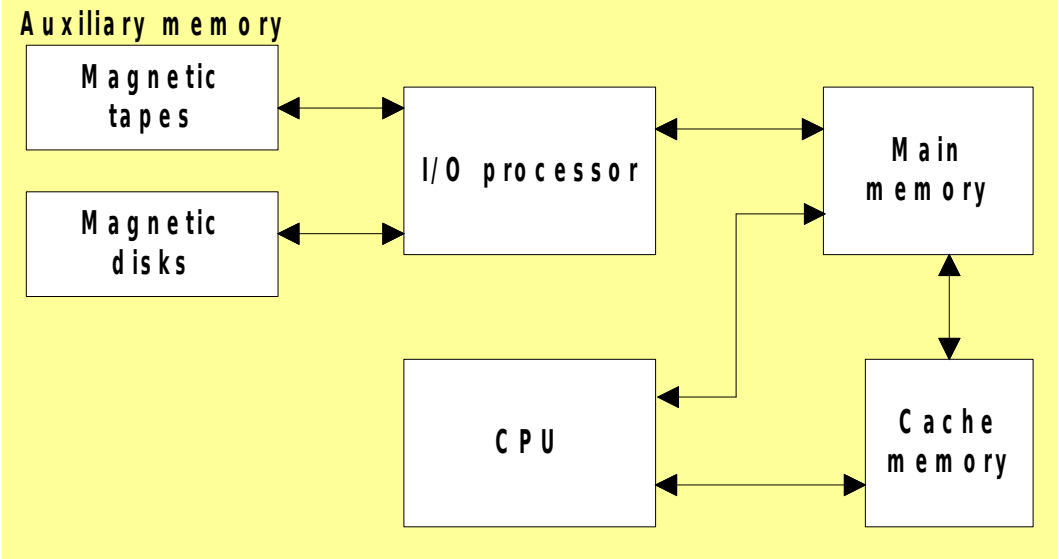
Chap. 6 Memory Organization

6-1 Memory Hierarchy

- Memory hierarchy in a computer system : ***Fig. 12-1***
  - **Main Memory** : memory unit that communicates directly with the CPU (RAM)
  - **Auxiliary Memory** : device that provide backup storage (Disk Drives)
  - **Cache Memory** : special very-high-speed memory to increase the processing speed (Cache RAM)

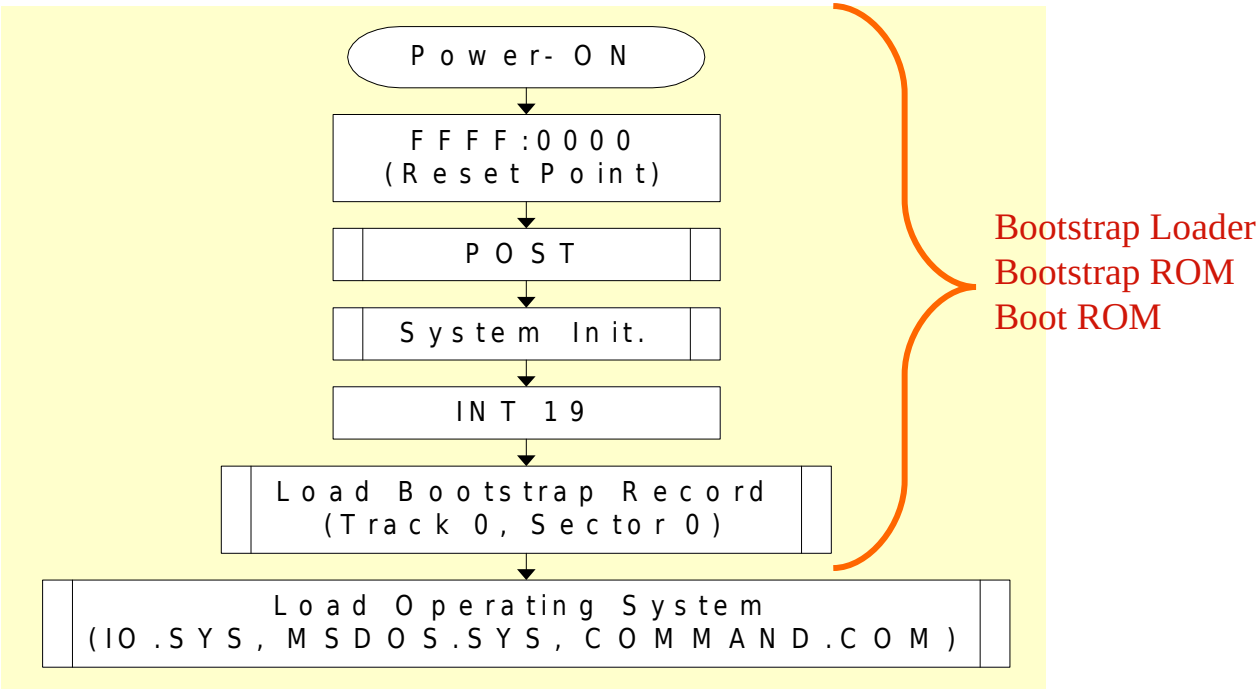


- Multiprogramming
  - enable the CPU to process a number of independent program concurrently
- Memory Management System : sec. 6-5
  - supervise the flow of information between auxiliary memory and main memory

