

Computer Architecture and Organization : CT 2

⌚ Last edited time	@December 16, 2023 10:17 PM
⌚ Created	@December 14, 2023 7:46 PM
👤 Created by	👤 Borhan
🏷 Tags	

Syllabus: Chapter 2, 4 and chapter 7 micro operation পর্যন্ত।

Resources:

- Chapter 4, Cache Memory: https://www.youtube.com/watch?v=OfqzoQ9Kw9k&list=PL3R9-um41JswfYyN5cxhfAmlD_JTaYYwE&index=1
 - Memory Access Method : https://www.youtube.com/watch?v=4FB7qdXC_z8
 - Memory Hierarchy: <https://www.youtube.com/watch?v=zwovvWfkuSg>
- Register Transfer and Microoperation (1-4) : https://www.youtube.com/watch?v=mNNywW00eEc&list=PLoAiV2nk2_iEK1YJf_Dx2t-UuBpnK_Byh
- Memory Hierarchy - Cache Memory : <https://web.njit.edu/~sohna/cs650/lec7.pdf>

Ch.2 : Performance Measuring

- **Clock Cycle Time**
 - Time between two consecutive rising or trailing edges of a periodic clock signal
 - it allows counting unit computation

$$\text{CPU Time} = CC \times CT$$

$$f = \frac{1}{CT}$$

CC = Cycle Count

CT = Cycle Time

f = Clock frequency/rate

- **CPI (Clock Cycle per Instruction)**

- an alternative performance measure, because it is easier

$$CPI = \frac{\text{CPU clock cycle for the program}}{\text{Instruction count}}$$

$$CPU \text{ time} = \text{Instruction count} \times CPI \times CT = \text{Instruction count} \times CPI \times \frac{1}{f}$$

If CPI for each instruction category is known, then overall CPI can be computer as

$$CPI = \frac{\sum_{i=1}^n CPI_i \times I_i}{\text{Instruction Count}}$$

I_i = the number of times and instruction of type i is executed

CPI_i = average number of clock cycles

- **MIPS (Million Instruction-per-second)** : The rate of instruction execution per unit time
 - a performance measure for machines
 - It measures the rate of average instructions
 - It does not track execution time

$$MIPS = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Clock rate}}{CPI \times 10^6}$$

$$xMHz = x \times 10^6 \text{ Hz}$$

- **MFLOP (Million floating-point instructions per second):** The rate of floating point instruction execution per unit time
 - measure of machine's performance
 - is defined the subset of floating-point instructions

$$MFLOPS = \frac{\text{Number of floating-point operations in a program}}{\text{Execution time} \times 10^6}$$

- **Summarize performance regarding larger sets of programs**

- **Arithmetic Mean**

$$= \frac{1}{n} \sum_{i=1}^n \text{Execution time}_i$$

- **Geometric mean**

$$= \sqrt[n]{\prod_{i=1}^n \text{Execution time}_i}$$

- **Amdahl's Law**

- Is a formula used to find **maximum improvement possible by improving particular part of a system**
- In parallel computing it is mainly used to **predict the theoretical maximum speed up** for program processing using multiple processor
- related to speed up of parallel computer
- when a program run on parallel computer then the computation may be
 - serial
 - parallel
 - both
- there will be certain part of program
 - consider sequential fraction of program = F
 - parallel computation of program = $1 - F$
- Amdahl's formula, speed up, $S(P) = \frac{1}{F + \frac{1-F}{P}}$
 - Speedup is the ratio of the time it takes to execute a program in serial(one processor) to the time it takes to execute in parallel (many processors.)
- **Proof:**

It based on a **fixed problem size**.

Amdahl's law tells that a given problem size, **the speedup doesn't increase linearly** as the number of processor increases. Speed up tends to become saturated.

P = no. of processors

$$\text{Speed up formula, } S(p) = \frac{T(S)}{T(P)} = \frac{T(1)}{T(P)} \dots (1)$$

Execution time for single processor = T

Execution time for parallel processor =

Sequential Computing + Parallel Computing

$$= F.T + \frac{1-F}{P}T$$

From, equation (1),

$$S(P) = \frac{T(s)}{T(P)} = \frac{T}{F.T + \frac{1-F}{P}T} = \frac{1}{F + \frac{1-F}{P}}$$

$$S(P) \leq 1$$

$$\text{or, } S(p) = \frac{1}{F + \frac{1-F}{P}}$$

Ch. 4 : Cache Memory

- small size fast memory
- placed between main memory & CPU
- high speed volatile memory
- contains most frequently accessed instruction data
- Located inside the CPU chip (internal cache L1 and L2) or motherboard (external cache, L3)

CPU → L1 → L2 → L3 → Main Memory

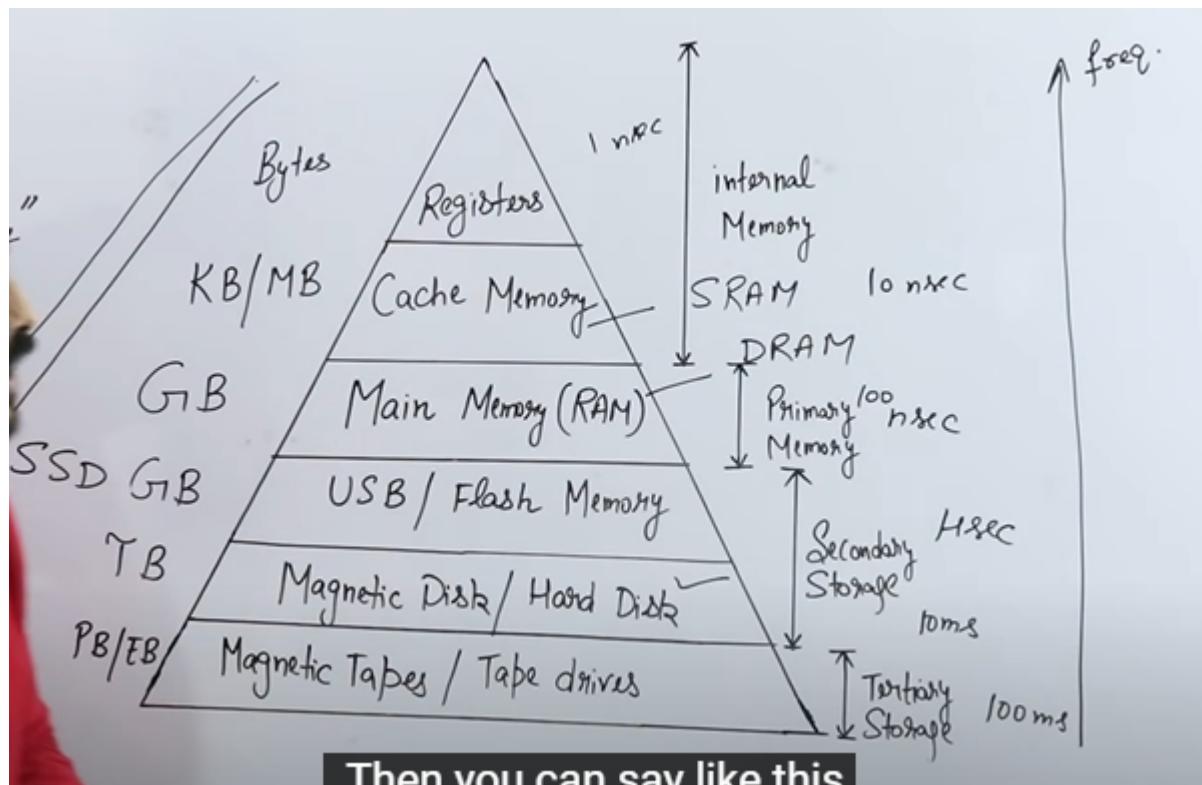
Memory Access method

- Sequential
 - whenever any access request arrives, the memory is searched from the beginning till the required data is found
 - simple but slow
 - time is variable
- Random access
 - Data present at any memory location can be accessed directly
 - Time is constant
 - Fastest
 - Example : RAM
- Direct Access
 - It is a semi-random mode of operation in which data is stored in blocks/tracks which can be accessed randomly. Then find the memory location from the

block/track sequentially.

