MEMORY ORGANIZATION

Topics for discussion

- Memory organization
- Memory hierarchy
- Types of memory
- Memory management hardware

MEMORY ORGANIZATION

- Memory hierarchy
- Main memory
- Auxiliary memory
- Associative memory
- Cache memory
  - Storage technologies and trends
  - Locality of reference
  - Caching in the memory hierarchy
- Virtual memory
- Memory management hardware.

RANDOM-ACCESS MEMORY (RAM)

- Key features
  - RAM is packaged as a chip.
  - Basic storage unit is a cell (one bit per cell).
  - Multiple RAM chips form a memory.
- 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.

Cont…

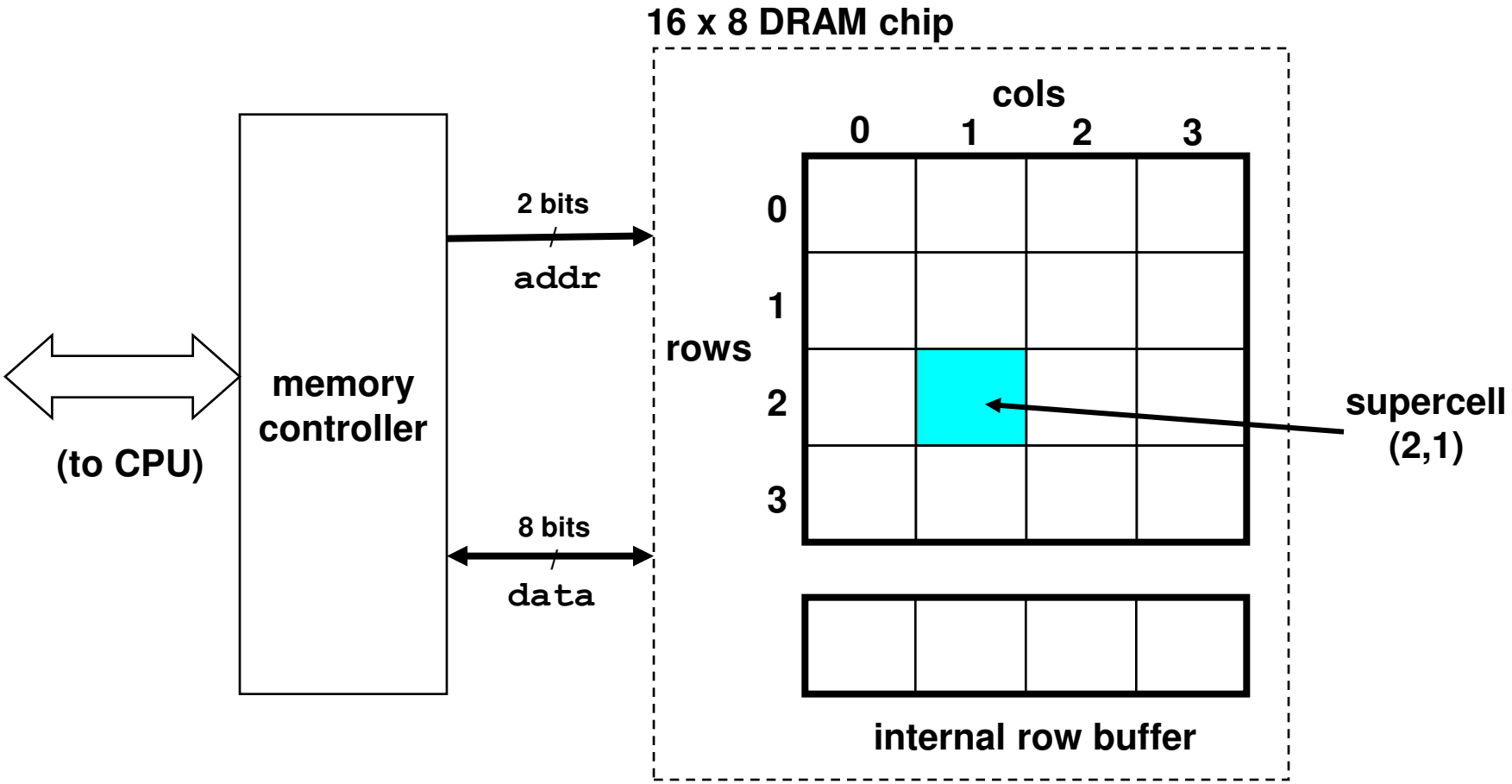
- Dynamic RAM (DRAM)
  - Each cell stores bit with a capacitor and transistor.
  - Value must be refreshed every 10-100 ms.
  - Sensitive to disturbances.
  - Slower and cheaper than SRAM.

SRAM VS DRAM SUMMARY

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

CONVENTIONAL DRAM ORGANIZATION

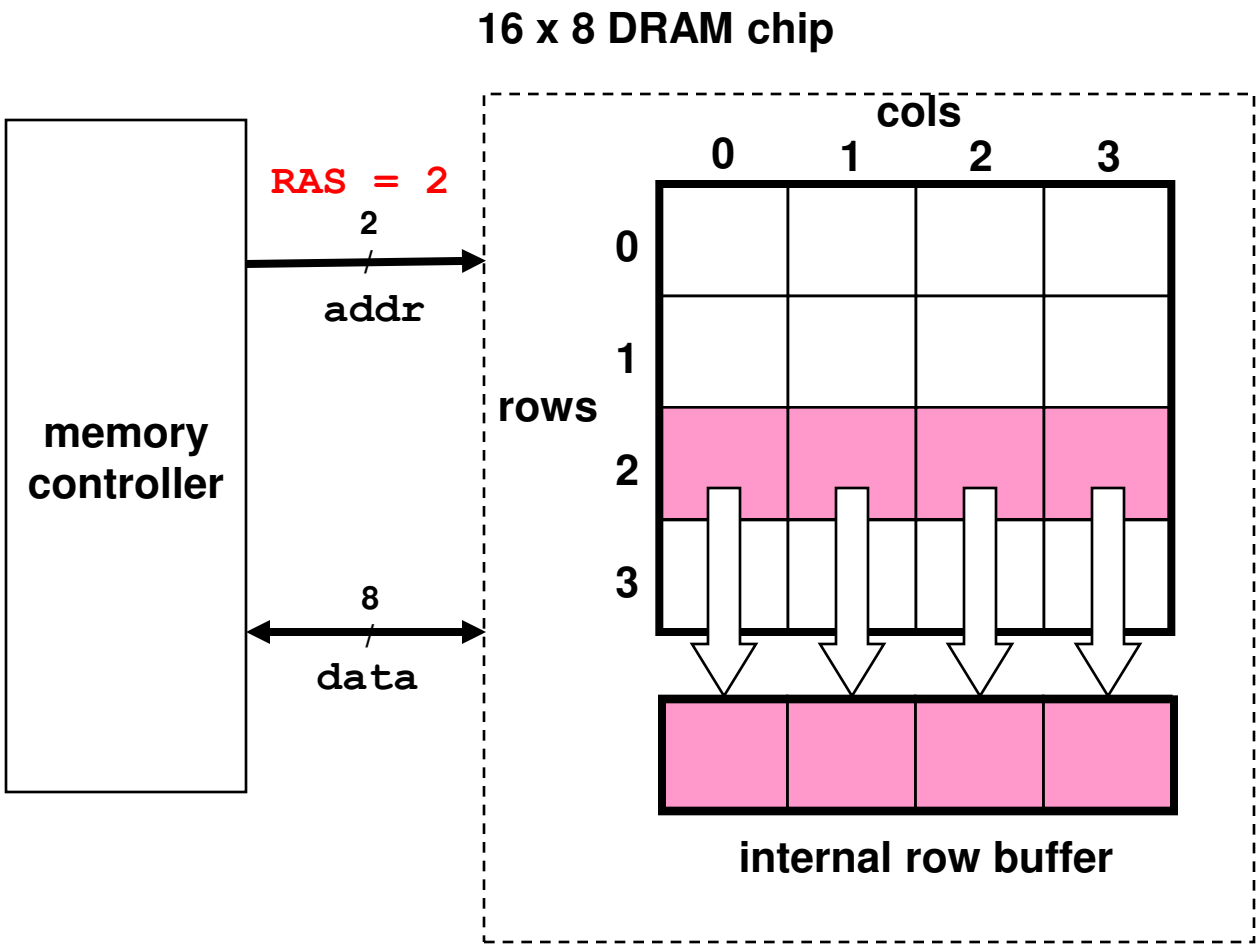
- d x w DRAM:
  - dw total bits organized as d supercells of size w bits



READING DRAM SUPERCELL (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

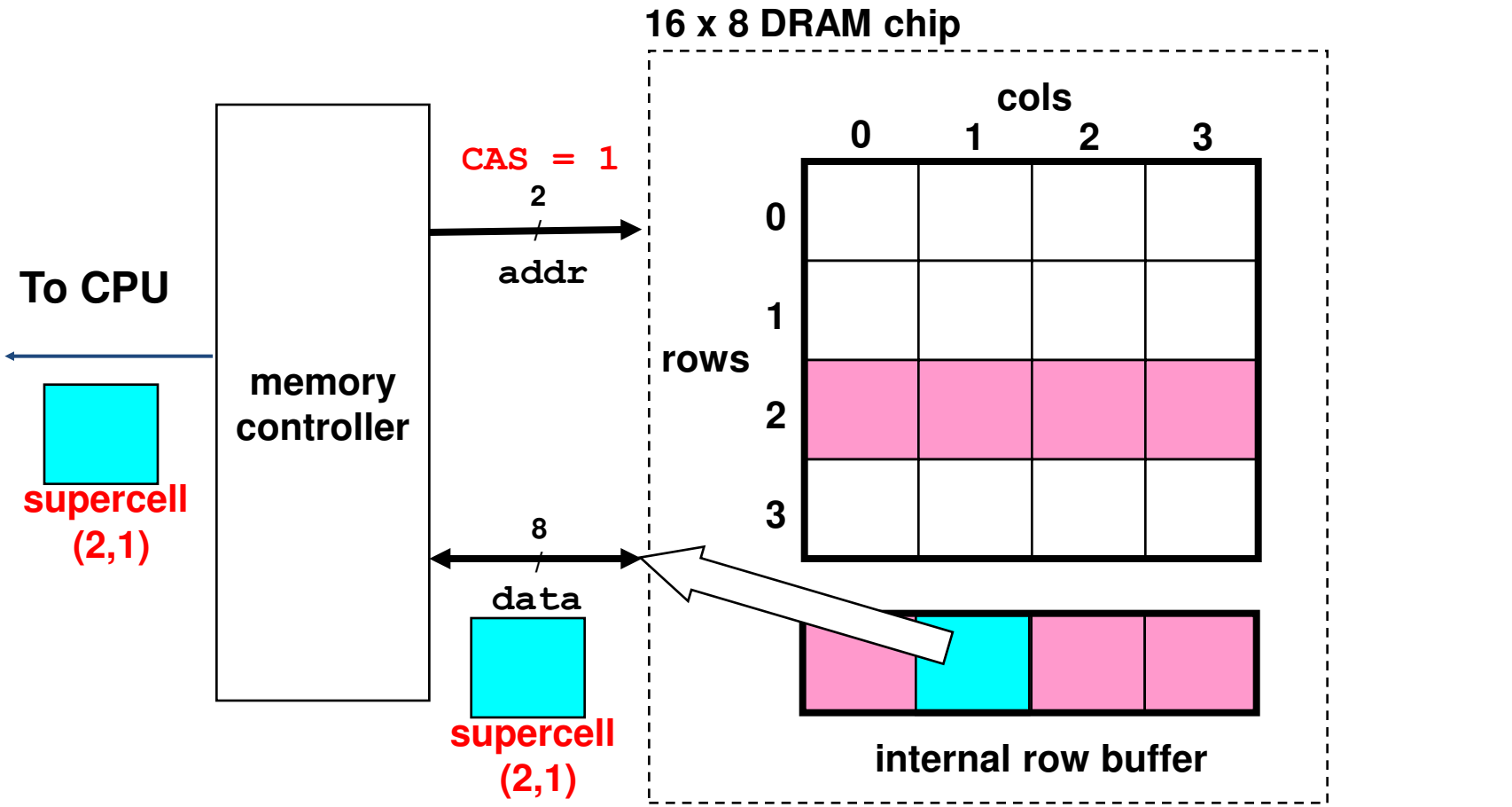
Step 1(b): Row 2 copied from DRAM array to row buffer.



