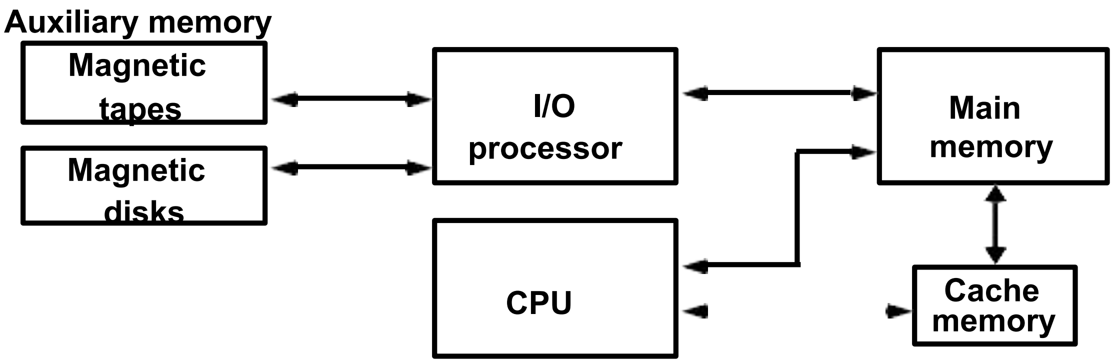
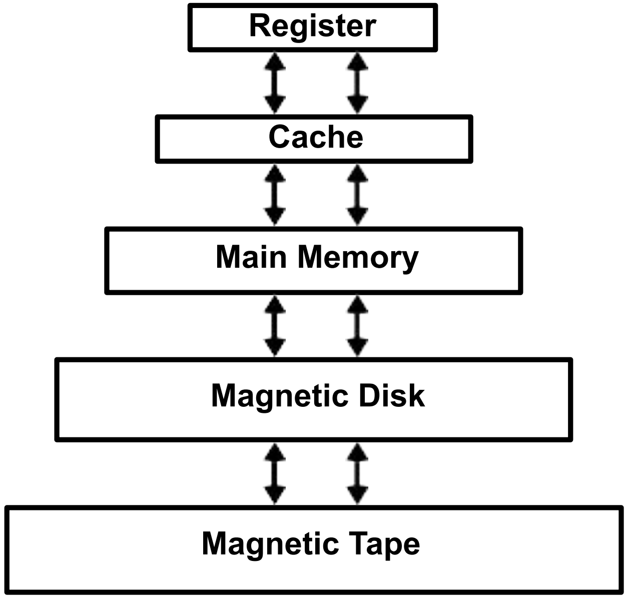
MEMORY ORGANIZATION

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Virtual Memory
- Memory Management Hardware

MEMORY HIERARCHY

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system





Memory Unit is an essential component in any digital computer since it is needed for storing programs and data. Memory is just like a brain which is useful for storing information into the computer.

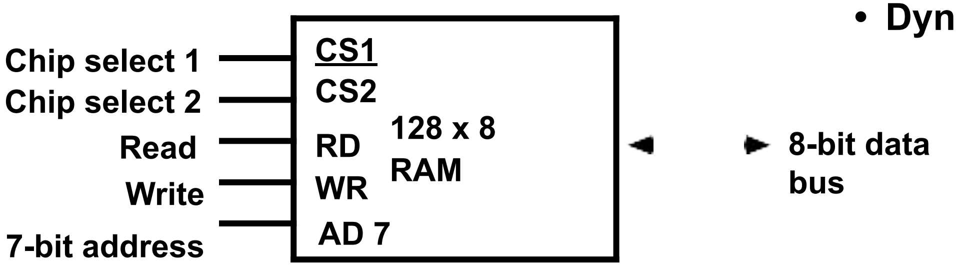
Several types of memory are there:

1. **Main memory/Primary memory/Volatile memory** is used to communicate directly with the CPU. CPU is mainly used to execute any task into the computer. CPU can execute only that program which will be present into the main memory. If we are going to save any program by default it get saved into the secondary memory but during execution DMA/Operating system transfers the program to primary memory.
2. **Auxiliary memory/secondary memory/Non-volatile memory** is used to store the data permanently into the computer. Examples are Magnetic disk and Magnetic tapes. Previously everyone is using magnetic tapes but now a days magnetic disks are used. CPU cannot access the data of auxiliary memory.
3. **Cache memory** is small and faster memory which has higher accessibility. Cache memory is very fast memory and size of the memory is very small. It is mainly used to store the data which is frequently used. Main memory access time is slower but cache memory access time is fast. So if there are any frequently used instructions then operating system/IO transfers the instructions of main memory to cache memory so that access will be fast.
4. **I/O processor** is mainly used to transfer the data from secondary memory to main memory.

MAIN MEMORY

RAM and ROM Chips

Typical RAM chip

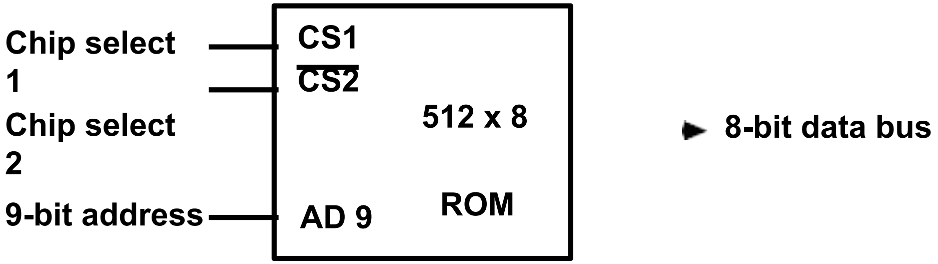


Types of Memory

- Static
- Dynamic

CS1	$\overline{\text{CS2}}$	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



Uses of ROM

- Bootstrap Loader
- Computer Startup

The **static RAM** consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. It is very fast but the cost is high. Cache memory is designed with the help of SRAM. The **dynamic RAM** stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. Main memory is designed with the help of DRAM.

RAM is a volatile memory when we switch off the computer the entire contents of memory will be lost but ROM is a permanent memory that means it stores contents permanently. Most portion of the memory is RAM only but less is ROM.

Bootstrap loader: Among other things, the ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again. The startup of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use. RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size. To demonstrate the chip interconnection, we will show an example of a 1024 x 8 memory constructed with 128 x 8 RAM chips and 512 x 8 ROM chips.

MEMORY ADDRESS MAP

Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

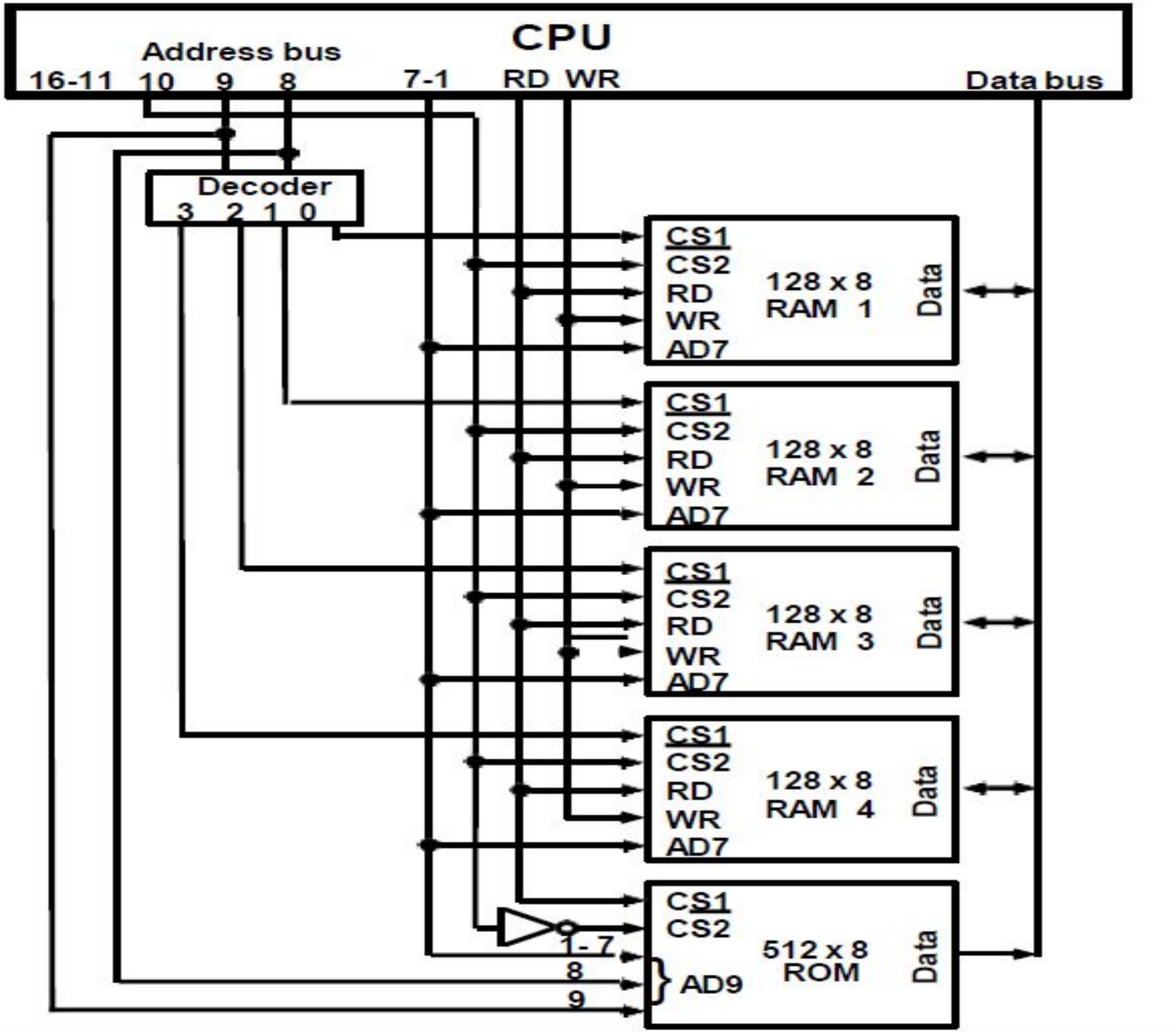
Component	Hexa address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Memory Connection to CPU

