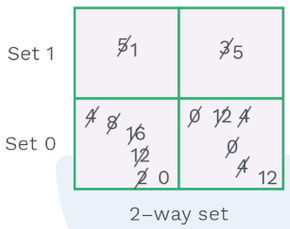
Sol: 76 **(GATE-2017 (Set – 1) 2 Marks)**

PRACTICE QUESTIONS

Q17 **Consider a 2–way set associative cache with block size 4B and cache size 16B CPU requests for main memory blocks in following orders**
4, 0, 8, 12, 16, 4, 12, 0, 5, 3, 1, 2, 4, 5, 0, 12 and 5.
Cache is initially empty then calculate these miss using LRU policy
a) Conflict miss
b) Capacity miss
c) Compulsory/cold miss

Sol: 2–way set associative

- Cache size = 16B
- Block size = 4B
- $\text{No. of blocks} = \frac{\text{Cache size}}{\text{Block size}} = \frac{16\,B}{4\,B} = 4$
- $\text{No. of sets(S)} = \frac{N}{R-\text{way}} = \frac{\text{No. of blocks}}{2} = \frac{4}{2} = 2$



2–way set

4 → Compulsory miss
0 → Compulsory miss
8 → Compulsory miss
12 → Compulsory miss
16 → Compulsory miss
} 5 compulsory miss

4 → Conflict miss
12 → Conflict miss
0 → Conflict miss
} 3 conflict miss

5 → Compulsory miss
3 → Compulsory miss
1 → Compulsory miss
2 → Compulsory miss
} 4 compulsory miss

4 → Capacity miss
5 → Capacity miss
0 → Capacity miss
12 → Capacity miss
} 4 capacity miss

5 → Hit

Total no. of conflict miss = 3
Total no. of capacity miss = 4
Total no. of compulsory miss = 9

Note:

Cache line size = Main memory block size
No. of cache lines = Number of blocks in cache memory

Cache coherence problem:
Distinct images of the same data may be present in the cache and main memory at some instant of time and if processors are allowed to update their own copies freely, then an inconsistent view of memory can result.

5.8 UPDATING TECHNIQUES

There are two updating techniques (Protocols)

1) Write through Protocol

2) Write back the protocol

These protocols are used to avoid data inconsistent problems.

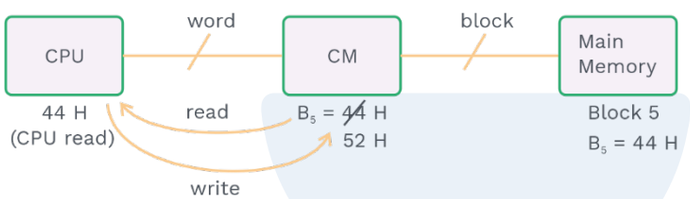


Fig. 5.24

In MM block B_5 address is 44 H but in cache B_5 = 52 H this is known as cache coherence (data inconsistent)

1) Write through protocol:
To avoid data inconsistent problem, cache and main memory location updated simultaneously by CPU.

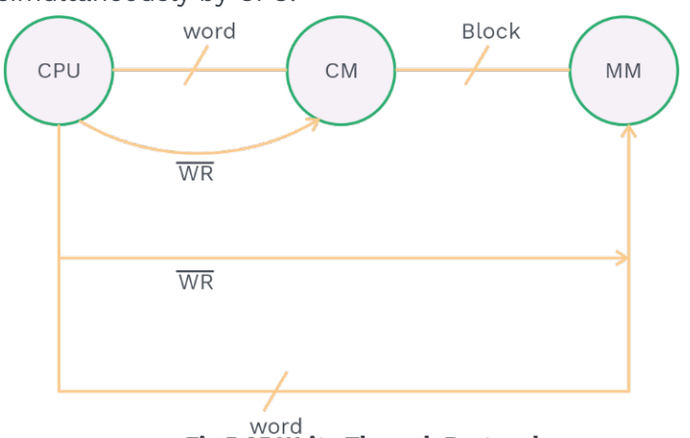


Fig 5.25 Write Through Protocol

- Here, cache and main memory both is updated simultaneously by processor After updating byte/word is accessed from cache memory.

Read cycle time:

- If time taken by cache memory to read the data is C and time taken by main memory is M. Hit ratio of read operation is H_r. Then average read cycle time is

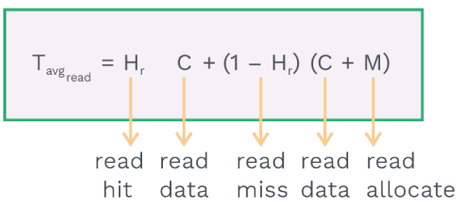


Fig. 5.26 Calculation of Read Cycle Time using Write Through Protocol

Write cycle time:

- If simultaneous write operation time is W and hit ratio of write operation is H_w the average write cycle time

Case i:

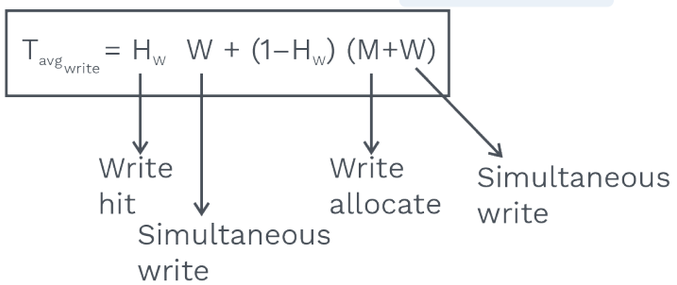
If no write allocation policy

$$T_{avg_{write}} = \text{Max}\,(\text{cache access time}, \text{Main memory access time})$$

$$T_{avg_{write}} = \text{Main memory access time}$$

Case ii:

If write allocation policy



- If frequency of read operation is F_r and frequency of write operation is F_w then the average write through protocol and efficiency.

Average time:

$$T_{avg_{write\ through}} = \left(T_{avg_{read}} * F_r\right) + \left(T_{avg_{write}} * F_w\right)$$

Efficiency (η):

$$\eta = \frac{1}{T_{avg_{write\ through}}} \text{WPS}$$ Where WPS = word/sec

Note:

$$\text{Simultaneous word updation Time}(w) = \max\left(\begin{matrix}\text{word updation,}\\ \text{time in CM}\end{matrix}\ \begin{matrix}\text{word updation}\\ \text{time in main memory}\end{matrix}\right)$$

- If hit ratio of write operation is 100% then use simultaneous access.

C → Cache Memory Access Time

M → Main Memory Access Time

$$T_{avg_{read}} = HC + (1-H)M$$

$$T_{avg_{write}} = W$$

2) Write back protocol:

In the write back protocol, the cache location is only updated and mark it as updated with an associated flag bit called dirty or modified bit. Main memory allocation is updated when the block containing this marked word is to be removed from the cache to make space for a new block. This is also known as copy back protocol.

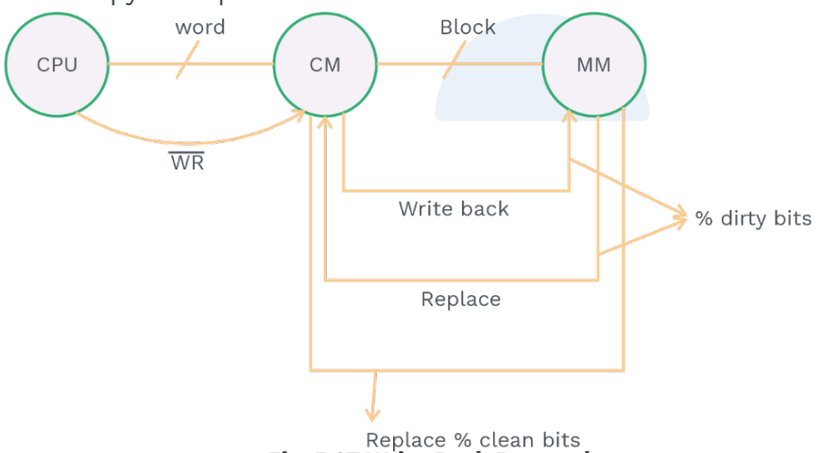


Fig. 5.27 Write Back Protocol

Read cycle time:

If the hit ratio of read operation is "H". Cache memory access time is "C" and main memory access time is "M".

$$T_{avg_{read}} = HC + (1-H)\left[\begin{matrix}\text{\% dirty}\\ \text{bits}\end{matrix}\ \begin{matrix}(C+M+M)\\ \text{read data}\ \ \text{read allocate}\ \ \text{write back}\end{matrix}\ \text{+ \% clean bits } (C+M)\right]$$

Write cycle time:
If hit ratio of write the data is H_w the calculate write cycle time

$$T_{avg_{write}} = H_w\, C + (1 - H_w)\left[\%\ \text{dirty bits}\ (C + M + M) + \%\ \text{clean bits}\ (C + M)\right]$$

(C + M + M): write data, write allocate, write back

Average access time of write back:
If frequency of read operation is "F_r" and frequency of write operation is F_w then

$$T_{avg_{writeback}} = \left(T_{avg_{read}} * F_r\right) + \left(T_{avg_{write}} * F_w\right)$$

PRACTICE QUESTIONS

Q18 **Consider 64KB 4–way set associative write through cache. CPU generates 34 bit physical address, 20% write requests and remaining are used for read requests. Hit ratio of read and write operation 50% and 60% Cache Memory access time 100 ns and main memory access time is 950 ns. What is the average memory access time when considering both read and write operation?**

Sol: **726**

- 4–way set associative cache
- CM size = 64 KB
- Physical Address = 34 bits
- F_w = 20%
- F_r = 80%
- H_r = 50%
- H_w = 60%
- C = 100 ns
- M = 950
- Block size not given assumes 1 w.
- Write through cache.