- Time is variable

Memory Hierarchy



Access time, size both are increasing from top to bottom

Cost and frequency both are increasing from bottom to top

Working of Cache

- The CPU initially looks in the cache memory for the data it needs
- If the data there, it will retrieve it and process it
- If the data is not there, then the CPU access the system memory and then puts a copy of the new data in Cache memory
- Next time, if the CPU needs to access the same data again it will retrieve the data from cache instead of main memory

Data transfer between cache and main memory → in blocks

Data transfer between CPU and cache memory → in words

Cache Hit: If the required data is found in cache

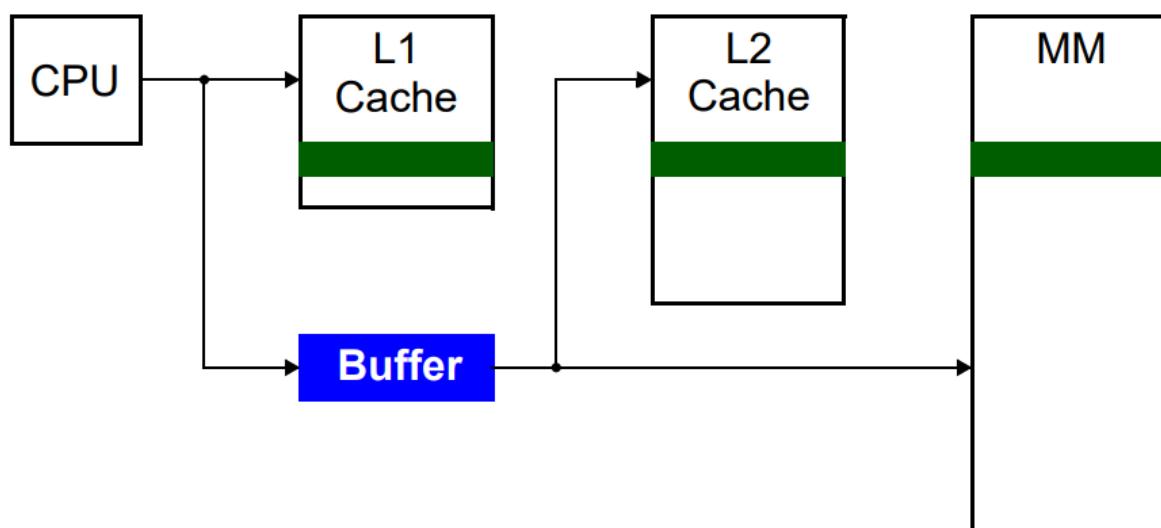
Cache Miss: If the required data is not found in cache

Cache access time (Cache Hit time): Time required to access word from the cache

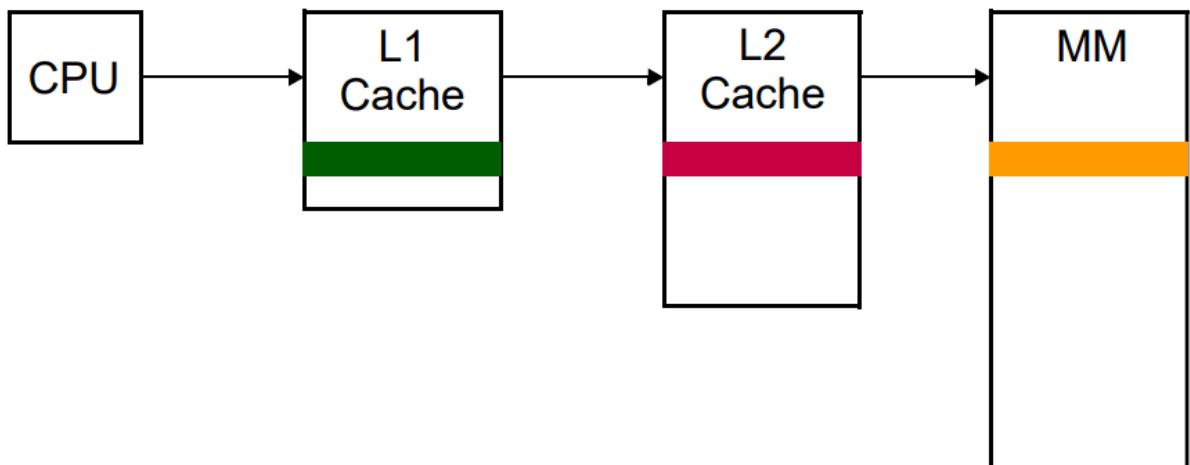
Miss Penalty (Cache Miss Time Penalty) : The time required to fetch the required block from main memory

$$\text{Miss Penalty} = \text{Cache} + \text{Main Memory Access Time}$$

Write through procedure: Changing something in Cache memory, immediately changes in main memory too. **Information is written to both the block in the cache and to the block in the MM.**



Write back procedure: Changing something in Cache memory, doesn't immediately change in main memory but when it removes from cache. Information is written only to the block in the cache. The modified cache block is written to MM only when it is replaced. This requires an additional information (either hardware or software), called dirty bits.



Cache Performance

- it is measured by in terms of **Hit Ratios**
- Hit Ratio = $\frac{\text{Hits}}{\text{Total}} = \frac{\text{No. of Hits}}{\text{Total Number of CPU references}}$
- Miss Ratio = $\frac{\text{Miss}}{\text{Total}} = \frac{\text{No of miss}}{\text{Total no of CPU references}}$
- Average access time of CPU = Hit Ratio \times Cache Access Time +
 $(1 - \text{Hit Ratio}) \times \text{Miss Penalty}$
 $= h \times T_c + (1 - h) \times T_m$

Miss Penalty = Cache Access Time + Main Memory Access Time

Example:

The access time of a cache memory is 100ns and that of main memory 1000ns . It is estimated that 80 percent of the memory request are for read and remaining 20 percent for write. The hit ratios for read access is only is 0.9. A write through procedure is used.

- (a) What is the average access time of the system considering only memory read cycle and write cycle separately ?

Ans:

$$\text{Average read access time} = h \times T_c + (1 - h) \times T_m = 200\text{ns}$$

$$\begin{aligned} \text{Total write access time} &= 1 \times \\ &\max(\text{Cache memory access time}, \text{main memory access time}) \\ &= (100, 1000) = 1000\text{ns} \end{aligned}$$

(b) What is the average access time for both read and write cycle?

Ans:

$$\begin{aligned}\text{Average access time for both} &= 80\% \text{ for read} + 20\% \text{ for write} \\ &= 0.8 \times 200 + 0.2 \times 1000 \\ &= 360\text{ns}\end{aligned}$$

(c) What is hit ratio taking into consideration the write cycle?

Ans:

$$\text{Hit Ratio taking} = 0.8 \times 0.9 + 0.2 \times 0 = 0.72$$

[Hit Ratio for only write is assumed as 0]

Locality of Reference

Also known as **Principle of Locality**.

- Locality of reference is the tendency of processor to **access the set of memory locations repetitively** over a short period of time.