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

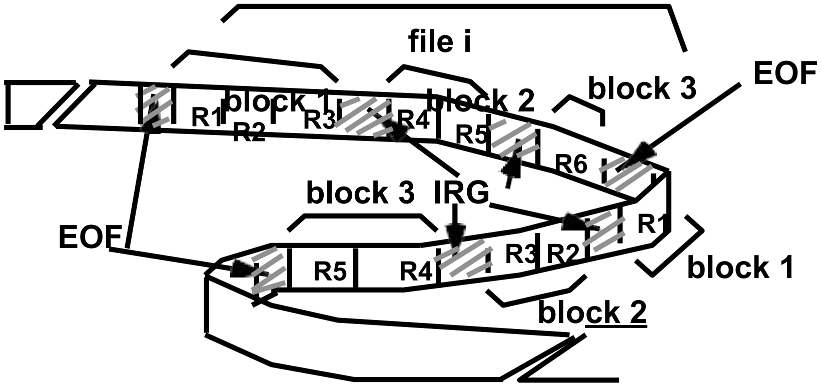
- **The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs**

CONNECTION OF MEMORY TO CPU



AUXILIARY MEMORY

Information Organization on Magnetic Tapes



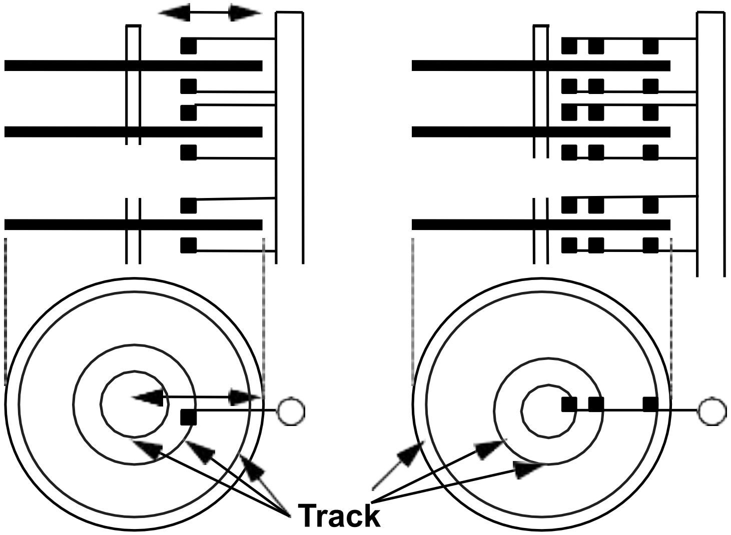
Characteristics of Auxiliary Memory

1. Access Mode
2. Access Time
3. Transfer rate
4. Capacity
5. Cost

- Access Time
  - Seek Time
  - Transfer Time

Organization of Disk Hardware

Moving Head Disk **Fixed Head Disk**

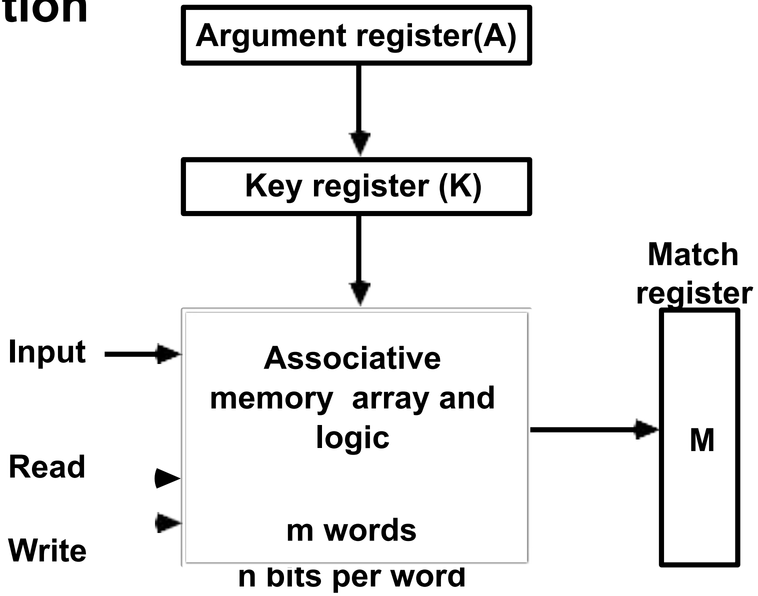


ASSOCIATIVE MEMORY

-Accessed by the content of the data rather than by an address

-Also called Content Addressable Memory (CAM)

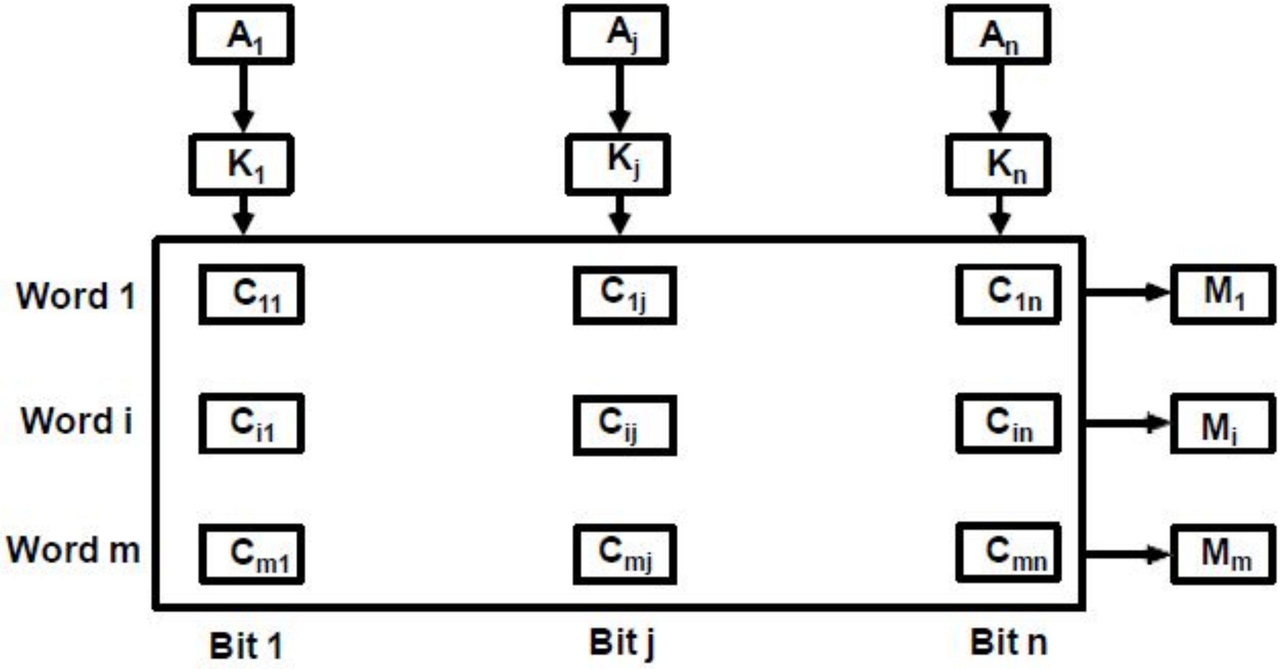
Hardware Organization



A:	101 111100
K:	111 000000
Word 1:	100 111100 (No Match)
Word 2:	101 000001 (Match)

- 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



MATCH LOGIC

$$x_j = A_j F_{ij} + A'_j F'_{ij}$$

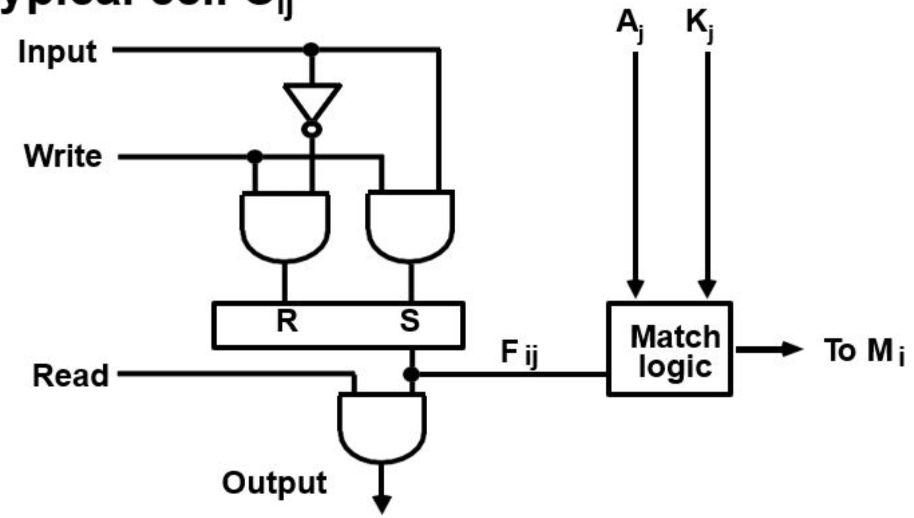
$$x_j + K'_j = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

