ECSE324 : Computer Organization

Memory
Chapter 8

Christophe Dubach

Fall 2022

Revision history:

Warren Gross – 2017

Christophe Dubach – W2020, F2020, F2021, F2022

Brett H. Meyer – W2021, W2022

Some material from Hamacher, Vranesic, Zaky, and Manjikian, *Computer Organization and Embedded Systems, 6 th ed*, 2012, McGraw Hill, and “Introduction to the ARM Processor using Altera Toolchain.”

Timestamp: 2022/10/20 08:26:00

Disclaimer

It is possible (and even likely) that I will (sometimes) make mistakes and give incorrect information during the live lectures. If you have any doubts, please check the textbook, or ask for clarification online.

Introduction

What is Memory? What is Storage?

Inspiron 15 5000 Intel Non-Touch

$779.99

7th Generation Intel® Core™ i5-7200U Processor

Windows 10 Home 64-bit English

8GB, DDR4, 2400MHz, up to 16GB

1TB 5400 rpm SATA Hard Drive

15.6-in. display

Introducing our newest line of portable powerhouses. Standard with 7th generation Intel® processing power, these laptops are designed with you in mind.

Shipping **Free**

Featured at **$779.99**

1 Year In-Home Support included!

Inspiron 15 5000 Intel Touch

~~$928.99~~ **$799.99**

7th Generation Intel® Core™ i5-7200U Processor

Windows 10 Home 64-bit English

8GB, DDR4, 2400MHz, up to 16GB

1TB 5400 rpm SATA Hard Drive

15.6-in. **touch display**

Upgrade to a touch screen display and experience the true versatility a laptop has to offer. **1 Year In-Home Support included!**

Starting Price ~~$928.99~~

Instant Savings $129.00

Shipping **Free**

Featured at **$799.99**

3 Year In-Home Support included!

Inspiron 15 5000 Intel Non-Touch

~~$1,127.99~~ **$899.99**

7th Generation Intel® Core™ i7-7500U Processor

Windows 10 Home 64-bit English

8GB, DDR4, 2400MHz, up to 16GB

1TB 5400 rpm SATA Hard Drive

15.6-in. display

Upgraded with the new 7th gen Intel® Core™ i7 processor and a 4GB GDDR5 AMD graphics card. **3 Year In-Home Support included!**

Starting Price ~~$1,127.99~~

Instant Savings $228.00

Shipping **Free**

Featured at **$899.99**

Inspiron 15 5000 Intel Non-Touch

$949.99

7th Generation Intel® Core™ i5-7200U Processor

Windows 10 Home 64-bit English

8GB, DDR4, 2400MHz, up to 16GB

256GB Solid State Drive

15.6-in. display

Introducing our newest line of portable powerhouses.

Shipping **Free**

Featured at **$949.99**

What is Memory? What is Storage?

Tech Specs & Customization | Features | Ratings & Reviews | Support

Tech Specs & Customization New Inspiron 15 3000 (Intel®)

< View all configurations

Configurations | Software & Accessories | Support & Services

Configurations

Processor — More Info

7th Generation Intel® Core™ i5-7200U Processor (3MB Cache, up to 3.10 GHz)

Operating System

Windows 10 Home 64-bit English

Memory[i] — More Info

8GB, 2400MHz, DDR4; up to 16GB (additional memory sold separately)

Hard Drive — More Info

1TB 5400 rpm Hard Drive

Video Card — More Info

Intel® HD Graphics 620

New Inspiron 15 3000 (Intel®)

Starting Price	~~$788.99~~
Instant Savings	$159.00
Shipping	**Free**
Featured at	**$629.99**

DFS Financing

$18.00/mo. | 48 months at 13.99% †

APR Range: 13.99% to 28.99% †

Learn More

★ Get $32.00 back

in rewards

Ships 08-03-2017

Order Code ni153567_ftsb_s111e

Add to Cart

What is Memory? What is Storage?

iPad

Overview iOS Tech Specs Buy

Models

Wi-Fi

Wi-Fi + Cellular

Capacity[1]

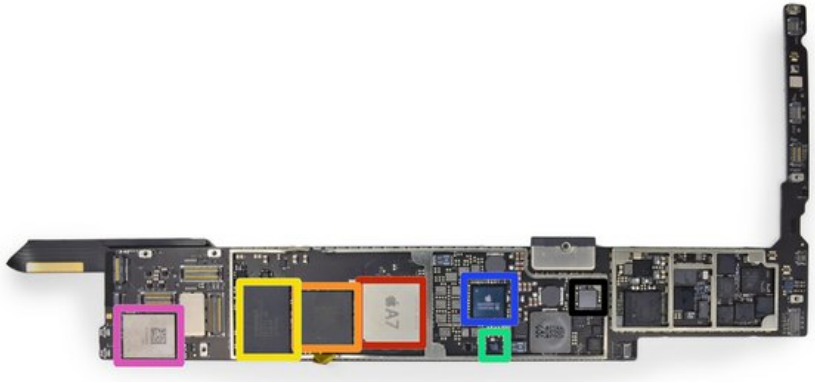
Wi-Fi	Wi-Fi + Cellular
32GB	32GB
128GB	128GB

Size and Weight[2]

Height: 240 mm (9.4 inches)	**Height:** 240 mm (9.4 inches)
Width: 169.5 mm (6.6 inches)	**Width:** 169.5 mm (6.6 inches)

What is Memory? What is Storage?

What is the difference between memory and storage? How do they interact?



source: www.ifixit.com

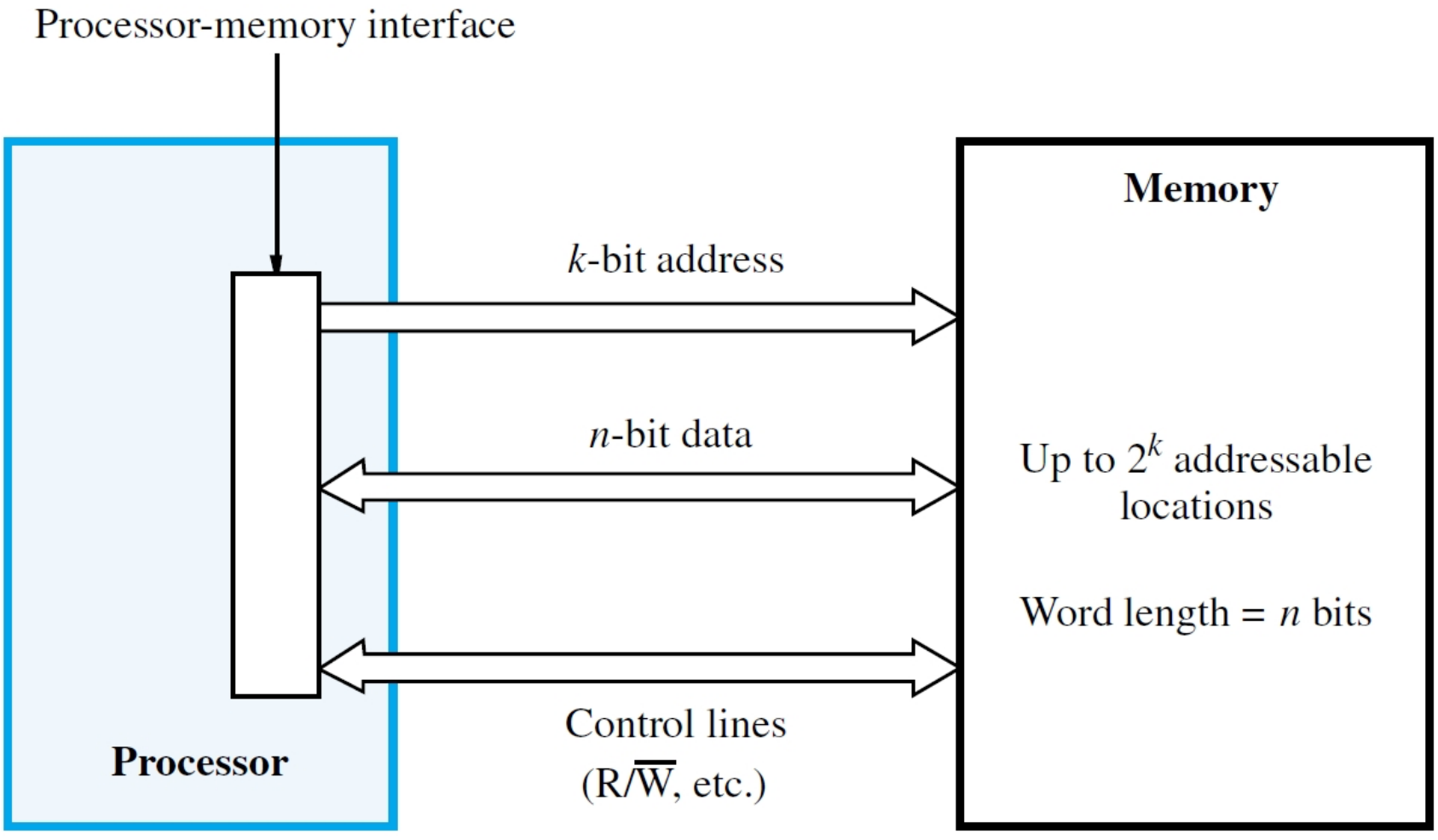
iPad Air LTE board

- Elpida 1 GB LPDDR3 SDRAM *memory*: volatile (temporary) space for data that loses its contents when powered off; addressable
- Toshiba 16 GB NAND Flash *storage*: non-volatile (permanent) space for data; only accessible through the OS

What are the different technologies used to implement memory?

Memory Technology

To the Processor, the World is Memory



Random Access Memory (RAM)

There are two key metrics (amongst others) used to describe memory:

- Memory access time: the time from *initiation* to *completion* of a word or byte transfer
- Memory cycle time: the minimum time between *initiation* of successive transfers

Random access memory (RAM) means that access time is independent of the accessed location.

Memory Technology

Semiconductor RAM Memories

Textbook§8.1, 8.2

Semiconductor RAM

Semiconductor RAM is organized as a 2-D array of cells, each storing a single bit.

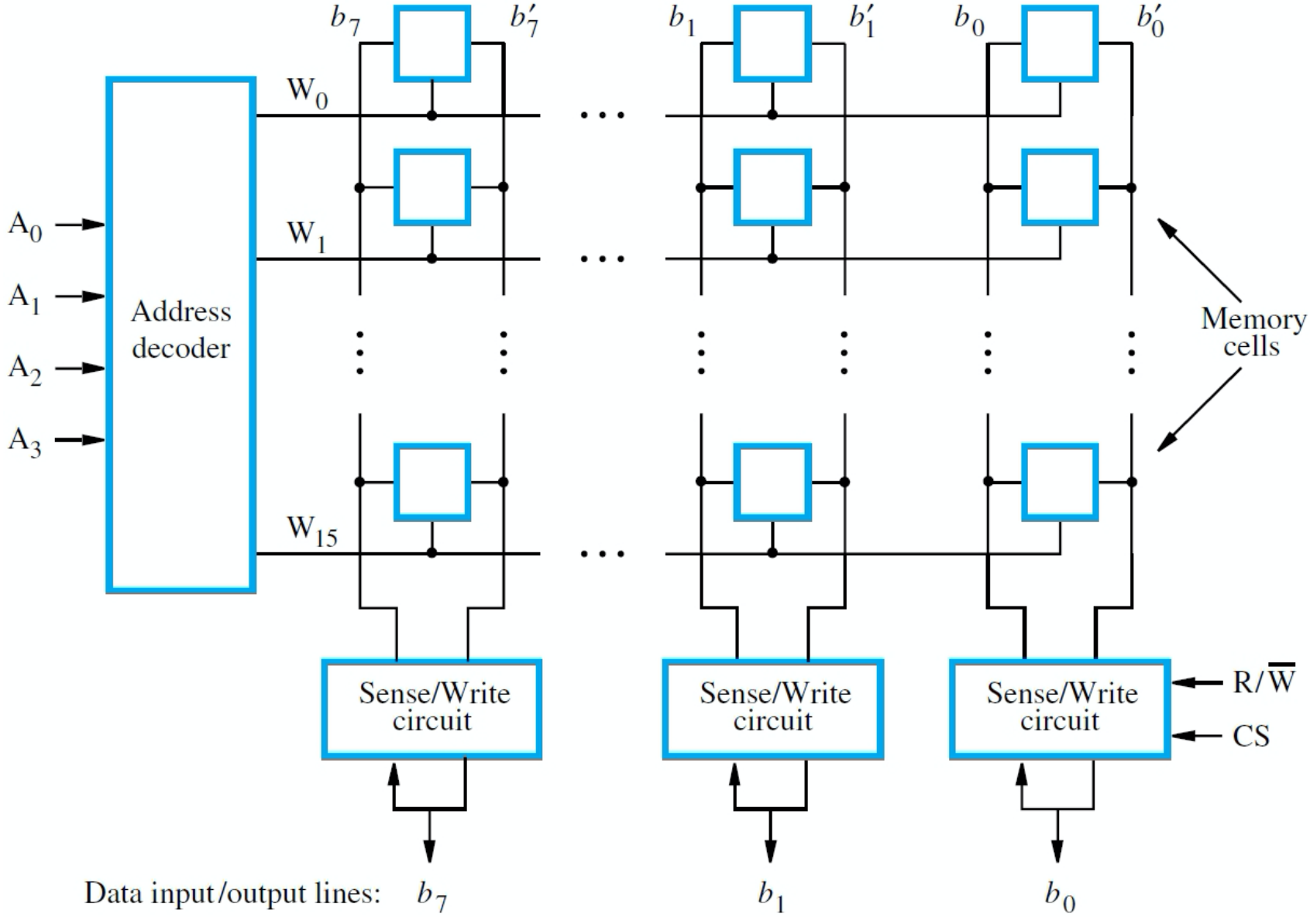
- Each row of the array stores one memory word

Semiconductor RAM

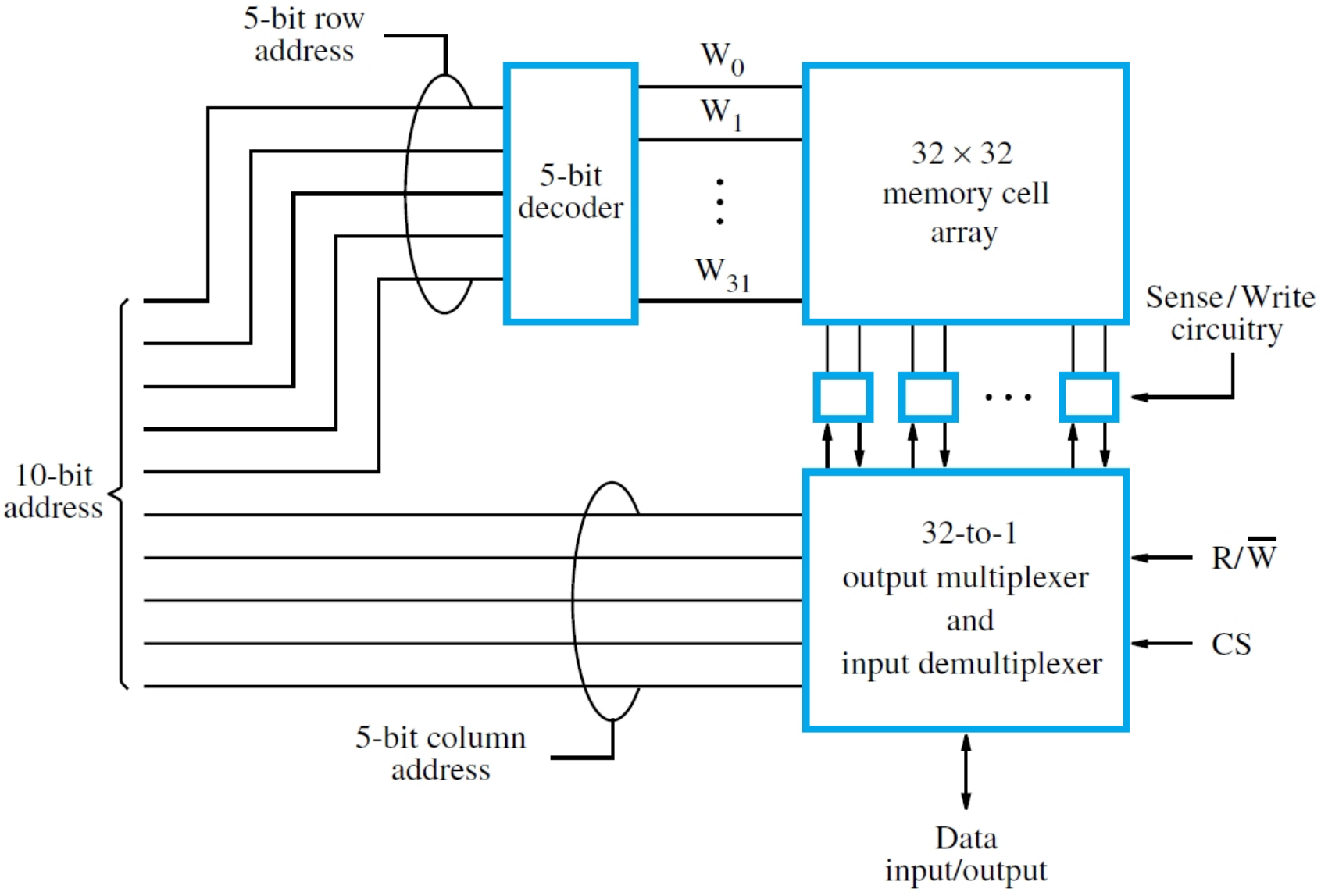
Semiconductor RAM is organized as a 2-D array of cells, each storing a single bit.

