CMPE012 – Computer Engineering 12 (and Lab)

Computing Systems and Assembly Language Programming
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 Ubio32

Maxwell James Dunne
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
  - Max Dunne (mdunne@soe.ucsc.edu)
- Teaching Assistants
  - Megan Boivin
Who to Email for Questions:

- **Ask in class**

- If it is specific to you (i.e. grades or of that nature) please contact the TA.

- If not specific, use Piazza. The TA’s will push you to Piazza if the question would benefit other students.
Email Etiquette

- I am always available to help if needed.
- Prefix the subject with “[CMPE12]” for all emails you send me.
- Make your subject descriptive, something generic doesn’t help very much and does not allow me to prioritize well.
- “Help” is not generally descriptive.
Our online presence...

- **Canvas** [https://canvas.ucsc.edu/](https://canvas.ucsc.edu/)
  - Assignments - Getting and submitting
  - Solutions

- **Online forum (Piazza)**
  - [https://piazza.com/ucsc/summer2017/cmpe12l](https://piazza.com/ucsc/summer2017/cmpe12l)
  - Main forum for communication

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

- Midterm
  - TBD but approximately halfway through the class
- Final
  - Last day of class
  - Thursday, August 17, 1:00–3:30 p.m.

- Note: The final cannot 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

- Binary Numbers
- Binary Number Math
- Logic Gates and Functions
- LC-3 Assembly Programming
- LC-3 Architecture
- LC-3 Advanced Features
What we will cover in this class

• Signed Magnitude
• Bias Notation
• Fixed and floating-point numbers and arithmetic
  – IEEE 754 Floating-Point
  – Floating Point Arithmetic
    – Addition
    – Subtraction
    – Multiplication
    – Not division
What we will cover in this class

- Microcontrollers and embedded systems
- PIC32 microcontroller
- MIPS assembly
- PIC32 I/O and interrupts
- MPLAB IDE (integrated Development Env.)
- PicKit3 Debugger
Potential Advanced Topics

- CMOS Systems
- Synthesis of Logic Gates
- NegaTernary
- Other Aspects of Architecture of Interest
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.
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, 2s 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
6. Interrupts and I/O
   a. Causes of interrupts            
   b. Interrupt service routines
   Memory mapped I/O
Course work

• Class (CMPE12) Requirements
  – Attending lectures is highly recommended
  – Doing the weekly/Semi-weekly homework
    • Posted online, due on Canvas
    • must be your OWN work
  – Midterm and 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
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
Lab rules

- Each lab assignment consists of two things
  - Lab work (code, design file, etc.)
  - Lab write-up
- Lab assignment score = code checkoff + write-up
- Assignments are turned in through Canvas
- **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 Canvas 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
Attendance

• Highly recommended for the class

• 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

- 25% Labs
- 15% Homework
- 25% Midterm
- 35% Final

- Things you don’t turn in score as a 0
- Cannot make up missed exams
- Same Grade for the Lab as the class
- You must score at least 50% on Labs, Midterm and Final to pass the class. This is a necessary but not sufficient condition for passing.
  - Lab is total score, not individual
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 your needs
Academic Dishonesty (Cheating)

- Cheating is presenting someone else’s work as your own
- 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 tomorrow
Responsibility

- First homework assignment is to read and indicate you understand it.
History of Computers
The 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

- 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
The First Programmer

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
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”.
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

Maxwell James Dunne
History of Computers

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)

```
car  so
```
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
Integer Numbers

The Number Bases of Integers

Textbook Chapter 2

CMPE-012/L
Number Systems

• Unary, or marks:
  - \(/\)/ = 7
  - \(/\)/ + \(/\)/ = \(/\)/\(/\)/\(/\)/\(/\)/\(/\)/\(/\)/

• Grouping lead to Roman Numerals:
  - \(\text{VII} + \text{V} = \text{VII} = \text{XII}\)

• Better: Arabic Numerals:
  - \(7 + 5 = 12 = 1 \cdot 10 + 2\)
Positional Number System

- The value represented by a digit depends on its position in the number.
- Ex: 1832

```
  1000
  100
  10
  1
```
Sexagesimal: A Positional Number System

- First used over 4000 years ago in Mesopotamia
- Base 60 (Sexagesimal), alphabet: 1..60, written as 60 different groups
- But the Babylonians used only two symbols, 1 and 10, and didn’t have the zero
  - Needed context to tell 1 from 60!
- Example
  - $5,45_{60} = 5 \cdot 60^1 + 45 \cdot 60^0$
  - $300 + 45 = 345$
  - $3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