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

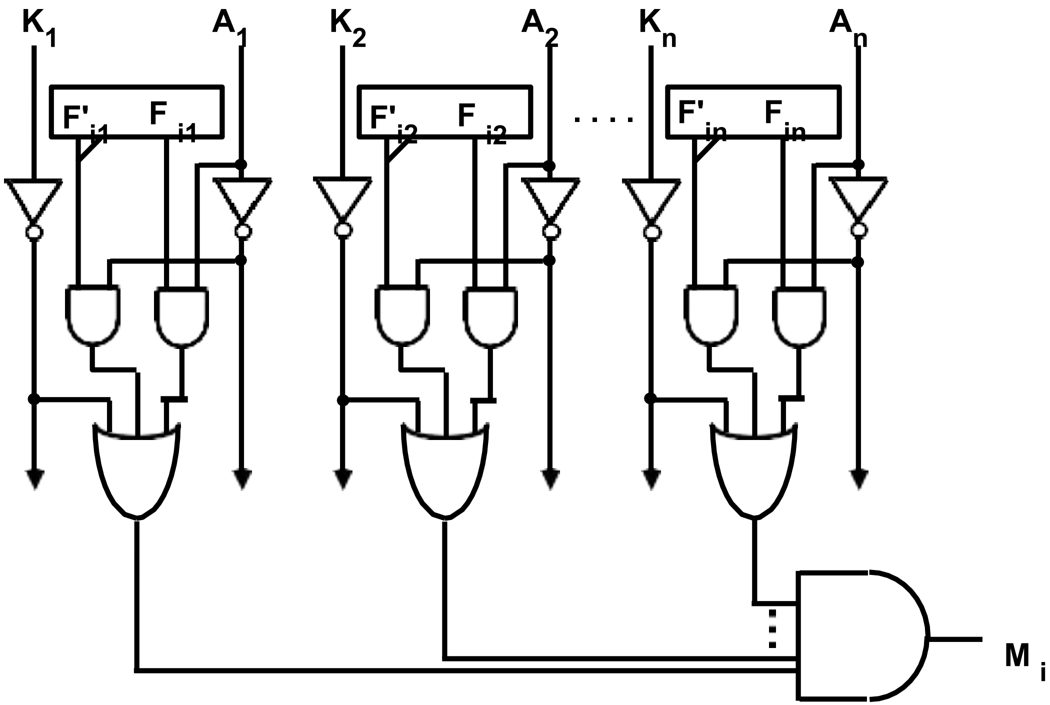
$$M_i = (x_1 + K'_1)(x_2 + K'_2)(x_3 + K'_3) \cdots (x_n + K'_n)$$

$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A'_j F'_{ij} + K'_j)$$

Internal organization of a typical cell C_{ij}



MATCH LOGIC



CACHE MEMORY

Locality of Reference

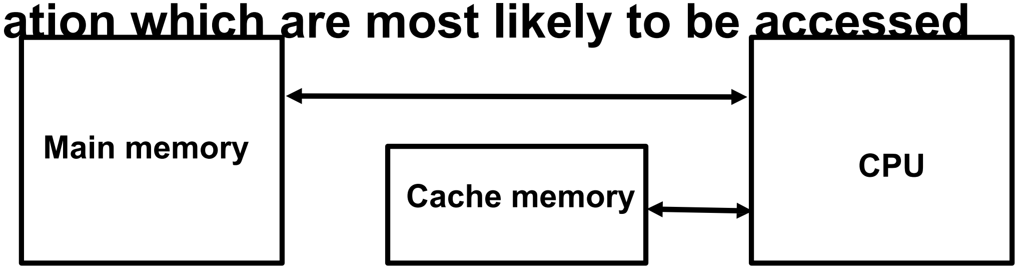
- The references to memory at any given time interval tend to be confined within a localized areas
- This area contains a set of information and the membership changes gradually as time goes by
- *Temporal Locality*

  The information which will be used in near future is likely to be in use already (e.g. Reuse of information in loops)
- *Spatial Locality*

  If a word is accessed, adjacent(near) words are likely accessed soon (e.g. Related data items (arrays) are usually stored together; instructions are executed sequentially)

Cache

- The property of Locality of Reference makes the Cache memory systems work
- Cache is a fast small capacity memory that should hold those information which are most likely to be accessed



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 μs, Tm = 1.2μs, h = 0.85

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

MEMORY AND CACHE MAPPING - ASSOCIATIVE MAPPLING -

Mapping Function

Specification of correspondence between main memory blocks and cache blocks

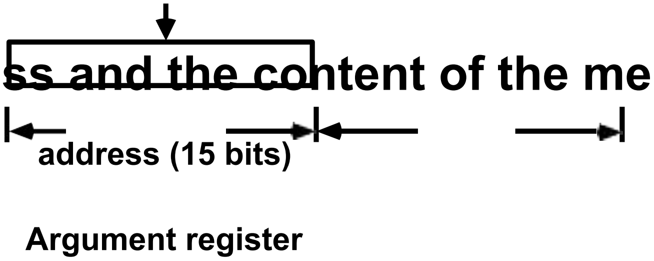
Associative mapping

Direct mapping

Set-associative mapping

Associative Mapping

- Any block location in Cache can store any block in memory
  -> Most flexible
- Mapping Table is implemented in an associative memory
  -> Fast, very Expensive
- Mapping Table
  Stores both address and the content of the memory word



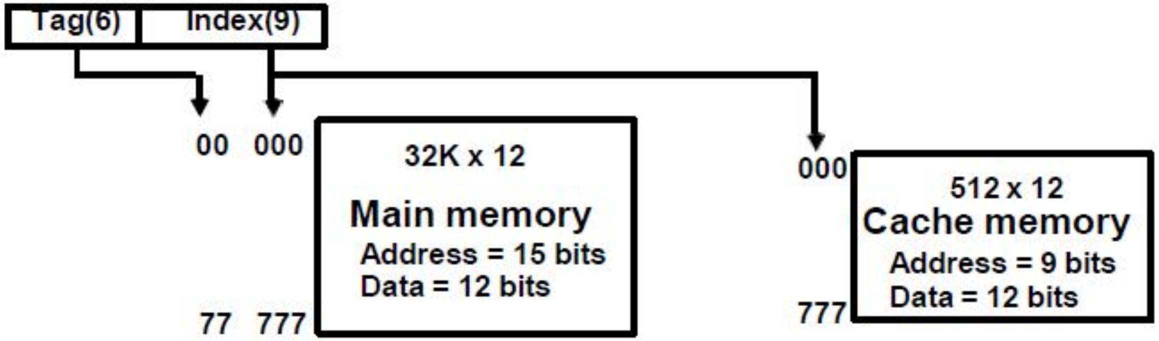
CAM

Address	Data
01000	3450
02777	6710
22235	1234

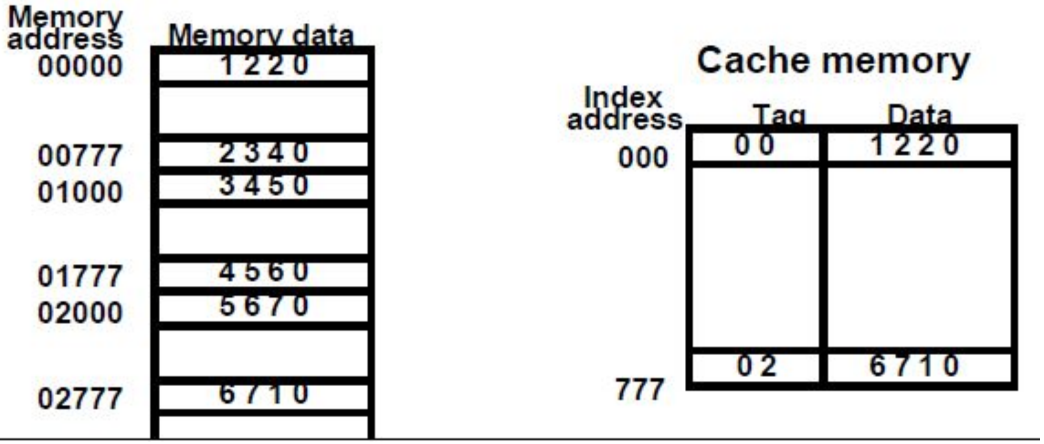
MEMORY AND CACHE MAPPING-DIRECT MAPPING

-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' 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



Memory address	Memory data
00000	1 2 2 0
00777	2 3 4 0
01000	3 4 5 0
01777	4 5 6 0
02000	5 6 7 0
02777	6 7 1 0

Cache memory

Index address	Tag	Data
000	0 0	1 2 2 0
777	0 2	6 7 1 0

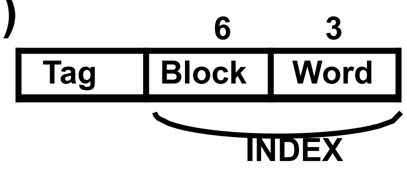
DIRECT MAPPING

Operation

- CPU generates a memory request with (TAG;INDEX)
- Access Cache using INDEX ; (tag; data) Compare TAG and tag
- If matches -> Hit
  Provide Cache[INDEX](data) to CPU
- If not match -> Miss
  M[tag;INDEX] <- Cache[INDEX](data)
  Cache[INDEX] <- (TAG;M[TAG; INDEX])
  CPU <- Cache[INDEX](data)