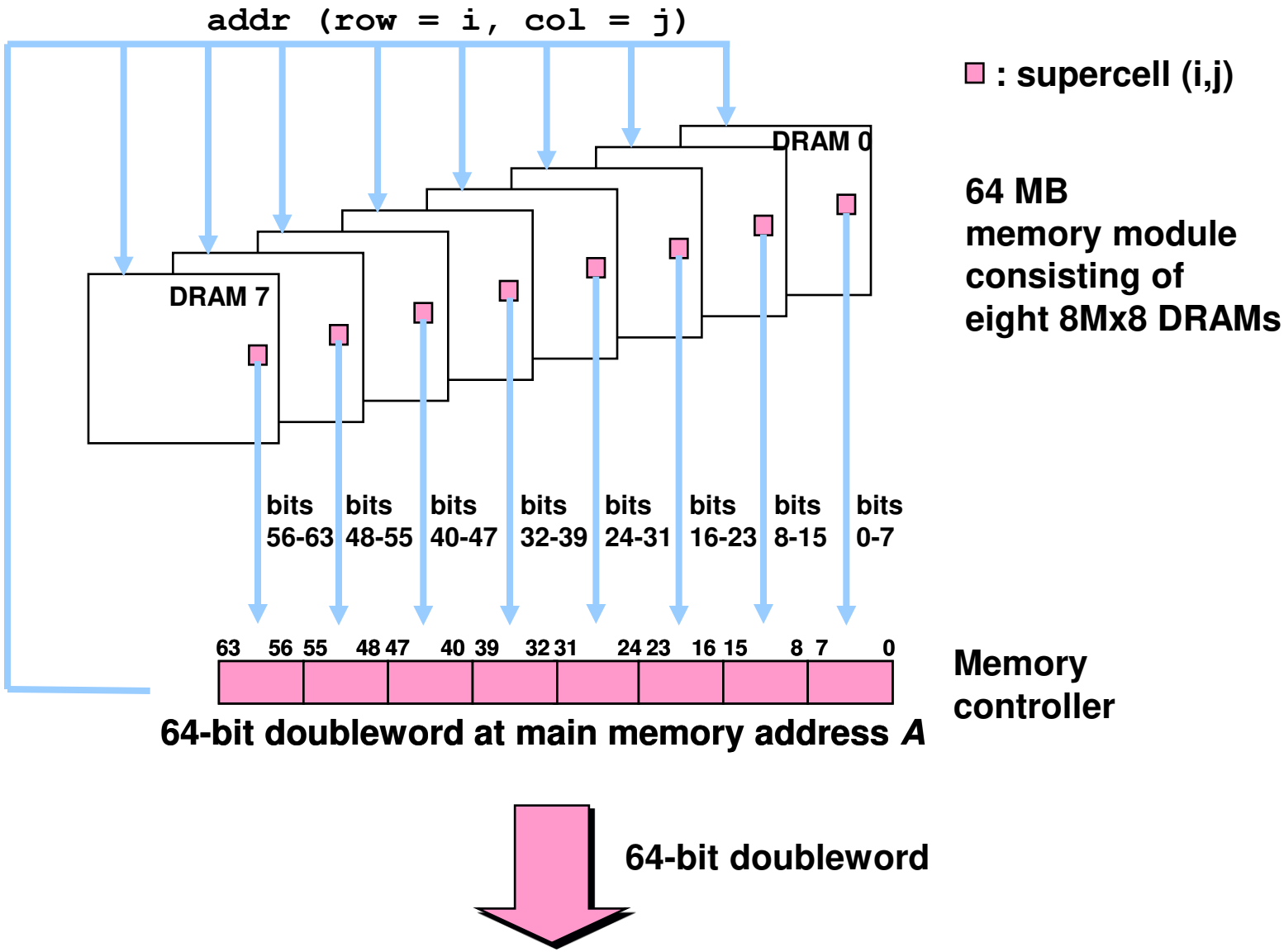
READING DRAM SUPERCELL (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



MEMORY MODULES



ENHANCED DRAMS

- All enhanced DRAMs are built around the conventional DRAM core.
  - Fast page mode DRAM (FPM DRAM)
    - Access contents of row with [RAS, CAS, CAS, CAS, CAS] instead of [(RAS,CAS), (RAS,CAS), (RAS,CAS), (RAS,CAS)].
  - Extended data out DRAM (EDO DRAM)
    - Enhanced FPM DRAM with more closely spaced CAS signals.
  - Synchronous DRAM (SDRAM)
    - Driven with rising clock edge instead of asynchronous control signals.

Cont…

- Double data-rate synchronous DRAM (DDR SDRAM)
  - Enhancement of SDRAM that uses both clock edges as control signals.
- Video RAM (VRAM)
  - Like FPM DRAM, but output is produced by shifting row buffer
  - Dual ported (allows concurrent reads and writes)

NONVOLATILE MEMORIES

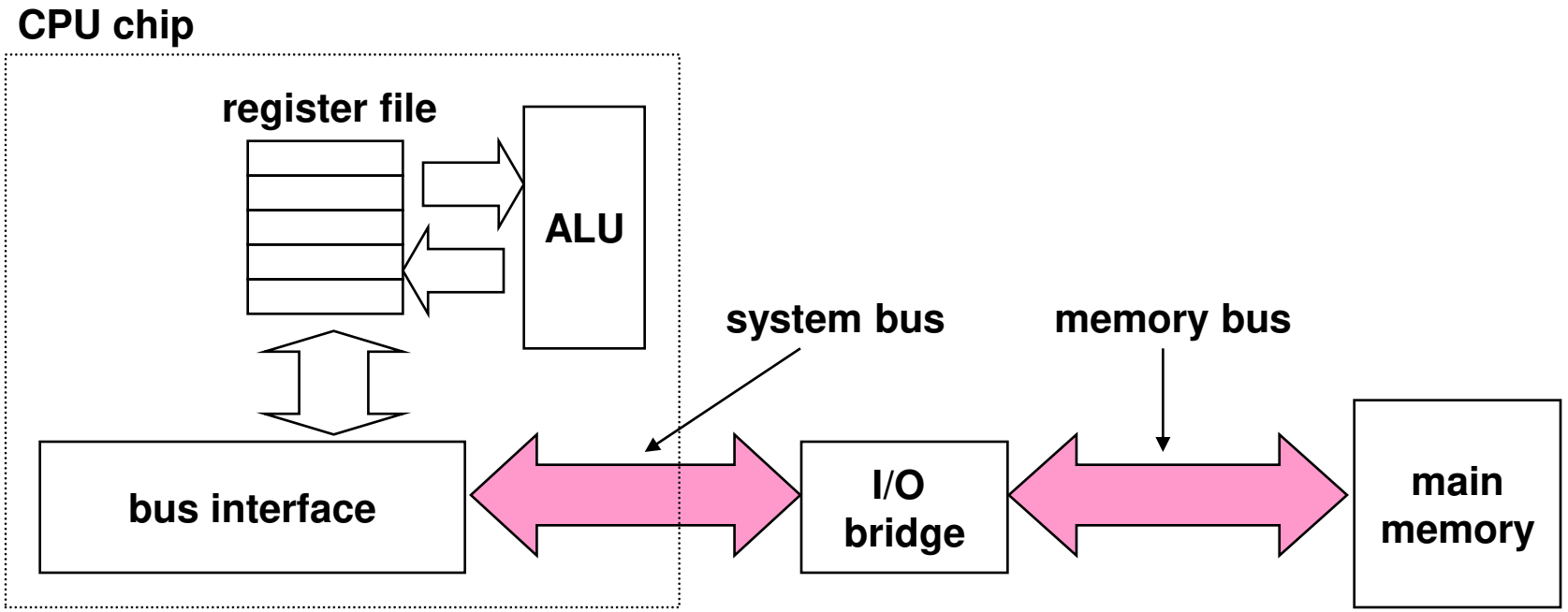
- DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off.
  - Generic name is read-only memory (ROM).
  - Misleading because some ROMs can be read and modified.
- Types of ROMs
  - Programmable ROM (PROM)
  - Eraseable programmable ROM (EPROM)
  - Electrically eraseable PROM (EEPROM)
  - Flash memory

Cont…

- Firmware
  - Program stored in a ROM
    - Boot time code, BIOS (basic input/output system)
    - graphics cards, disk controllers.

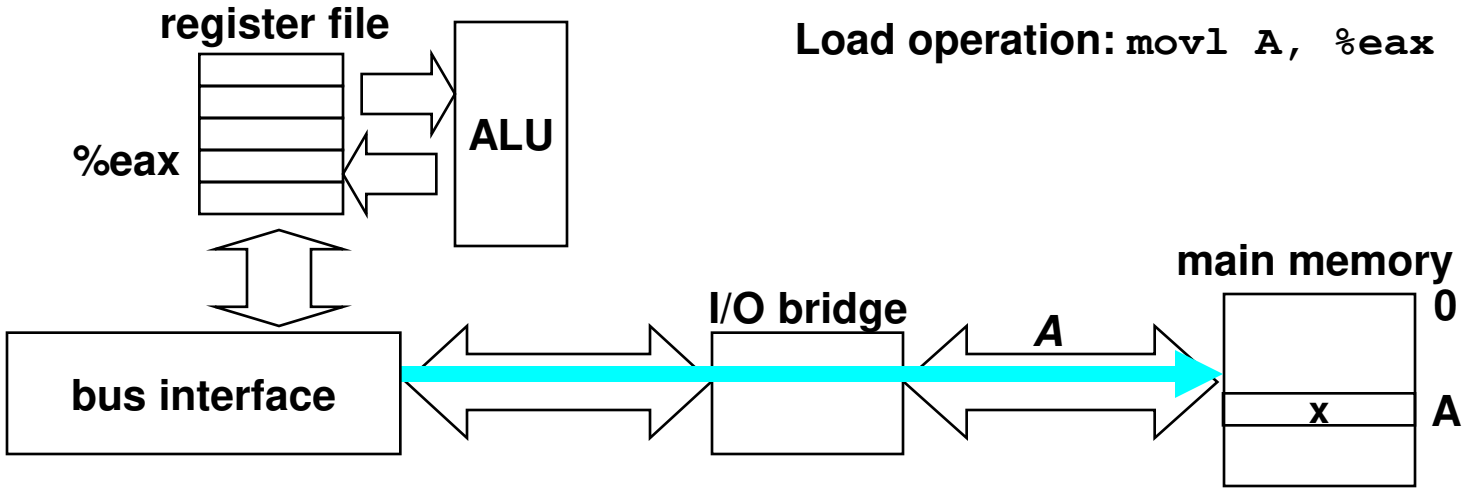
TYPICAL BUS STRUCTURE CONNECTING CPU AND MEMORY

- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.



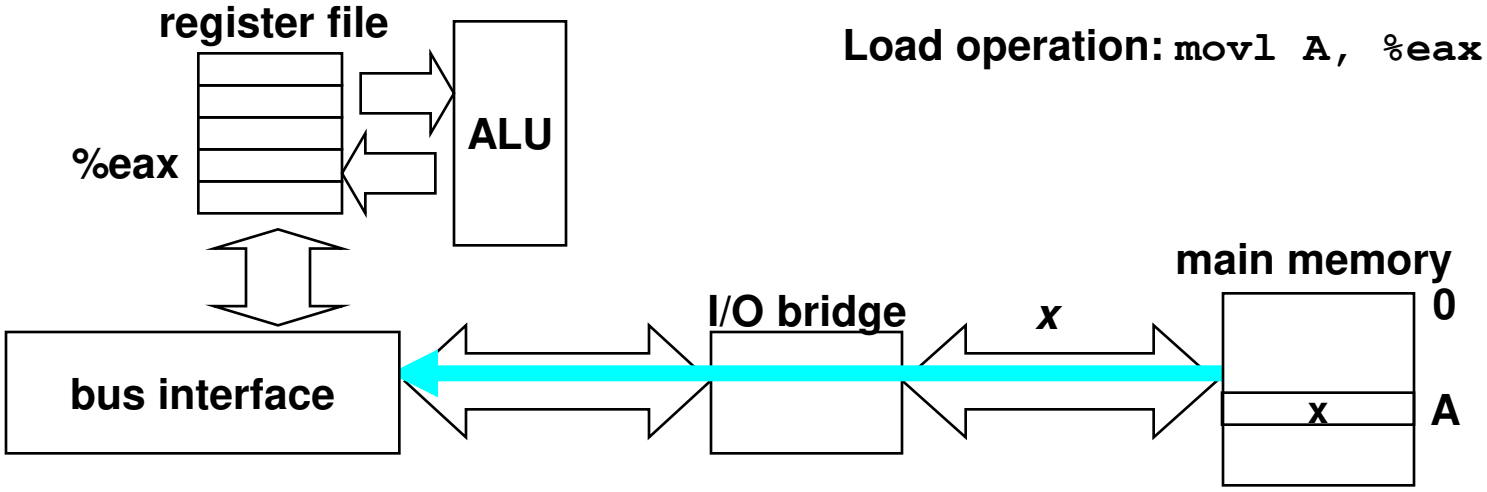
MEMORY READ TRANSACTION (1)

- CPU places address A on the memory bus.



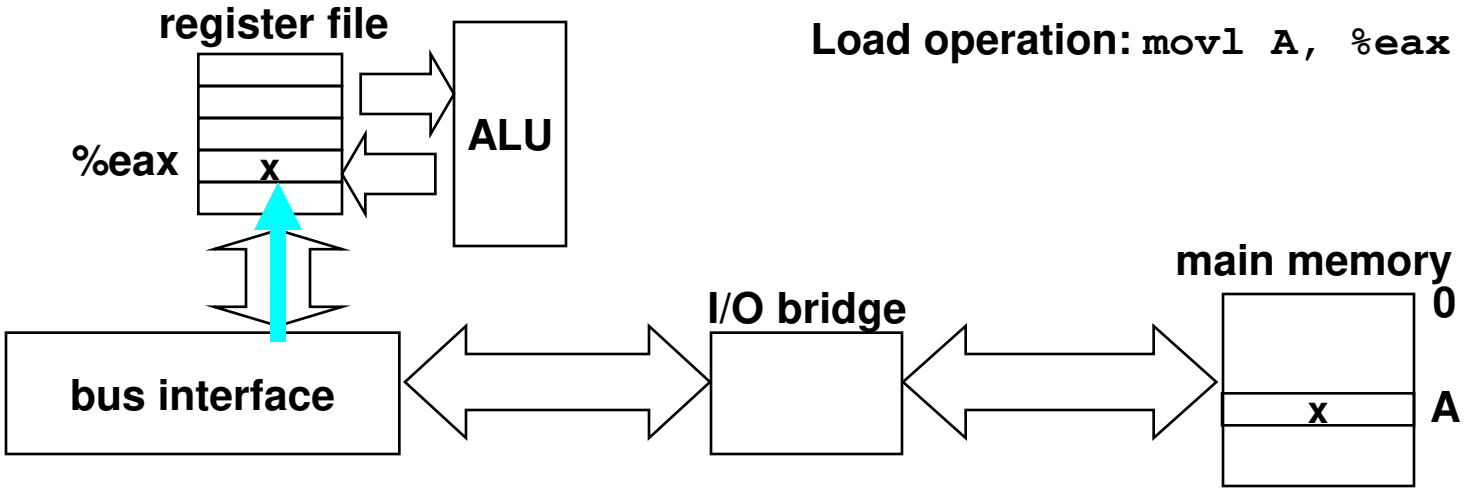
MEMORY READ TRANSACTION (2)

- Main memory reads A from the memory bus, retreives word x, and places it on the bus.