Babylonian Numbers
Positional Number Systems

- Select a number as the base $b$
- Define an alphabet of $b-1$ symbols plus a symbol for zero to represent all numbers
- Use an *ordered* sequence of 1 or more digits $d$ to represent numbers
- The represented number is the sum of all digits, each multiplied by $b$ to the power of the digit’s position $p$

$$\text{Number} = \sum_{p=0}^{\text{num digits}} (d_p \cdot b^p)$$
Arabic/Indic Numerals

• Base (or radix): 10 (decimal)
  – The alphabet (digits or symbols) is 0..9
• Ours based on the Arabic symbols
  – Has the ZERO!!!
• Numerals introduced to Europe by Leonardo Fibonacci in his *Liber Abaci*
  – In 1202
  – So useful!
Arabic/Indic Numerals

- The Italian mathematician Leonardo Fibonacci
- Also known for the Fibonacci sequence
  - 1, 1, 2, 3, 5, 8, 13, 21

<table>
<thead>
<tr>
<th>European</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic-Indic</td>
<td>٠</td>
<td>١</td>
<td>٢</td>
<td>٣</td>
<td>٤</td>
<td>٥</td>
<td>٦</td>
<td>٧</td>
<td>٨</td>
<td>٩</td>
</tr>
<tr>
<td>Eastern Arabic-Indic (Persian and Urdu)</td>
<td>٠</td>
<td>١</td>
<td>٢</td>
<td>٣</td>
<td>٤</td>
<td>٥</td>
<td>٦</td>
<td>٧</td>
<td>٨</td>
<td>٩</td>
</tr>
<tr>
<td>Devanagari (Hindi)</td>
<td>०</td>
<td>१</td>
<td>२</td>
<td>३</td>
<td>४</td>
<td>५</td>
<td>६</td>
<td>७</td>
<td>८</td>
<td>९</td>
</tr>
<tr>
<td>Tamil</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
<td>₹</td>
</tr>
</tbody>
</table>
Numbers for Computers

- There are many ways to represent a number
- Representation does not affect computation result
- Representation affects difficulty of computing results
- Computers need a representation that works with (fast) electronic circuits
- Computers generally only have 2 states

B < D
Binary Number System

- Base (radix): 2
- Digits (symbols): 0, 1
- Binary Digits, or bits

Example:
- \(1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\)
  \(- 1001_2 = 8 + 1 = 9_{10}\)

- \(1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0\)
  \(- 11000_2 = 16 + 8 = 24_{10}\)
Knowing The Powers Of Two

- Know them in your sleep

<table>
<thead>
<tr>
<th>$2^0$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^1$</td>
<td>2</td>
</tr>
<tr>
<td>$2^2$</td>
<td>4</td>
</tr>
<tr>
<td>$2^3$</td>
<td>8</td>
</tr>
<tr>
<td>$2^4$</td>
<td>16</td>
</tr>
<tr>
<td>$2^5$</td>
<td>32</td>
</tr>
<tr>
<td>$2^6$</td>
<td>64</td>
</tr>
<tr>
<td>$2^7$</td>
<td>128</td>
</tr>
<tr>
<td>$2^8$</td>
<td>256</td>
</tr>
<tr>
<td>$2^9$</td>
<td>512</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>1024</td>
</tr>
</tbody>
</table>
Octal Number System

- Base (radix): 8
- Digits (symbols): 0, 1, 2, 3, 4, 5, 6, 7
- $345_8 = 3 \times 8^2 + 4 \times 8^1 + 5 \times 8^0$
  
  $= 3 \times 64 + 4 \times 8 + 5 \times 1 = 192 + 32 + 5 = 229_{10}$

- $1001_8 = 1 \times 8^3 + 0 \times 8^2 + 0 \times 8^1 + 1 \times 8^0$
  
  $= 1 \times 512 + 0 \times 64 + 0 \times 8 + 1 \times 1 = 512 + 1 = 513_{10}$

- In C, octal numbers are represented with a leading 0 (0345 or 01001).
Hexadecimal Number System

- Base (radix): 16
- Digits (symbols): 0-9, A–F (a–f)
- In C: leading “0x” (e.g., 0xa3)
- In LC-3: leading “x” (e.g., “x3000”)
- Hexadecimal is also known as “hex” for short

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>
Examples of Converting Hex to Decimal

