Introduction to Computer Engineering

CS/ECE 252, Fall 2015
Computer Sciences Department
University of Wisconsin – Madison

Karu Sankaralingam
Two Big Components

• What is a computer and how to build a computer?

• How to program a computer?
What is a Computer?

- **Arithmetic**: Basic arithmetic and logical operations on numbers
- **Storage**: Storing the results of these operations
- **Branching**: Choosing which operations to perform next, based on the result of another operation
- **Input/Output** (often written "I/O"): Receiving signals from external sources (input) and sending signals to the outside world (output)
How to Program them?

• Fixed-program computer
  – Do only one thing: computer linear equations, play movie, modify camera photos, etc.

• Stored-program computer
  – Focus of this course
Abstraction

description of a system by defining its interface and ignoring implementation
User’s address and payment information
ISA: Instruction Set Architecture

• It specifies the basic objects, or "primitives" that the computer contains--often including "registers" (small memories) and a larger data "random-access" memory--collectively known as the "architecture".

• It specifies the instruction set--that is, the set of things that can be done to these primitives (e.g., "add the numbers in these two registers and store the result in this third register" might be a valid instruction in the instruction sets of some types of computer).

• It specifies a technique for storing instructions as physical objects. Computers are, after all, machines. They don't deal in abstract commands, but in moving around physical objects (electrons, photons, pineapples, whatever).
Example

ldi r30, 2
ldi r31, 7
add r30, r31

PC = PC + 1
Programming Language

• Because the ISA deals in very basic primitives, programmers often use a programming language, which allows them to write more human-friendly code which will then be translated into instructions from the actual ISA by a compiler.

• A programming language has less rigid rules for what valid code can be than the ISA, but there are still rules--called the language's syntax. The meaning of various valid constructions using the programming language--what those lines of code actually do--is also a part of the programming language's definition--called the language's semantics.
Example

\[ x = 2 \]
\[ y = 7 \]
\[ y = x + y \]
Algorithm

• Step-by-step sequence to solve a problem.
• Every step in it must be precise and unambiguous, to the point that it could be rephrased as a combination of the basic types of operations that computers perform. By having this property we can ensure that a computer can execute every step in an algorithm.
• It must terminate. That is, we will only consider algorithms that have a definite ending after finishing some number of steps.
• Complexity (time, space it takes to solve problem)
Libraries

Another use of abstraction – how to use other people’s work basically 😊
gmaps.directions(origin, destination)
Microarchitecture

• To actually build the computer, then, we will create black-box functional units that perform each of these basic operations--storing a number, adding two numbers, etc.

• The arrangement of these basic functional units is called the computer's microarchitecture.

• State machines control "when" one of these functional units gets to do something

• Many many ways to implement one ISA!
State Machine concept
Computers as State Machines

1. Get next instruction.
2. Add the two given numbers.
3. Add
4. Store the result in the specified location.
Computers as State Machines

- **GET NEXT INSTRUCTION**
  - **ADD THE TWO GIVEN NUMBERS**
  - **STORE THE RESULT IN THE SPECIFIED LOCATION**
  - **JUMP AHEAD THE SPECIFIED NUMBER OF INSTRUCTIONS**

- **add**
- **jump**
PC = PC + 1
Implementing these circles and boxes is done with logic gates and transistors.
binary

Computers are machines. But we said that computers operate on numbers, whereas physical machines can only manipulate physical things! So we have a scheme for representing numbers by something physical. This scheme is called "binary."
Logic gates and boolean logic

• not, and, or, xor, nand, nor

• These operate one on one binary thing – called a “bit”

• 8 bits is called a “byte”

• We can string together these gates and perform any computation! Wow!!!!!!!
Transistor: the final level!

Control

Input

Output

0 voltage or HIGH Voltage
Transistor: the final level!

- not gate:
  - if input = 0, output = 1
  - if input = 1, output = 0

Truth table:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Turing Completeness!
Two Big Components

• What is computer and how to build a computer?

