Course Overview & Introduction

CSE 220: System Fundamentals I

Joydeep Mitra

Instructor Information

- Prof. Joydeep Mitra
  - Personal Webpage (https://www.cs.stonybrook.edu/people/faculty/JoydeepMitra)
- Office Hours:
  - New Computer Science Bldg., Room 131.
  - Mo and We 1.00 PM – 2.30 PM.
  - By appointment.
- Email: jmitra@cs.stonybrook.edu

Course Overview

- CSE 220 introduces the fundamental ideas of computer organization and architecture.
  - Processor architecture, memory system, etc.
- What will you learn?
  - Number representations
  - Digital circuit design
  - Techniques to design efficient digital architectures
  - Fundamentals of assembly language programming
  - An understanding of how software instructs hardware

Why Take CSE 220?

- Comprehend the architectural differences between different kinds of computing devices.
- Understand software properties that result from software interacting with hardware:
  - Performance.
  - Security.
  - Correctness.

Logistics

- Course Website (https://sites.google.com/stonybrook.edu/cse220/home)
  - Login with your NetID creds; clear cache if necessary.
  - Provides course overview.
- All course material will be available on Blackboard
  - Course schedule, Lecture notes, videos, practice problems.
- Course-related discussion will happen on Piazza
  - Email jmitra@cs.stonybrook.edu if you haven't received an invite
  - You will see banner to contribute to Piazza; you are **not** required to pay
- TAs will lead recitations and hold office hours; check course website for timings and location.

Textbook and Software

- Required: *Harris and Harris, Digital Design and Computer Architecture, Second Edition, Morgan Kaufmann, 2013. (ISBN 978-0-12-394424-5)*.
- We will use MARS – a simulator for writing and executing MIPS assembly:
  - You will need this for assignments!
  - Download tailored version from Blackboard, **Course Documents->MARS** (NOT from the web).
- *Homework programs should be submitted via GitHub and Git.*
  - Create an account on GitHub.
  - Install *Git*; available by default in most Linux and Mac. On Windows install Git Bash.
  - Instructions on how to submit will be discussed later before the first assignment.
  - GitHub walkthrough videos will be available on Blackboard.

Grading Scheme

- Programming Assignments: 35% (5%, 6%, 8%, 8%, 8%)
- Midterm Exam: 20%
- Comprehensive (cumulative) final exam: 30%
- Quizzes: 10%
- Recitation attendance: 5%
- Course Grade Cutoffs: A [93-100], A- [90-93), B+ [87-90), B [83-87), B- [80-83), C+ [77-80), C [73-77), C- [70-73), D+ [67-70), D [63-67), F [0-63)
    - Subject to adjustment

Homework Assignments

- There will be five programming assignments on assembly programming.
- Low-level assembly programming is cumbersome!
- Start early to avoid last-minute struggles.
- Your programs will be evaluated against test cases; make sure to test your programs thoroughly.
- There will be no extensions to deadlines unless illness or truly unavoidable circumstances.

Academic Integrity

- You may discuss general concepts with classmates (e.g., help with tooling or clarifying concepts)
- Do NOT share source code or files that amount to copying
- Do NOT directly copy code from the internet
- Your work will be checked for plagiarism
- Cheating will lead to academic dishonesty charges before University’s Academic Judiciary

Quizzes

- Held *online via Blackboard* once in every 10-14 days.
- The quiz must be completed within a 24-hour time window in one sitting.
- Practice questions (on the course website) are a good source to study for the quizzes and exams in general.

Exams

- Midterm Exam: (tentatively) *Wednesday, Oct 13 (8:30 AM - 9:50 AM).*
- Final Exam: *Wednesday, Dec 15 (8:00 AM - 10:30 AM).*
- Exams will be closed-book and closed-notes.
- Electronic devices, textbooks, notes will not be permitted. Will amount to cheating if caught!
- There will be no make-up exams!
- Exams will be in-person as of now. Might be online if COVID regulations change.

