Contents

Final Notes on Textbook Chapter 7

Introduction to Memory Systems

Memory System Speed Requirements

Introduction to Caches
Outline of Slide Set 8 for Lecture Section 01

Final Notes on Textbook Chapter 7

Introduction to Memory Systems

Memory System Speed Requirements

Introduction to Caches
Final Notes on Textbook Chapter 7

In ENCM 369 this term, we have skipped Section 7.4—Multicycle Processor—to make sure to have time for other important topics, mostly memory systems and floating-point numbers.

We’ll also skip . . .

- Section 7.6: HDL (hardware description language) representation of processor designs. HDLs are a big topic in ENEL 453.

- Section 7.7: Exception handling, as an extension of the multicycle processor of Section 7.4. It’s somewhat sad to skip this, because the FSM of textbook Figure 7.64 is a really cool FSM.
More about skipped sections in Chapter 7

- Section 7.8: Advanced Microarchitecture. Simple pipelining, as described in Section 7.5, is a really important idea in processor design. But the state of the art in processor design has gone far beyond simple pipelining! See Section 7.8 (and ENCM 501) for an overview of how current processor chips work.

- Section 7.9: x86 Microarchitecture. The history and present status of Intel Corp. is fascinating from both business and technical viewpoints!
Figure 7.67 from Section 7.8

This is a sketch of a pipelined datapath designed to start two instructions per clock cycle, whenever possible . . .

Image is Figure 7.67 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.
The pipeline? No, many pipelines!

This is not final exam material!

In 2017, a quad-core processor for a high-end laptop or desktop computer might have 6 or so pipelines in each core, so could have a total of 24 pipelines.

And each pipeline might have many instructions “in flight” at any given moment, so the processor chip could at that moment have hundreds of instructions started but not yet finished!

My opinion: It’s astonishing that these chips are not very expensive, and actually work!
Outline of Slide Set 8 for Lecture Section 01

- Final Notes on Textbook Chapter 7
- Introduction to Memory Systems
- Memory System Speed Requirements
- Introduction to Caches
Introduction to Memory Technology

Chapters 6 and 7 of the textbook, and related lectures and labs suggest that memory is simple.

In real-world computers, the memory systems are usually not at all simple!

Expect to read Sections 8.1 to 8.4 of the textbook multiple times—don’t give up if it seems too confusing the first time through!
Know the difference between MEMORY and DISK STORAGE!

MEMORY is an ELECTRONIC system with NO MOVING PARTS.

It is constructed on silicon chips using millions or billions of tiny transistors, capacitors and wires per chip.

It is mainly used for INSTRUCTIONS and DATA of RUNNING PROGRAMS.

The next slide shows the main memory circuits of a laptop from 2007 . . .
Know the difference between MEMORY and DISK STORAGE! (continued)

DISK STORAGE is partly electronic and partly MECHANICAL with LOTS of MOVING PARTS.

Access time to disk is MUCH, MUCH, MUCH LONGER than access time to memory.

Cost per bit of disk is MUCH less than cost per bit of memory.

Disk storage is mainly used for FILE SYSTEMS.

The next slide shows a magnetic disk drive with the top of its case removed . . .
Image is Figure 8.19 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.
Describing Capacity: kilobytes, megabytes, gigabytes, kilobits, megabits, gigabits

B means byte (8 bits). b means bit.

But often B and b get mixed up, so watch out for mistakes!

When describing memory size, powers of two are always used (e.g., 1 KB = 1024 bytes, 1 MB = 1,048,576 bytes).

When describing disk capacity or data transfer rate, powers of ten are more frequently used (e.g, 500 GB = 500,000,000,000 bytes).
Memory capacity definitions

1 KB = 1 kilobyte = $2^{10}$ bytes = 1,024 bytes.

1 MB = 1 megabyte = $2^{20}$ bytes = 1,048,576 bytes.

1 GB = 1 gigabyte = $2^{30}$ bytes = 1,073,741,824 bytes.

So, how many one-bit memory cells does it take to make a 256 MB memory circuit?
Does it bother you that 1 Mb might or might not be exactly 1,000,000 bits?

