Use pencil!

Engineering 100 (section 250) Microprocessors and Toys
An Introduction to Computing Systems

Mark Brehob—Electrical Engineering and Computer Science
Erik Hildinger—Technical Communications

Overview

Welcome to Engineering 100, Microprocessors and Toys!

Today's class will be broken into a few different parts:

- Introduction to the staff, the class and a general overview of what we'll be doing.
- Discussion about engineering and abstraction
- Introduction to digital logic

For more technical aspects of the class, there will typically be a handout of some sort and the lecture will primarily be on the board.

Introduction to Engineering

What's an engineer? Definition, characteristics, job description etc.

- How does an engineer differ from a tradesperson? Electrical engineer vs. electrician? Mechanical engineer vs. car mechanic?

- How does an engineer differ from a scientist?

One definition:

> Scientific and mathematical principles
> Applies...to practical ends
> Design, manufacture, test and operate (In other words *different phases of a system's lifetime.*)

Different types of engineers

What are some types?

What makes them different?

What do you suppose computer engineering is?

Class overview

Goal: experience the life cycle of a substantial computer engineering project

Microprocessor-based educational toy

- microprocessor: general-purpose, programmable computer
- device drivers for I/O devices
- educational toy (runs as application on your microprocessor).
- User interacts with toy via input/output devices.
- Toy responds according to custom algorithm, computed on your microprocessor.
- Examples:
    - music synthesizers, interactive stuffed animals, video communicators, rhythm/music games, talking globes, interactive audio books, ...

Project done on development board: Altera DE2 FPGA (field-programmable gate array)

- SRAM, SDRAM
- LEDs
- switches and buttons
- PS/2 keyboard
- USB mouse
- audio interface (speaker, microphone)
- VGA interface
- LCD display
- secure digital card
- serial port
- video camera
- also infrared, Ethernet, Flash RAM

Course will introduce you to basics of many parts of compute engineering

- digital logic
- implementing algorithms in hardware
- implementing a computer in hardware
- programming a computer in assembly language
- interfacing to I/O devices
- digital audio
- operating systems

Workload

- 4 credits ==> 12 hours/week
- lecture (3 hours)
- lab (2 hours)
- discussion (1 hour)

- group meeting (1 hour)
- CSE work outside lab (2-3 hours)
- TC work (2-3 hours)

Levels of abstraction

- applications and operating system (high-level language, e.g. C++)
- applications and operating system (machine-level language)

---------- hardware/software boundary----------

- computer processor
- datapath and control
- combinational and sequential logic
- logic gates
- transistors

Introduction to Digital logic

Say we live in the rather black and white world where things (variables) are either true (**T**) or false (**F**). So if ***S*** is "Mark is going to the Store" and ***C*** is "Mark likes Computer games" then we'll assume that each phrase is either true or false (as opposed only sort of liking computer games). We can then use connectives to combine the variables.

Mark is going to the store AND Mark likes computer games.

The above statement is only true if both phrases are true. Let that sentence be ***X***. We can now draw the "truth table" for ***X*** (we'll use the other tables in a minute). When is X true?

S	C	X
F	F	
F	T	
T	F	
T	T	

AND

S	C	
F	F	
F	T	
T	F	
T	T	

_____

S	C	
F	F	
F	T	
T	F	
T	T	

_____

C	
F	
T	

_____

What if the statement, ***Y***, were "Mark is going to the store OR Mark likes computer games"? When is that true? How about Mark does NOT like computer games?

(OR vs. XOR)

How about having ***B*** be "Bob's house is brown".

S	C	B	
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S AND C AND B

S	C	B	
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S OR C OR B

S	C	B	
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S OR C OR NOT B

S	C	B	
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S OR (C AND B)

Representation of Boolean Logic (section 2.6)

Using AND, OR, NOT and XOR gets old. So symbols have been used to represent these notions.

	Electrical/Computer Engineering	Gate
Y AND Z		
Y OR Z		
NOT Y		
Y XOR Z		

In digital logic we usually think of "1" as being TRUE and "0" as being "FALSE"…

And now for something completely different… (maybe)
Binary (and Hex) numbers (1.2)

Consider the number 123

1 2 3

Each *place* has a value. We normally work in base 10, so each place is 10 times bigger than the last.

In binary we work in base 2. Consider the number 10010_2 (the subscript indicates the base).

1 0 0 1 0

Now what do you suppose the value of 10.11_2 is?

1 0 . 1 1

Let's convert 21 into base 2.

(Time allowing we'll cover this in class. You need to know it in any case.)

Now consider base 16. We need symbols for 0-15 but only have them for 0-9 in decimal. So we'll use A=10, B=11, C=12, D=13, E=14, F=15.

What is $1F_{16}$ in decimal? What is 44 in hexadecimal (base 16)?

Converting between base 2 and base 16 is easy. Just group the binary digits into groups of 4 starting at the decimal point[1]. So what is 10011001_2 in hex? We commonly use hex to represent large numbers which we'd prefer to use binary for just to make it more readable…

And the payoff…

Why did we do binary numbers on the first day? How are they relevant to digital logic?

The point is that we can represent binary numbers using basic logic. Consider a device that adds two one-digit binary numbers and outputs a 2 digit binary number. Let the inputs be A and B and the output be R[1:0].

> (The R[1:0] is a way two write that there are two outputs, R1 and R0. We might also write R[7:0] to indicate that there are 8 outputs: R7, R6, R5, R4, R3, R2, R1, and R0.)

$$\begin{array}{r} A \\ +\ B \\ \hline\hline R_1R_0 \end{array}$$

Write the truth table for this adder. R1 is to be the most significant digit (farthest to the left in the 2's place in this example) while R0 is to be the least significant digit (farthest to the right, in the 1's place). Then draw the logic gates.

A	B	R_1	R_0
0	0		
0	1		
1	0		
1	1		

The point is that we can do arithmetic using basic logic. You may say "great, I can add two one-bit numbers". But it turns out we can use this basic idea of using logic states to represent numbers to do all kinds of math. A modern computer can easily do 5-10 billion additions of 64-bit numbers in a second! All based on this basic idea.

Consider adding two 3-digit binary numbers.

- If we used the truth table scheme, how many rows would there be?
  - What about for a 64-bit addition?
- Can we break the problem down in a way that works better?
  - How?
- Let's work on it…