Memory and Its Organization in Computing Machines

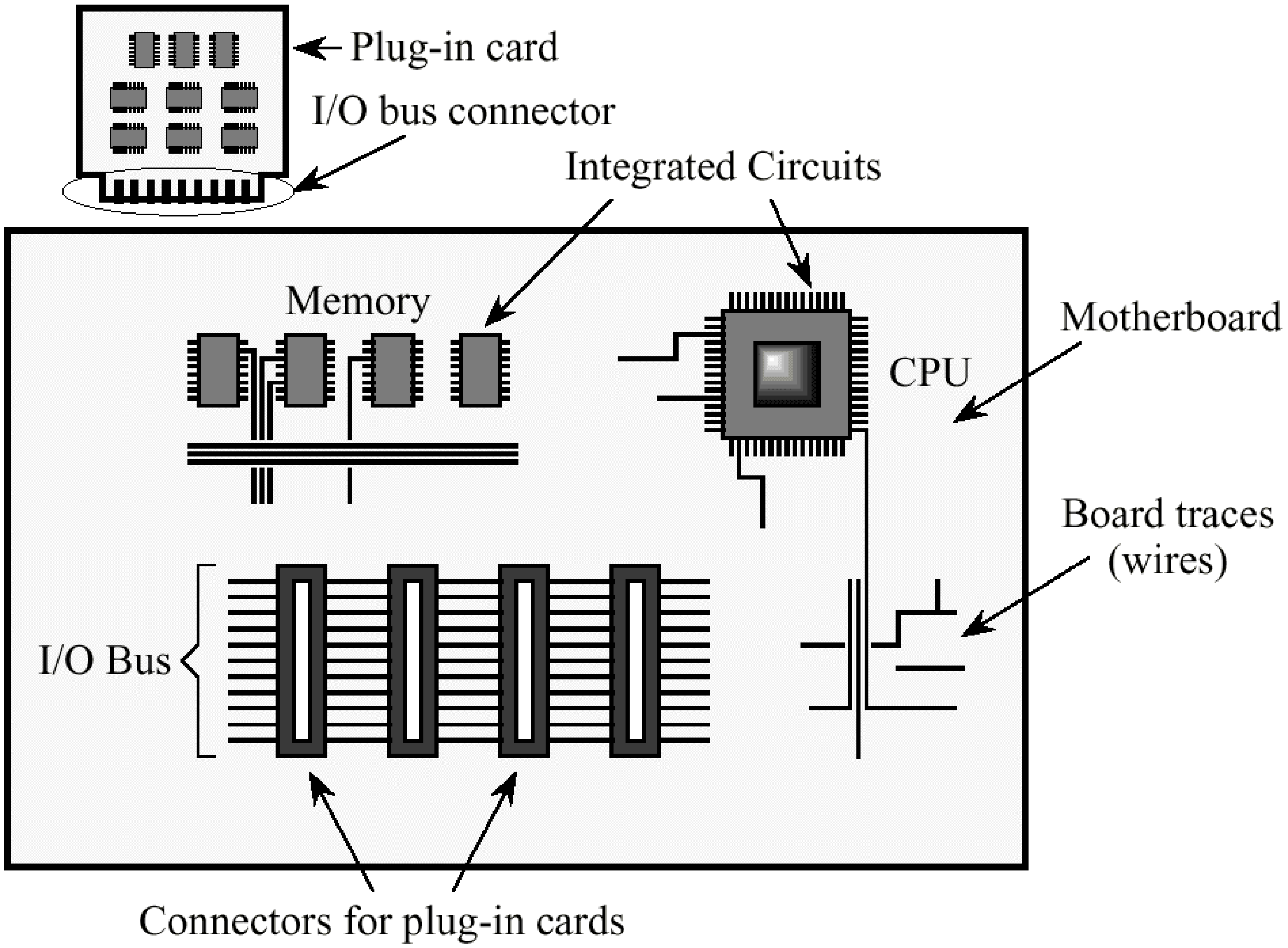
Buses, Bridged Architectures and Cache Basics

Objectives

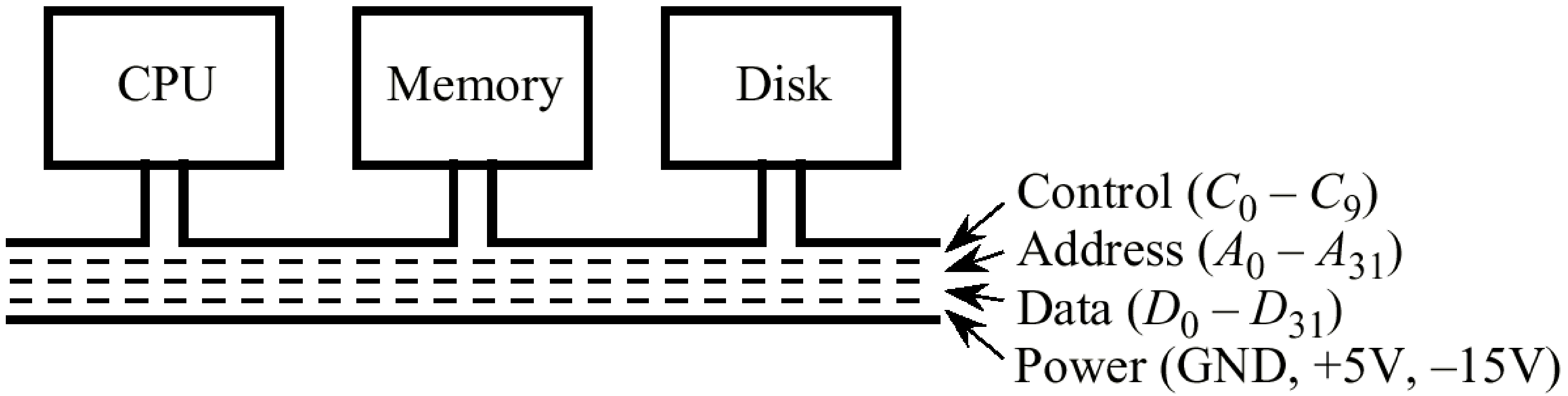
- After this lecture, you should be able to…….
  - **Explain the requirement for and structure of computer buses**
  - **Interpret waveform diagrams for synchronous and asynchronous bus transactions**
  - **Indicate 3 kinds of bus arbitration**
  - **Describe 3 different methods of communication used for data transfer operations in computers**
  - **Indicate the hierarchy of different memory elements in a computer**
  - **Identify the structures associated with RAM and ROM and cache memory**
  - **Determine performance of different cache arrangements**

Simple Bus Architecture

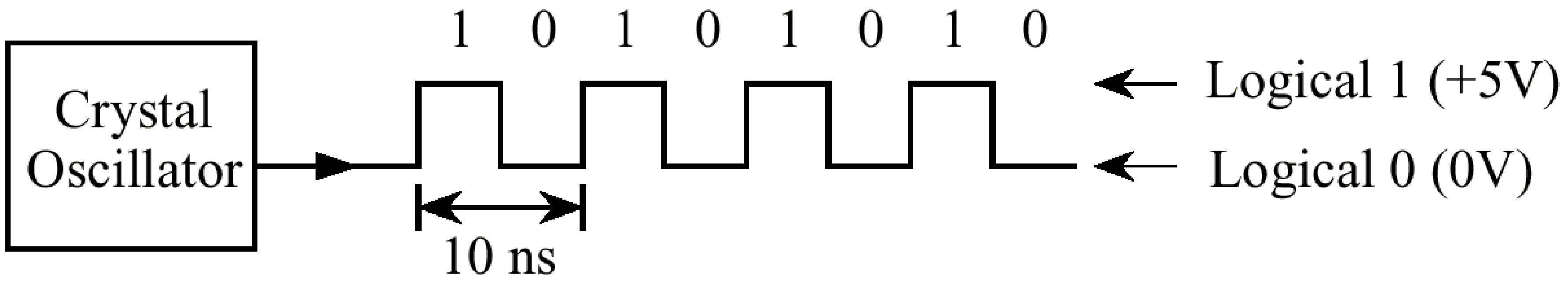
- A simplified motherboard of a personal computer (top view):



