CPSC 2310 Exam 1 Study Guide
Does not include any C review

Chapter 1

Phases of the Compiler System

1. Preprocessor *(.i file)* — Modifies files according to preprocessor directives (#include, #define, etc)
2. Compiler *(.s file)* — Translates source files into an assembly-language program
3. Assembler *(.o files)* — Translates assembly language to machine instructions, produces a relocatable object file (which appears like gibberish)
4. Linker (*executable file*) — Links together many object files into a single executable.

Hardware Organization of a System

CPU — **Central Processing Unit**. Responsible for actually performing machine instructions

ALU — **Arithmetic Logic Unit**. Performs arithmetic and logical operations.

PC — **Program Counter.** Word sized register that holds the address of the current instruction

USB — **Universal Serial Bus**. Allows external devices to connect to computer

System Buses — Carries information between components

System I/O Devices — Connect outside information to the computer (mouse, keyboard, screen, etc)

System Cache — Allows for fast access to frequently used data. Much much faster than even RAM.

There are layers of cache: L0 (fastest), L1, L2, L3, RAM, Hard Storage (slowest)

Operating Systems & Hardware Abstractions

Primary Purposes of OS:

- Protect the hardware from misuse
- Provide programs with simplified mechanisms to manipulate low-level devices
- Allocate system resources

Processes — Abstraction for a running program, allows for easier OS-level concurrency.

Threads — Inside a single process, multiple execution units, allow for easier async I/O. More lightweight than processes.

Virtual Memory — Abstraction over main memory, give each process the impression they have access to main memory

Program Code & Data (0th address) — Size unchanged after program start

Heap — Dynamic memory, grows and shrinks as needed (grows away from 0th)

Shared Libraries — stdlib, stdio, etc.

Stack — Grows and shrinks as needed (grows towards 0th)

Kernel Virtual Memory — Reserved for OS

Networking — Network driver can be thought of as just another I/O device

Virtual Machine — Emulation of an entire computer, including hardware, OS, and programs

Concurrency & Parallelism

Concurrency — the general concept of a system with multiple, simultaneous activities

Thread-Level — Rapidly switching between threads and processes gives illusion of parallelism

Parallelism — refers to the use of concurrency to make a system run faster

Multiprocessor System — Multiple processors, one Operating System

Unique L1 and L2 chaches

Shared L3 cache

Hyper-Threading — allows a single CPU to execute multiple control flows, by duplicating some parts of the CPU

Instruction-Level Parallelism — Modern CPUs can execute multiple instructions at once.

Static — Determined at compile time which instructions to parallelize

Dynamic — Determined at runtime by the processor which instructions to parallelize

The C Language

- Developed in 1969 by Dennis Ritchie at Bell Labs
- "Portable assembly"
- Small and simple language
- Includes a standard library

Binary, Decimal, and Hexadecimal

Binary — Base 2. May or may not have $_2$ after

Decimal — Base 10.

Hexadecimal — Base 16. Usually starts with 0x

<u>Binary → Decimal</u>

Multiply each power of 2 by the relevant bit

1110 0001

$(1)(2^0) + (0)(2^1) + (0)(2^2) + (0)(2^3) + (0)(2^4) + (1)(2^5) + (1)(2^6) + (1)(2^7)$

1 + 32 + 64 + 128 = 225

<u>Decimal → Binary</u>

Divide by two repeatedly until you get 0, then reconstruct

55 / 2 = 27 r 1
27 / 2 = 13 r 1
13 / 2 = 06 r 1
6 / 2 = 3 r 0
3 / 2 = 1 r 1
1 / 2 = 0 r 1
55 = 0011 0111

<u>Binary → Hexadecimal</u>

Group into 4s and convert each one individually according to its sum

1100 0110 1000 0111

<u>Hexadecimal → Binary</u>

Convert each digit into its 4-bit decimal representation and string them together

0xFABC = 1111 1100 1101 1110

A 6 8 7

1100 0110 1000 0111 = 0xA687

Writing 2^n in Hexadecimal

Write 1 with n 0 zeroes, and then convert to hex by grouping.

2^5 = 0010 0000 = 0x20

Boolean Algebra

Bitwise Not (~)

Inverts all the given bits
~10001110 = 0111 0001

Bitwise Or (|)

For a | b, 1 1 where either is 1
0110 | 1010 = 1110

Leftshift (<<)

Just logical
a << k = a * 2^k
Remove k leftmost bits, and add k 0s on the right

1100 1111 << 3 = 0001 1001

Bitwise And (&)

For a & b, 1 where both a and b are 1
0110 & 1011 = 0100

Bitwise XOR (^)

For a ^ b, 1 where either is 1 but not both
0110 ^ 1010 = 1100

Rightshift (>>)

Can be logical (unsigned) or arithmetic (signed)
a >> k = a / 2^k

If Logical
Remove k rightmost bits, and add k 0s on the left
1000 0001 >> 4 = 0001 0000

If Arithmetic
Remove k rightmost bits. If the significant bit of the original was 1, add k 1s, else add k 0s
1110 1100 >> 2 = 1011 0011

Data Sizes

Word Size — Maximum number of bits the processor can process at once, typically 8 bytes (64 bits)

Virtual Address Size is 2^w - 1 (4 GB on 32-bit systems, 16 exabytes on 64-bit systems)

C Declaration		**Size (bytes)**	
Signed	**Unsigned**	**32 Bit Systems**	**64 Bit Systems**
char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned int	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char*		4	8
float		4	4
double		8	8

Multiple-Byte Objects

When an object spans multiple bytes, you need to make a choice about

- The address of the object
- What order to put the bytes

  Little Endian — Least Significant Byte to Most Significant Byte

  Big Endian — Most Significant Byte to Least Significant Byte

  2147483647 = 0x7FFFFFFF

  Could be stored as FFFFFF7F (Little Endian) or 7FFFFFFF (Big Endian)

  Usually, endianness is completely invisible to the programmer

  <u>When Endianness Matters</u>

  - When communicating over a network, network devices must be cognizant of transmitting data in the correct order according to networking standards
  - When representing an address in assembly.
  - Occasionally when typecasting in C

Representing Strings

In C, they are an array of characters terminated by a null byte (\0000)

Characters are *encoded* into numbers, by some text encoding (ASCII, Unicode, etc)

Representing Integers (Signed vs Unsigned)

For signed bits, use the first bit to represent the sign of the number.

The range for unsigned data is 0 to (2^w - 1)

The range for signed data is (-2^{w-1}) to (2^{w-1} - 1)

Ways to Represent Signed Integers

- *Sign-Magnitude*. The value of the integer bits added together, except for the most significant bit, the most significant bit tells you the sign of the number (1 = negative, 0 = positive)
- *1's Compliment*. If num is positive (MSB is 0), add up the values. If negative (MSB is 1), complement the bits and then add them.
- *2's Compliment*. Compliment the bits and then add one. The most significant bit is essentially -128

Converting Between Integer Types

When converting up, no data will be lost, but when converting to a smaller representation, the lower bits will be dropped. Additionally, consider if the conversion is between a signed and an unsigned value, additional complications apply

RH Side (From)	LH Side (To)	Method
`unsigned char`	`char`	The bit pattern is preserved, the highest order bit becomes sign bit
`unsigned char`	`short`	Zero extend
`unsigned char`	`unsigned short`	Zero extend
`unsigned char`	`unsigned long`	Zero extend
`unsigned short`	`char`	Preserve low-order byte (lose high order byte)
`unsigned short`	`long`	
`unsigned short`	`unsigned char`	
`unsigned long`		
`unsigned long`		
`unsigned long`		
`unsigned long`		