The Hardware/Software Interface
CSE 351 Spring 2018

Instructor:
Dan Grossman

Teaching Assistants:
Natalie Andreeva
Parker DeWilde
Ruta Dhaneshwar
Bryan Hanner
Britt Henderson
Travis McGaha
Eric Mullen
Sam Wolfson

http://xkcd.com/676/
Welcome to CSE351!

- See the key abstractions “under the hood” to describe “what really happens” when a program runs
  - How is it that “everything is 1s and 0s”?
  - Where does all the data get stored and how do you find it?
  - How can more than one program run at once?
  - What happens to a Java or C program before the hardware executes it?
  - Why is recursion not even slightly magical?
  - And much, much, much more...

- An *introduction* that will:
  - Profoundly change/augment your view of computers and programs
  - Connect your source code down to the hardware
  - Leave you impressed that computers ever work
Concise To-Do List

- Review syllabus, course goals, collaboration policy, etc.: http://courses.cs.washington.edu/courses/cse351/18sp/
- Email-list settings, if necessary
- Beginning-of-course survey due Wednesday
- Lab 0, due Monday, April 2
  - Make sure you get our virtual machine set up and are able to do work
  - Basic exercises to start getting familiar with C
  - Get this done as quickly as possible
- Homework 1, also due Monday, April 2
- Section Thursday
  - Please install the virtual machine BEFORE coming to section
  - Includes activities to help you with Lab 0 and Homework 1 – and more!
Who: Course Staff

❖ Your Instructor:
  ▪ Excited to be teaching 351 for the 2\textsuperscript{nd} time!
  ▪ Compare: 341 10x, 331 3x, 332 2x 😊

❖ TAs:
  ▪ Available in section, office hours, via email, discussion board
  ▪ An invaluable source of information and help

❖ Get to know us
  ▪ We are here to help you succeed!
  ▪ And enjoy helping you explore a new world
Acknowledgments

Many thanks to the many people whose course content we are liberally reusing with at most minor changes

- UW: Gaetano Borriello, Luis Ceze, Peter Hornyack, Hal Perkins, Ben Wood, John Zahorjan, Katelin Bailey, Ruth Anderson, Justin Hsia, ...
- CMU: Randy Bryant, David O’Halloran, Gregory Kesden, Markus Püschel
- Harvard: Matt Welsh (now at Google-Seattle)
- Not listed: dozens of TAs
Who are You?

- ~ 165 students
  - Yikes – will do my very best to make it feel like 50
  - You all belong here!

- CSE majors, EE majors, and more
  - Most of you will find almost everything in the course new

- Get to know each other and help each other out!
  - Learning is much more fun with friends
  - Working well with others is a valuable life skill
  - Diversity of perspectives expands your horizons
Staying in Touch

- Course web page
  - Schedule, policies, labs, homeworks, and everything else
- Course discussion board
  - Keep in touch outside of class – help each other
  - Staff will monitor and contribute
- Course mailing list cse351a_sp18@u.washington.edu
  - Low traffic – mostly announcements; your @uw.edu is subscribed
- Office hours, appointments, drop-ins
  - Will spread throughout the week
- Staff e-mail (Dan + TAs): cse351-staff@cs.uw.edu
  - For things that are not a good fit for the discussion board
- Anonymous feedback
  - Comments about anything related to the course where you would feel better not attaching your name: goes only to Dan
Textbooks

- **Computer Systems: A Programmer’s Perspective**
  - Randal E. Bryant and David R. O’Hallaron
  - Website: [http://csapp.cs.cmu.edu](http://csapp.cs.cmu.edu)
  - Must be 3rd edition
    - [http://csapp.cs.cmu.edu/3e/changes3e.html](http://csapp.cs.cmu.edu/3e/changes3e.html)
    - [http://csapp.cs.cmu.edu/3e/errata.html](http://csapp.cs.cmu.edu/3e/errata.html)
  - This book really matters for the course!
    - How to solve labs
    - Practice problems and homework