Read cycle time:

$$T_{avg_{read}} = H_r * C + (1 - H_r)(C + M)$$

= 0.5 * 100 + 0.5 * (100+950)
= 50 + 1050 * 0.5
= 50 + 525
= 575 ns

Write cycle time:

Simultaneous word updation time (w) = $\max(100, 950)$

w = 950 ns

$$T_{avg_{write}} = H_w * w + (1 - H_w)[M + w]$$

= 0.6 * 950 + 0.4 * 1900 = 570 + 760
= 1330 ns

Average access time:

$$T_{avg_{write\ through}} = (T_{avg_{read}} * F_r) + (T_{avg_{write}} * F_w)$$

= 575 * 0.8 + 1330 * 0.2 = 460 + 266
= 726 ns

Q19 **Consider a system write back cache which is used in the 36 bit CPU. Size of cache memory is 32 KB. CPU generates 60% read request and 40% write requests. Access time of cache and main memory is 60 and 800 respectively. When miss occurred in CM then 8w data block is transferred from physical memory to cache memory. Hit ratio of read and write operation is 60% and 80%. What is the average memory access time?**

Sol: **418.4**

- Write back
- Cache memory size = 32 KB
- Word size = 36 bits
- Block size = 8w
- Fr = 60% (% clean bits)
- F_w = 40% (% dirty bits)
- C = 60 ns
- M = 800 ns
- H_w = 80%
- H = 60%

Read cycle time:

$$T_{avg_{read}} = H * C + (1 - H)\left[\%\,\text{dirty bits}\,(C + M + M) + \%\,\text{clean bits}\,(C + M)\right]$$

$= .6 * 60 + .4\ [.4 * 1660 + .6 * 860]$

$= 508$ n

Write cycle time:

$$T_{avg_{write}} = H_w C + (1 - H_w)\left[\%\,\text{dirty bits}\,(C + M + M) + \%\,\text{clean bits}\,(C + M)\right]$$

$= .8 * 60 + .2\ [.4 * 1660 + .6 * 860]$

$= 284$ ns

Average access time:

$$T_{avg_{writeback}} = \left(T_{avg_{read}} * F_r\right) + \left(T_{avg_{write}} * F_w\right)$$

$= 508 * .6 + 284 * .4$

$= 418.4$ ns

Q20 **Consider a cache memory of size 16 KB and block size is 2 words. Cache is designed using write through protocol with the access time of cache memory and main memory as 20 and 250 ns, respectively. Hit ratio of read and write operations are 80% and 60%, respectively. If read cycle time is X ns and write cycle time is Y ns, then calculate the X + 6Y in nanosecond.**

a) 1420 ns **b) 295 ns** **c) 258 ns** **d) 790 ns**

Sol: **a)**

Given:

CM size = 16 KB

Block size = 2 words

$T_{CM} = 20$ ns

$T_{MM} = 250$ ns

$H_r = 80\%$

$H_w = 60\%$

Write through protocol:

Simultaneous write operation time (T_w) = Max (Word updation time in CM, word updation time in MM)

T_w = Max (20 ns, 2502)

$T_w = 125$ ns

Read cycle time (X)

read avg $T = H_r\, T_{CM} + (1 - H_r)\,(T_{CM} + T_{MM})$

$= 0.8 \times 20 + 0.2 \times 270 = 16 + 54$

$= 70$

X = 70 ns

Write cycle time (Y)

write avg T = $H_w T_w + (1 - H_w) (T_{MM} + T_w)$

= 0.6 × 125 + 0.4 × 375

= 225 ns

Y = 225 ns

So, X + 6Y = 70 + 6 × 225

= 1420 ns

Cache inclusion policies:

There is two kind of policies

1) Inclusion

2) Exclusion

1) Inclusion:

In multilevel cache, higher level (L_1) content is present in lower level (L_2). Thus, we can say the higher level cache content is subset of the lower level cache.

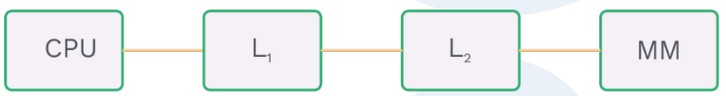


Fig. 5.28

What happened if:

a) Hit in L_1:

CPU reads the block from Level–1 only.

b) Miss in L_1 and Hit in L_2:

Copy the missed block from L_2 to L_1. There may be a block evicted from L_1 and there is no any kind of involvement of L_2 in this case.

c) Miss in L_1 and miss in L_2:

Copy the block in L_2 and L_1 from main memory.

- L_2's evicted block will be made invalid in L_1 (to make inclusion)
- L_1's evicted block has no involvement of L_2.

2) Exclusion:

In multilevel cache, higher level (L_1) content should not necessarily be present in L_2.

Fig. 5.29

What happened if:

a) Hit in Level–1 (L_1):

CPU reads the word from L_1 only.

b) Miss in L_1 and Hit in L_2:

Transfer the block from L_2 to L_1. The evicted block is moved to L_2.

c) Miss in L_1 and miss in L_2:

Copy the block from main memory to level–1 (L_1) only no need to copy in L_2 (invalid block in L_2) because of exclusion.

Chapter Summary

- Memory hierarchy design is based on size, cost, speed etc.
- In simultaneous access CPU can access the data from any level directly.
- In hierarchy design, block is transferred from lower level to higher level cache.
- Cache is small, fast and costly than main memory.
- Performance of the cache depends on:
  - **a)** Size of the cache
  - **b)** Size of the block
  - **c)** Levels of the cache
  - **d)** Mapping techniques
  - **e)** Replacement policy
  - **f)** Updation policy

- **Mapping technique:**
  - **a)** Direct mapping
  - **b)** Set associative mapping

- **Fully associative mapping:**
  - **a)** '0' bit is needed to represent the line in a fully set associative.
  - **b)** Compulsory misses can be reduced by increasing the block size.
  - **c)** Capacity miss occurs when the cache is full, and it should not be a compulsory miss. These misses can be reduced by increasing the cache size.
  - **d)** Conflict misses occurred in R–way set associative mapping; these misses can be reduced by increasing the set associativity.
  - **d)** Hit latency in direct mapping depends on two factor comparator delay and MUX delay.
  - **e)** Hit latency in R–way set associative mapping depends on three factors MUX delay, comparator delay and OR gate delay.
  - **f)** Hit latency in fully associative mapping depends on comparator and OR gate delay.

- **Cache coherence:**
  Data inconsistent problem is known as a cache coherence problem. To avoid the cache, coherence updating techniques are used (write through and write back).

6 IO Interface DMA Transfer and Secondary Storage

6.1 SECONDARY MEMORY

Disk

Magnetic disk:

- Magnetic disk is a storage device. The platter of the magnetic disk has both side recording (storage) surfaces. Getting the track is random access, whereas getting the desired sector of the track is sequential access.

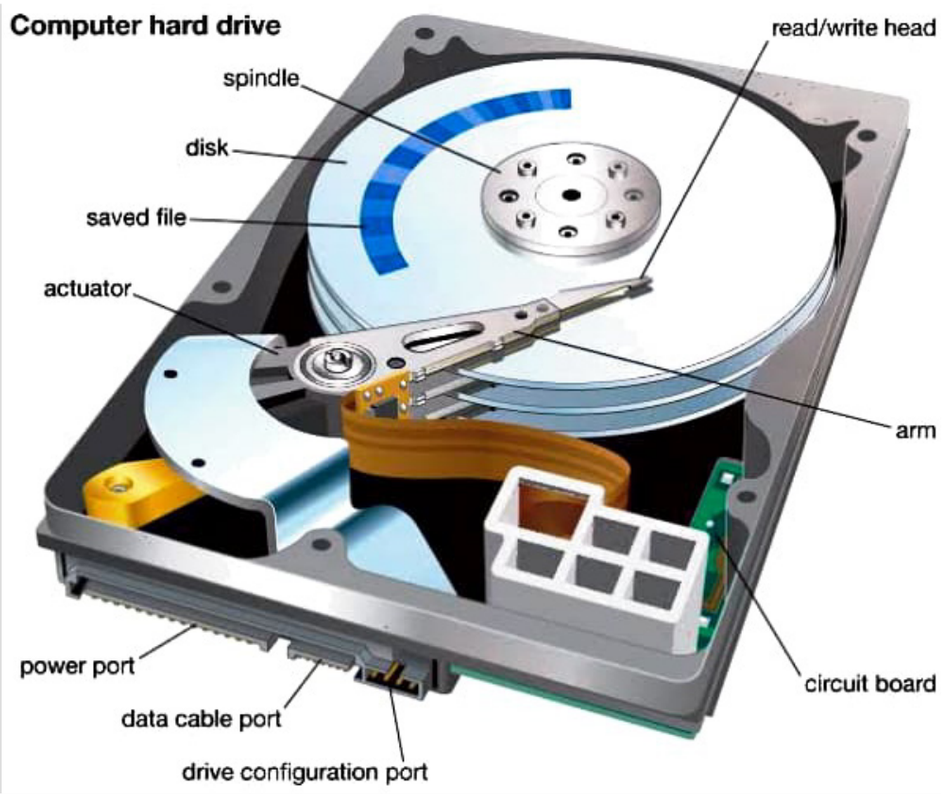


Fig. 6.1 Disk Hardware

Physical structure of the disk:

- Platter has two surfaces, both the surfaces are storage surfaces.
- All the platters are mounted with one common spindle.
- To perform the read/write operation, one pointer is needed at every surface. That pointer is called Read/Write head.
- All Read/Write heads are attached to one common arm assembly.
- Read/Write head can move only in two ways, either forward or backwards.