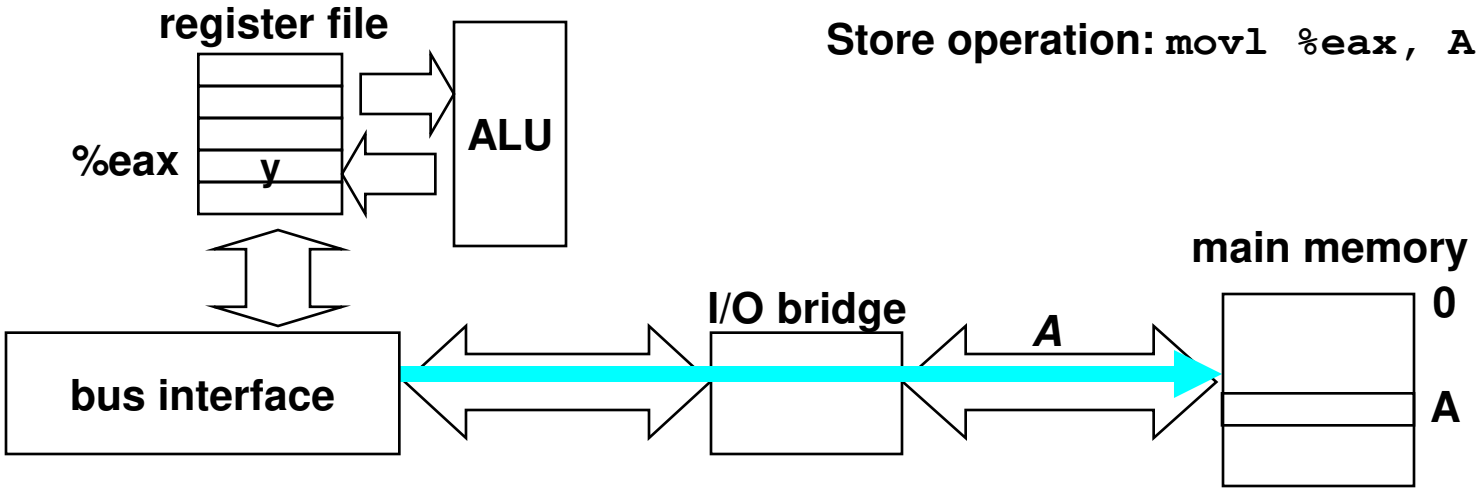
MEMORY READ TRANSACTION (3)

- CPU read word x from the bus and copies it into register %eax.



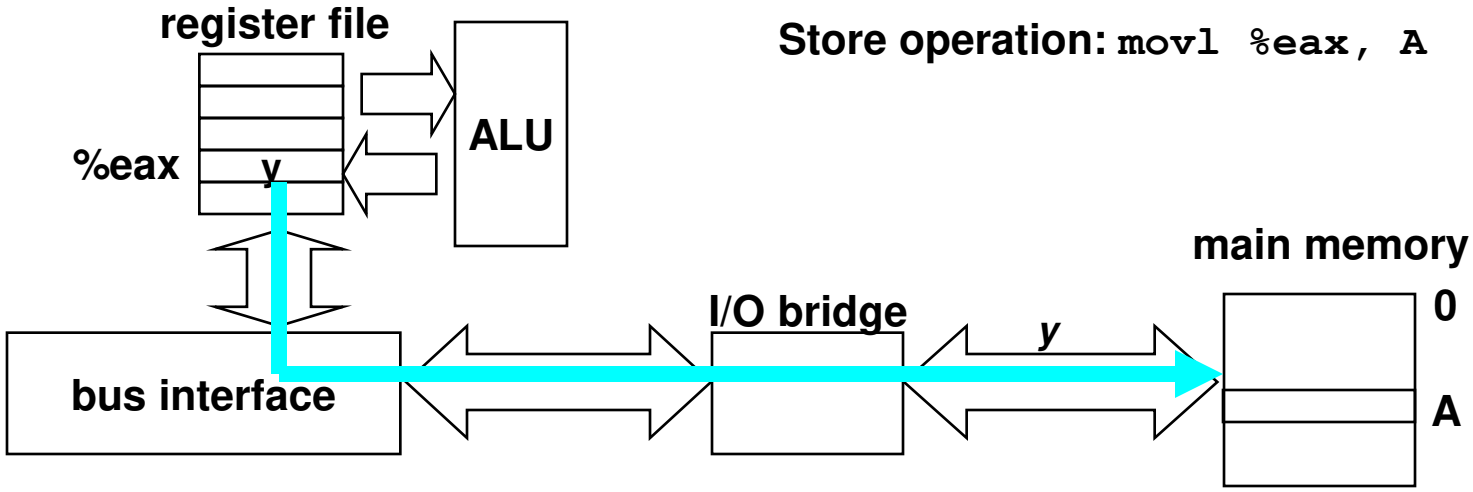
MEMORY WRITE TRANSACTION (1)

- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



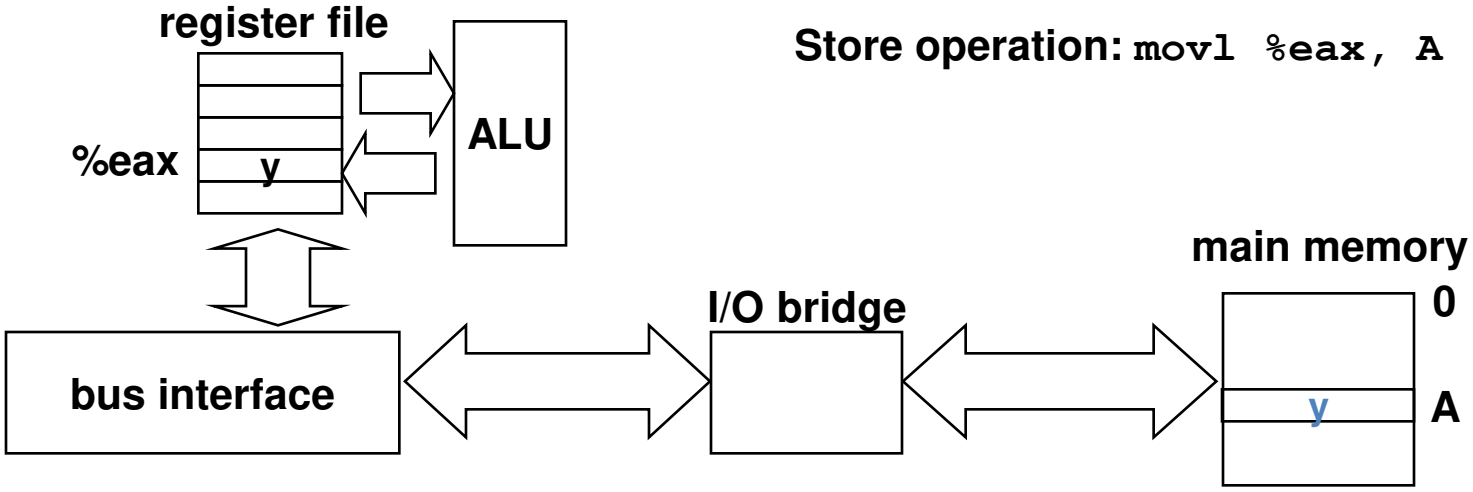
MEMORY WRITE TRANSACTION (2)

- CPU places data word y on the bus.

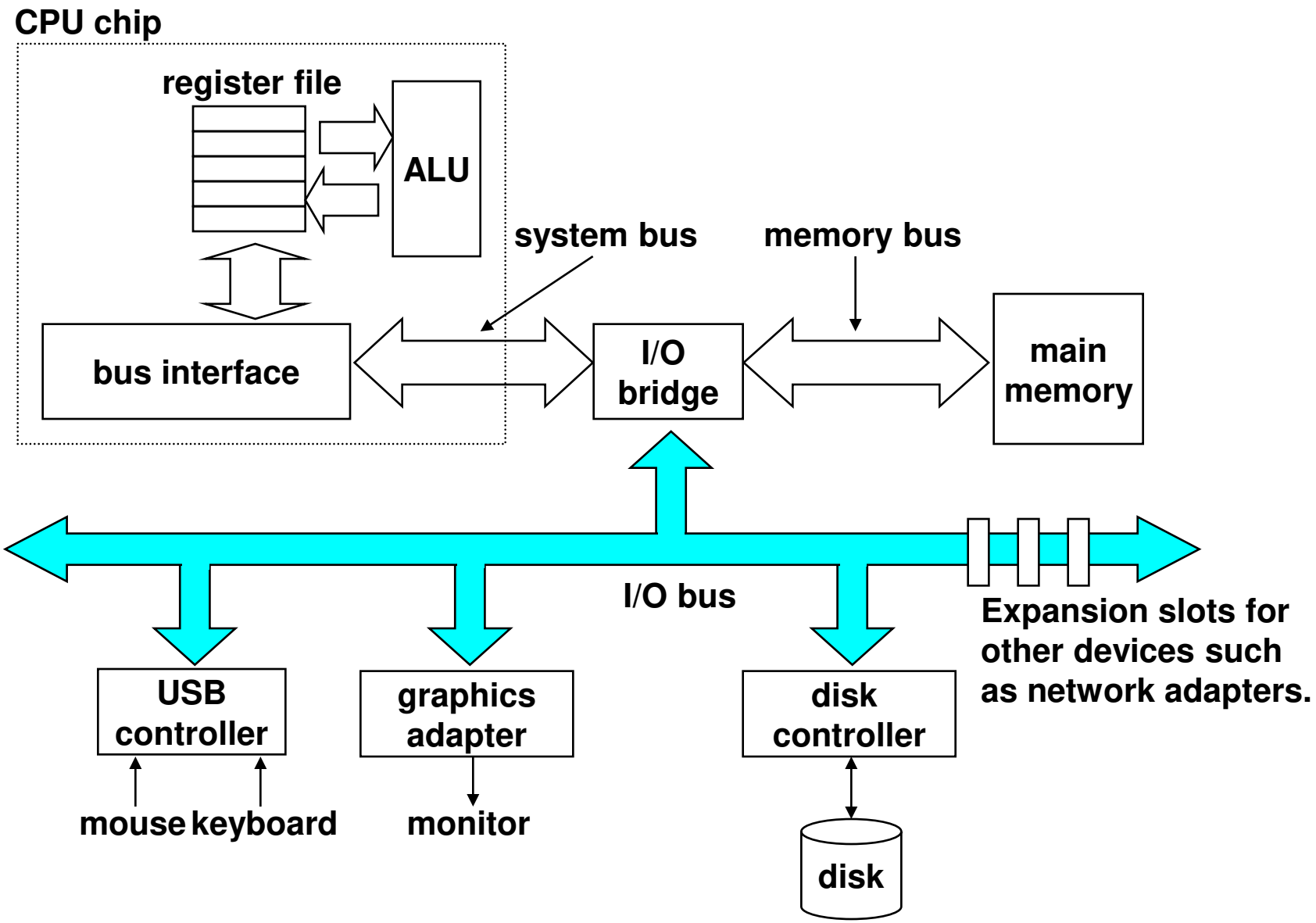


MEMORY WRITE TRANSACTION (3)

- Main memory read data word y from the bus and stores it at address A.

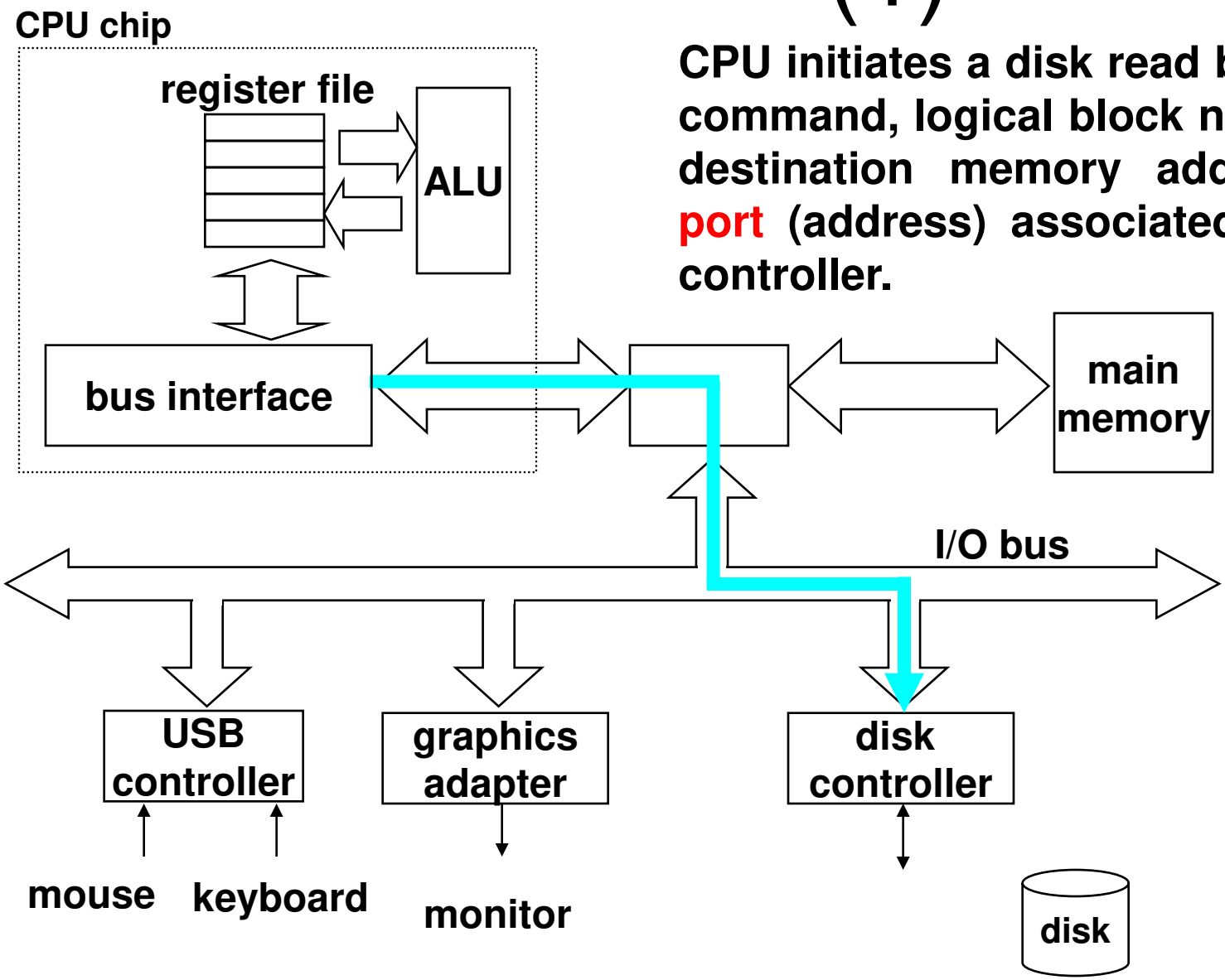


I/O BUS


CPU chip
register file
ALU
system bus
memory bus
bus interface
I/O bridge
main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
graphics adapter
disk controller
mouse
keyboard
monitor
disk

