CENG 3420
Computer Organization & Design

Lecture 01: Introduction

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Textbook: Chapters 1.3 & 1.4)

Spring 2023

Course Administration

Instructor:

- Bei Yu (byu@cse.cuhk.edu.hk)
- Office: SHB 907
- Office Hrs: H14:30–16:30

Tutors:

- Chen Bai (cbai@cse.cuhk.edu.hk)
- Shixin Chen (sxchen22@cse.cuhk.edu.hk)
- Hongduo Liu (hdliu21@cse.cuhk.edu.hk)
- Ziyi Wang (ziyiwang21@cse.cuhk.edu.hk)

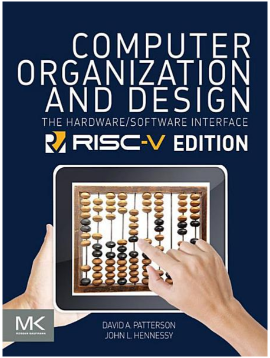
Grading Information

Grade Determinates

- 5% Attendance
- 15% Homework
- 15% Midterm (Mar. 18)
- 25% Three Labs (Individual project)
- 40% Final Exam

- Late submission **per day** is subject to 10% of penalty.
- A student must gain at least 50% of the full marks in order to pass the course.
- A student must attend at least 80% of lectures in order to gain all class attendance credits.

General References



Textbook:

- *Computer Organization and Design*, RISC-V Edition
- Soft copy, `amazon.cn`, or `amazon.com`

Manuals:

- LC-3 Instruction Set Architecture (ISA)
- Lab tutorials (slides)

Slides:

- On the course web page before lecture
- Summary may be uploaded afterwards

Course Content

- Introduction to the major components of a computer system, how they function together in executing a program.
- Introduction to CPU datapath and control unit design
- Introduction to techniques to improve performance and energy-efficiency of computer systems
- Introduction to multiprocessor architecture

Course Content

- Introduction to the major components of a computer system, how they function together in executing a program.
- Introduction to CPU datapath and control unit design
- Introduction to techniques to improve performance and energy-efficiency of computer systems
- Introduction to multiprocessor architecture

Philosophy

To learn what determines the capabilities and performance of computer systems and to understand the interactions between the computer's architecture and its software so that future software designers (compiler writers, operating system designers, database programmers, application programmers, ...) can achieve the best cost-performance trade-offs and so that future architects understand the effects of their design choices on software.

Why Learn This Stuff?

- You want to call yourself a "computer scientist/engineer"
- You want to build HW/SW people use (so need performance/power)
- You need to make a purchasing decision or offer "expert" advice

Both hardware and software affect performance/power

- Algorithm determines number of source-level statements
- Language/compiler/architecture determine the number of machine-level instructions
- Processor/memory determine how fast and how power-hungry machine-level instructions are executed

What You Should Already Know

- Basic logic design & machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, run, debug programs in an assembly language
  - Will be introduced in tutorial
- Create, compile, and run C/C++ programs
- Create, organize, and edit files and run programs on Unix/Linux

- This course is all about how computers work
- But what do we mean by a computer?
  - Different types: embedded, laptop, desktop, server
  - Different uses: automobiles, graphics, finance, genomics ...
  - Different manufacturers: Intel, Apple, IBM, Sony, Oracle ...
  - Different underlying technologies and different costs
- Analogy: Consider a course on "automotive vehicles"
  - Many similarities from vehicle to vehicle (e.g., wheels)
  - Huge differences from vehicle to vehicle (e.g., gas vs. electric)
- **Best way to learn**:
  - Focus on a specific instance and learn how it works
  - While learning general principles and historical perspectives

What is a Computer?