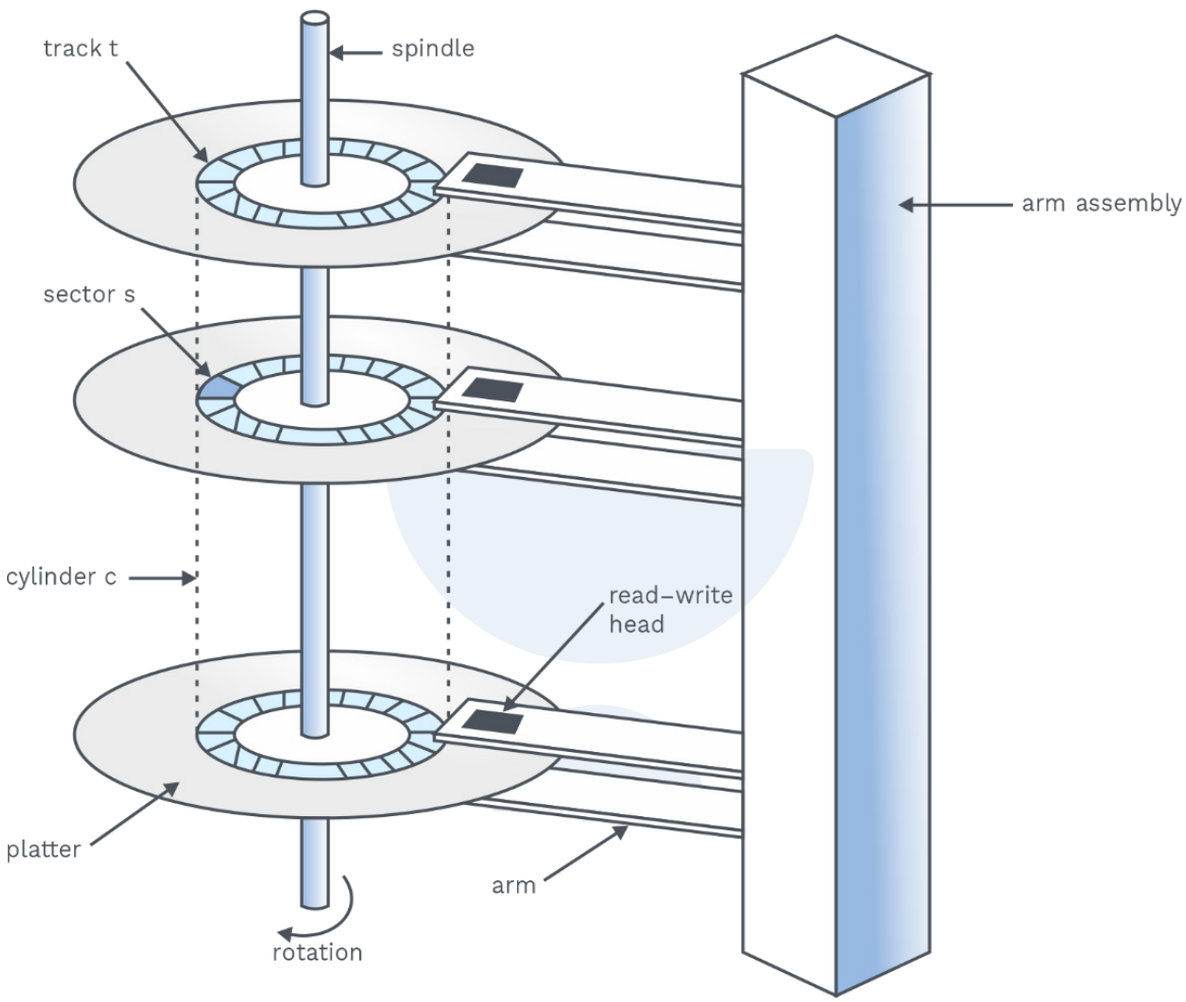


Fig. 6.2 Physical Structure of the Disk

Top view of the disk

- Top view of the disk is the top most surface.
- Surface contains tracks.
- Track contains sectors.
- Sector contains the data (in byte/word).
- Sector is the smallest unit of disk which can be read/write at once.
- Consecutive set of sectors is called cluster.

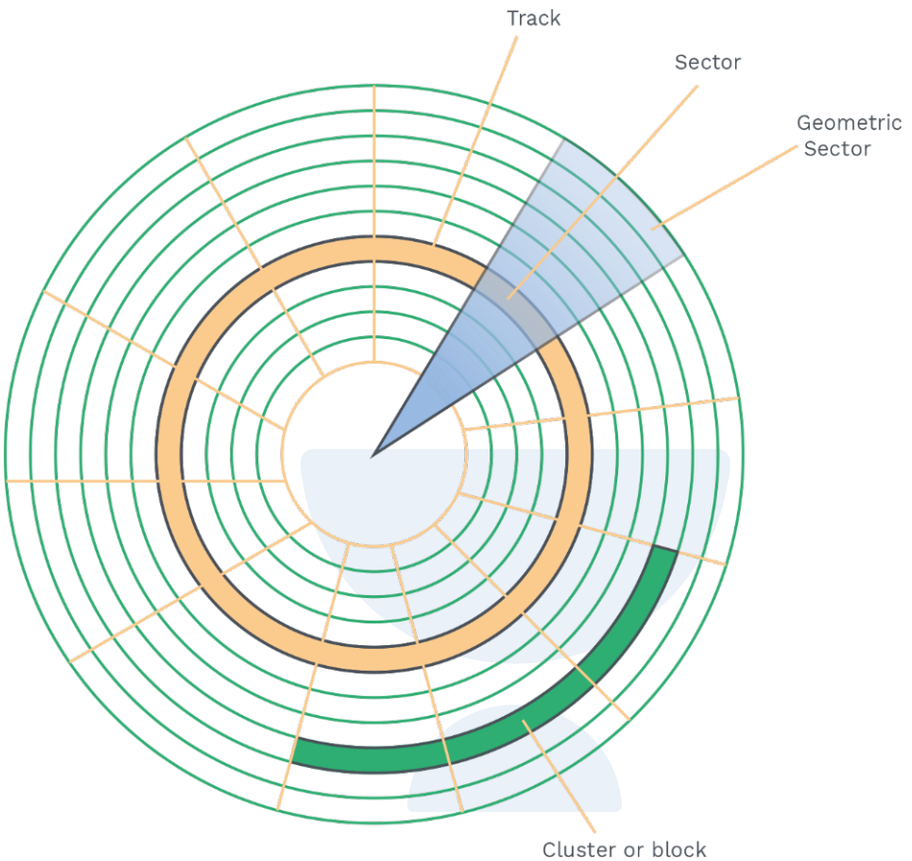


Fig. 6.3 Top View of the Disk

Magnetic disk capacity:

- If platters given instead of the surfaces

$$\text{Capacity}(C) = 2 * \left(\begin{matrix}\text{Number of}\\ \text{platter}\end{matrix}\right) * \left(\begin{matrix}\text{Number of tracks}\\ \text{per surface}\end{matrix}\right) * \left(\begin{matrix}\text{Number of}\\ \text{sectors per track}\end{matrix}\right) * \left(\begin{matrix}\text{Capacity of}\\ \text{the sector}\end{matrix}\right)$$

$$\text{Capacity}(C) = 2 * P * T * S * B$$

Where:

P = No. of Platters
T = No. of tracks per surface
S = No. of sectors per track
B = Data in bytes (capacity of sector)

Sector capacity:

There are two types of storage in a sector.

1) Sector having the constant capacity

2) Sector having the variable capacity (Not in Gate)

Sector having constant capacity:

Constant capacity means capacity of the sector is constant in each track.

The storage density varies from sector to sector.

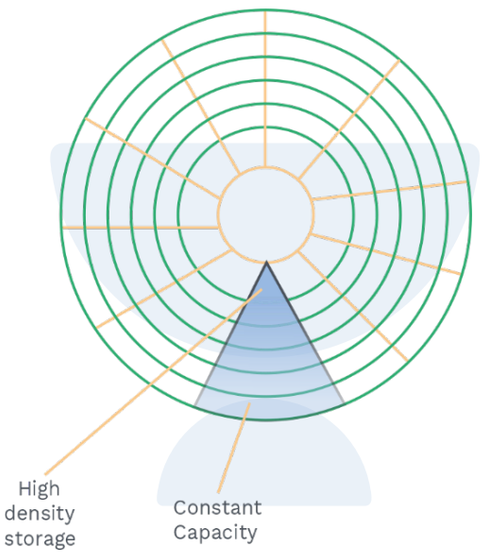


Fig. 6.4

Note:

Default constant capacity is used if each sector having constant capacity then capacity of disk is

$$\text{Capacity of disk} = \begin{pmatrix}\text{Number of}\\ \text{surfaces in a disk}\end{pmatrix} * \begin{pmatrix}\text{Number of tracks}\\ \text{per surface}\end{pmatrix} * \begin{pmatrix}\text{Capacity of}\\ \text{the sector}\end{pmatrix}$$

Disk access time:

- To calculate the access time of 1 sector of disk need to calculate:
  - **i)** Seek time
  - **ii)** Rotational delay/latency
  - **iii)** Transfer time
  - **iv)** Extra delay (if given)

- **Seek time:** Time required to move the read/write head over the desired track.
- **Rotational delay:** Time required to rotate the desired sector under read/write head. This time is taken as half rotation time on average.
- **Transfer time:** Time required to read or write one sector.

Access time:
Access time is the sum of all these delays, seek time, rotational delay, transfer time and extra time (If given in the question).

$$\text{Disk Access time of 1 sector} = \text{Seek time} + \text{Rotational delay / latency} + \text{Transfer time} + \text{Extra delay}$$

Rotational delay/latency:
Rotation latency/delay can be two types:

1) If current and target sector given:

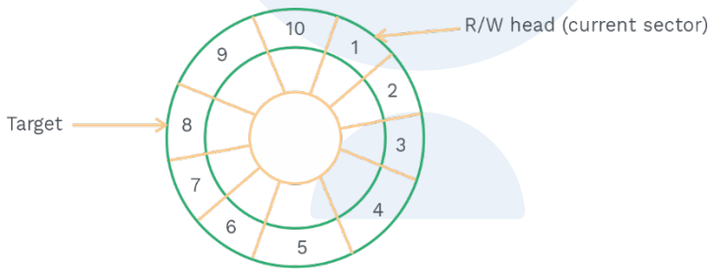


Fig. 6.5

Suppose the track contains 10 sectors, and the rotation time of the disk is 30 ms.