Simplified Illustration of a Bus

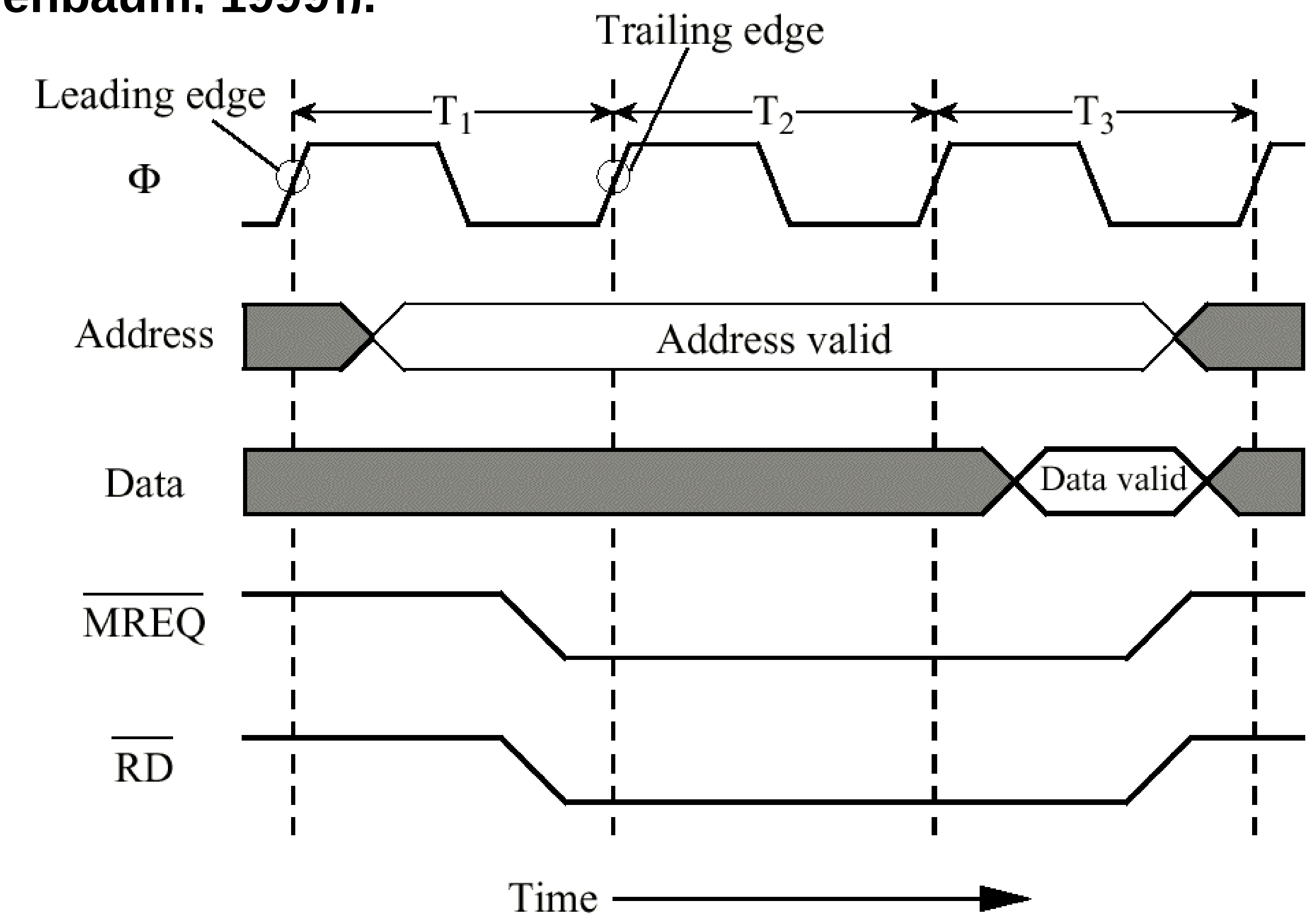


100 MHz Bus Clock



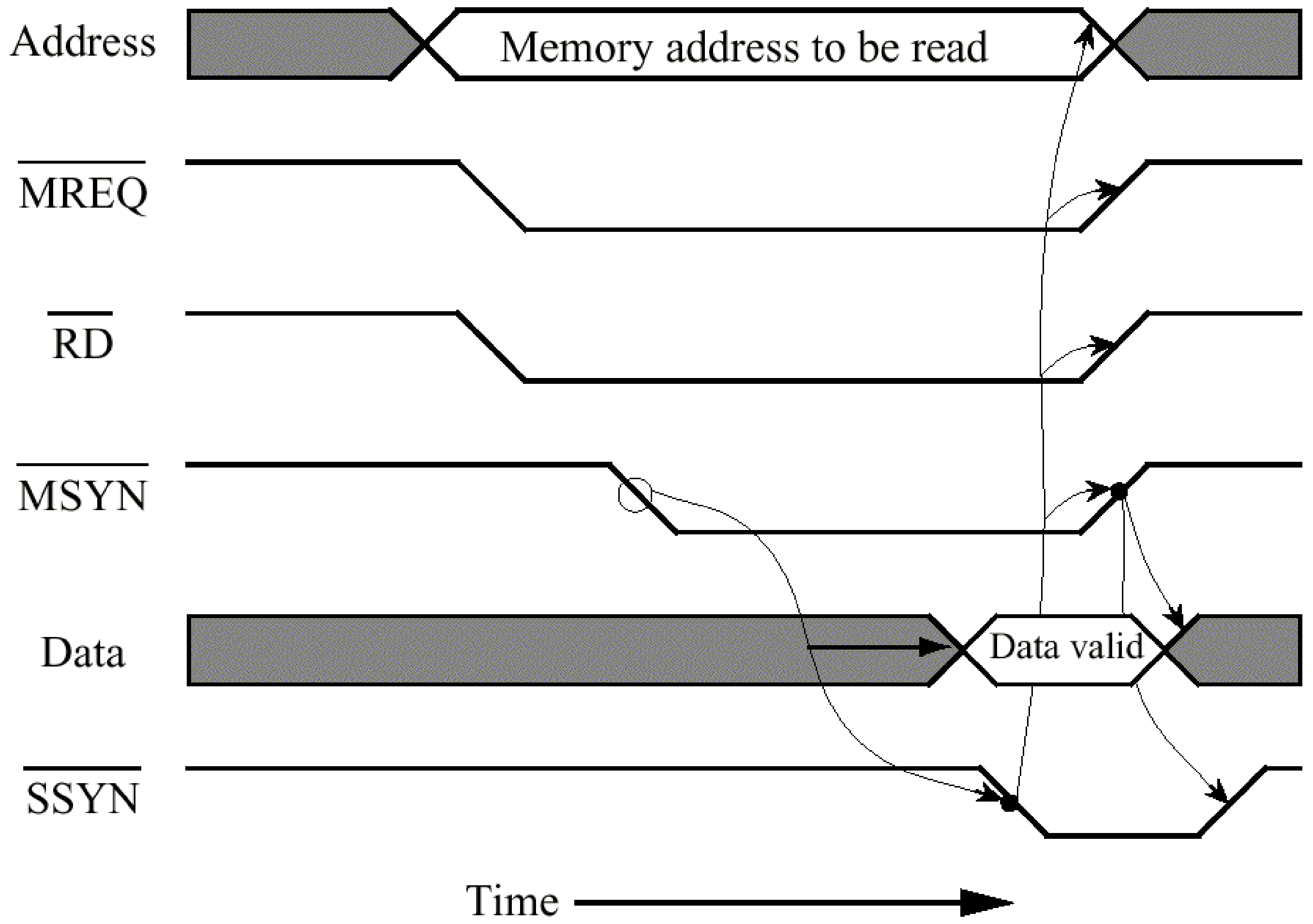
The Synchronous Bus

- **Timing diagram for a synchronous memory read (adapted from [Tanenbaum, 1999]).**



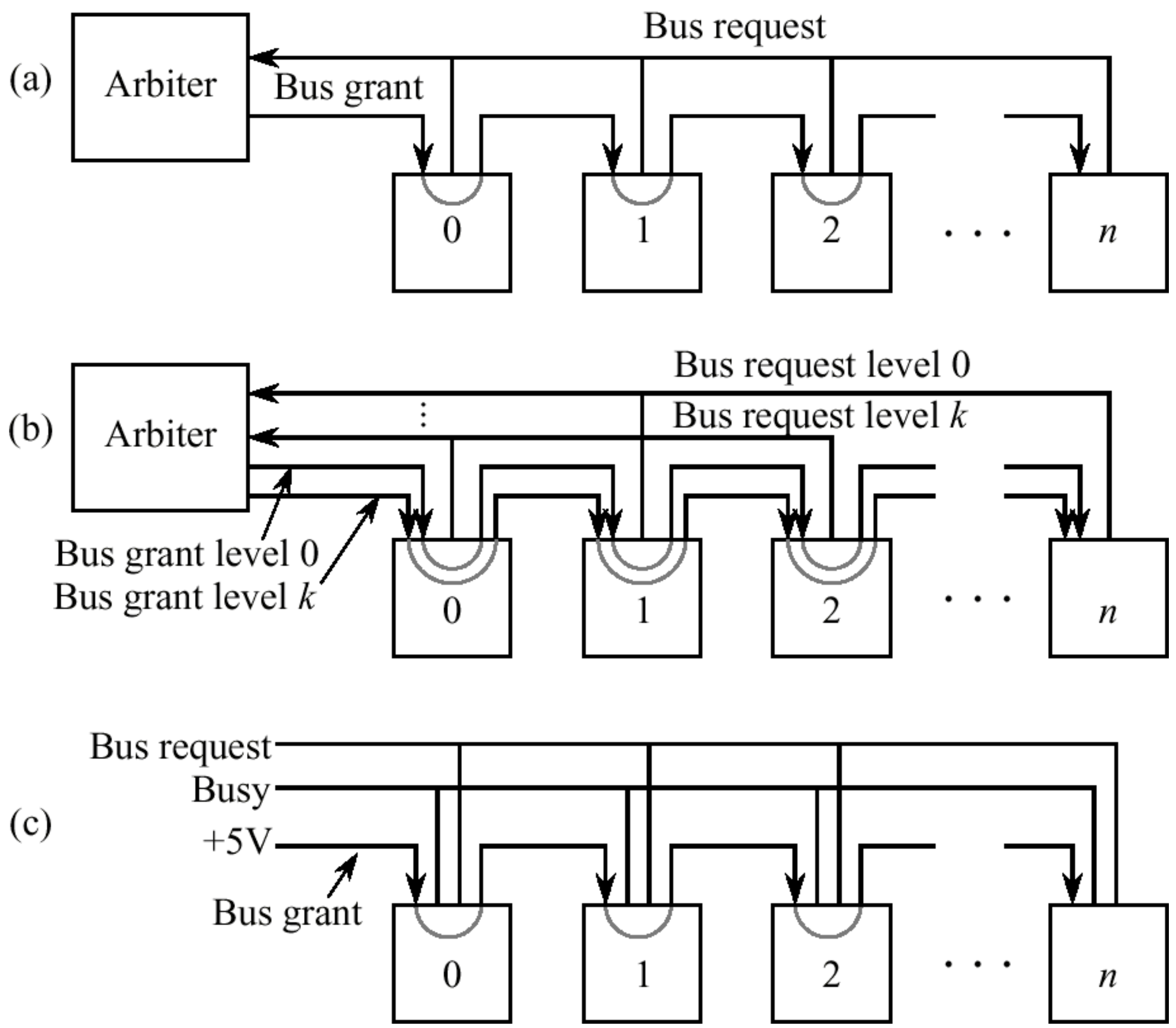
The Asynchronous Bus

- **Timing diagram for asynchronous memory read (adapted from [Tanenbaum, 1999]).**



Bus Arbitration

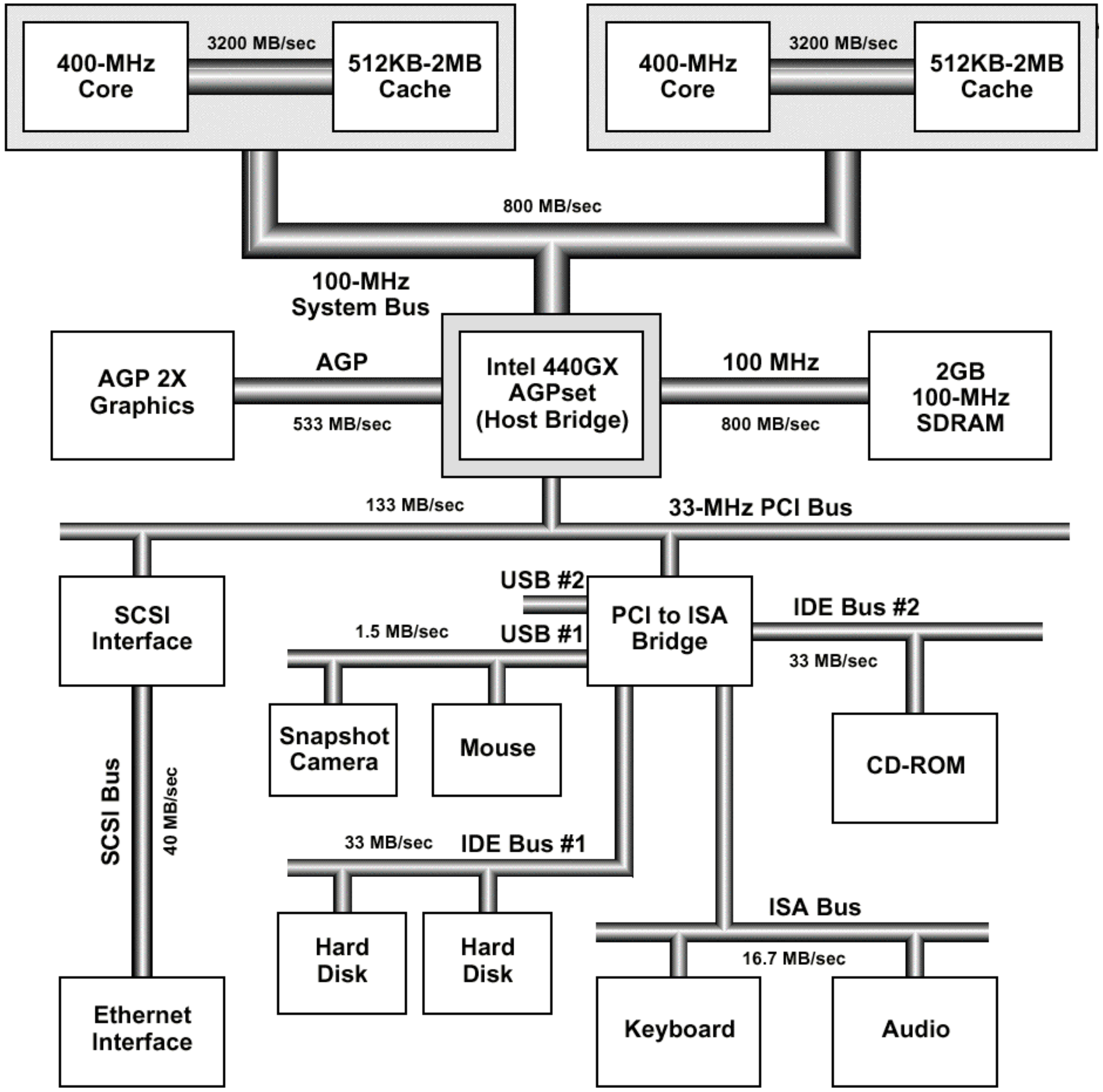
- **(a)Simple centralized bus arbitration; (b) centralized arbitration with priority levels; (c) decentralized bus arbitration. (Adapted from [Tanenbaum, 1999]).**



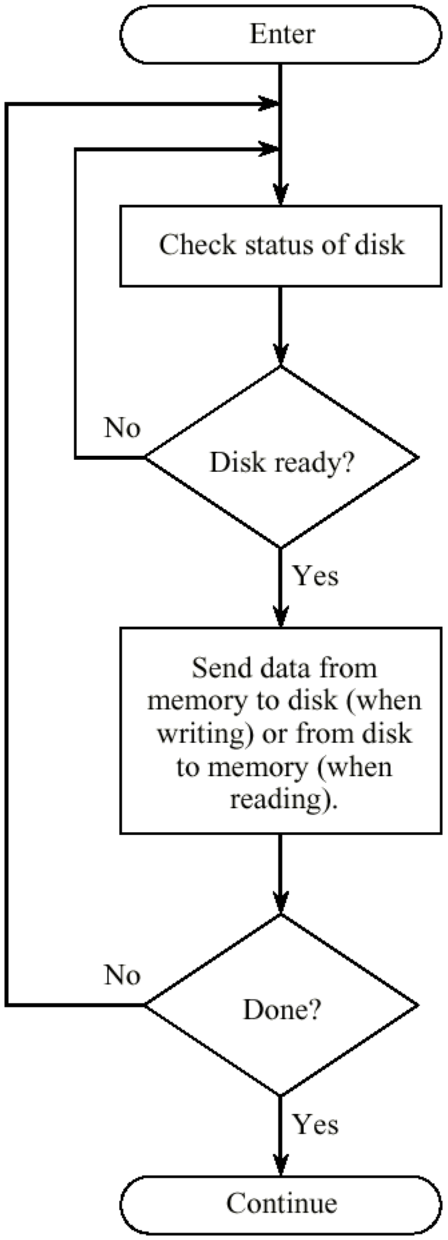
Bridge Based Bus Architecture

- Bridging with dual Pentium II Xeon processors on Slot 2.