- \( A3_{16} = A \times 16^1 + 3 \times 16^0 \)
  
  \[
  = 10 \times 16 + 3 \times 1 \\
  = 160 + 3 \\
  = 163_{10}
  \]

- \( 3E8_{16} = 3 \times 16^2 + E \times 16^1 + 8 \times 16^0 \)
  
  \[
  = 3 \times 256 + 14 \times 16 + 8 \times 1 \\
  = 768 + 224 + 8 \\
  = 10000_{10}
  \]
Base Conversion

Three cases:

I. From any base $b$ to base 10
II. From base 10 to any base $b$
III. From any base $b$ to any other base $c$
From Base \( b \) to Base 10

- Base (radix): \( b \)
- Digits (symbols): \( 0 \ldots (b-1) \)
- \( S_{n-1}S_{n-2} \ldots S_2S_1S_0 \)

Value = \( \sum_{i=0}^{n-1} (S_i b^i) \)

Use summation to transform any base to decimal.
From Base $b$ to Base 10

- Example: $1234_5 = ?_{10}$

\[
1 \cdot 5^3 + 2 \cdot 5^2 + 3 \cdot 5^1 + 4 \cdot 5^0
\]

\[
1 \cdot 125 + 2 \cdot 25 + 3 \cdot 5 + 4 \cdot 1
\]

\[
125 + 50 + 15 + 4
\]

\[
194_{10}
\]
From Base 10 to Base $b$ – Method 1

- Use successive divisions
- Remember the remainders
- Divide again with the quotients
From Base 10 to Base $b$ – Method 1

Example: $2010_{10} = \text{?}_5$

$2010/5 = 402 \text{ R} 0$
$402/5 = 80 \text{ R} 2$
$80/5 = 16 \text{ R} 0$
$16/5 = 3 \text{ R} 1$
$3/5 = 0 \text{ R} 3$

$5^4 = 625$
324\textsubscript{10} = \_2 = 101000100

324/2 = 162 \text{ R0}
162/2 = 81 \text{ R0}
81/2 = 40 \text{ R1}
40/2 = 20 \text{ R0}
20/2 = 10 \text{ R0}
10/2 = 5 \text{ R0}
5/2 = 2 \text{ R1}
2/2 = 1 \text{ R1}
1/2 = 0 \text{ R1}
From Base 10 to Base $b$ – Method 2

- Know your powers of the second base
- Subtract out the largest power of second base that fits
- Multiple by scalar, in case of binary, only a 1, so easy
- Put 1 in position for binary, scalar in position for other bases
- Repeat with remainder
From Base 10 to Base $b$ – Method 2

Example: $210_{10} = \square_2$

\[210 - 128 = 82\]
\[82 - 64 = 18\]
\[18 - 16 = 2\]
\[2 - 2 = 0\]

\[\begin{array}{c}
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 2 & 4 & 8 & 16 & 32 & 64 & 256
\end{array}\]

Example: $57_{10} = \square_3$

\[57 - 27 = 30 - 27 = 3\]
\[3 - 3 = 0\]
From Base $b$ to Base $c$

- Use a known intermediate base
- The easiest way is to convert from base $b$ to base 10 first, and then from 10 to $c$
- Or, in some cases, it is easier to use base 2 as the intermediate base (we’ll see them soon)
Decimal To Binary Conversion: Method 1

- Divide decimal value by 2 until the value is 0
- Example: $444_{10}$
  - Divide 444 by 2; what is the remainder? $0, 1$
  - Divide 222 by 2; what is the remainder?
  - ...
  - Result is 0: done

- Write the remainders starting from the least significant position (the right to the left)
Decimal To Binary Conversion: Method 2

- Know your powers of two and subtract... 256 128 64 32 16 8 4 2 1
- Example: $61_{10}$
  - What is the biggest power of two that fits?
  - What is the remainder?
  - What fits?
  - What is the remainder?
  - ...
  - What is the binary representation?
Binary to Octal Conversion

- Group into 3 starting at least significant bit
  - Why 3? 0-7
  - Add leading 0 as needed
    - Why not trailing 0s?
- Write one octal digit for each group
  000
  111
  7
Binary to Octal Conversion: Examples

- 100 010 111 (binary)
  4 2 7 (octal)

- 010 101 110 (binary)
  2 5 6 (octal)
Octal to Binary Conversion

- Write down the 3-bit binary code for each octal digit
- Example:
  - 047

<table>
<thead>
<tr>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>
Binary to Hex Conversion