Recitation

- Attendance at recitation is expected; counts for credit.
- You may miss one recitation without penalty.
- *You must attend your own recitation section for credit.*
- There will be no makeups for missed recitations.
- TAs will do exercises and help you with assignments in recitations.
- Recitations are a good way to meet fellow students and discuss with them.
- *Do not attend lecture or recitation if you test positive for COVID*. Contact the instructor with the test report.
- You won't be penalized if you miss recitation due to COVID. But you must show relevant documentation.

Disability

- If you have any kind of disability, please contact contact the Student Accessibility Support Center at room 128 in the Educational Communications Center.
- Their phone number is 631-632-6748.
- Disability accommodations will *only* be made for students officially registered with the Student Accessibility Support Center (SASC).
- Inform me at least 1 week in advance if you are planning to take an exam at the SASC.
- All documentations of disability are confidential.

How to Succeed in CSE 220

- Practice good time management.
- Visit office hours whenever in doubt.
- Feel free to ask questions and interact during class.
- Discuss freely on Piazza.
- Attend recitation and actively engage with your TAs.
- Do the practice problems.

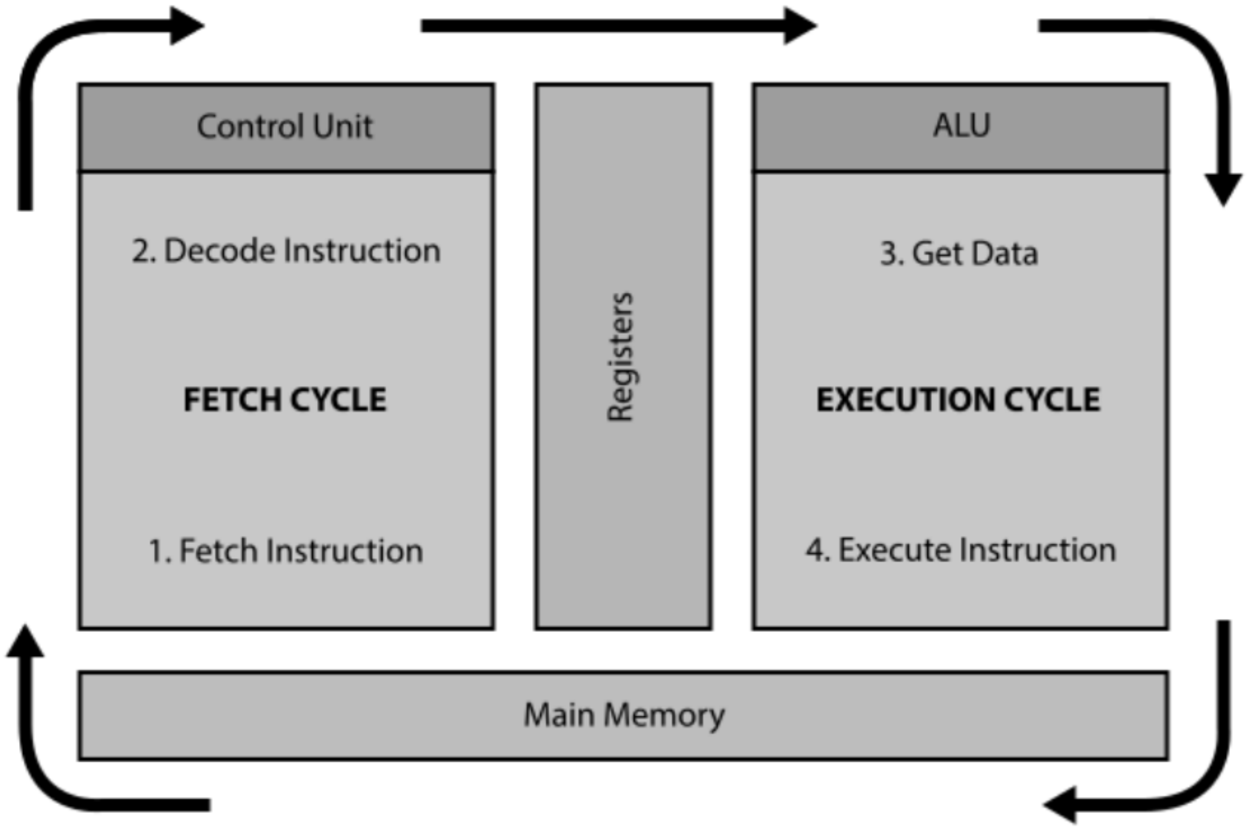
Architecture vs. Organization

- *Computer Architecture* describes the programmer's view of a computer system (what the system does)
    - e.g., the instruction set, data representation mechanism, memory addressing technique.
- *Computer organization* (or *microarchitecture*) refers to the physical components of a computer and how they interact to realize an architecture
    - e.g., control signals, interfaces between the computer and peripherals, and memory technology.
    - An architecture may be implemented by more than one microarchitecture.

Von Neumann Architecture

- Modern computers follow the Von Neumann architecture, but are substantially more complex
- Instructions and data are stored in **main memory**.
- A separate **central processing unit (CPU)** to process instructions and data.
  - *Control unit* performs instruction fetching and decoding.
  - *Arithmetic Logic Unit (ALU)* performs arithmetic and logical operations.
  - *Registers* hold small number of instructions and data.
- **Fetch-Decode-Control Cycle***: Data is transferred from main memory to CPU, CPU processes it and stores the results back to main memory.*