- Each row of the array stores one memory word – *note! a memory word may be different in size than a processor word!!*
- *E.g.,* consider a 16x8 RAM with an 8-bit word size and 16 words.
  - How many bits does this memory store?
  - How many bits are needed for the memory address?

16x8 RAM

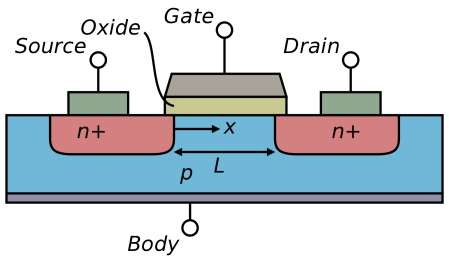

b_7
b'_7
b_1
b'_1
b_0
b'_0
W_0
W_1
W_{15}
A_0
A_1
A_2
A_3
Address decoder
Memory cells
Sense/Write circuit
Sense/Write circuit
Sense/Write circuit
R/$\overline{W}$
CS
Data input/output lines: b_7
b_1
b_0

1024x1 RAM


5-bit row address
5-bit decoder
W_0
W_1
W_{31}
32×32 memory cell array
Sense/Write circuitry
10-bit address
32-to-1 output multiplexer and input demultiplexer
R/$\overline{W}$
CS
5-bit column address
Data input/output

Static RAM

Static RAM (SRAM) is made out of complementary MOS (CMOS) transistors.

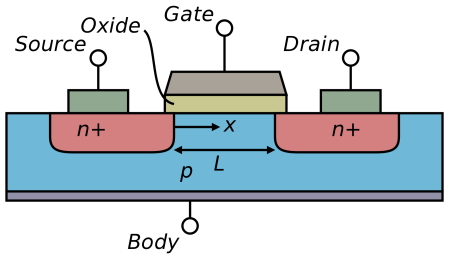


Cross-sectional view of a MOSFET (n-type).

- SRAM is volatile: it retains state as long as power is applied
- SRAM is fast, but expensive: access time is typically a few ns, but each bit requires six transistors
- SRAMs are typically no larger than a few a Mbit
- Today, SRAMs often* implement on-chip “cache” or “scratch-pad” memories, but not main memory

Static RAM

Static RAM (SRAM) is made out of complementary MOS (CMOS) transistors.

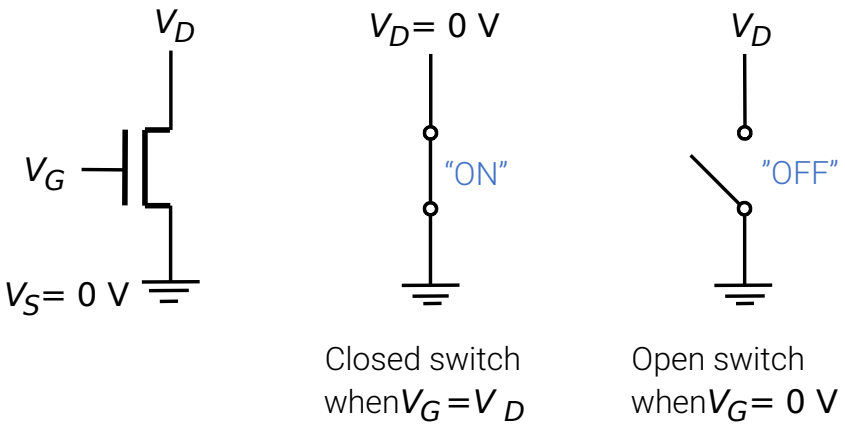


Cross-sectional view of a MOSFET (n-type).

- SRAM is volatile: it retains state as long as power is applied
- SRAM is fast, but expensive: access time is typically a few ns, but each bit requires six transistors
- SRAMs are typically no larger than a few a Mbit
- Today, SRAMs often* implement on-chip “cache” or “scratch-pad” memories, but not main memory

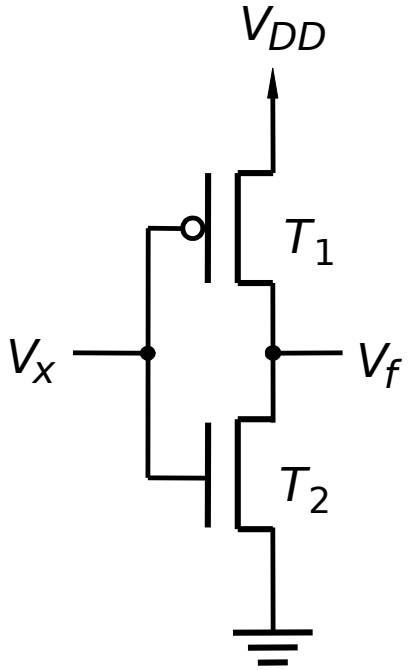
* Except, of course, when main memory is small.

Complementary MOS (CMOS)



CMOS Inverters are a key building block of SRAM

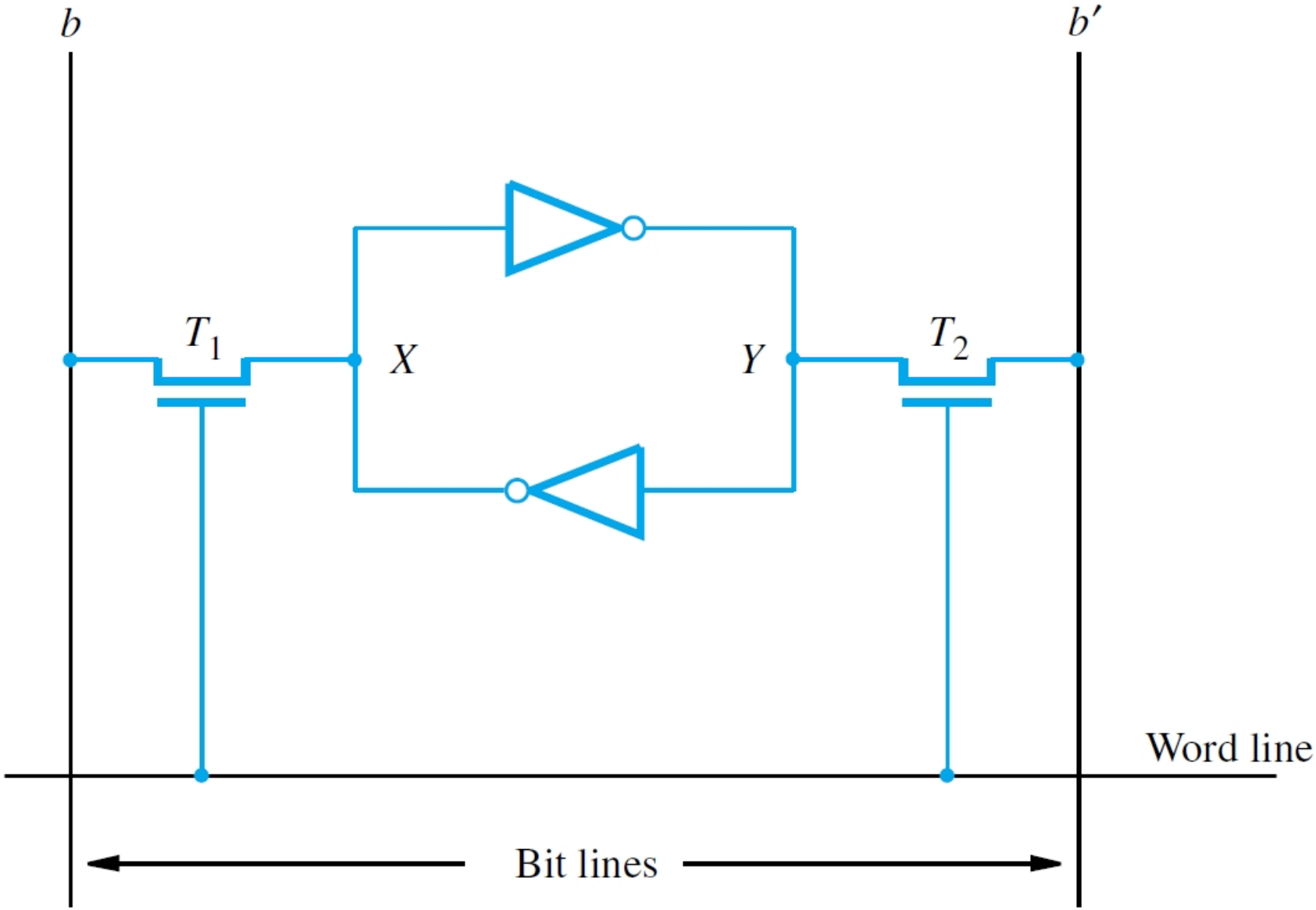
CMOS inverters are made out of one PMOS and one NMOS.



x	T_1	T_2	f
0	on	off	1
1	off	on	0

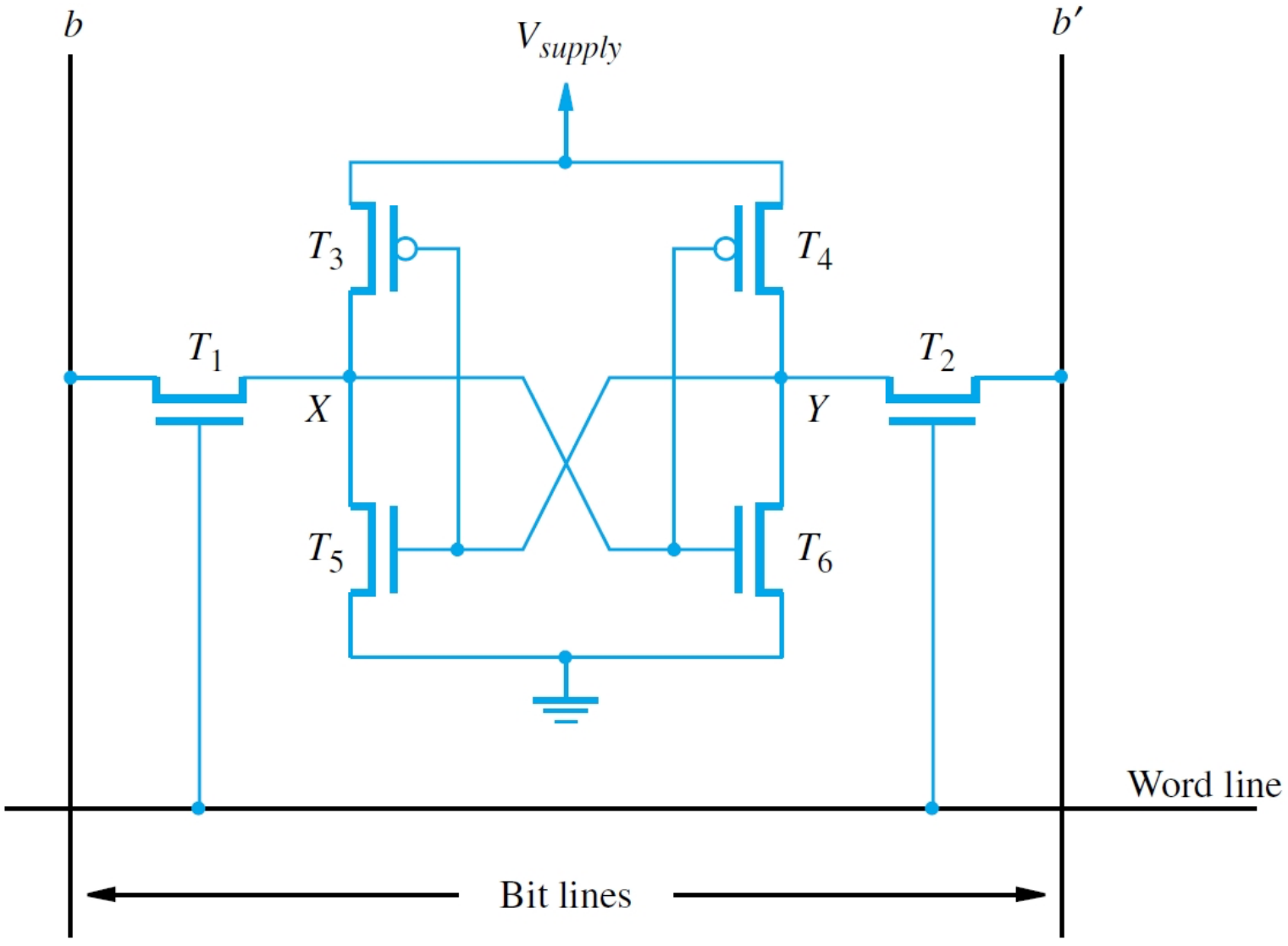
6T SRAM Bit Cell

An SRAM cell is storing a ‘1’ when X is VDD.



6T SRAM Bit Cell

Transistors must be carefully sized for read stability and writeability.



Dynamic RAM

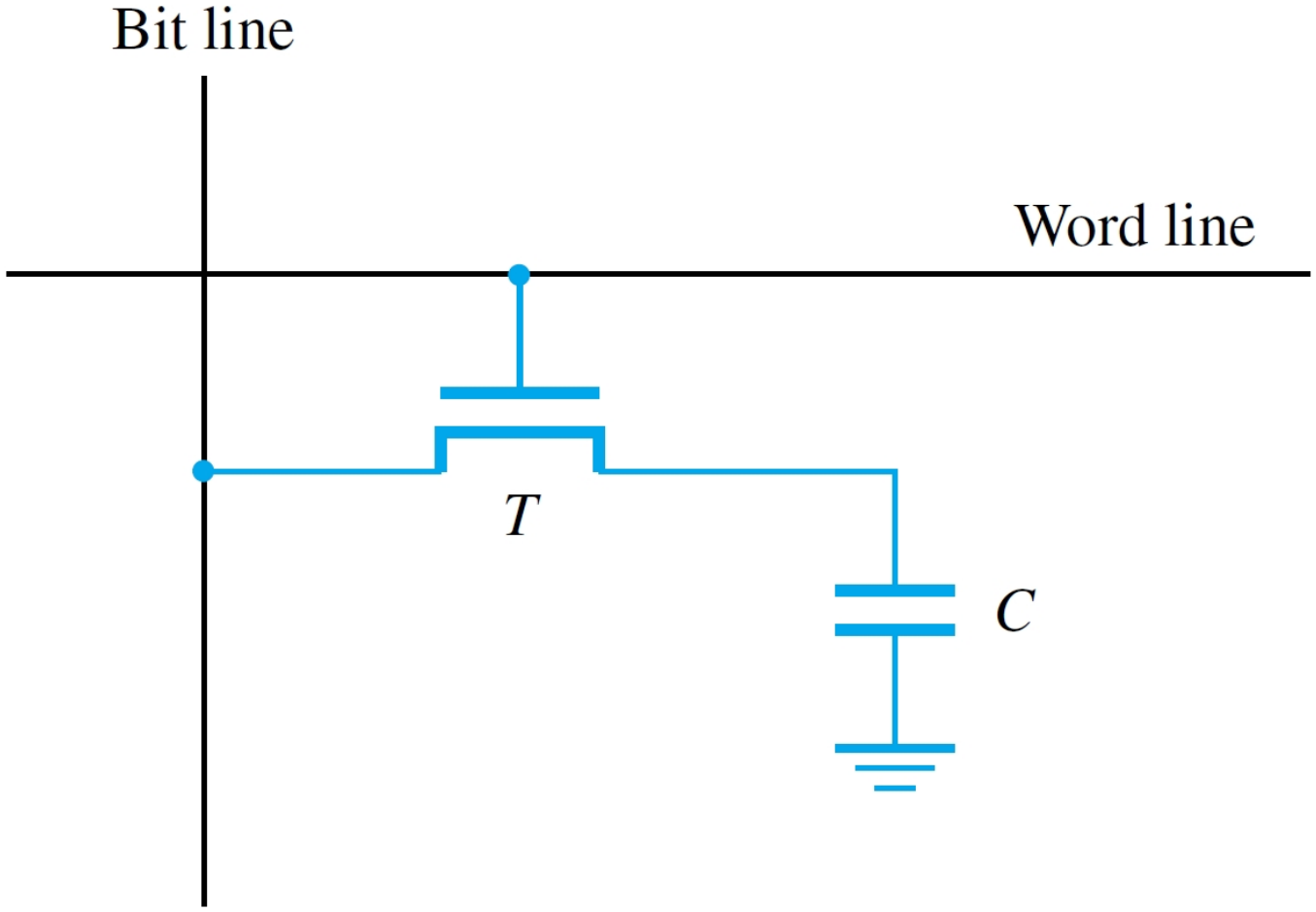
Dynamic RAM (DRAM) is also volatile, but: without refreshing, will lose state even when powered.

SRAM is self-reinforcing: cross-coupled inverters hold state when powered. DRAM is not: a single capacitor leaks its contents within 10s of ms; DRAM must be read periodically (refreshed) to preserve its state.