• Group into 4 starting at least significant bit
  – Why 4? 0-15
  – Add leading 0 if needed
• Write one hex digit for each group

byte = 8 bits
Binary to Hex Conversion:

Examples

- $1001\ 1110\ 0111\ 0000$ (binary)
  
  $9\ E\ 7\ 0$ (hex)

- $0001\ 1111\ 1010\ 0011$ (binary)
  
  $1\ F\ A\ 3$ (hex)
Hex to Binary Conversion

- Write down the 4-bit binary code for each hex digit
- Example:
  - 0x 3 9 c 8

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bin</th>
<th>Hex</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>a</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>b</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>c</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>d</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>e</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>f</td>
<td>1111</td>
</tr>
</tbody>
</table>
## Conversion Table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>12</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>13</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>14</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>15</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>16</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>17</td>
<td>1111</td>
</tr>
</tbody>
</table>
More Conversions

• Hex → Octal
  – Do it in 2 steps
  – Hex → binary → octal

• Decimal → Hex
  – Do it in 2 steps
  – Decimal → binary → hex

• So why use hex and octal and not just binary and decimal?
  - Chunking
  - xDEADBEEF
  - 32 binary digits
Positional Addition

\[ \begin{array}{c}
113_5 \\
+ \\
42_5 \\
\hline
210_5
\end{array} \]
Addition

Just like other addition

Examples:

\[
\begin{array}{c}
100001 \quad (33) \\
+011101 \quad (29) \\
\hline
111110 \quad (62)
\end{array}
\]

\[
\begin{array}{c}
0001010 \quad (10) \\
+0001110 \quad (14) \\
\hline
11000 \quad (24)
\end{array}
\]
A Little Bit on Adding

More generally, it's just like decimal!!

\[
\begin{align*}
0 + 0 &= 0 \\
1 + 0 &= 1 \\
1 + 1 &= 2, \text{ which is } 10 \text{ in binary, sum is } 0, \text{ carry is } 1. \\
1 + 1 + 1 &= 3, \text{ sum is } 1, \text{ carry is } 1.
\end{align*}
\]

\[
\begin{array}{c}
x \quad 0011 \\
+ y \quad +0001 \\
\hline
\text{sum} \quad 0100
\end{array}
\]
010101 + 1011101 = 1110010
Largest Number

• What is the largest number that we can represent in \( n \) digits...
  – In base 10? \( 10^n - 1 \)
  – In base 2? \( 2^n - 1 \)
  – In octal?
  – In hex?
  – In base 7?
  – In base \( b \)? \( b^n - 1 \)

• How many different numbers can we represent with \( n \) digits in base \( b \)?
  \( b^n \)
Data Representation

Using binary numbers to represent information
Data Representation

- Goal: Store numbers, characters, sets, database records in the computer.
Definition: A unit of information. It is the amount of information needed to specify one of two equally likely choices.

- Example: Flipping a coin has 2 possible outcomes, heads or tails. The amount of info needed to specify the outcome is 1 bit.
## Storing Information

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>1e-4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

- Use more bits for more items
- Three bits can represent 8 things: 000, 001, ..., 111
- $N$ bits can represent $2^N$ things

<table>
<thead>
<tr>
<th>$N$ bits</th>
<th>Can represent</th>
<th>Which is approximately</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>16</td>
<td>65,536</td>
<td>65 thousand (64K where $K=1024$)</td>
</tr>
<tr>
<td>32</td>
<td>4,294,967,296</td>
<td>4 billion</td>
</tr>
<tr>
<td>64</td>
<td>$1.8446... \times 10^{19}$</td>
<td>20 billion billion</td>
</tr>
</tbody>
</table>
Integer Representation

Assume our representation has a fixed number of bits $n$ (e.g. 32 bits).

- Which 4 billion integers do we want?
  - There are an infinite number of integers less than zero and an infinite number greater than zero.