6-2 Main Memory

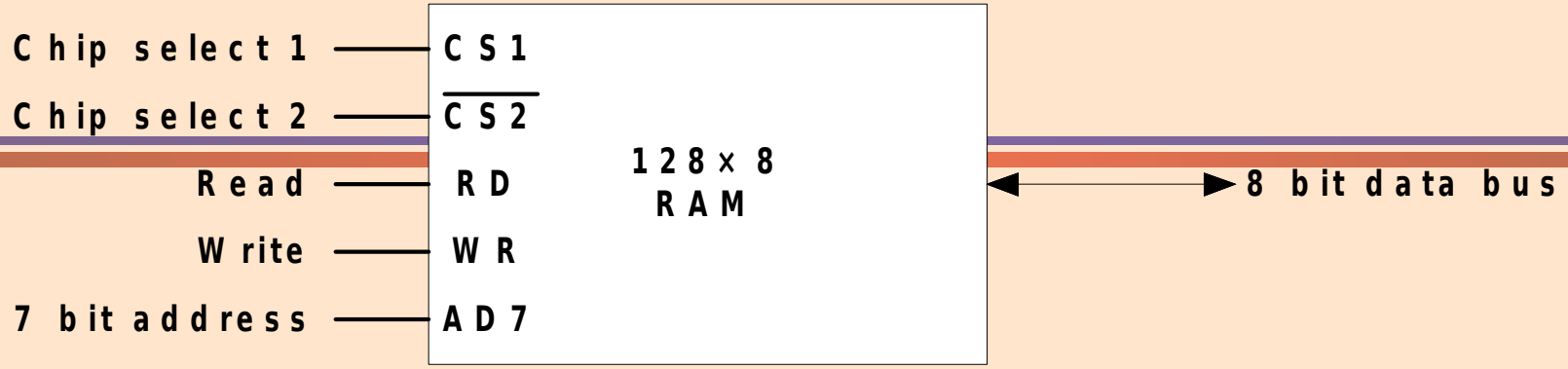
Bootstrap Loader

- A program whose function is to start the computer software operating when power is turned on



RAM and ROM Chips

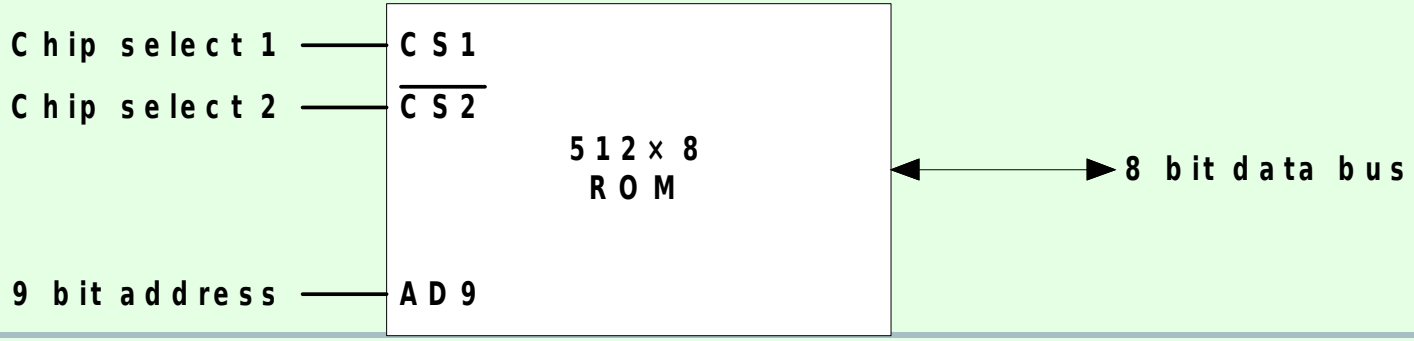
- Typical RAM chip : ***Fig. 12-2***
  - » 128 X 8 RAM : 2^7 = 128 (7 bit address lines)
- Typical ROM chip : ***Fig. 12-3***
  - » 512 X 8 ROM : 2^9 = 512 (9 bit address lines)