- DRAM is slower than SRAM, but cheaper: the DRAM cell is simpler, and DRAM is much denser than SRAM
- DRAM arrays can be quite large, up to Gbits
- DRAM is often used to implement off-chip main memory

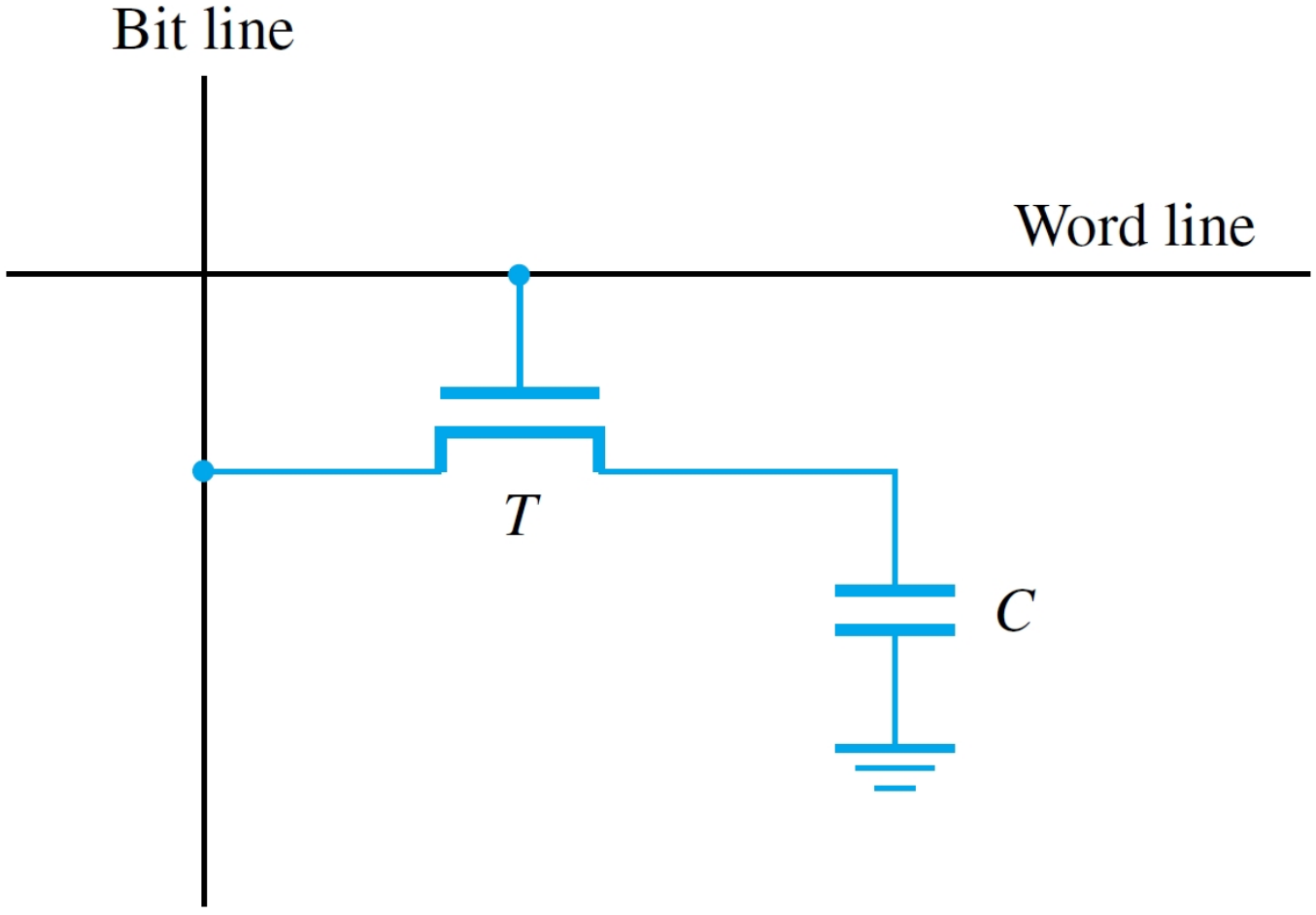
DRAM Bit Cell

A DRAM cell is storing a ‘1’ when the voltage across C is VDD*. The charge in C leaks through T even when T is off.



DRAM Bit Cell

A DRAM cell is storing a ‘1’ when the voltage across C is VDD*. The charge in C leaks through T even when T is off.



* In practice, voltages less than VDD are recognized as ‘1’, too.

Reading DRAM

Because DRAM is built for density, out of the smallest circuit elements possible, accesses are optimized for speed where possible.

- When read, a sense amplifier (sense amp) connected to the bit line detects if the charge in the capacitor is above a threshold
- If above threshold, the sense amp drives the bit line to VDD (‘1’), recharging the capacitor.
- If below threshold, the sense amp pulls the bit line to GND (‘0’), discharging the capacitor.

Reading a DRAM cell refreshes its contents. Note that an entire row is read and refreshed at the same time.

To refresh the entire DRAM, each row must be periodically read.

Refresh Overhead

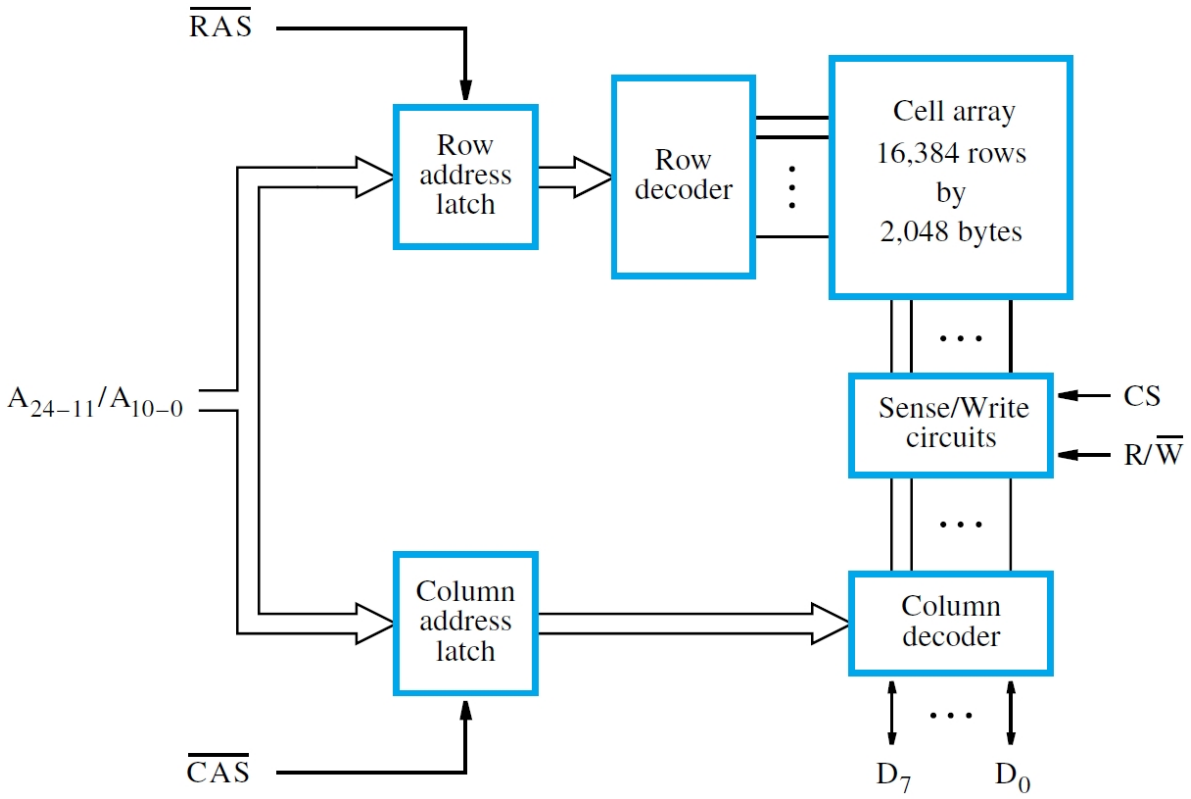
Assume that each row needs to be refreshed every 64 ms, that the minimum time between two row accesses is 50 ns, and that all rows are refreshed in 8192 cycles.

Read/write operations have to be delayed until refresh is finished. What is the refresh overhead?

256 Mb Asynchronous DRAM (32M x 8)

The 25-bit address is broken into 14 bits for row select, 11 for column.

- First, A_{24-11} is driven, and $\overline{RAS}$ asserted, reading a row.
- Then, A_{10-0} is driven, and $\overline{CAS}$ asserted, selecting a byte.



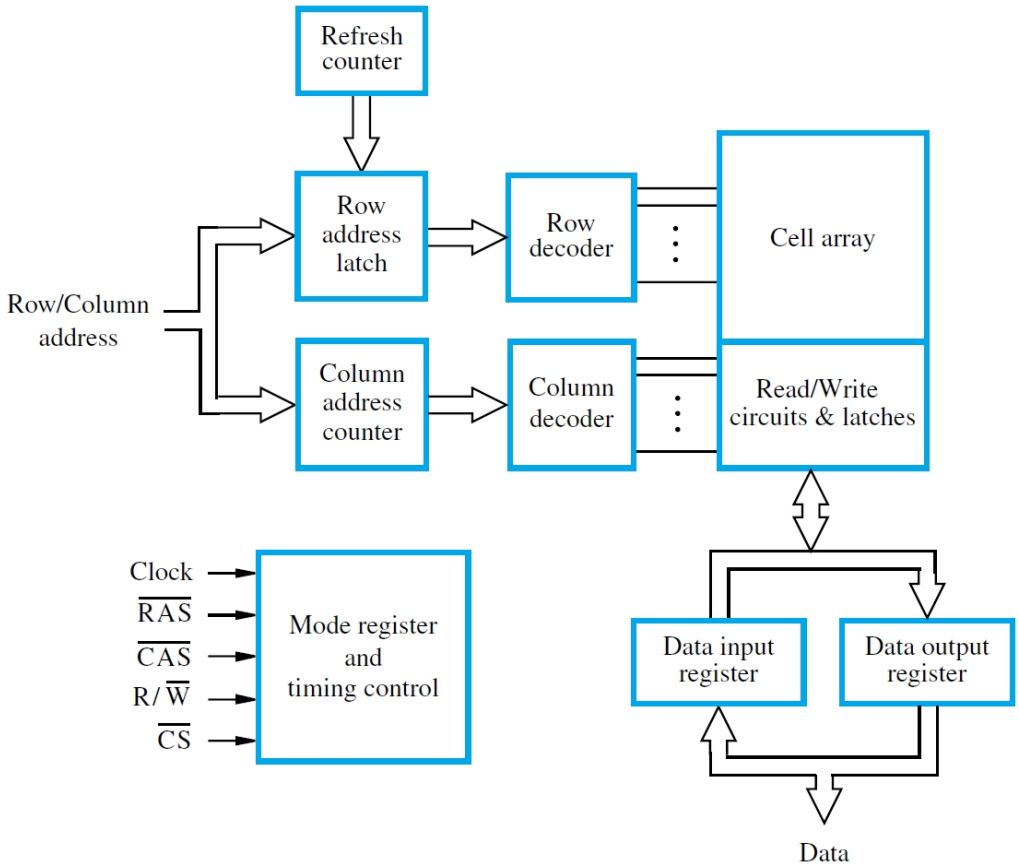
$\overline{RAS}$, $\overline{CAS}$, and refresh are managed by an external memory controller.

Fast Page Mode

- In the preceding example, each read accesses (and refreshes) all 16,384 cells in the addressed row.
- Only 8 bits are transferred, however.
- For more efficient access to data in the same row (*page*), latches in sense amps buffer cell contents.
- Subsequent reads to the same row only require a new column address and $\overline{CAS}$ strobe.
- This is called fast page mode and it speeds up block transfers.

Synchronous DRAM

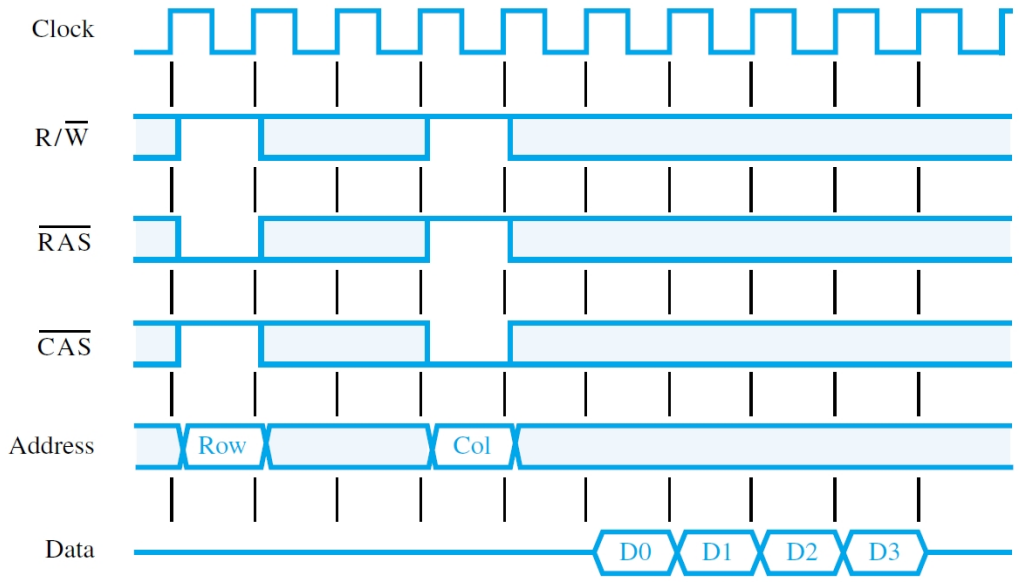
- Synchronous DRAM (SDRAM) integrates an on-chip memory controller
- A clock helps generate internal timing signals (*i.e.*, $\overline{RAS}$ and $\overline{CAS}$)
- Refresh is also built-in
- The “dynamic” nature of the chip is invisible to the user



Efficient Block Transfers

SDRAM can operate in different modes, which determines how signals are generated internally.

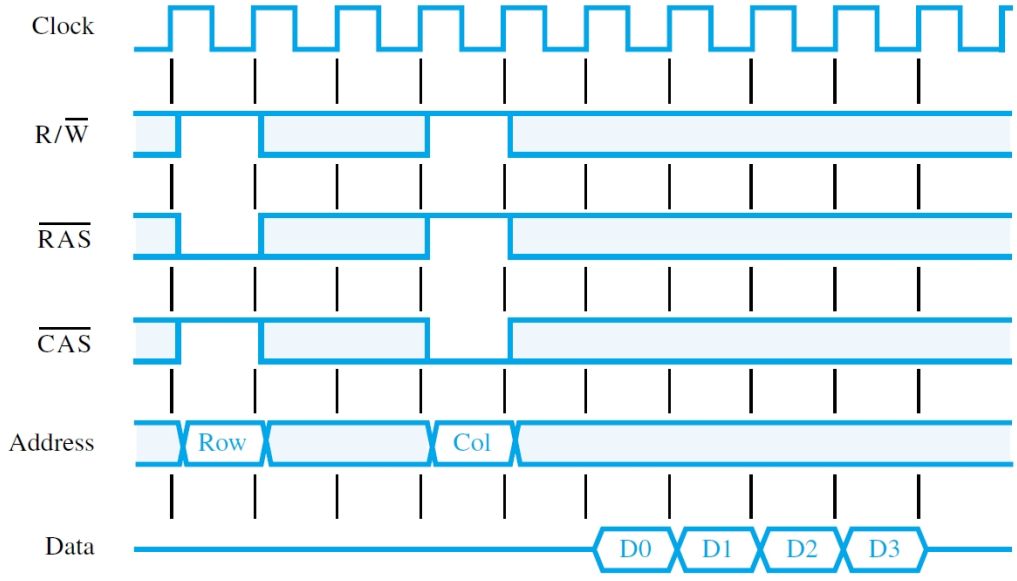
- *E.g.*, burst mode automatically accesses consecutive locations in memory.
- Column address and $\overline{CAS}$ are asserted for one cycle.
- SDRAM circuitry increments the column counter internally for each additional access.



A burst of length 4; RAS delay of 2 cycles; CAS delay of 1 cycle.

Memory Latency and Bandwidth

- Memory latency (ns) is time from initiation to when the first word of a block transfer is on `Data`.
- The time between subsequent accesses to consecutive words is much shorter.
- Memory bandwidth (bps) measures the maximum rate at which data may be transferred.

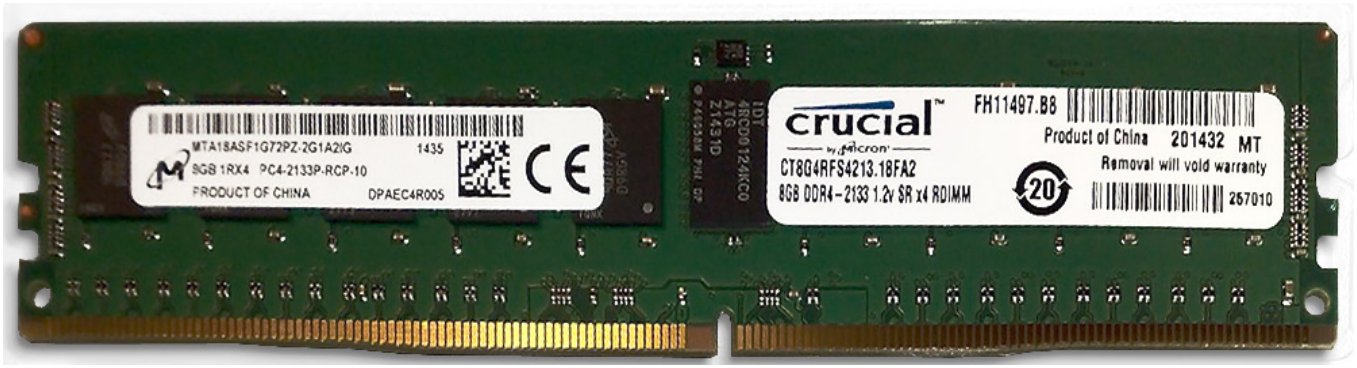


Latency is 5 cycles. If the clock is 500 MHz, the latency is 5 x 1/500e6 = 10 ns. The remaining three words in the transfer are read at one word every 2 ns.