- **A good C book – any will do**
  - *The C Programming Language* (Kernighan and Ritchie)
  - *C: A Reference Manual* (Harbison and Steele) [Dan’s preference]
Course Components

- **Lectures**
  - Introduce the concepts; supplemented by textbook and videos

- **Sections**
  - Apply concepts, important tools and skills for labs, clarification of lectures, exam review and preparation

- **Online homework assignments (5)**
  - Problems to solidify understanding; submitted as Canvas quizzes

- **Programming lab assignments (5.5)**
  - Provide in-depth understanding (via practice) of an aspect of system

- **Exams (2)**
  - **Midterm:** Friday, April 27 (in class)
  - **Final:** Wednesday, June 6 2:30-4:20pm
Collaboration and Academic Integrity

- All submissions are expected to be yours and yours alone

- You are encouraged to discuss your assignments with other students (ideas), but we expect that what you turn in is yours

- It is NOT acceptable to copy solutions from other students or to copy (or start your) solutions from the Web (including Github)

- Our goal is that *YOU* learn the material so you will be prepared for exams, interviews, and the future
More logistics stuff?

Questions before we get to course content??
The Hardware/Software Interface

- Why do we need a hardware/software interface?
- Why do we need to understand both sides of this interface?
C/Java, assembly, and machine code

```
if (x != 0) y = (y+z)/x;
```

High Level Language (e.g. C, Java)

```
cmpl $0, -4(%ebp)
je .L2
movl -12(%ebp), %eax
movl -8(%ebp), %edx
leal (%edx,%eax), %eax
movl %eax, %edx
sarl $31, %edx
idivl -4(%ebp)
movl %eax, -8(%ebp)
.L2:
```

Assembly Language

```
1000001101111100001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
0111010000011000
10001011010001000110000000111101111000010000001001000001110000000000
```

Machine Code
C/Java, assembly, and machine code

The 3 program fragments are equivalent

You’d rather write C! (more human-friendly)

Hardware executes strings of bits
  - In reality everything is voltages
  - The machine instructions are actually much shorter than the number of bits we would need to represent the characters in the assembly language
HW/SW Interface: Historical Perspective

- Hardware started out quite primitive

Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.

Photo: Corbis

HW/SW Interface: Historical Perspective

- Hardware started out quite primitive
  - Programmed with very basic instructions (*primitives*)
  - e.g. a single instruction for adding two integers

- Software was also very basic
  - Closely reflected the actual hardware it was running on
  - Specify each step manually
HW/SW Interface: Assemblers

- Life was made a lot better by assemblers
  - 1 assembly instruction = 1 machine instruction
  - More human-readable syntax
    - Assembly instructions are character strings, not bit strings
  - Can use symbolic names
HW/SW Interface: Higher-Level Languages

- Higher level of abstraction
  - 1 line of a high-level language is *compiled* into many (sometimes very many) lines of assembly language
HW/SW Interface: Compiled Programs

Note: The compiler and assembler are just programs, developed using this same process.
Roadmap

C:

```c
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```java
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg = c.getMPG();
```

Assembly language:

```
get_mpg:
    pushq  %rbp
    movq   %rsp, %rbp
    ...
    popq   %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101101000011111
```

Computer system:

- OS:
  - Windows 10
  - OS X Yosemite

Memory & data
Integers & floats
x86 assembly

Procedures & stacks
Executables
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C
**Course Perspective**

- **CSE351 will make you a better programmer**
  - Purpose is to show how software really works
    - Understanding of some of the abstractions that exist between programs and the hardware they run on, why they exist, and how they build upon each other
  - Understanding the underlying system makes you more effective
    - Better debugging
    - Better basis for evaluating performance
    - How multiple activities work in concert (e.g. OS and user programs)
    - “Stuff everybody learns and uses and forgets not knowing”

- **CSE351 presents a world-view that will empower you**
  - The intellectual and software tools to understand the trillions+ of 1s and 0s that are “flying around” when your program runs
Writing Assembly Code?? In 2018??