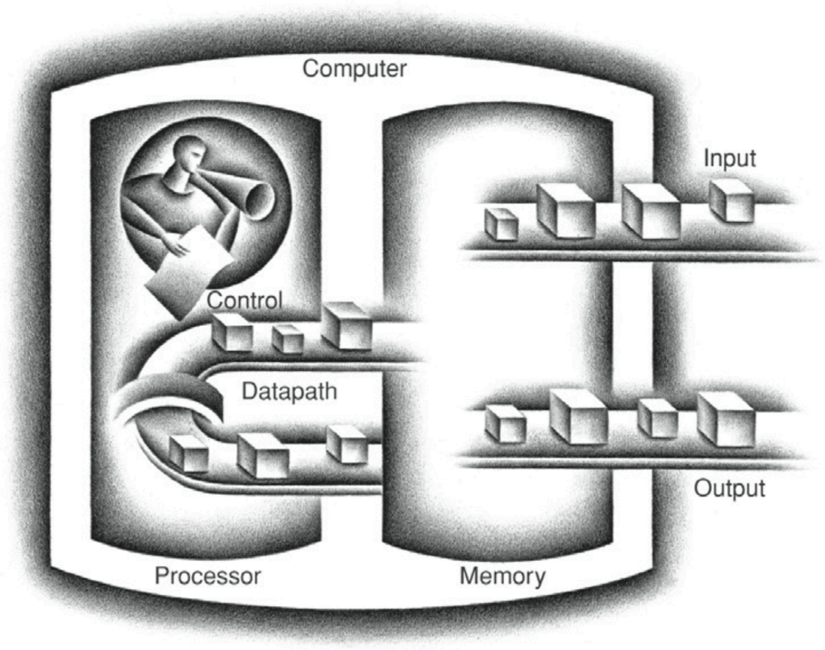
Components

- processor (datapath, control)
- input (mouse, keyboard)
- output (display, printer)
- memory (cache, main memory, disk drive, CD/DVD)
- network

Our primary focus: the processor (datapath and control) and its interaction with memory systems

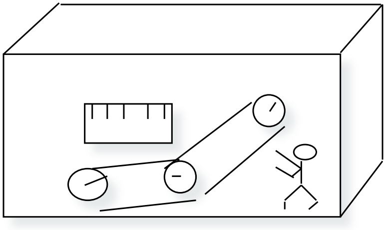
- Implemented using tens/hundreds of millions of transistors
- Impossible to understand by looking at each transistor
- We need abstraction!

Major Components of a Computer



Machine Organization

- Capabilities and performance characteristics of the principal Functional Units (FUs). (e.g., register file, ALU, multiplexors, memories, ...)
- The ways those FUs are interconnected (e.g., buses)
- Logic and means by which information flow between FUs is controlled
- The machine's Instruction Set Architecture (ISA)
- Register Transfer Level (RTL) machine description

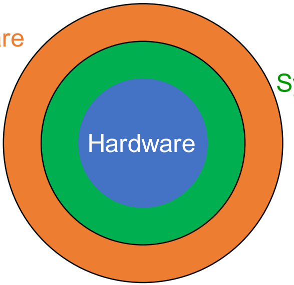


Control needs to have circuitry to

- Decide which is the next instruction and input it from memory
- Decode the instruction
- Issue signals that control the way information flows between datapath components
- Control what operations the datapath's functional units perform

Datapath needs to have circuitry to

- Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
- Interconnect the functional units so that the instructions can be executed as required
- Load data from and store data to memory



Operating System

- Supervising program that interfaces the user's program with the hardware (e.g., Linux, iOS, Windows)
- Handles basic input and output operations
- Allocates storage and memory
- Provides for protected sharing among multiple applications

Compiler

- Translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute

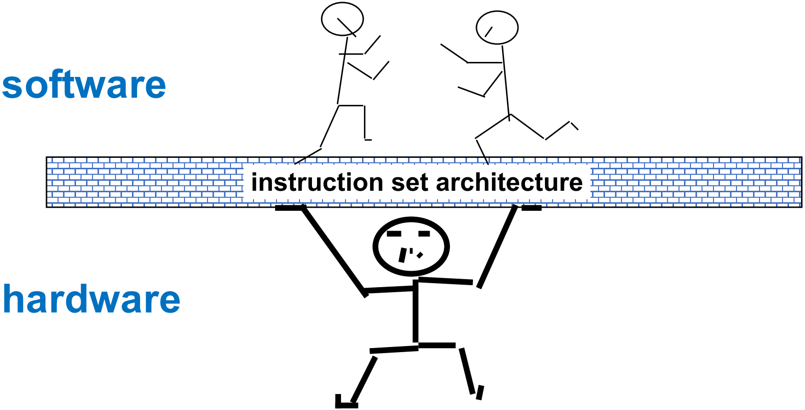
Advantages of Higher-Level Languages ?

- Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, ...)
- Improve programmer productivity – more understandable code that is easier to debug and validate
- Improve program maintainability
- Allow programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
- Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine

As a result, very little programming is done today at the assembler level

Instruction Set Architecture (ISA)

The interface description separating the software and hardware



Instruction Set Architecture (ISA)

- ISA, or simply architecture – the abstract interface between the hardware and the lowest level software that includes all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, ...
- Enables implementations of varying cost and performance to run identical software
- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI)
- ABI: The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

Instruction Categories

- Load and Store instructions
- Bitwise instructions
- Arithmetic instructions
- Control transfer instructions
- Pseudo instructions

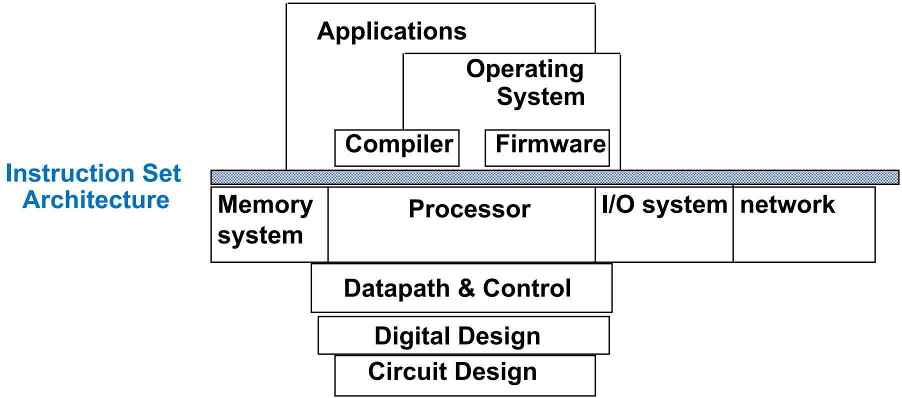
4 Base Instruction Formats: all 32 bits wide

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]				rd	opcode	U-type

Table: Register names and descriptions

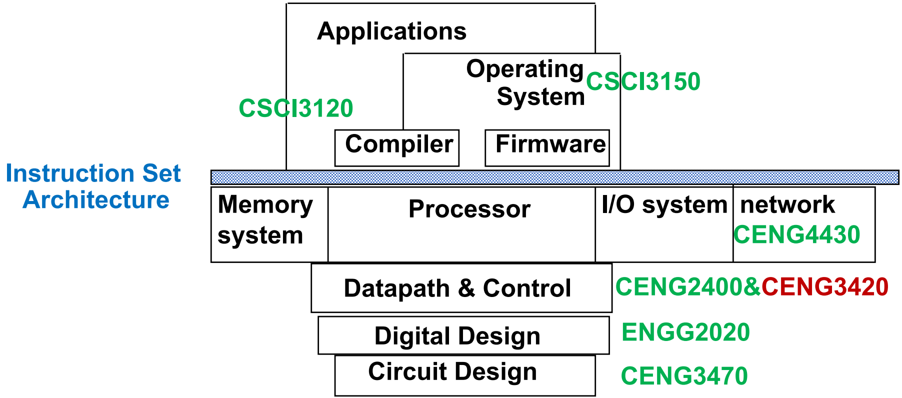
Register Names	ABI Names	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary / Alternate link register
x6-7	t1 - t2	Temporary register
x8	s0 / fp	Saved register / Frame pointer
x9	s1	Saved register
x10-11	a0-a1	Function argument / Return value registers
x12-17	a2-a7	Function argument registers
x18-27	s2-s11	Saved registers
x28-31	t3-t6	Temporary registers

How Do the Pieces Fit Together?



- Coordination of many levels of abstraction
- Under a rapidly changing set of forces
- Design, measurement, and evaluation

How Do the Pieces Fit Together?



- Coordination of many levels of abstraction
- Under a rapidly changing set of forces
- Design, measurement, and evaluation