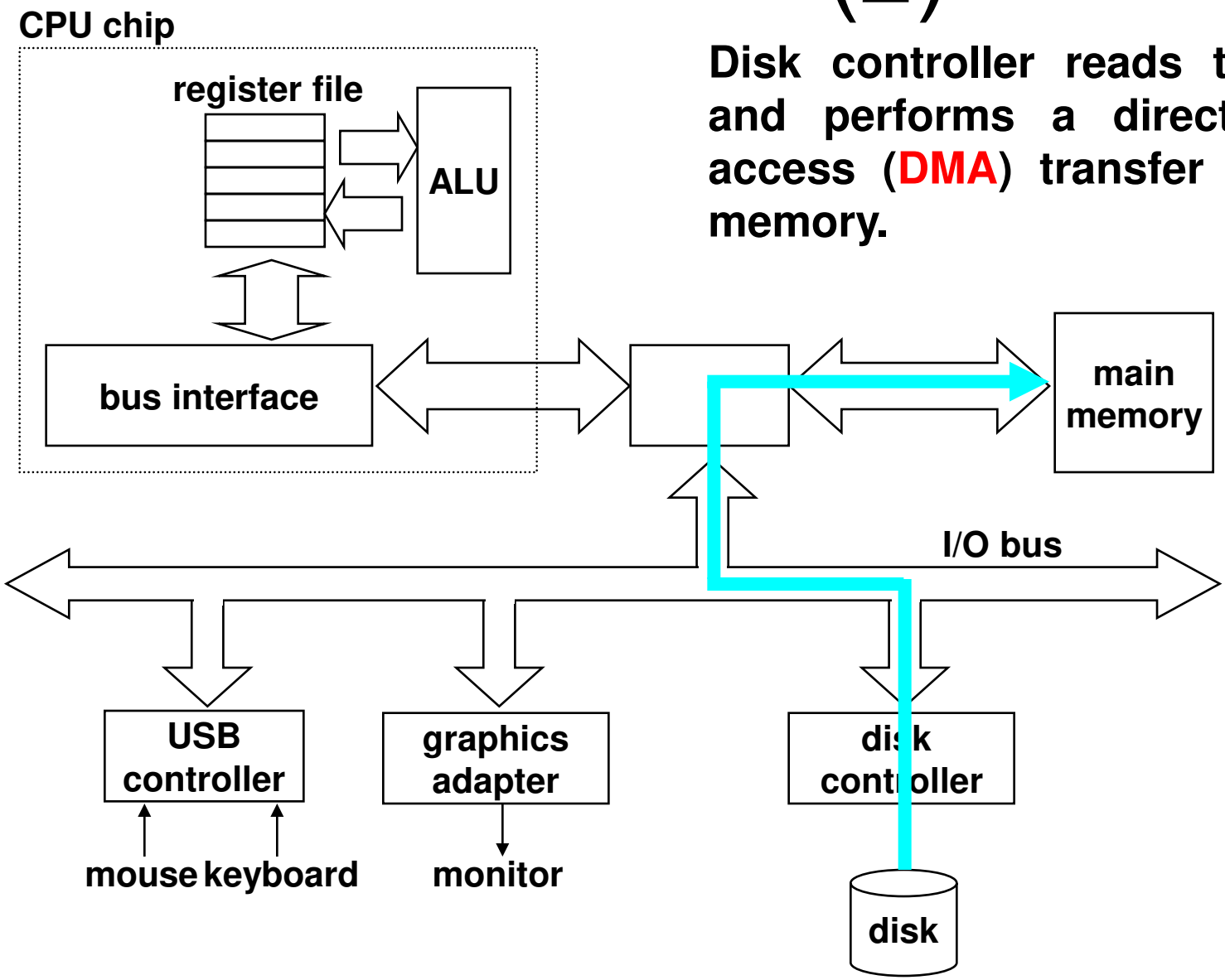
READING A DISK SECTOR (1)

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.



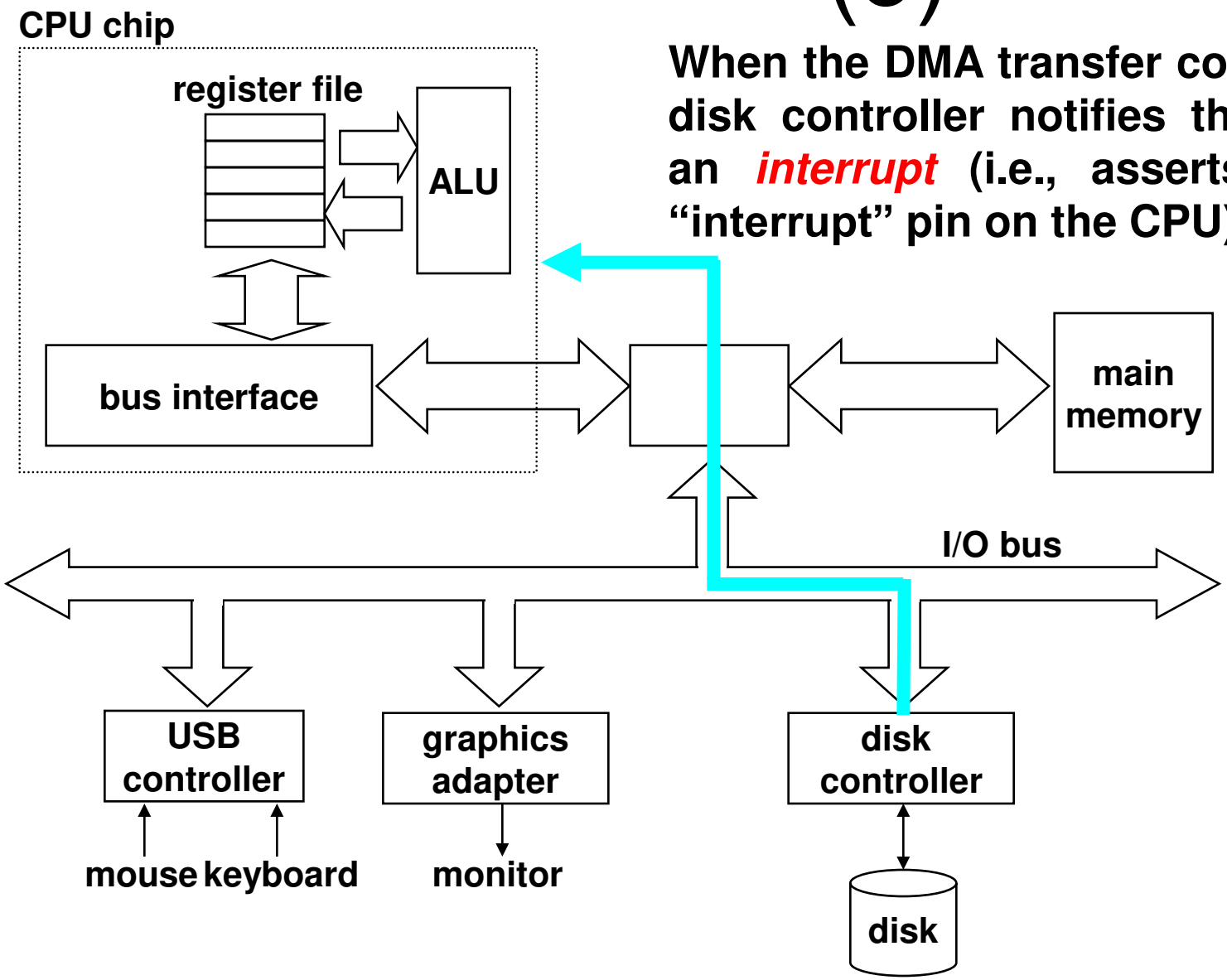
READING A DISK SECTOR (2)

Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.



READING A DISK SECTOR (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)



MEMORY HIERARCHIES

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte and have less capacity.
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