Fetch-Decode-Control Cycle

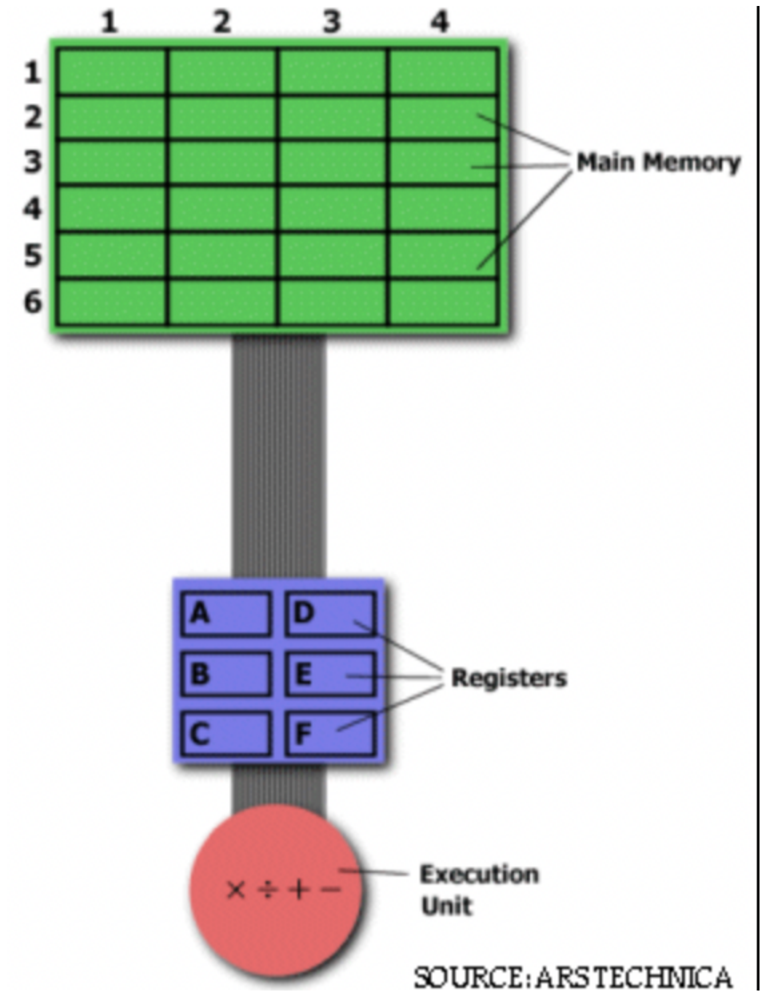


Instruction Set

- A collection of fundamental commands that the CPU “understands”.
  - Hard-wired into the computer’s design.
- The kind of instruction set depends on the CPU
  - **RISC** (**R**educed **I**nstruction **S**et **C**omputer) CPUs, the instruction set is small with simple instructions
  - **CISC** (**C**omplex **I**nstruction **S**et Computer) CPUs, the instruction set is larger with instructions that vary in length and complexity

RISC vs CISC Machines

- The difference is best understood by comparing RISC with CISC using an instruction to multiply two numbers. Assume:
  - A main memory with 6 rows and 4 columns.
  - An execution unit responsible for computation.
  - The execution unit an only operate on data in the 6 registers (A,B,C,D,E,F).



RISC vs CISC Machines

- In a CISC CPU, the computation will be performed by 1 instruction
  `MULT 2:3, 5:2`
- The instruction is baked into the processor and does the following:
  - loads from memory[2,3] and memory[5,2] into registers A and B.
  - Computes A * B
  - Stores the product in in memory[2,3].
- Key benefits:
  - Complex computation can be performed with 1 instruction.
  - The compiler has to work less to translate to assembly.
  - Since code has fewer instructions, less memory is needed to store it.

RISC vs CISC Machines

- Since RISC CPUs use instructions that run in 1 clock cycle, MULT command needs to be encoded with multiple instructions:

```
LOAD A, MEM[2:3]          # read from memory[2:3] and store in register A
LOAD B, MEM[5:2]          # read from memory[2:3] and store in register B
PROD A, B                 # multiply A and B
STORE MEM[2:3], A         # store product in memory[2:3]
```

- Key benefits of the RISC strategy:
  - Since each instruction takes 1 clock cycle, the MULT command takes the same time as the CISC machine.
  - The RISC instructions are simpler hence take less transistor space in the processor.