(a) Block diagram

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

(b) Function table



Memory Address Map

- Memory Configuration : **512** bytes **RAM** + **512** bytes **ROM**
  - » **1** x **512** byte **ROM** + **4** x **128** bytes RAM
- Memory Address Map : ***Tab. 12-1***
  - » Address line **9 8**
    - RAM 1 **0 0** : 0000 - 007F
    - RAM 2 **0 1** : 0080 - 00FF
    - RAM 3 **1 0** : 0100 - 017F
    - RAM 4 **1 1** : 0180 - 01FF
  - » Address line **10**
    - ROM **1** : 0200 - 03FF
- Memory Connection to CPU : ***Fig. 12-4***
  - » 2 x 4 Decoder : RAM select (**CS1**)
  - » Address line **10**
    - RAM select : $\overline{\mathbf{CS2}}$
    - ROM select : **CS2** **Invert**

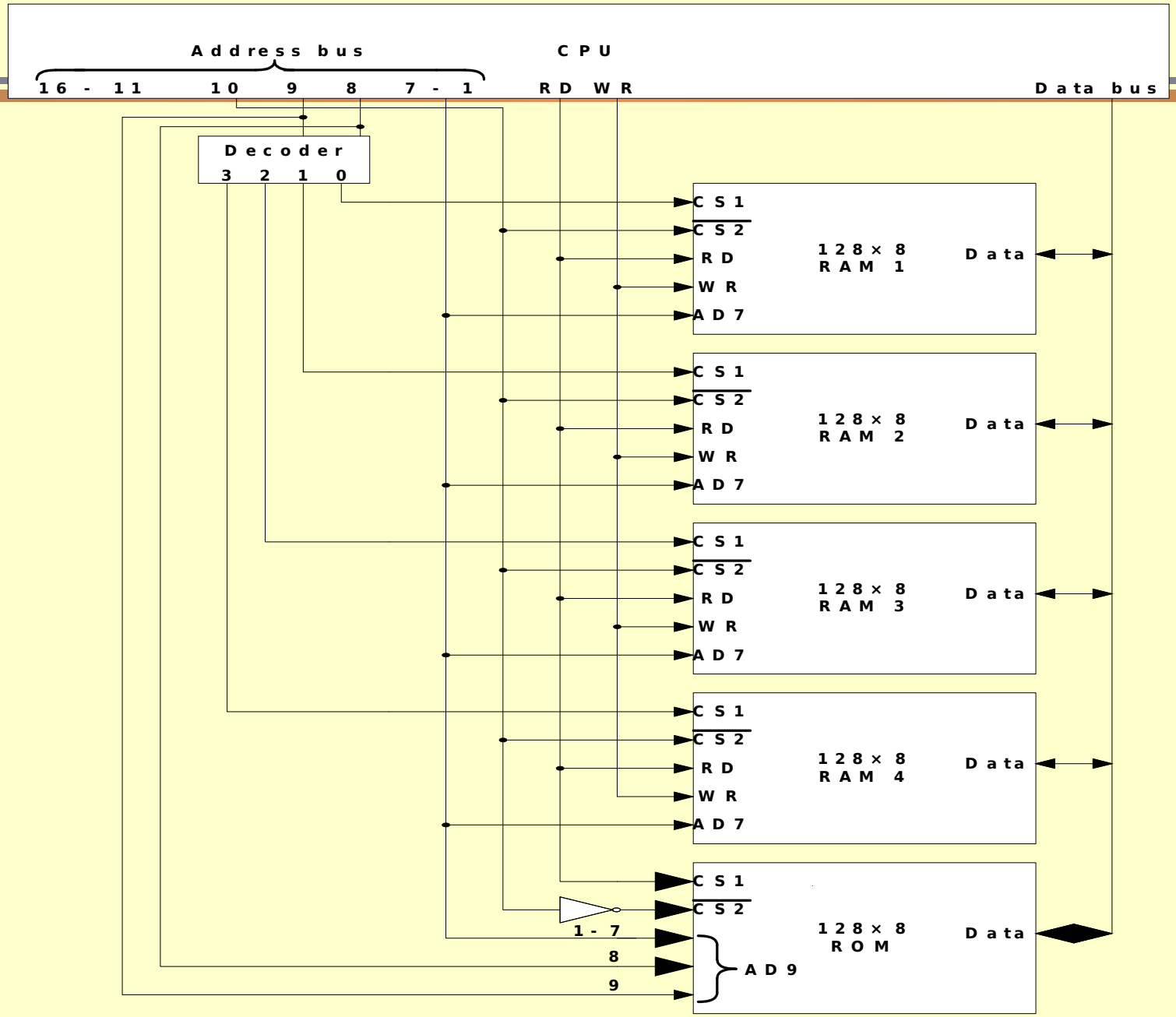

Address bus
16 - 11
10
9
8
7 - 1
CPU
RD WR
Data bus
Decoder
3 2 1 0
CS1
CS2
RD
WR
AD7
128 × 8
RAM 1
Data
CS1
CS2
RD
WR
AD7
128 × 8
RAM 2
Data
CS1
CS2
RD
WR
AD7
128 × 8
RAM 3
Data
CS1
CS2
RD
WR
AD7
128 × 8
RAM 4
Data
CS1
CS2
1 - 7
8
9
AD9
128 × 8
ROM
Data