Two types of locality of reference:

- **Temporal Locality**
 - if a program access one memory address, there is a good chance that it will access the same address again
 - It refers to reuse of specific data/resource within a short period of time
 - **Example:** Loop
- **Spatial Locality (Data locality)**
 - if a program access one memory address, there is a good chance that it will access other nearby addresses
 - It refers to use of data elements within relatively class storage location
 - **Example:** Nearly “every program” exhibits spatial locality

Level of Cache Memory

Level 1 (L1) Cache:

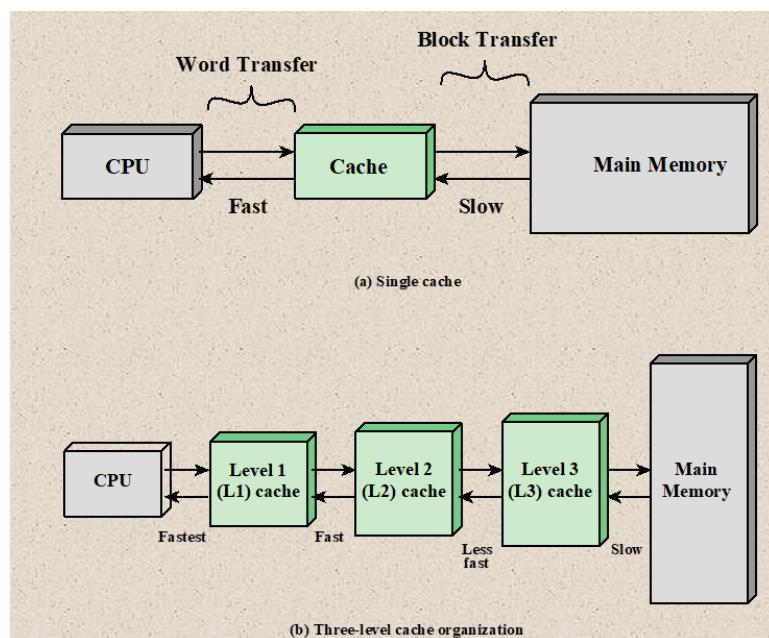
- fastest cache because it is within the processor
- size **8KB to 64KB**, that's why instruction and data are stored together
- uses the high-speed SRAM(Static RAM)
- It is also called **Internal cache or Primary cache**.

Level 2(L2) Cache:

- Larger but slower than L1 Cache
- Stores recently accessed information
- Also known as **Secondary Cache**, it is designed to reduce the time needed to access data in case where data has already been accessed previously
- It comes between L1 and RAM.
- Size ranges from **64KB to 4MB**

Level 3 (L3) Cache:

- It is built in the **motherboard**
- It is used to feed/support L2 cache
- It is faster than system memory
- size ≥ 3 MB



Cache Mapping

- Cache Mapping is a technique by which content of main memory is brought into the cache memory
- Main memory divides in blocks and cache memory divides in lines

Connections:

CPU → Cache → Main Memory → Secondary Memory (Hard Disk)

- Paging is a technique by which content of secondary memory is brought into main memory

3 Types:

- **Associative Mapping**
 - Fastest, most flexible, uses Associative Memory
 - In associative mapping cache are made up of Associative memory. Associative memory is used to store both the address and content/data of the memory word.
 - It permits any location in cache to store any word from main memory. It enables any word from main memory at any place in the cache memory.
 - If associative cache memory doesn't have any space/empty location, then it uses Replacement policy/algorithm to which one will be replaced
 - FIFO
 - LRU (Least Recently Used)
 - LFU (Least Frequently Used)
 - Random

Ex 1: MM Size: 4 GB
 Cache Size: 1 MB
 Block Size: 4 KB
~~Cache : 1 MB~~

Sol. MM Size = 4 GB = $2^2 \times 2^{30}$ B = $2^{(2+30)}$ B = 2^{32} B
 \therefore No. of P.A. bits = $\log_2 2^{32} = 32$

$$\text{Block Size} = 4 \text{ KB} = 2^2 \times 2^{10} \text{ B} = 2^{12} \text{ B}$$

$$\text{No. of Blocks in MM} = 2^{32} / 2^{12} = 2^{20}$$

$$\therefore \text{No. of Tag bits} = \log_2 2^{20} = 20$$

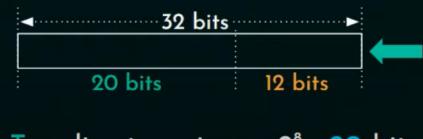
$$\text{Cache Size} = 1 \text{ MB} = 1 \times 2^{20} \text{ B} = 2^{20} \text{ B}$$

$$\text{No. of Lines in Cache} = 2^{20} / 2^{12} = 2^8$$

1. P.A. bits' split?

2. Tag directory size?

$$\therefore \text{Block offset} = \log_2 2^{12} = 12$$

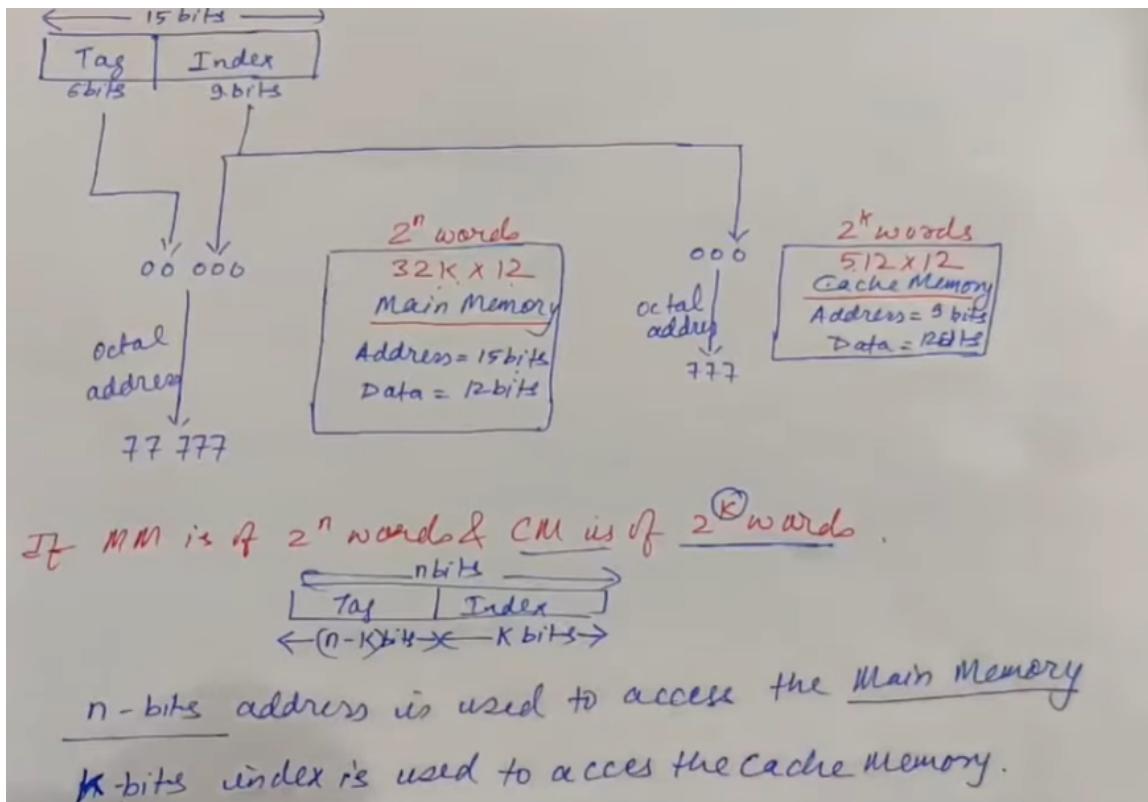


$$\therefore \text{Tag directory size} = 2^8 \times 20 \text{ bits}$$

$$= 5120 \text{ bits}$$

- Direct Mapping

- Associative memory are expensive compared to RAM. (because of added matching logic is attached)
- For RAM, Direct Mapping is used
- Simplest technique
 - it maps each block of main memory into only one possible cache line
 - CPU address space is divided into 2 parts
 - Tag : Most Significant Bits
 - Index : Least Significant Bits



- **Direct Mapping drawbacks**

- Two words with the same index in their address with different tag values cannot reside in cache memory at the same time

Questions:

- A digital computer has memory unit of $64K \times 16$ and a cache memory of $1K$ words. The cache uses direct mapping with a block size of four words.
 - How many bits are there in the tag, index, block (line) and word (block offset) field of address formula ?

