CS 2461
Computer Architecture 1
i.e., Introduction to Computer Systems

https://cs2461-2020.github.io/
Fall 2020
Instructor: Dr. Bhagi Narahari

1

What is CS 2461 about?!

- Look 'under the hood' to see how a computer works
  - Explore the interface between hardware and software
  - Understand the components in a processor
  - Bottom up approach: from transistors on up to algorithm design
    - i.e., the hardware stack
- With this knowledge you can
  - Understand the link between hardware and software
  - Write better, more efficient software
  - Design better hardware
    - Link between hardware and software
  - Appreciate the abstractions that are built on top of these foundations

2

2

Pre-requisites

- Pre-requisites
  - CS 1112 – Data structures and algorithms
  - CS 1311 – Discrete Math 1
- Co-requisite: CS 2113
  - Knowledge of C programming language
  - I will be synchronizing with instructor
- Programming practice…system skills
  - Practice, practice,…and more practice

3

3

Course Objectives

- To understand the structure and operation of a modern computer system from the ground up.
  - Understand basic hardware concepts and build simple circuits
    - digital circuits -gates, bits, bytes, number representation
  - Understand the Von Neumann architecture/computing model
    - structure and operation, (assembly language)
  - Basic "system" concepts
    - runtime stack, simple I/O devices, Unix OS
    - Software security Introduction – stack smashing attacks
- How high level languages are implemented on the machine (using the C language)
  - How are C programs translated to assembly and implemented on a machine
  - Proficiency in the C programming language
- Understand how software/program performance is linked to program and machine properties

4

4

Two recurring themes in Computer Sci.

- Abstraction: Productivity Enhancer
  - You don't need to worry about the details
    - You can drive a car without knowing about the internal combustion engine….until something goes wrong: where is that smoke coming from !!
  - The notion that we can concentrate on one "level" of the big picture at a time, with confidence that we can then connect with other levels.
  - Framing abstraction appropriately is a very important skill
    - THIS is the whole point of mathematical modeling in Engineering/CS fields
- Hardware and Software
  - abstraction does *not* mean being clueless about HW or SW
  - In particular, hardware and software are inseparably connected, especially at the level we will be studying
    - Even if you specialize in one, you must understand the capabilities of the other

5

5

What are Computers meant to do ?

- Solve problems that are described in English (or Greek or French or Hindi or Chinese or ...) and use a box filled with electrons and magnetism to accomplish the task.*
  - This is accomplished using a system of well defined (sometimes) transformations that have been developed over the last 50+ years.
  - As a whole the process is complex, examined individually the steps are simple and straightforward
  - *Definition from the textbook*
- *So how do you get the electrons to run around and do our task ?*

6

6

Two Big Ideas in Computing

- Universal Computational Devices
  - Turing's Thesis: every computation can be performed by some "Turing Machine" - a theoretical universal computational device
    - You will see this in the Foundations course CS 3313
- Problem Transformation (Abstraction!)
  - The ultimate objective is to transform a problem expressed in natural language into electrons running around a circuit (using a succession of transformations)
    - That's what Computer Science and Computer Engineering are all about: a continuum that embraces software & hardware.
    - Note the role of compilers/translators

7

7

Big Idea #1: Universal Computing Device

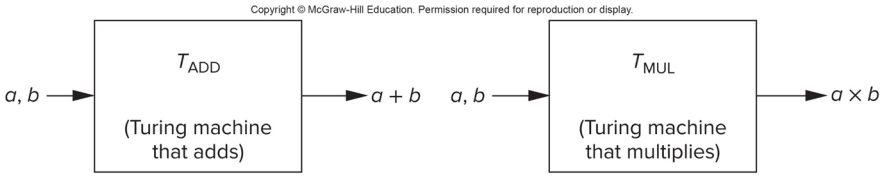
- All computers, given enough time and memory, are capable of computing exactly the same things
  - Smartphone, laptop, supercomputer
    - Limited only by time and memory (and energy)
- Anything that can be computed, can be computed by a computer
  - If you can describe something in terms of computation, it can be done by a computer
- Formal (mathematical) model of computing = Turing Machine

8

8

Turing Machine

- Mathematical model of a device that can perform any computation – Alan Turing (1937)
  - Ability to read/write symbols on an infinite 'tape'
  - State transitions: based on current state and input symbol
- Every computation can be performed by some Turing machine (*Turing's thesis*)

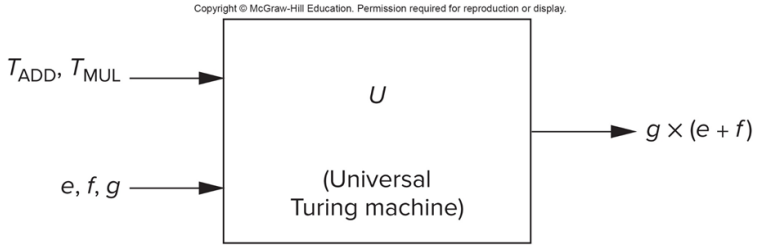


9

9

Universal Turing Machines

- A machine that can implement all Turing machines
  - This is also a Turing machine !
  - Inputs: data, plus a description of computation (other TMs)
- U is programmable – so is a computer !
  - Instructions are part of the input data
  - A computer can emulate a universal Turing machine



10

10

Turing machines, Computers,….so which one do we use ?

