Memory Organization

Memory Hierarchy

- The memory unit is an essential component in any digital computer since it is needed for storing programs and data
- Not all accumulated information is needed by the CPU at the same time
- Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU

Memory Hierarchy

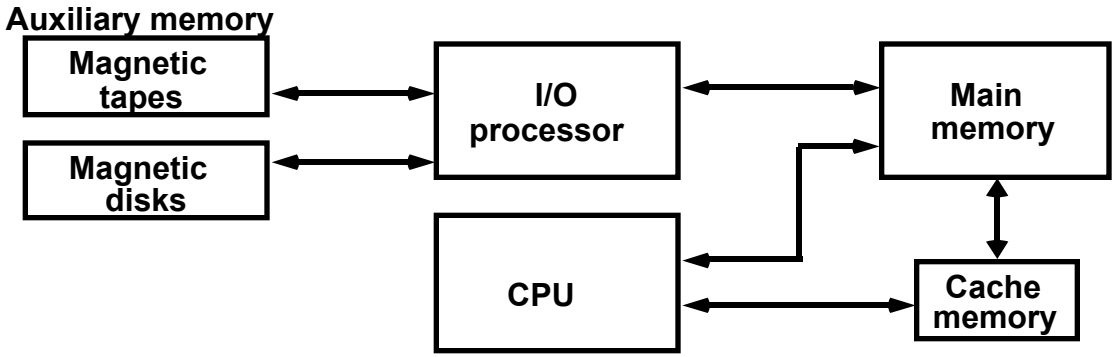
- The memory unit that directly communicate with CPU is called the main memory
- Devices that provide backup storage are called auxiliary memory

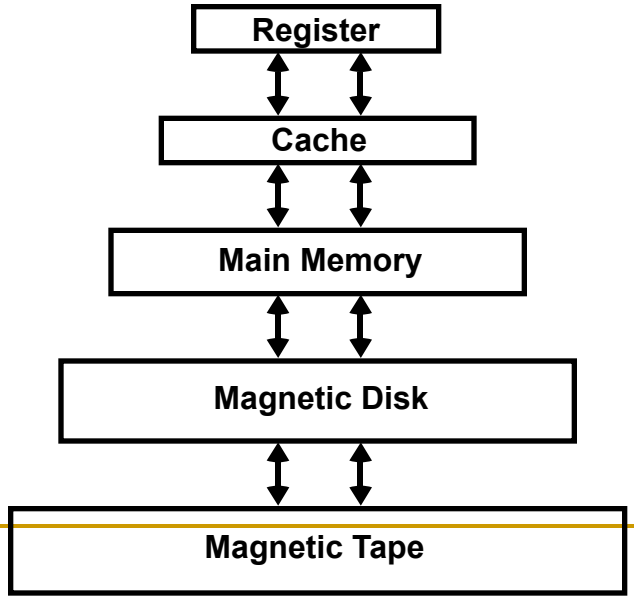
- The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory
- Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system

Memory Hierarchy

- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
- The typical access time ratio between cache and main memory is about 1to7
- Auxiliary memory access time is usually 1000 times that of main memory

MEMORY HIERARCHY





Main Memory

- Most of the main memory in a general purpose computer is made up of RAM integrated circuits chips, but a portion of the memory may be constructed with ROM chips

- RAM– Random Access memory
  - Integrated RAM are available in two possible operating modes, S*tatic and Dynamic*
- ROM– Read Only memory

Random-Access Memory (RAM)

- Static RAM (SRAM)
  - Each cell stores bit with a six-transistor circuit.
  - Retains value indefinitely, as long as it is kept powered.
  - Relatively insensitive to disturbances such as electrical noise.
  - Faster and more expensive than DRAM.

- Dynamic RAM (DRAM)
  - Each cell stores bit with a capacitor and transistor.
  - Value must be refreshed every 10-100 ms.
  - Sensitive to disturbances.
  - Slower and cheaper than SRAM.

ROM

- ROM is used for storing programs that are **PERMANENTLY** resident in the computer and for tables of constants that do not change in value once the production of the computer is completed
- The ROM portion of main memory is needed for storing an initial program called *bootstrap loader,* witch is to start the computer software operating when power is turned off

RAM

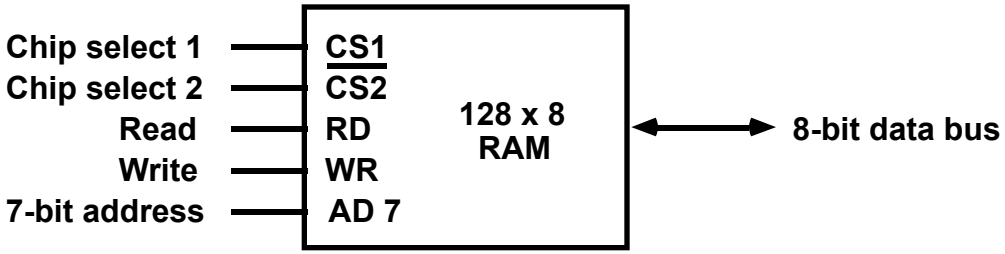
- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip when needed

- The Block diagram of a RAM chip is shown next slide, the capacity of the memory is 128 words of 8 bits (one byte) per word

MAIN MEMORY

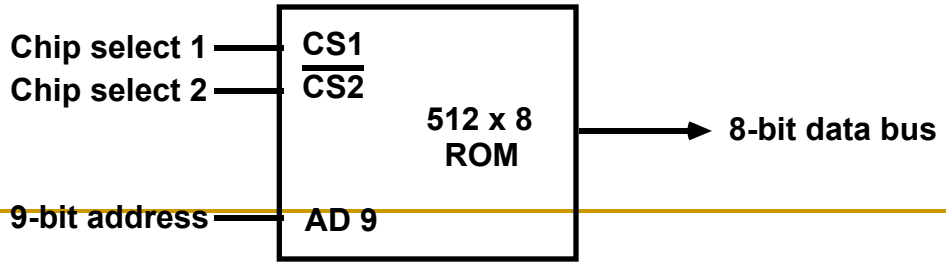
RAM and ROM Chips

Typical RAM chip



CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedence
0	1	x	x	Inhibit	High-impedence
1	0	0	0	Inhibit	High-impedence
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedence

Typical ROM chip



Memory Address Map

- Memory Address Map is a pictorial representation of assigned address space for each chip in the system
- To demonstrate an example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM
- The RAM have 128 byte and need seven address lines, where the ROM have 512 bytes and need 9 address lines

MEMORY ADDRESS MAP

Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

Component	Hexa address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Memory Connection to CPU