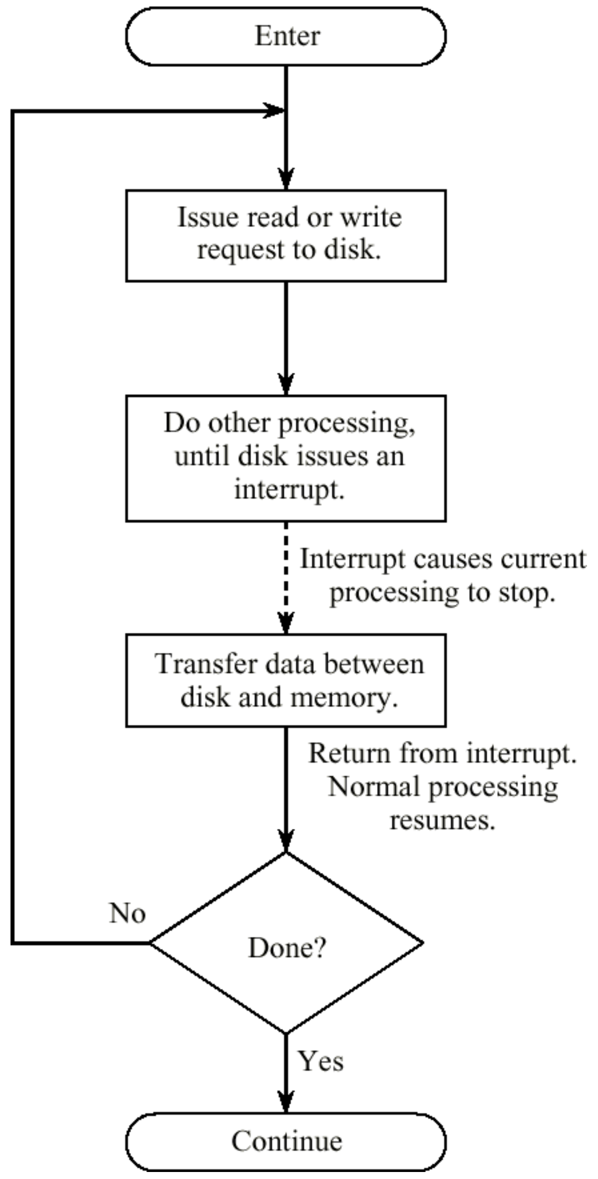
(Source: http://www.intel.com.)



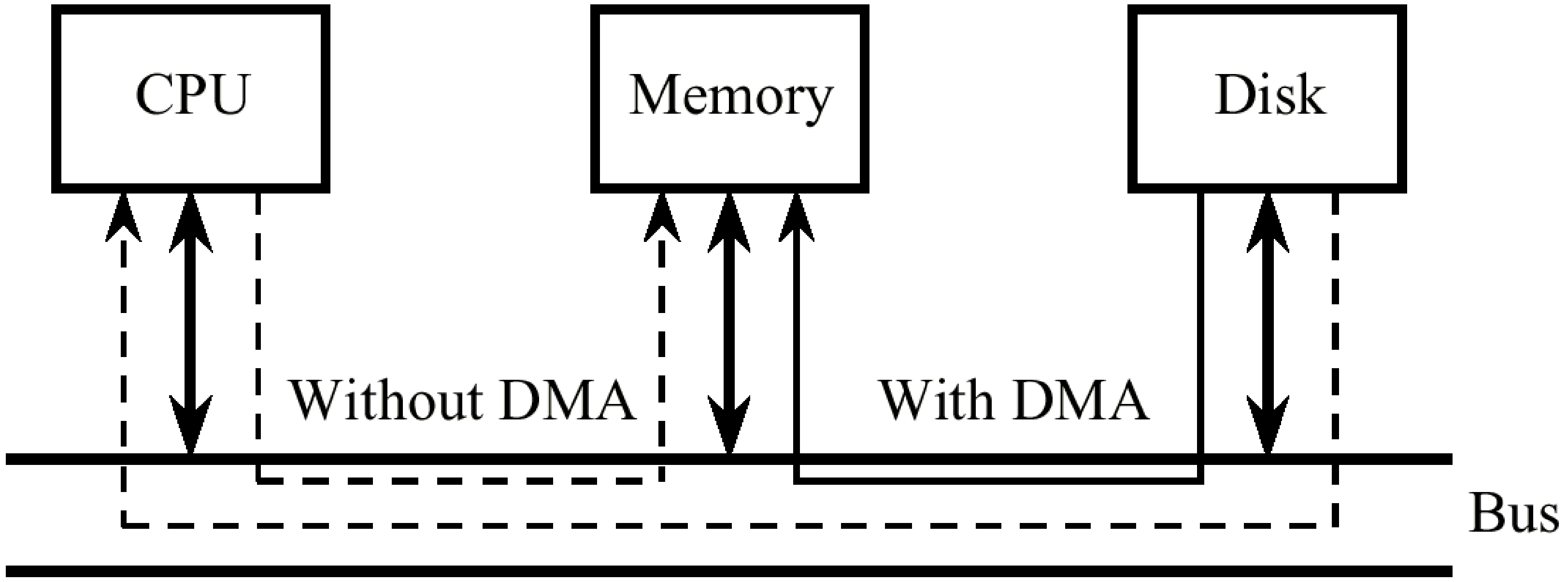
Programmed I/O Flowchart for a Disk Transfer


Enter
Check status of disk
No
Disk ready?
Yes
Send data from memory to disk (when writing) or from disk to memory (when reading).
No
Done?
Yes
Continue

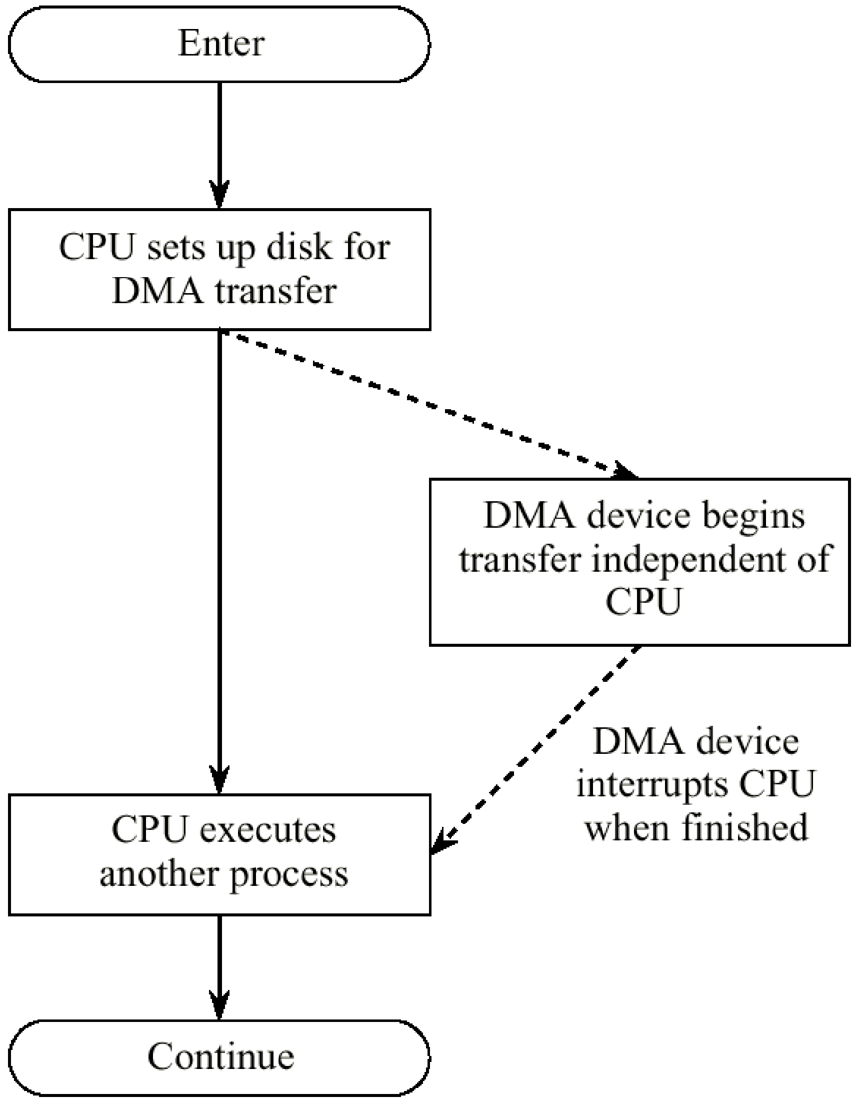
Interrupt Driven I/O Flowchart for a Disk Transfer


Enter
Issue read or write request to disk.
Do other processing, until disk issues an interrupt.
Interrupt causes current processing to stop.
Transfer data between disk and memory.
Return from interrupt. Normal processing resumes.
Done?
No
Yes
Continue

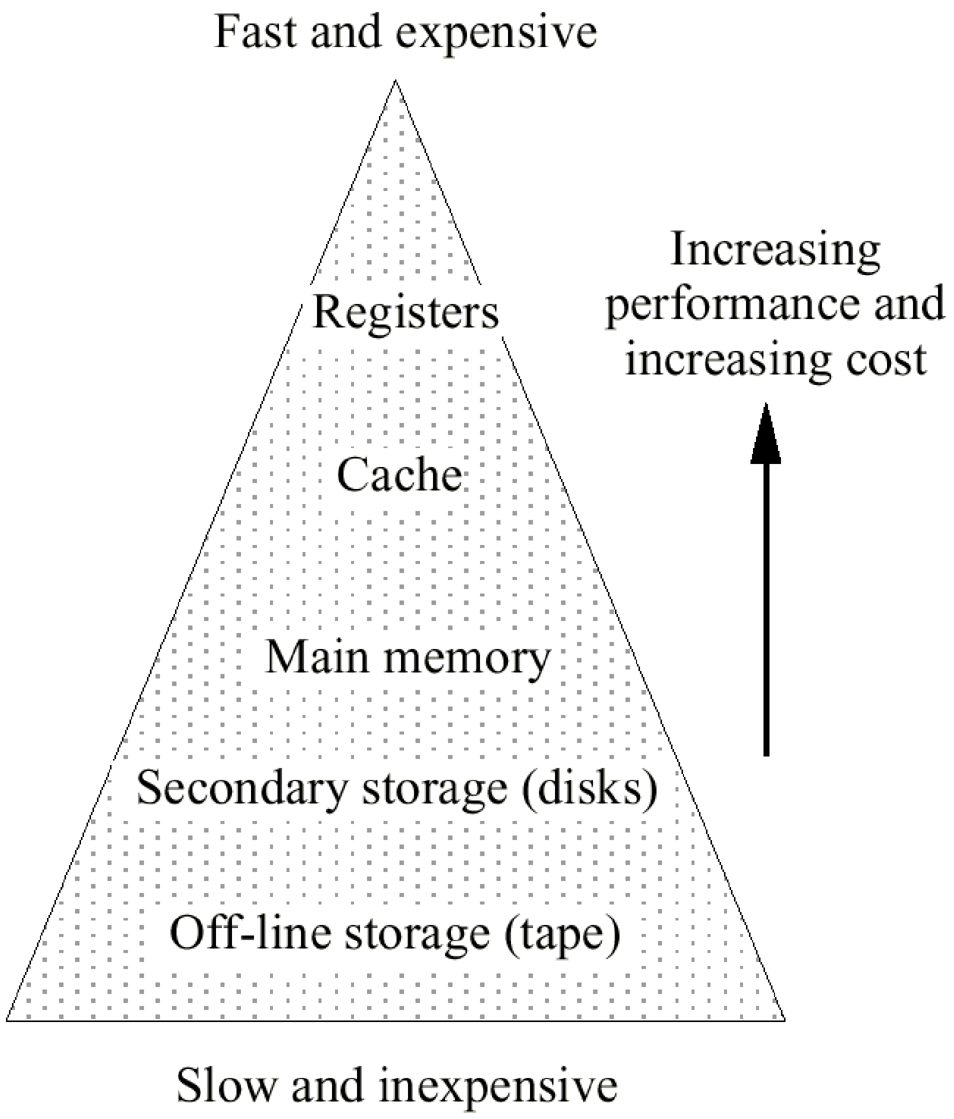
DMA Transfer from Disk to Memory Bypasses the CPU



