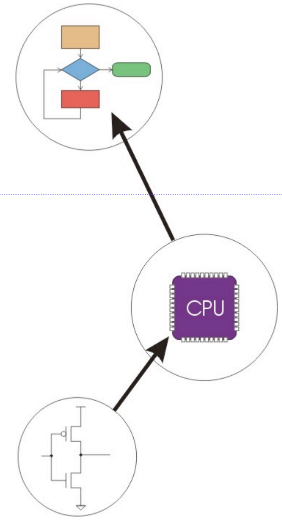
Chapter 1
Welcome Aboard

Computer System: Layers of Abstraction



Software

Application Program

Algorithms

Language

Hardware

Instruction Set Architecture
(and I/O Interfaces)

Microarchitecture

Circuits

Devices

Recurring Themes

Abstraction

- Allows us to manage seemingly insurmountable complexity
- One billion components require abstraction

Hardware v. Software

- Separation of hardware and software is artificial

Recurring Theme #1: Abstraction

Abstraction hides details

Examples
- "Turn off the light!"
- "Stick out your tongue."

Computing examples
- Java methods
- C functions
- LC-3 instructions
- Logical gates
- Transistors

Bottom line
- Often best to operate at highest level of abstraction
- Dangerous to completely ignore lower levels of abstraction

Recurring Theme #2: Hardware v. Software

Artificial distinction

Greatness arises from blurring the HW/SW line

- **CISC vs RISC (complex vs "reduced" instruction set computing)**
- **MMX/SSE**
- **Intel's Itanium - EPIC (explicitly parallel instruction computing)**
- **Research proposals: RAW/Trips. . .**

Bottom line

- **We really care about *computation***
- **Hardware best understood by those who know software**
- **Software best understood by those who know hardware**

Very Big Ideas

Universality

- **All computers can compute the same thing***

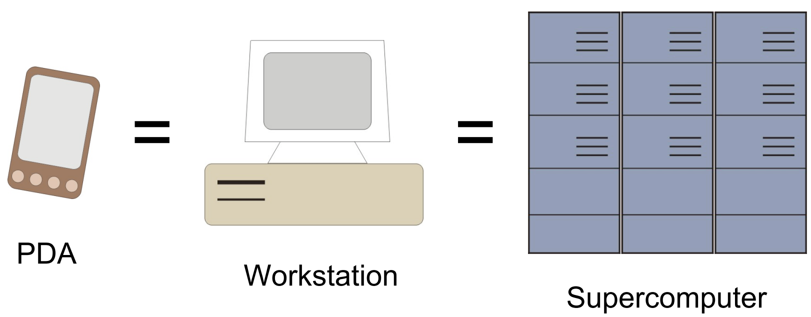
Layered Abstraction

- **We can build very complex systems from simple components**

Big Idea #1: Universal Computing Device

All computers *can* computing exactly the same things*

***given enough time and memory**

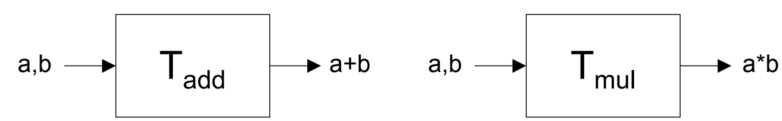


Turing Machine

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 symbol**

Every computation can be performed by some Turing machine. ***(Turing's thesis)***

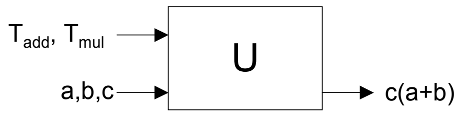


Turing machine that adds *Turing machine that multiplies*

Universal Turing Machine

Turing described a Turing machine that could implement all other Turing machines

- Inputs: data, plus a description of computation (Turing machine)



Universal Turing Machine

U is programmable – so is a computer!

- Instructions are part of the input data
- A computer can emulate a Universal Turing Machine, and vice versa

Therefore, a computer is a universal computing device!