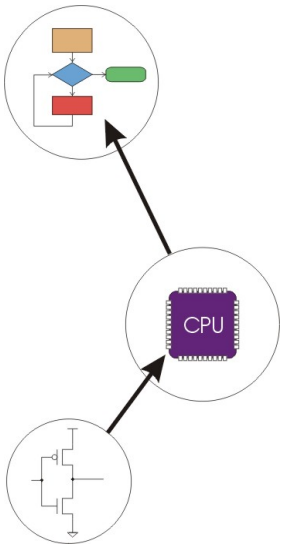
- If all of them can do your work which one do you choose and what is the difference ?

- Performance
  - Speed/Latency (how long to solve your single program)
  - Throughput (how many tasks at a time can the server handle)
- Cost
- Energy/Power
  - Particularly important in smartphones, embedded systems,..
    - Because energy source is a battery

11

11

Big Idea #2: Transformation between layers (Abstraction!): Putting the electrons to work!



- **Problems**
- **Algorithms**
- **Program**
- **Instruction Set Architecture**
- **Microarchitecture**
- **Circuits**
- **Devices**

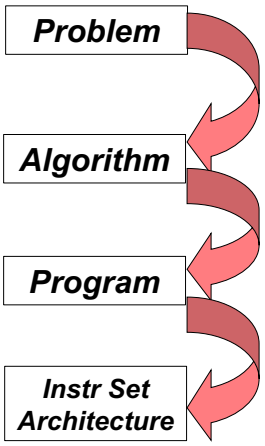
12

12

How do we solve a problem using a computer?

A systematic sequence of transformations between abstraction layers.

Example: Problem = Sort a set of numbers

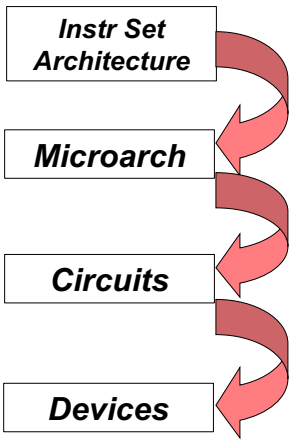


Software Design: choose algorithms and data structures. Bubble Sort

Programming: use language to express design and implement algorithm. C lang.

Compiling/Interpreting: convert language to machine instructions. Use gcc compiler (to compile to ARM ISA)

…and even more layers…

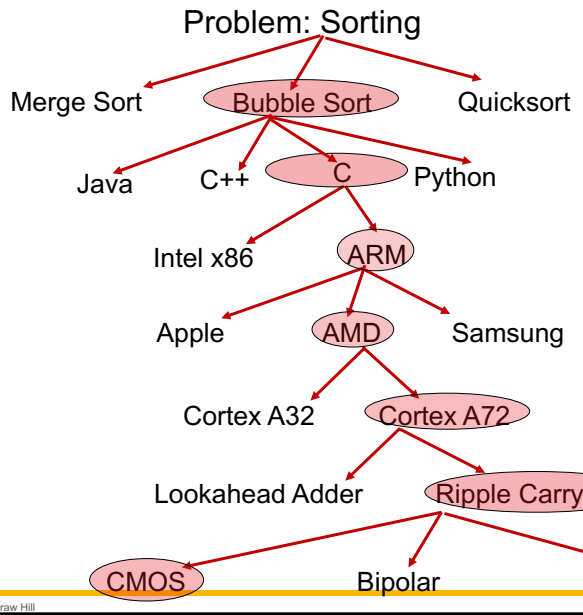


Processor Design: choose structures to implement ISA. Choose ARM ISA

Logic/Circuit Design: gates and circuits to implement components of AMD ARM processor

Process Engineering & Fabrication: develop and manufacture transistors, wires, etc.