- To read the 8th sector data 7 sector rotation needed

$$\text{rotational delay} = 7 * \frac{30}{10} = 21\text{ms}$$

General formula:

$$\text{Rotational latency / delay} = \frac{\text{Disk rotation time}}{\text{No. of sectors per track}} * \text{No. of sectors to rotate}$$

2) If the current and target sectors have not given:
Use average rotational latency
So,

$$\text{Rotational delay / latency} = \frac{\text{Disk rotation time}}{2}$$

Note:

One track can be transferred, in one disk rotation.
Hence, time taken to transfer a sector -

$$1\text{sector transfer time} = \frac{\text{Disk rotation time}}{\text{Total no. of sectors per track}}$$

PRACTICE QUESTIONS

Q1 **Consider a disk with following parameters:**
Number of platters = 64
Number of tracks per surface = 1024
Number of sectors per track = 128
Capacity of the sector = 256 Bytes
Find the capacity of the disk and the number of bits required for addressing the disk?

Sol: **4 GB, 24 bits**

$$\text{Capacity of the disk}(C) = 2 * P * T * S * B$$

Capacity of the disk = 2 * 64 * 1024 * 128 * 256 bytes
$= 2^{1+6+10+7+8}$
$= 2^{32}$
= 4 GB

- Addressing the disk means, number of bits required to represent each sectors.
  Number of sectors = 2 * 64 * 1024 * 128
  $= 2^{24}$
  Number of bits = $\log_2 2^{24}$
  = 24 bits

Q2 **Consider a disk with 64 surfaces, 4K tracks per surface, 2K sectors per track, and the sector contains 1024 bytes of data. Seek time of the disk is 12ms and the disk rotational speed is 6000 rpm (revolution per minute). Calculate the disk access time and disk transfer rate.**

Sol: **17.005 ms, 200 MBPS**

- Number of surfaces = 64
- Number of tracks per surface = 4K
- Number of sectors per track = 2K
- Capacity of the sector = 1024 bytes
- Seek time = 12 ms
- 6000 revolution = 60 sec
- 6000 revolution = 60000 msec
- 1 revolution = $\frac{60000}{6000}$ msec = 10 msec

Access time:

$$\text{Disk Access time} = \text{Seek Time} + \text{Rotational delay} + \text{1 sector transfer time} + \text{Additional delay}$$

$$= 12 + \frac{\text{Disk rotation time}}{2} + \frac{\text{Disk rotation time}}{\text{No. of sectors per track}}$$

$$= 12 + \frac{10}{2} + \frac{10}{2K}$$

= 12 + 5 + 0.005 (K = 1000 because we are calculating in time)

= 17.005 ms

- Disk Transfer Rate

In one disk rotation, one track can be transferred.

1 Track Capacity = 2K * 1024

$= 2^{11+10}$ = 2 MB

So, we can say, 2MB of data can be sent in 1 rotation (10 ms)

10 msec — 2 MB

1 sec — ?

$$= \frac{2\,MB}{10 * 10^{-3}} = \frac{2 * 10^3}{10} MBPS\,(\text{Mega bytes per sec.})$$

= 200 MBPS

Q3 **Consider a hard disk with rotational speed of 3200 rpm. Average latency of disk is 8.4 milliseconds. A 1 MB file is stored in the hard disk. The hard disk contains 16 platters. Each surface contains 64 tracks. Each track contains 128 sectors. Each sector holds 64 B of data. What is the percentage of the disk is occupied by the file and disk data transfer rate (bandwidth), respectively?**

a) 4.75%, 13.98 MBPS **b) 6.25%, 15 MBPS**

c) 5%, 16 MBPS **d) 6.25%, 13.98 MBPS**

Sol: **d)**

Disk capacity = (No. of platters/surfaces*2)*(No. of tracks/surface)*(No. of sectors/track)*(No. of bytes/sector)

$= 16 \times 2 \times 64 \times 128 \times 64B$

$= 2^{4+1+6+7+6}\ B = 2^{24}\ B = 16\ MB$

$$\text{Percentage of disk occupied by the file} = \frac{\text{File size}}{\text{Disk capacity}}$$

$$= \frac{1MB}{16MB} * 100\%$$

$= 6.25\%$

Disk data transfer rate is calculated as

$\Rightarrow$ Number of surfaces*Surface rate

$$\Rightarrow \text{Number of surfaces*} \frac{\text{Track capacity}}{\text{Time taken by one revolution}}$$

$$\Rightarrow \frac{32 \times 128 \times 64}{\left(\frac{60}{3200}\right)} BPS \quad \left[\because 3200 \text{ revolution} \rightarrow 60 \text{ sec}, \ 1 \text{ revolution} \rightarrow \frac{60}{3200} \text{ sec}\right]$$

$$\Rightarrow \frac{32 \times 128 \times 64 \times 3200}{60} BPS$$

$\Rightarrow 13981013.33$ BPS

$\Rightarrow 13.98$ MBPS

$\therefore$ Option (d) is correct.

Previous Years' Question

Question: Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit-serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively:

a) 256 Mbyte, 19 bits
b) 256 Mbyte, 28 bits
c) 512 Mbyte, 20 bits
d) 64 Gbyte, 28 bit

Sol: a) **(GATE-2007)**

Multiple sector access time:

Sequential access:

All sectors are in the form of cluster (same track)

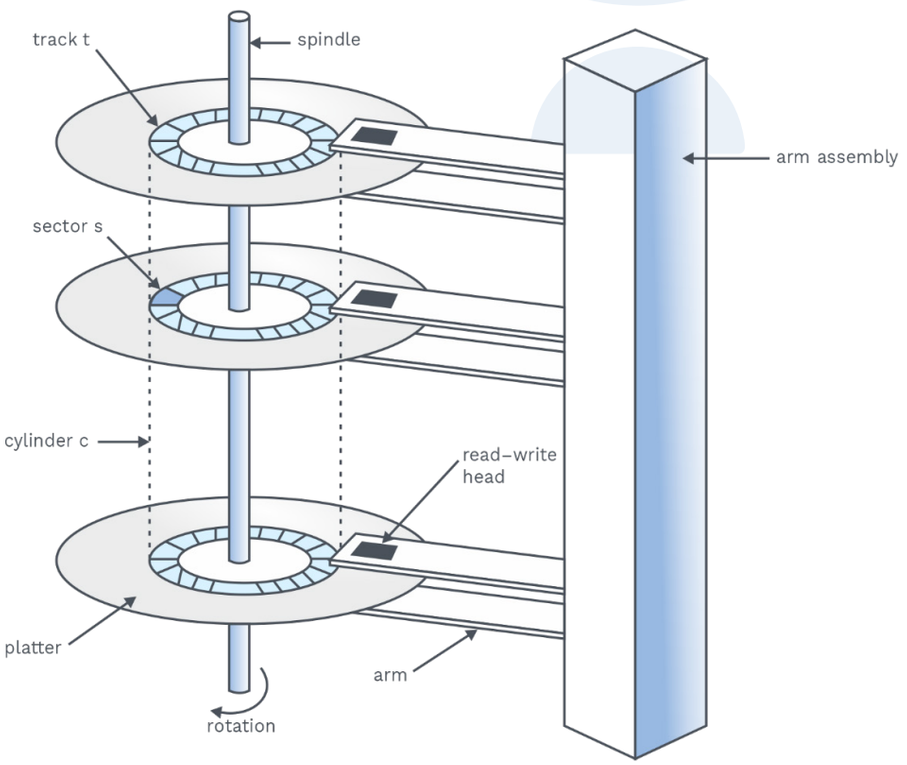


Fig. 6.6 Disk Access

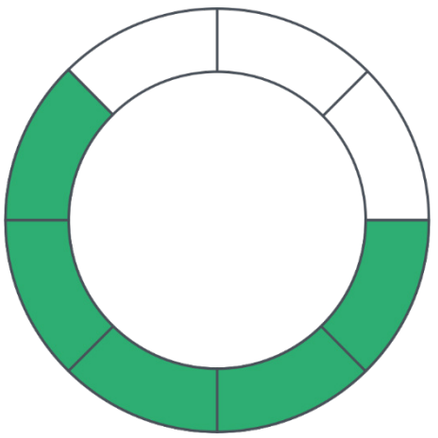


Fig. 6.7

Suppose, we need to transfer P sectors then access time is:

$$\text{Access time} = \underset{\text{Time}}{\text{Seek}} + \underset{\text{latency}}{\text{Rotational}} + P * \underset{\text{transfer time}}{\text{1 sector}}$$

Random access time:
For random access, each time needs to calculate seek time, rotational time and sector transfer time. Because directly we are accessing the desired track.

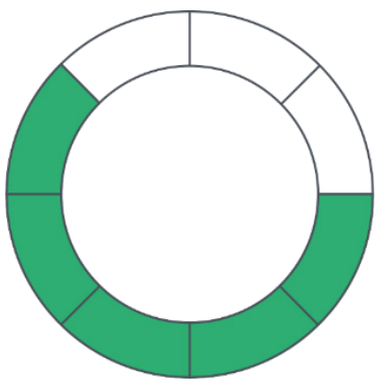


Fig. 6.8

To transfer the P sectors access time is:

$$\text{Access time} = P * \left[\text{Seek time} + \text{rotational latency} + \text{1sector transfer time}\right]$$

Previous Years' Question

Question: Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is:

a) 14020 **b)** 14000 **c)** 25030 **d)** 15000

Sol: a) **(GATE-2015)**

Cylinder:

- To save the seek time data stored and accessed in the form of a cylinder.
- Collection of all the tracks with the same radius is a cylinder.