• How to program a computer?

• Main idea: Abstraction
Next Class & Announcements

• Read Chapter 1 *BEFORE* class
• Come prepared with questions
• HW-1 assigned; due Sep 11\textsuperscript{th}. 
1) What is a computer? How do you build one?
2) How do you program a computer?

Computer must do all four things:
Arithmetic: basic math and logical operations on numbers
Storage: save the results of these operations somewhere in memory (caches, RAM, etc)
Branching: Decision making; choosing which operations to perform next
Input/Output: interact with outside world; get signals from external sources (input) and send signals to outside (output)

Two types of computers:
1) Fixed program computers can only do one "thing" (equations, movies, whatever)
2) Stored-program computers: programmable computers, can take programs, put them in memory, fetch and perform operations

Focus: how do we BUILD this stored program?

ABSTRACTION:
Description of a system by defining its interface and ignoring implementation
- how do two different parts of a system talk to each other?
- don't describe how individual parts work

Why is abstraction useful?
- Allows you to be organized, forces you to think about individual pieces in the system before building a single component
  - You can innovate and change components without having to revise the whole system
  - You can troubleshoot individual components to make sure they adhere to interfaces

Wafflecat! \o/

How do computers use abstraction?

System
------
Programming language
------
ISA (Instruction Set Architecture)
------
COMPUTER
------
Microarchitecture
------
Logic Gates
------
Transistors
Languages are made up of words, but these can be decomposed into a subset of "instructions". The computer implements these instructions into the microarchitecture.

Logic Gate examples (NOT, AND, etc)

Transistors: use silicon to build these gates

ISA: Basic objects, or "primitives" of a computer. Specifies what a computer CAN do. Often uses "registers" (very small, fast memory) and individual instructions (add, store, etc). Technique for storing instructions into memory as well as talking to input/output devices

Example:
ldi r30,2 # ld: load, i: immediate. Load an immediate value (2) into register r30
ldi r31,7 # load 7 into r31
add r30,r31

VERY LOW LEVEL!

What to do if you have more variables than registers?? STORE THEM IN MEMORY! Move them into registers as necessary with a LOAD instruction

Because this is so low-level, this actually allows us to build a machine. THINK: Why would you want to program in ISA?

Programming language: a more human-friendly level. A compiler is used to translate this into ISA instructions. The SYNTAX (rules of the language) are less rigid. Semantics explain what the syntax operations actually do.

Example:
x=2
y=7
y = x + y

Compiler will convert this into ISA. Compiler is just another program! Primitive compiler is written in ISA/Assembly, and a more sophisticated one is built on top of that.

Algorithm: step-by-step sequence to solve a problem
1) Must be precise and unambiguous!
2) It must terminate w/ a definite ending after x amount o steps

Libraries: another use of abstraction. How do you use other people's code. Ex: gmaps in Python. You have to adhere to their interfaces.

Microarchitecture: the thing that actually implements the machine. An organization of building blocks which sequence a certain way and execute any program written in the ISA. "State Machines" control when these units does something.
What has intel been changing? The MICROARCHITECTURE! Even if you tweak HOW the implementation is done, the ISA result will be the same. Improvements to microarchitecture: efficiency, improve performance, make them consume less power, etc.

State machine: CAT EXAMPLE

Computers as state machines.

What CAN'T the state machine in the example do? The example is NOT a programmable computer. It's a "fixed" computer that can only add numbers.

Implementing the circles/boxes:

BINARY: a representation that computers use via 0's and 1's

Gates: not, and, or, xor, nand, nor
These operates on "bit" (one binary thing). 8 bits = byte. Stringing these together with bytes allow us to do ALL SORTS OF THINGS!!

Building blocks -> microarchitecture -> ISA -> programming language

Transistor: the final level!
Control: output/input

TURING COMPLETENESS: building a computer that we can run ANY program on.

Next time: wrap up w/ case study of search engine interface.