From Theory to Practice

In theory

- Computers can compute anything that's possible to compute
- Given enough *memory* and *time*

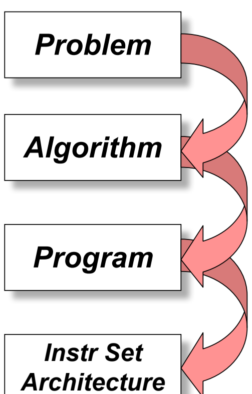
In practice

- Solving *real* problems requires computing under constraints
  - "engineering" constraints
- Time
  - Weather forecast, next frame of animation, ...
- Cost
  - Cell phone, automotive engine controller, ...
- Power
  - Cell phone, handheld video game, ...

Big Idea #2: Layered Abstraction

How do we solve a problem using a computer?

- Systematic sequence of transformations between layers of abstraction. . .

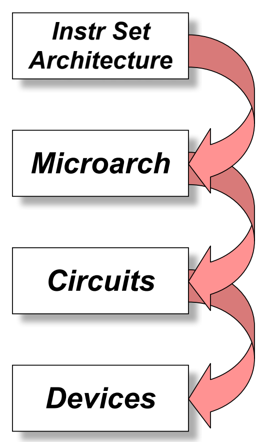


Software Design:
choose algorithms and data structures

Programming:
use language to express design

Compiling/Interpreting:
convert language to machine instructions

Deeper and Deeper…



Processor Design:
choose structures to implement ISA

Logic/Circuit Design:
gates and low-level circuits to implement structures

Process Engineering & Fabrication:
develop and manufacture lowest-level components (transistors)

Descriptions of Each Level

Problem Statement

- **Stated using "natural language"**
- **May be ambiguous, imprecise**

Algorithm

- **Step-by-step procedure, guaranteed to finish**
- **Definiteness, effective computability, finiteness**

Program

- **Express the algorithm using a computer language**
- **High-level language, low-level language**

Instruction Set Architecture (ISA)

- **Specifies the set of instructions the computer can perform**
- **Data types, addressing mode**

Descriptions of Each Level (cont.)

Microarchitecture

- **Detailed organization of a processor implementation**
- **Different implementations of a single ISA**

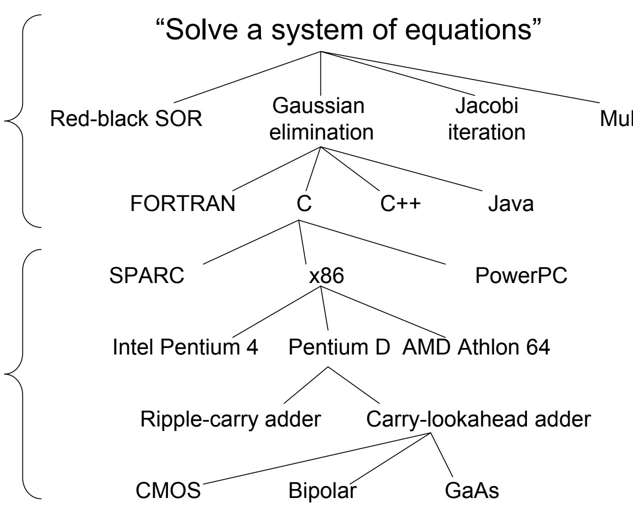
Logic Circuits

- **Combine basic operations to realize microarchitecture**
- **Many different ways to implement a single function (*e.g.,* addition)**

Devices

- **Properties of materials, manufacturability**

Many Choices at Each Level



Tradeoffs:
cost
performance
power
(*etc.*)

Course Outline

Bits and Bytes

- **How do we represent information using electrical signals?**

Digital Logic

- **How do we build circuits to process information?**

Processor and Instruction Set

- **How do we build a processor out of logic elements?**
- **What operations (instructions) will we implement?**

Assembly Language Programming

- **How do we use processor instructions to implement algorithms?**
- **How do we write modular, reusable code? (subroutines)**

I/O, Traps, and Interrupts

- **How does processor communicate with outside world?**

C Programming

- **How do we write programs in C?**
- **How do we implement high-level programming constructs?**

Next Time

Lecture

- **Chapter 2: Bits and Bytes**

Reading

- **Chapter 2 - 2.5**

Quiz

- **Don’t forget!**

Upcoming

- **Homework 1 due on Friday, September 15**