It should! Engineers should try to avoid ambiguity and imprecision in technical communication!

IEEE standard 1541—unfortunately not used as much as it could be—proposes new prefixes and symbols: kibi- (Ki), mebi- (Mi), gibi- (Gi), etc., for powers of two.

Example: $1 \text{ Mib} = 1 \text{ mebibit} = 2^{20}$ bits. These prefixes are nice, but won’t be used in ENCM 369—we will stick with the more commonly used prefixes and symbols.
ROM and RAM: ROM

ROM: “read-only memory”.

ROM is useful in many simple embedded systems, where programs do not change. (The programs in these systems are sometimes called “firmware” instead of “software”.)

An important use of ROM in smartphones, laptops, desktops, and servers is holding instructions that start the process of loading the operating system kernel into memory.
ROM and RAM: RAM

RAM: “random-access memory”.

“Random-access” means accesses do not have to be in sequence by address. (But ROM allows random access for reads.)

A more accurate (but unpronounceable) name would be RWM: “readable-writable memory”—being writable is what makes RAM different from ROM.
Silicon memory technologies

Silicon is a semiconductor, so often the term “semiconductor memory” is used.

Important types of silicon memory are **DRAM, SRAM**, and **flash**.

See Sections 5.5.2–5.5.4 of the textbook for brief notes on DRAM and SRAM.

See Section 5.5.6 of textbook for brief notes on flash memory.
DRAM: “dynamic RAM”

DRAM is **cheap**, compared to SRAM.

DRAM has one transistor plus one very, very small capacitor per bit—**very high density**

DRAM access time is typically tens of nanoseconds, much longer than a processor clock cycle, which is currently 0.5 ns (500 ps) or less.
DRAM is used for main memory in smartphones, tablets, laptops, desktops, and servers, in various “flavours” such as DDR3 SDRAM (3rd generation double-data-rate synchronous DRAM).

DRAM chips are installed on DIMMs (dual inline memory modules) that plug into computer motherboards.
Two SO-DIMMs from a 2007 MacBook

“SO” means “small outline”.

4 DRAM chips per DIMM are “hiding” under the white paper labels. These two 256 MB DIMMs were replaced with two 1 GB DIMMs to upgrade RAM from 512 MB to 2 GB.
**SRAM: “static RAM”**

SRAM typically uses 6 transistors per bit—lower density than DRAM.

SRAM is more expensive than DRAM.

SRAM is much faster than DRAM—access time can be as low as 1 or 2 processor clock cycles.

These days, SRAM is usually on the same chip as the processor cores. See, for example, the L3 cache in textbook Figure 7.80, page 465, on the next slide.

Chip dimensions are about 21 mm by 14 mm.

Each computer in ICT 320 has a chip similar to this. The technology is 7 years old, but is still awesome in its complexity!
Image is Figure 7.80 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.
Flash memory

Compared to DRAM, flash is cheaper per bit, but much slower. The name “flash” comes from the electrical process used to erase a block of memory, and has nothing to do with relative speed.

Unlike DRAM or SRAM, flash maintains data when power is turned off.

Flash is useful for file systems for computers where hard disks would be too slow or too bulky.

“Solid-State Drives”, which have replaced magnetic disks in many current laptop and desktop computer designs, store files in flash memory.
Three things containing flash memory
Outline of Slide Set 8 for Lecture Section 01

Final Notes on Textbook Chapter 7

Introduction to Memory Systems

Memory System Speed Requirements

Introduction to Caches
Memory System Speed Requirements

Suppose we are trying to build a memory system for a MIPS processor with a simple 5-stage pipeline like the one we’ve just studied.

What requirements does the processor design place on the memory system?
Keeping things relatively simple

When we start looking at caches, we’ll just look at supporting instruction fetch, LW and SW.