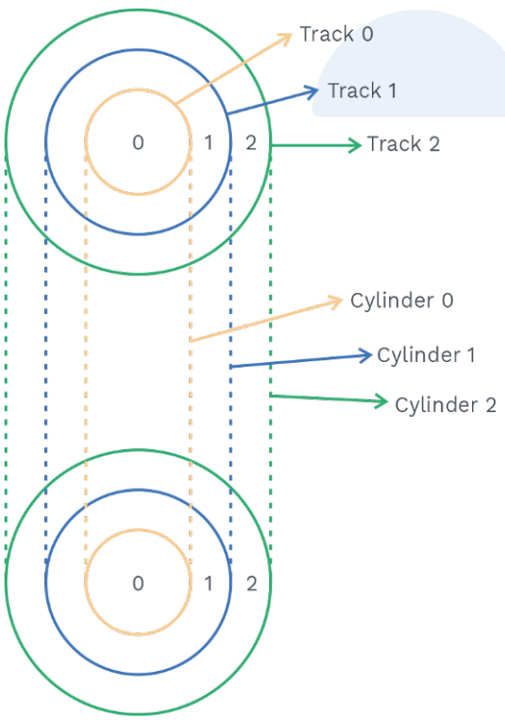


Fig. 6.9 Cylindrical View of the Disk

In the above diagram, total 3 tracks are present in the above surface.
So, we can say that

No. of tracks in the surface	=	No. of cylinders present in the disk

Disk addressing:

- Disk addressing is used to find the sequence number of the specified sector with the help of cylinders, surfaces and sectors.
- Direct sequence number of the sector is accessed with no time.

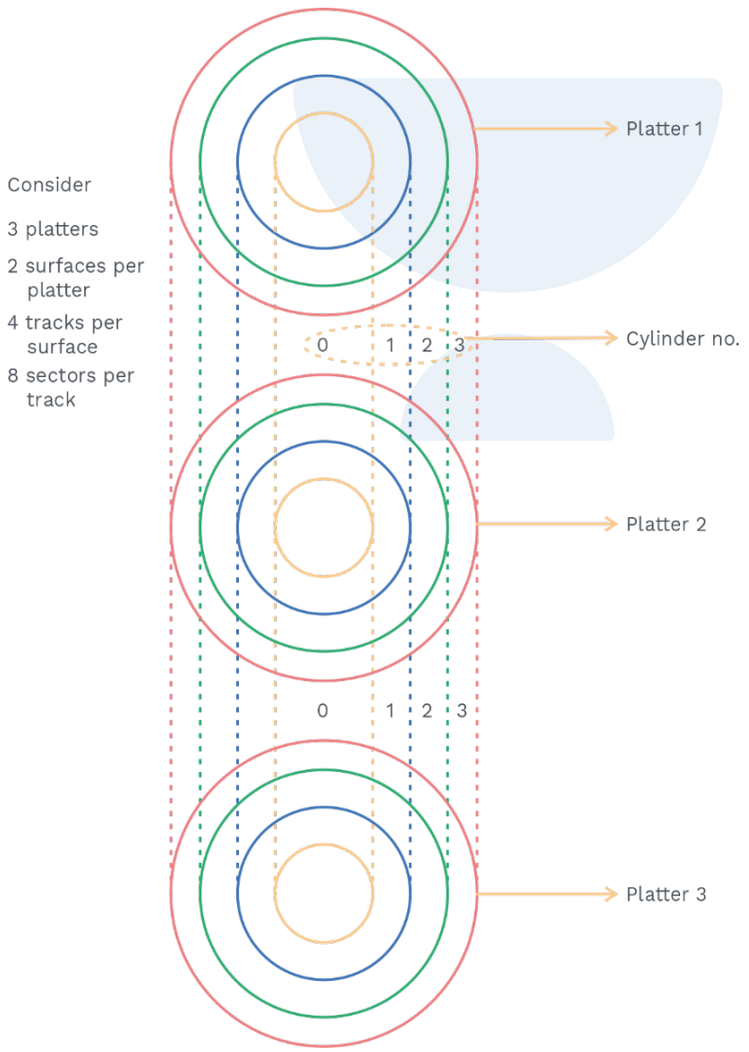


Fig. 6.10 Disk Addressing

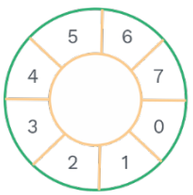


Fig. 6.11

- 4 cylinders are present in the disk because no. of tracks per surface is 4.

Address representation:
<cylinder number, surface number, sector number>
Number of cylinders = 4 (0 to 3)
Number of surfaces = 6 (0 to 5)
Number sectors = 8 (0 to 7)

0th cylinder:

		address
0th surface	<0, 0, 0>	0
	<0, 0, 1>	1
	⋮	⋮
	<0, 0, 7>	7
1st surface	<0, 1, 0>	8
	<0, 1, 1>	9
	⋮	⋮
	<0, 1, 7>	15
	⋮	⋮
5th surface	<0, 5, 0>	40
	⋮	⋮
	<0, 5, 7>	47

(8 * 5 + 7)

1st cylinder:

General formula:

To find address of the sector's sequence number

Address = < D , S , T >

Cylinder number | Surface number | Sector number

$$\text{Sector sequence number} = \left(D * \text{Number of sectors per cylinder } (N_D)\right) + \left(S * \text{Number of sectors per track } (N_P)\right) + T$$

$$\text{Sector sequence number} = (D * N_D) + (S * N_P) + T$$

PRACTICE QUESTIONS

Q4 **Consider a disk with the following parameters:**
Number of platters = 6
Number of recording surface per platter = 2
Number of tracks per surface = 32
Number of sectors per track = 32
address of the sector is given as a triple <D, S, T>, where D is cylinder number, S is the surface number and T is the sector number. 0th sector representation (address) is <0, 0, 0>, the 1st sector address is <0, 0, 1>, and so on. The address <12, 8, 20> corresponds to sector number?

Sol: **4884**

No. of cylinders in the disk = No. of tracks per surface

= 32
No. of surfaces = 6 * 2
= 12
No. of sectors per
cylinder (N_D) = 12 * 32
= 384
No. of sectors per track (N_P) = 32

< 12, 8, 20 >
↓
< D, S, T >

D = 12
S = 8
T = 20
Sequence No. = (D * N_D) + (S * N_P) + T
= (12 * 384) + (8 * 32) + 20 = 4884

Q5 **In the above question (Q3) if sequence no. of the sector is 4250. What is the address of the sector as a triple <D, S, T> format?**

Sol: **<11 0 26>**

N_D = 384
N_P = 32

$$D = \left\lfloor \frac{\text{sequence No. of the sector}}{N_D} \right\rfloor \quad D = \left\lfloor \frac{4250}{384} \right\rfloor$$

= 11
S = (sequence no. mod N_D)/N_P
= (3250 mod 384)

$$S = \left\lfloor \frac{26}{32} \right\rfloor$$

S = 0
T = (Sequence no. % N_D) % N_P
= (3250 Mod 384) Mod 32
= 26 Mod 32
T = 26
Triple format = <D S T> = <11 0 26>

Previous Years' Question

Question: A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces, and 1000 cylinders. The address of a sector is given as a triple (c, h, s), where c is the cylinder number, h is the surface number and s is the sector number. Thus, the 0^{th} sector is addressed as (0, 0, 0), the 1^{st} sector as (0, 0, 1), and so on. The address corresponds to the sector number:

a) 505035 **b)** 505036 **c)** 505037 **d)** 505038

Sol: c) **(GATE-2009 (2 Marks))**

Previous Years' Question

Question: Consider a hard disk with 16 recording surfaces (0–15) having 16384 cylinders (0–16383) and each cylinder contains 64 sectors (0–63). Data storage capacity in each sector is 512 bytes. Data are organised cylinder–wise, and the addressing format is. A file of size 42797 KB is stored in the disk, and the starting disk location of the file is. What is the cylinder number of the last sector of the file if it is stored in a contiguous manner?

a) 1281 **b)** 1282 **c)** 1283 **d)** 1284

Sol: d) **(GATE-2013 (2 Marks))**

I/O organisation:

- I/O is one of the major components to communicate with the CPU.
- Basic feature of the I/O is to exchange the data with other devices.

I/O devices (Peripheral devices):

- All devices that are connected to the CPU externally except the main memory are called as I/O devices.
- These devices are connected to the CPU through single bus arrangement.

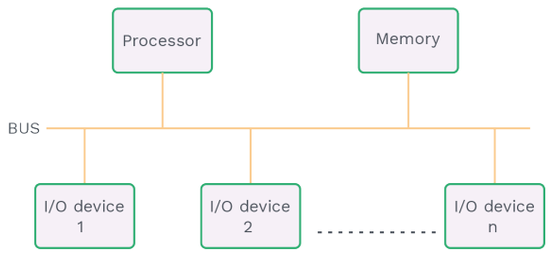


Fig. 6.12 Computer Architecture

- These I/O devices are categorised into three types:

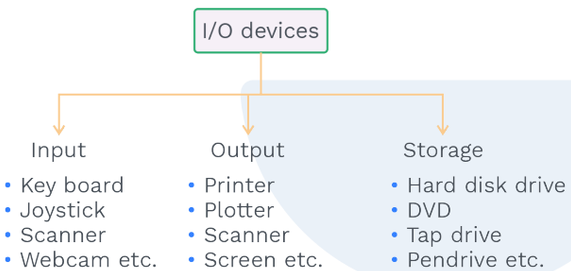


Fig. 6.13 Classification of I/O Devices

How the CPU connected to I/O device:

- CPU does not connect to the I/O device directly.
- CPU connected to the I/O device by I/O interface.
- I/O Interface is (Hardware + Software) interface. The software interface is nothing but a device driver.

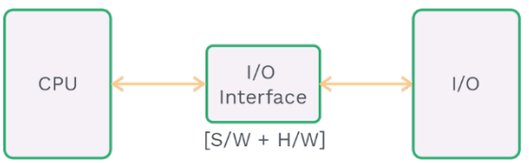


Fig. 6.14 I/O Interface

Need for interface:

- CPU is an electronic device, whereas I/O are electromagnetic device.
- For good synchronisation I/O interface is required because the CPU is fast in terms of speed, whereas I/O devices are usually slow.
- For conversion of format I/O interface is required because data codes of I/O devices are different from the word format in the CPU.
- Operations of I/O devices should not be overlapped (two or more devices can't perform an operation at the same time).
- **Versions of interface:** There are three versions:

  i) **I/O interface:** I/O interface, which provides only interfacing.

ii) **DMA controller:** DMA controller provides I/O interface and DMA facility.
iii) **I/O Processor:** I/O processor provides DMA controller (I/O interfaces also) and I/O instruction execution facility.