Double-Data-Rate (DDR) SDRAM

Modern SDRAM uses both rising and falling edges of the clock (“double data rate”).

E.g., DDR4 has a clock of 2133 MHz, and can support up to 2400 M transfers per second.

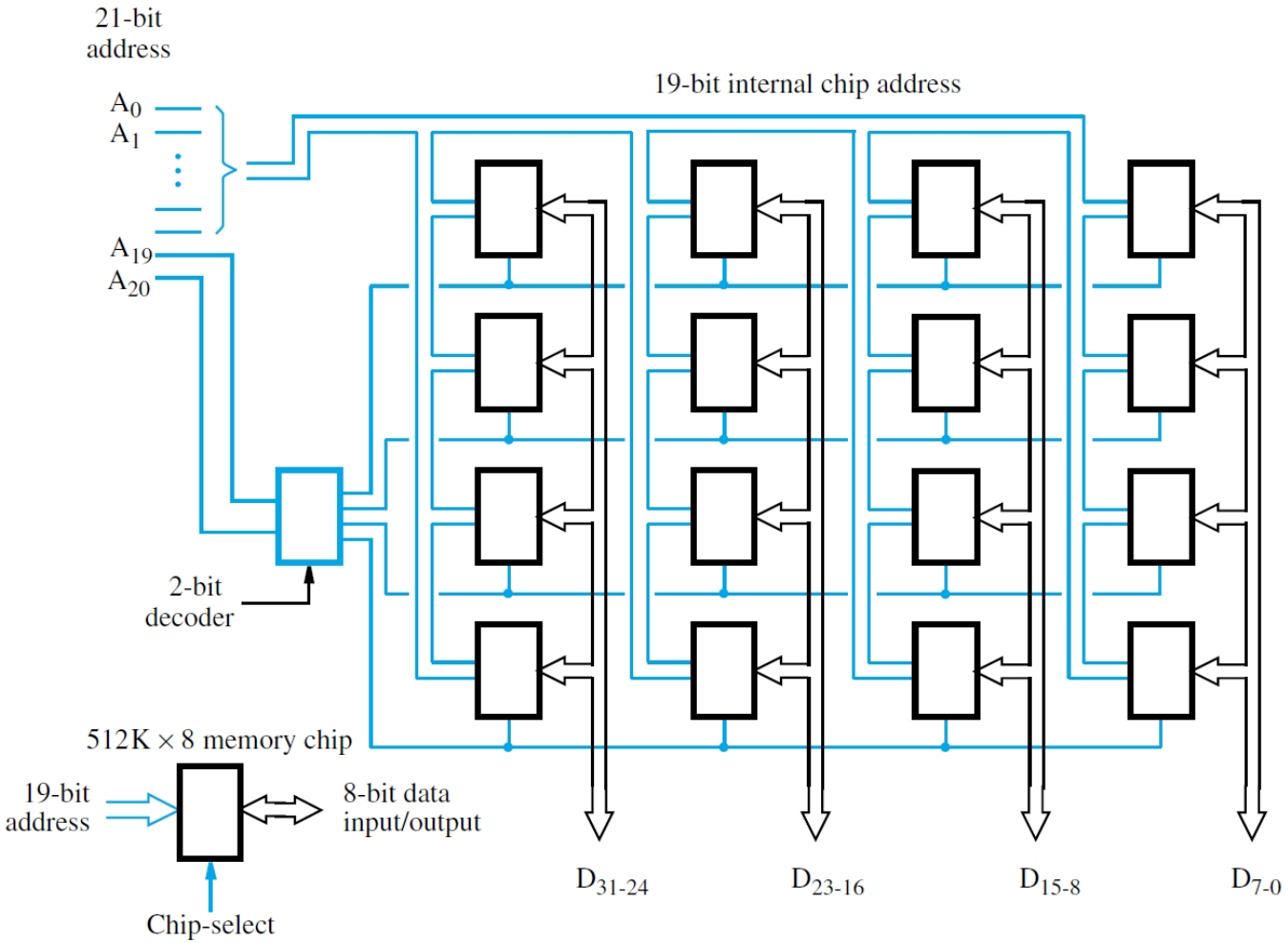


8 GByte DDR4-2133 ECC 1.2 V RDIMM

(Registered dual-inline memory module)

Multi-chip Memories

Multiple smaller memories can be integrated to create a larger memory. *E.g.*, a 2M x 32 SRAM:



Memory Technology

Read-only Memories

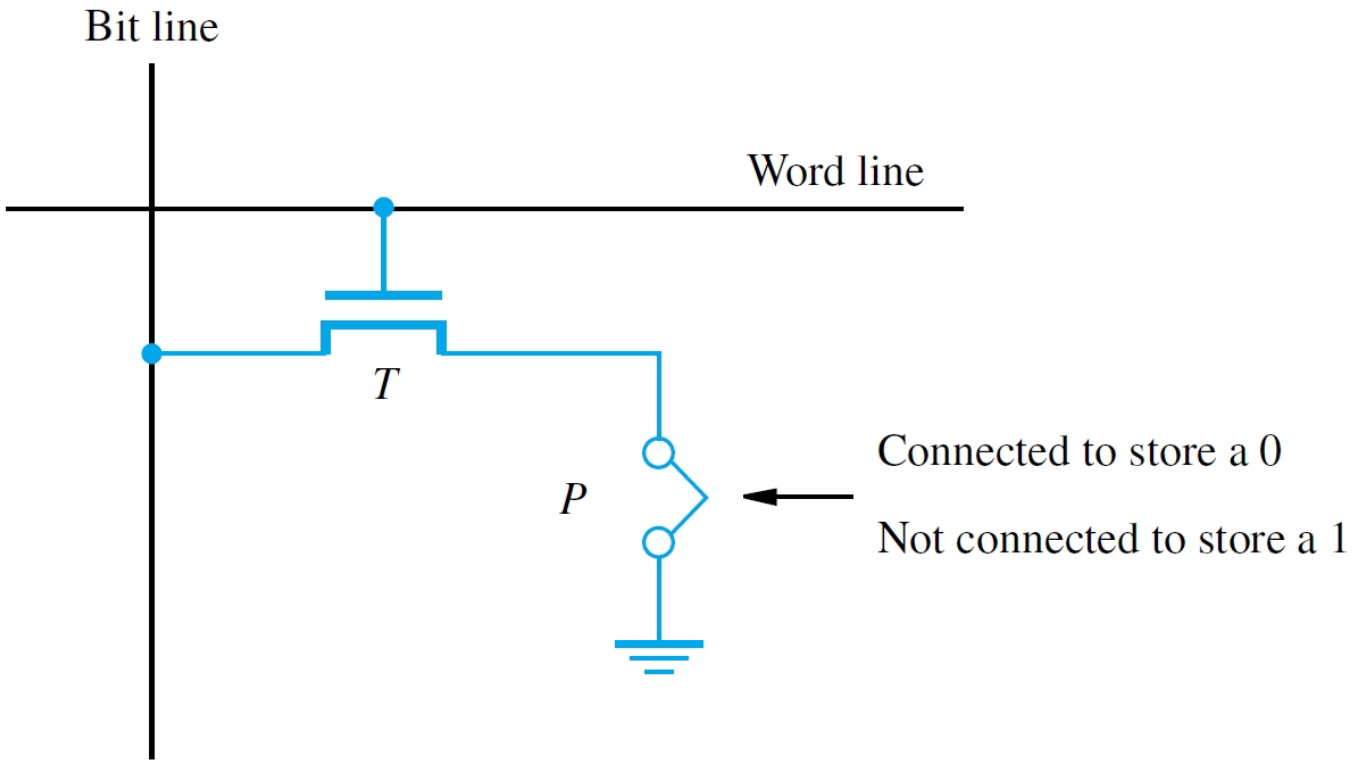
Textbook§8.3

Non-volatile Memories

Non-volatile memories (NVM) are essential for today's embedded systems. NVM

- retain their contents even when unpowered; and,
- are slower than volatile memories, and require special procedures for write accesses.
- NVM are usually used for long-term storage: *e.g.*,
  - code and related data in embedded systems (addressable memory), and
  - solid state drives (SSD) when more storage is needed (managed by the OS)

Read-only Memory (ROM)



PROM, EPROM, and EEPROM

A programmable ROM (PROM) is written once, at manufacturing time, and cannot be later modified: *e.g.*, a fuse is burned out with a large current.

Other types of ROM can be erased and re-written in the field.

- An erasable programable ROM (EPROM) uses a special transistor instead of a fuse.
- Injecting charge allows the transistor to turn on.
- Erasure requires UV light exposure to remove all charge.
- An electrically erasable programmable ROM (EEPROM) supports the selective erasure of cells.

Flash Memory

Flash is a high-density, low-power, low-cost, and very widely adopted NVM.

- Flash cells are designed to be erased in larger blocks, increasing density
- Writing individual cells requires reading a block, erasing it, and writing it back with changes
- Flash cells wear out: wear leveling distributes writes to avoid wearing out some cells before others

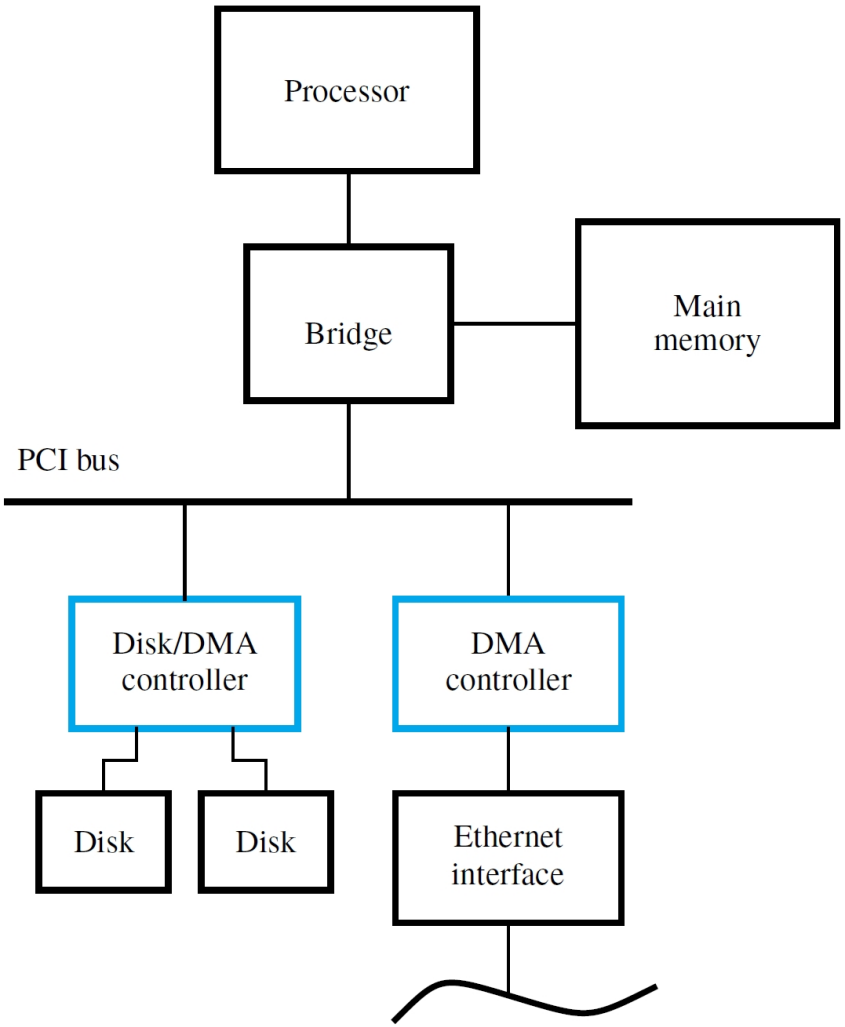
Direct Memory Access (DMA)

Textbook§8.4

Direct Memory Access

CPU overhead for block transfers is high: an address calculation, and load/store instruction, per byte or word.

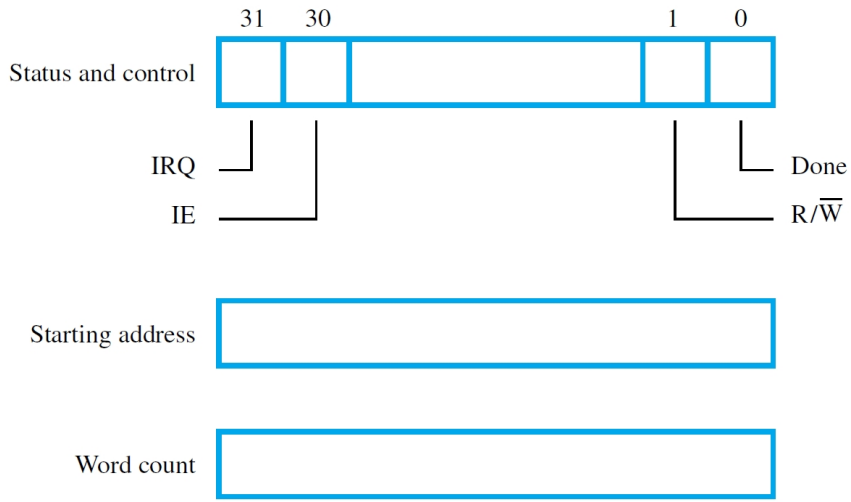
- A DMA controller is an I/O device that manages block transfers between memory and other devices.
- The CPU initiates the transfer, which completes without further CPU involvement.



DMA Controller

DMA controllers may be shared; individual I/O devices may also have DMA controllers.

- CPU writes control registers (starting address, count, $R/\overline{W}$), and initiates transfer.
- The controller keeps track of progress with a counter.
- An interrupt can be used to signal transfer completion.
- DMA can also be invoked to make repeated transfers triggered by a timer.



Caches

Textbook§8.5, 8.6

The Memory Problem

Problem: we want a very large, very fast memory.

- DRAM can be large, but is slow.
- SRAM can be fast, but not large.

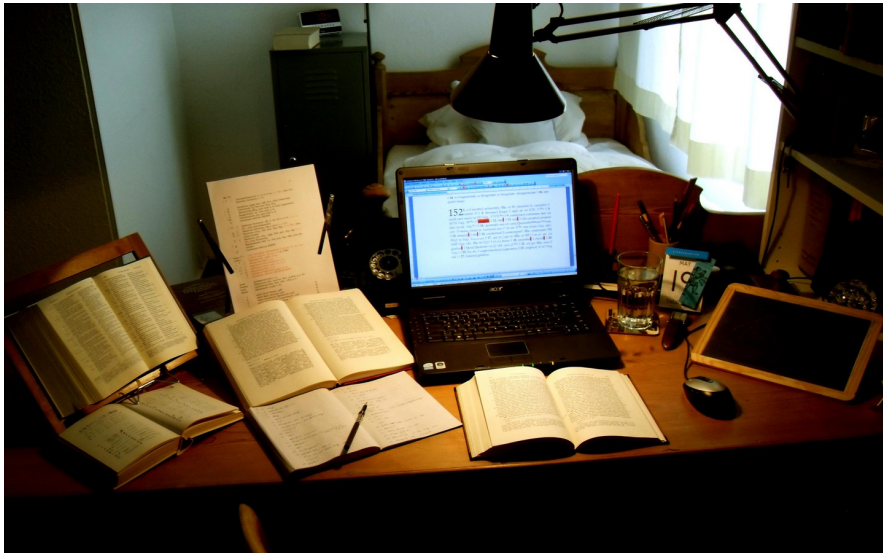
Solution: use both DRAM and SRAM such that the memory appears to the CPU to be large, and fast.

The solution should be transparent to the programmer.

The Memory Problem

Library: large, slow access

Desk: small, fast access



Unlimited amounts of fast memory?

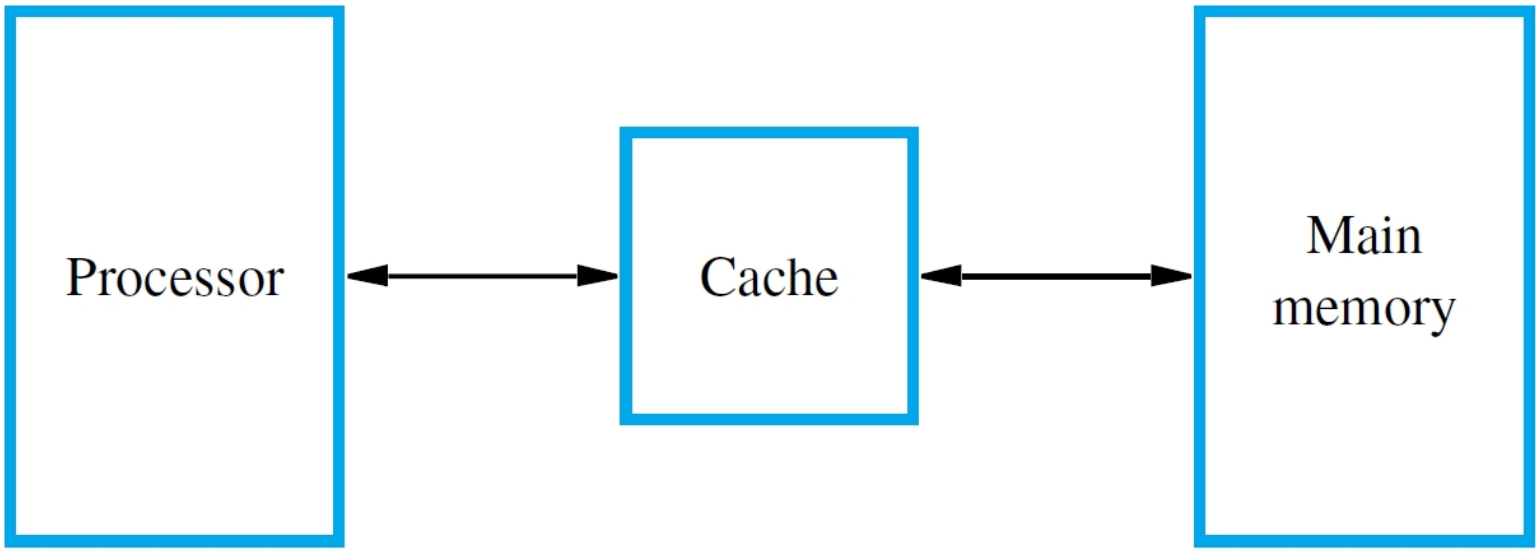
The memory problem is as as old modern computing.