Direct Mapping with block Size of 8 words

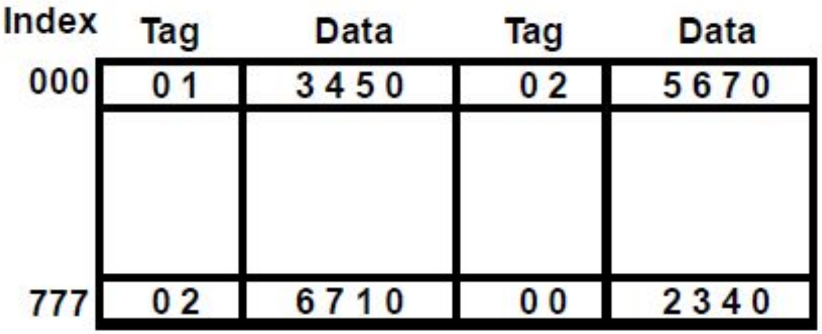
	Index	tag	data
Block 0	000 007	0 1 0 1	3 4 5 0 6 5 7 8
Block 1	010 017		
Block 63	770 777	0 2 0 2	6 7 1 0



MEMORY AND CACHEMAPPING - SET ASSOCIATIVE MAPPING -

- Each memory block has a set of locations in the Cache to load

Set Associative



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

Operation

- CPU generates a memory address(TAG; INDEX)
- Access Cache with INDEX, (Cache word = (tag 0, data 0); (tag 1, data 1))
- Compare TAG and tag 0 and then tag 1
- If tag i = TAG -> Hit, CPU <- data i
- If tag i ≠ TAG -> Miss,

Replace either (tag 0, data 0) or (tag 1, data 1),
Assume (tag 0, data 0) is selected for replacement, (Why (tag 0, data 0) instead of (tag 1, data 1) ?)
M[tag 0, INDEX] <- Cache[INDEX](data 0)

BLOCK REPLACEMENT POLICY

Many different block replacement policies are available

LRU(Least Recently Used) is most easy to implement

Cache word = (tag 0, data 0, *U0*);(tag 1, data 1, *U1*), Ui = 0 or 1(binary)

Implementation of LRU in the Set Associative Mapping with set size = 2

Modifications

```
Initially all U0 = U1 = 1
When Hit to (tag 0, data 0, U0), U1 <- 1(least recently used)
(When Hit to (tag 1, data 1, U1), U0 <- 1(least recently used))
When Miss, find the least recently used one(Ui=1)
    If U0 = 1, and U1 = 0, then replace (tag 0, data 0)
        M[tag 0, INDEX] <- Cache[INDEX](data 0)
        Cache[INDEX](tag 0, data 0, U0) <- (TAG,M[TAG,INDEX], 0); U1 <- 1
    If U0 = 0, and U1 = 1, then replace (tag 1, data 1)
        Similar to above; U0 <- 1
    If U0 = U1 = 0, this condition does not exist
    If U0 = U1 = 1, Both of them are
```

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

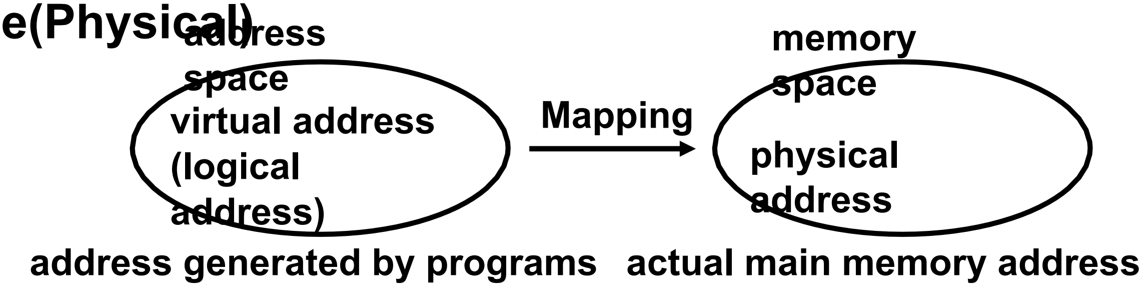
For a read miss, candidate block must be
written back to the memory

Memory is not up-to-date, i.e., the same item in

VIRTUAL MEMORY

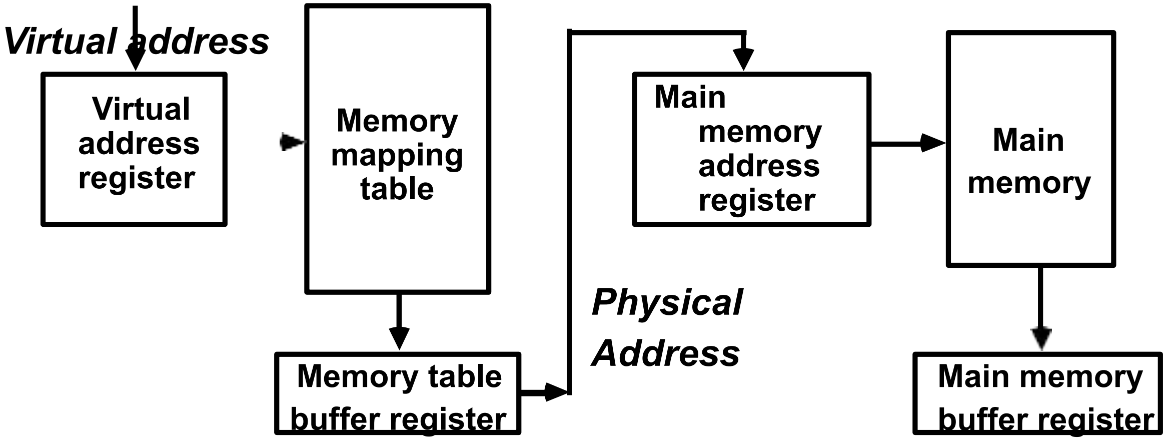
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)


address space
virtual address (logical address)
Mapping
memory space
physical address
address generated by programs
actual main memory address

Address Mapping

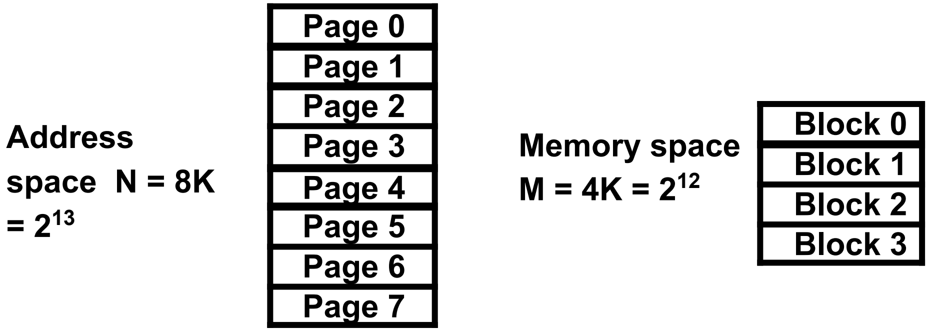
Memory *Mapping Table* for *Virtual Address -> Physical Address*



ADDRESS MAPPING

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

1K words group



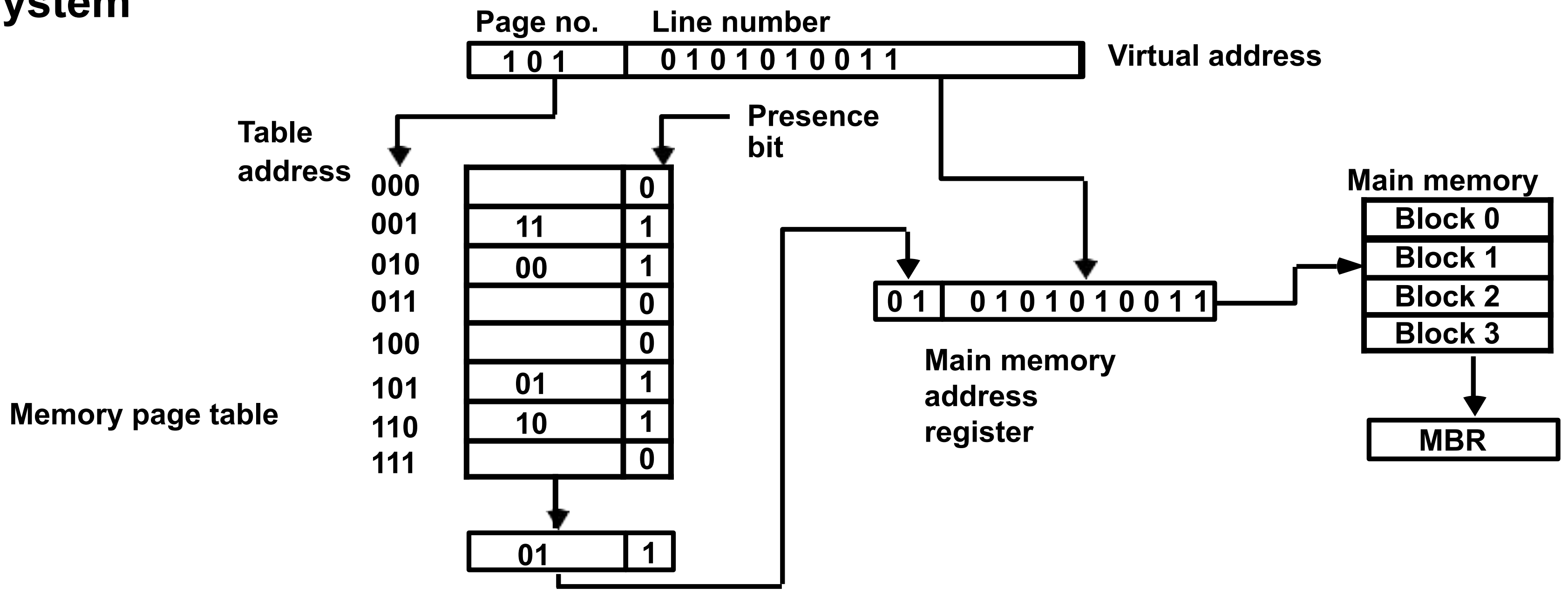
Address space N = 8K = 2^{13}

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Memory space M = 4K = 2^{12}