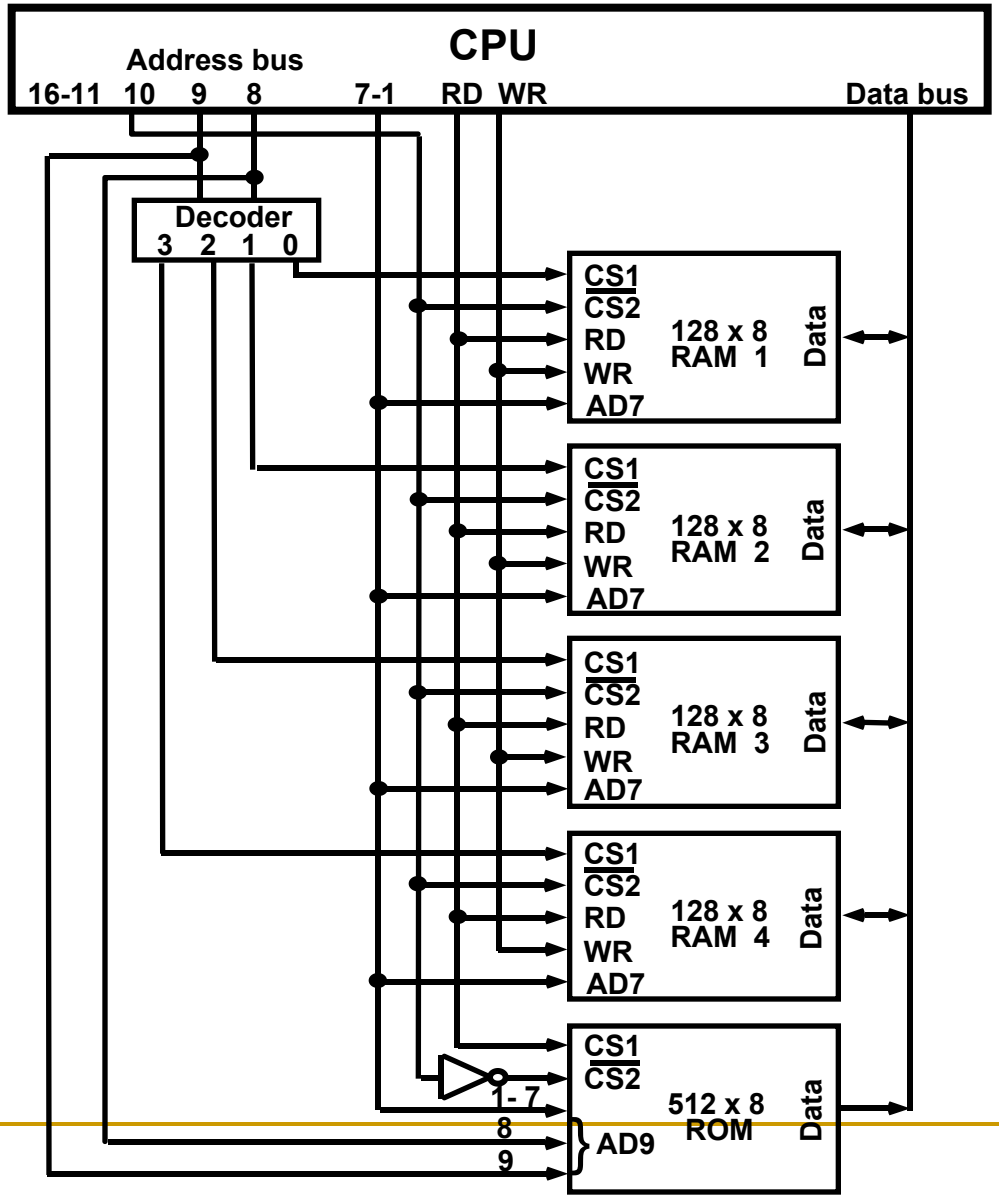
- **RAM and ROM chips are connected to a CPU through the data and address buses**

- **The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs**

Memory Address Map

- The hexadecimal address assigns a range of hexadecimal equivalent address for each chip
- Line 8 and 9 represent four distinct binary combination to specify which RAM we chose
- When line 10 is 0, CPU selects a RAM. And when it's 1, it selects the ROM

CONNECTION OF MEMORY TO CPU



Auxiliary Memory

- The average time required to reach a storage location in memory and obtain its contents is called the **access** time

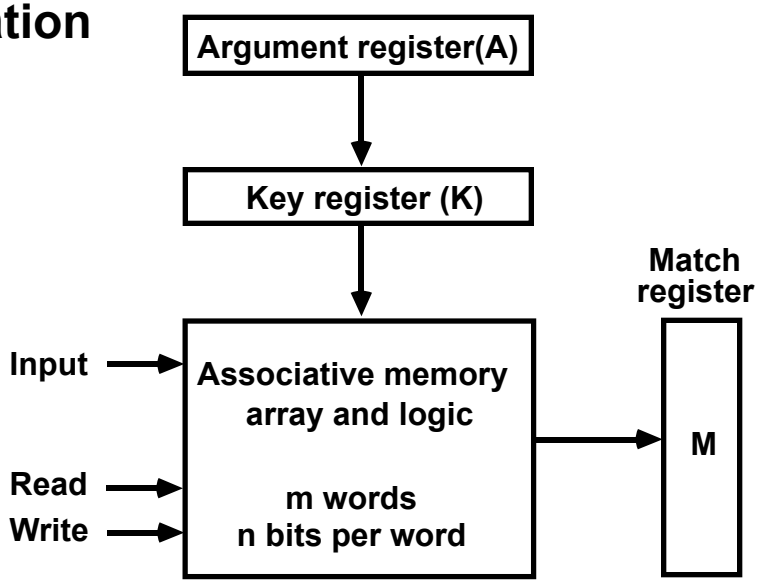
- The access time = seek time + transfer time
  - Seek time: required to position the read-write head to a location
  - Transfer time: required to transfer data to or from the device

ASSOCIATIVE MEMORY

- Content-addressed or associative memory refers to a memory organization in which the memory is accessed by its content (as opposed to an explicit address).
- Also called Content Addressable Memory (CAM)
- Searching becomes easy.

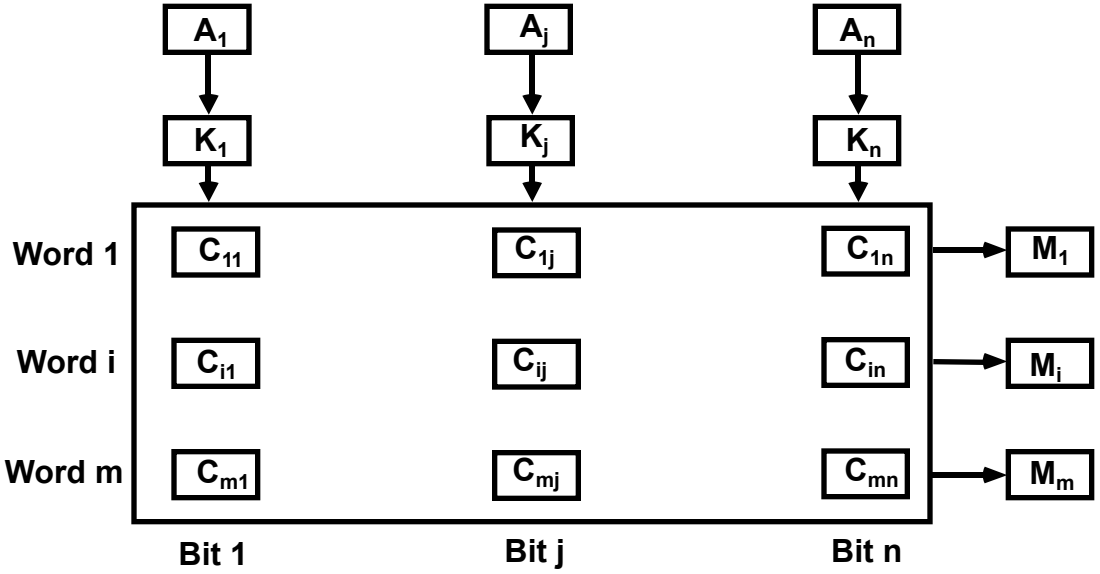
ASSOCIATIVE MEMORY

Hardware Organization

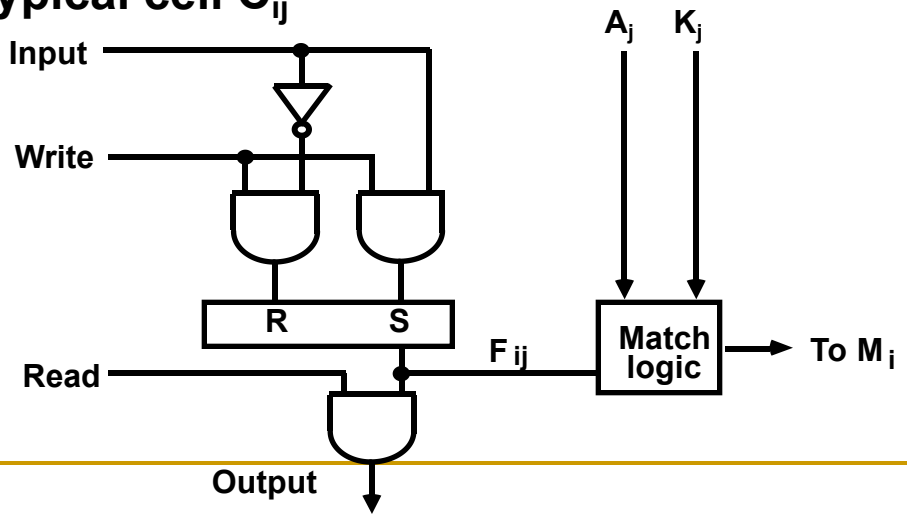


- Compare each word in CAM in parallel with the content of A(Argument Register)
- If CAM Word[i] = A, M(i) = 1
- Read sequentially accessing CAM for CAM Word(i) for M(i) = 1
- K(Key Register) provides a mask for choosing a particular field or key in the argument in A (only those bits in the argument that have 1's in their corresponding position of K are compared)