Memory and I/O:

There are three ways that connect CPU to memory and I/O:

1) Separate buses for both
2) Common data, address bus
3) Common address, data and control bus

1) Separate buses for both:

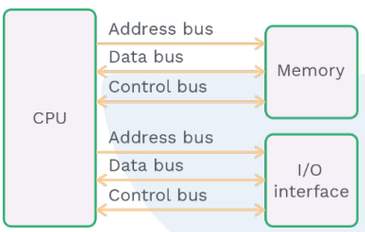


Fig. 6.15

- Separate buses for both I/O and memory.
- If the CPU wants to access memory, then CPU can access memory only, not I/O. Separate connections for both memory and I/O. So, it is costly.
- Easy to implement.
- CPU can perform operations either with I/O or with memory but not parallel.

2) Common data, address bus:

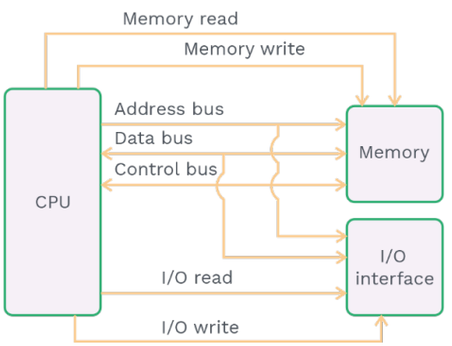


Fig. 6.16

- CPU communicates with memory and I/O by generating the address.
- If the CPU wants to access the memory, then the control signal of I/O will be disabled then the CPU can perform read/write operations in memory.

- CPU can distinguish between memory and I/O by control signals.
- CPU can perform either read operation or write operation at a time in memory or I/O interface.
- This type of communication is called isolated I/O or I/O mapped I/O, or port mapped I/O. Here address bus is common for both memory and I/O interface, but control signals are different, address can be same as we have common bus and control signals are the one to make the difference between them.

3) Common data, address and control bus:

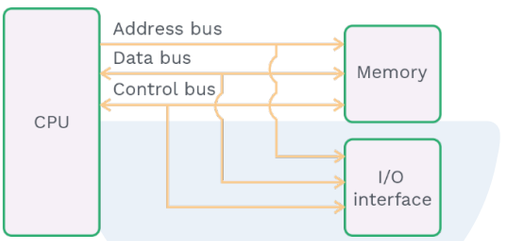


Fig. 6.17

- Here CPU uses common buses for both memory and I/O interface.
- CPU can communicate with memory by generating addresses here, some addresses of memory assigned for I/O devices.
- I/O device does not show their own addresses; they use memory addresses. This type of communication is called memory mapped I/O.
- This is difficult to implement, and this is cheap compared to I/O mapped I/O. Difference between memory mapped I/O and I/O mapped I/O.

	Memory mapped I/O	**I/O mapped I/O**
1)	I/O not having their own addresses but some addresses of memory is assigned to I/O devices	I/O mapped I/O have separate addresses for I/O
2)	Memory wastage	No memory wastage
3)	CPU distinguishes between memory and I/O using addresses	CPU distinguishes between memory and I/O using control signals.
4)	Number of addresses for memory will be less than I/O mapped as we are allocating some of the addresses to the I/O devices.	Number of instructions and addresses are more to access memory but less to access I/O.
5)	All memory access instructions and addressing modes are used for I/O also	Separate instructions and modes for I/O.

Table 6.1 Comparison between Memory Mapped and I/O Mapped I/O

Data transfer:
1 bit of data can be transferred at a time

Fig. 6.18

Parallel data transfer:

Fig. 6.19

Synchronous data transfer (serial):

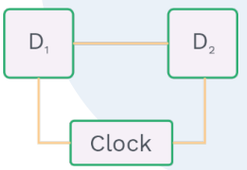


Fig. 6.20

Common clock is used for both devices to control the speed in synchronous data transfer.

Asynchronous data transfer (serial):

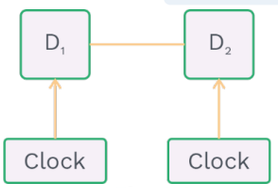


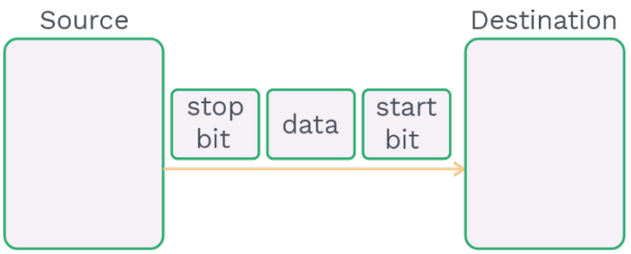
Fig. 6.21

- Here, we use separate clock for both devices.
- Here, devices have the communication but not internally (through clock).

PRACTICE QUESTIONS

Q6 **How many 8 bit characters can be transmitted per second over 14400 band serial communication links using parity synchronous mode of transmission with 2 start bits, 8 data bits, 1 stop and 1 parity bit?**

Sol: **1200 characters per second:**



To send data first bit is start bit (it can be either 0 or 1) last bit is stop bit (reverse of start bit).

- In 1 second 14400 bits can be transmit.

To send 1 character number of bits = start bits + data bits + parity bits + stop bits.

Number of bits = 2 + 8 + 1 + 1 = 12 bits

- Number of characters in one sec $= \frac{14400}{12}$

$= 1200$ characters per sec

Q7 **An asynchronous serial communication is employing 8 character bits, 2 parity bits, 2 stop bits and 2 start bits. To gain the rate of 800 characters/second what is the minimum transfer rate?**

a) 1400 bits per sec **b) 1400 bytes per sec**
c) 11200 bytes per sec **d) 6400 bits per sec**

Sol: **b)**

To send 1 character total (8 + 2 + 2 + 2) 14 bits need to transfer.

- For 1 character = 14 bits
- Number of character send in 1 sec = 800 character
- Number of bits in 1 sec. = 800 * 14 bits per sec.

$$= \frac{800}{8} \times 14 \quad = 100 * 14$$

$= 1400$ bytes per sec

Hence, option (b) is correct

Q8 **Using a parity synchronous mode of 8 bit character along with 2 stop bits, 1 parity bit and 2 start bits. What is the efficiency of the transmission line?**
a) 61.53% **b) 66.66%** **c) 62.50%** **d) None of these**

Sol: **a)**

Total bits per character = Start bits + data bits + parity bits + stop bits
= 2 + 8 + 2 + 1
= 13 bits

data bits = 8 bit

So,

$$\text{Efficiency} = \frac{8}{13} * 100 = 61.538$$

Hence, option (a) is correct.

Modes of transfer:

1) Programmed I/O or program controlled I/O
2) Interrupt driven or interrupted initiated I/O
3) Direct memory access (DMA)

Programmed I/O:

Overview:

- Whenever the CPU is executing a program related to I/O, it executes that instruction by issuing a command to a suitable I/O module.
- There is no furnishing through which I/O can tell the CPU about data transfer.
- With programmed I/O, I/O will perform the requested command and set the appropriate bits in the I/O status register.
- Further, I/O will not take any action to alert the CPU.
- I/O devices have their own status and wait.
- CPU executes the program periodically and checks the status of I/O devices related to CPU.
- If any I/O device has its status set, then only the CPU performs data transfer for it.

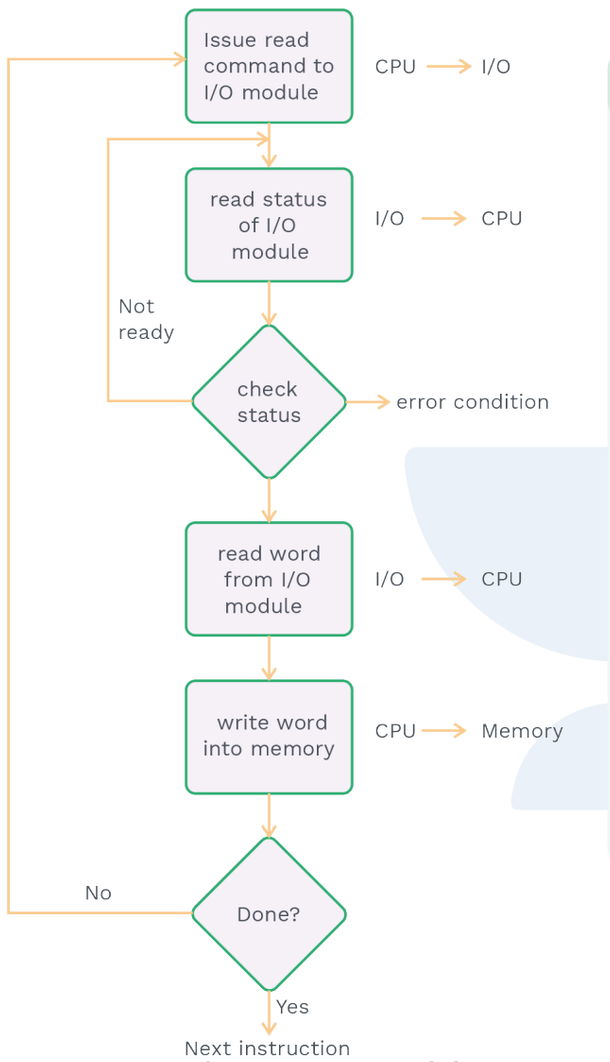


Fig. 6.22 Programmed I/O

Previous Years' Question

Question: The following are some events that occur after a device controller issues an interrupt while process L is under execution.