Why do we need 'experts' at each layer – Choices at each layer



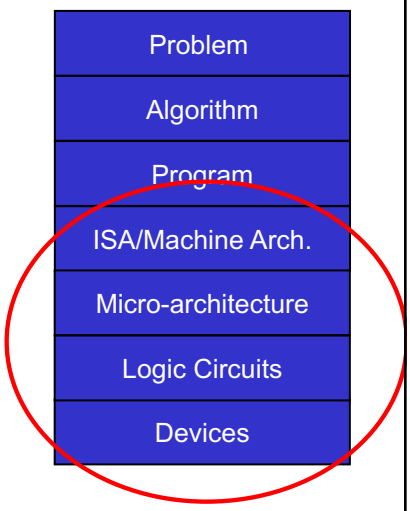
Tradeoffs: Cost, Performance, Power, …

15

15

Recap: Our Computing Technology Stack & transformation between layers

- The problem..in words
- The algorithm..still in words but formalized
- The program…captures algorithm in specific HLL syntax (C, Java,..)
- The ISA…an instruction set architecture that specifies what operations processor can perform
- The micro-architecture…potentially many ways to implement processor with given ISA
- The logic circuits…implement various components of microarchitecture as assemblies of logic gates
- The devices…logic gates built from transistors fabricated using technologies (CMOS, GaAs,..)



16

16

Focus of this course: The Machine/Hardware Level

This is going to be 'all new' material for most of you…

- Machine Architecture
  - This is the formal specification of all the functions a particular machine can carry out, known as the *Instruction Set Architecture* (ISA).
    - We will study the ISA, and Assembly Language programming of a simple computer LC3 – why select a simple "unrealistic" computer?
- Microarchitecture
  - The implementation of the ISA in a specific CPU - i.e. the way in which the specifications of the ISA are actually carried out.
    - We give an overview of the microarchitecture; CS 3462 covers this topic

17

17

The Machine Level –contd.

- Logic Circuits
  - Each functional component of the microarchitecture is built up of circuits that make "decisions" based on simple rules
    - We will study the basic building blocks of logic circuits
    - You will learn to implement hardware logic circuits in the labs
- Devices
  - Finally, each logic circuit is actually built of electronic devices such as CMOS or NMOS or GaAs (etc.) transistors.
    - Device electronics – not in this course

18

18

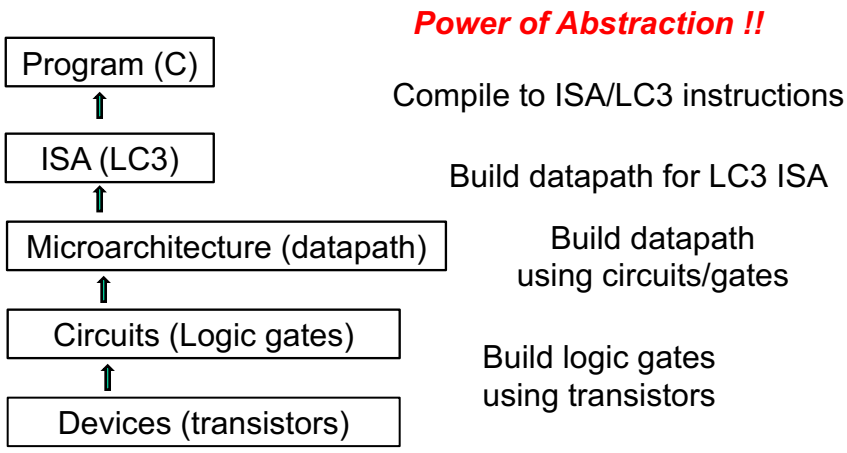
Approach

- Bottom-up: from bits to C
  - Establish link between hardware and software
  - Learn C in context of hardware
    - Used for systems programming & misused by many!
    - Learn what actually happens when your programs run
    - If you understand the hardware the language loses a lot of its mystery!
- Why C: the ultimate high-level, low-level language
  - High level enough that you can write large scale programs in It
  - Low level enough that you can reach down and twiddle the bits.

19

19

Bottom up Approach….Power of Abstraction



Our starting point is "how to represent data"

20

20

How does this course "interface" with other courses ?

- CS2113 Software Engineering: learn C, learn software development, testing, debugging,…
- CS 3410 Systems Programming:
  - HW: use a CPU board and sensors to deploy a system to solve a problem (parking assist in a car)
  - SW: systems programming (in C) to process data gathered from sensors
- CS 3411 Operating systems:
  - How does software 'manage' the HW resources
    - CS2461 barely touches the tip of the iceberg when it comes to systems concepts!
  - A lot more programming in C
- CS 3313 Foundations:
  - Mathematical foundations of computing devices

21

21

Your Initial To-do List

1. Get your "reading resources" - textbook
2. Go to the course web page.
   - Lecture videos and notes will be posted
3. Prepare for class:
   - Read notes/slides, textbook and watch any videos that are posted for the next class/topic.
     - There are going to be in-class exercises during class!!

4. Submit the "Feedback Questions" survey by 1pm before each class

22

22