ORGANIZATION OF CAM



Internal organization of a typical cell C_{ij}



ASSOCIATIVE MEMORY cont…

$$x_j = A_j F_{ij} + A_j' F_{ij}'$$

$$M_i = x_1 x_2 x_3 \ldots x_n$$

$$x_j + K_j' = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

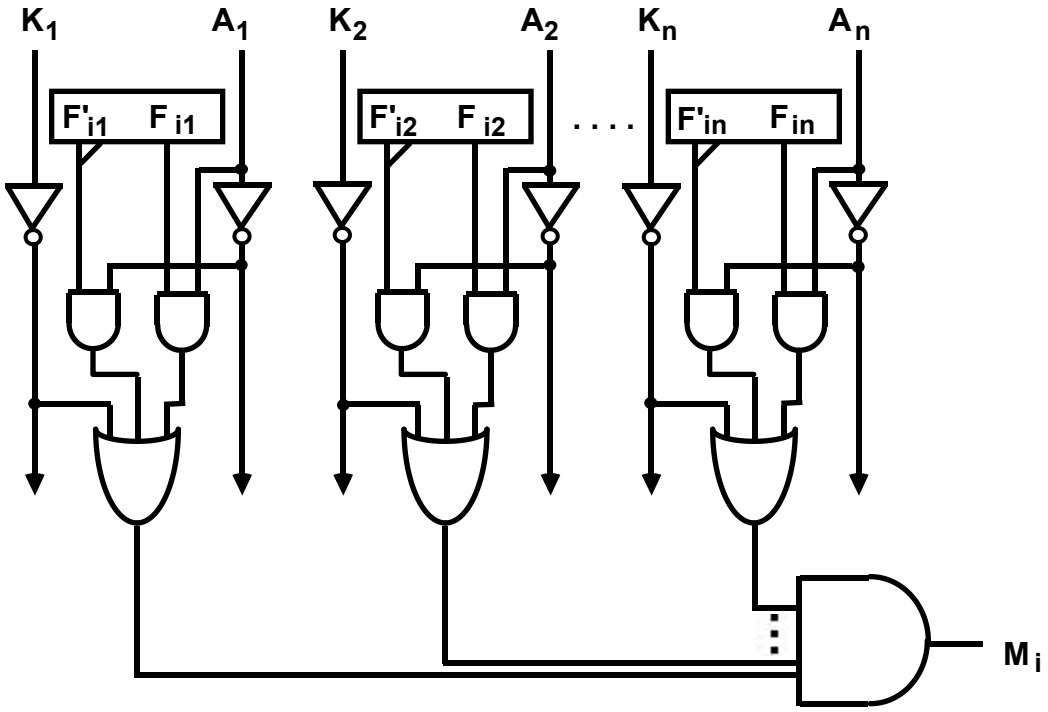
When $K_j = 1$, we have $K_j' = 0$ and $x_j + 0 = x_j$. When $K_j = 0$, then $K_j' = 1$ and $x_j + 1 = 1$. A term $(x_j + K_j')$ will be in the 1 state if its pair of bits is not compared. This is necessary because each term is ANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when $K_j = 1$.

The match logic for word i in an associative memory can now be expressed by the following Boolean function:

$$M_i = (x_1 + K_1^{'})(x_2 + K_2^{'})(x_3 + K_3^{'}).....(x_n + K_n^{'})$$

$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A_j^{'} F_{ij}^{'} + K_j^{'})$$

MATCH LOGIC



Cache memory

- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced,
- Thus reducing the total execution time of the program
- Such a fast small memory is referred to as cache memory
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component

Cache memory

- When CPU needs to access memory, the cache is examined
- If the word is found in the cache, it is read from the fast memory
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word

Cache memory

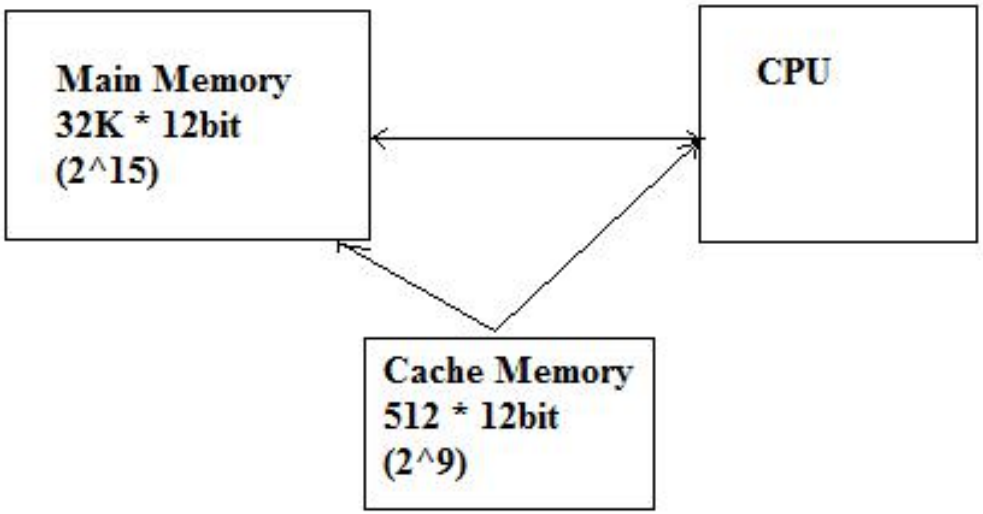
- The performance of cache memory is frequently measured in terms of a quantity called **hit ratio**
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**
- Otherwise, it is a **miss**
- **Hit ratio = hit / (hit+miss)**

Cache memory

- The basic characteristic of cache memory is its fast access time,
- Therefore, very little or no time must be wasted when searching the words in the cache
- The transformation of data from main memory to cache memory is referred to as a **mapping** process, there are three types of mapping:
  - Associative mapping
  - Direct mapping
  - Set-associative mapping

Cache memory

- To help understand the mapping procedure, we have the following example:



Associative mapping

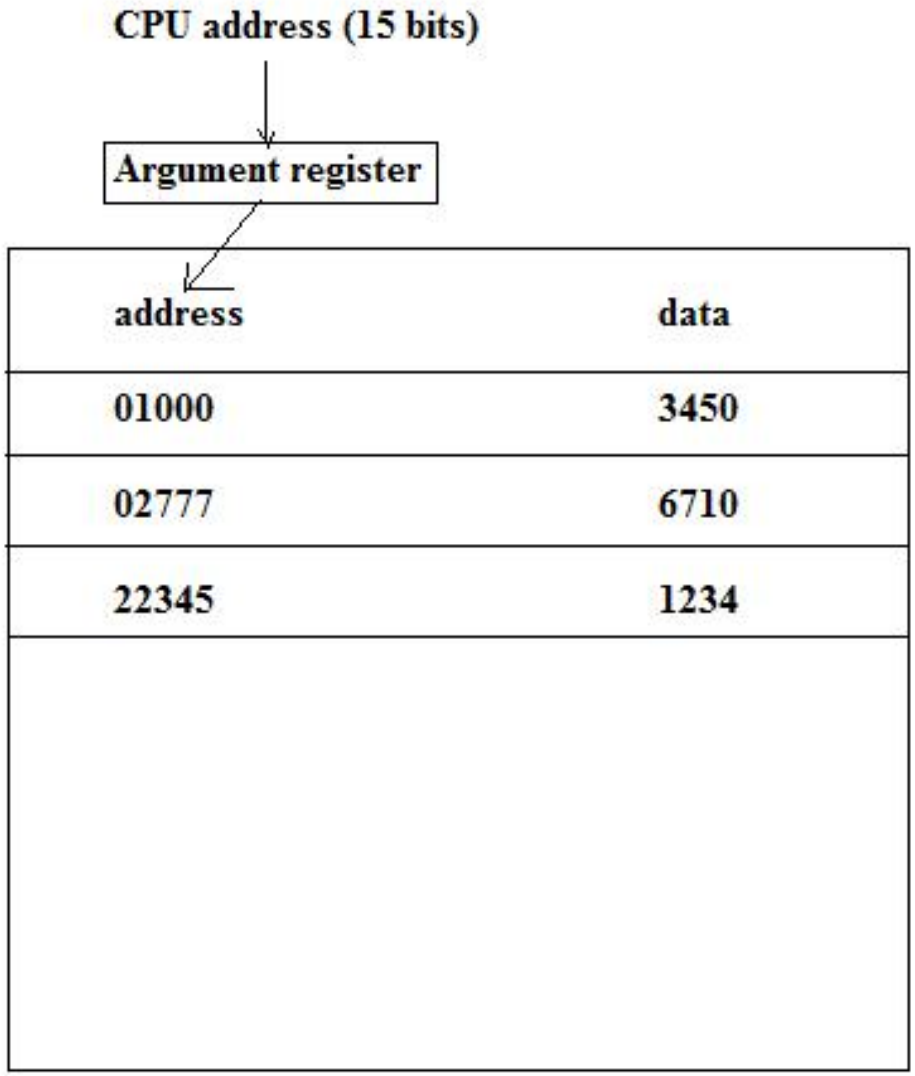
- The fastest and most flexible cache organization uses an associative memory
- The associative memory stores both the address and data of the memory word
- This permits any location in cache to store ant word from main memory

- The address value of 15 bits is shown as a five-digit **octal** number and its corresponding 12-bit word is shown as a four-digit octal number

Associative mapping

- A CPU address of 15 bits is places in the argument register and the associative memory us searched for a matching address
- If the address is found, the corresponding 12-bits data is read and sent to the CPU
- If not, the main memory is accessed for the word
- If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache

Associative mapping



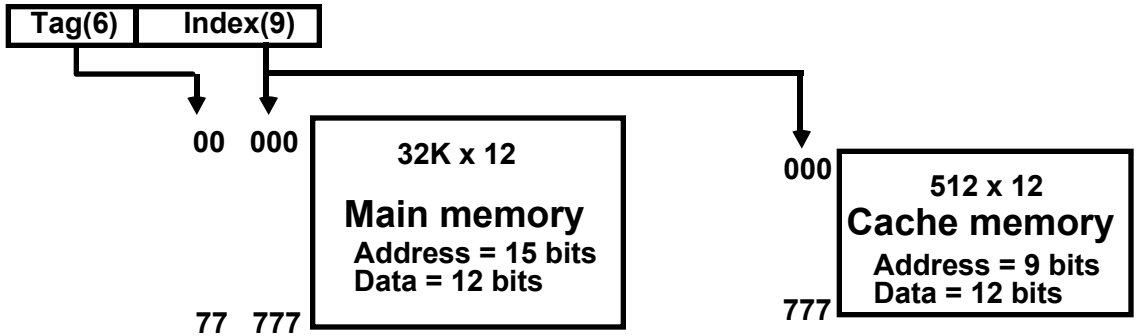
Direct Mapping

- Associative memory is expensive compared to RAM
- In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)
- The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field

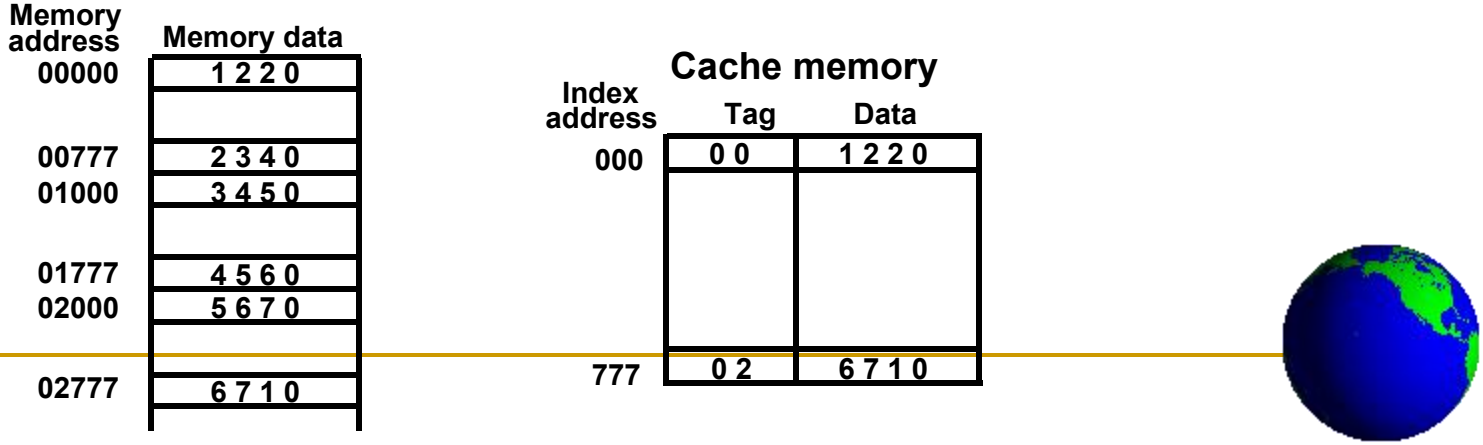
MEMORY AND CACHE MAPPING – DIRECT MAPPING

- Each memory block has only one place to load in Cache
- Mapping Table is made of RAM instead of CAM
- n-bit memory address consists of 2 parts; k bits of Index field and n-k bits of Tag field
- n-bit addresses are used to access main memory and k-bit Index is used to access the Cache

Addressing Relationships

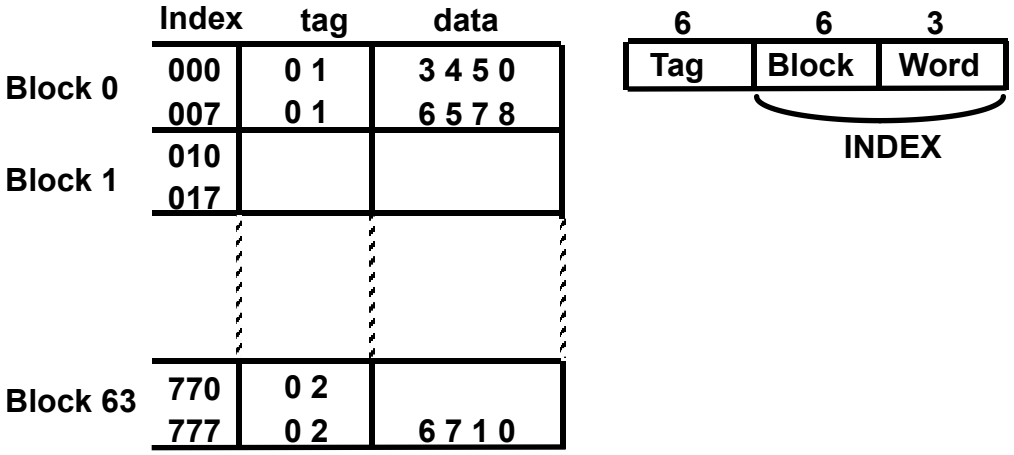


Direct Mapping Cache Organization



DIRECT MAPPING

Direct Mapping with block size of 8 words



Set-Associative Mapping

- The disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time
- Set-Associative Mapping is an improvement over the direct-mapping in that each word of cache can store two or more word of memory under the same index address