AUXILIARY MEMORY

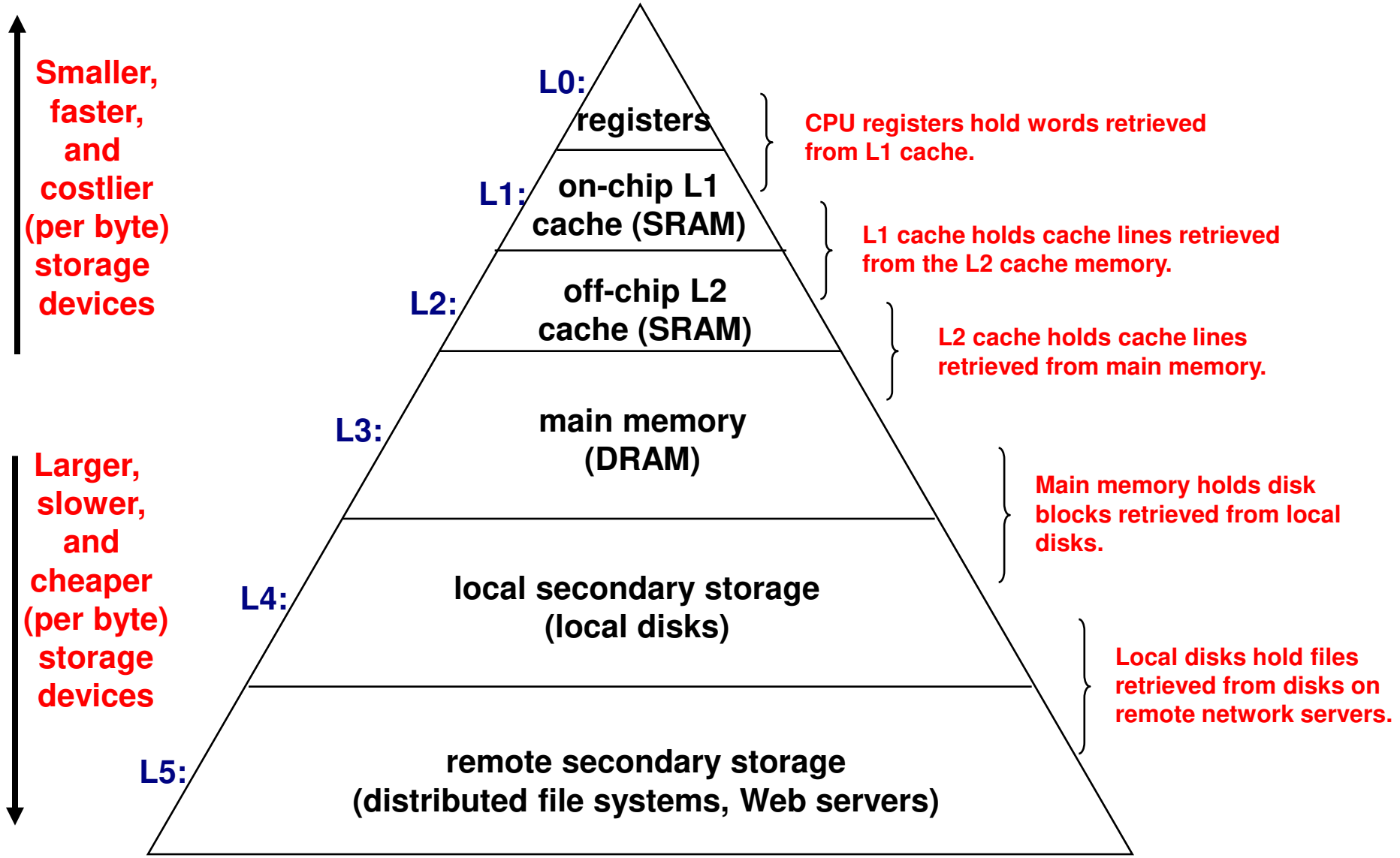
Physical Mechanism

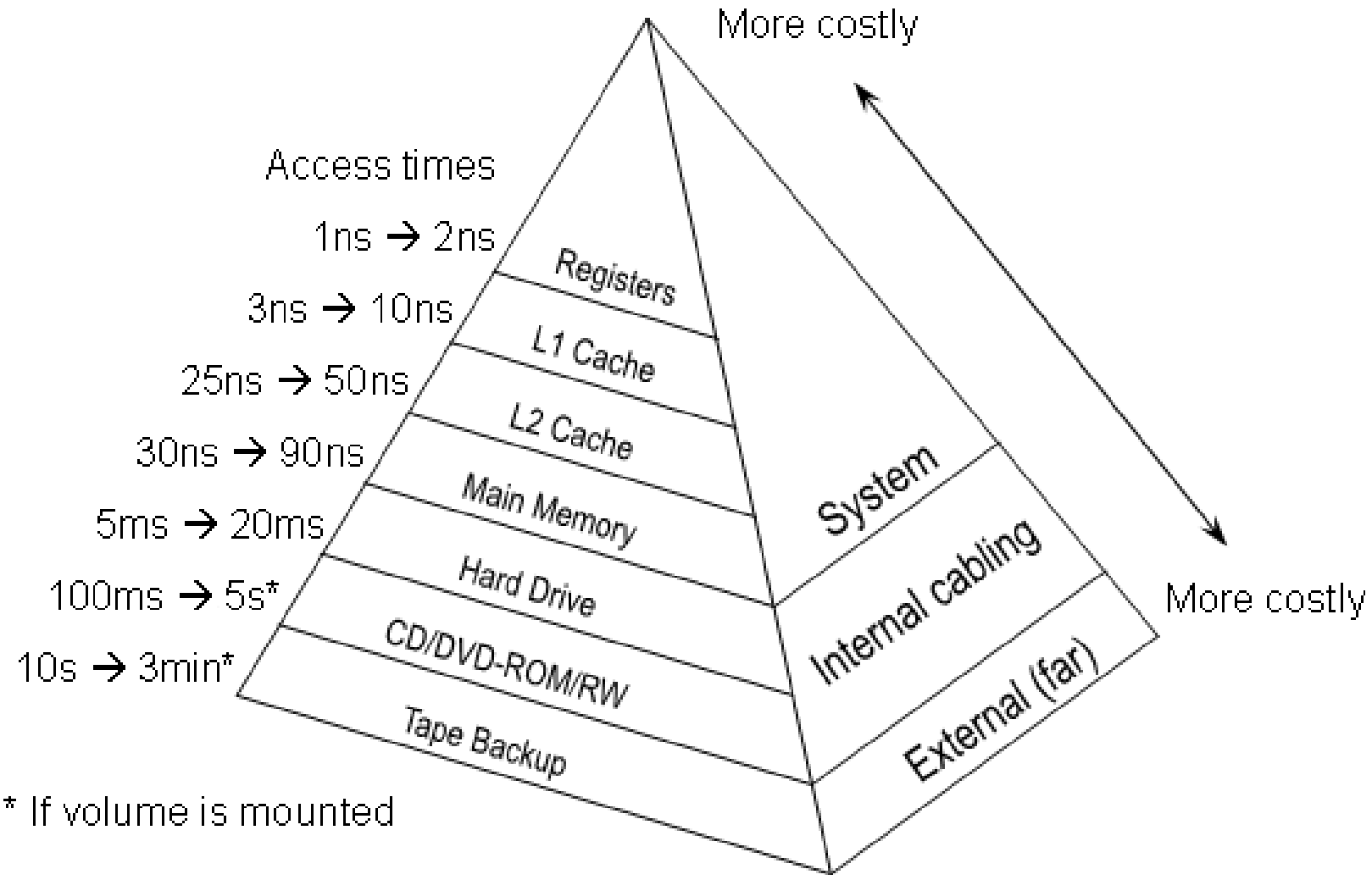
- Magnetic
- Electronic
- Electromechanical

Characteristic of any device

- Access mode
- Access Time
- Transfer Rate
- Capacity
- Cost

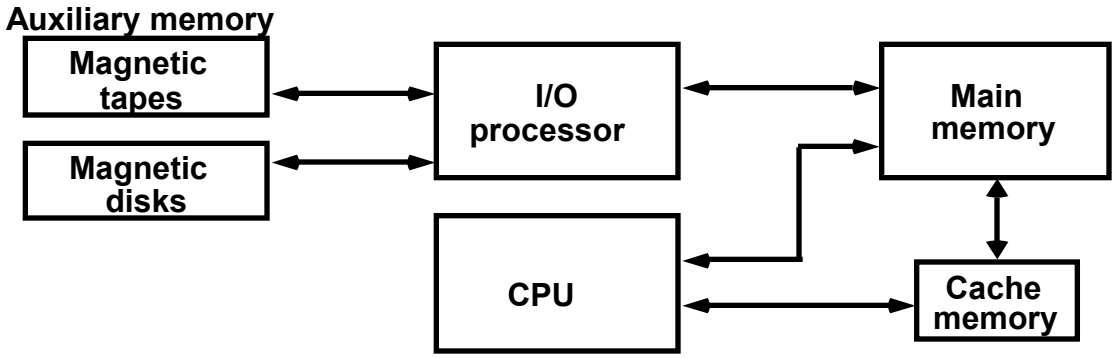
AN EXAMPLE MEMORY HIERARCHY

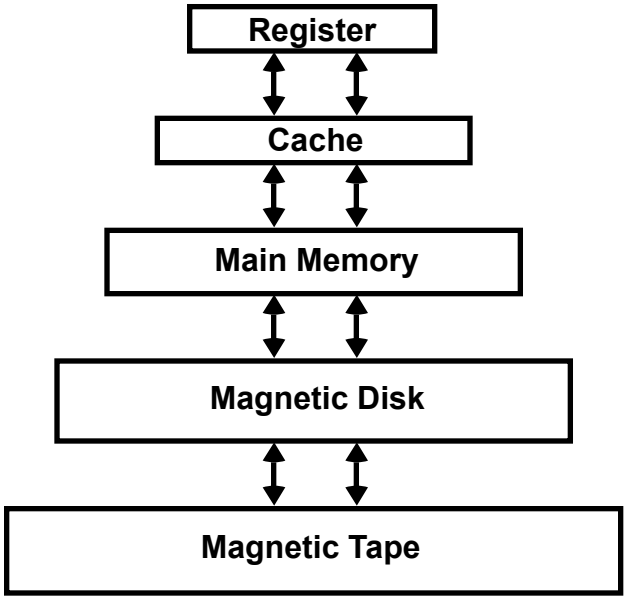



More costly
Access times
1ns → 2ns
3ns → 10ns
25ns → 50ns
30ns → 90ns
5ms → 20ms
100ms → 5s*
10s → 3min*
Registers
L1 Cache
L2 Cache
Main Memory
Hard Drive
CD/DVD-ROM/RW
Tape Backup
System
Internal cabling
External (far)
More costly
* If volume is mounted

MEMORY HIERARCHY

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





ACCESS METHODS

- **Sequential**
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location – e.g. tape
- **Direct**
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location – e.g. disk

Cont..

Random

- Individual addresses identify locations exactly
- Access time is independent of location or previous access – e.g. RAM

- **Associative**
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access – e.g. cache

PERFORMANCE

- **Access time**
  - Time between presenting the address and getting the valid data
- **Memory Cycle time**
  - Time may be required for the memory to "recover" before next access
  - *Cycle time* is access + recovery
- **Transfer Rate**
  - Rate at which data can be moved

MAIN MEMORY

SRAM vs. DRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Simpler to build, smaller
  - More dense
  - Less expensive
  - Needs refresh
  - Larger memory units (DIMMs)
- Static
  - Faster
  - Cache

Cont…

1K x 8:

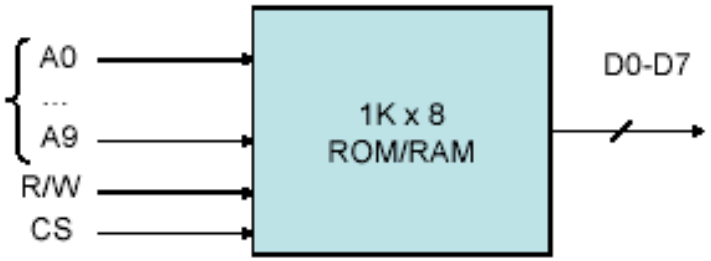
1K = 2n,

n: number of address lines

8: number of data lines

R/W: Read/Write Enable

CS: Chip Select.



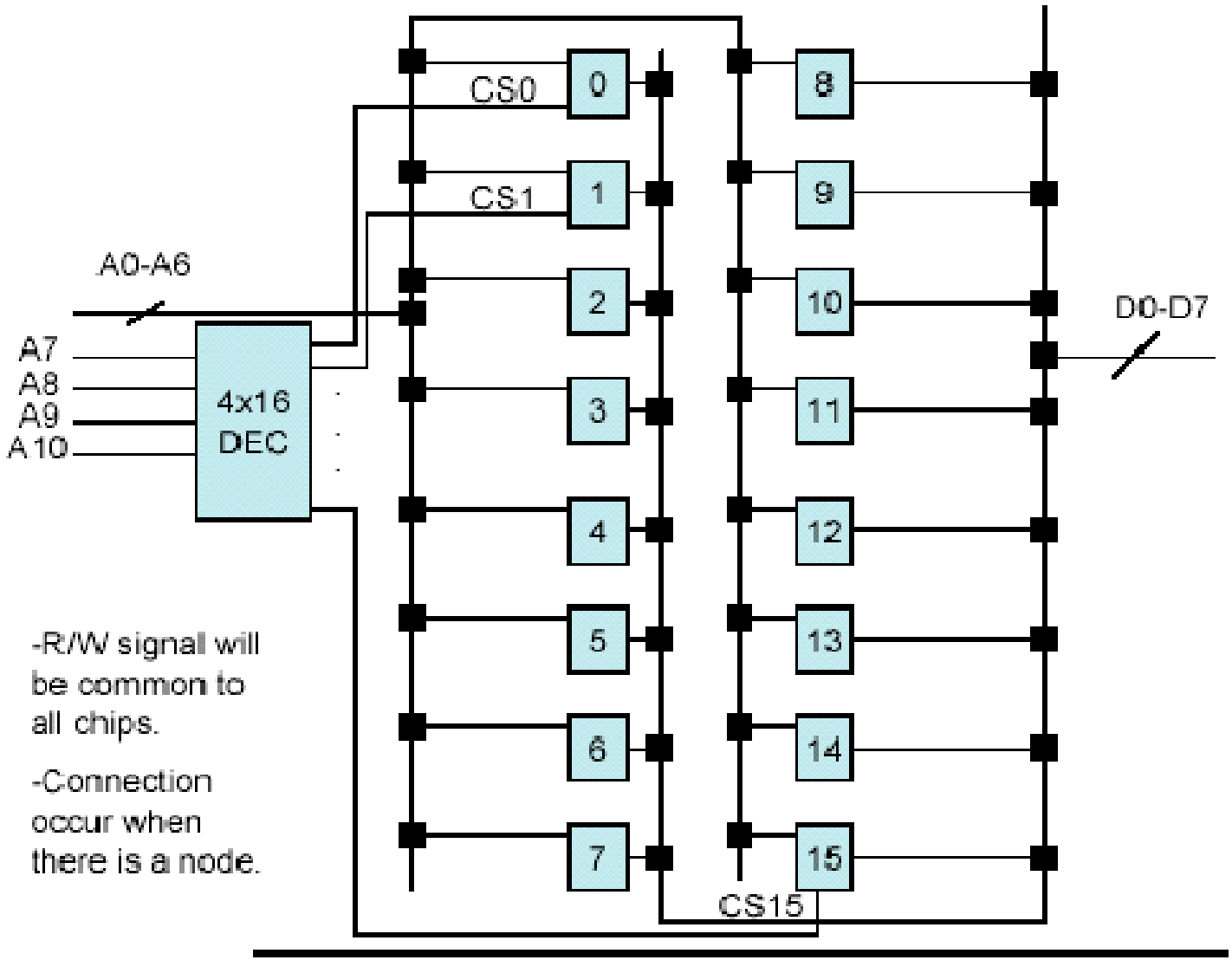
PROBLEMS

a) For a memory capacity of 2048 bytes, using 128x8 chips, we need 2048/128=16 chips.

b) We need 11 address lines to access 2048 = 2^{11}, the common lines are 7 (since each chip has 7 address lines; 128= 2^{7})

c) We need a decoder to select which chip is to accessed. Draw a diagram to show the connections.

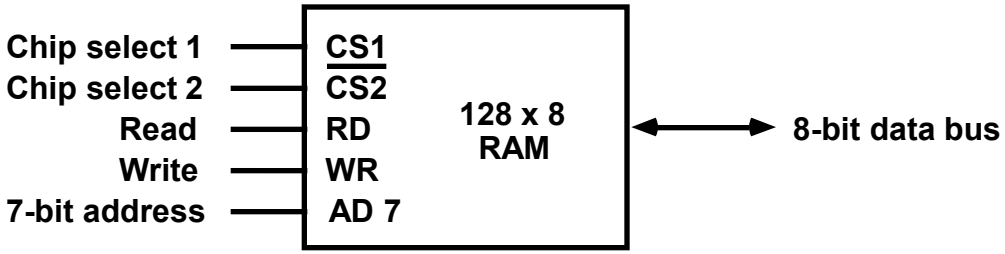
Cont…


CS0
CS1
A0-A6
A7
A8
A9
A10
4x16
DEC
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
CS15
D0-D7
-R/W signal will be common to all chips.
-Connection occur when there is a node.