> *"Ideally one would desire an indefinitely large memory capacity such that any particular...word would be immediately available...We are...forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."*

A. W. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design of an electronic computing instrument", 1946.

Memory Hierarchy

We create the illusion of large, fast memory by keeping a copy of (*caching*) frequently used data in a small memory (cache); accesses to cached data (fast) do not require accesses to memory (slow).



- Programmers use load and store instructions as usual
- Specialized hardware manages the movement of data between memory and the cache

Memory Hierarchy

Memory hierarchy design matters to hardware engineers:

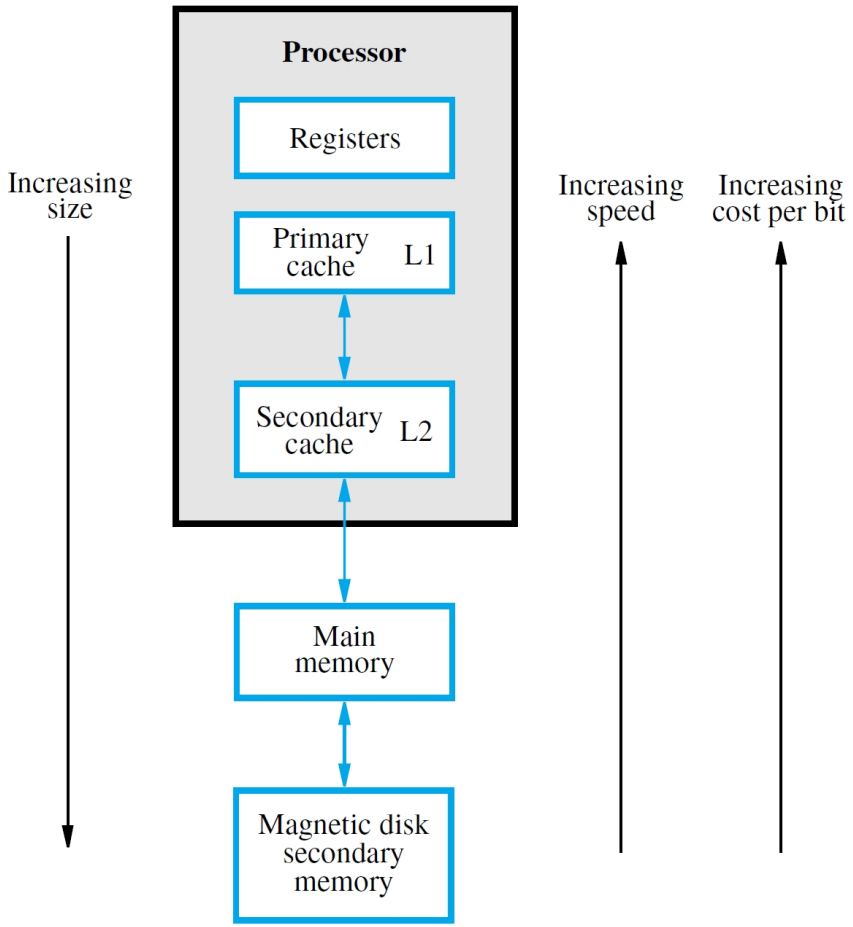
- Different systems call for different amounts of memory
- Different manufacturing technologies mean different amounts of delay for memory accesses
- Memory hierarchy must be carefully tuned for each system

Memory hierarchy design matters to software engineers:

- Memory hierarchy cannot hide all memory access delay
- Delay is hidden better for some access patterns than others
- Understanding memory hierarchy makes it easier to write fast software

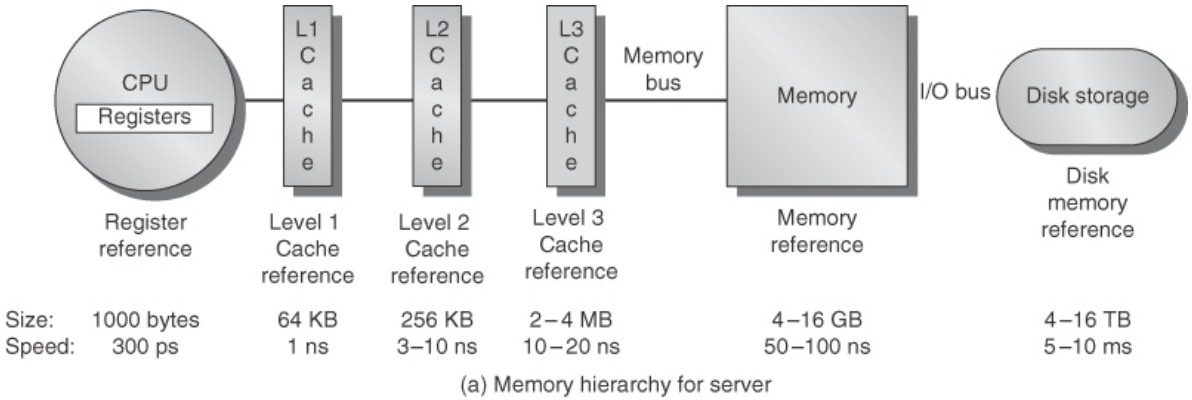
Memory Hierarchy

Modern memory hierarchies incorporate multiple levels of cache, which may be split into instruction and data caches, or unified, shared by multiple processors, or private.

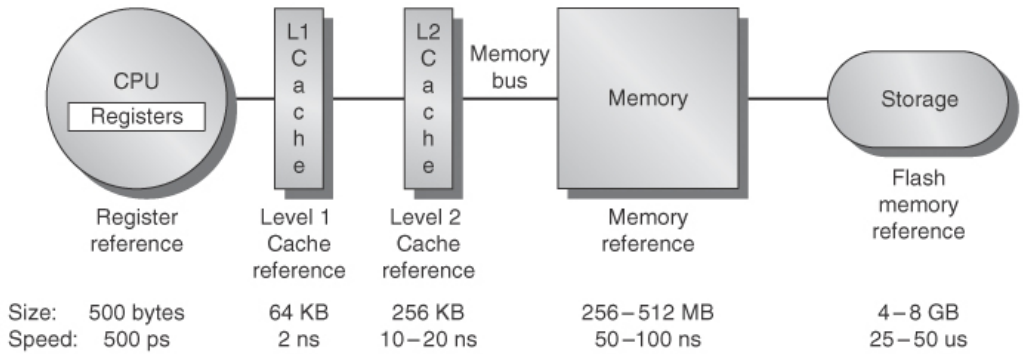


Memory Hierarchy

Even when two different systems have the same number of levels or hierarchy, different use cases may mean different sizes for each memory.



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Locality, Locality, Locality

Why does caching work?

- Programs and data are naturally structured in convenient ways
- Software engineering, compilation, and computer hardware, have evolved together
- Consequently, memory accesses tend to follow predictable patterns
- This is called the principle of locality, which caches exploit

Temporal Locality

Recently accessed items are likely to be accessed again soon: loops, data reuse.

Spatial Locality

Items near an accessed item are likely to be accessed soon: code without branches, arrays.

Cache Basics

Caches are too small to store copies of the entire address space; at any given time, some recently accessed things will be in the cache, other things will not.

Each time the CPU (a) fetches an instruction, or (b) accesses data:

- Look for it in the cache.
- Is it there? That's a cache hit
  - Deliver the desired item to the processor
- Is it not there? That's a cache miss
  - Copy the item from main memory into the cache
  - Deliver the desired item to the processor

Hit and Miss Rate

Caches improve performance as long as enough accesses hit in cache.

- It is not uncommon for caches to have a hit rate of >95%.
- Instruction accesses are especially predictable; data accesses less so.

$$\textit{hit rate} = \textit{cache hits}/\textit{memory accesses}$$

$$\textit{miss rate} = 1 - \textit{hit rate}$$

Where are items put in the cache?

The cache is a RAM; where should a particular item be stored in it? Where do we look for the item that we want?

- Caching divides main memory into blocks (a.k.a. *cache lines*), each consisting of several consecutive data elements.
- *E.g.*, a typical cache line size is 64B.
- When a miss occurs, the block containing the desired item is transferred from main memory.
- Where the block goes is determined by the mapping function.

Some mapping functions are simple; others are more complex, but result in a higher hit rate.

Direct-mapped Cache

Assume a cache size of n blocks, and m words per block.

The simplest mapping function is direct mapping:

- Every block in memory maps to a *single* block in cache
- Memory block j goes in cache block (j mod n)

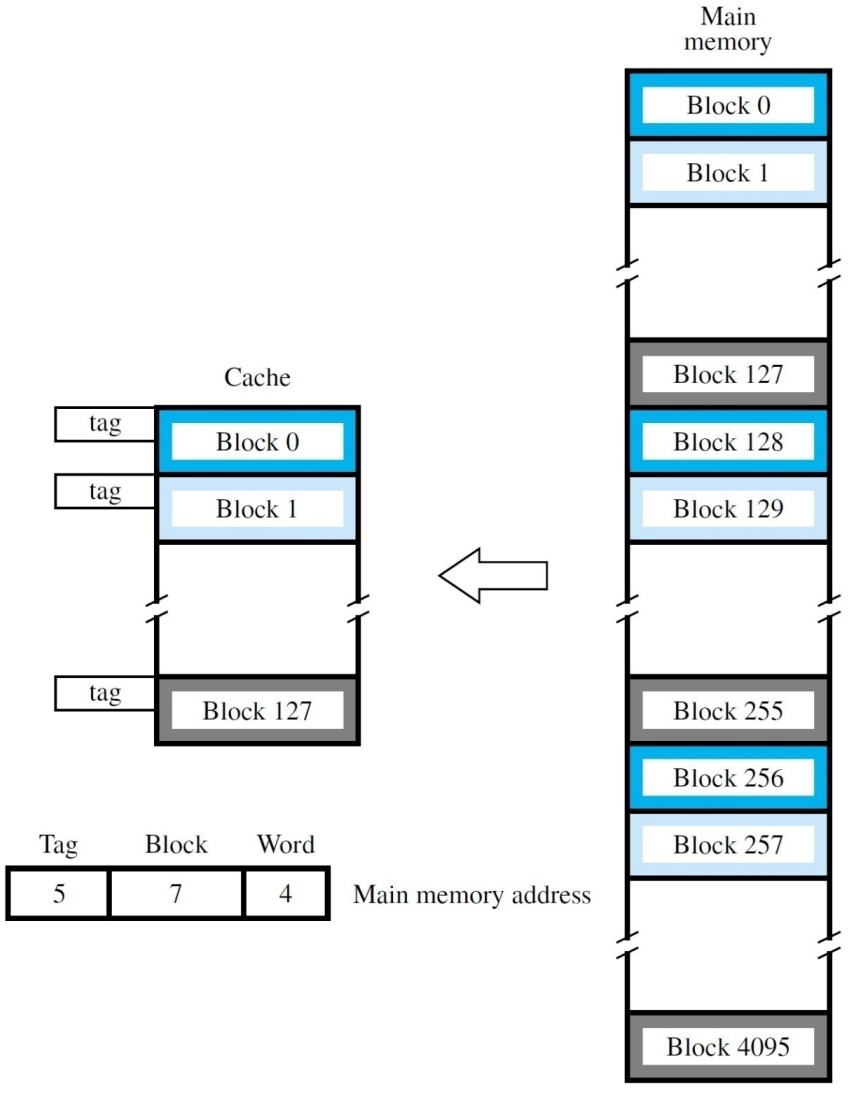
Direct-mapped Cache

Block size: m = 16B (16 words)
Cache size: n = 128 blocks
Main memory: 64KB $\rightarrow$ 4K blocks
Address size: 16 bits

Memory block *j* goes in cache block (*j* mod 128)

The 16-bit address is divided into three parts: word, block, and tag:

- Word selects the appropriate cache column (1 of 16)
- Block selects the appropriate cache row (1 of 128)



Direct-mapped Cache

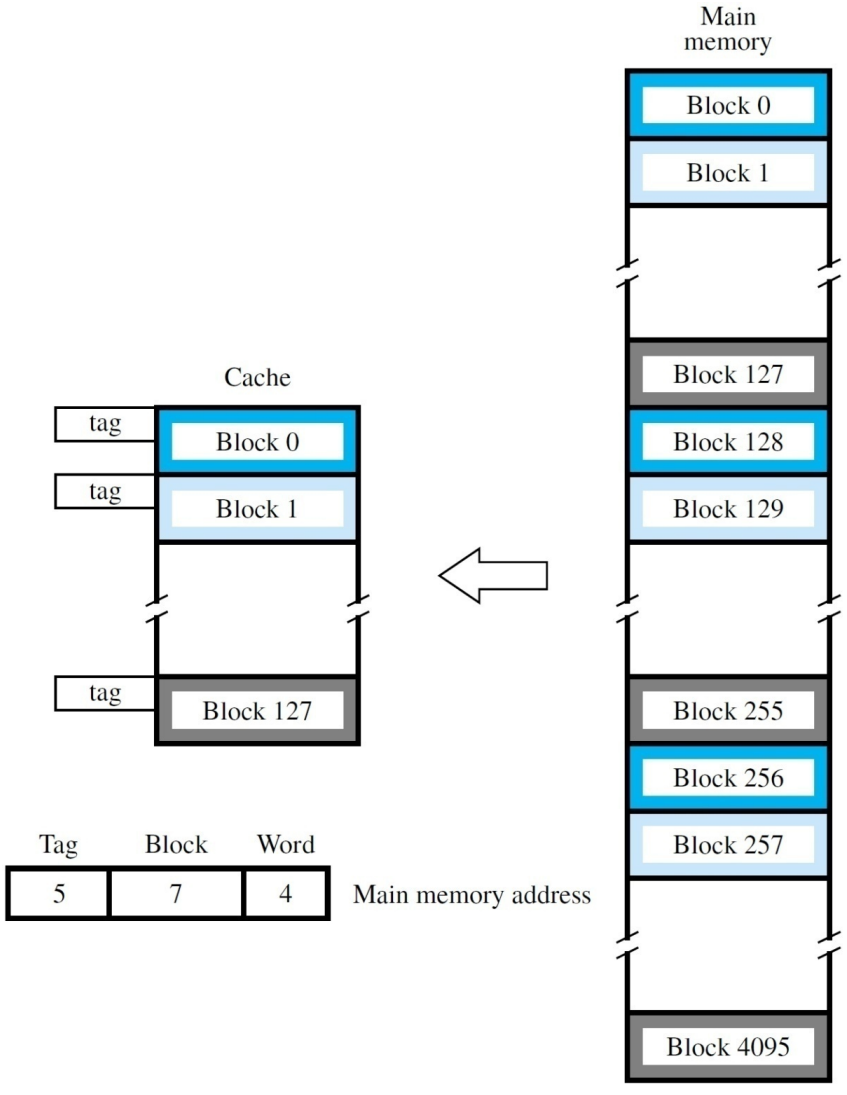
- Tag disambiguates different memory blocks that map to the same cache block

How many memory blocks map to each cache block?

When a block is stored in cache, the tag is also stored in the tag array. On an access, the tag for the requested word is compared with that in the tag array.

Match? *Hit!*

Miss? Replace block; update tag.