Set-Associative Mapping

- Each index address refers to two data words and their associated tags
- Each tag requires six bits and each data word has 12 bits, so the word length is 2*(6+12) = 36 bits

Set-Associative Mapping

Memory Address	Memory Data
00000	1220
00777	2340
01000	3450
01111	2222
01777	4560
02000	5670
02777	6710

Index Address	Tag	Data	Tag	Data
000	01	3450	02	5670
111	01	2222		
777	02	6710	00	2340

PERFORMANCE OF CACHE

Memory Access

All the memory accesses are directed first to Cache
If the word is in Cache; Access cache to provide it to CPU
If the word is not in Cache; Bring a block (or a line) including that word to replace a block now in Cache

- How can we know if the word that is required is there ?
- If a new block is to replace one of the old blocks, which one should we choose ?

Performance of Cache Memory System

Hit Ratio - % of memory accesses satisfied by Cache memory system
Te: Effective memory access time in Cache memory system
Tc: Cache access time
Tm: Main memory access time

$$Te = Tc + (1 - h)\, Tm$$

Example: Tc = 0.4 μ s, Tm = 1.2μ s, h = 0.85%

$$Te = 0.4 + (1 - 0.85) * 1.2 = 0.58\mu s$$

CACHE WRITE

Write Through

When writing into memory

If Hit, both Cache and memory is written in parallel
If Miss, Memory is written
For a read miss, missing block may be overloaded onto a cache block

Memory is always updated
-> Important when CPU and DMA I/O are both executing

Slow, due to the memory access time

Write-Back (Copy-Back)

When writing into memory

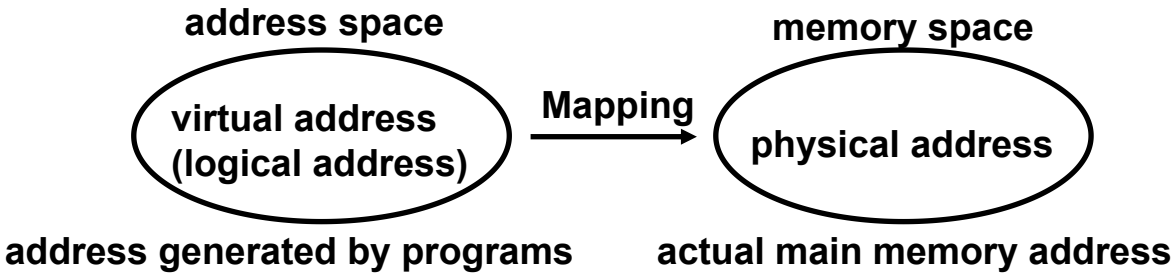
If Hit, only Cache is written
If Miss, missing block is brought to Cache and write into Cache
For a read miss, candidate block must be written back to the memory

Memory is not up-to-date, i.e., the same item in Cache and memory may have different value

VIRTUAL MEMORY

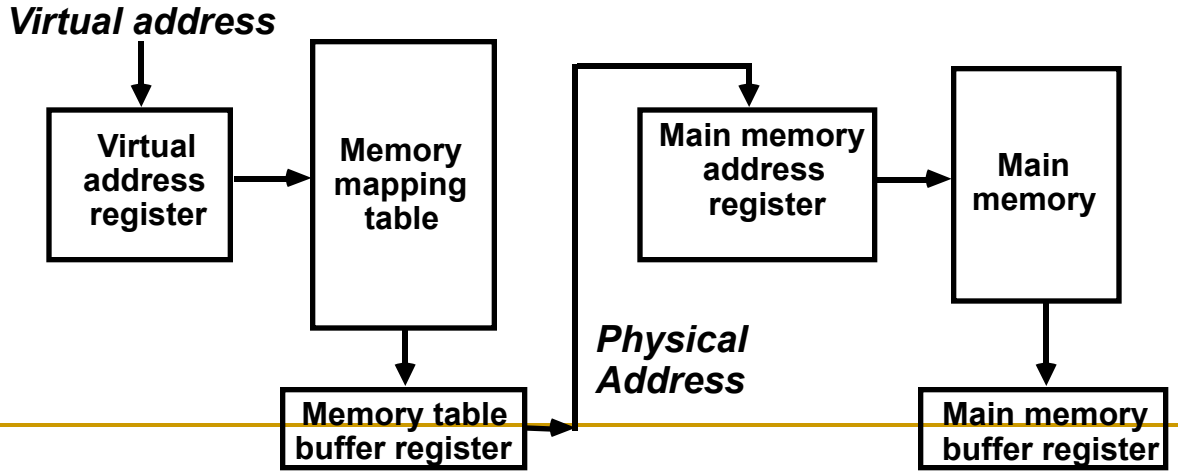
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)



Address Mapping

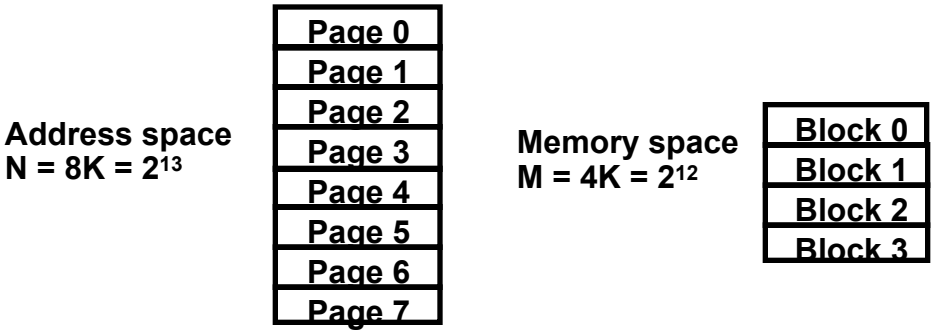
Memory *Mapping Table* for *Virtual Address -> Physical Address*



ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called *pages and blocks respectively*