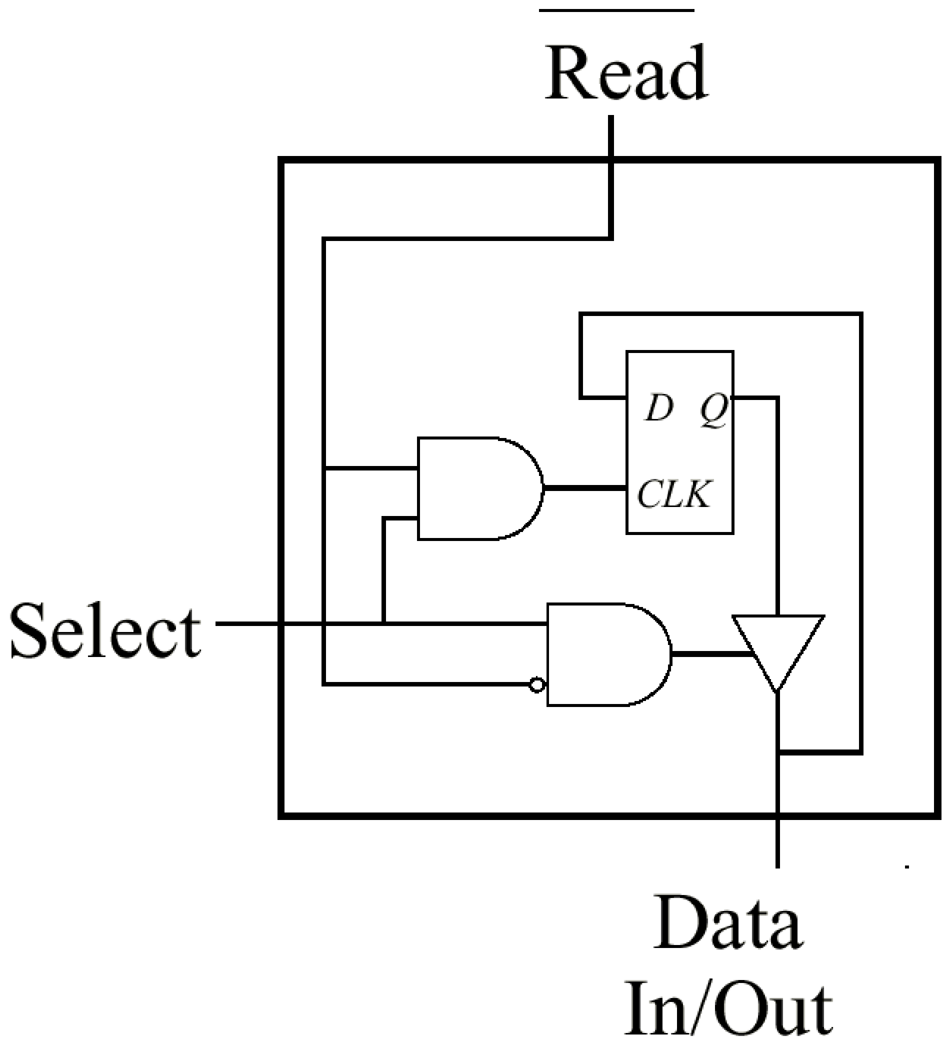
DMA Flowchart for a Disk Transfer


Enter
CPU sets up disk for DMA transfer
DMA device begins transfer independent of CPU
DMA device interrupts CPU when finished
CPU executes another process
Continue