TABLE 12-1 Memory Address Map for Microprocomputer

Component	Hexadecimal 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

6-3 Cache Memory

- Locality of Reference
  - the references to memory *tend to be confined within a few localized areas* in memory
- Cache Memory : **a fast small memory**
  - keeping the most frequently accessed instructions and data in the fast cache memory
- Cache
  - cache size : usually 256 K byte (or 512 K byte)
  - mapping method : 1) associative, 2) direct, 3) set-associative
  - replace algorithm : 1) LRU, 2) LFU, 3) FIFO
  - write policy : 1) write-through, 2) write-back
- Hit Ratio
  - the ratio of the number of hits divided by the total CPU references (hits + misses) to memory
    - » ***hit*** : the CPU finds the word in the cache
    - » ***miss*** : the word is not found in cache (CPU must read main memory)
  - ***For example,*** cache memory access time = 100 ns, main memory access time = 1000 ns, hit ratio = 0.9

MEMORY AND CACHE MAPPING - ASSOCIATIVE MAPPING -

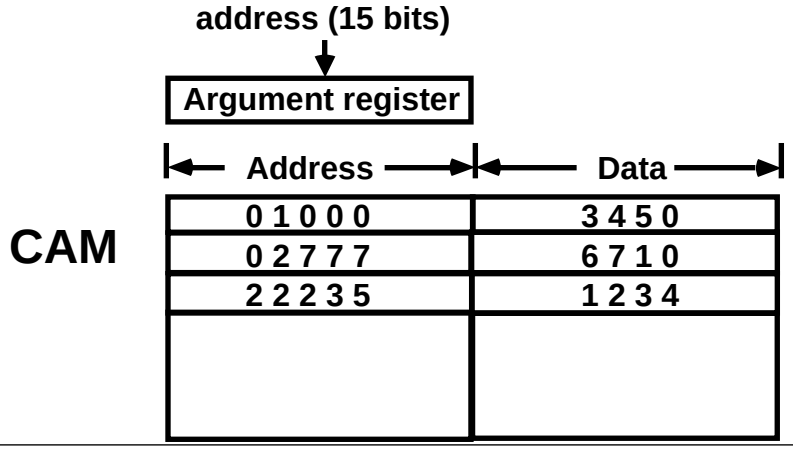
Mapping Function

Specification of correspondence between main memory blocks and cache blocks

Associative mapping
Direct mapping
Set-associative mapping

Associative Mapping

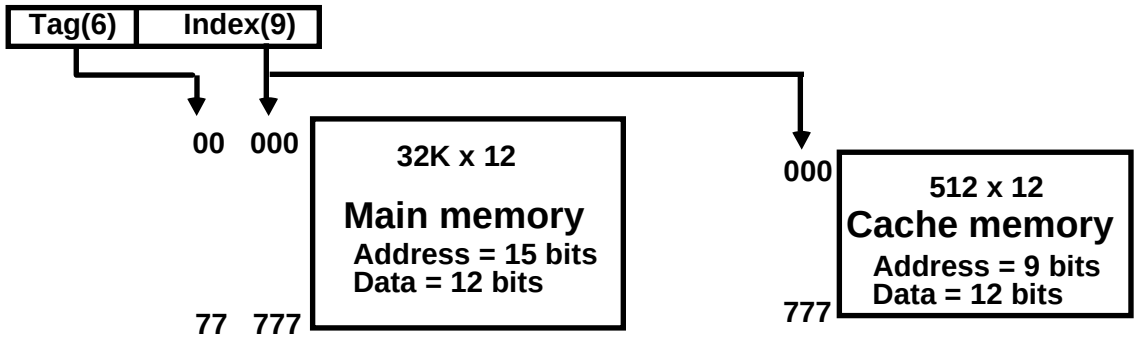
- Any block location in Cache can store any block in memory
  ⇒ Most flexible
- Mapping Table is implemented in an associative memory
  ⇒ Very Expensive
- Mapping Table
  Stores both address and the content of the memory word



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 (Content Addressable Memory)
- 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

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

6	6	3
Tag	Block	Word

INDEX

SET ASSOCIATIVE MAPPING

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

Set Associative Mapping Cache with set size of two

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

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)
  Cache[INDEX](tag 0, data 0) ← (TAG, M[TAG,INDEX]),
  CPU ← 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

Initially all U0 = U1 = 1
When Hit to (tag 0, data 0, U0=0), U1 ← 1(least recently used)
(When Hit to (tag 1, data 1, U1=0), 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=0) ← (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 candidates,**
  - **Take arbitrary selection**