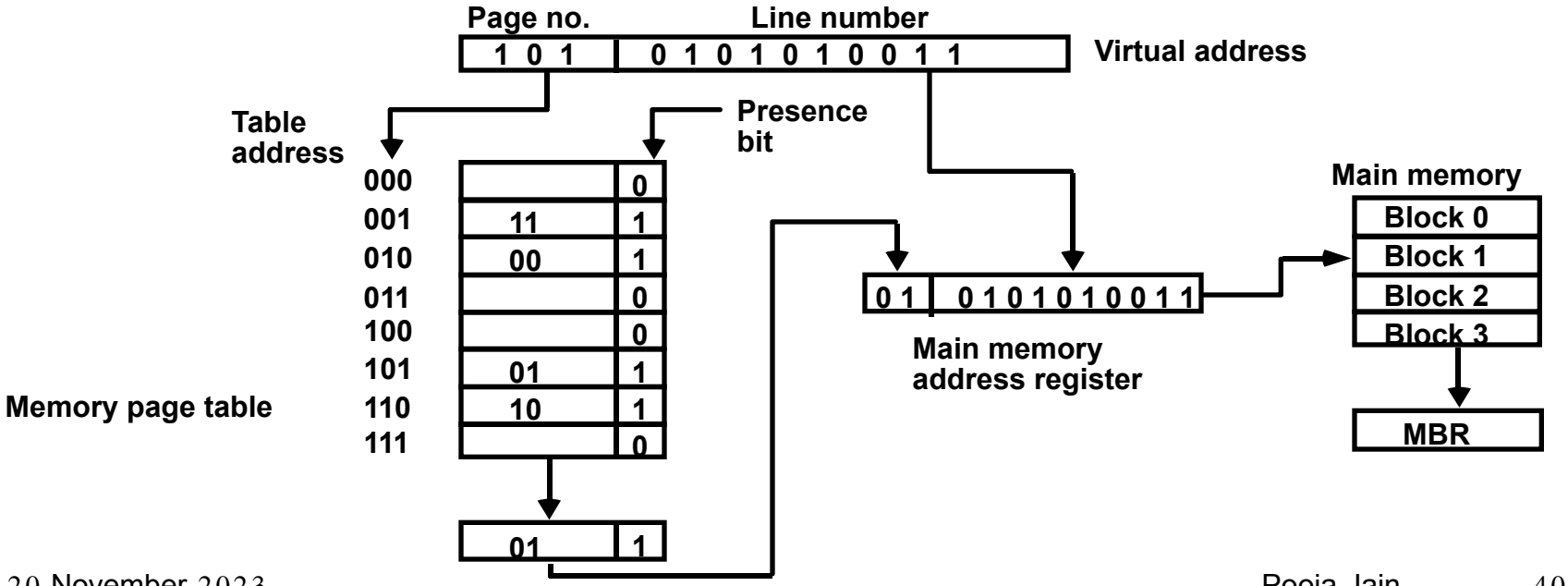
1K words group



The address in the page table denotes the page number and the content of the word gives the block number where the page is stored in main memory.

A presence bit in each location indicates whether the page has been transferred from auxiliary memory into the main memory

Organization of memory Mapping Table in a paged system



ASSOCIATIVE MEMORY PAGE TABLE

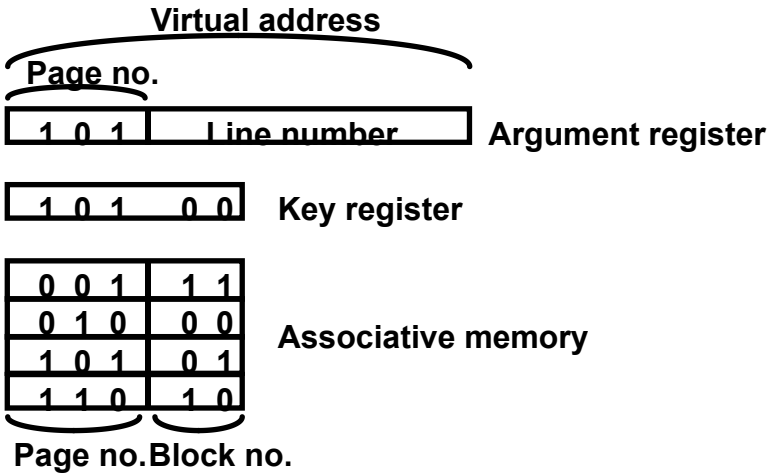
Assume that

Number of Blocks in memory = m
Number of Pages in Virtual Address Space = n

Page Table

- Straight forward design -> n entry table in memory
Inefficient storage space utilization
<- n-m entries of the table is empty

- More efficient method is m-entry Page Table
Page Table made of an Associative Memory
m words; (Page Number:Block Number)



Page Fault

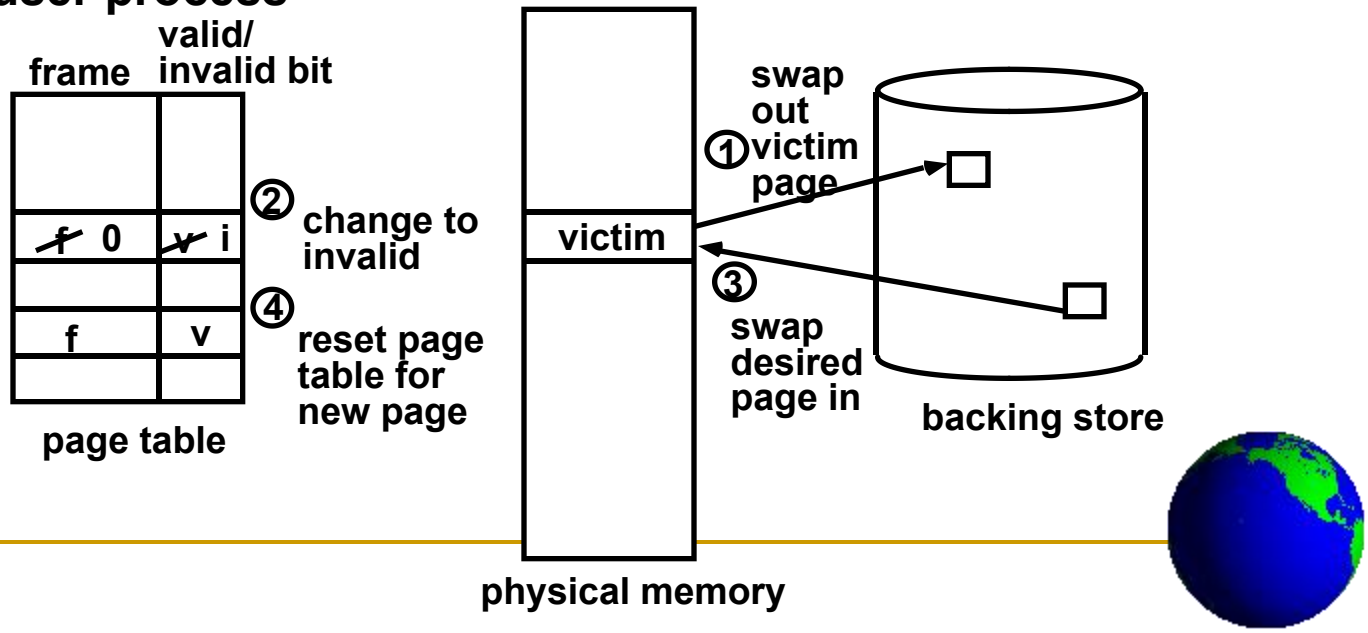
Page number cannot be found in the Page Table

PAGE REPLACEMENT

Decision on which page to displace to make room for an incoming page when no free frame is available

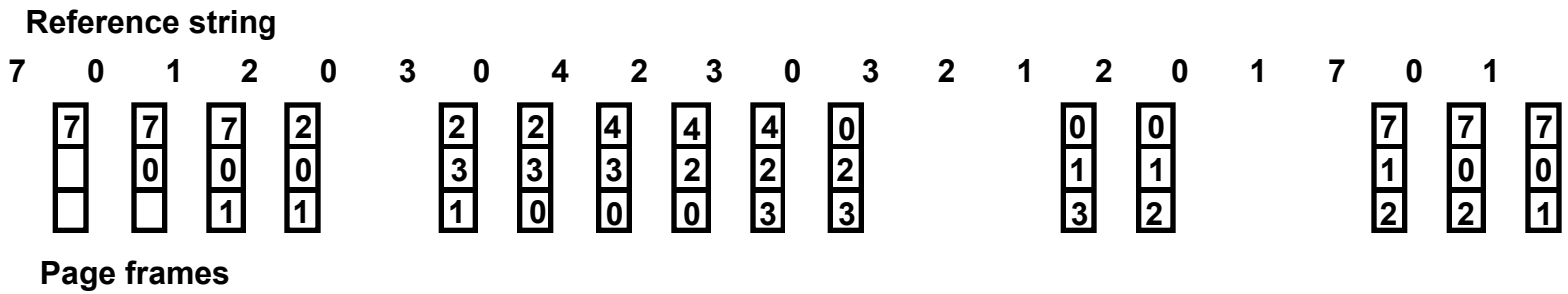
Modified page fault service routine

1. **Find the location of the desired page on the backing store**
2. **Find a free frame**
   - **If there is a free frame, use it**
   - **Otherwise, use a page-replacement algorithm to select a *victim* frame**
   - **Write the victim page to the backing store**
3. **Read the desired page into the (newly) free frame**
4. **Restart the user process**