MAIN MEMORY

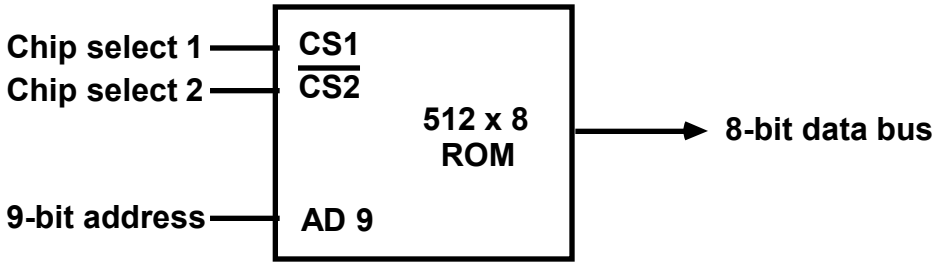
RAM and ROM Chips

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

Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

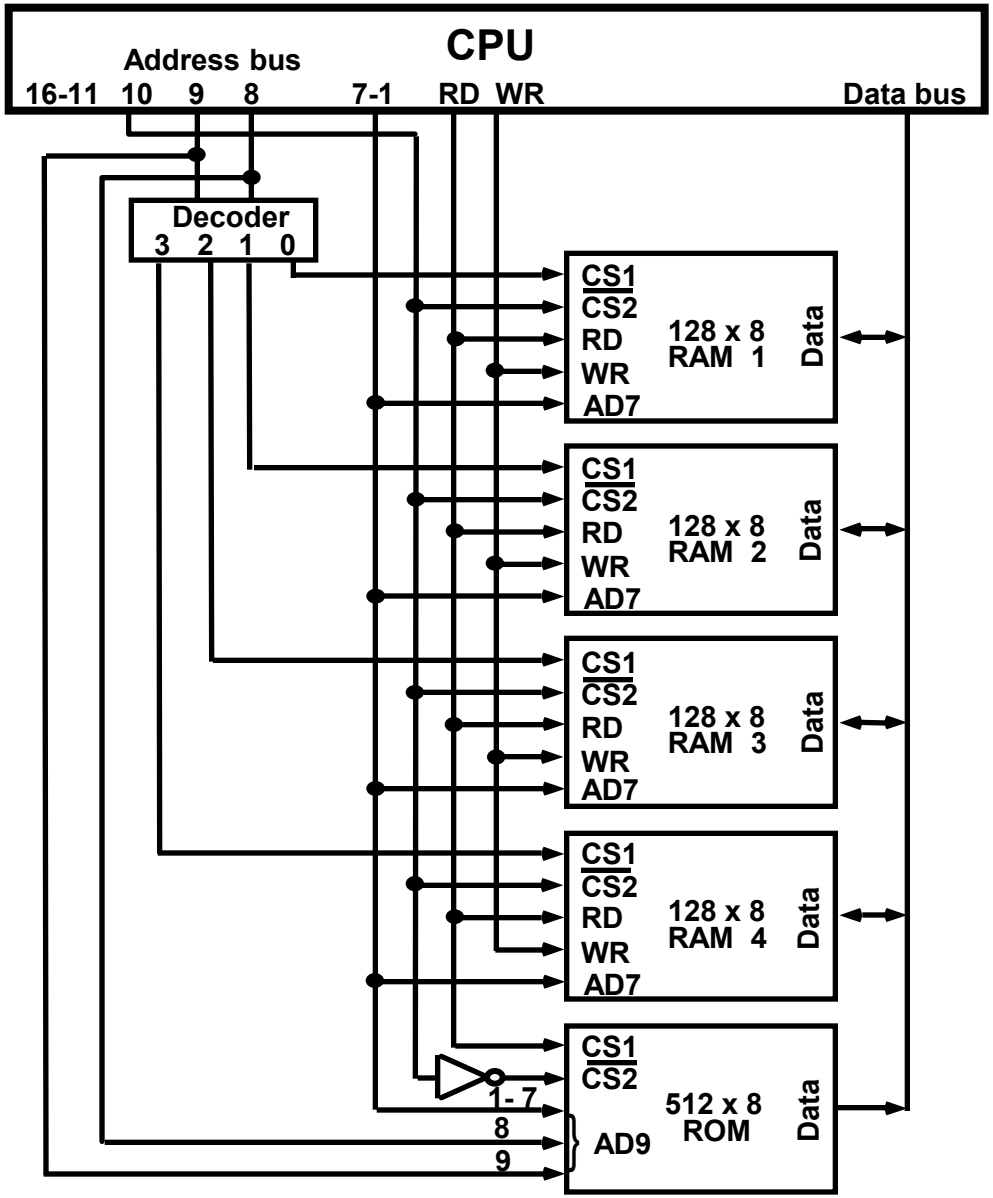
Component	Hexa address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Memory Connection to CPU

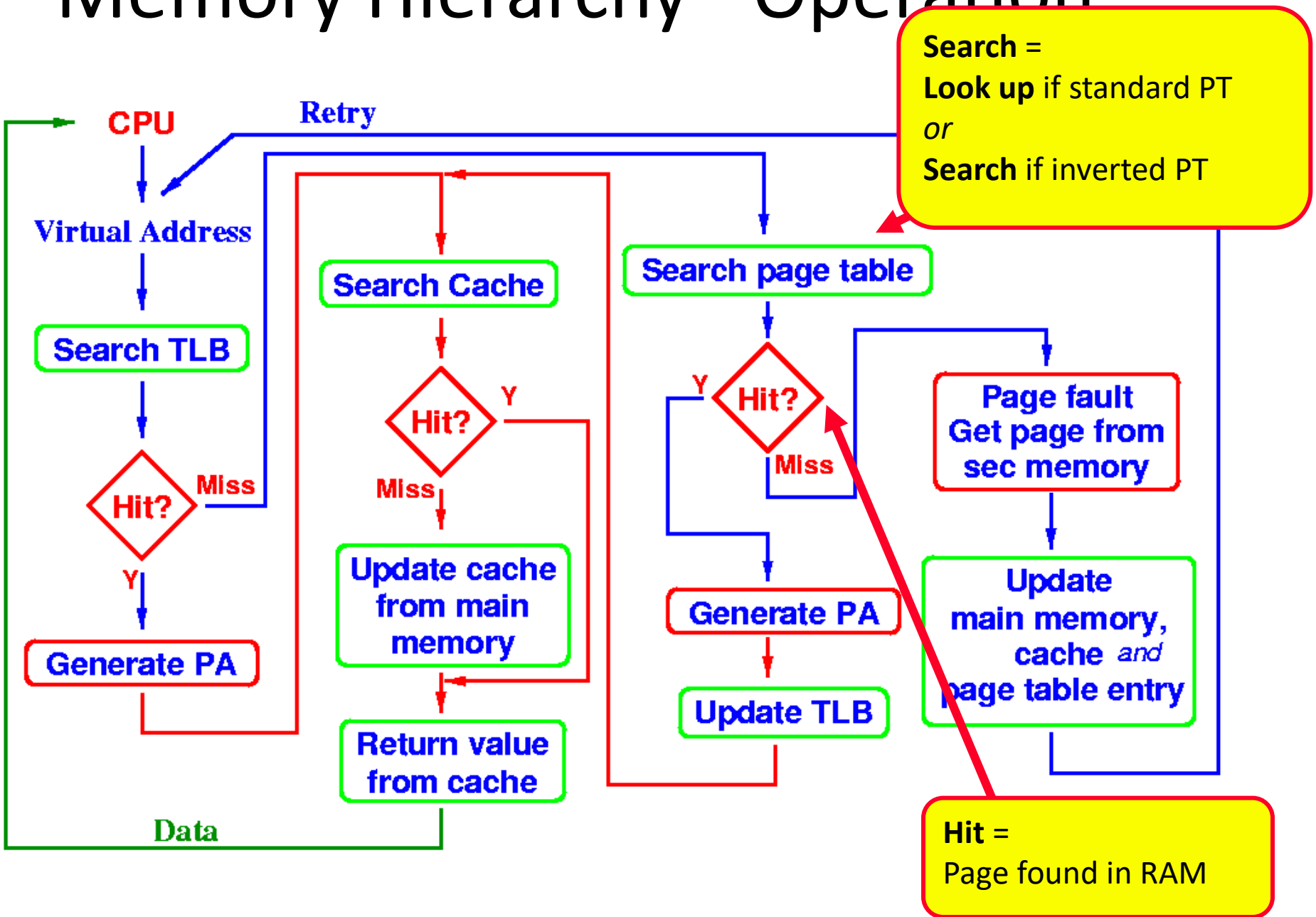
- **RAM and ROM chips are connected to a CPU through the data and address buses**

- **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**

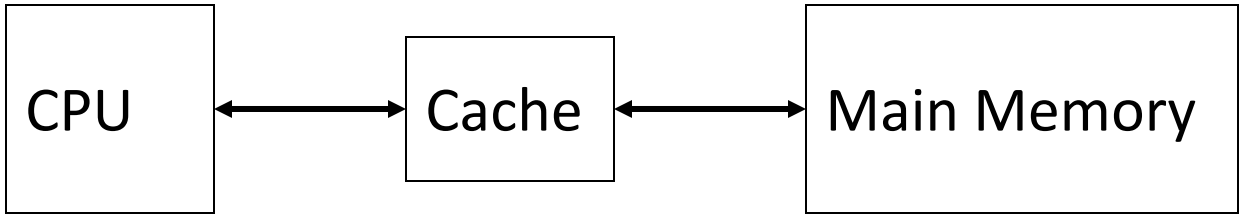
CONNECTION OF MEMORY TO CPU



Memory Hierarchy - Operation



Cache & Locality



- Cache sits between the CPU and main memory
  - Invisible to the CPU
- Only useful if recently used items are used again
- Fortunately, this happens a lot. We call this property *locality of reference.*

Locality of reference

- *Temporal locality*
  - Recently accessed data/instructions are likely to be accessed again.
    - Most program time is spent in loops
    - Arrays are often scanned multiple times
- *Spatial locality*
  - If I access memory address *n*, I am likely to then access another address close to *n* (usually *n*+1, *n*+2, or *n*+4)
    - Linear execution of code
    - Linear access of arrays

How a cache exploits locality

- Temporal – When an item is accessed from memory it is brought into the cache
  - If it is accessed again soon, it comes from the cache and not main memory
- Spatial – When we access a memory word, we also fetch the next few words of memory into the cache
  - The number of words fetched is the *cache line* size, or the *cache block size* for the machine

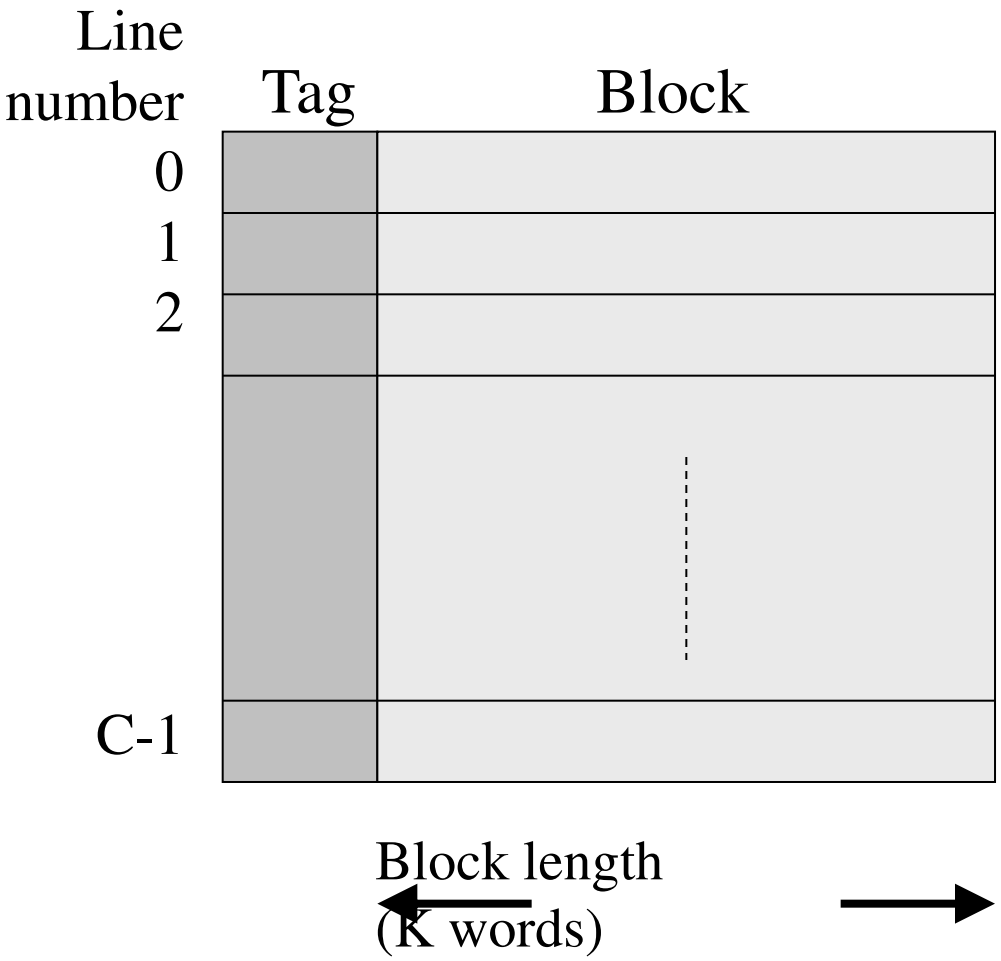
Going Deeper with Principle of Locality

- Cache "misses" are unavoidable, i.e., every piece of data and code thing must be loaded at least once
- What does a processor do during a miss? It waits for the data to be loaded.
- Power consumption varies linearly with clock speed and the square of the voltage.