Ans:

Main memory is represented as :

$64K \times 16$

= $64K$ words of 16 bits each

Main memory word size/ data = 16 bits

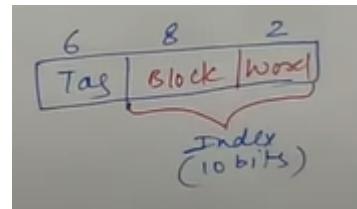
Main memory size = $64K = 2^6 \times 2^{10} = 2^{16}$ words

Address bits = 16

Cache memory size = $1K$ words = 2^{10} words

Index = 10 bits

$$\text{Tag} = (16 - 10) = 6 \text{ bits}$$



$$\text{Number of block} = \frac{2^{10}}{4} = 256 = 2^8 \text{ blocks}$$

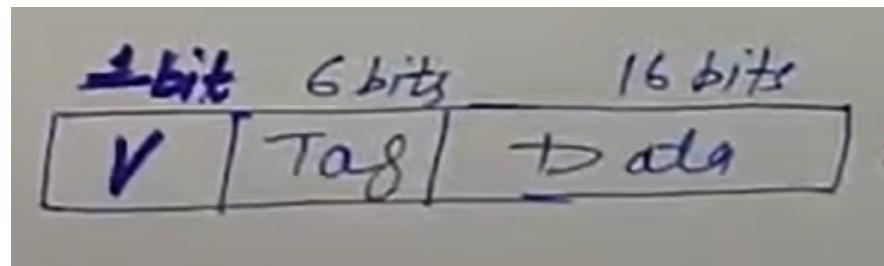
$$\text{Number of word in 1 block} = 4 \text{ words} = 2^2 \text{ words}$$

$$\text{Word} = 2 \text{ bits}$$

b) How many bits are there in each word of cache and how are they divided into functions ? Include a valid bit.

[Valid bit is one bit only (0 or 1)]

Ans: 23



c) How many blocks can accommodate ?

$$\text{Ans: } \frac{1K}{4} = 256 \text{ blocks}$$

- **Set-Associative Mapping**

- combines of Direct and Associative Mapping
- Each word of cache can store two or more words of memory under the same index address, creating a set
- Each data word is stand together with it's tag.

Question

A two way set-associative cache memory uses block of four words. The cache memory can accommodate a total of 2048 words from main memory. The main memory size is $128K \times 32$.

a) Formulate all permitted information to construct the cache memory. (Tag, index, data, blocks (set no.), words)

Ans:

Main is represented as = $128K \times 32$ words =
 $128K$ words of 32 bits

Word or data size of main memory = 32 bits

The main memory size = $128K$ words = $2^7 \cdot 2^{10} = 2^{17}$ words

Memory address = 17 bits

The cache is 2-way set associative = $\frac{2048}{2} = 1024 = 2^{10}$ words

Index = 10 bits

Tag = (17-10) = 7 bits

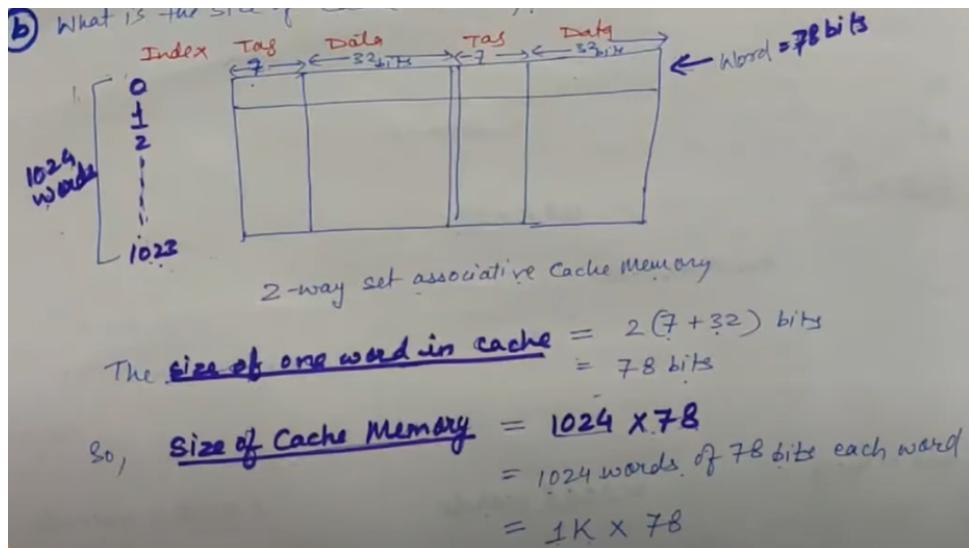
Total number of blocks = $\frac{1024}{4} = 128 = 2^8$ blocks

Block = 8 bits

Number of words in a block = 4 = 2^2

Word = 2 bits

b) What is the total size of Cache Memory ?



Virtual Memory

- It appears to be present but actually not
- It provides an illusion of a large memory
- Virtual Memory is a technique allows users to use more memory for a program than the real memory of a computer
- Virtual Memory is the concept that gives an user to the user that they will have main memory equal to the capacity of secondary storage.
- It is a temporary memory is used along with the RAM or main memory.
- VM is a Memory Management Capability of an operating system that uses hardware and software to allow a computer compensate for physical memory shortage by temporary memory data from RAM to disk Storage.
- The basic idea of VM is to keep only those parts of the program currently in use in the memory and the rest on disk drive.

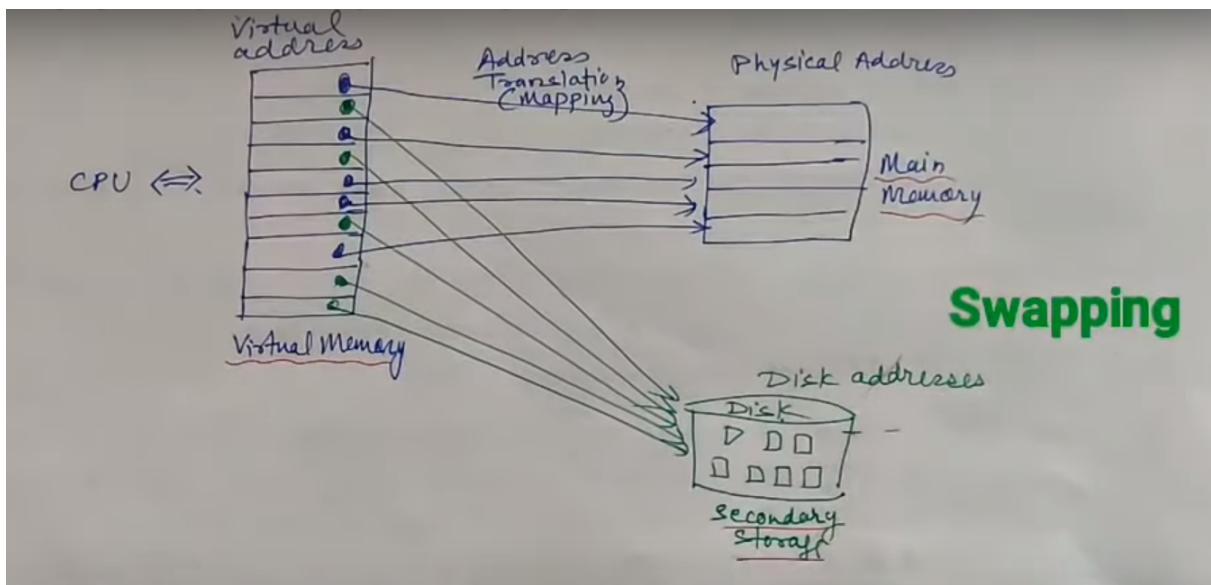
How the VM is implemented

- To implement VM, a portion of Hard disk (HDD) is reserved by the system. This portion can be either a file or a separate partition.