Block 0
Block 1
Block 2
Block 3

Organization of memory Mapping Table in a paged system



Page no. Line number

1 0 1 | 0 1 0 1 0 1 0 0 1 1 Virtual address

Table address

Presence bit

Table address		Presence bit
000		0
001	11	1
010	00	1
011		0
100		0
101	01	1
110	10	1
111		0

Memory page table

01 | 1

0 1 | 0 1 0 1 0 1 0 0 1 1

Main memory address register

Main memory

Block 0
Block 1
Block 2
Block 3

MBR

ASSOCIATIVE MEMORYPAGE 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

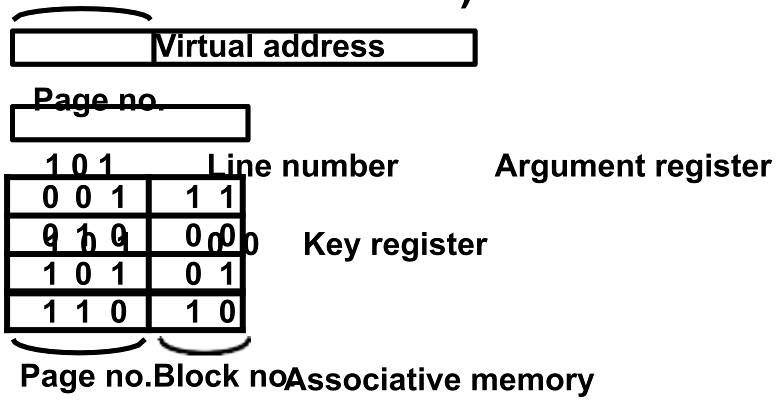
Inefficient storage space utilization

<- 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)



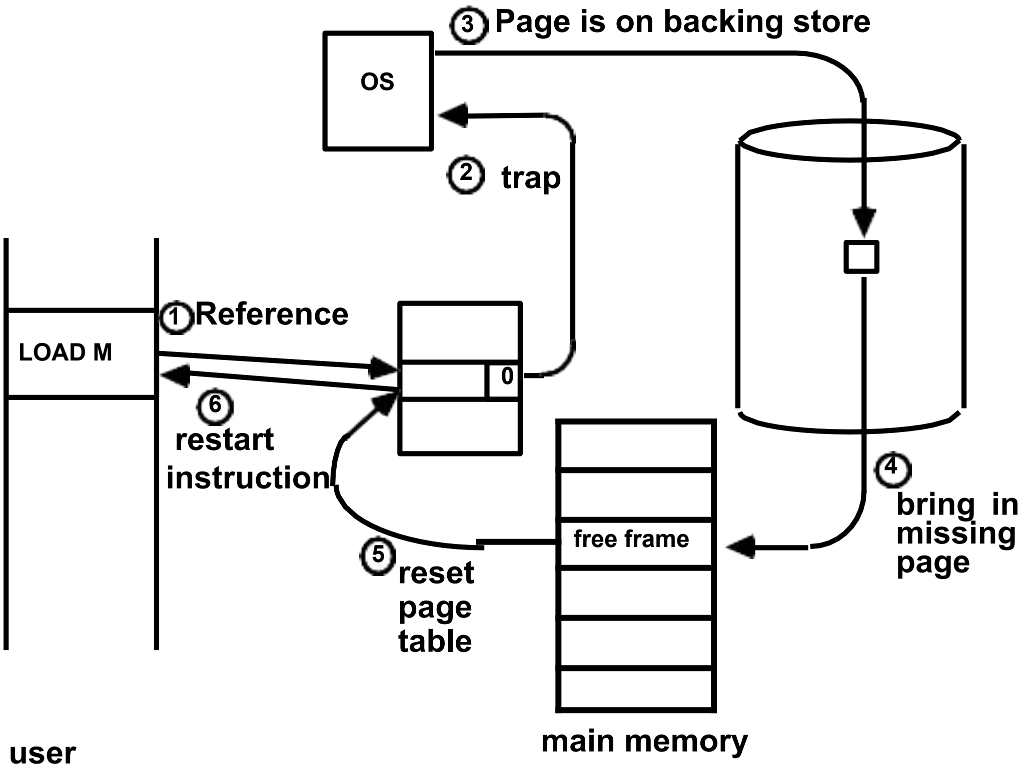
0 0 1	1 1
0 1 0	0 0
1 0 1	0 1
1 1 0	1 0

Page Fault

Page number cannot be found in the Page Table

PAGE FAULT

1. Trap to the OS
2. Save the user registers and program state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the backing store(disk)
5. Issue a read from the backing store to a free frame
   a. Wait in a queue for this device until serviced
   b. Wait for the device seek and/or latency time
   c. Begin the transfer of the page to a free frame
6. While waiting, the CPU may be allocated to some other process
7. Interrupt from the backing store (I/O completed)
8. Save the registers and program state for the other user
9. Determine that the interrupt was from the backing store
10. Correct the page tables (the desired page is now in memory)
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, program state, and new page table, then resume the interrupted instruction.



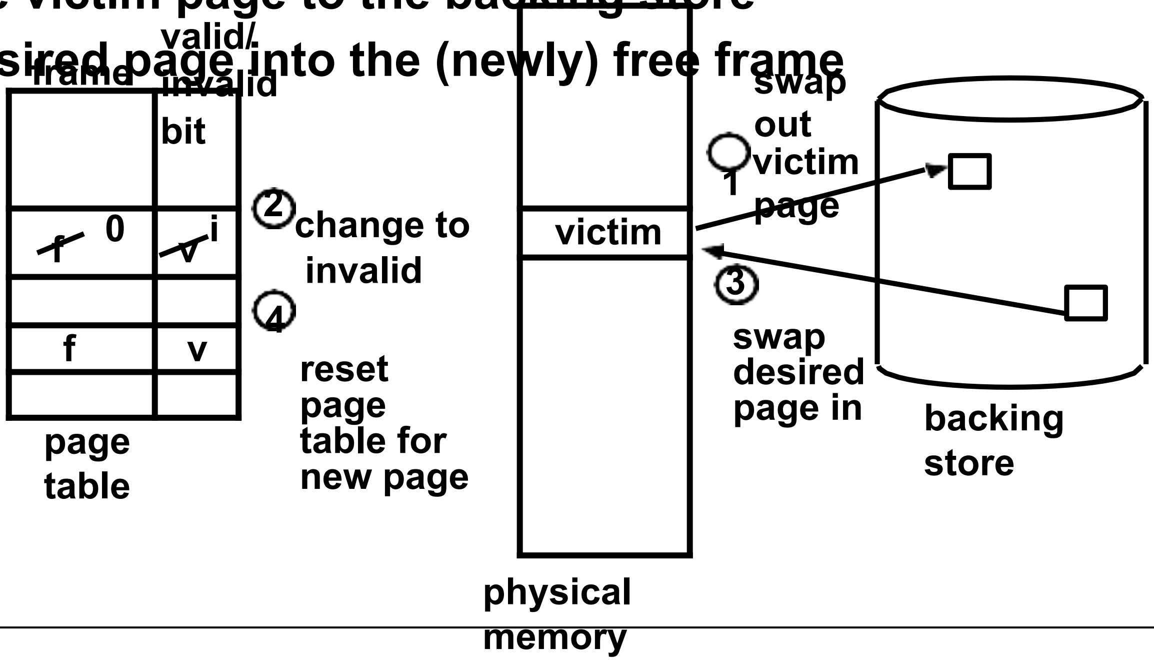
Processor architecture should provide the ability
to restart any instruction after a page fault.

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

Reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0

1 7 0 1

7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

Page frames

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

Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

Replace that page which will not be used for the longest period of time

Reference string

7 0 1 2 0 3 0 4 0 3 2 1 2 1 7 0

2 3 0 1

7	7	7	2	2	2	2	2	7
	0	0	0	0	4	0	0	0
		1	1	3	3	3	1	1

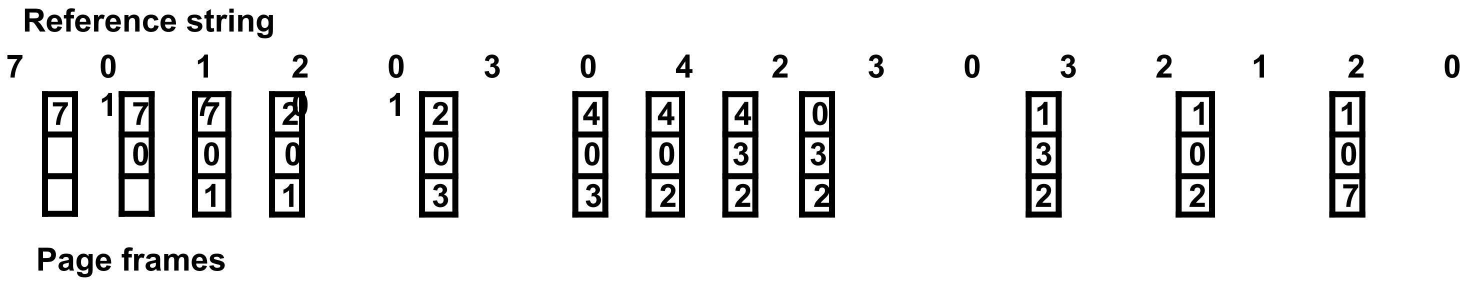
Page frames

PAGE REPLACEMENT ALGORITHMS

LRU

- OPT is difficult to implement since it requires future knowledge
- 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



Reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0

1 7 0 1

7	7	7	2	2	4	4	4	0	1	1	1
	0	0	0	0	0	0	3	3	3	0	0
		1	1	3	3	2	2	2	2	2	7

Page frames

- LRU may require substantial hardware assistance
- The problem is to determine an order for the frames defined by the time of last use

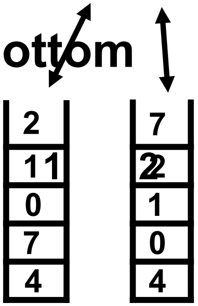
PAGE REPLACEMENT ALGORITHMS

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

Reference string

4 7 0 7 1 0 1 2 1 2 7 1 2



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

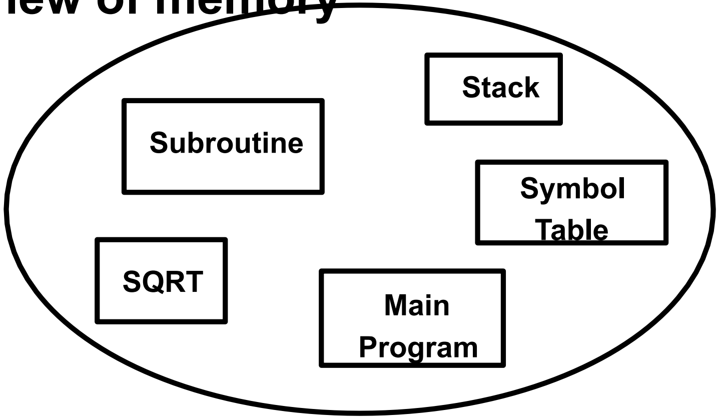
Basic Functions of MM

- *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

User's view of memory



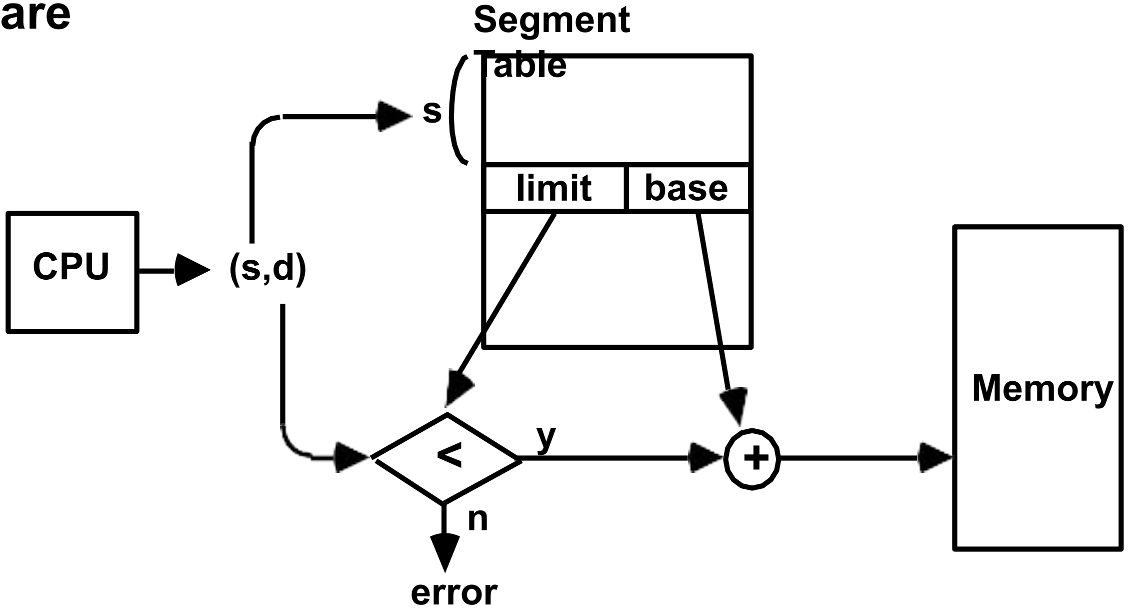
User's view of a program

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.

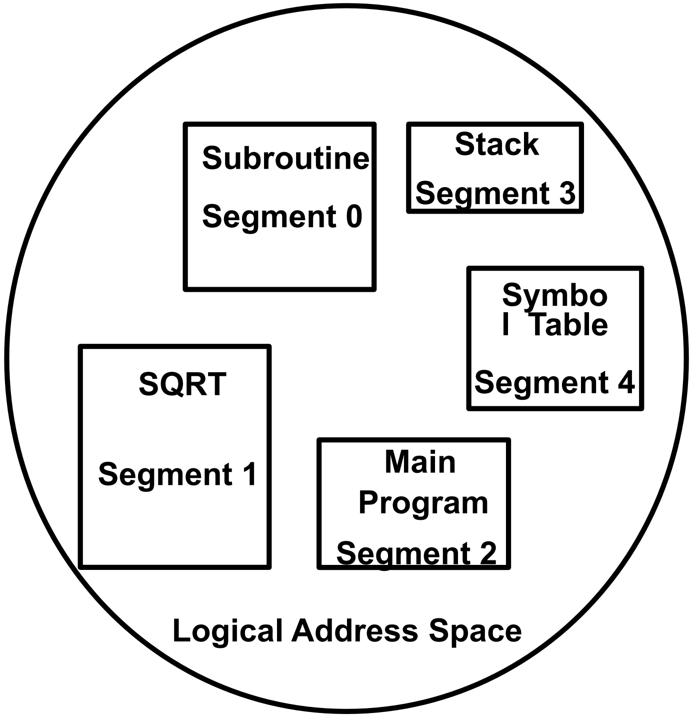
SEGMENTATION

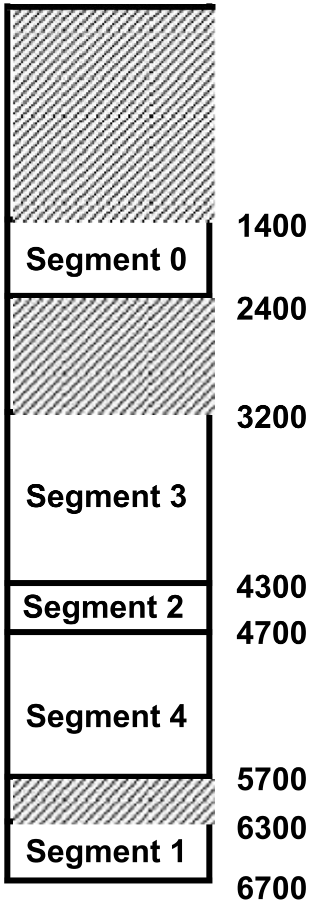
- A memory management scheme which supports user's view of memory
- A logical address space is a collection of segments
- Each segment has a name and a length
- Address specify both the segment name and the offset within the segment.
- For simplicity of implementations, segments are numbered.