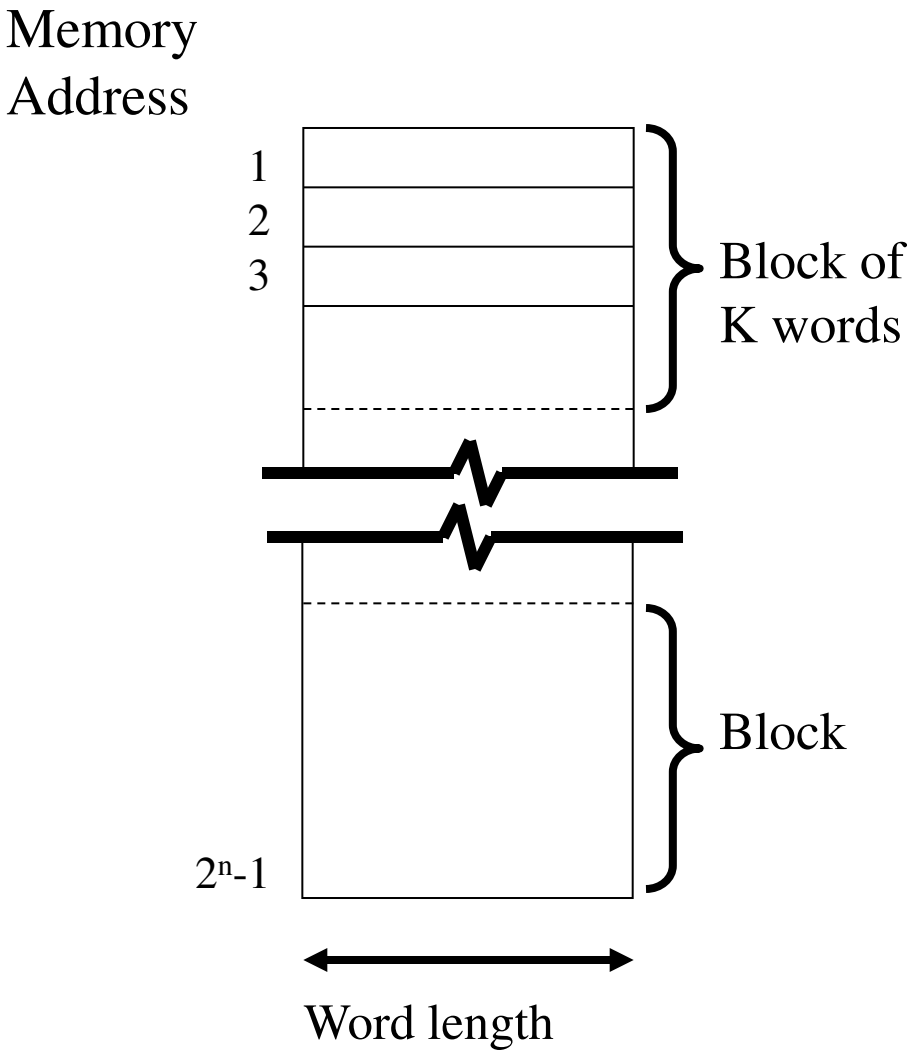
Cache Structure

- Cache includes tags to identify the address of the block of main memory contained in a line of the cache
- Each word in main memory has a unique n-bit address
- There are $M=2^n/K$ block of K words in main memory
- Cache contains C lines of K words each plus a tag uniquely identifying the block of K words

Cache Structure (continued)



Memory Divided into Blocks



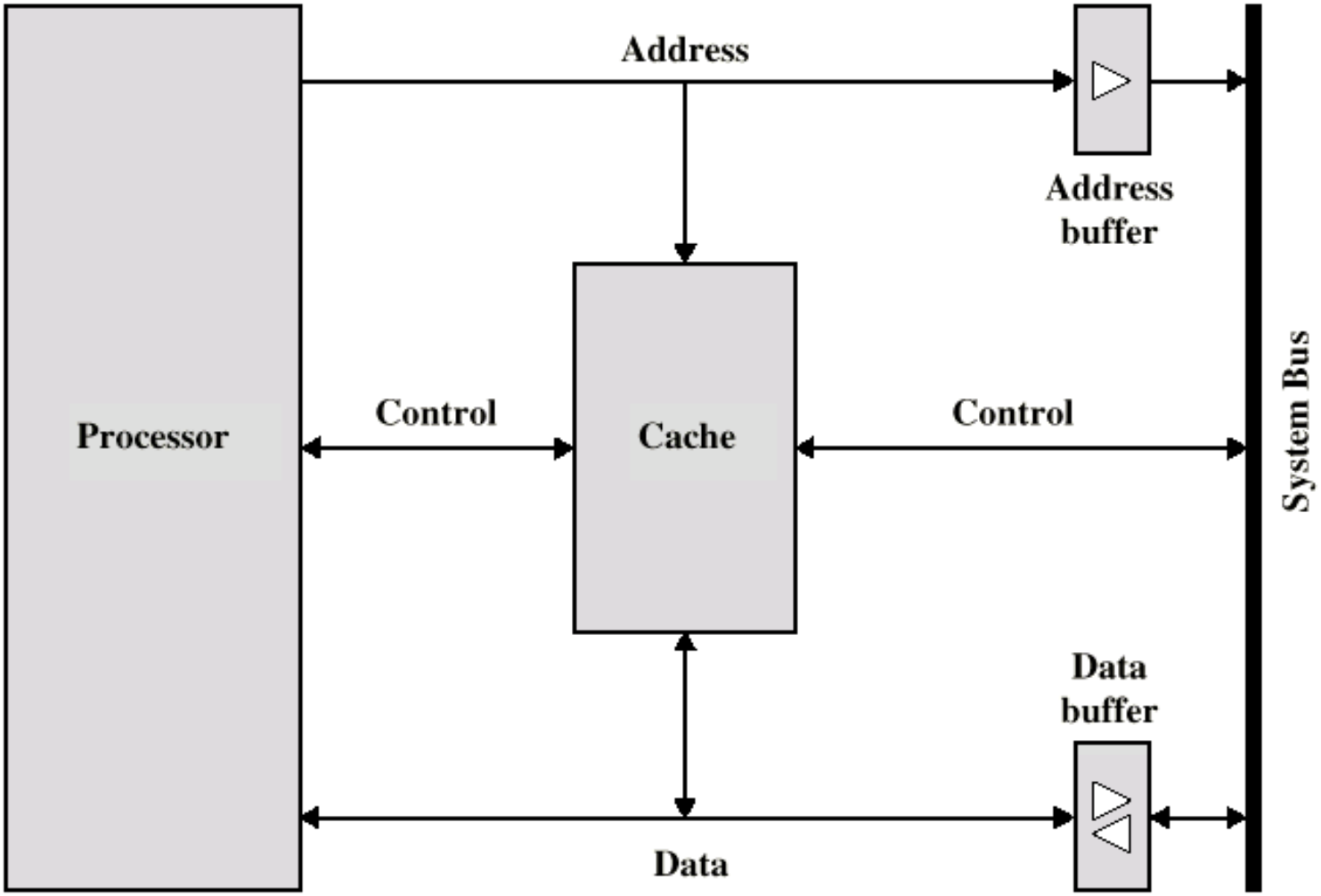
Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Cache size

- Cost – More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Larger decoding circuits slow up a cache
  - Algorithm is needed for mapping main memory addresses to lines in the cache. This takes more time than just a direct RAM

Typical Cache Organization



Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
- It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Associative
  - Set Associative

Cache Example

These notes use an example of a cache to illustrate each of the mapping functions. The characteristics of the cache used are:

- Size: 64 kByte
- Block size: 4 bytes – i.e. the cache has 16k (2^{14}) lines of 4 bytes
- Address bus: 24-bit– i.e., 16M bytes main memory divided into 4M 4 byte blocks

Direct Mapping Traits

- Each block of main memory maps to only one cache line – i.e. if a block is in cache, it will always be found in the same place
- Line number is calculated using the following function

$$i = j \text{ modulo } m$$

where

i = cache line number

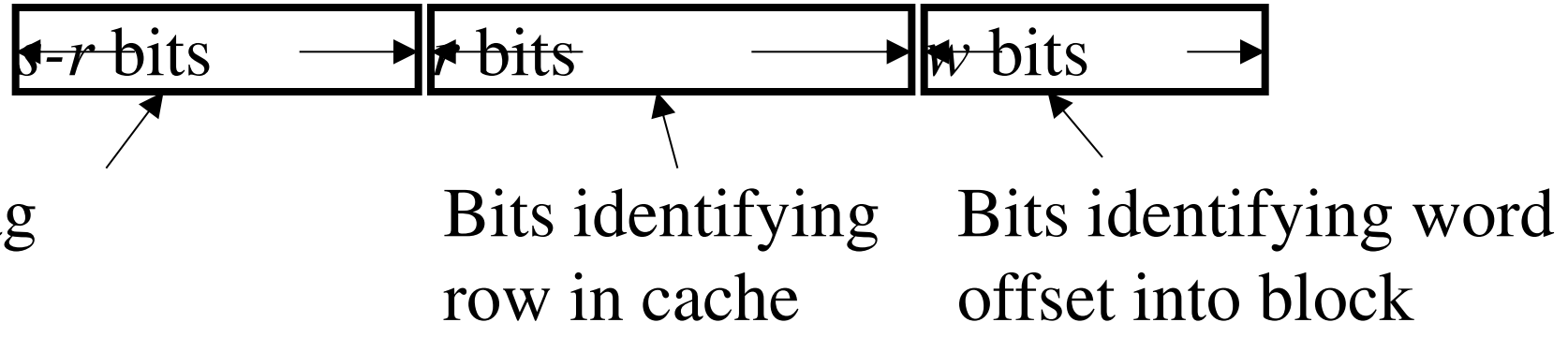
j = main memory block number

m = number of lines in the cache

Direct Mapping Address Structure

Each main memory address can by divided into three fields

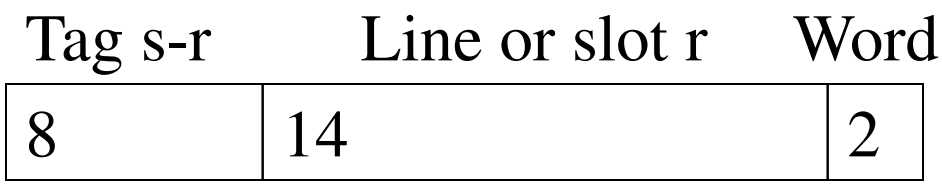
- Least Significant w bits identify unique word within a block
- Remaining bits (s) specify which block in memory. These are divided into two fields
  - Least significant r bits of these s bits identifies which line in the cache
  - Most significant s-r bits uniquely identifies the block within a line of the cache



Direct Mapping Address Structure (continued)

- Why are the r-bits used to identify which line in cache?
- More likely to have unique r bits than s-r bits based on principle of **locality of reference**

Direct Mapping Address Structure Example

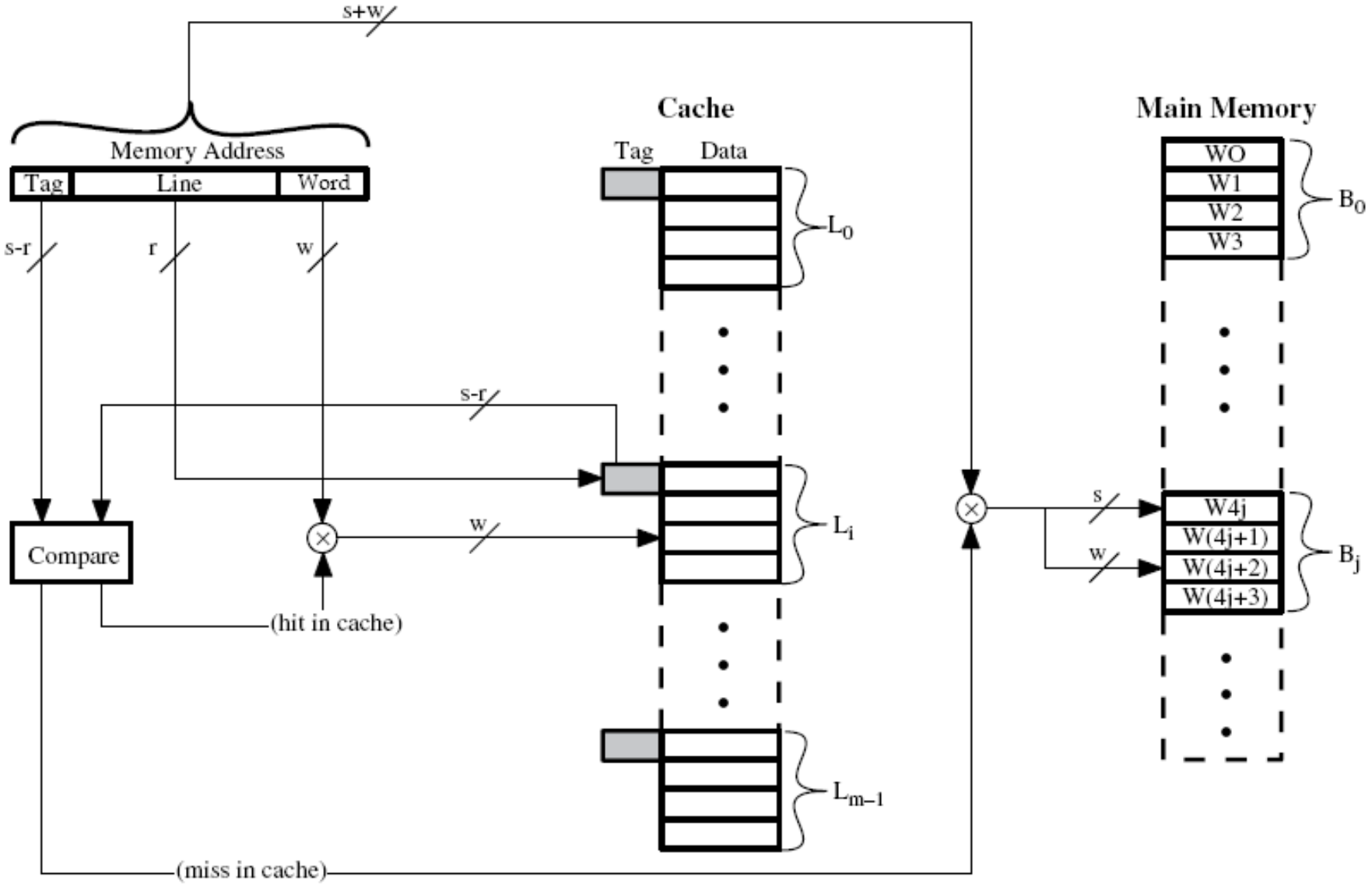


- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag (=22–14)
- 14 bit slot or line
- No two blocks in the same line have the same tag
- Check contents of cache by finding line and comparing tag

Direct Mapping Cache Line Table

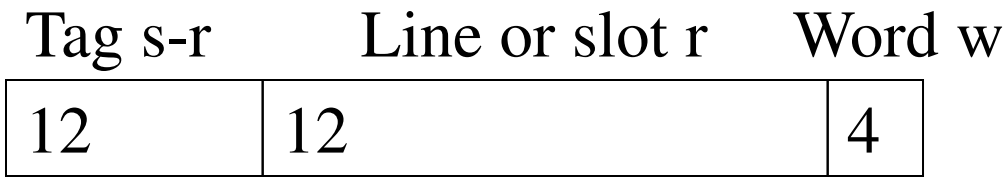
Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2^s–m
1	1, m+1, 2m+1…2^s–m+1
m–1	m–1, 2m–1, 3m–1…2^s–1

Direct Mapping Cache Organization



Direct Mapping Examples

What cache line number will the following addresses be stored to, and what will the minimum address and the maximum address of each block they are in be if we have a cache with 4K lines of 16 words to a block in a 256 Meg memory space (28-bit address)?