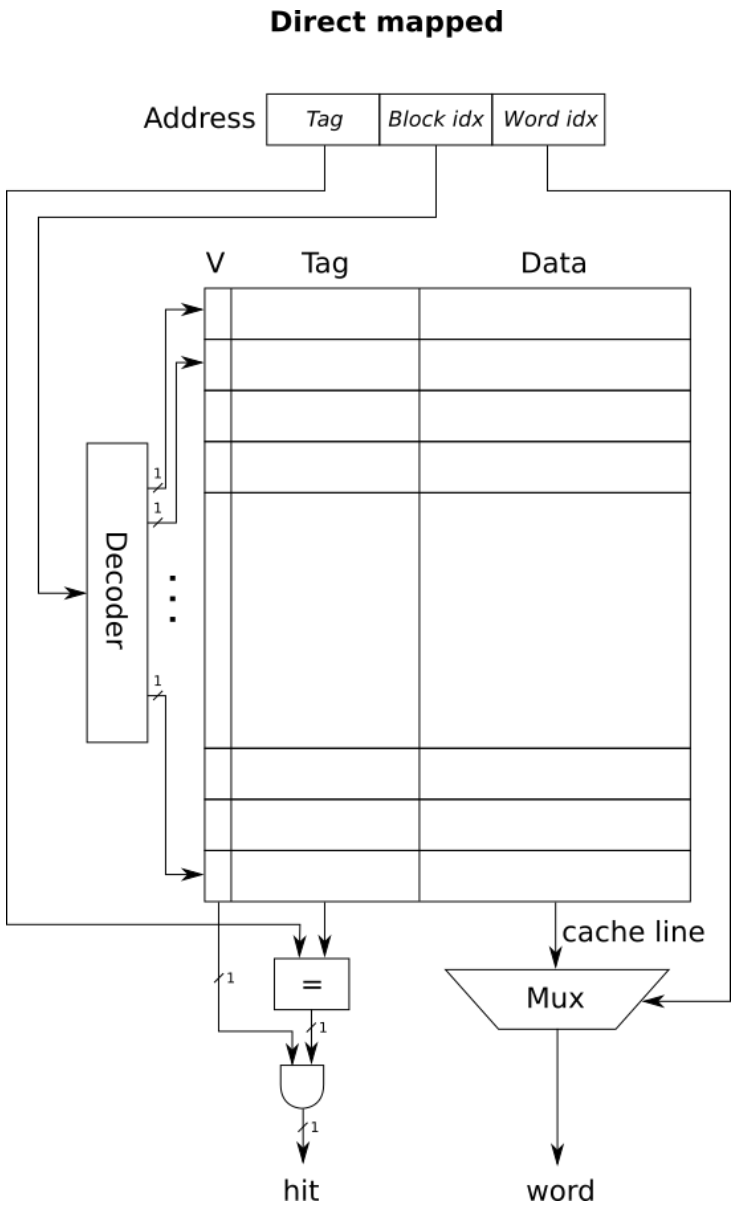
Valid Bit

What happens when a cache block is accessed for the first time? The tag could match, but the data would be invalid.

Each cache block also has a valid bit, initialized to '0,' and set to '1' whenever a block is copied into the cache.

- For a hit to occur, valid must be 1.
- Valid bits are reset under different circumstances, *e.g.*, whenever a new program begins executing.

Direct-mapped Cache



Direct-mapped Cache

The advantage of direct-mapped caches: simple (and fast) hardware maps memory addresses to cache blocks.

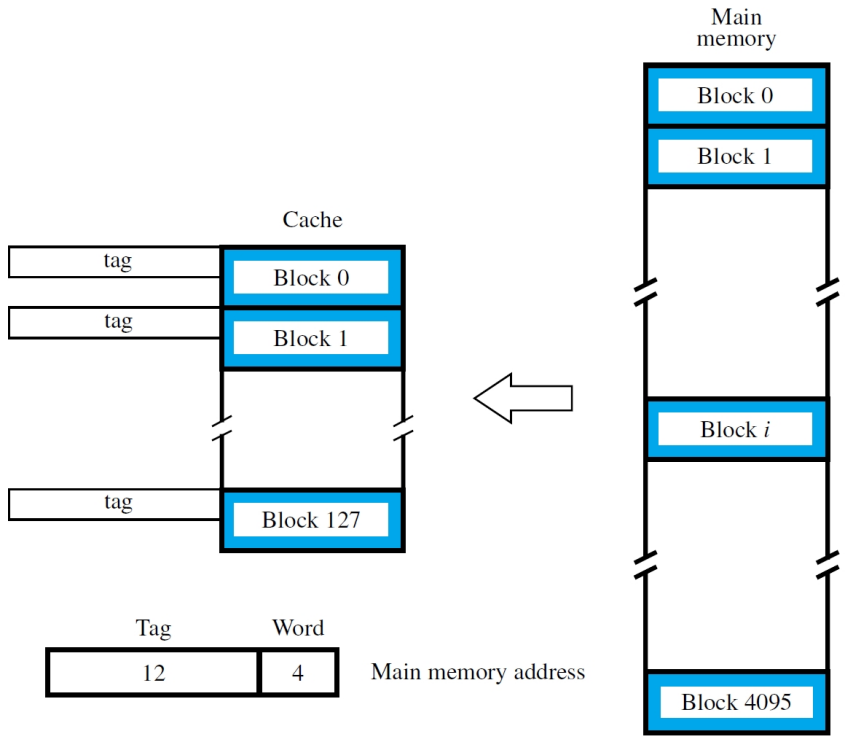
The drawback of direct-mapped caches: multiple blocks may contend for the same location.

- Newly requested blocks always overwrite blocks previously stored at a given location
- If multiple frequently accessed memory blocks map to the same cache block, they will replace each other, resulting in more misses, and costly accesses to main memory

Fully-associative Cache

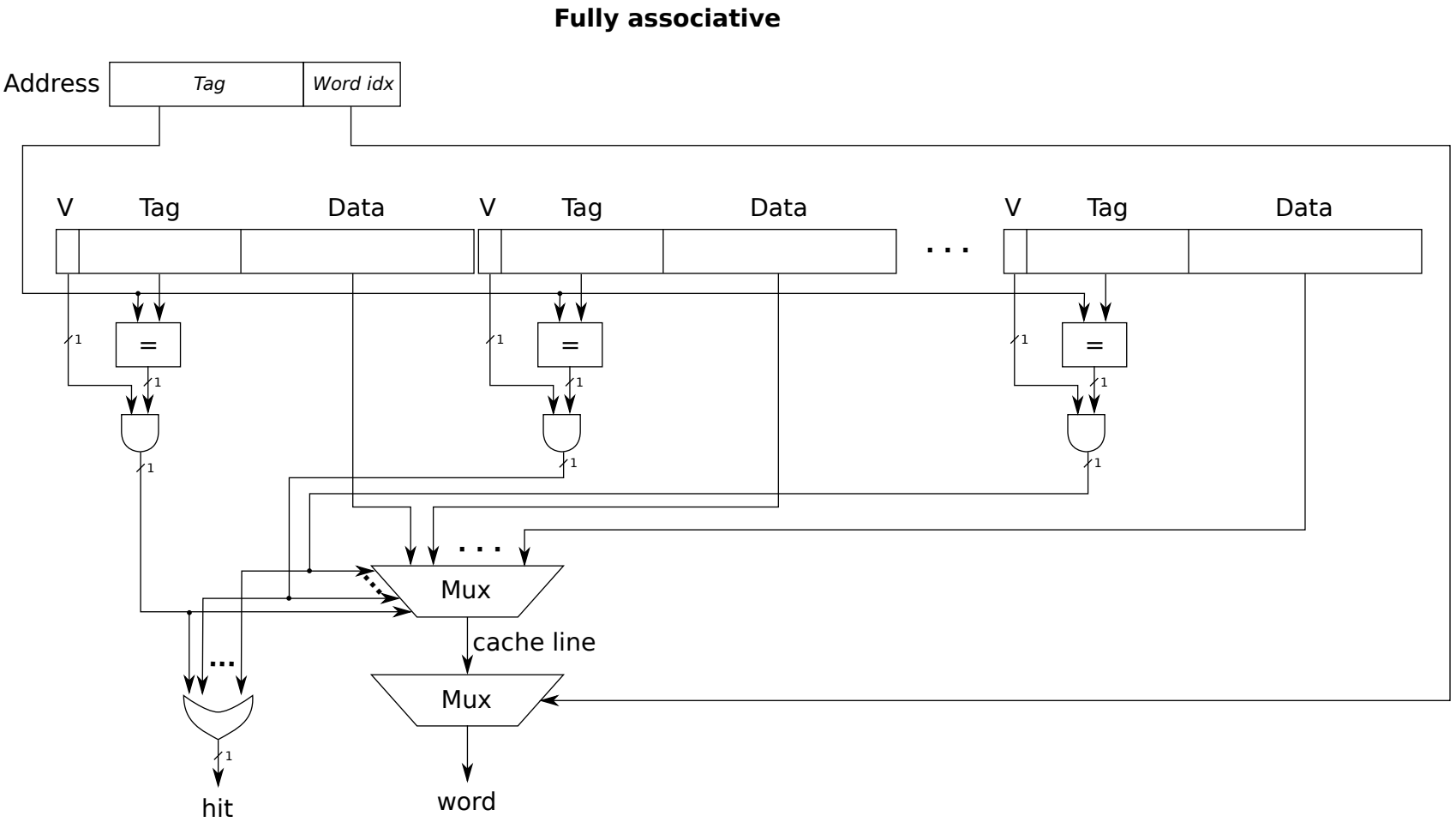
With fully-associative mapping, a memory block can be placed in *any* cache block.

- Blocks are only replaced when the cache is full
- There is no block field in the address: only tag and word
- Every cache block is searched simultaneously for a matching tag



This is *slower* and more *expensive*, but achieves the highest hit rate.

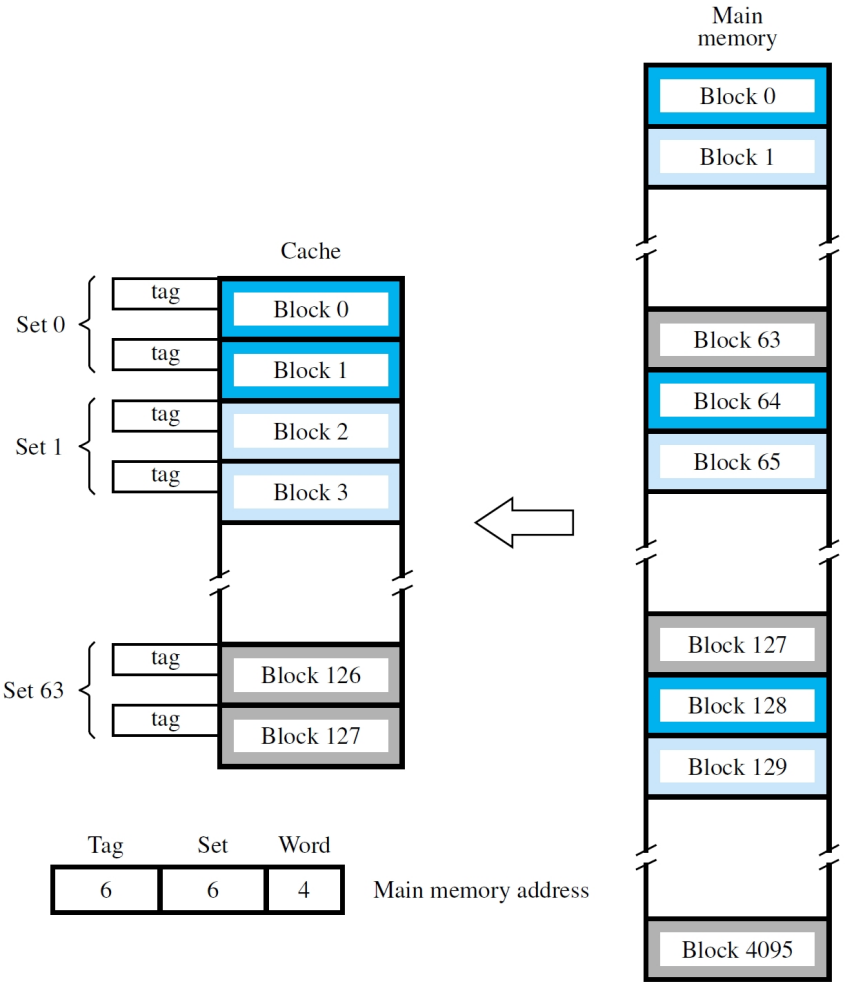
Fully-associative Cache



Set-associative Cache

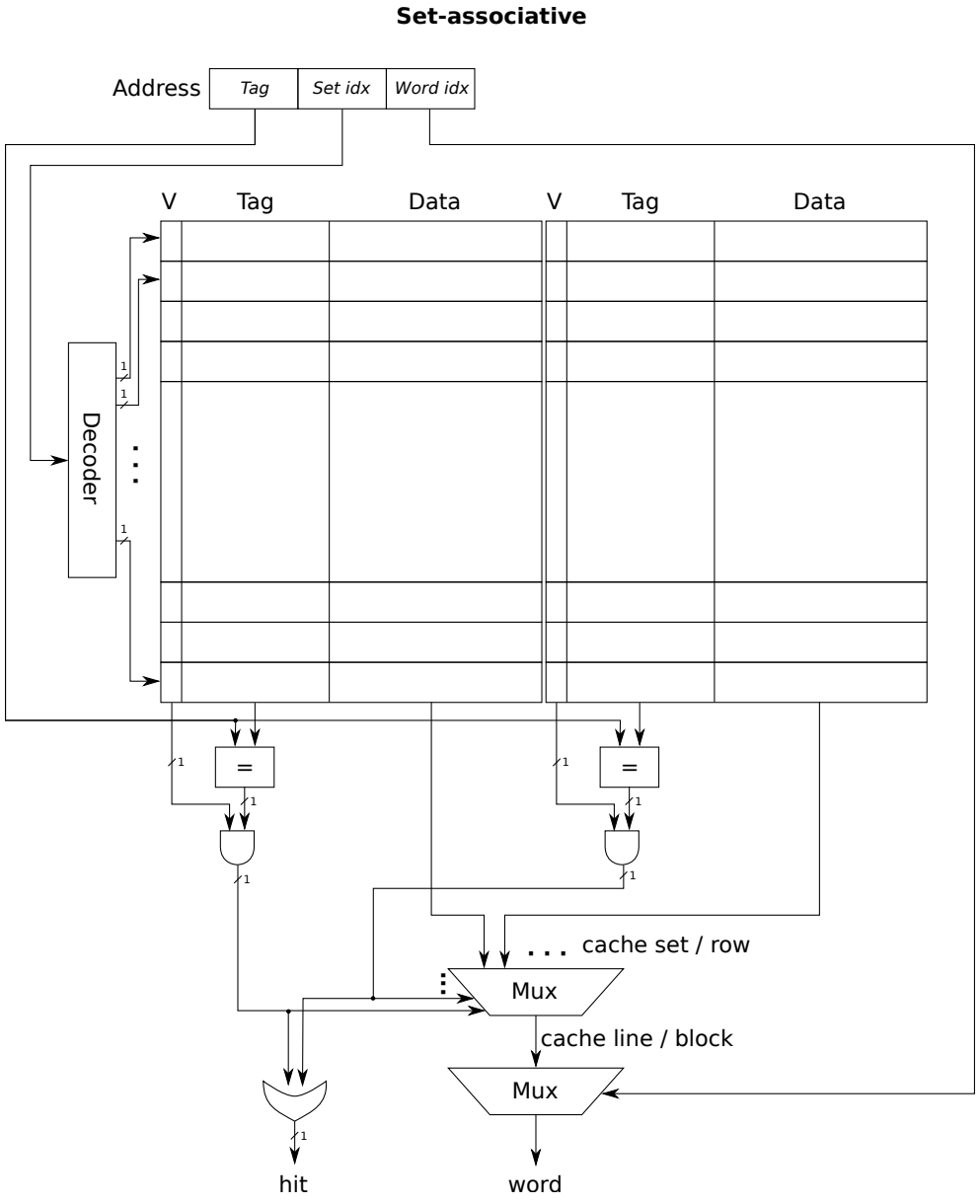
With set-associative mapping, a memory block can be placed in *limited number* of cache blocks.
k-way set-associativity:

- Blocks are grouped into sets of *k* blocks
- Memory blocks are *directly* mapped to a set
- The tags of the *k* blocks are searched in parallel



This strikes a trade-off between direct-mapped and fully associative caches, improving hit rate without the high cost of full associativity.

Set-associative Cache



Every Cache is Set-associative

Associativity determines the number of cache blocks in which a memory block may be placed.

Assuming a cache with n blocks:

- 1-way set-associative caches are direct mapped
  (there are n sets of one block)
- k-way set-associative (*e.g.*, $k \in \{2, 4, 8, 16, \ldots\}$)
  (there are n/k sets of k blocks)
- n-way set-associative caches are fully associative
  (there is one set with n blocks)

Block Replacement Policies

Block replacement (determining which block in the set to replace on a cache miss) is trivial for direct-mapped caches; a strategy for associative caches is needed, however.

- Least-recently-used (LRU): hardware tracks the relative timing of accesses to each block in the set
- First-in-first-out (FIFO): replacement rotates through blocks in the set
- Random: a block in the set is chosen at random for replacement

Each policy choice has pros and cons related to hardware complexity and resulting miss rate. See more possibilities here.

Writes to Cache

Depending on the organization of memory, writes to cache are handled in a variety of different ways.

There are two commonly used policies:

- *Write-through*: update the accessed block in the cache, if present, and main memory;
- *Write-back*: only write to the cache.

Write-through

Write-through simplifies memory system design at the cost of using more memory bandwidth and energy: each write to cache results in a write to main memory.

- Hit: write to both cache and main memory
- Miss: write only to main memory

Write-back

A *write-back* policy reduces memory bandwidth on most writes, but increases the complexity of block replacment, and complicates memory system design in general, especially for multiprocessors.