PAGE REPLACEMENT ALGORITHMS

FIFO



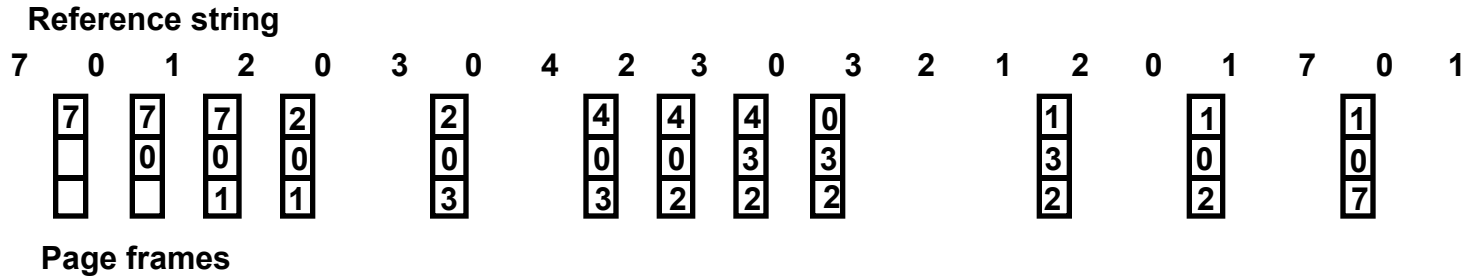
FIFO algorithm selects the page that has been in memory the longest time
Using a queue - every time a page is loaded, its identification is inserted in the queue
Easy to implement
May result in a frequent page fault

PAGE REPLACEMENT ALGORITHMS

LRU

- LRU uses the recent past as an approximation of near future.

Replace that page which has not been used for the longest period of time



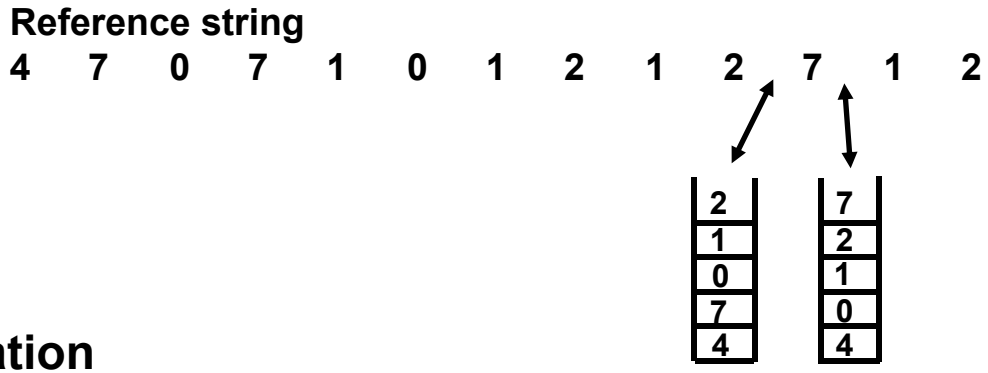
- LRU may require substantial hardware assistance

- The problem is to determine an order for the frames defined by the time of last use

PAGE REPLACEMENT ALGORITHMS

LRU Implementation Methods

- **Counters**
  - **For each page table entry - time-of-use register**
  - **Incremented for every memory reference**
  - **Page with the smallest value in time-of-use register is replaced**
- **Stack**
  - **Stack of page numbers**
  - **Whenever a page is referenced its page number is removed from the stack and pushed on top**
  - **Least recently used page number is at the bottom**



LRU Approximation

- **Reference (or use) bit is used to approximate the LRU**
- **Turned on when the corresponding page is referenced after its initial loading**
- **Additional reference bits may be used**

MEMORY MANAGEMENT HARDWARE

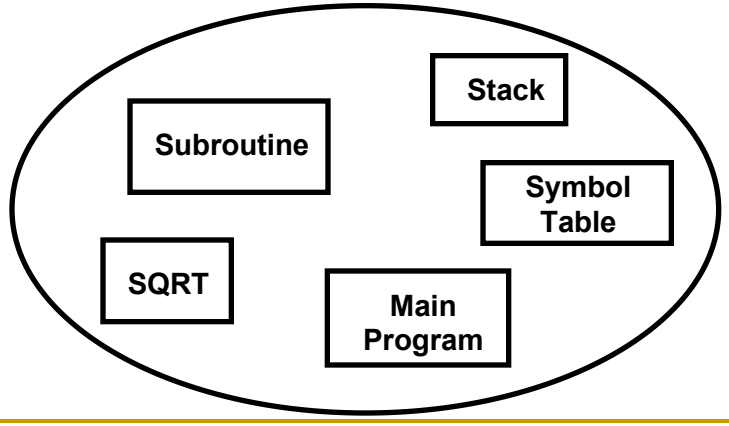
Basic Functions of MM

- ***Dynamic Storage Relocation*** **- mapping logical memory references to physical memory references**
- **Provision for *Sharing* common information stored in memory by different users**
- ***Protection*** **of information against unauthorized access**

Segmentation

- **A segment is a set of logically related instructions or data elements associated with a given name**
- **Variable size e.g. subroutine, array of data, a table of symbols etc**

User's view of memory



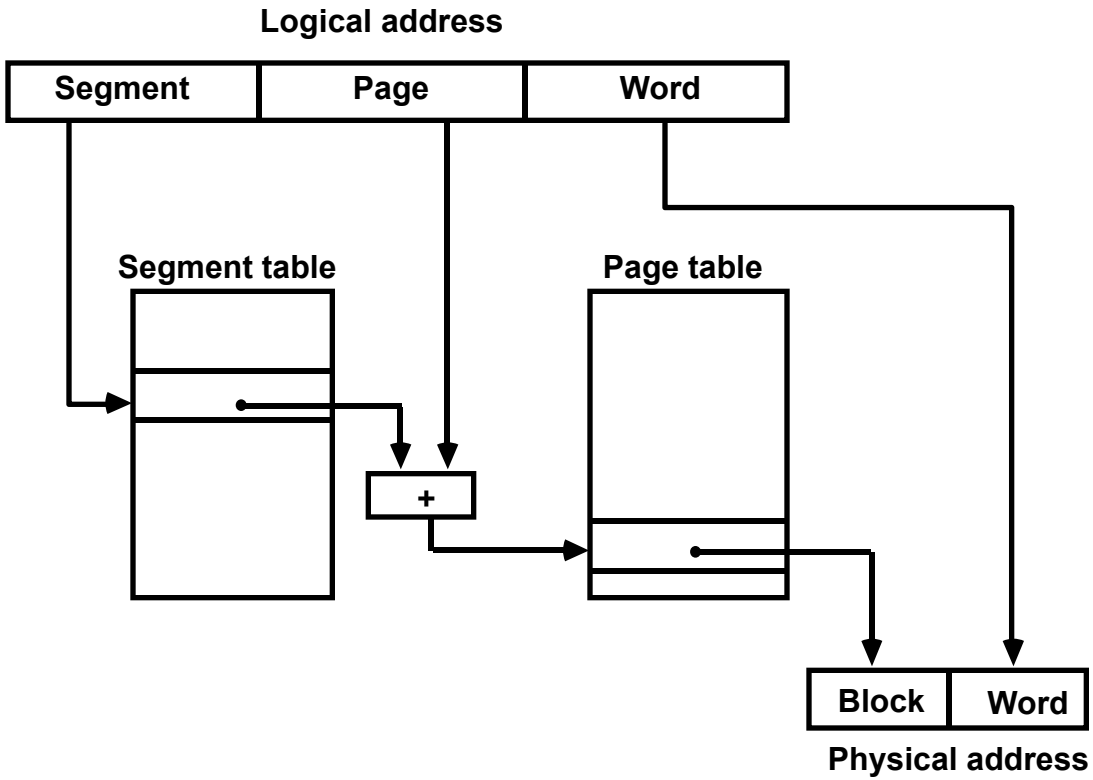
User's view of a program

The user does not think of memory as a linear array of words. Rather the user prefers to view memory as a collection of variable sized segments, with no necessary ordering among segments.

