CMPE012 – Computer Engineering 12 (and Lab)

Computing Systems and Assembly Language Programming

Spring

Winter 2016
About: Me

- Undergraduate at UCSC in Computer Engineering and Electrical Engineering.
- Masters from UCSC
- Worked on a mountain lion tracking collar in Graduate School
- Renovated and built new hardware for CMPE118 based off the Ubo32
The Course

- Introduction to computer systems and assembly language and how computers compute in hardware and software.
- Topics include digital logic, number systems, data structures, compiling/assembly process, basics of system software, and computer architecture.
- May include a very basic introduction to C.
- Prerequisite(s): course 3 or 8, or Computer Science 10 or 12A or 5C or 5J, or 5P, or Biomolecular Engineering 60, or suitable programming experience; previous or concurrent enrollment in course 12L required.
The Team

- Instructor: [CMPE 12]
  - Max Dunne (mdunne@soe.ucsc.edu)
  - Office hours: MW 1-3PM

- Teaching Assistants
  - Rodriguez, Ryan <ryarodri@ucsc.edu>
  - Yep, Danny <dyeap@ucsc.edu>
  - Hematti, Ehsan <ehsan@soe.ucsc.edu>
  - Dmitriy Rivkin <drivkin@ucsc.edu>
  - Pritesh
W < I
Our online presence...

- **Ecommons** [http://ecommons.ucsc.edu](http://ecommons.ucsc.edu)
  - Assignments - Getting and submitting
  - Calendar

- **Online forum (Piazza)**
  - Main forum for communication

- **Auxiliary Website**
  - Slides and Videos
Exam Dates

- Midterm
  - Tentatively Tuesday May 10th In class
- Final
  - Monday, June 6th, 8:00–11:00 a.m.

- Note: The final can not be taken at a different time, please plan accordingly
The textbook

Patt, Patel

*Introduction to Computing Systems: From Bits and Gates to C and Beyond*

McGraw-Hill, Second Ed.
What we will cover in this class

Part 1: Introduction to Logic Design
- Logic Gates and Functions (Ch 3)
- Integer Number Representation (Ch 2)
- Computing Systems & Abstraction (Ch 1)
What we will cover in this class

Part 2: The LC-3 computer system
- LC-3 Architecture (Ch 4)
- LC-3 Instruction Set Architecture (Ch 5)
- LC-3 Machine Language Programming (Ch 6)
- LC-3 Assembly Language Programming (Ch 7)
- LC-3 Input and Output (Ch 8)
- LC-3 TRAPS and subroutines (Ch 9)
- LC-3 Stack (Ch 10)
What we will cover in this class

Part 3: The PIC32 (Arduino) microcontroller (not in book)

- Microcontrollers and embedded systems
- PIC32 microcontroller
- MIPS assembly
- PIC32 I/O and interrupts
- MPLAB IDE (integrated Development Env.)
- PicKit3 Debugger
What we will cover in this class

Part 4: Fractional numbers
- Fixed and floating-point numbers and arithmetic
  - IEEE 754 Floating-Point
- Floating Point Arithmetic
  - Addition
  - Subtraction
  - Multiplication
  - Not division
What we will NOT cover in this class

- C
- Hardware design (CMPE 100/110)
- Extensive C coding (CMPE 13)
- Software engineering (many)
- Algorithms (CS 101)

This class is intended to be a bottom to top overview of computer systems. Other classes will cover material in greater details.
Extended course description

Required skills to pass the course.

1. Number representations, including
   a. arbitrary base conversion
   b. binary, hex, decimal, 2’s C
   c. bitwise operators
   d. Binary fixed point numbers
   e. single-precision floating-point format
2. Binary Arithmetic, including
   a. Signed magnitude add/sub
   b. Unsigned add/sub/mul
   c. Two’s compliment add/sub/mul
   d. IEEE floating point add/sub/mul
3. Computing Systems
   a. Basic logic gates (and, or, not, xor)
   b. Determining the function of simple combinational circuits
   c. Adder and mux logic blocks
4. Assembly language programming
   a. Arithmetic and bitwise operations
   b. Procedure calls
   c. Stack & memory operations
   d. Assembly implementation of C control structures
5. An understanding of acceptable and unacceptable collaboration, the need to ensure permission to collaborate in a class, and an automatic urge to acknowledge collaborators and others who have assisted in a project.
Extended course description

Core topics:

1. Assembly language programming including
   a. Arithmetic and bitwise operations
   b. Arrays, stacks,
   c. Procedure calls
   d. Addressing modes
   e. Both CISC and RISC architectures
2. An understanding of basic computing systems including
   a. Basic logic gates and/or/xor/not
   b. Basic logic blocks (adder, mux)
   c. Registers, memory, CPU, I/O
   d. Steps to execute an instruction
   e. data structures
3. Binary arithmetic
   a. Signed magnitude add/sub
   b. Unsigned add/sub/mul/div
   b. Two's compliment add/sub/mul
   c. Floating point add/sub/mul
4. Number representations, including
   a. Arbitrary base conversion
   b. Binary, hex, decimal, 2's C
   c. Bitwise operators
   d. Binary fixed point numbers
   e. Arbitrary bases (e.g., 3, 60)
   f. Biased representation
   g. IEEE Floating point format
5. An understanding of basic system software including
   a. Assembly and compilation
   b. Loading and linking
   c. The basic functions of the operating system
   d. Interrupts and I/O
   e. Causes of interrupts
   f. Interrupt service routines
   g. Memory mapped I/O
Lab Kit

- Digilent Uno32
  - Arduino-compatible
- Digilent IO Shield
- PICKit3 Debugger
- Plastic case+cables+header
- Costs from BELS
  - ~$30 to rent for a quarter
  - ~$150 to buy
- Used in CMPE 13 and 118 too!

Get from BELS later in the quarter
Course work

• Class (CMPE12) Requirements
  – Attending lectures is highly recommended
  – Doing the weekly homework
    • Posted online, due on Ecommons
    • must be your OWN work
  – Quizzes + Midterm + Final

• Lab (CMPE12L) Requirements
  – Going to your lab section meetings for check offs and help
  – Weekly/Semi-weekly lab assignments
    • Posted online, submitted online
    • Lab report as text file

Maxwell James Dunne – Spring 2016
Lab work

• Part 1: Logic design with Multimedia Logic
  – Simulated logic gates

• Part 2: Programming assignments in LC-3
  – Simulated architecture

• Part 3: Programming assignments in PIC32/Arduino
  – Real hardware kit
Lab rules

- Each lab assignment consists of two things
  - Lab work (code, design file, etc.)
  - Lab write-up
- Lab assignment score = code + write-up
- Assignments are turned in through eComms
- **Must demo lab to TA/tutor in the week after the deadline**
  - They will use your submitted files
- Must submit both the lab code and the write-up
- Make sure to make the deadline, since even 1 second late will stamp your assignment as... **LATE**
Late Policy

• Each student has three “free” late days that they can use for any assignment in the class, at any time, without advanced permission. Any time you wish to use these late days you must fill out the late form the time you wish to use in 6 hour increments, this does not need to occur before you use them. When these late days are used up, late assignments will receive a zero.

• For this class there will be two sets of late days, one for the labs and one for the homework.
Get your lab checked off

- Submit the lab assignment by the deadline
  - Both the lab file(s) and the write-up file
  - Submission into eCommons is what really counts
- Go through the grading checklist in lab with the TA/tutor in person, in your lab
- Lock in your score with your TA/tutor
- Check off is required
- Only the submitted files will be considered “official”, regardless of what you showed in class
Quiz Format

• Quizzes generally on Thursdays out of 5 points.

• 1 point is for following format shown

<table>
<thead>
<tr>
<th>CMPE12 Quiz #1</th>
<th>Max Dunne mdunne</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Answer 1 goes here</td>
<td></td>
</tr>
<tr>
<td>2) Answer 2 goes here</td>
<td></td>
</tr>
</tbody>
</table>
Attendance

- Highly recommended for the class
  - How will I know if you cut?
  - Pop quizzes
  - Announcements

- Lab Attendance
  - Easier to get through the labs with help
  - TA/Tutors will be available then
  - Labs are not designed to be done within your lab section
    - Expect to spend much more time working outside of section than in
Grades for CMPE12/L

- 35% Labs
- 10% Homework
- 5% Quizzes
- 20% Midterm
- 30% Final

- Things you don’t turn in score as a 0
- Cannot make up missed exams
- Same Grade for the Lab as the class
Special needs

• Students with special needs should refer to the Disability Resource Center (DRC)
• Notify me within first week of class
• We will accommodate your needs
• Confirm with me and SOE office at least 1 week before each exam so we may try to accommodate you needs
Academic Dishonesty (Cheating)

- Cheating is presenting someone else’s work as your own
- All code turned in will be run against a code-checker
- Anyone caught cheating will immediately fail the class and the lab, and be reported to their college
- Copying each other’s code is never acceptable.