In Windows it is a file called pasefile.sys

In Linux, it is a separate partition.

- When the system needs more memory or RAM, then it transfer some of its data from RAM to hard disk. (Swapping between the main memory and hard disk)
- The size of virtual memory is limited by
 - the addressing scheme of computer system
 - the amount of the secondary memory available

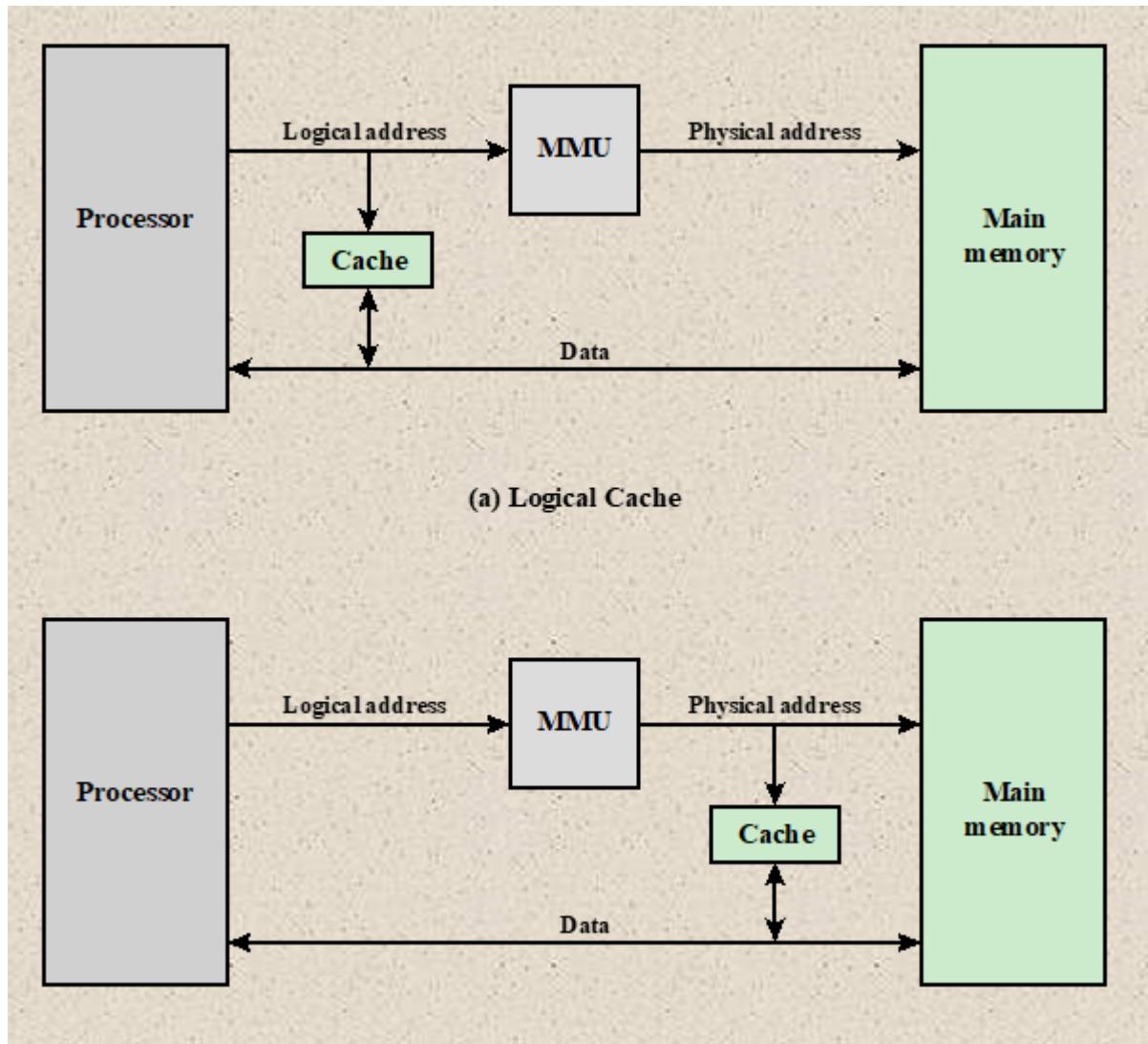


In virtual memory, there are the address of main memory and secondary memory. When a secondary memory address gets, it transfer it's to HDD to Main memory (which is known as Swap-in). Besides, it transfer those paused main memory task to HDD which is called Swap-out.

Virtual Address/Logical address: Virtual address is the address used by the programmers. The set of such addresses is called address space or virtual memory.

Physical Address: An address of main memory is called Physical Address. The set of such locations in main memory is called the Memory Space or Physical Memory. Physical address is stored in Memory Address Register (MAR).

- Virtual memory system provides a mechanism for translating program-generated address into main memory locations **dynamically**. The address translation or wrapping is handled automatically by the hardware by means of a “Mapping Table”.



Cache replacement policies

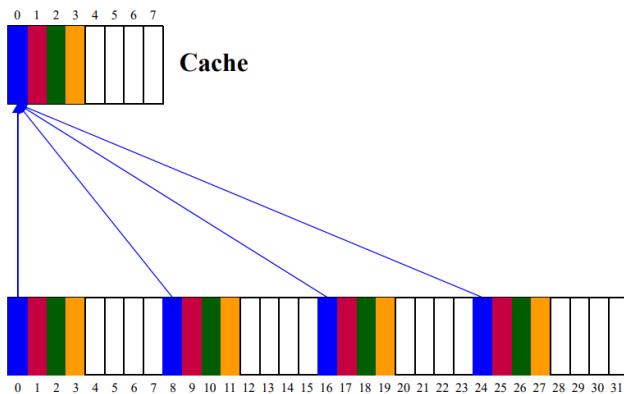
Cache replacement policies (also known as **cache replacement algorithms** or **cache algorithms**) are optimizing instructions or algorithms which a computer program or hardware-maintained structure can utilize to manage a cache of information

- FIFO : Like a FIFO queue
- LIFO : stack
- Optimal Replacement: won't be referred for the longest period of time in future, theoretical, cannot be implemented
- MRU : most recently used block, works with cyclic pattern

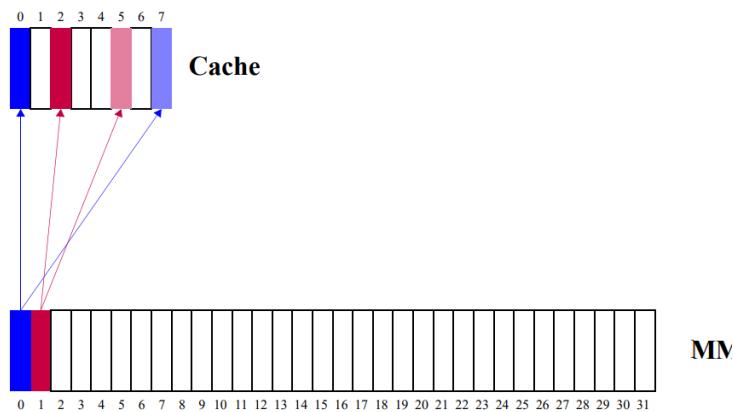
- LRU: temporal locality, least recently used block
- LFU : least frequently used, frequently is recorded, in case of tie fifo used

