Computer Science CSCI 261 Fall 2020

Computer Architecture and Assembly Language

Dr. Peter Walsh
Department of Computer Science
Vancouver Island University
peter.walsh@viu.ca

Course Overview

○ Objectives:

- master the principles of computer architecture
- introduce computer organization concepts
- introduce time-oriented programming
- work with assembly language and C
- explore concepts in real-time and embedded systems

○ Prerequisite: Min. "C" in CSCI 161

○ No face-to-face instruction

○ Course Outline and Information Web Pages:

- http://csci.viu.ca/~pwalsh/teaching/261/261/Info-Sheet.html
- http://csci.viu.ca/~pwalsh/teaching/261/261/261.html

Hardware/Software Resources

○ Student IT Requirements:

- high-speed Internet connection
- computer with audio and video capabilities

○ Laboratory (Physics Room 115):

- lab contains 17 cub machines running Linux
- there is no physical access to Room 115
- access the cubs using ssh and/or PuTTY
- simulators replace microcontroller boards

○ Key Internet Applications:

- VIUOnline (Zoom)
- VIUTube (Video Portal)
- VIULearn (Assessment)

Course Delivery

○ Lectures:

- lecture videos posted to VIUTube

○ Labs:

- pre-lab videos posted to VIUTube
- scheduled lab time is reserved for Q and A and student evaluation using Zoom
- no labs the first week of term

Course Delivery cont.

○ Office Hours:

- reserved for answering email questions

○ Quizzes:

- administered through VIULearn
- dates TBD

○ Lab Tasks:

- see course page for task specification
- Zoom for on-line evaluation
- git for off-line evaluation

Student Attendance for CSCI 261

○ On-Campus

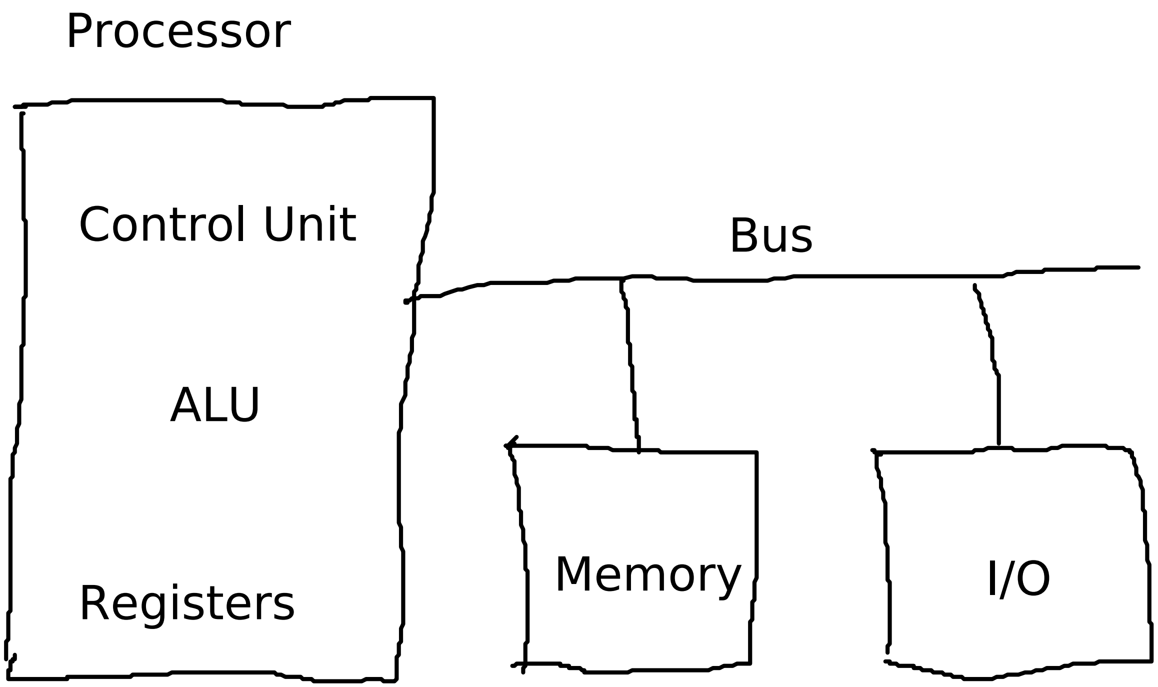
- you are NOT required to be on-campus

○ Off-Campus

- you are expected to attend your scheduled labs by Zoom
- you must submit your lab task solutions by Zoom or git prior to assigned deadlines
- you must complete quizzes through VIULearn prior to assigned deadlines
- my goal is to answer all email questions during my office hours
- you may view all other course work-products at your leisure

Computer System Architecture

- Typical Computer Science Abstraction



Processor

Control Unit

ALU

Registers

Bus

Memory

I/O

Computer Components

○ Control Unit (CU)

- fetch, decode and monitor the execution of instructions

○ Arithmetic and Logic Unit (ALU)

- perform data transformations under CU control

○ Registers

- storage locations in the processor

○ Processor

- CU, ALU and registers

Computer Components cont.