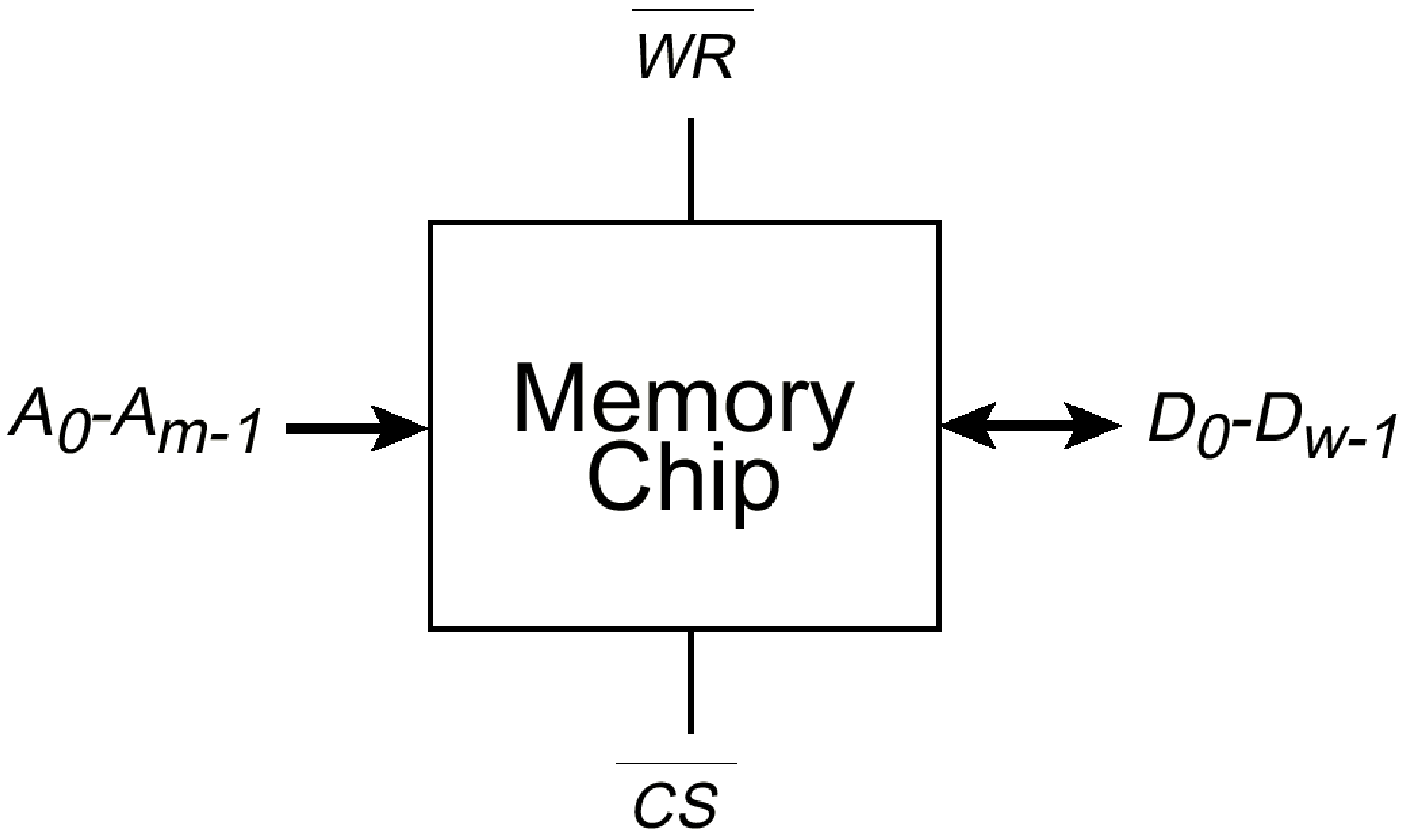
The Memory Hierarchy



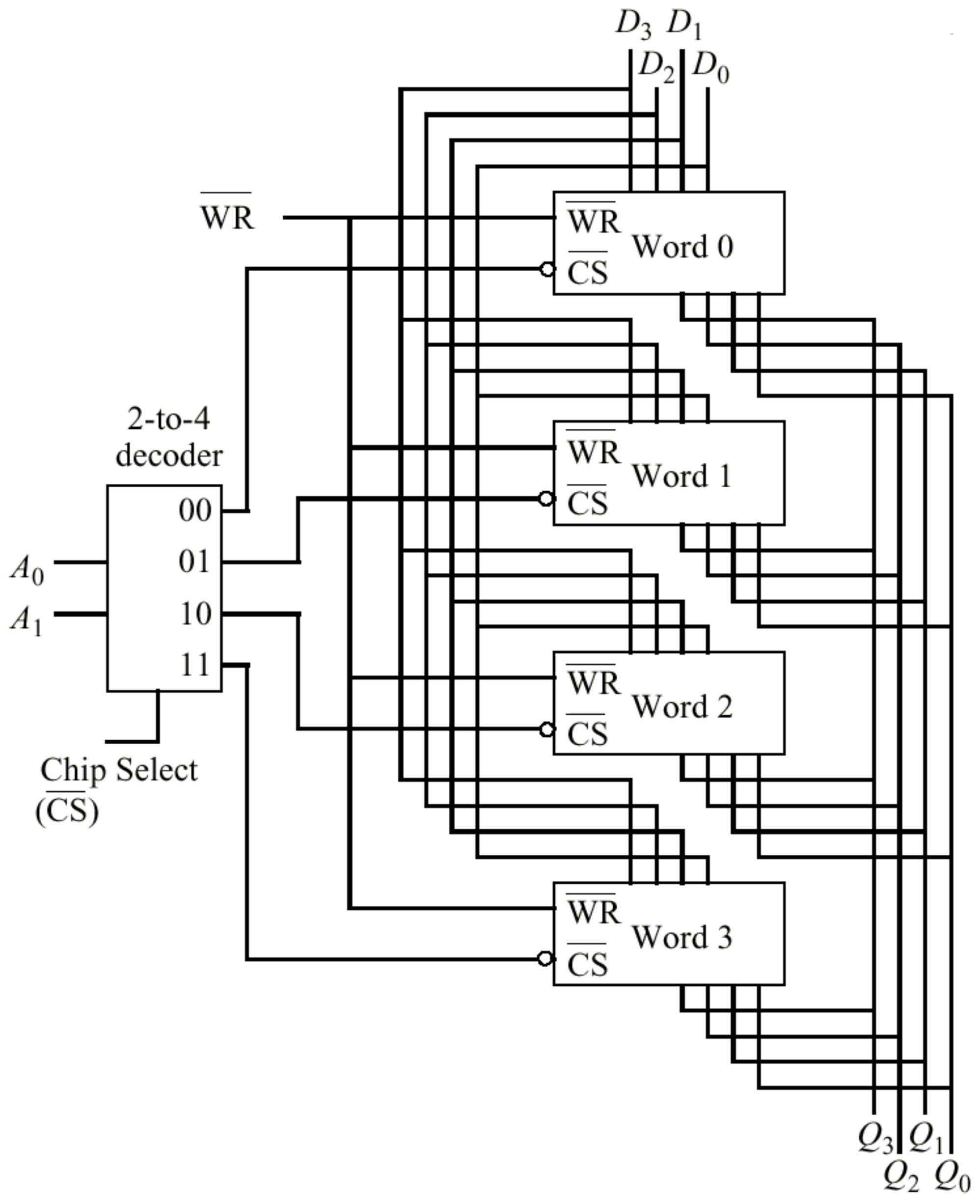
Functional Behavior of a RAM Cell



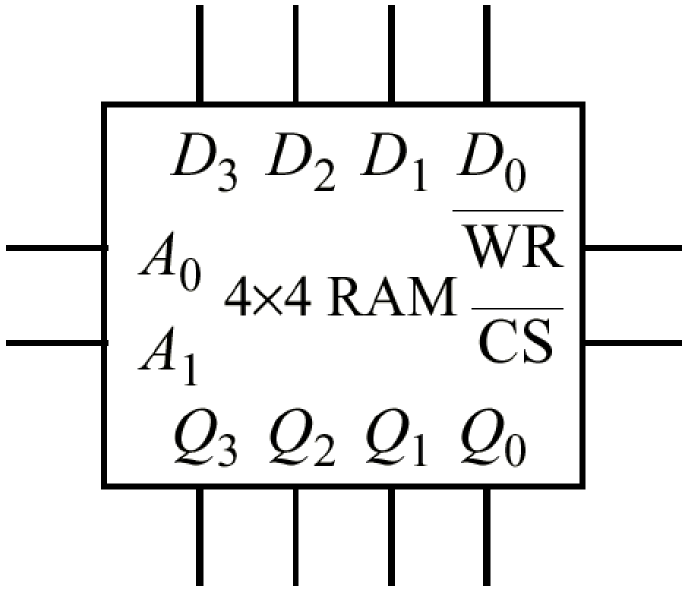
Simplified RAM Chip Pinout



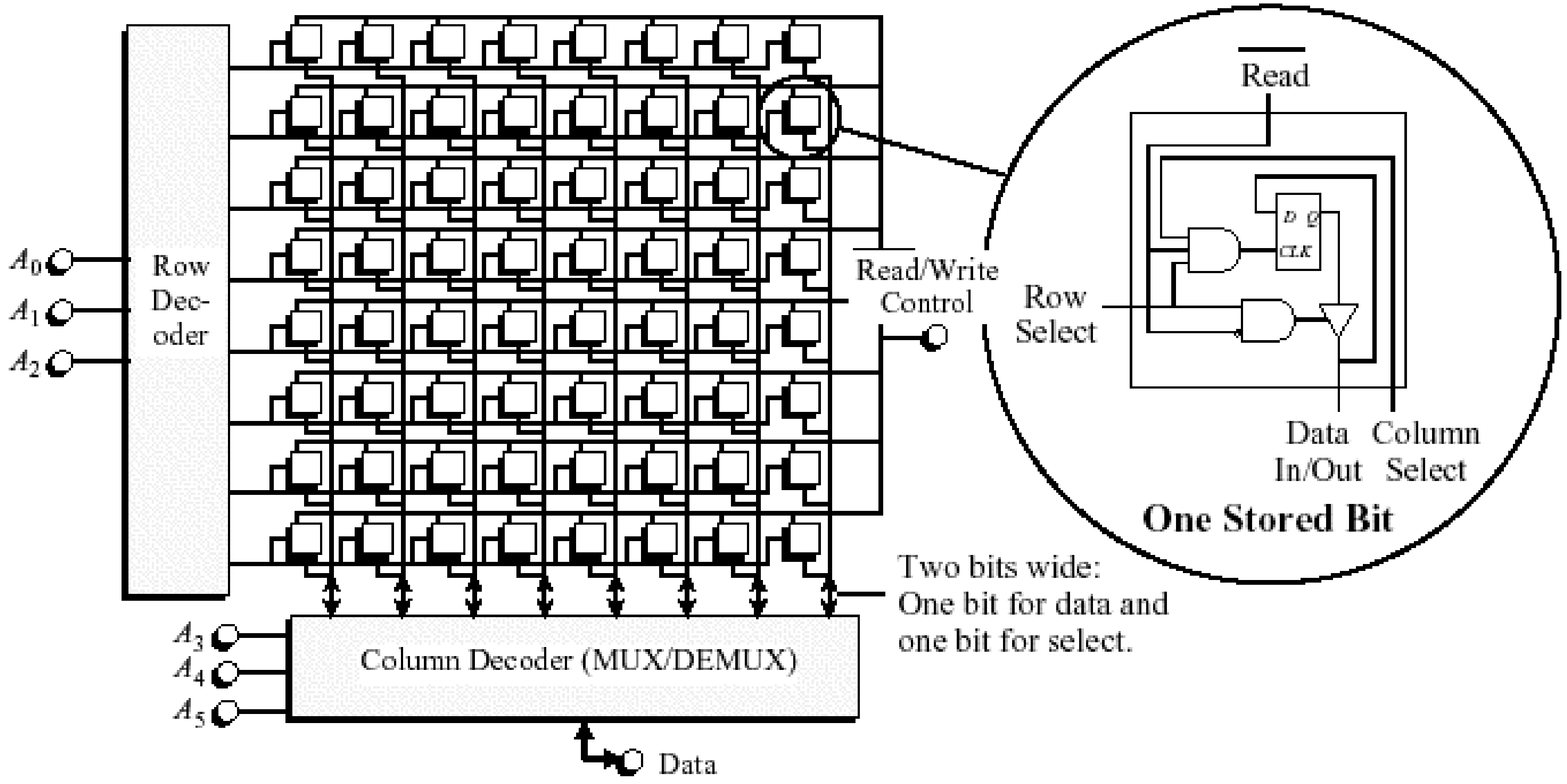
A Four-Word Memory with Four Bits per Word in a 2D Organization


D_3 D_1
D_2 D_0
$\overline{WR}$
$\overline{WR}$ Word 0 $\overline{CS}$
$\overline{WR}$ Word 1 $\overline{CS}$
$\overline{WR}$ Word 2 $\overline{CS}$
$\overline{WR}$ Word 3 $\overline{CS}$
2-to-4 decoder
00
01
10
11
A_0
A_1
Chip Select ($\overline{CS}$)
Q_3 Q_1
Q_2 Q_0

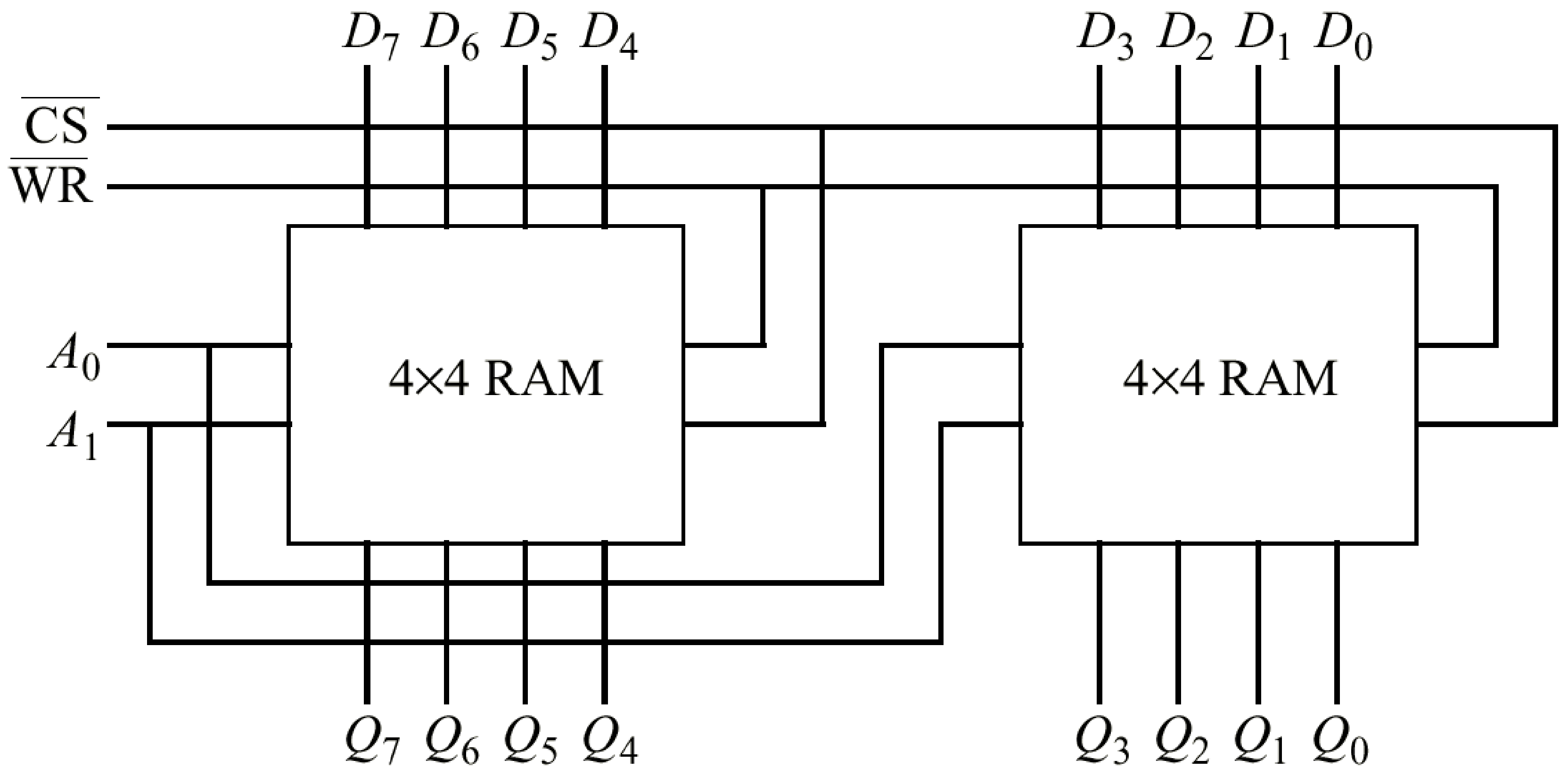
A Simplified Representation of the Four-Word by Four-Bit RAM



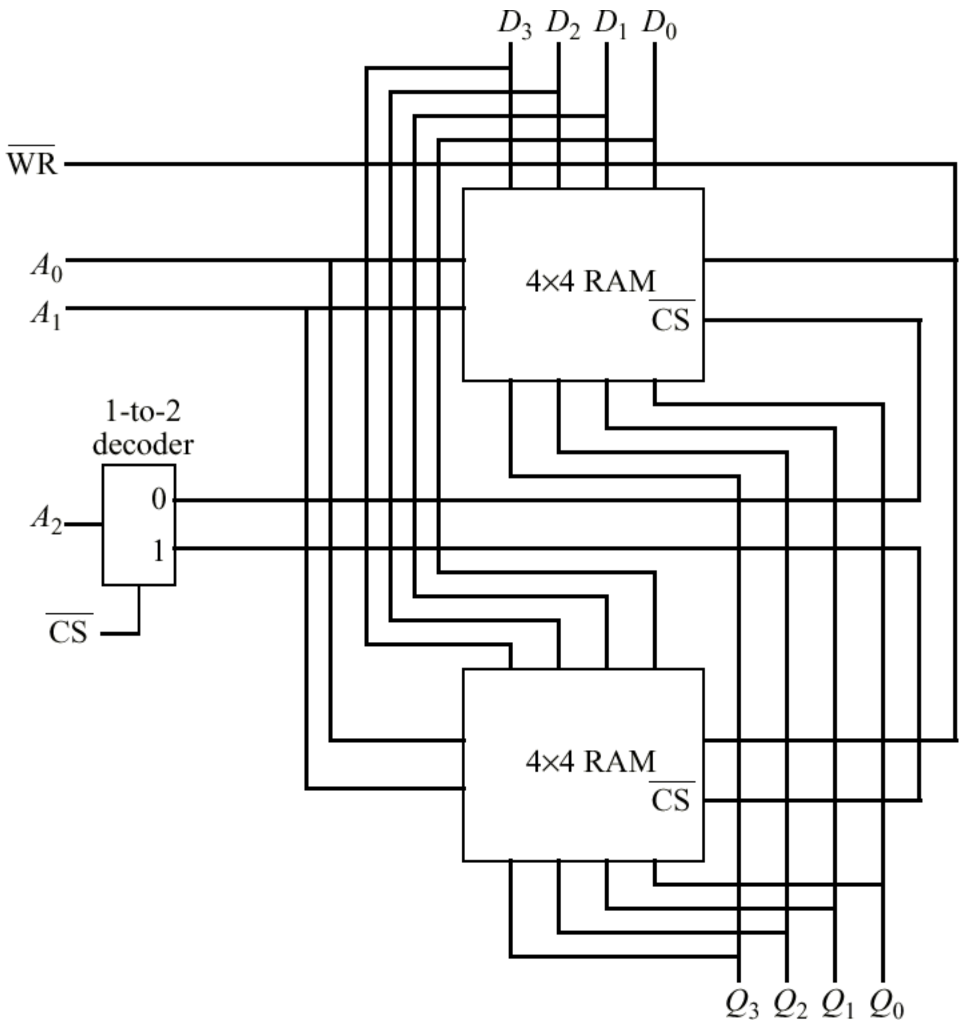
2-1/2D Organization of a 64-Word by One-Bit RAM