We will not worry about these extra complexities . . .

- To implement the entire MIPS instruction set we would need also to support loads and stores of bytes, halfwords (16-bit chunks) and doublewords (64-bit chunks).
- Multiple issue (starting 2 or more instructions in 1 clock cycle within a single core) requires more complex memory interfaces.
- Supporting multiple cores also requires more complex memory interfaces.
A simple model of memory:
A BIG ARRAY OF WORDS

A definition for the word *model*: A description of a system that, while not perfectly accurate, helps explain or predict the behaviour of the system.

*Let’s make a sketch of the memory model introduced at the beginning of ENCM 369...*

*How was this model modified in Chapter 7 of the textbook?*
Can the simple big-array-of-words model be used for a hardware design?

*Could memory be organized as a big array (or two big arrays) of 32-bit groups of one-bit DRAM cells?*

*Could memory be organized as a big array (or two big arrays) of 32-bit groups of one-bit SRAM cells?*
What can be done?

Can we combine

- a very fast small-capacity memory (SRAM)
- and a slower large-capacity memory (DRAM)

to make a system that performs almost as well as an impossible-to-build very fast large-capacity memory?

Answer: YES, such a system can often—but not always—perform like very big, very fast memory.

The small, very fast part of the system is called a cache.
Memory organization in real computers

See page 1 of the “Introduction to Memory Organization” lecture document. This is way more complicated than the simple big-array-of-words model!

We will not study all parts in detail, but will focus on key components:

▶ caches (textbook Section 8.3);
▶ address translation and virtual memory (Section 8.4).
Outline of Slide Set 8 for Lecture Section 01

Final Notes on Textbook Chapter 7

Introduction to Memory Systems

Memory System Speed Requirements

Introduction to Caches
What is a cache?

A cache is a small, very fast memory system that holds copies of some, not all of the instructions or data in main memory. Processor cores look in caches, NOT in main memory, to find instructions and data.
System with one level of cache, no address translation

This is from page 2 of “Introduction to Memory Organization”. Keep this picture in mind as we discuss cache organization.
Simple examples of cache operation

We’ll consider the system with one level of cache and no address translation.

*What happens in instruction fetch?*

*What happens in the data memory access step of a load instruction?*

*What about data memory access in a store instruction?*
Definitions: *Hits* and *Misses*

*Cache hit*: Event in which the processor finds the instruction or data it needs in the cache.

*Cache miss*: Event in which the instruction or data the processor needs is NOT in the cache. A miss causes a long stall while instructions or data are copied from the main memory to the cache.

(These definitions apply to instruction fetches and data memory *reads*. We will have to modify them a little for data memory *writes*.)
Capacity: Cache vs. Main Memory

A typical recent medium-quality laptop (Apple MacBook Air, March 2015) has 4GB of main memory—DRAM.

It can cache up to 3MB of instructions and data in SRAM within the processor chip.

What fraction of main memory contents can be cached?

Does that mean that caches are not very useful?
Locality of Reference

This is a property computer **programs** have, to varying degrees.

(It doesn’t make sense to say that a processor or memory system has good or bad locality of reference.)

Locality of reference is what makes caches valuable.
Temporal locality of reference

*Temporal* means “related to time”.

If a program accesses a memory word, the program is likely to access the same word in the near future.

*What are some examples of temporal locality?*
**Spatial locality of reference**

In this context, *spatial* means “related to the *address space* of a computer”, not “related to physical space”.

If a program accesses a memory word, the program is likely to access other *words with nearby addresses* in the near future.

*What are some examples of spatial locality?*
Locality of Reference and Cache Hit Rates

Programs with high degrees of locality of reference tend to have high cache hit rates—95 to 99% of memory accesses are cache hits.

So caches can enable many programs to have performance close to performance of a hypothetical computer in which all memory accesses take 1–2 clock cycles.

*What about programs with bad locality of reference?*