Placement Policy:

- Direct Mapping
 - each block of MM has only one place it can appear in the cache
 - Cache address = (MM block address)% (block in cache)



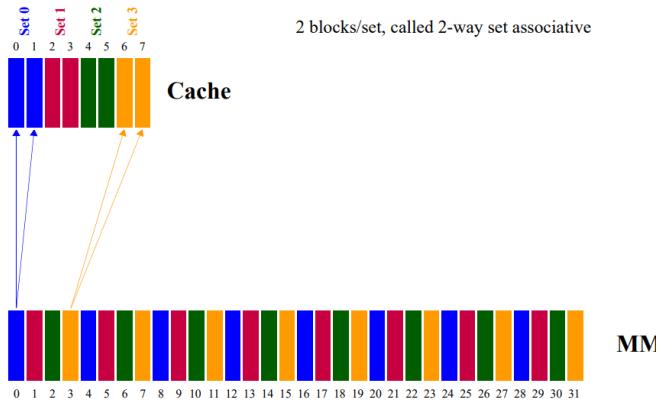
- **Associative Mapping**
 - A block of MM can be anywhere in the cache



- **Set Associative Mapping**
 - A block of MM can be placed anywhere within the set

- Cache address = (MM block address)%(block in cache)

Set Associative Mapping (k-way)



Identification Policy

- **Direct Mapping**
 - 1 Comparison
 - Cache address = (MM block address)%(block in cache)
- **Associative Comparison**
 - MM block can appear anywhere in the cache
 - need to compare all the tags
- **Set Associative Mapping**
 - K comparison
 - Cache address = (MM block address)%(block in cache), have to check a particular set that has K blocks

Ch.7 : Register Transfer and Microoperations

Microoperations

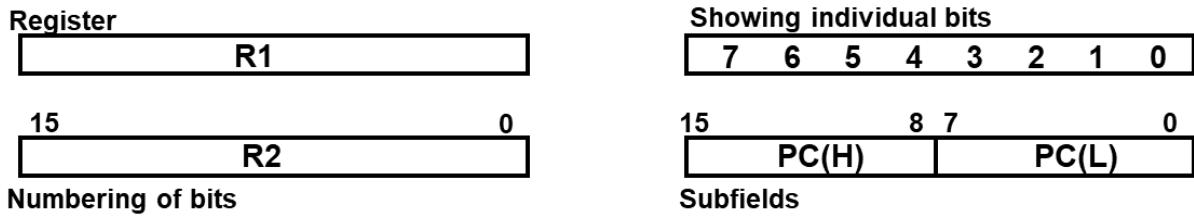
The operations on the data in registers are called microoperations.

A microoperation is an elementary operation performed on information stored in one or more register. Result of operation may replace the previous binary information of a register or may be transferred to another register.

Example : $a++$, (replaced the previous register) $a = b+c$; (transferred to another register)

Register Transfer Language

The symbolic notation used to describe the microoperation transfers among register is called Register Transfer Language.



Information transfer from one register to another is designated in symbolic form by meant of replaced operations,

$R1 \leftarrow R2$, Information transferring from $R2$ to $R1$

Under a predefined condition/Control function

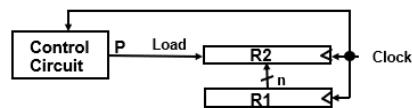
If $P = 1$ then $R1 \leftarrow R2$

Symbol : $P : R1 \leftarrow R2$

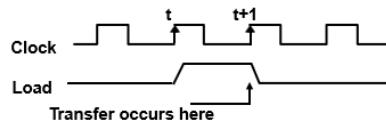
Implementation of controlled transfer

$P: R2 \leftarrow R1$

Block diagram



Timing diagram



- Letters, with numerical or not → Denotes a register → MAR, R2
- Parentheses → Denotes a part of register → R2(0-7), R(L)

- Arrow → Denotes the transfer of info → $R2 \leftarrow R1$
- Comma → Separates two microoperations, **two or more operations are to occur simultaneously**
- Colon: denotes termination of control function → $P :$

Microoperations

- **Register Transfer**

Copying the contents of one register to another is a register transfer.

- not destructive
- takes one clock pulse

It implies that

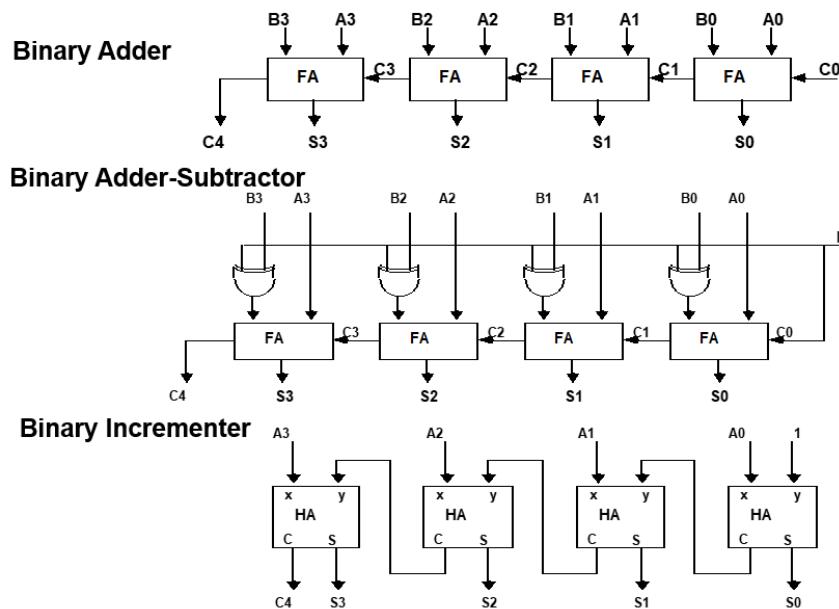
- the data lines from source register to destination register
- parallel load in the destination register
- control lines to perform action

- **Arithmetic Microoperations**

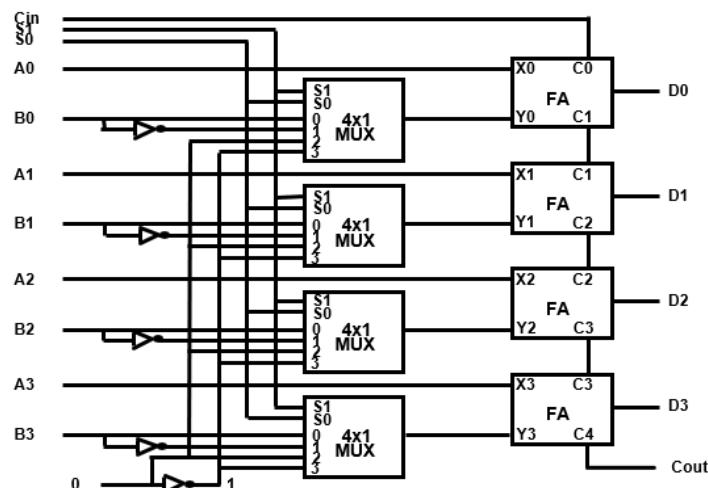
- Addition → $C \leftarrow A + B$
- Subtraction → $C \leftarrow A - B$
- 1's compliment → $C \leftarrow \bar{A}$
- 2's compliment → $C \leftarrow \bar{B} + 1$
- Subtraction : $C \leftarrow A + \bar{B} + 1$
- Increment → $A \leftarrow A + 1$
- Decrement → $C \leftarrow C - 1$

Multiplication and division are not basic micro operation but valid operation.