Two Four-Word by Four-Bit RAMs are Used in Creating a Four-Word by Eight-Bit RAM



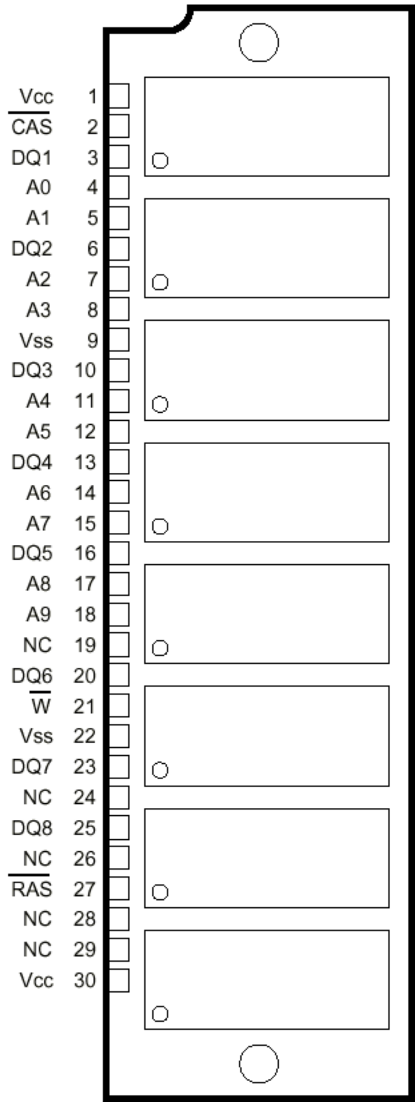
Two Four-Word by Four-Bit RAMs Make up an Eight-Word by Four-Bit RAM



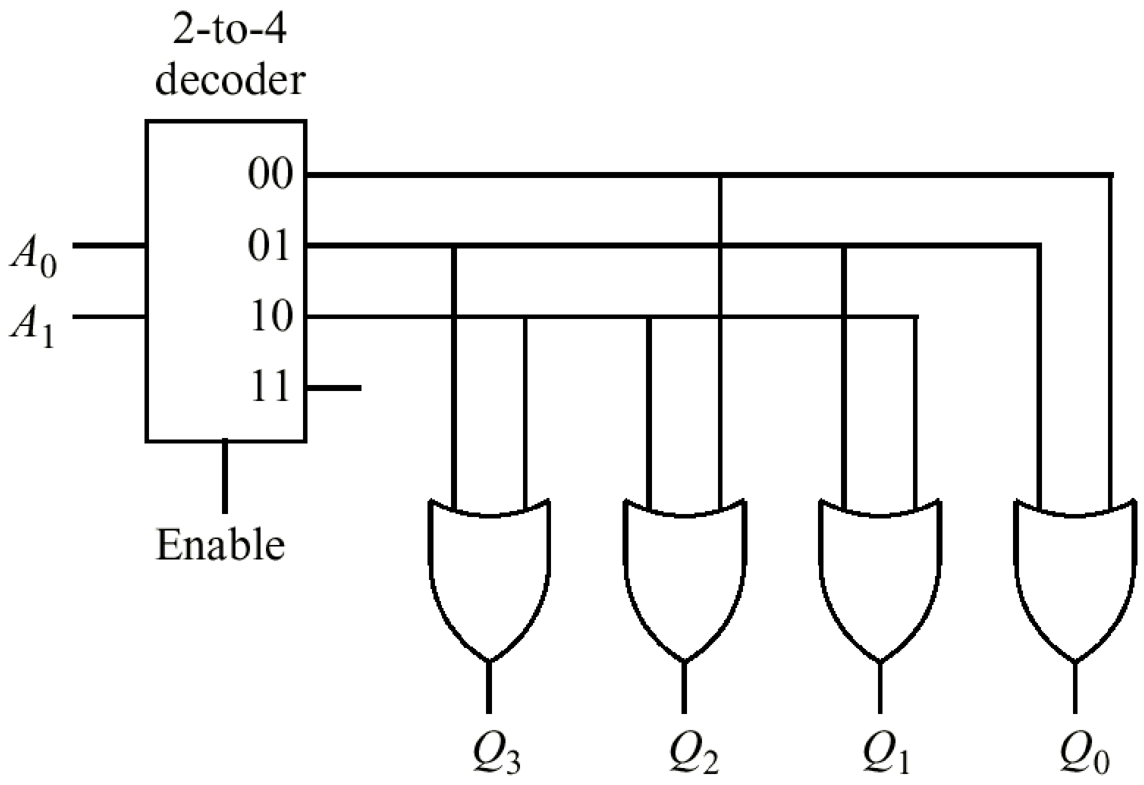
Single-In-Line Memory Module

- **Adapted from(Texas Instruments, *MOS Memory: Commercial and Military Specifications Data Book*, Texas Instruments, Literature Response Center, P.O. Box 172228, Denver, Colorado, 1991.)**

PIN NOMENCLATURE	
A0-A9	Address Inputs
$\overline{\text{CAS}}$	Column-Address Strobe
DQ1-DQ8	Data In/Data Out
NC	No Connection
$\overline{\text{RAS}}$	Row-Address Strobe
V_{CC}	5-V Supply
V_{SS}	Ground
$\overline{\text{W}}$	Write Enable

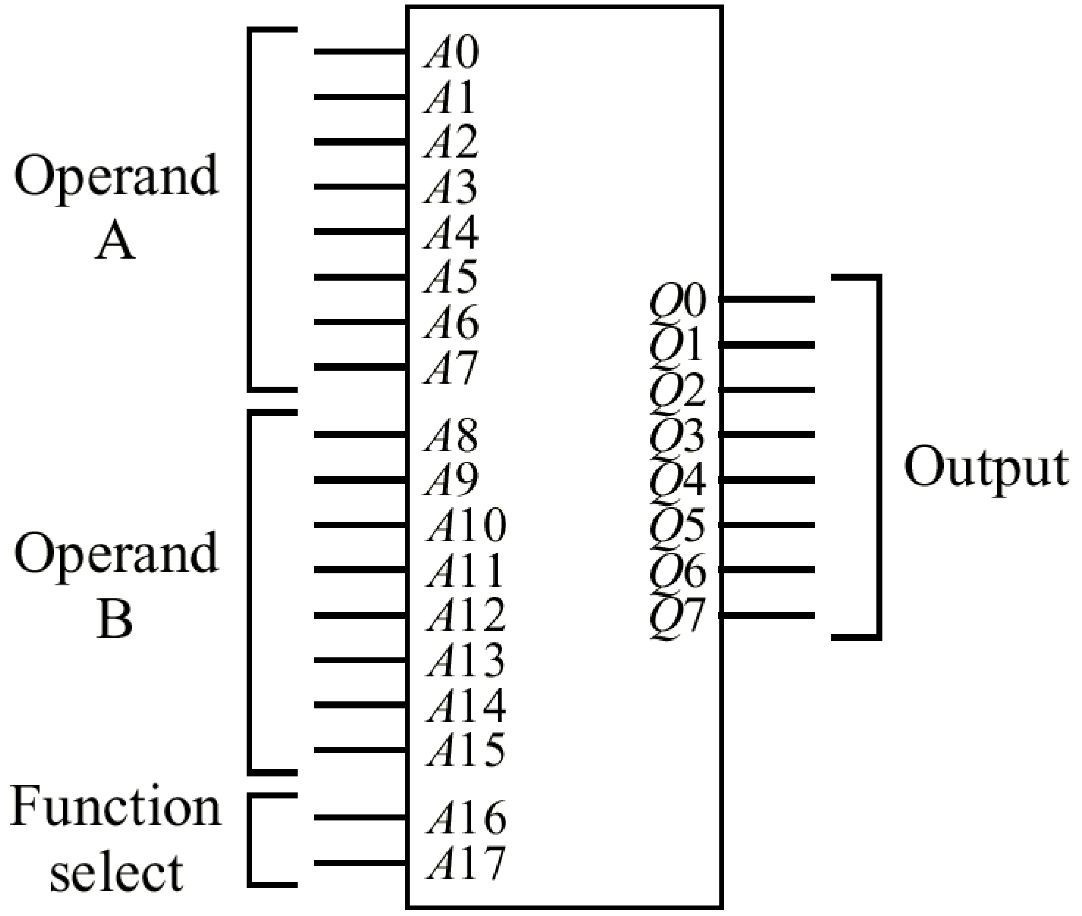


A ROM Stores Four Four-Bit Words



Location	Stored word
00	0101
01	1011
10	1110
11	0000

A Lookup Table (LUT) Implements an Eight-Bit ALU



A17	A16	Function
0	0	Add
0	1	Subtract
1	0	Multiply
1	1	Divide

Useful Info on Caches

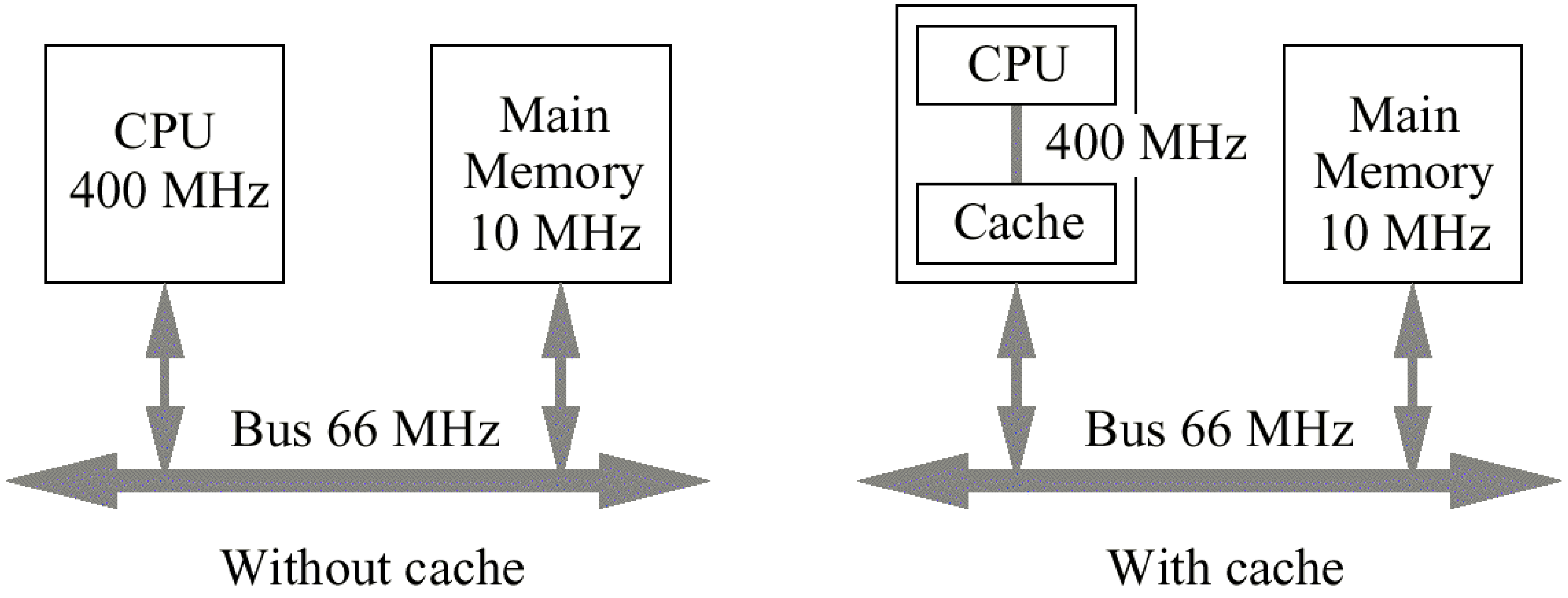
- Here is a useful source of learning material associated with the operation of cache memory under the following URL:

  http://www.cs.iastate.edu/~prabhu/Tutorial/title.html

- Gurpur Prabhu (Iowa State University) has devised a website that recounts aspects of the use of cache memory resources on computers. There are useful animations of the operation and data replacement policies adopted by differently organized computer caches.