Segmentation Hardware



SEGMENTATION EXAMPLE

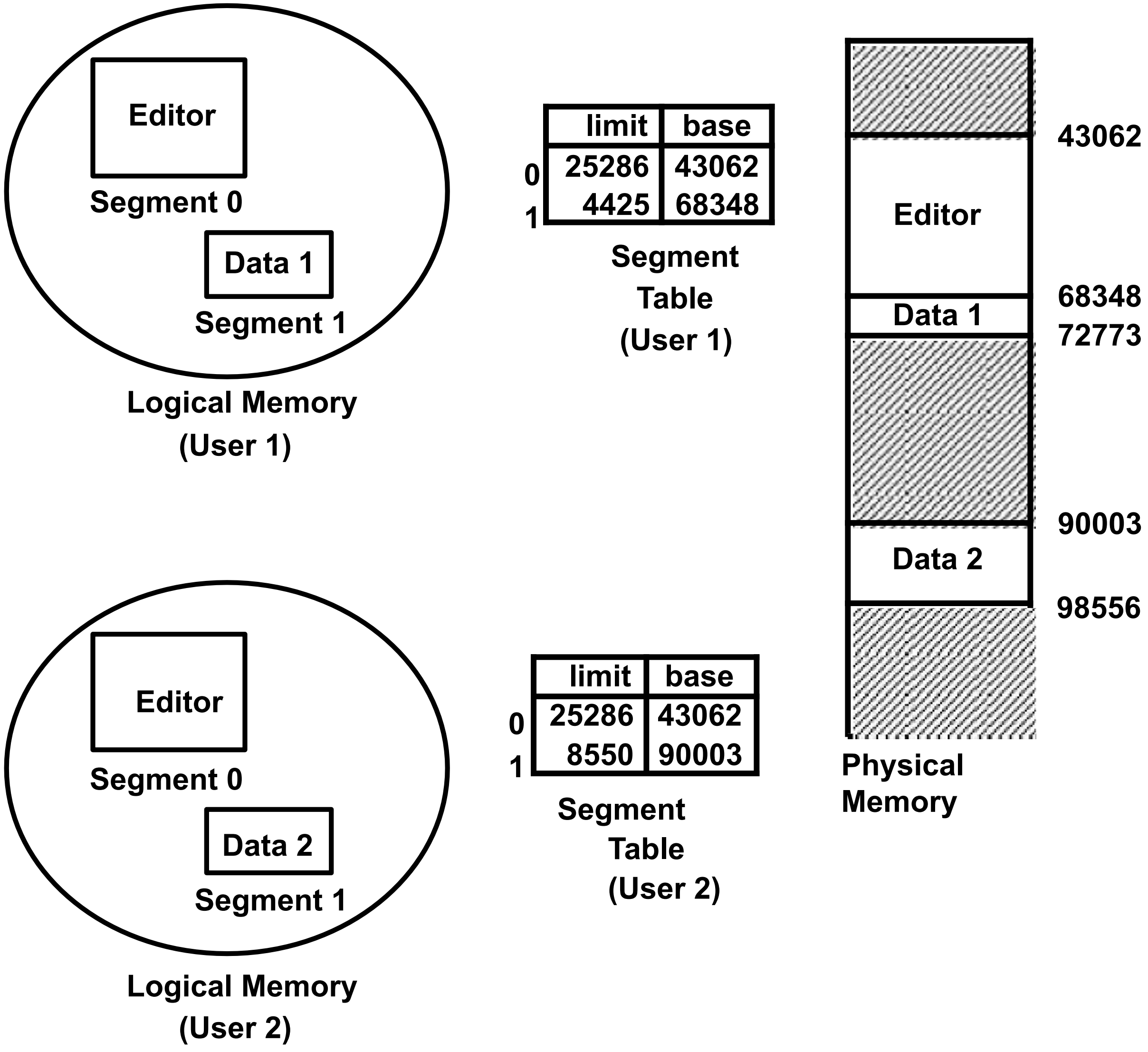




Segment Table

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

SHARING OF SEGMENTS



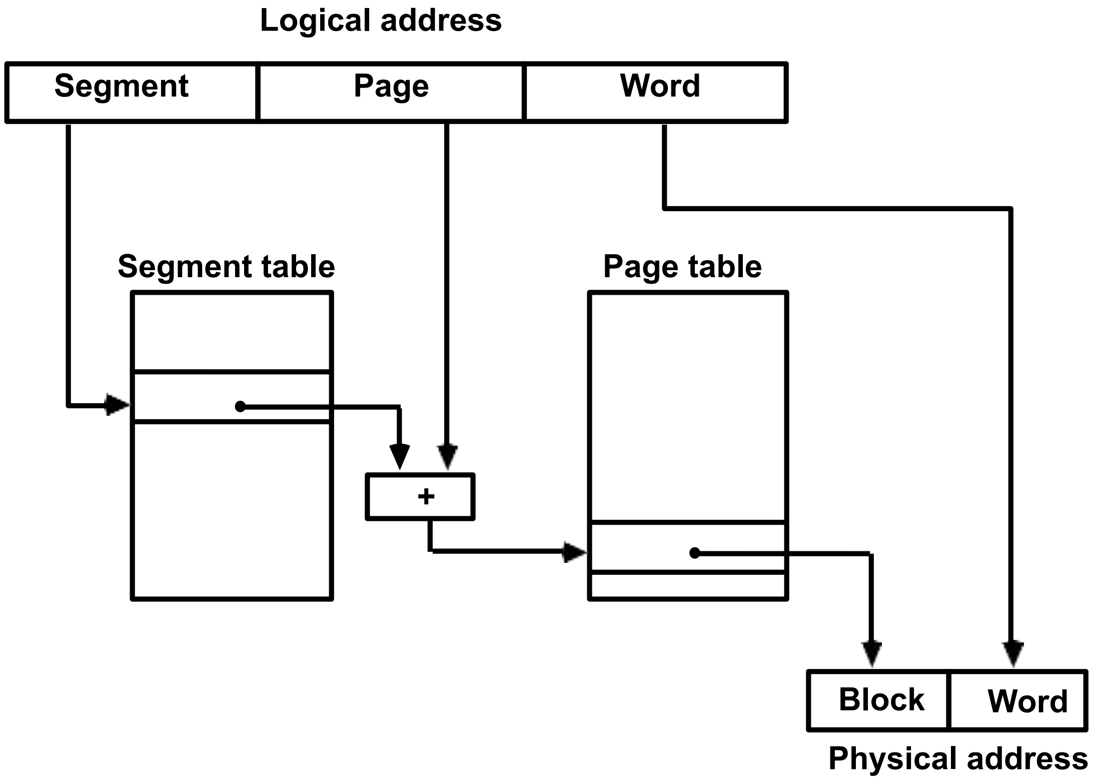
Segment Table (User 1)

	limit	base
0	25286	43062
1	4425	68348

Segment Table (User 2)

	limit	base
0	25286	43062
1	8550	90003

SEGMENTED PAGE SYSTEM



IMPLEMENTATION OF PAGE AND SEGMENT TABLES

Implementation of the Page Table

- Hardware registers (if the page table is reasonably small)
- Main memory
  - Page Table Base Register(PTBR) points to PT
  - Two memory accesses are needed to access a word; one for the page table, one for the word

- Cache memory (TLB: Translation Lookaside Buffer)
  - To speedup the effective memory access time, a special small memory called associative memory, or cache is used

Implementation of the Segment Table

Similar to the case of the page table

EXAMPLE

Logical and Physical Addresses

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

Segment 4 8	Page 8	Word

2^{20} x 32
Physical
memory

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

12 8

Block Word	

Logical and Physical Memory Address Assignment

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

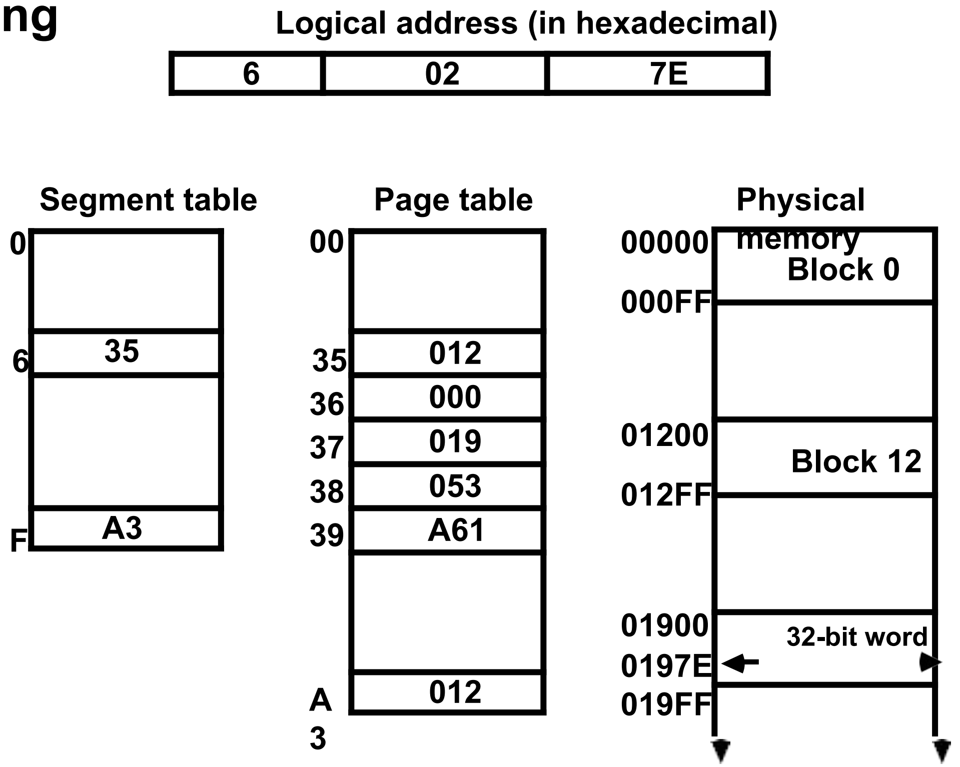
(a) Logical address assignment

Segment Page	Block
600	012
601	000
602	019
603	053
604	A61

(b) Segment-page versus memory block assignment

LOGICAL TO PHYSICAL MEMORY MAPPING

Segment and page table mapping



Logical address (in hexadecimal)

6 | 02 | 7E

Segment table

0

6 35

F A3

Page table

00

35 012

36 000

37 019

38 053

39 A61

A3 012

Physical memory

00000 Block 0

000FF

01200 Block 12

012FF

01900 32-bit word

0197E

019FF

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)

Q1. i.) How many 128 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes?
ii.) How many lines of the address bus must be used to access 2048 byte of memory? How many of these lines will be common to all chips?
iii.) How many lines must be decoded for chip select? Specify the size of the decoders?

i) According to question, 128 x 8 RAM = 1024 bits and 1 bit = 1/8Byte Thus, 1024 bits = 1024 /8 = 128 Byte
1 RAM chip = 128 Byte
Lets consider there are n number of Chips to provide 2048 Bytes
Now, 1RAM Chip = 128 Byte
128n = 2048
n = 16 Therefore, 16 Chips will be needed to provide memory capacity of 2048 bytes.

ii) 2048 (2^{11}) locations needs same address lines as one need bits for representing 2048
11 lines of address bus required to access 2048 bytes of memory.
Here chips are of 128x8 in size,
128 (2^{7}) locations means we need 7 address lines. Therefore 11 lines of address bus required to access 2048 bytes of memory. And 7 address lines will be common to all chips.

iii) 4 address lines must be decoded to select the right chip. 4 address lines will need 16 chips, Size of decoder = 4x16.

Q2. A computer uses RAM chips of 1024 x 1 capacity.

a) How many chips are needed and how should their address lines be connected to provide a memory capacity of 1024 bytes?

b) How many chips are needed to provide a memory capacity of 16K bytes?

c) Explain in words how the chips are to be connected to the address bus.

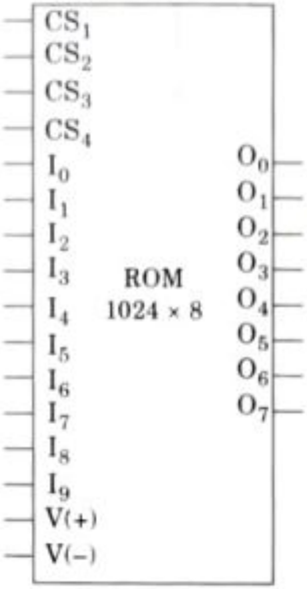
(i) As given available chips = 1024 x 1 capacity and Required capacity = 1024 x 8 capacity
Number of Chips=(1024X8)/(1024X1) = 8

(ii) Number Of Chips Required = (16X1024X8)/(1024X1) = 128

(iii) Use 14 address lines (16 k = 2^{14})
10 lines specify the chip address
4 lines are decoded into 16 chip-select inputs.