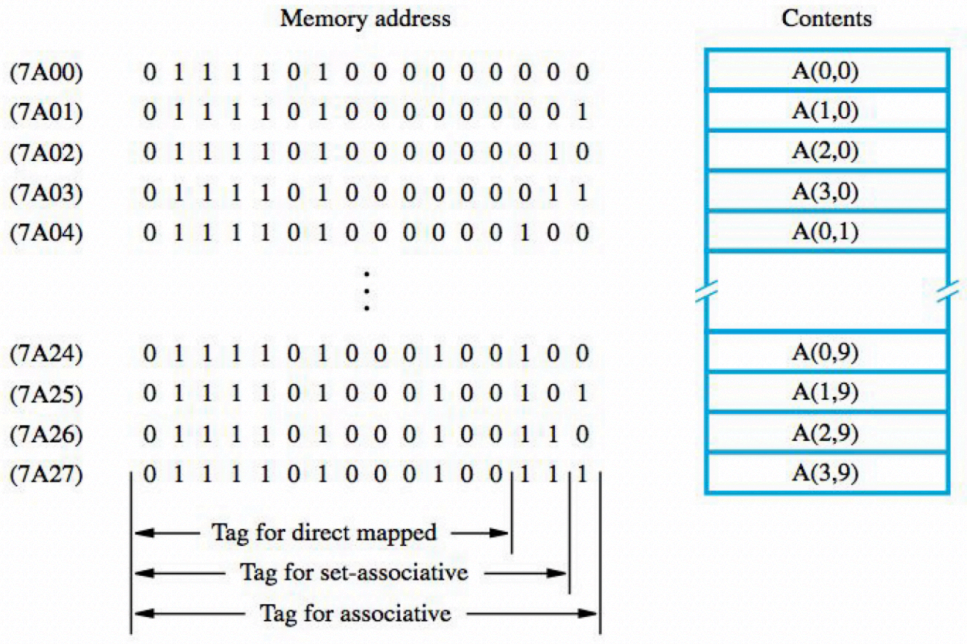
- Hit: write to the cache. Update main memory only when that cache block is removed from the cache. A *dirty bit* (or modified bit) is set to indicate cache block has been modified and is no longer identical to the block in main memory.
- Miss: first copy the block containing the addressed word from main memory into the cache, and then write the new word in the cache block.

Caching Example

Assume a 4x10 array of 16-bit numbers is stored in an array *A* in column-major order.

Let's look at cache behavior when we normalize the elements of the first row of *A* with respect to the average value of elements in that row.

$$A(0,i) \leftarrow \frac{A(0,i)}{\left(\sum_{j=0}^{9} A(0,j)\right)/10} \qquad \text{for } i = 0, 1, \ldots, 9$$




```
SUM := 0
for j := 0 to 9 do
        SUM := SUM + A(0,j)
end
AVG := SUM/10
for i := 9 downto 0 do
        A(0,i) := A(0,i)/AVG
end
```

Caching Example

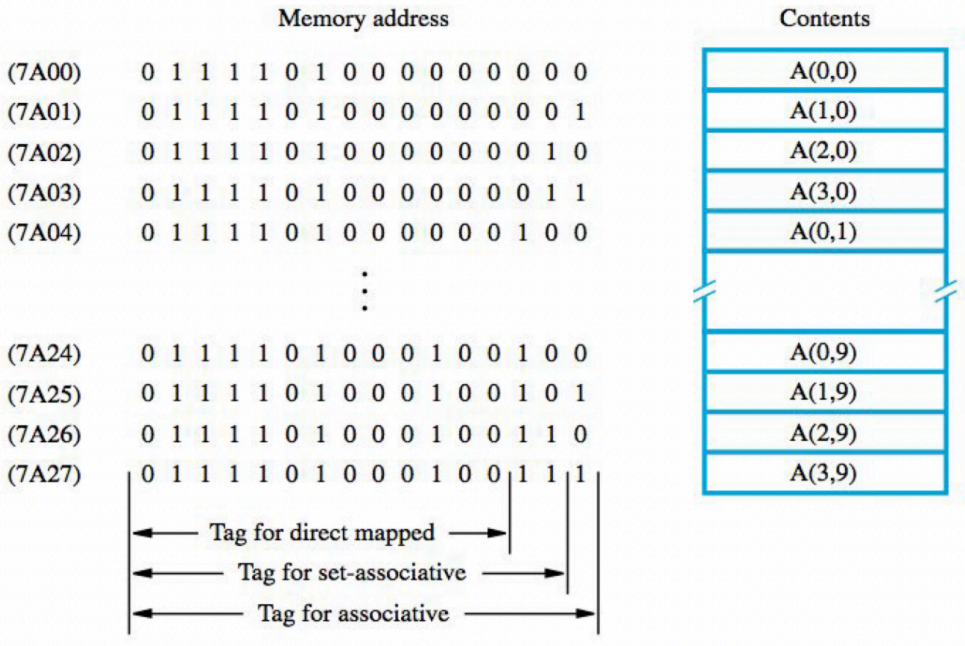
Consider a cache with the following characteristics:

- Memory word: 16 bits
- Word-addressable with 16-bit addresses
- Block size: one word
- Cache size: 8 blocks
- LRU replacement

Let's look at what happens for three different caches:

- direct-mapped
- fully associative
- 4-way set associative

$$A(0,i) \leftarrow \frac{A(0,i)}{\left(\sum_{j=0}^{9} A(0,j)\right)/10} \qquad \text{for } i = 0, 1, \ldots, 9$$




```
SUM := 0
for j := 0 to 9 do
        SUM := SUM + A(0,j)
end
AVG := SUM/10
for i := 9 downto 0 do
        A(0,i) := A(0,i)/AVG
end
```

Caching Example: Direct-mapped Cache Results

Everything maps to just two sets!

	Contents of data cache after pass:								
Block position	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$	$i = 6$	$i = 4$	$i = 2$	$i = 0$
0	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)	A(0,6)	A(0,4)	A(0,2)	A(0,0)
1									
2									
3									
4	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)	A(0,7)	A(0,5)	A(0,3)	A(0,1)
5									
6									
7									

Only two hits: when $i = 9$ and $i = 8$!

Caching Example: Fully-associative Cache Results

The cache lacks the capacity to store the working set.

Block position	Contents of data cache after pass:				
	$j = 7$	$j = 8$	$j = 9$	$i = 1$	$i = 0$
0	A(0,0)	A(0,8)	A(0,8)	A(0,8)	A(0,0)
1	A(0,1)	A(0,1)	A(0,9)	A(0,1)	A(0,1)
2	A(0,2)	A(0,2)	A(0,2)	A(0,2)	A(0,2)
3	A(0,3)	A(0,3)	A(0,3)	A(0,3)	A(0,3)
4	A(0,4)	A(0,4)	A(0,4)	A(0,4)	A(0,4)
5	A(0,5)	A(0,5)	A(0,5)	A(0,5)	A(0,5)
6	A(0,6)	A(0,6)	A(0,6)	A(0,6)	A(0,6)
7	A(0,7)	A(0,7)	A(0,7)	A(0,7)	A(0,7)

Only two misses in the second loop: when $i = 1$ and $i = 0$!

Caching Example: Set-associative Cache Results

Everything maps to a single set, but we have four ways.

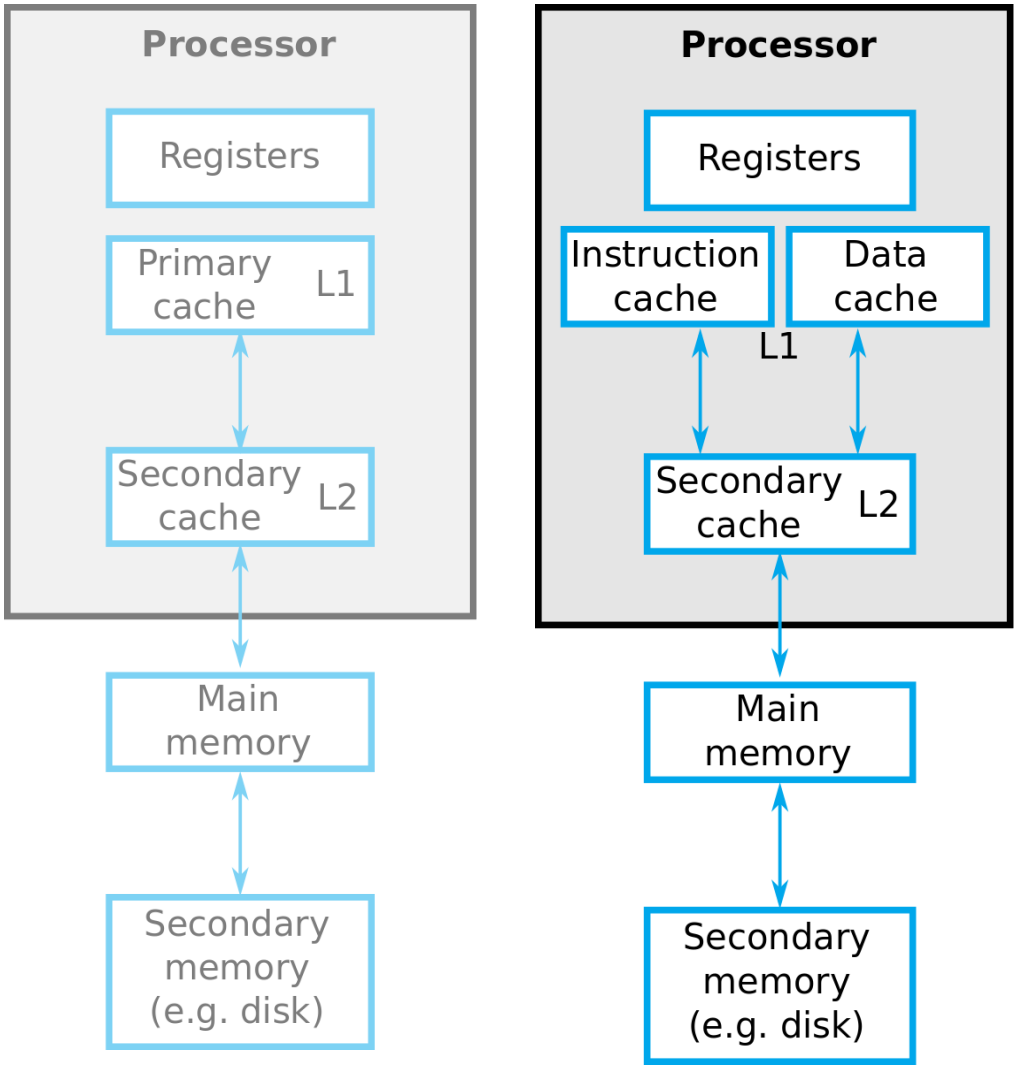
	Contents of data cache after pass:					
	$j = 3$	$j = 7$	$j = 9$	$i = 4$	$i = 2$	$i = 0$
Set 0	A(0,0)	A(0,4)	A(0,8)	A(0,4)	A(0,4)	A(0,0)
	A(0,1)	A(0,5)	A(0,9)	A(0,5)	A(0,5)	A(0,1)
	A(0,2)	A(0,6)	A(0,6)	A(0,6)	A(0,2)	A(0,2)
	A(0,3)	A(0,7)	A(0,7)	A(0,7)	A(0,3)	A(0,3)
Set 1						

Six misses in the second loop in this case: when $i \in \{0, 1, \ldots, 5\}$

Split L1 Cache

L1 is usually split into instruction and data caches; later levels are unified.

- *Harvard architecture*: unified L1 would slow things down
- Instruction and data access patterns are quite different
  - Instruction accesses are predictable: loops; basic blocks
  - Instruction accesses are read-only
  - Splitting L1 cache results in higher hit rates



Secondary Storage

Textbook§8.10

Secondary Storage

In addition to memory, computer systems often have additional storage.

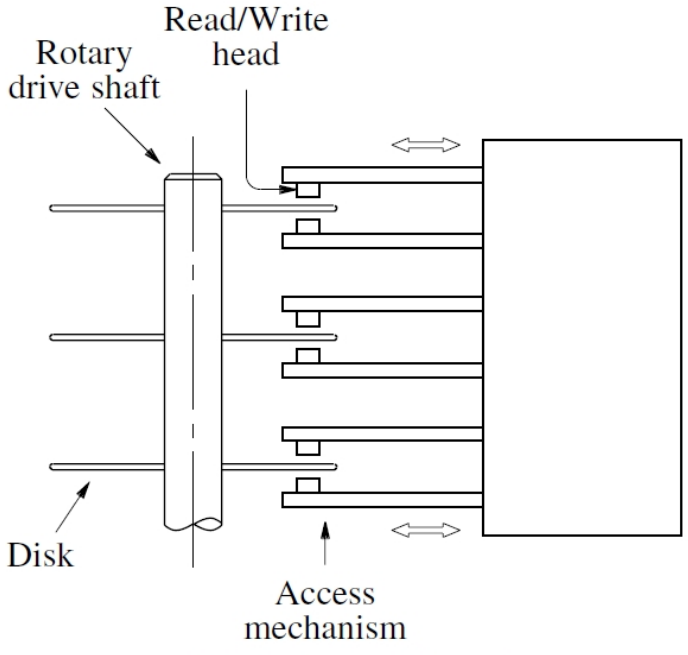
- Non-volatile, long-term storage
- Managed by the OS (not directly addressable by the CPU)
- Two main technologies today:
  - Flash-based solid-state drives (SSD): *e.g.*, in mobile devices and some laptops
  - Magnetic hard-disk drives (HDD): *e.g.*, in workstations, servers
- HDD are lower cost / bit at the moment, but this may change as technology evolves

Magnetic Hard Disk Drives

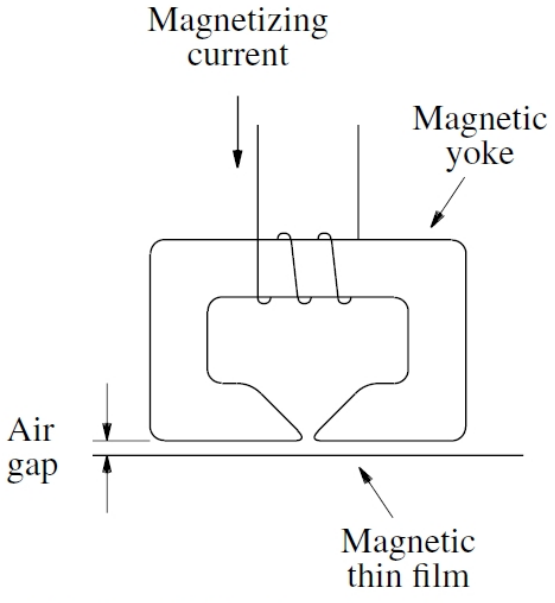
Hard drive drives consist of one or more magnetic platters on a common spindle.

- Platters are covered with a thin magnetic film
- Platters rotate on spindle at a constant rate (1000s of RPM)
- Read/write heads, close to the surface, detect bits stored as a magnetic field in concentric tracks
- The magnetic yoke and magnetizing coil in the head sense or change the polarity of the field on the surface of the platter

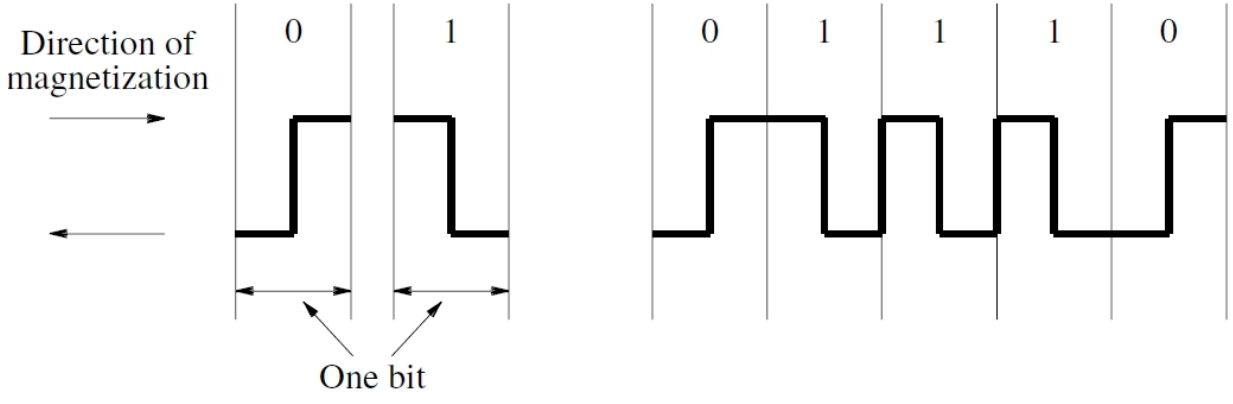
Magnetic Hard Disk Drives



(a) Mechanical structure



(b) Read/Write head detail

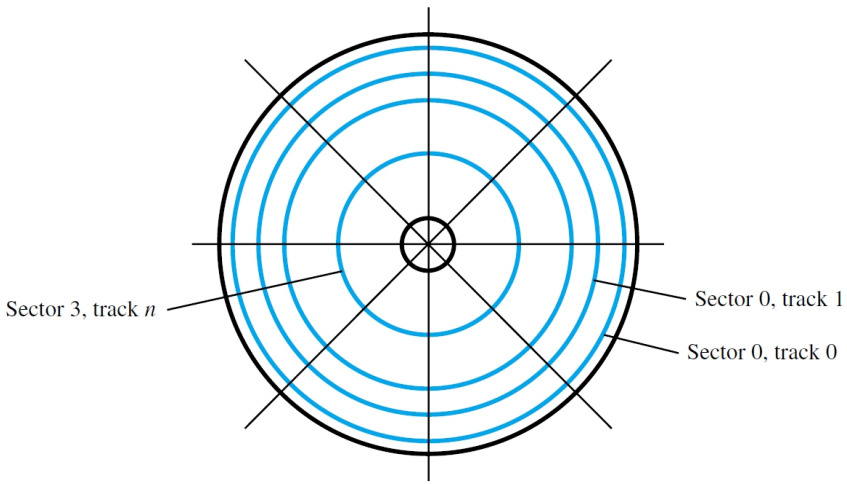


(c) Bit representation by phase encoding

Magnetic Hard Disk Drives

Each disk is divided into concentric tracks, and each track into sectors. A cylinder is a set of tracks on a stack of disks; such tracks can accessed simultaneously without moving the read/write heads.