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

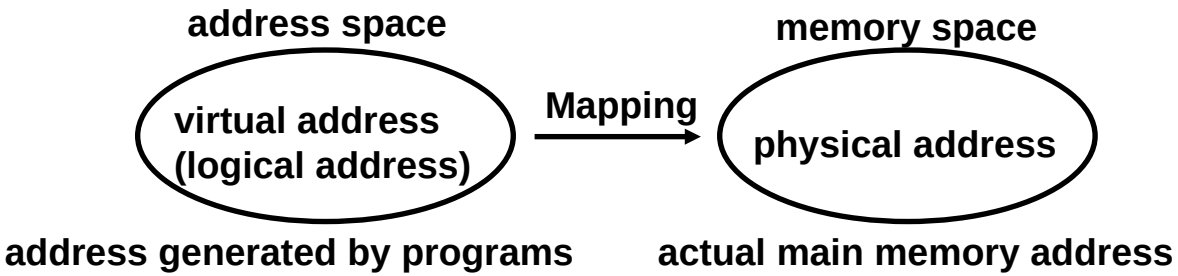
If Hit, only Cache is written
If Miss, missing block is brought to Cache and write into Cache
For a read miss, candidate block must be written back to the memory

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

6.4. VIRTUAL MEMORY

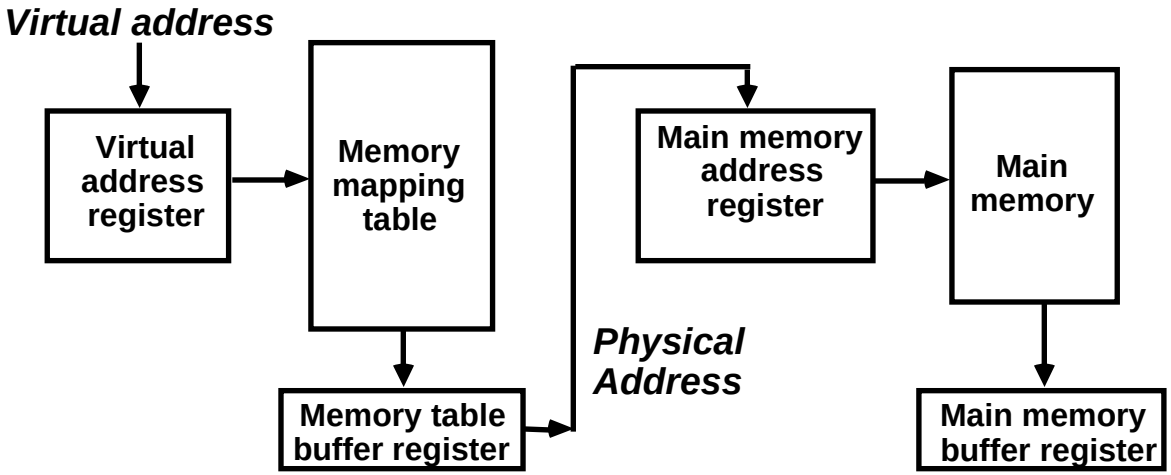
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)



Address Mapping

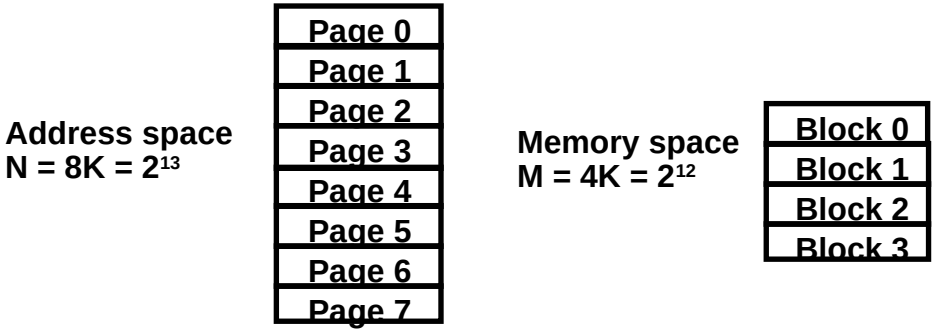
Memory *Mapping Table* for *Virtual Address -> Physical Address*