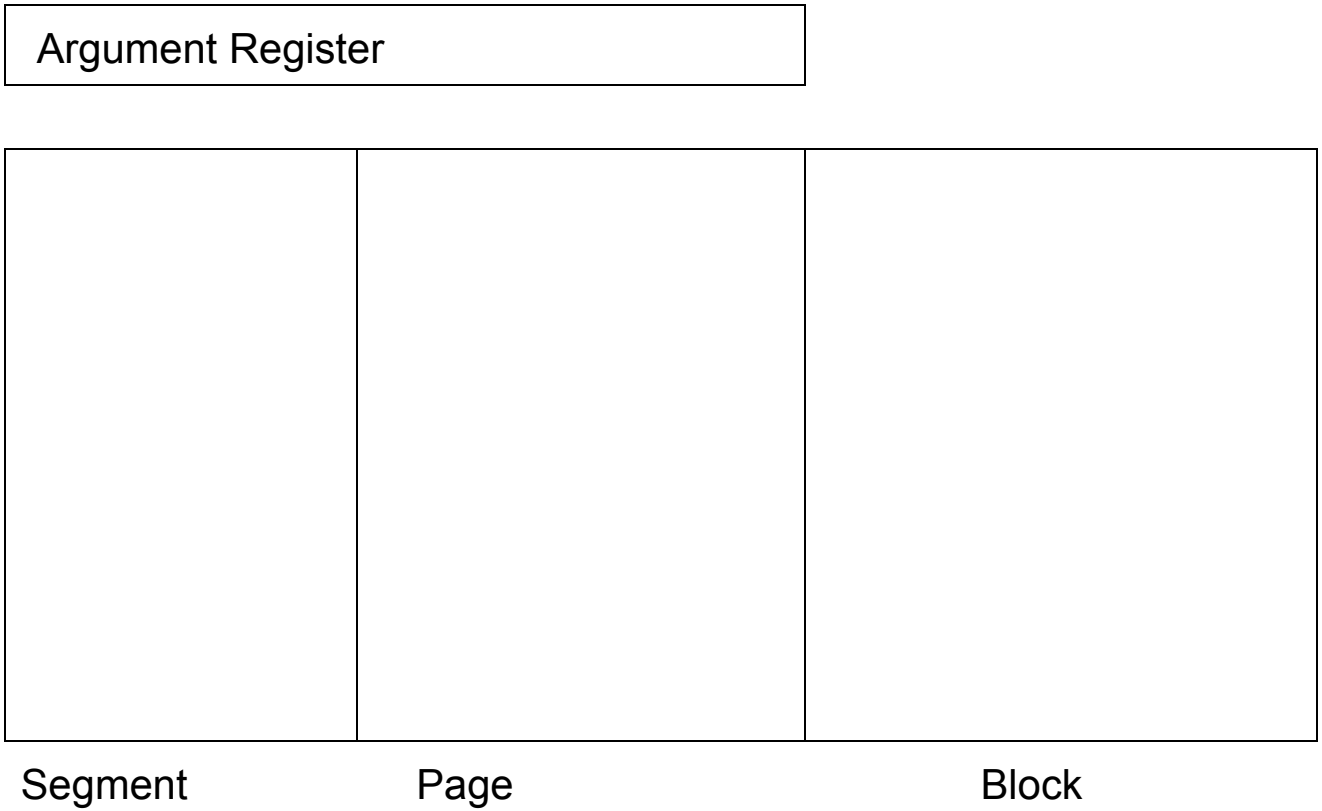
Logical address

- The address generated by a segmented program is called a logical address.
- Its associated with variable length segments rather than fixed length pages
- The logical address can be larger, equal or even smaller than the length of the physical memory address.
- In addition to relocation info, each segment has protection info associated with it.

SEGMENTED PAGE SYSTEM



Associative memory TLB



EXAMPLE

Logical and Physical Addresses

2^{20} x 32 Physical memory

Logical address format: 16 segments of 256 pages each, each page has 256words

4	8	8
Segment	Page	Word

Physical address format: 4096 blocks of 256 words each, each word has 32bits

12	8
Block	Word

Logical and Physical Memory Address Assignment

Hexa address	Page number
60000	Page 0
60100	Page 1
60200	Page 2
60300	Page 3
60400 604FF	Page 4

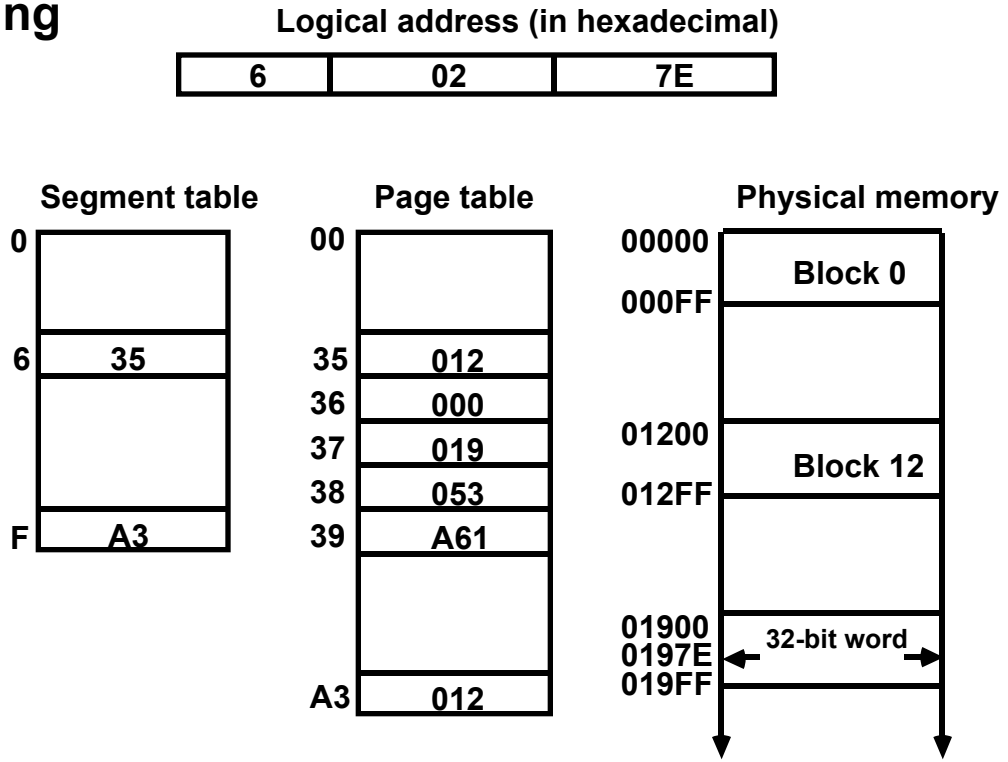
(a) Logical address assignment

Segment	Page	Block
6	00	012
6	01	000
6	02	019
6	03	053
6	04	A61

(b) Segment-page versus memory block assignment

LOGICAL TO PHYSICAL MEMORY MAPPING

Segment and page table mapping



Associative memory mapping

Segment	Page	Block
6	02	019
6	04	A61

MEMORY PROTECTION

Protection information can be included in the segment table or segment register of the memory management hardware

- Format of a typical segment descriptor

Base address	Length	Protection

- The protection field in a segment descriptor specifies the *Access Rights* to the particular segment

- In a segmented-page organization, each entry in the page table may have its own protection field to describe the Access Rights of each page

- Access Rights:

Full read and write privileges.
Read only (write protection)
Execute only (program protection)
System only (O.S. Protection)