BINARY ADDER / SUBTRACTOR / INCREMENTER



ARITHMETIC CIRCUIT



S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B'	$D = A + B'$	Subtract with borrow
0	1	1	B'	$D = A + B' + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

- **Logic Operation**

A group of bits called strings is stored in the register, the logic microoperation consider each bit of register as compared to the arithmetic microoperation where we take whole content of register to perform arithmetic operation.

- And $\rightarrow C \leftarrow A \wedge B$
- OR $\rightarrow C \leftarrow A \vee B$

- **Shift Operation**

Shift microoperation transfer binary bits between the register serially. This shift operation is also used to perform binary multiplication and division.

Left shift $\rightarrow shl$

Right shift $\rightarrow shr$

For one bit shifting

$R2 \leftarrow shr R2$

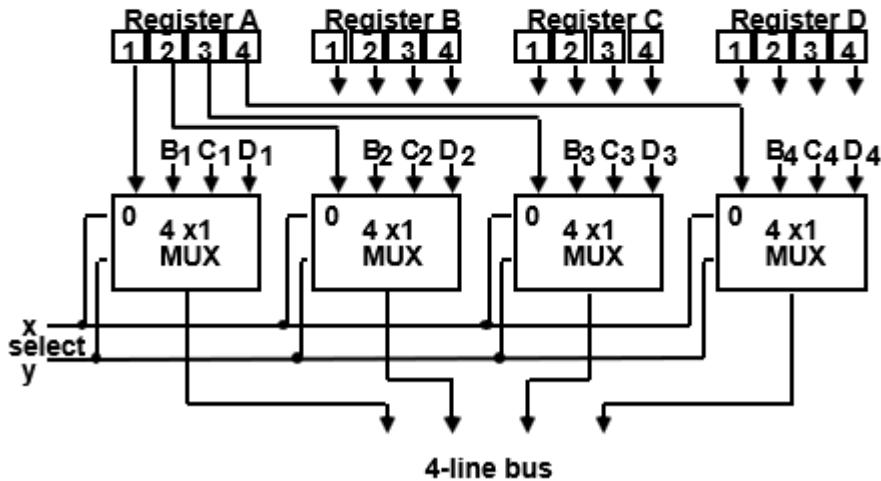
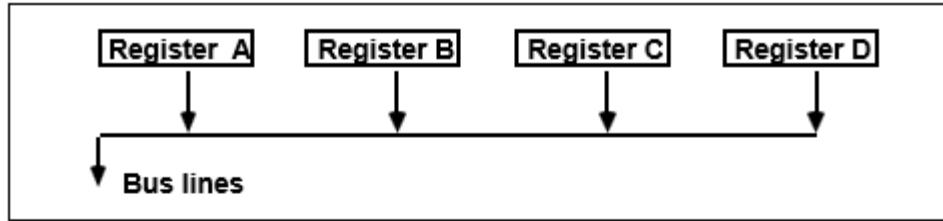
Bus & Memory Transfer

A digital computer has many registers and rather than connecting wires between all registers to transfer information between them, a common bus is used.

Bus is a path over which info is transferred from any of several sources to any of several destination.

From a register to Bus : $Bus \leftarrow R$

From Bus to Register : $R \leftarrow Bus$



Types of Bus:

- Data Bus
 - A data bus carries data.
 - It is an **electrical path** that connects CPU, memory, I/O devices and secondary storage.
 - The bus contain parallel group of lines.
 - The number of lines in bus affect the speed.
- Address Bus
 - An address bus carries address information.
 - It is set of wired similar to the data bus but **it only connects CPU and memory.**
 - Whenever the processor needs data from the memory, it places the address of data on the address bus.
 - The address is carried to the memory where the data from the requested address is fetched and placed on the data bus.
- Control Bus

- The control bus carries control information from the control unit to the other units
- The control information is used for **directing the activities of all units**
- The control unit controls the functioning of other units, ex : input/output devices, etc..

Bus Arbitration

Bus master: The device that is allowed to initiate data transfer on the bus at any given time is called data bus master.

Bus arbitration is the **process** by which the **next device to become the bus master** is selected and bus mastership is transferred to it. Selecting of bus master is usually done on priority basis.

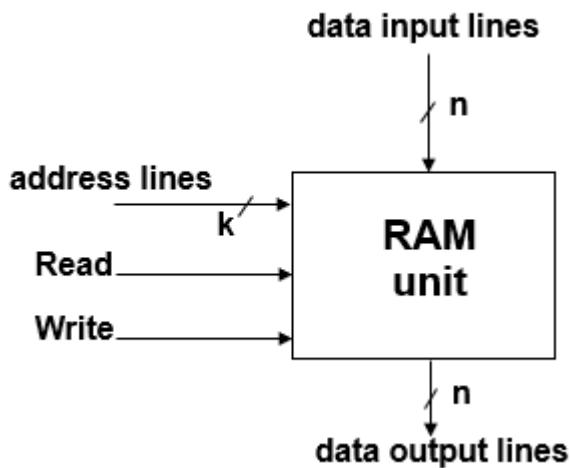
There are two approaches

- Centralized arbitration
- Distributed arbitration

Memory (RAM)

- sequential circuits containing some number of registers
- these registers hold the words of memory
- each of the r register is indicated by an address
- these addresses range from 0 to $r-1$
- each register can hold n bits of data
- Ram contains $r = 2^k$ words
 - n - data input line
 - n data output line
 - k address lines
 - a read control line

- a write control line



Memory is usually accessed in computer by putting desired address in a special register, Memory Address Register (MAR or AR)

When memory is accessed, the contents of the MAR send to the memory unit's address lines.

Memory read: $R1 \leftarrow M[MAR]$

- the contents of the MAR get sent to the memory address lines
- Read = 1
- The contents of the address are put on the memory's output data lines
- these get sent over the bus to be loaded into register R1

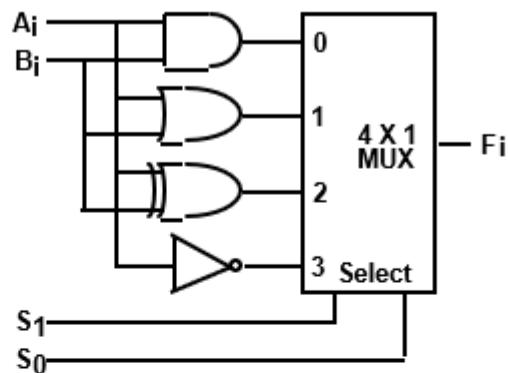
Memory write : $M[MAR] \leftarrow R1$

- The contents of the MAR get sent to the memory address lines
- Write = 1
- The values of R1 get sent over the bus to the data input lines of the memory
- the values get loaded into the specified address in the memory

X	Y	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	

Corresponding to logic micro-operations

x y	0 0 1 1 0 1 0 1	Boolean Function	Micro- Operations	Name
	0 0 0 0	F0 = 0	F ← 0	Clear
	0 0 0 1	F1 = xy	F ← A ∧ B	AND
	0 0 1 0	F2 = xy'	F ← A ∧ B'	
	0 0 1 1	F3 = x	F ← A	Transfer A
	0 1 0 0	F4 = x'y	F ← A' ∧ B	
	0 1 0 1	F5 = y	F ← B	Transfer B
	0 1 1 0	F6 = x ⊕ y	F ← A ⊕ B	Exclusive-OR
	0 1 1 1	F7 = x + y	F ← A ∨ B	OR
	1 0 0 0	F8 = (x + y)'	F ← (A ∨ B)'	NOR
	1 0 0 1	F9 = (x ⊕ y)'	F ← (A ⊕ B)'	Exclusive-NOR
	1 0 1 0	F10 = y'	F ← B'	Complement B
	1 0 1 1	F11 = x + y'	F ← A ∨ B	
	1 1 0 0	F12 = x'	F ← A'	Complement A
	1 1 0 1	F13 = x' + y	F ← A' ∨ B	
	1 1 1 0	F14 = (xy)'	F ← (A ∧ B)'	NAND
	1 1 1 1	F15 = 1	F ← all 1's	Set to all 1's