P) The processor pushes the process status of L onto the control stack.

Q) The processor finishes the execution of the current instruction.

R) The processor executes the interrupt service routine.

S) The processor pops the process status of L from the control stack.

T) The processor loads the new PC value based on the interrupt.

Which of the following is the correct order in which the above events occur?

a) QPTRS **c)** TRPQS

b) PTRSQ **d)** QTPRS

Sol: a) **(GATE-2018)**

Total time = Total time to check status of I / O device + Total data transfer time

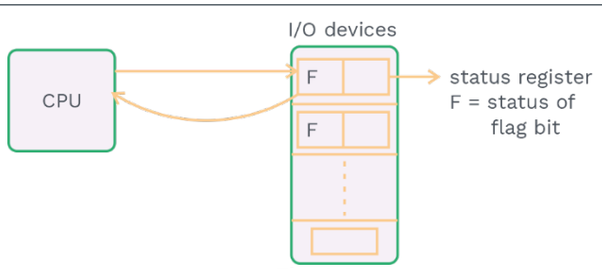


Fig. 6.23

- By default, time required by I/O device to read or share status register based on its own speed is 1 byte.

PRACTICE QUESTIONS

Q9 **Consider a device which is operating on program control mode of I/O with 100 KBPS (kilobytes per second) operating speed. The data transfer is 40 bytes from I/O. The size of status register is 4 bytes. The total time needed to perform the data transfer is _____ μs.**

Sol: **440**

Programmed I/O

- Operating speed = 100 KBPS
  100 KB −−− 1 sec
  1 B −−− ?
  1 Byte transfer time = $\frac{1B}{100KB}$
  $= \frac{1000}{100} \mu sec$
  1 Byte transfer time $= 10 \mu sec$

  Total time = Status check of I/O + data transfer time
  = 4 B status check + 40 B data transfer time
  = 4 * 10 + 40 * 10
  $= 440 \mu sec$

  Note: If status register size not given take default 1 byte
- The problem with programmed I/O is CPU wastage is more because to transfer or reception of data CPU has to wait for long (CPU check periodically I/O devices).

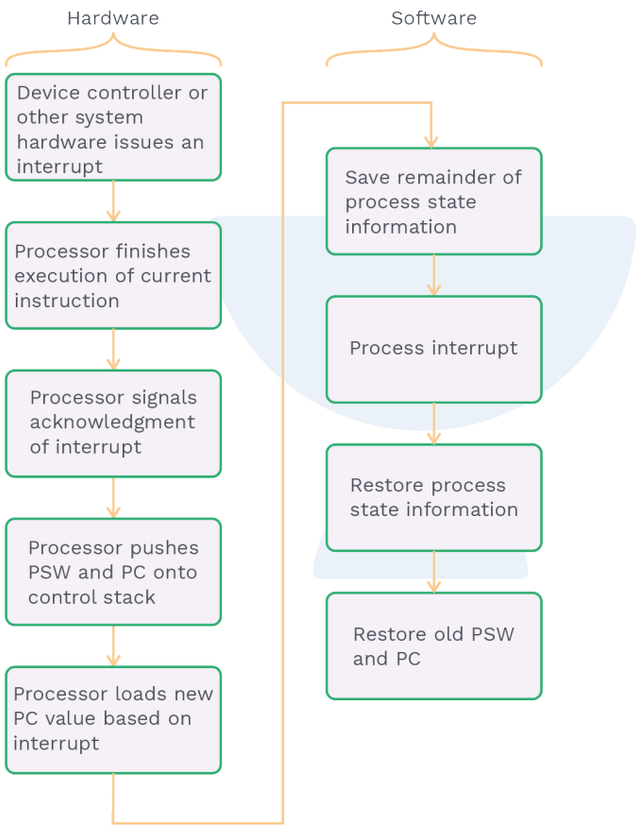
Interrupt driven I/O

Interrupt processing:

The sequence of steps that happens after an I/O device raises an interrupt are as follows:

- First CPU will complete the current instructions.
- Store the status of the current process in the stack.
- CPU will go to another process execution accordingly (stack).

- When execution is done then, resume the previous process by taking out the values from the stack.
- The occurrence of an interrupt is in CPU hardware and in software.
- When an I/O device completes an I/O operation, the following sequence of hardware and software events occurs.



Simple Interrupt Processing

Fig. 6.24 Interrupt Driven I/O

Vector address:
Vector gives the location of ISR or vector gives address or reference of ISR (interrupt service routine).

ISR:
A routine, execution of which services the interrupt.
There are two types of interrupts in computer system.

- Vectored interrupt
- Non vectored interrupt

Vector interrupt	Non vectored interrupt
Device sends interrupt signal and vector to the CPU	Device sends only interrupt signal Eg: Software Interrupt
CPU reach to ISR by using vector and execute it to provide service.	CPU first execute DSR (default service routine) to get ISR then CPU reach to ISR to service interrupt

Table 6.2 Comparison between Vector and Non Vector interrupt

Maskable interrupt:
Either the CPU can reject the interrupts, or CPU can accept the interrupts. Accepted interrupts some times kept in waiting (pending) according to priority.

Non maskable interrupt:
CPU always accepts these interrupts because these are high priority interrupt.

Internal interrupt:
CPU generates errors for itself during any instruction execution. CPU solve these errors first, then only CPU can execute the current instruction.
Example: Page fault, segmentation fault etc.

External interrupt:
If any device sends an interrupt to the CPU, these interrupts are also known as H/W interrupt.

Simultaneous interrupts:
CPU services the interrupt of the highest priority device first.

- To handle the priority, there are two solutions software solution and hardware solution.
- In a software solution, the CPU will run a program (software) first. Based on priority, the CPU resolve the interrupt.
- In hardware solution, there are two types.
- Serial solution (Daisy Chaining).
- Parallel solution.

Parallel solution not required for gate exam

Daisy chaining:

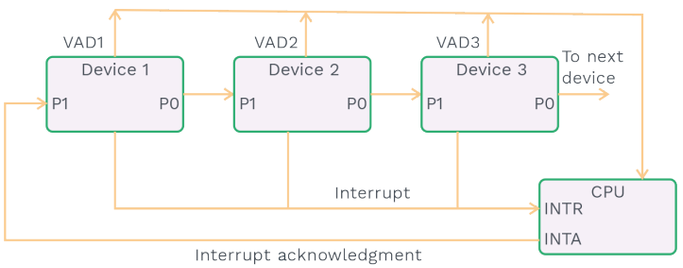


Fig. 6.25 Daisy Chaining

1) INTR: (Interrupt request):
I/O device sends the interrupt request to CPU.

2) INTA (Interrupt acknowledgement):
CPU sends the acknowledge (ack) to device 1.

3) If device 1 sent the interrupt then only device 1 accept the INTA and sends the vectored address (VAD) to the CPU by data bus.

4) If device 1 did not send the interrupt, then device 1 forward it to the next device whichever device sent the interrupt it to the CPU, that device accepted the INTA and send the vectored address to the CPU through a data bus and so on.
Here, device 1 has the highest priority.
Total time required in interrupt I/O

$$\text{Time} = \text{Interrupt overhead} + \text{Service Time}$$

Interrupt overhead:
Initial time for extra work before servicing the interrupt.

PRACTICE QUESTIONS

Q10 **Consider the processor which takes 10 cycles to service the interrupt. If the CPU runs on 20 MHz clock rate and interrupt overhead is 0.07 microseconds then total time CPU spends for interrupt service is _____ microseconds?**

Sol: **0.57**

Interrupt overhead = 0.07 μsec

Clock rate = 20 MHz

Number of cycles = 10

Total time = overhead time + Service time

Service time = 10 cycles

$$\text{Cycle time} = \frac{1}{\text{clock rate}} = \frac{1}{20}\,\mu\,\text{sec}$$

So, total time = overhead time + service time

$$= 0.07 + 10 * \frac{1}{20}$$

= 0.07 + 0.50
= 0.57 µsec

Q11 **Consider a CPU which takes 0.50 microseconds as interrupt overhead time when device generates interrupt to the CPU. An I/O device has a bandwidth of 40 KBPS.**

1) Calculate the total time in programmed I/O for 14 bytes data transfer.
2) Calculate the total time required in interrupt I/O for 14 bytes data transfer.
3) What is the estimate of performance gain of I/O transmission using interrupt driven I/O mode over programmed I/O mode?

Sol: **375 µs, 350.50 µs, 1.07**

Data transfer rate = 40 KBPS
Interrupt overhead = 0.50 µsec

1 sec = 40 KB

? = 1 B

$$\text{1 byte transfer time} = \frac{1B}{40KB}$$

= 0.025 ms
= 25 µsec

1) Programmed I/O:

Programmed I/O time = Status check time + data transfer time

Time = 1B transfer time (default) + data transfer time

= 25 + 14 * 25

= 25 + 350

= 375 μsec

2) Interrupt I/O:

Total time = interrupt overhead = service time

= 0.50 + 14 bytes transfer time

= 0.50 + 14 * 25

= 0.50 + 350

= 350.50 μsec

3) Performance gain (Speed up):

$$\text{Speed up} = \frac{\text{Older technique time}}{\text{New technique time}}$$

$$= \frac{375}{350.50}$$

= 1.07

DMA (Direct memory access):

- Enables data transfer between I/O and memory without the intervention of the CPU.
- To implement the DMA technique, a specific hardware is needed. This hardware is called DMAC (DMA Controller).

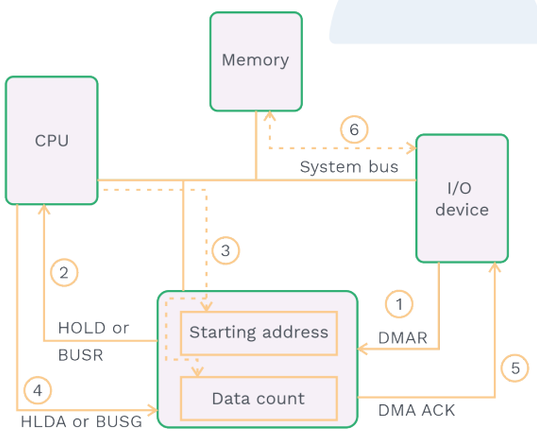


Fig. 6.26 Direct Memory Access

Steps to transfer the data between I/O and memory:

1) I/O device sends the DMAR (DMA request) to DMAC.