- Don’t do it—not worth it.
DO NOT CHEAT!
Week #1

- Labs start next week
Responsibility
History of Computers
The history of computers is interesting (or should be if you are in this class) and relevant to our professional lives.
The First Computer Hardware

Charles Babbage, born 1791

- Father of the computer
  - 1830 **Difference engine** - used mechanical power
  - calculated mathematical tables
  - smallest imperfections caused errors
  - Funded by the British government **17,000**

- Funding was pulled, even his colleagues thought it wouldn't work
  - conceived of **analytical engine** to perform many types of calculations
  - son built a model of the machine
  - working version finally built 1991
Ada, the countess of Lovelace

- Mother of computer programming – the first programmer!
- A gifted mathematician.
- She helped develop instructions for computations on the analytical engine.
- Saw Babbage's theoretical approach as workable.
History of Computers

The First Electrical computer

1890 Herman Hollerith
- Able to count the census in 6 weeks rather than 7 years
  - Used Jacquard’s punch cards
    - Sorted into bins
    - Count number of cards
- Developed in 1800 by a French silk weaver
- Electrical power
- Tabulating machine company merged into IBM in 1924
History of Computers

Aiken, Zuse, Atanasoff, Berry

- 1936 - Harvard graduate student Howard Aiken began thinking of modern equivalent of analytical engine...
- 1939 Germany - Konrad Zuse completed first programmable, general-purpose calculating device to solve mathematical problems
  - Paper was in short supply during war, used film tape
- 1939 - Iowa State Professor John Atanasoff developed the first electronic digital computer, the Atanasoff-Berry Computer (ABC)
  - Above is a picture of Berry
1944 Harvard professor Howard Aiken completed the Mark I

- Assistant Grace Hopper
  - Developed compiler for the computer
- 8 feet high, 55 feet long steel and glass
- used noisy electromechanical relays
- 5-6 times faster than a person
- not very efficient

Enter data into computer using paper tape
First Computer “Bug”

Found on the 9th of September, 1945, by Grace Murray Hopper while she was working on the Harvard University Mark II Aiken Relay Calculator (a primitive computer).

Coined term “debug”.

First actual case of bug being found.
ENIAC, UNIVAC by John Machly & J Presper

WWII - ENIAC Electronic Numerical Integrator and Computer
- based on the ABC
- machine to calculate trajectory tables for new guns
- First general-purpose computer

June 14, 1951
- UNIVAC 1 - Universal Automatic Computer
- First general-purpose commercial computer
Four generations of computers

1. 1951-1958 Vacuum Tube
   - about the size of light bulbs
   - thousands of them
   - is the bug a problem with tube or program?
   - machine code and punch cards

2. 1959-1964 Transistor
   - transfers electronic signals across resistor
   - assembly language
   - 1954 - FORTRAN

Transistor: mighty mite of electronics
Four generations of computers

3. 1965-1970 Integrated Circuit
   - complete electronic circuit on a small chip of silicon
   - silicon is a semiconductor - will transmit electrical signal when specific chemical impurities are introduced to lattice structure.
   - IBM 360 series of IBM
   - first time small and medium businesses could afford a computer.
   - unbundle software - sell software separately
   - birth of software industry

4. 1971-PRESENT Microprocessor (VLSI)
   - extension of third generation
   - get specialized chips for memory and logic
The Next Generation?

- I think the next generation is upon us and you are seeing it in your daily lives. I call it the SOC – System-on-a-Chip generation.
  - Put everything on a single chip
    - CPU
    - GPU
    - Memory (or at least part of it)
    - IO
  - Enables very low power with high performance
    - Smart phones, tablets, etc.
  - Also hear it called Ubiquitous Computing
    - Computing everywhere - Internet of Things (IoT)
History Summary

• Knowing something about the evolution of computers is helpful to understanding why things are the way they are now.
• Computing devices have been around a long time.
• Digital computers are fairly new.
• Rate of improvement and growth is amazing, Moore’s Law.
Digital Logic: From Transistors to Gates

Textbook Chapter 3
Announcements

• Auxiliary Website is up
  - [https://classes.soe.ucsc.edu/cmpe012/Winter16/](https://classes.soe.ucsc.edu/cmpe012/Winter16/)

• Office Hours
  - Tuesdays 2:30PM-4:30PM
  - Fridays 1PM-3PM
MSI
The Transistor

- Transistor: building block of computers
- Microprocessors contain tons of transistors
  - Intel Core i7-5960X (2015): 2.6 Billion
  - Qualcomm Snapdragon 810 (2015): multi billions
  - AMD 6-core Opteron (2009): 904 million
  - Intel Core i7 Quad (2008): 731 million
  - Intel Itanium 2 (2003): 220 million
  - Intel Pentium 4 (2000): 42 million
  - Intel 4004 (1971): 2300
The Transistor: Past and Present

● 1947 first point-contact transistor

Discrete transistor
Moore's Law

6nm Quantum effects

Microprocessor Transistor Counts 1971-2011 & Moore's Law

Intel

The number of active transistors per chip will double every 18 months.”

