Introduction

Chapter 1

Overview

Reading: Text chapter 1, "The Story of Mel" available on the web

Plan

- Introduction
  - What did you learn in 1st year
- Computer Architecture
- Levels of Programming Language
  - Machine Language
  - Assembly Language
  - High Level Languages
- Architecture Families

What did you learn in COMP 1701/1631?

- problem solving strategies
  - introductory algorithm design and data representation
  - structured programming in a high level language
  - etc.
- All important, but none require detailed knowledge of how a computer actually works!

What did you learn in COMP 1701/1631?

- Question: How does a computer work?
- Question in this course: How does a computer work from the programmer's perspective?
- To answer, we must study computer architecture and low level (assembly language) programming.

What did you learn in COMP 1701/1631?

- A **high level** programming language requires little/no knowledge of computer architecture.
- A **low level** language requires detailed knowledge of architecture. The programmer must know how the machine actually works.
- **Computer organization** refers to hardware design.
  Specifically, to the design of functional units and their interaction
  - The functional units include: CPU, memory, I/O interfaces, etc.
  - How are the designed and how are they connected are the concerns of computer engineers, not us the programmers

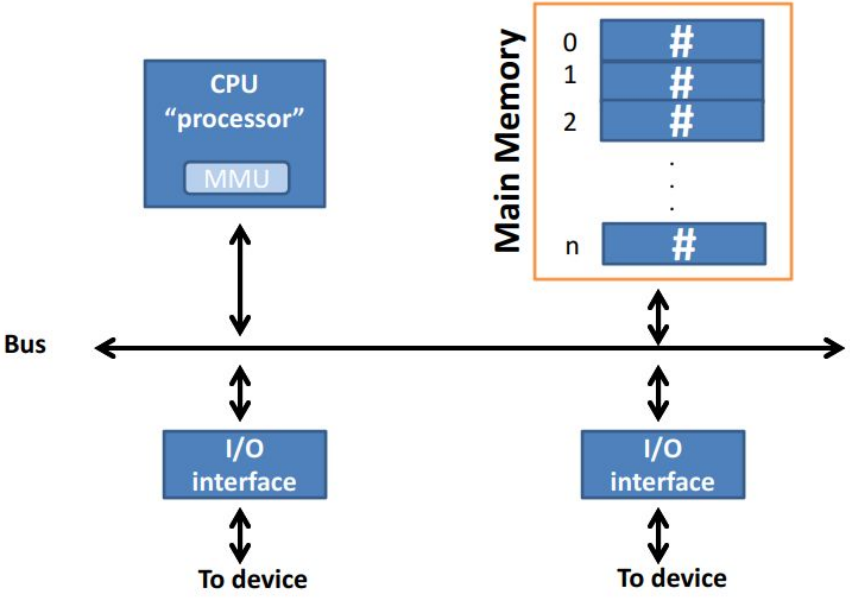
What did you learn in COMP 1701/1631?

- The hardware provides a set of resources (or “context”) for the low level programmer.
- **Computer architecture** is:
  - the set of hardware-provided resources that the low level programmer can/must interact with directly.
  - the apparent or virtual set of resources provided by the hardware available to the low-level (assembly language) programmer

What is a computer?

What is a computer (Adv)?

Computer Architecture



Typical Resources

- Instruction set: finite set of operations supported by CPU
- Registers: high-speed memory locations on-board CPU
- Address Space: addressable range of slower, main memory locations
- Addressing Modes: finite set of ways instructions can access data
- What about accessing I/O devices?
    - NOT relevant to this course

What do Computer System do?

Computer systems
store/process numbers,
and only numbers.

Von Neumann Architecture

Characteristics of the von Neumann architecture:

- It is also called a stored program architecture
- Numbers are represented internally as fixed-length binary numbers
- Each instruction is identified by a unique number
- Data is represented as one or more numbers
    - Data is stored either in a register or in main memory

Von Neumann Architecture

Characteristics of the von Neumann architecture:

- Memory contains both data and code
- The processor controls the execution of the current instruction and instructions are sequentially executed
    - The process of executing is called **The Fetch-Execute Cycle**
- There are architectural differences between modern computer makes and models, but these general characteristics are constant

Levels of Programming Language

Low Level		Low Level		High Level
Machine Language	←	Assembly Language	←	High Level Language

Machine Language

- An **executable program** is a sequence of instructions (and data), loadable into memory. Once loaded, the CPU can execute the sequence one instruction at a time (possibly with "branching"). This means a program is really just a sequence of numbers. This is **machine language**.
- CPU's native language
- A **loader** is a program which loads other programs into memory (e.g. from disk), and which (usually) tells the CPU to start executing that program.
- Initially, programmers coded directly in machine language!