- What bit patterns should we select to represent each integer AND where the representation:
  - Does not effect the result of calculation
  - Does dramatically affect the ease of calculation

- Convert to/from human-readable representation as needed.
Integer Representation

Usual answers:

1. Represent 0 and consecutive positive integers
   - Unsigned integers
2. Represent positive and negative integers
   - Two’s complement
   - To be covered Later
     - Signed magnitude
     - Biased

Unsigned and two’s complement the most common
Unsigned Integers

- Integer represented is binary value of bits:
  - 0000 -> 0, 0001 -> 1, 0010 -> 2, ...
- Encodes only positive values and zero
- Range: 0 to $2^n - 1$, for $n$ bits
Unsigned Integers

If we have 4 bit numbers:

To find range make \( n = 4 \). Thus \( 2^4 - 1 \) is 15
Thus the values possible are 0 to 15

7 would be 0111
17 not represent able
-3 not represent able

For 32 bits:

Range is 0 to \( 2^{32} - 1 \) = [0: 4,294,967,295]
Which is 4,294,967,296 different numbers
Signed

\[ \text{num} - \searrow \text{num} + \]

repeated zero's (number)

\[ \pm 0 \]
Two’s Complement

- Two’s Complement sets the top bit negative.
- This makes the hardware that does arithmetic simpler and faster than the other representations as we do not have multiple representations of values.
- How to get 2’s complement representation:
  - Positive: just as if unsigned binary
  - Negative:
    - Take the positive value
    - Take the 1’s complement of it
    - Add 1
Two’s Complement

Example, what is -5 in 4-bit 2SC?

1. What is 5? 0101
2. Invert all the bits: 1010 (basically find the 1SC)
3. Add one: 1010 + 1 = 1011 which is -5 in 2SC

To get the additive inverse of a 2’s complement integer

1. Take the 1’s complement
2. Add 1

\[ \begin{align*}
1011 \\
0100 \\
+ 1 \\
0101
\end{align*} \]

\(-8 + 2 + 1 = -5\)
Two’s Complement

Number of integers representable is $-2^{n-1}$ to $2^{n-1}-1$

So if 4 bits:
\[-8, ..., -1, 0, +1, ..., +7] = 8 + 1 + 7 = 16 = 2^4\] numbers

With 32 bits:
\[-2^{31}, ..., -1, 0, +1, ..., (2^{31}-1)] = 2^{31} + 1 + (2^{31}-1) = 2^{32}\] numbers
\[-2147483648, ..., -1, 0, +1, ..., 2147483647] \sim \pm 2B\]
Two’s Complement Conversion

What is -12 in 8-bit 2’s complement form?

12
00001100

+1

11111001

11110100


1111110111

↑

1111

4

11111111

8
Addition: 2’s complement

- Just like unsigned addition
- Assume 6-bit and observe:

\[
\begin{array}{cccc}
000011 & 101000 & 111111 & 111111 \\
(3) & (-24) & (-1) & (-1)
\end{array}
\]

\[
\begin{array}{cccc}
+111100 & +010000 & +001000 & +001000 \\
(-4) & (16) & (8) & (8)
\end{array}
\]

\[
\begin{array}{cccc}
(-1) & (-8) & (7)
\end{array}
\]

- Ignore carry-outs (overflow)
- Sign bit is in the \(2^{n-1}\) bit position
- What does this mean for adding different signs?
Addition: 2’s complement

More examples: Convert to 2SC and do the addition

-20 + 15

5 + 12

-12 + -25
Subtraction

General rules:

$1 - 1 = 0$
$0 - 0 = 0$
$1 - 0 = 1$
$10 - 1 = 1$
$0 - 1 = \text{need to borrow!}$

• Or replace $(x - y)$ with $x + (-y)$
• Can replace subtraction with additive inverse and addition
Subtraction: 2’s complement

Don’t. Just use addition:

\[ x - y \rightarrow x + (-y) \]

Example:

\[
\begin{array}{c}
10110 \quad (-10) \\
- 00011 \quad (3)
\end{array}
\]

\[
\begin{array}{c}
10110 \quad (-10) \\
+ 11101 \quad (-3)
\end{array}
\]

\[
\begin{array}{c}
10011 \quad (-13)
\end{array}
\]
Subtraction: 2’s complement

Can also flip bits of bottom # and add an LSB carry in, so for -10 - 3 we get:

```
  1 10110
  + 11100
  ____ 10011
```

“add 1”

“flip bits of bottom number”

(throw away carry out)

Addition and subtraction are simple in 2’s complement, just need an adder and inverter.
Subtraction: unsigned

For n-bits use the 2’s complement method

\[
\begin{align*}
11100 \ ( +28) \\
- \quad 10110 \ ( +22) \\
\hline
\end{align*}
\]

Becomes

\[
\begin{align*}
1 \\
11100 \\
+01001 \\
\hline
00110
\end{align*}
\]

Only take 5 bits of result