5pu
2,000

What Is a Transistor?

- A switch, which can close between the source and the drain
- Changing the voltage of the gate lets you change the current flow between the source and drain (closing or opening the switch)
- Think of a light switch, the gate is the switch that allows electricity to flow from the source to the drain
Metal-Oxide-Semiconductor transistor

NMOS Transistor (n-channel MOSFET)
FinFET

- Higher performance at lower voltages
- Less power
What is a transistor?

- Logically, each transistor is used as a switch
- Combined to implement logic functions
  - AND, OR, NOT
- Combined to build higher-level structures
  - Adder, multiplexer, decoder, register, ...
- Combined to build a processor
  - LC-3, Core i7, A9, etc
Simple switch circuit

Switch open:
- No current through circuit
- Light is off
- $V_{out}$ is +2.9V

Switch closed:
- Short circuit across switch
- Current flows
- Light is on
- $V_{out}$ is 0V

Switch-based circuits can easily represent two states: on/off, open/closed, voltage/no voltage.
n-type MOS transistor

n-type MOS (nMOS)

- when Gate has **positive** voltage, short circuit between #1 and #2 (switch **closed**)
- when Gate has **zero** voltage, open circuit between #1 and #2 (switch **open**)

Terminal #2 must be connected to GND (0V).
p-type MOS transistor

**p-type** is *complementary* to n-type

- when Gate has **positive** voltage, open circuit between #1 and #2 (switch open)
- when Gate has **zero** voltage, short circuit between #1 and #2 (switch closed)

Terminal #1 must be connected to +2.9V in this example.
Digital Values for Analog Signals

• Use the switch behavior of MOS transistors to implement logical functions: AND, OR, NOT

• Digital symbols:
  – We assign a range of analog voltages to each digital (logic) symbol
  – Assignment of voltage ranges depends on electrical properties of transistors being used
CMOS circuit

- CMOS is Complementary Metal Oxide Semiconductor
- Uses both n-type and p-type MOS transistors
  - p-type (pMOS)
    - Attached to + voltage
    - Pulls output voltage UP when input is zero
  - n-type (nMOS)
    - Attached to GND
    - Pulls output voltage DOWN when input is one
- Faster than using just one type
Truth Table

- The most basic representation of a logic function
- It is a perfect induction proof - Lists the output for all possible input combinations
- How many rows of the truth table needed?

\[
2 \Rightarrow n \cdot 2^k
\]
Truth Table: Inverter

- Inverted signals are denoted with an overbar
- Or with a prime symbol
  \( A' \)
- Or with a bubble in a circuit diagram

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( Y = A' )</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Inverter (NOT gate)

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 V</td>
<td>2.9 V</td>
</tr>
<tr>
<td>2.9 V</td>
<td>0 V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Table: AND Gate

- The result of an AND operation is 1 if and only if all inputs are 1
- Depict AND by the multiplication symbol
  - $A \cdot B$
- Or by lumping the signals together
  - $\overline{A \cdot B}$
- We don’t really build these gates...

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>$Y = A \cdot B$</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
FETs  BJT
NAND gate (NOT-AND)

Note: Parallel structure on top, serial on bottom.
AND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Add an inverter to a NAND.
Truth Table: OR Gate

- The result of an OR operation is 1 if and only if any inputs are 1.
- Depict OR by the addition symbol:
  - A+B

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>Y = A + B</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
NOR Gate: NOT-OR

Note: Serial structure on top, parallel on bottom.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Add an inverter to a NOR gate.
Truth table to transistors

- So giving some arbitrary truth table, how do you go about creating a transistor-based circuit for it?
- Typically this is only done for a handful of gate types.
- Recall:
  - PMOS (with the bubbles) on top
  - NMOS (no bubbles) on bottom
  - Series structure makes AND
  - Parallel structure makes OR
Transistor based designing

• How do you get from a truth table a transistor based circuit?

• Procedure:
  1. Find the rows with the ‘1’ output
  2. Use these to form the “pull-up” part of the circuit, remember p-type are active low
  3. Find the rows with the ‘0’ output
  4. Use these to form the “pull-down” part of the circuit, remember n-type are active high

  Note: This is not optimal
Simple example

- XOR Gate – one or the other, but not both

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Synthesis of an AOI Gate

- AOI means AND-OR-Invert

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0</td>
</tr>
</tbody>
</table>
## Synthesis of AOI Gate

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Why are our circuits so big?

- The circuits for the NAND and NOR were a lot smaller than the one we just did. Why is that?
- We just used the brute force method.
- To do correctly you need the Function and the dual of the function. E.g. for NAND:
  - \( F = (AB)' = A' + B' \)
  - \( F' = ((AB)')' = AB \)