- Data is written sector-by-sector (*e.g.*, 512 B)
- Formatting information (including track/sector markers) and error-correcting code (ECC) information is stored on disk
- The file system is on disk, too: data structures that the OS uses to keep track of files



HDD Access Time

- Seek time: time required to move the read/write head to the proper track
  - Depends on the initial position of the head
  - Typically 5 to 8 ms
- Latency: time to read addressed sector after the head is positioned over the correct track
  - On average, half the time for a full disk rotation

Access time = seek time + latency

- Flash access time is typically 35 to 100 μs (100x faster)

Virtual Memory

Textbook§8.8, 8.9

Virtual Memory

Physical memory capacity is almost always smaller than the address space size.

- A large program, or many smaller concurrent programs, may require more physical memory than is available
- Virtual memory uses secondary storage to hold data in excess of memory capacity (in “swap file” or “page file”)
- Virtual memory is the lowest tier of the memory hierarchy
  - Magnetic disk (5 ms) is five orders of magnitude (10^5) times slower than SDRAM (15 ns)
  - Virtual memory must be carefully managed (by the OS) to limit disk accesses

Virtual Memory

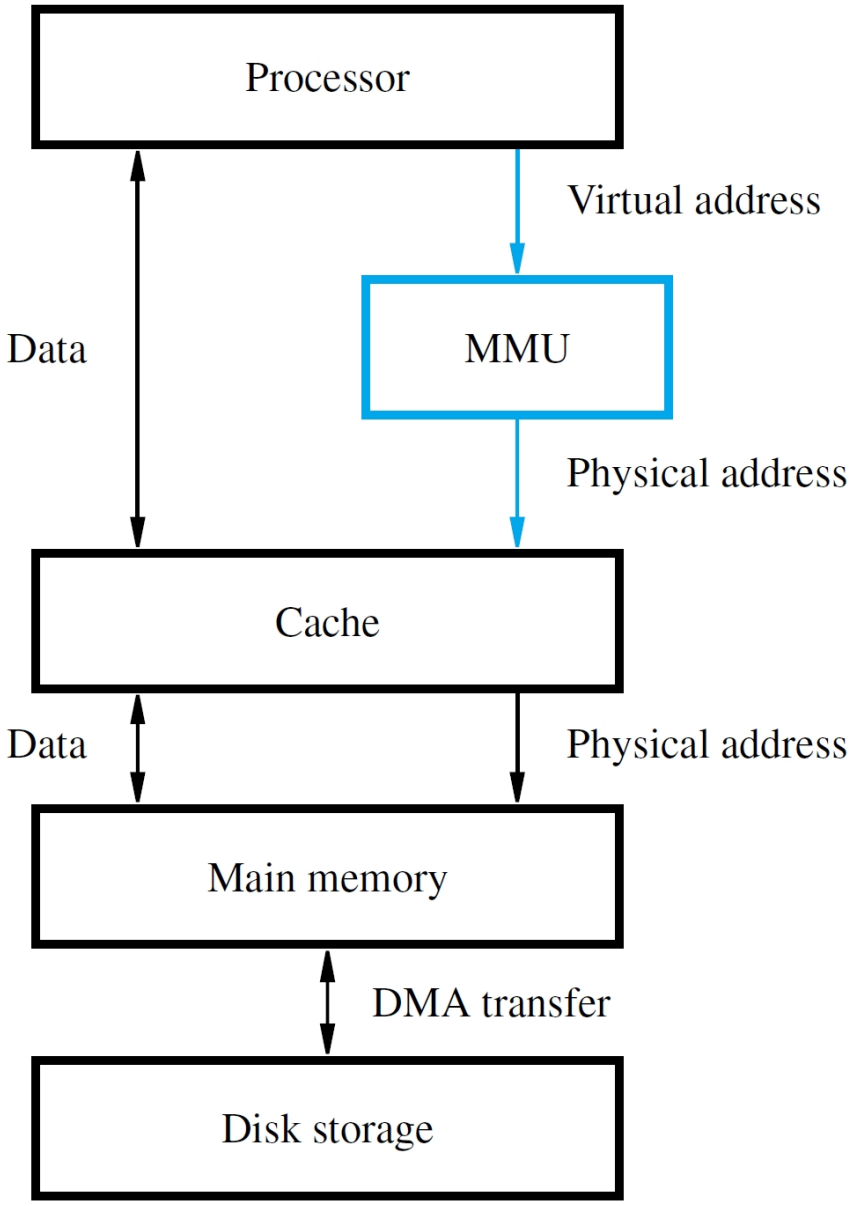
- Programs are written assuming exclusive access to the whole address space
- Processors access virtual addresses (logical addresses)
- Virtual addresses must be translated into physical addresses
- This works sort of like caching, but software (OS) managed:
  - When the requested data is in physical memory (*hit!* a valid translation exists), proceed like usual
  - When it is not (*miss!* there is no valid translation), the requested data must be moved from secondary storage to physical memory (replacing something else)
  - Fully associative mapping is used, *e.g.*, with LRU replacement

Memory Management Unit

A hardware memory management unit (MMU) performs translation from virtual addresses to physical addresses.

- The MMU maintains a table of translations from virtual to physical addresses
- When no physical address exists for a given application and virtual address, an interrupt occurs (page fault), and the OS intervenes
- The OS transfers the desired data from disk to memory using DMA (first copying some memory to disk, if physical memory is full)
- The MMU is then updated to include the new translation

Virtual Memory Organization



Address Translation

Virtual memory is organized into pages.

- Pages are fixed* size, often 2-16 KB
  - Pages are much larger than cache blocks
  - Disks have high access times, but bandwidth in MB/s
- For translation, addresses are divided into two fields
  - Upper bits give the virtual page number (VPN)
  - Lower bits give the offset of a word within a page
- Translation preserves offset bits, but replaces VPN with the appropriate page frame number (*i.e.*, physical page number)
- The page table (stored in main memory) keeps track of the mapping between virtual and physical page numbers

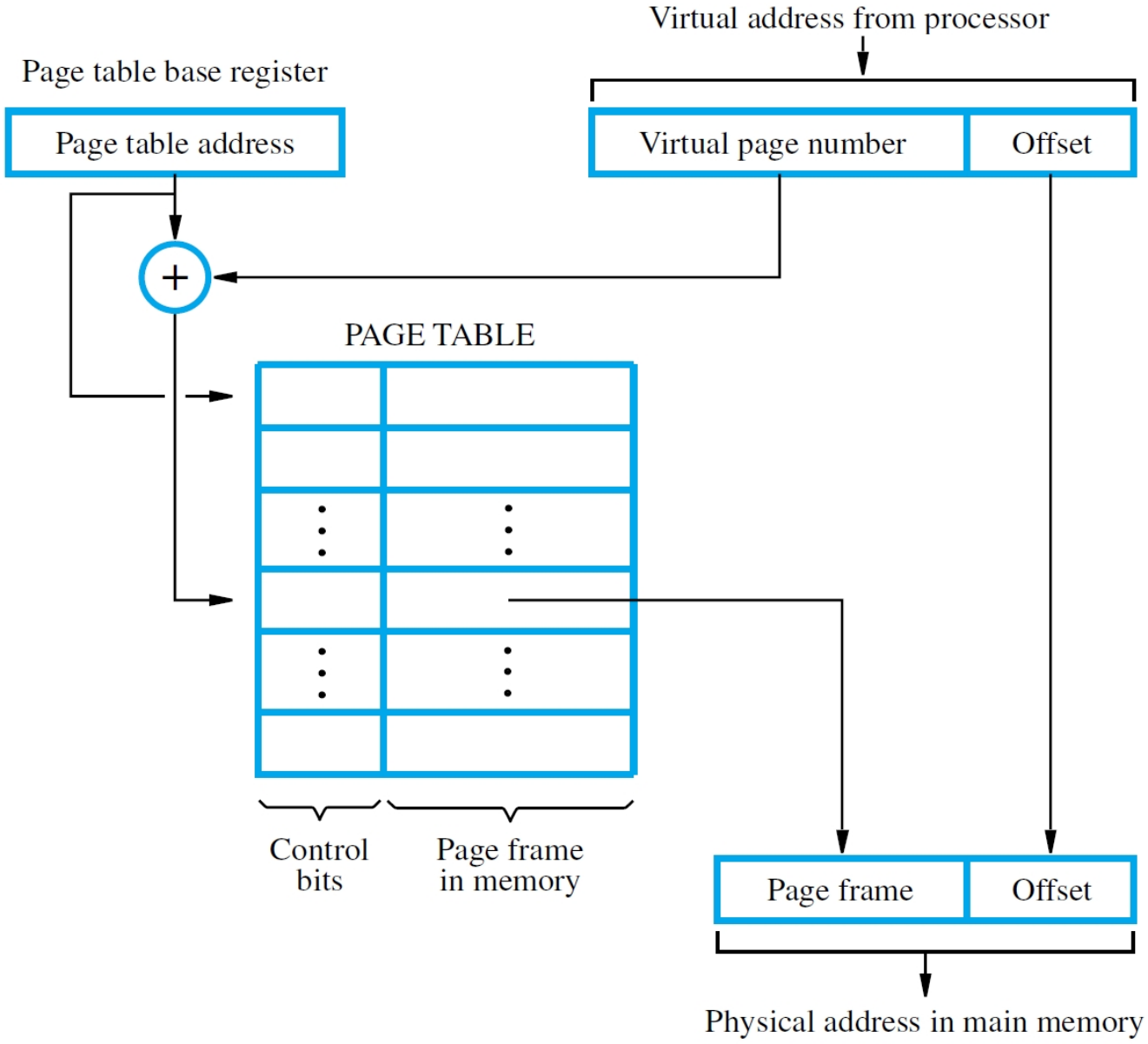
Address Translation

Virtual memory is organized into pages.

- Pages are fixed* size, often 2-16 KB
  - Pages are much larger than cache blocks
  - Disks have high access times, but bandwidth in MB/s
- For translation, addresses are divided into two fields
  - Upper bits give the virtual page number (VPN)
  - Lower bits give the offset of a word within a page
- Translation preserves offset bits, but replaces VPN with the appropriate page frame number (*i.e.*, physical page number)
- The page table (stored in main memory) keeps track of the mapping between virtual and physical page numbers

* except when they're variable size

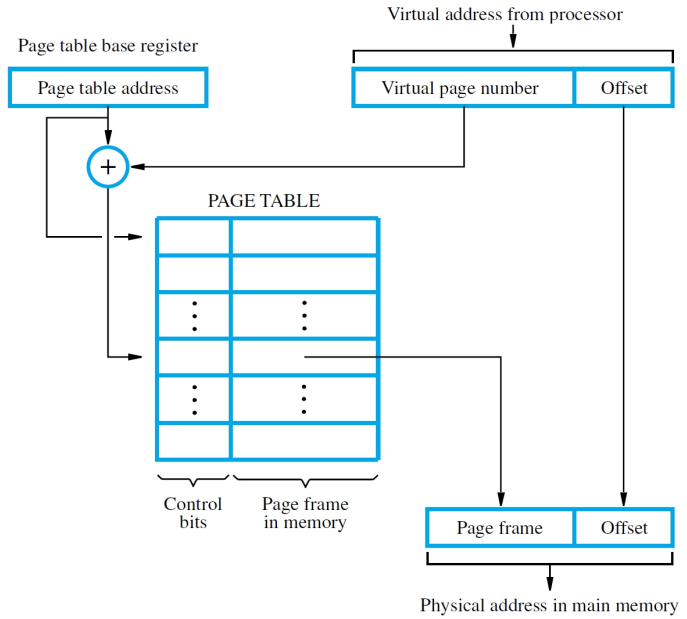
Page Table


Virtual address from processor
Page table base register
Page table address
Virtual page number
Offset
PAGE TABLE
Control bits
Page frame in memory
Page frame
Offset
Physical address in main memory

Page Table

The page table stores all translations from virtual to physical pages.

- The MMU stores the start address of the page table: page table base register (PTBR)
- PTBR + VPN = address of the page table entry (PTE) for the given VPN
- Each PTE maintains control bits (*valid? modified?*)
- Each PTE also stores the page frame number if the page is in memory
- Otherwise, it may indicate where on disk the page can be found
- PTEs also track process information, read/write permission, etc

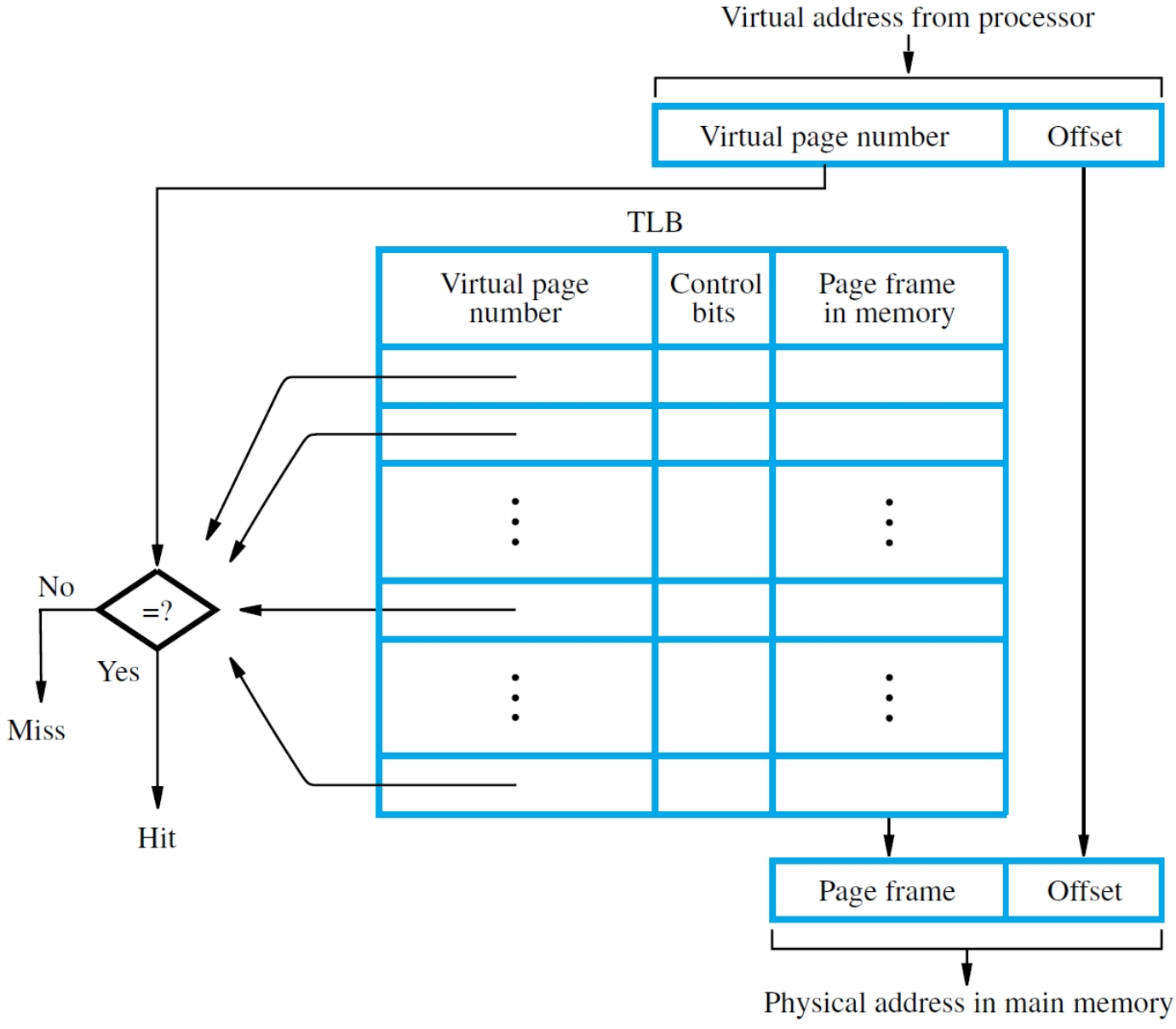


Translation Lookaside Buffer (TLB)

The MMU must perform translation for *each* memory access (*i.e.*, every fetch, every load or store). If each translation requires a references to the page table, this is slow!

- When physical memory is large, the page table has many entries
- It isn't practical to store the page table in the MMU
- The translation lookaside buffer (TLB) in the MMU caches recently accessed PTEs
- The TLB is fully associative; on a miss, the full table is accessed, and TLB updated (*e.g.*, using LRU replacement)
- Split L1 caches? Two TLBs: one for instructions accesses, another for data accesses

Translation Lookaside Buffer (TLB)



Page Faults

- A page fault occurs when a virtual address has no corresponding physical address
- The MMU raises an interrupt so the OS can place the appropriate page in the memory, and create the corresponding translation
- The OS uses LRU to select a page frame to replace, writing the old frame to memory if necessary
- Handling page faults takes a long time, requiring disk accesses!
- Usually the OS selects another program to execute while waiting
- The suspended program restarts later when the page is ready

Conclusions

This set of lectures introduced how computer system memory is organized. We've looked at:

- Memory technology: SRAM, DRAM, ROM, etc
- Direct memory access (DMA) hardware that assists with large memory transfers
- Caches for reducing the latency of memory accesses
- Secondary storage: hard-disk drives, and solid-state drives
- Virtual memory, which expands physical memory size using secondary storage

Next we'll look at how processors themselves are implemented, and how they execute instructions.