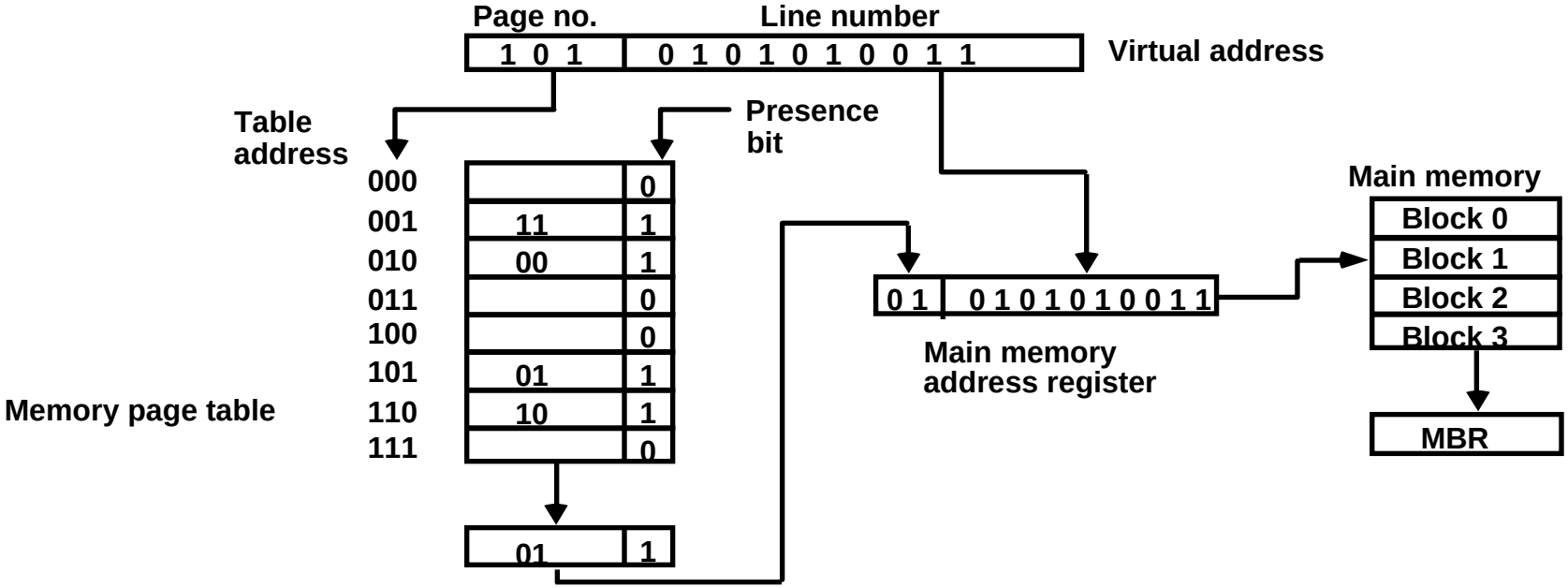
ADDRESS MAPPING

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

1K words group



Organization of memory Mapping Table in a paged system



ASSOCIATIVE MEMORY PAGE TABLE

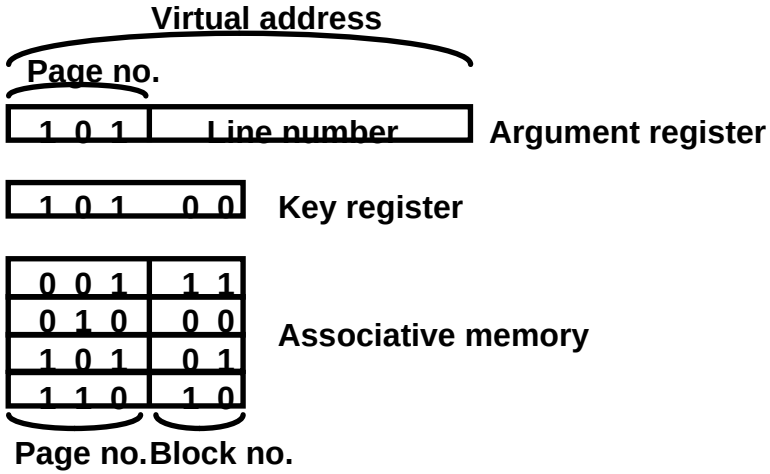
Assume that

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)