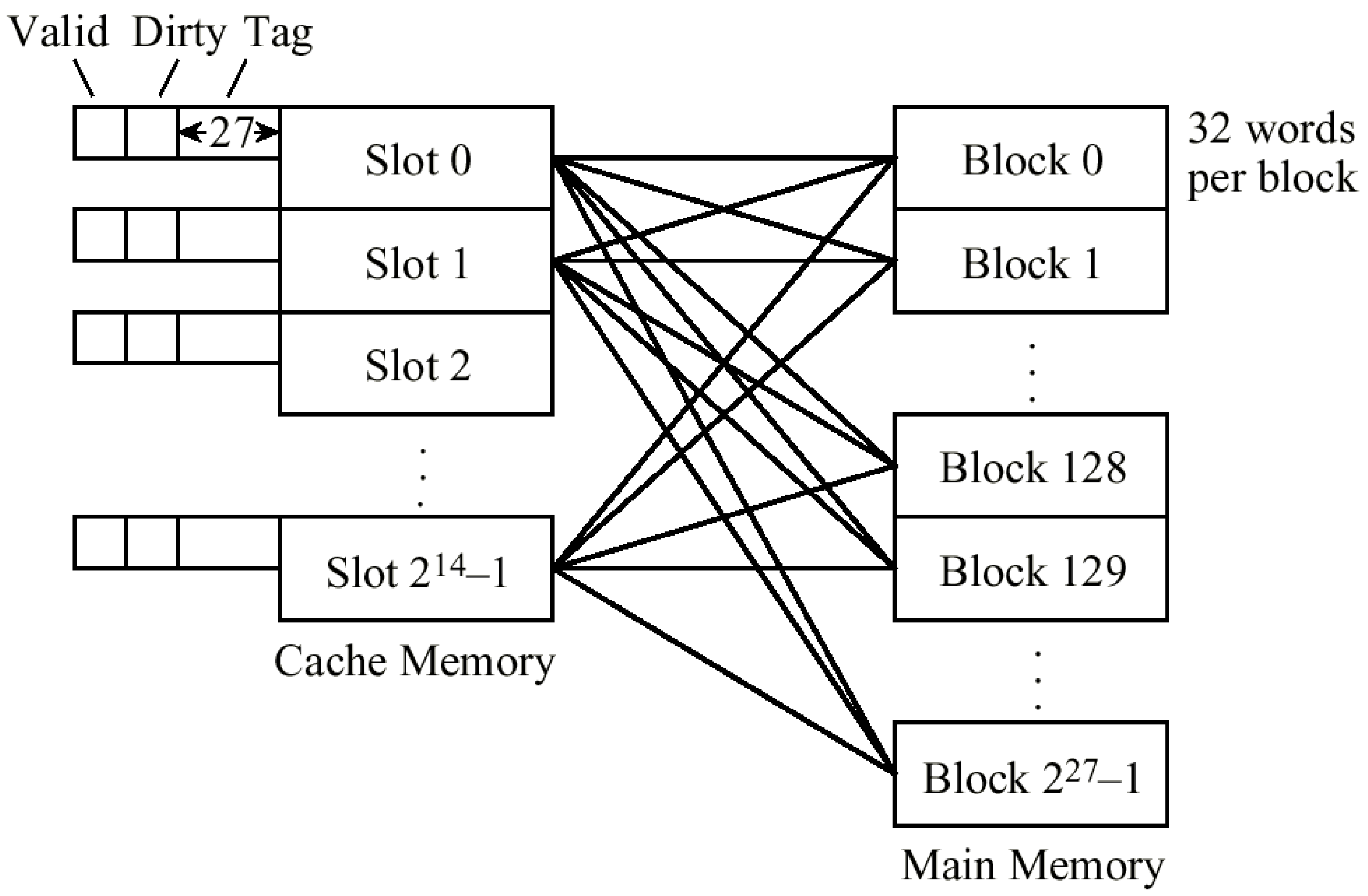
- You are encouraged to review this website and to refer to the relevant sections of the course textbook (sections 7.1 through 7.6)

Placement of Cache in a Computer System



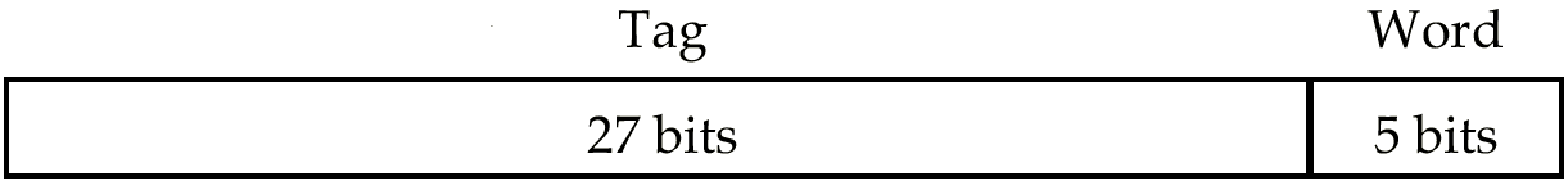
- The *locality principle*: a recently referenced memory location is likely to be referenced again (*temporal locality*); a neighbor of a recently referenced memory location is likely to be referenced (*spatial locality*).

An Associative Mapping Scheme for a Cache Memory

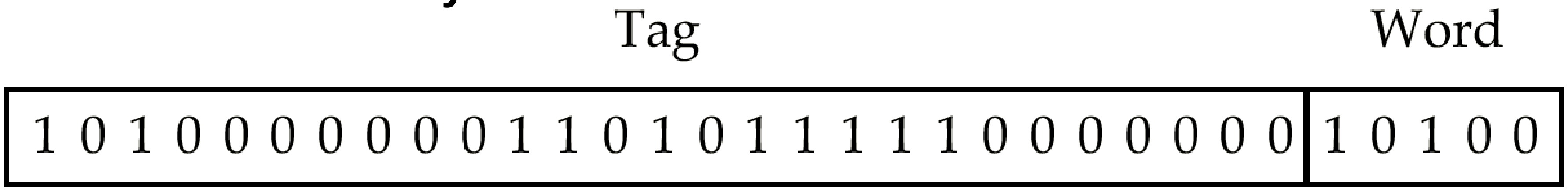


Associative Mapping Example

- Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:



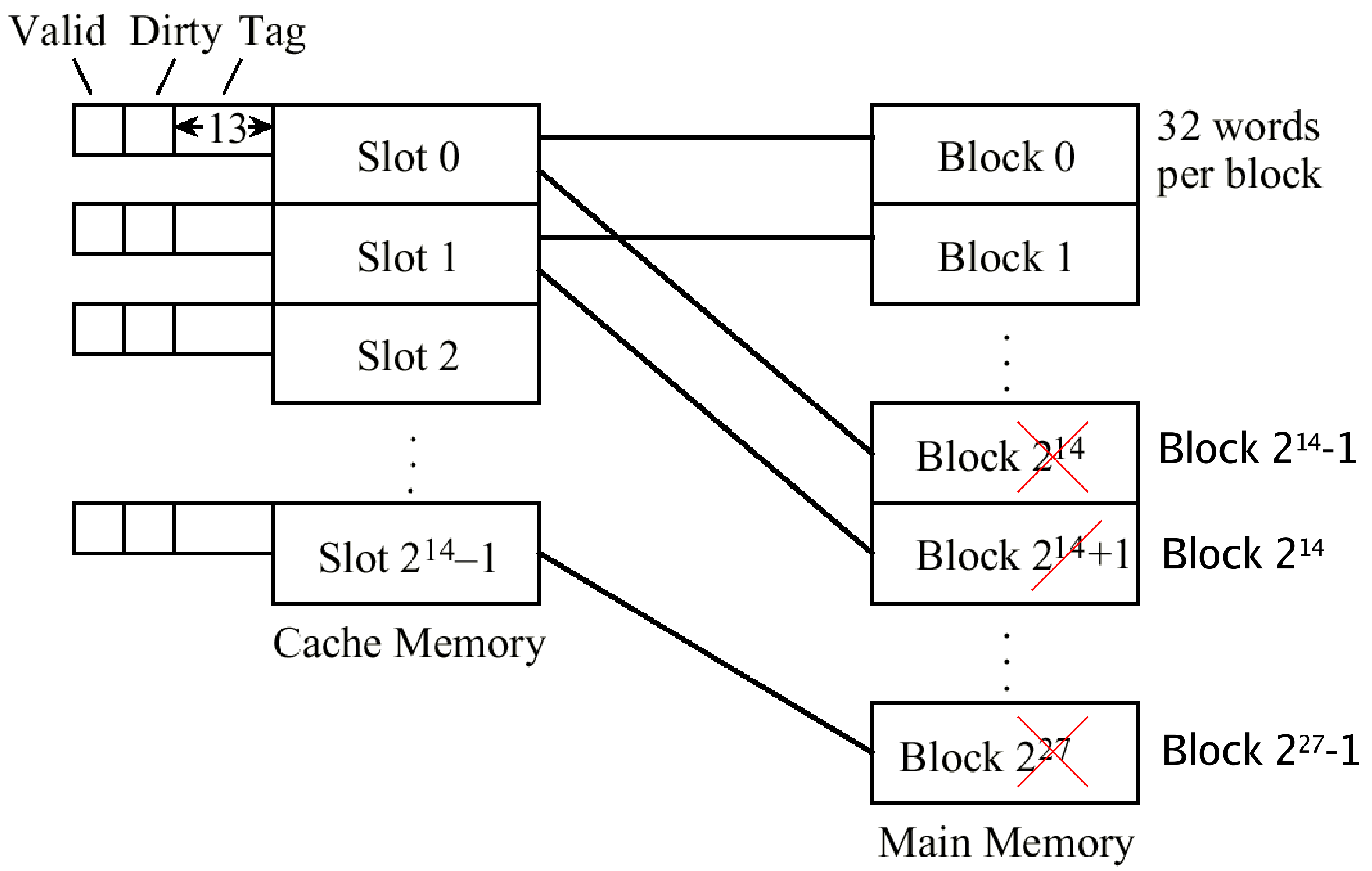
- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of a slot that has tag $(501AF80)_{16}$, which is made up of the 27 most significant bits of the address. If the addressed word is not in the cache, then the block corresponding to tag field $(501AF80)_{16}$ is brought into an available slot in the cache from the main memory, and the memory reference is then satisfied from the cache.



Replacement Policies

- When there are no available slots in which to place a block, a *replacement policy* is implemented. The replacement policy governs the choice of which slot is freed up for the new block.
- Replacement policies are used for associative and set-associative mapping schemes, and also for virtual memory.
- Least recently used (LRU)
- First-in/first-out (FIFO)
- Least frequently used (LFU)
- Random
- Optimal (used for analysis only – look backward in time and reverse-engineer the best possible strategy for a particular sequence of memory references.)