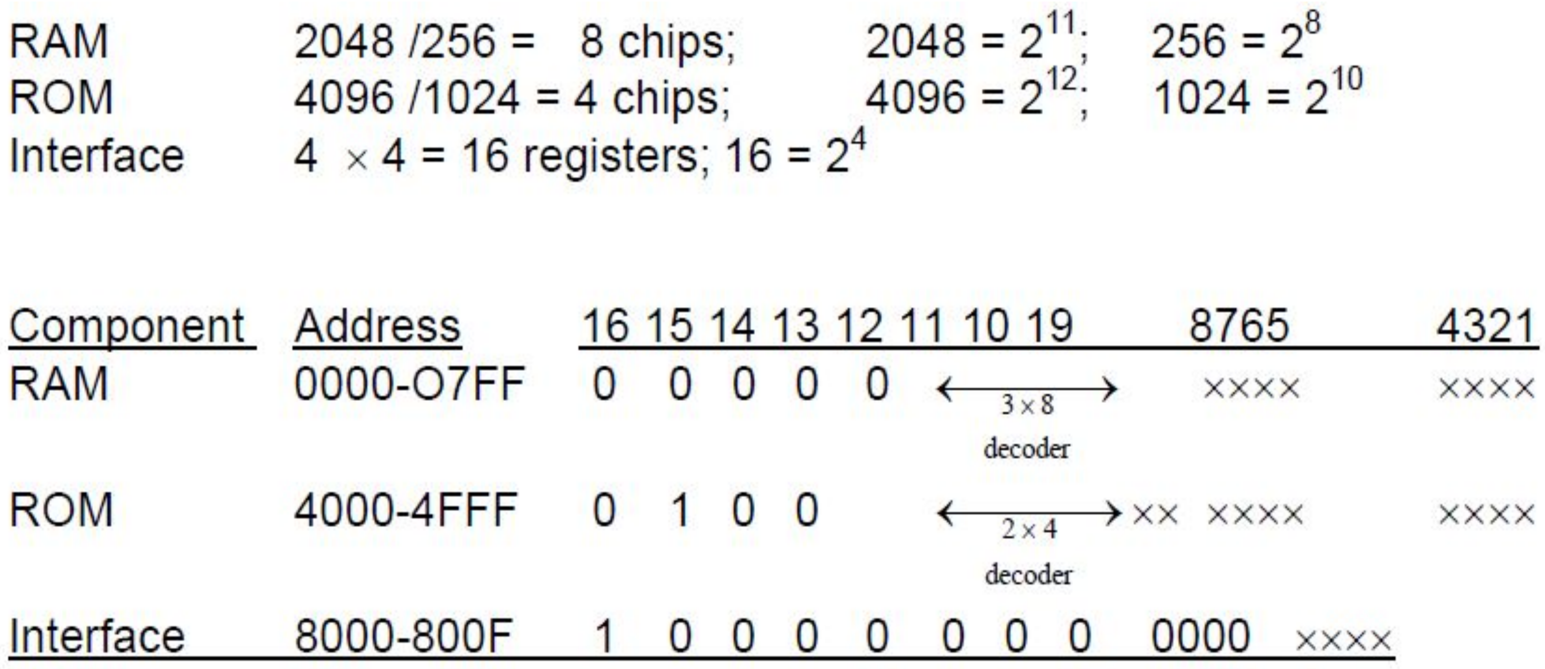
Q3. A ROM chip of 1024*8 has four select inputs and operates from a 5 volt power supply. How many pins are needed for the IC package ? Draw a block diagram and label all input and output terminals in the ROM.

Size of ROM Chip = 1024 x 8
Number of input = 10 pin [2^{10} = 1024]
Number of output = 8 pin
Number of chip select = 4 pin
Power = 2 pin
Total 24 pins are required.



Q5. A computer employs RAM chips of 256 x 8 and ROM chips of 1024 x 8. the computer system needs 2k bytes of RAM, 4k bytes of ROM, and four interface units, each with four registers. a memory-mapped I/O configuration is used. the two highest-order bits of the address bus are assigned 00 for RAM, 01 for ROM, and 10 for interface registers.

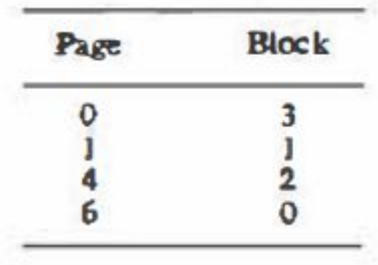
a. How many ram and ROM chips are needed?
b. Draw a memory-address map for the system.
c. Give the address range in hexadecimal for RAM, ROM, and interface.



RAM	2048 /256 = 8 chips;	$2048 = 2^{11}$;	$256 = 2^{8}$
ROM	4096 /1024 = 4 chips;	$4096 = 2^{12}$;	$1024 = 2^{10}$
Interface	$4 \times 4 = 16$ registers; $16 = 2^{4}$		

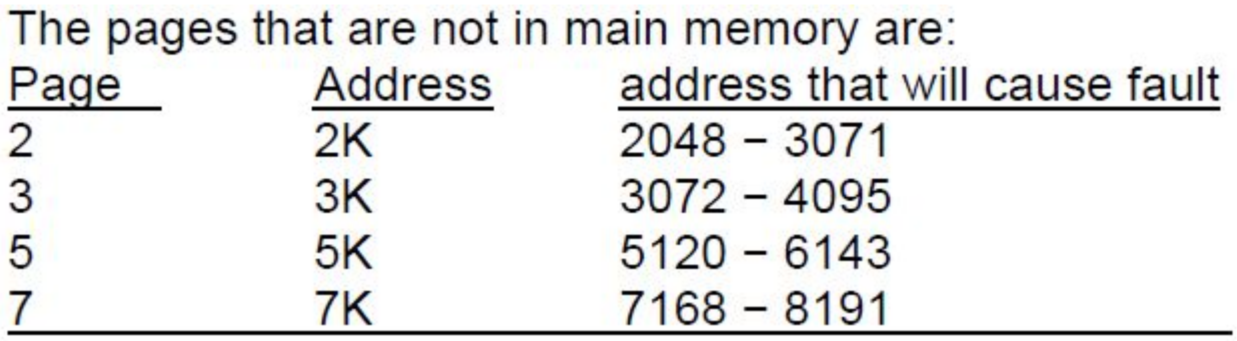
Component	Address	16 15 14 13 12 11 10 19	8765	4321
RAM	0000-O7FF	0 0 0 0 0 $\underset{\text{decoder}}{\underset{3\times 8}{\longleftrightarrow}}$	xxxx	xxxx
ROM	4000-4FFF	0 1 0 0 $\underset{\text{decoder}}{\underset{2\times 4}{\longleftrightarrow}}$ xx	xxxx	xxxx
Interface	8000-800F	1 0 0 0 0 0 0 0	0000	xxxx

Q20. A virtual memory has a page size of 1K words. There are eight pages and four blocks. The associative memory page table contains the following entries:



Page	Block
0	3
1	1
4	2
6	0

Make a list of all virtual addresses (In decimal) that will cause a page fault if used by the CPU.

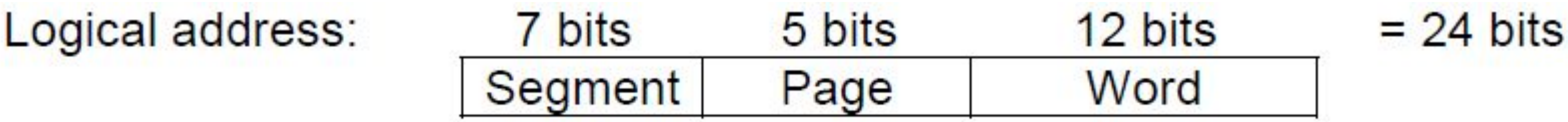


The pages that are not in main memory are:

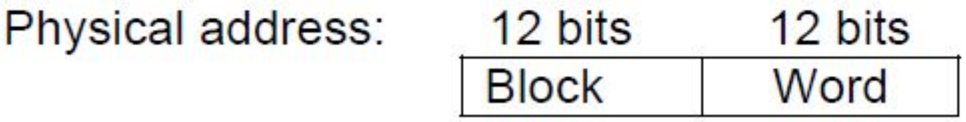
Page	Address	address that will cause fault
2	2K	2048 - 3071
3	3K	3072 - 4095
5	5K	5120 - 6143
7	7K	7168 - 8191

Q23. The logical address space in a computer system consists of 128 segments. Each segment can have up to 32 pages of 4K words in each physical memory consists of 4K blocks of 4K words in each. Formulate the logical and physical address formats.

Logical address space has 128 segments x 32 pages x 4 K words
i.e., $128 \text{ X } 32 \text{ X } 4 \text{ X } 2^{10}$
$= 2^7 \text{ x } 2^5 \text{ x } 2^2 \text{ x } 2^{10} = 2^{24} = 24$ bit
Physical memory has address space of 4 K blocks x 4 K words
$= 4 \text{ X } 2^{10} \text{ X } 4 \text{ X } 2^{10}$
$= 2^{24} = 24$ bits



Logical address:	7 bits	5 bits	12 bits	= 24 bits
	Segment	Page	Word	



Physical address:	12 bits	12 bits
	Block	Word