○ Microprocessor (Up)

- a processor on a single integrated circuit (IC or chip)

○ Microcontroller (Uc)

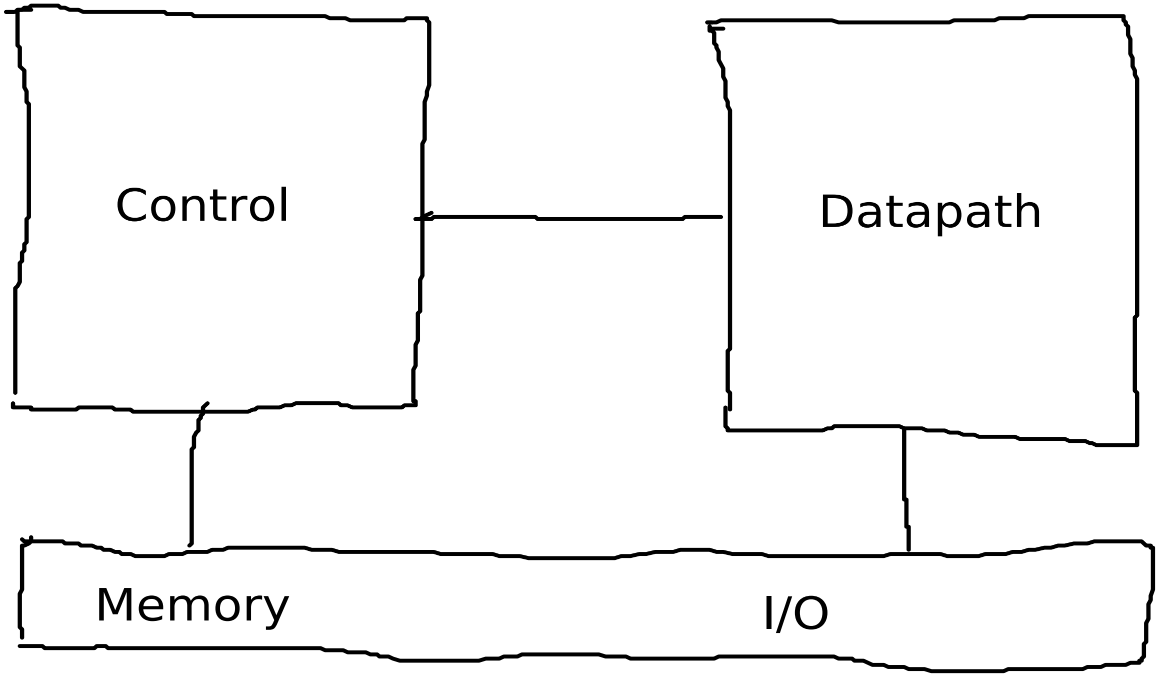
- a microprocessor with memory and I/O on a single chip

○ Single Board Computer

- computer on a single printed circuit board (PCB)

Computer System Architecture cont.

○ Typical Computer Engineering Abstraction



Computer Components cont.

○ Data-path

- processor components that perform data transformation

○ Control

- processor components that command and control the data-path, memory and I/O subsystems

Instructions

○ High level language (HLL)

- e.g. `x = 5 + 1`

○ Assembly language

- e.g. `mov R0, 5`

○ Machine language

- e.g. 1011000000000101

○ In general:

- 1 HLL instruction is translated into many assembly language instructions
- 1 assembly language instruction is translated into 1 machine language instruction

Instruction Translation Examples

```
x = 5 + 1 ---> mov R0, 5   # R0 <- 5
               add R0, 1   # R0 <- R0 + 1
               save R0, x  # x <- R0
```

```
mov R0, 5 ---> 10110000 00000101
```

Instruction Execution

○ Stored Program Concept

- machine language is stored in the computer along with relevant data
- the computer can manipulate a program in the same way it can manipulate data

○ Fetch and Execute Cycle

- one by one, machine instructions are fetched from memory and executed until the machine is halted

Tools (used in CSCI 160)

○ Compiler

- translates source code to object code
- e.g. `foo.c` to `foo.o`

○ Linker

- translates object code to machine code
- e.g. `foo.o` to `foo`

○ Loader

- loads the machine code into memory in preparation for execution

Additional Tools

○ Compiler (GNU compiler with -S switch)

- translates source code to assembly language code
- e.g. `foo.c` to `foo.s`

○ Assembler

- translates assembly language code to object code
- e.g. `foo.s` to `foo.o`

Why learn assembly language?

○ Efficiency

- programmers are unlikely to out-perform a modern compiler but, on occasion, we may need to violate compiler conventions

○ Resource Access

○ Foundation Knowledge

- CSCI 360 Operating Systems
- CSCI 355 Digital Design
- CSCI xxx Compiler Construction
- CSCI 460 Networks
- CSCI 461 Embedded and Real Time Systems