A Direct Mapping Scheme for Cache Memory



Direct Mapping Example

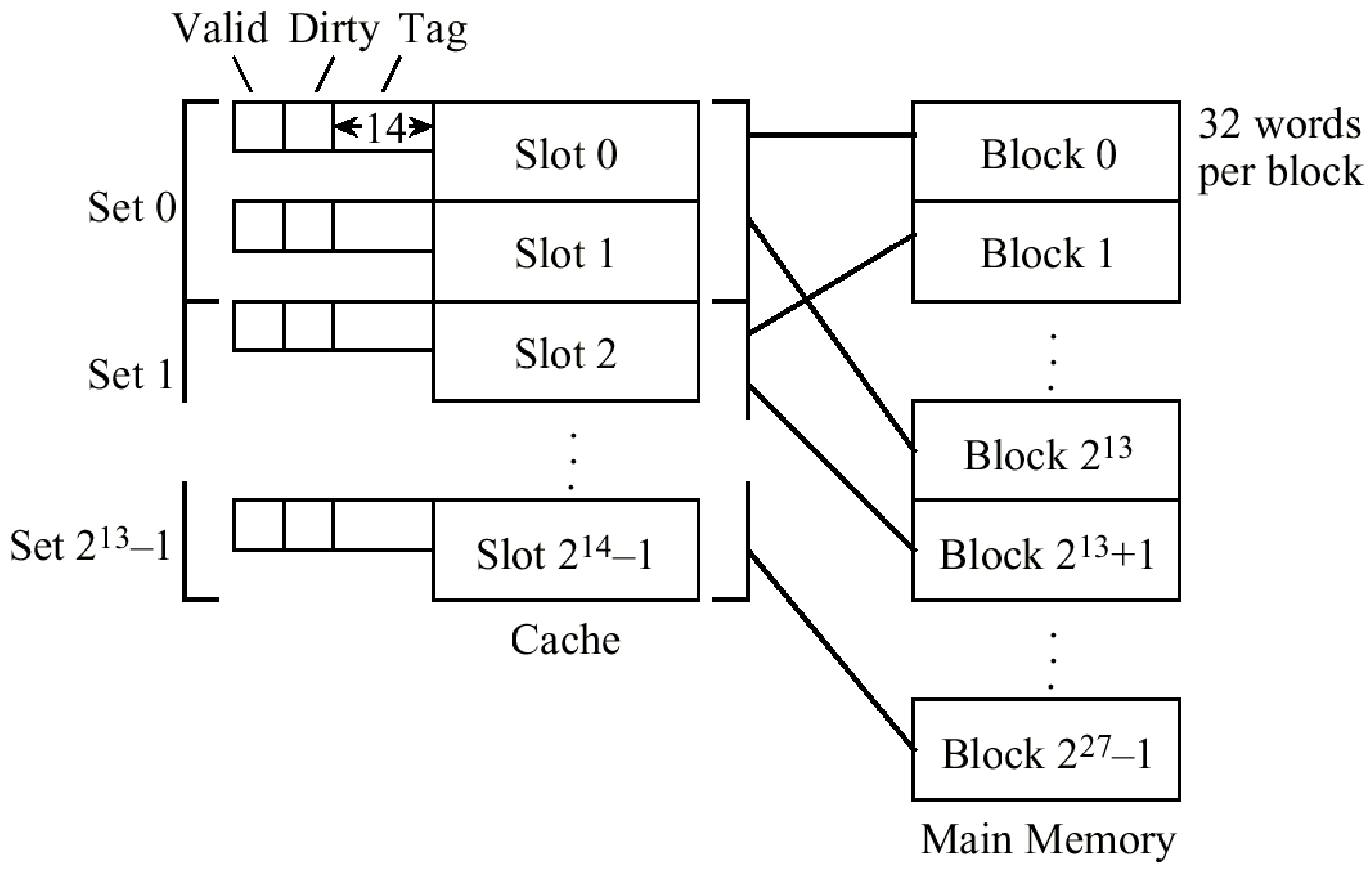
- **For a direct mapped cache, each main memory block can be mapped to only one slot, but each slot can receive more than one block. Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:**

Tag	Slot	Word
13 bits	14 bits	5 bits

- **If the addressed word is in the cache, it will be found in word $(14)_{16}$ of slot $(2F80)_{16}$, which will have a tag of $(1406)_{16}$.**

Tag	Slot	Word
1 0 1 0 0 0 0 0 0 0 1 1 0	1 0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

A Set Associative Mapping Scheme for a Cache Memory



Set-Associative Mapping Example

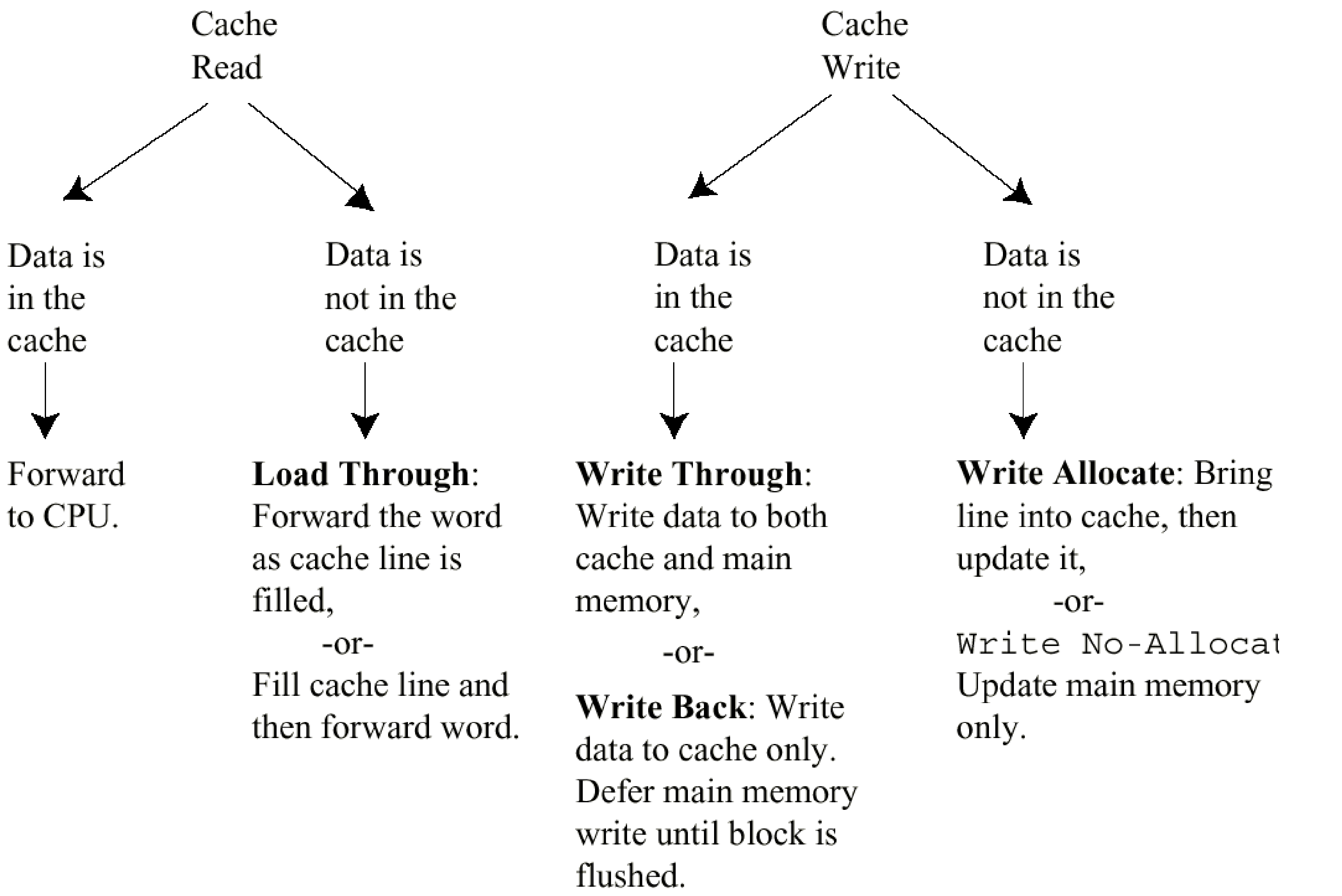
- **Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of 2^5 = 32 words per block, there are two blocks per set, and the cache consists of 2^{14} slots:**

Tag	Set	Word
14 bits	13 bits	5 bits

- **The leftmost 14 bits form the tag field, followed by 13 bits for the set field, followed by five bits for the word field:**

Tag	Set	Word
1 0 1 0 0 0 0 0 0 0 1 1 0 1	0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

Cache Read and Write Policies



Cache Read

Data is in the cache

Forward to CPU.

Data is not in the cache

Load Through: Forward the word as cache line is filled,

-or-

Fill cache line and then forward word.

Cache Write

Data is in the cache

Write Through: Write data to both cache and main memory,

-or-

Write Back: Write data to cache only. Defer main memory write until block is flushed.

Data is not in the cache

Write Allocate: Bring line into cache, then update it,

-or-

`Write No-Allocat`

Update main memory only.

Hit Ratios and Effective Access Times

- **Hit ratio and effective access time for single level cache:**

$$\text{Hit ratio} = \frac{\text{No. times referenced words are in cache}}{\text{Total number of memory accesses}}$$

$$\text{Eff. access time} = \frac{(\#\text{ hits})(\text{Time per hit}) + (\#\text{ misses})(\text{Time per miss})}{\text{Total number of memory access}}$$

- **Hit ratios and effective access time for multi-level cache:**

$$H_1 = \frac{\text{No. times accessed word is in on-chip cache}}{\text{Total number of memory accesses}}$$

$$H_2 = \frac{\text{No. times accessed word is in off-chip cache}}{\text{No. times accessed word is not in on-chip cache}}$$

$$\begin{aligned}T_{EFF} = {} & (\text{No. on-chip cache hits})(\text{On-chip cache hit time}) + \\ & (\text{No. off-chip cache hits})(\text{Off-chip cache hit time}) + \\ & (\text{No. off-chip cache misses})(\text{Off-chip cache miss time}) \\ & /\text{Total number of memory accesses}\end{aligned}$$

Direct Mapped Cache Example

- **Compute hit ratio and effective access time for a program that executes from memory locations 48 to 95, and then loops 10 times from 15 to 31.**

- **The direct mapped cache has four 16-word slots, a hit time of 80 ns, and a miss time of 2500 ns. Load-through is used. The cache is initially empty.**

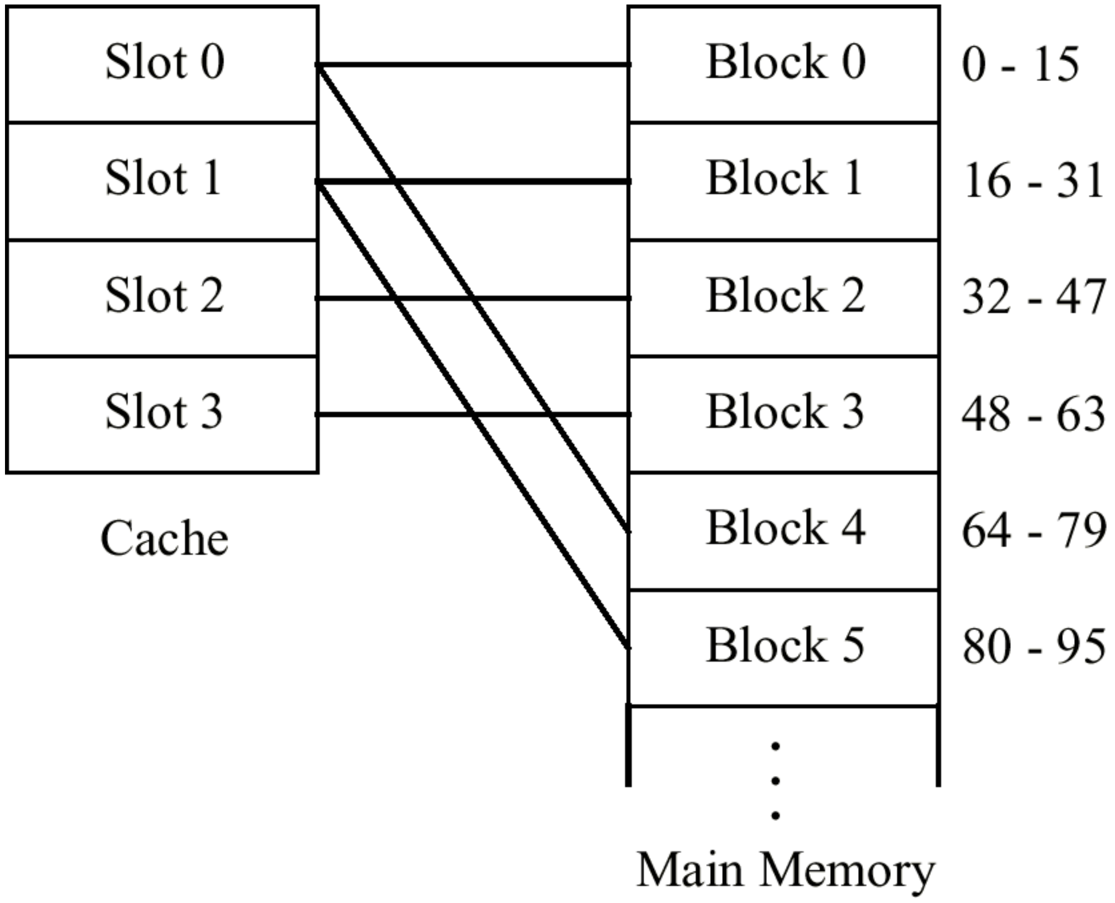


Table of Events for Example Program

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213 Total misses = 5			

Calculation of Hit Ratio and Effective Access Time for Example Program

$$Hit\ ratio = \frac{213}{218} = 97.7\%$$

$$EffectiveAccessTime = \frac{(213)(80ns) + (5)(2500ns)}{218} = 136ns$$