Lecture 1

Introduction

In the beginning...

CS 241: Foundations of Sequential Programs
Spring 2017
About the course

- www.student.cs.uwaterloo.ca/~cs241
  - read the Syllabus (policies, due dates, outline, etc.)
  - read the Announcements
  - read everything else on the main webpage

- Assignments
  - START EARLY!
  - Don’t fall behind!!!
  - 10 assignments in total, each with 9 subparts: about 100 things to submit
Public tests (aka “sanity tests”)

Release tests
  - Release tokens
    - 3 for each “part” of each assignment
    - once one is used, it regenerates after 12 hours
Your program must run correctly on the linux.student.cs environment.
Marking

- Assignments: 25%
- Midterm: 25% written on Wednesday, June 21, 7:00-8:20pm
- Final Exam: 50% written sometime in August
- Note: you must pass the weighted exam average to pass the course
Human Resources

- Instructors:
  - Dan Holtby (djholtby@uwaterloo.ca)
  - Kevin Lanctot (kevin.lanctot@uwaterloo.ca)

- ISA: Pierre-Louis Guidez. Ye Hua (Edward) Tan (part time) (cs241@uwaterloo.ca)

- Instructional Support Coordinator: Gang Lu (glu@uwaterloo.ca)

- IAs/TAs: run tutorials
Non-human Resources

- Textbooks: Nothing official.

- Course Notes: Nope.

- Discussion Forum: Piazza
  - be nice
  - no spam
  - no “me too”
  - no “thanks”
  - read first, search second, post last
Purpose of the course

- To learn (how unusual).
- Meta-thinking.
- A compiler is a program that reads a program and writes another program.
What’s in a name?

Foundations of Sequential Programs

✦ What really happens when I compile and run a program?
✦ By the end of the course, there should be very little mystery left about computers or computer programs.
What is: A computer?

- CPU - Central Processing Unit
  - See diagram on board (Over to you, Future Dan)
  - Control Unit
  - Arithmetic / Logic Unit
  - Memory Unit
  - Registers
  - Magic Smoke

- RAM - Random Access Memory
  - Just a bunch of bytes
  - Each byte has an address

Hungry for more? CS251
What is: A compiler?

Future Dan, draw the CS241 diagram.
What is: Machine language?

- von Neumann Architecture
  - Programs are stored in the same memory as data
- Machine specific language
- Binary
  - Easy for a computer to read
  - Terrible for a human to read
What is: Binary data? (Oops, I mean “What are”)

What is a bit?

- “Binary digit”. Zero or one.
  - Easy to represent. Each bit is ON or OFF, LOW or HIGH, etc.
  - Usually grouped together. Common groupings:
    - A byte is 8 bits
    - A nibble is 4 bits
    - A word is...ambiguous and architecture dependent
Integers

What does 0110 1010 mean?
▶ It could be an 8-bit integer, $64 + 32 + 8 + 2 = 106$

What does 1111 0001 mean?
Negative Integers

How can we represent negative integers?

- **Signed Magnitude**: Leftmost bit represents sign
  - 1111 0001 == ???
  - Addition is difficult

- **One’s Complement**: Flip da bits
  - 1111 0001 == ???
  - Addition is almost the same as unsigned addition, but “almost” isn’t good enough

- **Two’s Complement**: Flip da bits (then add 1 for good measure)
  - 1111 0001 == ???
  - Addition is identical to the unsigned version
Many different ways to encode characters as binary values
We’ll use ASCII, 0110 1010 == 'j'
There are also a few Unicode flavours:
  - UTF-32 - 1 32-bit unit per code point (they call them that instead of characters)
  - UTF-16 - 1 to 2 16-bit units per code point
    - UTF-16BE - Big Endian
    - UTF-16LE - Little Endian
  - UTF-8 - 1 to 4 8-bit units per code point. All ASCII characters are the same in UTF-8.
Hexadecimal

- Binary sucks (if you’re a human being)
- Mostly because it’s too long
- If we want to know a particular bit in a decimal number, we’ll need to convert the whole thing
- Hexadecimal (base 16) is short, and you only need to convert one digit
- \( 241 = 1111 \ 0001 = \text{0xF1} \)
Assembly language

- Human readable language
- Easy to translate to machine language
- Still machine specific
Assembler

- Convert Assembly language (text) to Machine language (binary)
In this course we use a 32-bit MIPS CPU. Not a real one, mind you.

- 32 “general purpose” registers
- several special registers
- very small set of instructions
  - and we don’t use all of them
- Reference sheet on the course webpage
The Language of MIPS

- $32 = 2^5$ registers, need 5 bits to write down a register ID
- Instructions are 32-bit (4 bytes)
- Three register parameters need 15 bits, leaving 17 to specify the instruction.
How does the CPU know which words are instructions?

- The special “PC” (program counter) register
- Pointer to next instruction
- After each* instruction $PC += 4$
One C statement: \( x = y + z; \)

What is this telling the CPU to do?