Machine Language

```
0000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0000020 01 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00
0000040 68 00 00 00 00 00 00 00 34 00 00 00 00 00 28 00
0000060 05 00 04 00 55 89 e5 83 ec 08 83 e4 f0 b8 0a 00
0000100 00 00 48 85 c0 7f fb 31 c0 c9 c3 00 00 2e 73 68
0000120 73 74 72 74 61 62 00 2e 74 65 78 74 00 2e 64 61
0000140 74 61 00 2e 62 73 73 00 00 00 00 00 00 00 00 00
0000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000220 0b 00 00 00 01 00 00 00 06 00 00 00 00 00 00 00
0000240 34 00 00 00 17 00 00 00 00 00 00 00 00 00 00 00
0000260 04 00 00 00 00 00 00 00 11 00 00 00 01 00 00 00
0000300 03 00 00 00 00 00 00 00 4c 00 00 00 00 00 00 00
0000320 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
0000340 17 00 00 00 08 00 00 00 03 00 00 00 00 00 00 00
0000360 4c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000400 04 00 00 00 00 00 00 00 01 00 00 00 03 00 00 00
0000420 00 00 00 00 00 00 00 00 4c 00 00 00 1c 00 00 00
0000440 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
0000460
```

Pros & Cons?

Pros:

- good for processor
- simple, fast, efficient code

Cons:

- hard for humans to understand
- difficult and slow code
- error prone

Assembly Language

- PCs only execute machine language programs
- Programming numerically, in binary, is:
  - difficult
  - slow
  - error-prone
- For each numeric machine instruction
  - assign it a symbolic name
  - referred to as a “mnemonic”
  - one-to-one correspondence
- Data values and memory addresses can also be given symbolic names

Assembly Language

- Typically provides other programmer conveniences
  - e.g. macros
- Differs between CPUs
  - most general principles are constant
  - generally the same for all CPUs in a chipset (i.e. Intel i9)
- An **assembler** is a program which translates assembly language source code into machine code.

Assembly Language

```
main:
     pushl     %ebp
     movel     %esp, %ebp
     subl $8, %esp
     andl $-16, %esp
     movl $10, %eax
.L5:
     decl %eax
     testl     %eax, %eax
     jg   .L5
     xorl %eax, %eax
     leave
     ret
```

Assembly Process

source code 1 → assembler → object module 1

source code 2 → assembler → object module 2

source code 3 → assembler → object module 3

…

source code n → assembler → object module n

object modules 1 … n → linker → executable program

Pros & cons?

Pros:

- allows programmers to work in a symbolic fashion

Cons:

- each assembly language is designed for a specific processor, or processor family
    - each program only runs on one type of computer.

Machine Code vs. Assembly Language

Is machine language more powerful than assembly language?

NO – they are the same thing!

One to one correspondence between MC instruction and AL mnemonic

High Level Languages

- Assembly language still force the programmer to “think” the same way the computer works
    - “Simple” procedures may require long instruction sequences
    - Coding in a more natural, abstract way is highly desirable
- High level languages satisfy this desire
- Translation from HLL to ML is required
    - Accomplished by a program called a compiler.
- Can be done with/without invoking an assembler
    - C/C++ automatically assembles by default

High level language

```
// TO PRODUCE count.s: g++ -S count.cpp
// TO PRODUCE mach:     as count.s -o count.o


int main()
{
  int i;

  i = 10;

  while (i > 0)
    i--;

  return 0;
}
```

Pros & Cons

Pros:

- allows programmers to work at a more abstract level
- more complex operations allowed, i.e. complete equations

Cons:

- the translation process is more involved, requires more programs
- the translation is not one-to-one
- the translation is not optimized

Why learn assembly language?

- necessary to understand how software/computers work
- necessary to understand how compilers & HLLs work
- helps in writing/debugging HL code
- assembly language is still used:
  - system programming (e.g. operating systems, device drivers, etc.)
- optimization (e.g. inline assembly)
- solving crashes, security issues, etc.

Architecture Families

- A **microprocessor** is a CPU on a single chip.
- The first microprocessor was the Intel 4004 in 1970: a 4-bit processor for a calculator.
- Microprocessors are developed generationally. Newer versions add features and improvements, but remain backward-compatible with previous versions.
- E.g. Motorola 680x0 family (68000, 68010, 68020, 68030, 68040, etc.)
- E.g. Intel x86 family (8086, 80286, 80386, 80486, 80586, …)

Architecture Families

Why backward-compatible?

- Able to run existing software on new hardware!
- Electrical/Computer Engineers can create a brand new processor, from scratch, in under 12 months!
- The software to control the processor (OS) can take years to write, ex. Look at how long it takes Microsoft to produce a new OS!

Practical Realities of the Course

- See how first year concepts are implementation at the low level
- Specifically:
  - Data Representations (ints, chars, bools, reals)
  - Simple statement (math, etc.)
  - Selection/Repetition
  - Data structures (arrays, strings, structures)
  - Functions (calls, parameter passing, value returning, local variables)
  - Recursion
  - Pointers
  - Dynamic memory allocation and linked structures

Next Lecture

Integer Representations

Text sections 2.1-2.4