2) DMA controller sends Hold signal or BUSR (Bus request) to the CPU. Hold is nothing, but DMAC wants to hold the system bus control to send the data between memory and I/O.

3) If the CPU grant the HOLD, then the CPU will send the starting address and data count.
Starting address: Memory address, starting from where data is to be transferred.
Data count: Number of bytes or words to be transferred.
4) CPU sends the HLDA (Hold acknowledge) signal to the DMA controller.
5) DMA controller sends the DMA ACK signal to I/O device. We can say that DMAC is ready to transfer the data by the system bus.
6) Data is being transferred between I/O and memory through buses only. How does the starting address and data count value change?

		After 1 byte	After 2 byte	.....		After 100 byte
Starting address	2000	2001	2002		2099	2100
Data count	100	99	98		1	0

Table 6.3

- When the data count becomes 0, DMA transfer stops sending the data.
- DMA is a special purpose processor which is used to transfer the data between memory and I/O. Special purpose processor means DMAC can generate the address for memory, and it can generate control signals for both memory and I/O.

Modes of DMA transfer:
There are three modes of DMA transfer
1) Burst Mode
2) Cycle Stealing Mode
3) Interleaving Mode

1) Burst mode:
In the burst mode, when DMAC gets the system bus control, a burst of data is transferred at one time then the CPU takes back the control. So that the CPU can continue the execution, the CPU will be blocked during burst (longer data) transfer.

2) Cycle stealing mode:

In the cycle stealing mode, I/O device takes some time to prepare word to be transferred. During word preparation time. CPU keep control of the buses. When the word is ready to transfer DMAC get the control of the buses for '1 cycle' in that only the prepared word is transferred to memory. After 1 cycle of time CPU takes back the control of the buses. CPU will be blocked during the word transfer.

3) Interleaving mode:

In this mode, when the CPU is busy with internal operations, or it does not need the buses, then, only the CPU gives the control of the buses to DMAC. CPU will not be blocked at all 0% CPU blocked during DMAC.

Assume,

- I/O device takes time to prepare the data = T_P
- Data transferred time to memory = T_T

$$\% \text{ time CPU is blocked} = \frac{T_T}{T_P + T_T} * 100$$

CPU is blocked when the data is transferred because DMAC has control of the buses.

Previous Years' Question

Question: Consider a disk drive with the following specification: 16 surfaces, 512 tracks/surface, 512 sectors/track, 1 KB/sector, rotation speed 3000 rpm. The disk is operand in cycle stealing mode whereby whenever one byte word is ready it is sent to memory ; similarly, for writing, the disk interface reads a 4 byte word from the memory in each DMA cycle. Memory cycle time is 40 nsec. The maximum percentage of time that CPU gets blocked during DMA operation is:

a) 10 **b)** 25 **c)** 40 **d)** 50

Sol: b) **(GATE CS-2005)**

PRACTICE QUESTIONS

Q12 **Assume a DMA controller that supports 6-bit count register. It is interfaced with a 32-bit CPU. A file of 1 KB size needs to be transferred to main memory using cycle steading mode. How many DMA cycles are required to complete the transmission?**

a) 2 **b) 8** **c) 6** **d) 4**

Sol: **d)**

DMA supports 6-bit count register

$\Downarrow$

64 words an be transmitted to main memory in 1 DMA cycle

$\Downarrow$

64 × 4B can be transmitted to main memory in 1 DMA cycle

$$\therefore \text{Number of DMA cycles required} = \frac{1KB}{64 \times 4B} = \frac{2^{10}}{2^8} = 2^2 = 4$$

Q13 **Consider 256 KBPS I/O device interfaced with 32-bit CPU using DMA module. Data is hot transferred to main memory until 64 word data is available is buffer machine cycle time is 2 μseconds. How much time will the DMA take to transfer 128 KB file to main memory? _____ (in milliseconds)**

Sol: **1.024 ms**

Amount of data to be available in buffer = 64 word

for transmission in 1 DMA cycle

Word size of CPU = 32 bit = 2^5 bits

File size = 128 KB = 2^{7+10+3} bit = 2^{20} bits

$$= \frac{2^{20}}{2^5} \text{words}$$

$= 2^{15}$ words

$$\text{Number of DMA cycle required} = \frac{\text{File size}}{\text{Data transferred in 1DMA cycle}}$$

$= \frac{2^{15}}{2^6} = 2^9$

$\therefore$ Total time taken = Number of DMA × Machine cycle time
$= 2^9 \times 2$ microseconds
$= 2^{10}$ microseconds
= 1024 microseconds
= 1.024 milliseconds

Q14 **Consider 10 KBPS IO device interfaced to 32-bit CPU as cycle stealing mode of DMA. Whenever 16 word data is available in the buffer, it is transferred to main memory. Machine cycle time is 100 μs. What proportion of CPU time is consumed in this operation.**
a) 20% **b) 25%** **c) 80%** **d) 75%**

Sol: **a)**

Data Preparation Time (X):-

$10\,KB \xrightarrow{Prepared} 1\,sec$

$1\,W\,(4B) \xrightarrow{Prepared} \frac{4}{10\,K}\,sec$

$16\,W \xrightarrow{Prepared} \frac{4}{10\,K} \times 16\,sec$

= 6.4 ms

Data Transfer Time (Y):-

1 word data accessing from buffer to main memory happens in 1 machine cycle

1 word data $\xrightarrow{transferred}$ 100 μs

16 word data $\xrightarrow{transferred}$ 1600 μs ≅ 1.6 ms

Percentage of CPU time consumed = $\frac{Y}{X+Y} \times 100\% = \frac{1.6}{1.6+6.4} \times 100\%$

$= \frac{1.6}{8.0} \times 100\%$

= 20%

Q15 **Consider 16 KBPS IO device interfaced to 64-bit CPU using DMA interface in cycle stealing mode. DMA contains 4-bit count register. Machine-cycle is 2 ms. What are the percentages of block time and busy time of CPU respectively?**

a) 70%, 30% **b) 75%, 25%**
c) 80%, 20% **d) 90%, 10%**

Sol: **c)**

DMA contains 4 bit count register
The word data is transmitted per count
Data available in buffer before transmission = 2^4 words
= 16 words
= 16 × 8 B [1 word = 64-bit = 8 B]
= 2^7 B
= 128 B

Preparation Time (X):-
16 KB data is prepared in 1 sec

$$1B \longrightarrow \frac{1}{16K}\,sec$$

$$128\,B \longrightarrow \frac{1}{16K} \times 128\,sec \cong 8\,ms\,(milli\,sec\,onds)$$

Transfer Time (Y):-
1 word data accessing takes place in machine cycle (2 ms)
16 word data accessing will take in 16 × 2 = 32 ms

$$\therefore \text{ Percentage time for which CPU is blocked} = \frac{Y}{X+Y} \times 100\% = \frac{32}{8+32} \times 100\%$$

$$= \frac{32}{40} \times 100\%$$
$$= 80\%$$

$$\therefore \text{ Percentage of time for which CPU is busy} = \frac{X}{X+Y} \times 100\% = \frac{8}{8+32} \times 100\%$$

$$= \frac{8}{8+32} \times 100\%$$
$$= 20\%$$

Chapter Summary

Disk:

- Platter of the magnetic disks has both sides recording surfaces (storage surface).
- Getting the desired sector of the track is sequential access.
- Consecutive set of sectors is known as a cluster.
- Storage density varies from sector to sector when the sector have constant capacity.

Disk access time:

- Disk access time is nothing but access time, of 1 sector of the disk.
- To calculate the disk access time seek time, rotational latency and transfer time need to calculate.
- Rotational delay can be calculated in two ways:
  a) If current and target sectors are given.
  b) If current and target sectors are not given then we consider rotational delay as $\left(\frac{\text{Disk rotation time}}{2}\right)$.
- In one disk rotation, 1 track can be transferred.
- One sector transfer time is known as the transfer time of the disk.

Multiple sector access time:

- More than one sector can be accessed. There are two ways to calculate access time.
  a) Sequential access
  b) Random access
- In sequential access, the sectors are in the form of cluster (multiple sector with same track)
- In random access for each sector, need to calculate disk access time.

Cylinder:

- To reduce the seek time, data is stored in the form of a cylinder.
- Number of cylinders present in the disk is the same as the number of tracks on the surface.

Disk addressing:

- To get the sequence number of the sector disk addressing is used.
- Address representation of the disk is <cylinder number, surface number, sector number>. This address format is also known as the triple format of the disk.

I/O organisation

- I/O devices are connected to the CPU with a single bus.
- I/O interface is both software and hardware. Software is a device driver.
- I/O interface is required for good synchronisation, conversion, control the disturb between CPU and I/O and etc.
- There are three ways to connect the CPU to memory and I/O
  1) Separate buses for both memory and I/O.
  2) Common data bus and address bus.
  3) Common data bus, address bus and control bus.

Memory mapped I/O:

- To access the memory and I/O number of instructions and addresses are more.
- Memory wastage is more in memory mapped I/O.

I/O mapped I/O:

- To access memory number of instructions and addresses are more but less to access I/O.
- No wastage of memory.

DMA (Direct memory access):

- Without the interference from the CPU, data can be transferred between I/O and memory with the help of DMA.
- DMA generates an address for memory and also generates control signals for both memory and I/O.
  There are three modes of transfer, including DMA transfer.
  1) In the burst mode, CPU will be blocked during burst or block or longer data transfer.
  2) In cycle stealing mode, CPU will be blocked during word transfer.
  3) In interleaving mode, CPU will not be blocked (0% block).