  - The space can be used for more general-purpose registers.
  - Pipelining is possible since each instruction takes 1 clock cycle.

RISC vs CISC Machines

- To improve performance, CISC machines aim to minimize no. of instructions in a program, sacrificing no. of cycles per instruction.

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \underset{\Uparrow}{\frac{\text{instructions}}{\text{program}}}$$

- To improve performance, RISC machines aim to minimize no. of cycles per instruction, at the cost of no. of instructions in a program.

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \underset{\Uparrow}{\frac{\text{cycles}}{\text{instruction}}} \times \frac{\text{instructions}}{\text{program}}$$

RISC vs CISC Machines

- Simple instructions, fewer in number
- Fixed length instructions
- Complexity in compiler
- Only load/store instructions access memory
- Three operands per instruction
- Single-cycle instructions
- Highly pipelined

- Many complex instructions
- Variable-length instructions
- Complexity in hardware
- Many instructions can access memory
- One or two register operands per instruction
- Multiple-cycle instructions
- Less pipelined

- CISC machines are commercially more successful for historical reasons (e.g., Intel's x86).
- Despite its advantages, RISC machines have taken time to develop due to lack of software support.
- **Recommended reading: The Case for the Reduced Instruction Set Computer, Patterson & Ditzel.**

Typical RISC Instructions

- We will focus on the **MIPS** (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages) architecture – a RISC type
  - **Load** data from memory to CPU register
  - Copy (**Store**) data from CPU register to memory
  - **Add, Subtract, Multiply, Divide**, etc. data in CPU registers
  - **AND, OR, XOR, NOT**, etc. data in CPU registers
  - **Shift** and **Rotate** data in CPU registers
  - **Jump** codes to execute specific instructions
  - **Call** a subroutine (function) and **return** to caller