Function table

$S_1 \quad S_0$	Output	μ -operation
0 0	$F = A \wedge B$	AND
0 1	$F = A \vee B$	OR
1 0	$F = A \oplus B$	XOR
1 1	$F = A'$	Complement

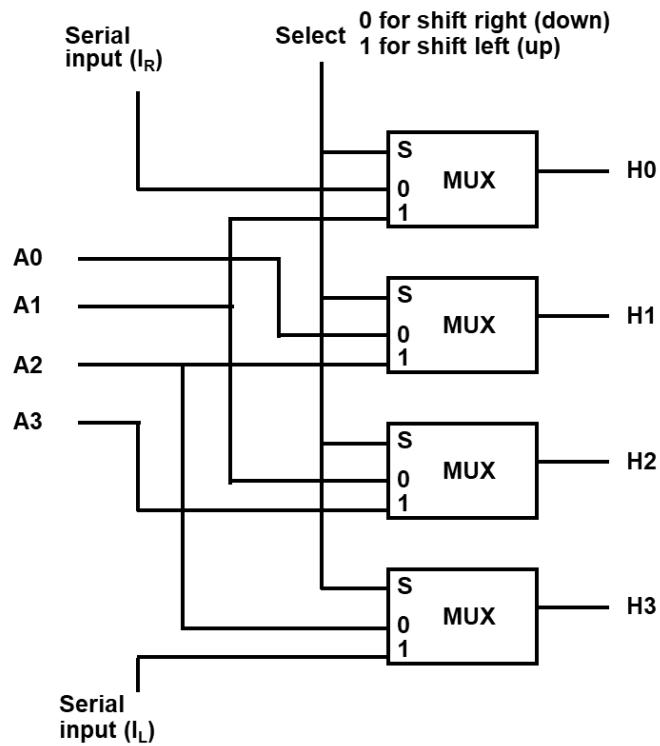
-cil for a circular shift left

-cir for a circular shift right

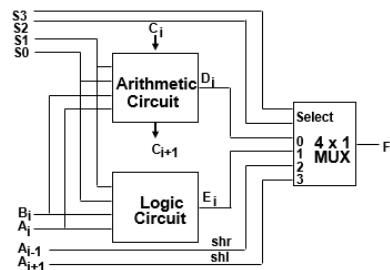
-ashl for an arithmetic shift left

-ashr for an arithmetic shift right

hardware implementation for shift microoperation



ARITHMETIC LOGIC SHIFT UNIT



Out of Syllabus

Page Replacement Algorithm

Page Replacement Algorithm decides which memory page to page or page-out (swap-out, write to disk) when a page of memory needs to be allocated.

When a program starts execution

- one or more pages transferred into main memory and
- the page-table is set to indicate their position

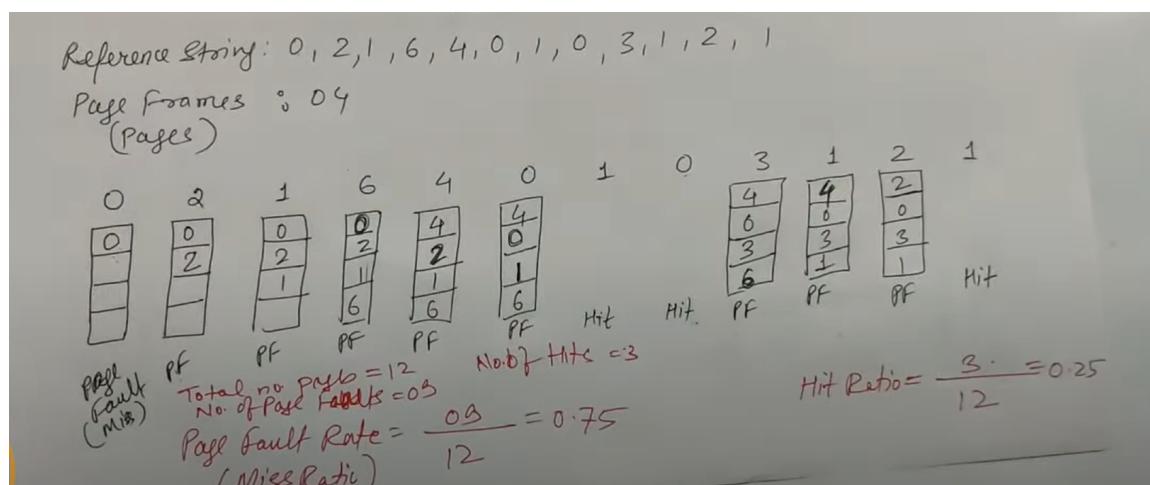
The program is executed from main memory until it attempts to reference a page that is still in auxiliary or secondary memory. This is called **Page Fault**.

Page Fault: If the required page is not present in main memory the page fault occurs, which requires I/O operation.

The goal of a replacement policy is to try to remove the page that least likely to be referenced in the immediate future.

Some Replacement Policies:

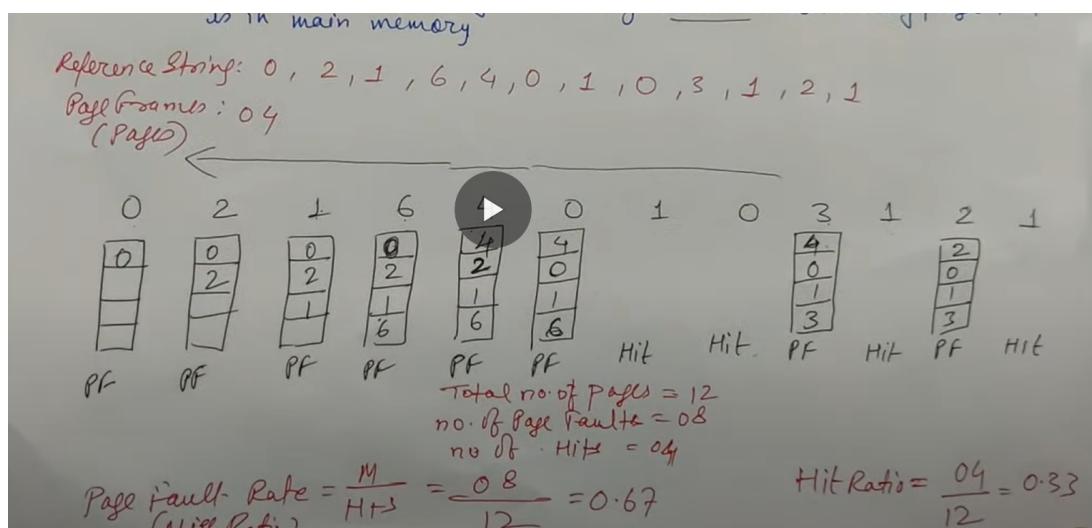
- FIFO
 - First-in-First-out
 - it is easy to implement
 - it selects the page for replacement that has been in memory for longest time
 - Oldest page in memory is selected by replacement
 - It keeps a list and replace page from tail add new page at the head



Page frames : number of pages that it can store in memory

- LRU
 - Least Recently Used
 - LRU selects a page for replacement which has not been used for the longest time in main memory
 - It keeps a list and replace page by **looking back into time**

- It is implemented associating a counter with every page that is in main memory



Optimal Page Replacement Algorithm

- It has lowest page fault rate of all algorithm
- Optimal policy replace the page that **will not be used for the longest period of time**
- It uses the time when a page is to be used next