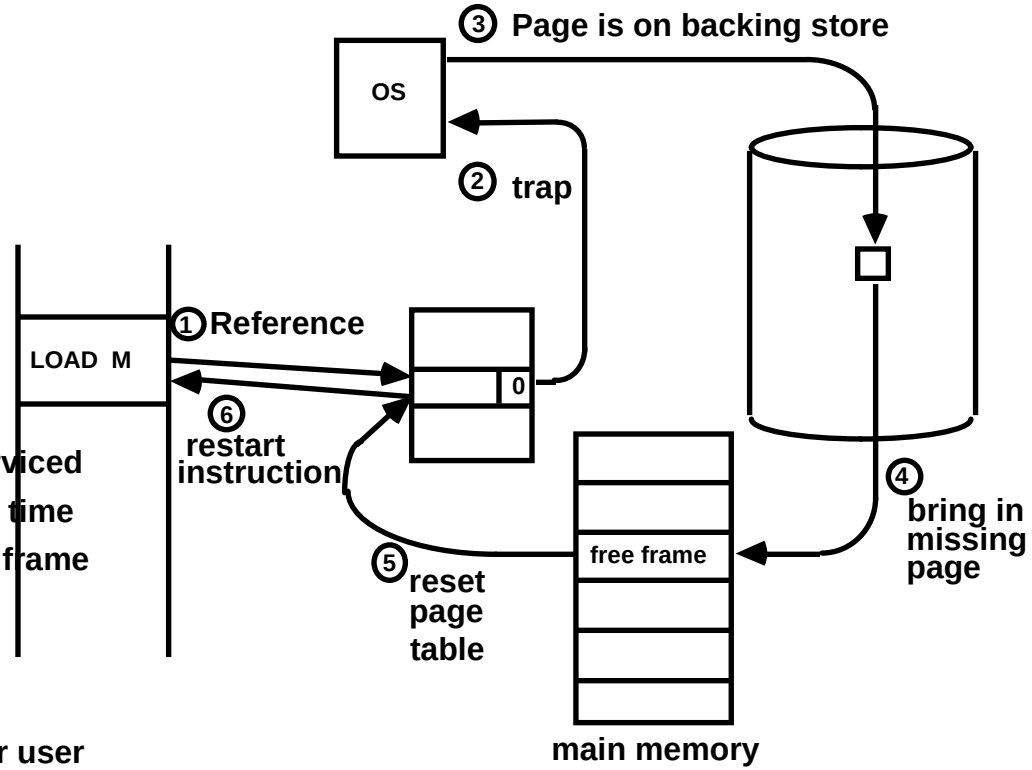


Page Fault

Page number cannot be found in the Page Table

PAGE 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.



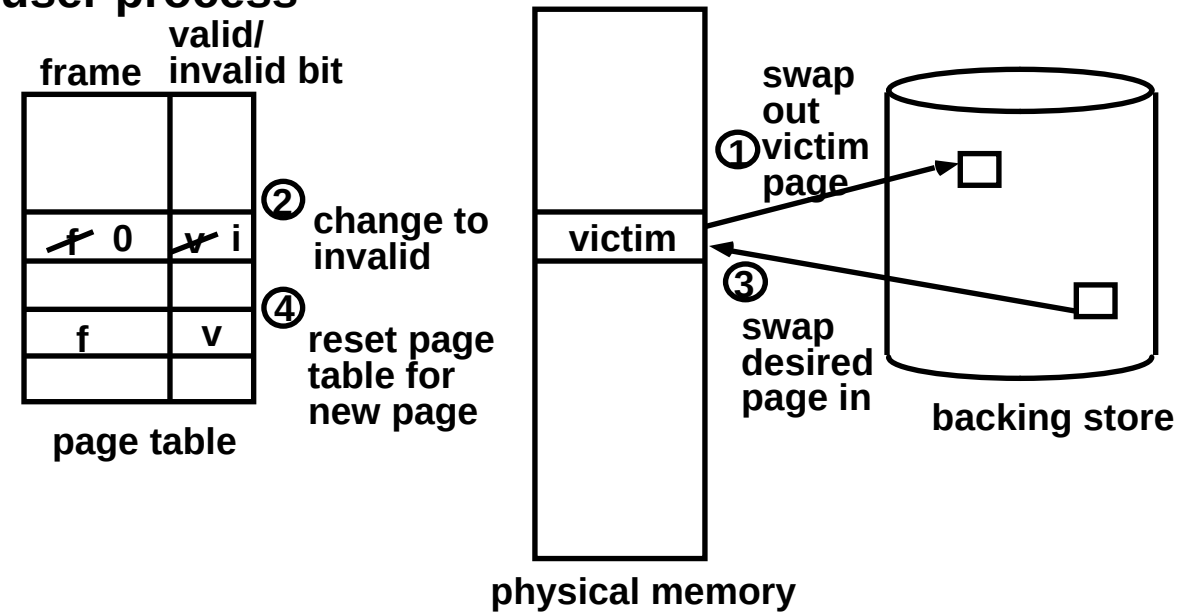
Processor architecture should provide the ability to restart any instruction after a page fault.