a.) $9ABCDEF_{16}$

b.) 1234567_{16}

More Direct Mapping Examples

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Line number (binary)	Addresses wi/ block			
		00	01	10	11
0101 0011	1000 1110 1110 10				
1110 1101	1000 1110 1110 11				
1010 1101	1000 1110 1111 00				
0110 1011	1000 1110 1111 01				
1011 0101	1000 1110 1111 10				
1111 0001	1000 1110 1111 11				

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line width = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits

Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block –
  If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high (thrashing)

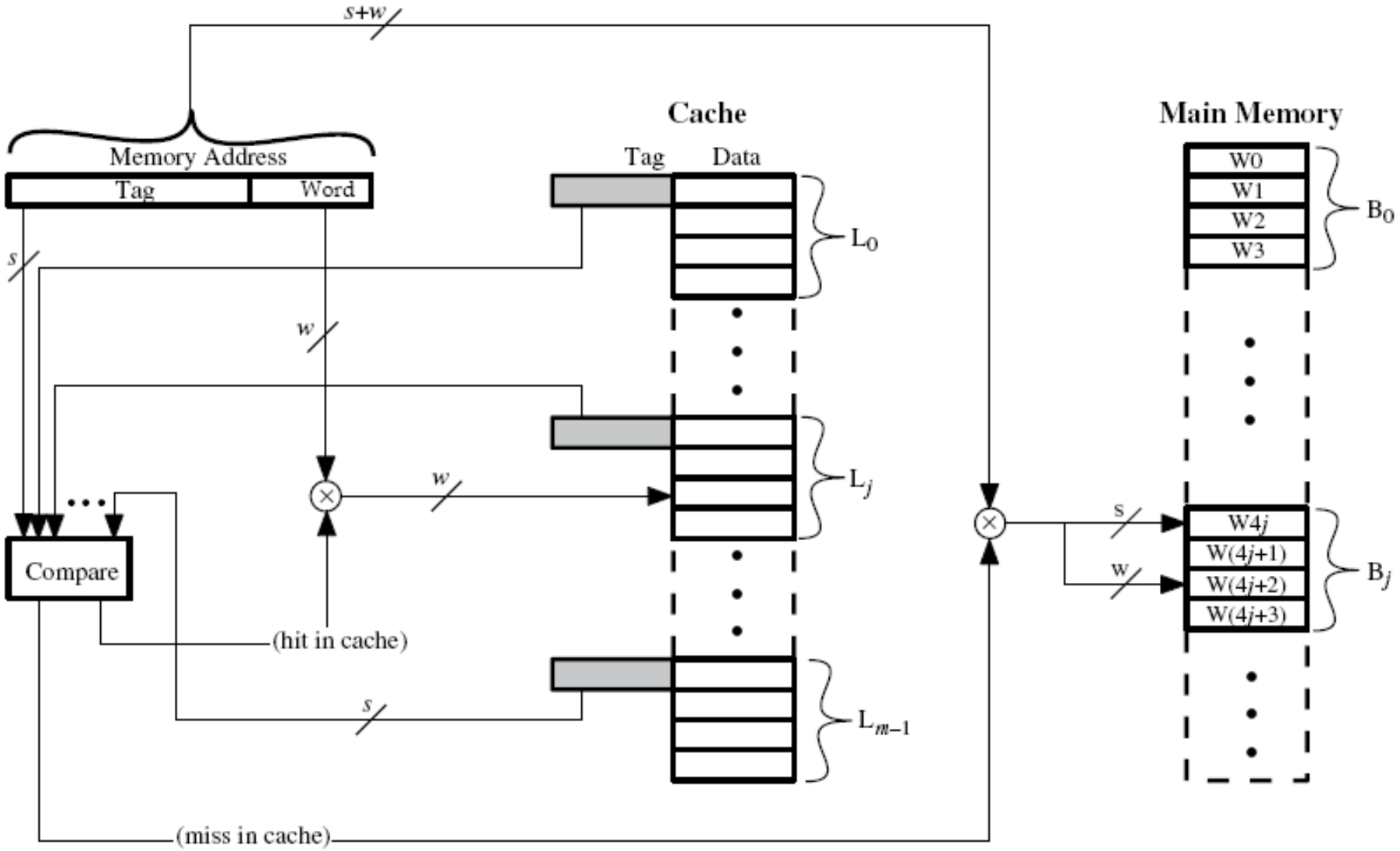
Associative Mapping Traits

- A main memory block can load into any line of cache
- Memory address is interpreted as:
  - Least significant w bits = word position within block
  - Most significant s bits = tag used to identify which block is stored in a particular line of cache
- Every line's tag must be examined for a match
- Cache searching gets expensive and slower

Associative Mapping Address Structure Example

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which of the four 8 bit words is required from 32 bit data block

Fully Associative Cache Organization



Fully Associative Mapping Example

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Addresses wi/ block			
	00	01	10	11
0101 0011 1000 1110 1110 10				
1110 1101 1100 1001 1011 01				
1010 1101 1000 1110 1111 00				
0110 1011 1000 1110 1111 11				
1011 0101 0101 1001 0010 00				
1111 0001 1000 1110 1111 11				

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

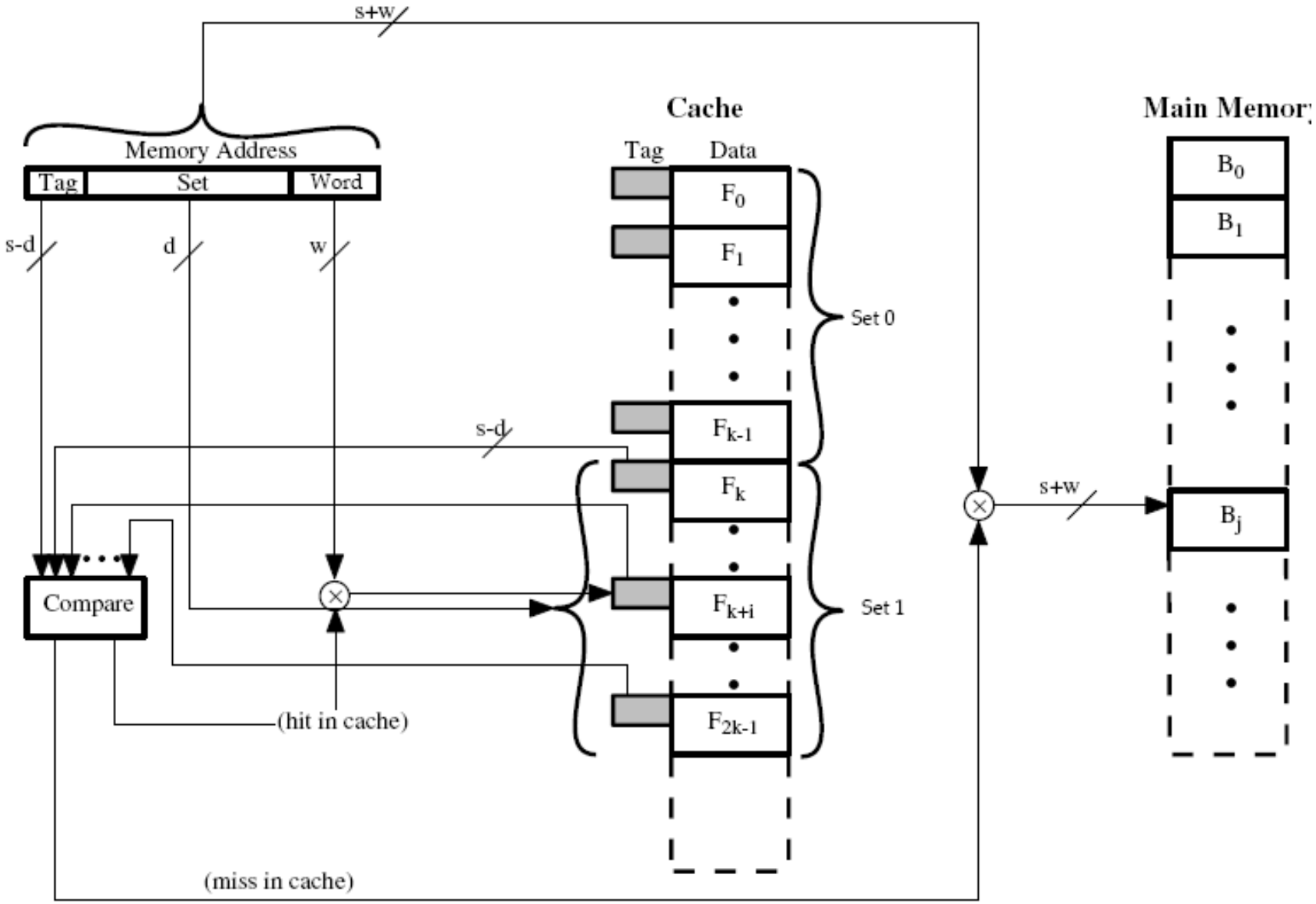
Set Associative Mapping Traits

- Address length is $s + w$ bits
- Cache is divided into a number of sets, $v = 2^d$
- k blocks/lines can be contained within each set
- k lines in a cache is called a k-way set associative mapping
- Number of lines in a cache $= v \bullet k = k \bullet 2^d$
- Size of tag = $(s-d)$ bits

Set Associative Mapping Traits (continued)

- Hybrid of Direct and Associative
  k = 1, this is basically direct mapping
  v = 1, this is associative mapping
- Each set contains a number of lines, basically the number of lines divided by the number of sets
- A given block maps to any line within its specified set – e.g. Block B can be in any line of set i.
- 2 lines per set is the most common organization.
  - Called 2 way associative mapping
  - A given block can be in one of 2 lines in only one specific set
  - Significant improvement over direct mapping

K-Way Set Associative Cache Organization



How does this affect our example?

- Let's go to two-way set associative mapping
- Divides the 16K lines into 8K sets
- This requires a 13 bit set number
- With 2 word bits, this leaves 9 bits for the tag
- Blocks beginning with the addresses 000000_{16}, 008000_{16}, 010000_{16}, 018000_{16}, 020000_{16}, 028000_{16}, etc. map to the same set, Set 0.
- Blocks beginning with the addresses 00000416, 008004_{16}, 010004_{16}, 018004_{16}, 020004_{16}, 028004_{16}, etc. map to the same set, Set 1.

Set Associative Mapping Address Structure

Tag 9 bits	Set 13 bits	Word 2 bits

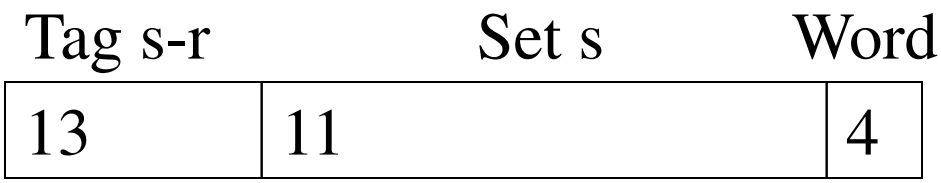
- Note that there is one more bit in the tag than for this same example using direct mapping.
- Therefore, it is 2-way set associative
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit

Set Associative Mapping Example

For each of the following addresses, answer the following questions based on a 2-way set associative cache with 4K lines, each line containing 16 words, with the main memory of size 256 Meg memory space (28-bit address):

- What cache set number will the block be stored to?
- What will their tag be?
- What will the minimum address and the maximum address of each block they are in be?

1. $9ABCDEF_{16}$
2. 1234567_{16}



Set Associative Mapping Summary

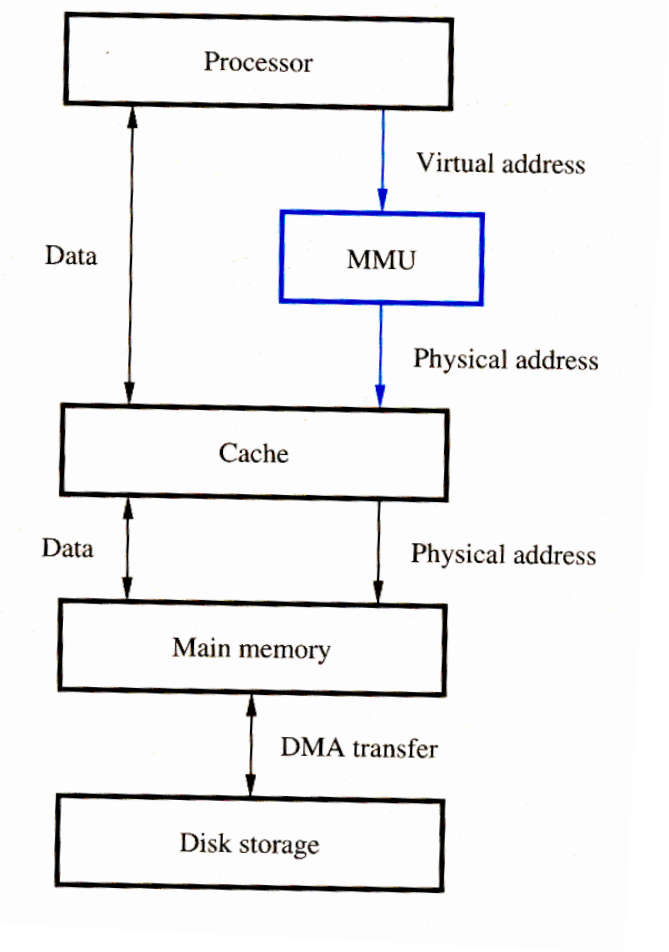
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2^{d}
- Size of tag = (s – d) bits

Virtual Memory

- In modern computers it is possible to use more memory than the amount physically available in the system
- Memory not currently being used is temporarily stored on magnetic disk
- Essentially, the *main memory acts as a* ***cache*** for the virtual memory on disk.

Memory Management Unit (MMU)

- Virtual memory must be invisible to the CPU
  - Appears as one large address space



- The MMU sits between the CPU and the memory system
- Translates virtual addresses to real (physical) addresses