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



6.5. MEMORY MANAGEMENT HARDWARE

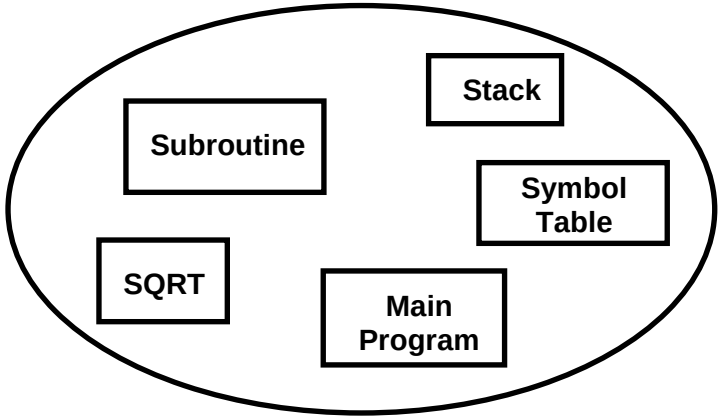
Basic Functions of MM

- *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



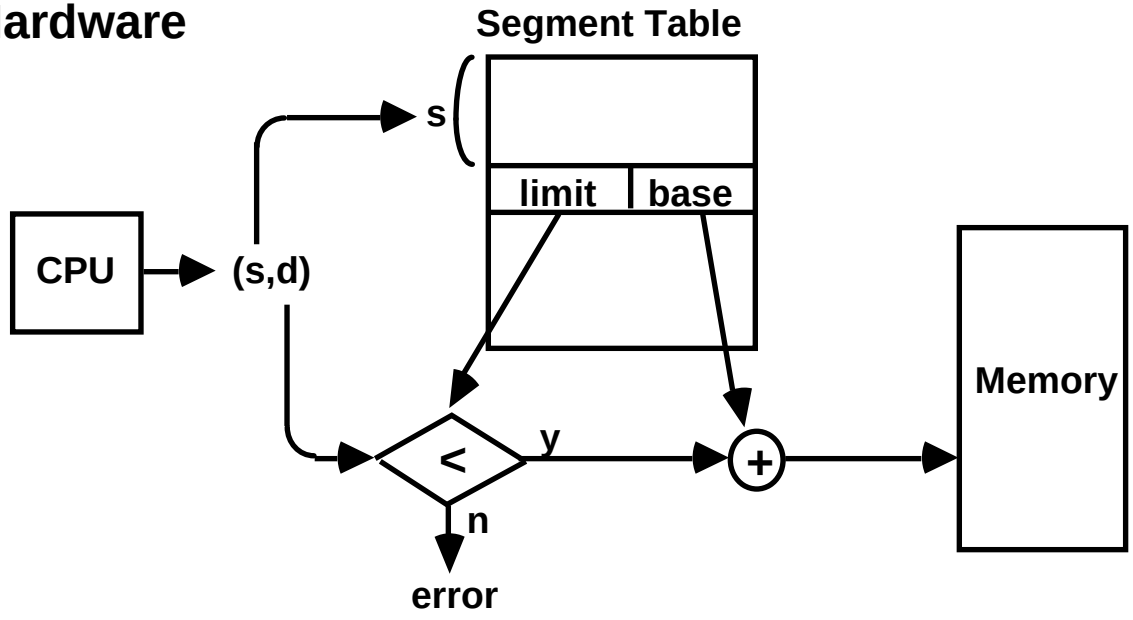
User's view of a program

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



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

4	8	8
Segment	Page	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

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

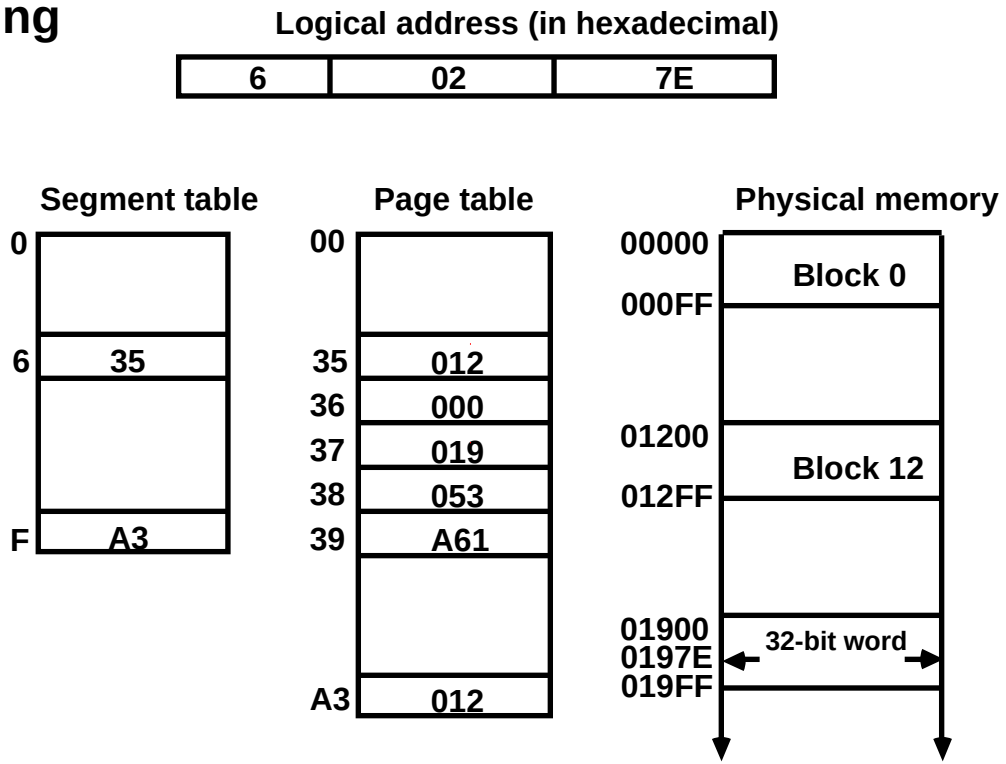
(a) Logical address assignment

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