- Chances are, you’ll never write a whole program in assembly
  - Compilers are much better and more patient than you are

- But: understanding assembly is the key to the machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language model breaks down
  - Tuning program performance
    - Understand optimizations done/not done by the compiler
    - Understanding sources of program inefficiency
  - Fighting malicious software

- Also needed for:
  - Implementing key pieces of system software / embedded systems
  - Using special units (timers, I/O co-processors, etc.) inside processor
Decimal Numbering System

- Ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Represent larger numbers as a sequence of digits
  - Each digit is one of the available symbols

Example: 7061 in decimal (base 10)
- $7061_{10} = (7 \times 10^3) + (0 \times 10^2) + (6 \times 10^1) + (1 \times 10^0)$
Octal Numbering System

- Eight symbols: 0, 1, 2, 3, 4, 5, 6, 7
  - Notice that we no longer use 8 or 9

- Base comparison:
  - Base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
  - Base 8: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14...

- Example: What is $7061_8$ in base 10?
  - $7061_8 = (7 \times 8^3) + (0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 3633_{10}$
Peer Instruction Question

❖ What is $34_8$ in base 10?

A. $32_{10}$
B. $34_{10}$
C. $7_{10}$
D. $28_{10}$
E. $35_{10}$

❖ Think on your own for a minute, then discuss with your neighbor(s)
Binary and Hexadecimal

- **Binary** is base 2
  - Symbols: 0, 1
  - Convention: \(2_{10} = 10_2 = 0b10\)
  - Example: What is 0b110 in base 10?
    \[0b110 = 110_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6_{10}\]

- **Hexadecimal** (hex, for short) is base 16
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Convention: \(16_{10} = 10_{16} = 0x10\)
  - Example: What is 0xA5 in base 10?
    \[0xA5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}\]
Converting to Base 10

- Can convert from any base to base 10
  - $0b110 = 110_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6_{10}$
  - $0xA5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}$

- We learned to think in base 10, so this is fairly natural for us

- Challenge: Convert into other bases (e.g. 2, 16)
Challenge Question

❖ Convert $13_{10}$ into binary

❖ Hints:
  - $2^3 = 8$
  - $2^2 = 4$
  - $2^1 = 2$
  - $2^0 = 1$

❖ Think on your own for a minute, then discuss with your neighbor(s)
Converting from Decimal to Binary

- Given a decimal number N:
  - List increasing powers of 2 from *right to left* until $\geq N$
  - Then from *left to right*, ask is that (power of 2) $\leq N$?
    - If **YES**, put a 1 below and subtract that power from N
    - If **NO**, put a 0 below and keep going

- Example: 13 to binary

```
<table>
<thead>
<tr>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```


Converting from Decimal to Base B

- Given a decimal number N:
  - List increasing powers of B from right to left until \( \geq N \)
  - Then from left to right, ask is that (power of B) \( \leq N \)?
    - If YES, put *how many* of that power go into N and subtract from N
    - If NO, put a 0 below and keep going

- Example: 165 to hex

<table>
<thead>
<tr>
<th>(16^2=256)</th>
<th>(16^1=16)</th>
<th>(16^0=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Converting Binary $\leftrightarrow$ Hexadecimal

- **Hex → Binary**
  - Substitute hex digits, then drop any leading zeros
  - Example: 0x2D to binary
    - 0x2 is 0b0010, 0xD is 0b1101
    - Drop two leading zeros, answer is 0b101101

- **Binary → Hex**
  - Pad with leading zeros until multiple of 4 bits, then substitute each group of 4
  - Example: 0b101101
    - Pad to 0b 0010 1101
    - Substitute to get 0x2D

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

- Convert 0b100110110101101
  - How many digits?
  - Pad:
  - Substitute:

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

Why does all of this matter?

- **Humans** think about numbers in **base 10**, but **computers** “think” about numbers in **base 2**
- **Binary encoding** is what allows computers to do all of the amazing things that they do!

You should have this table memorized by the end of the class

- Might as well start now!

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

- **AMAZING FACT:** You can represent anything countable using numbers!
  - Need to agree on an encoding
  - Kind of like learning a new language

- **Examples:**
  - Decimal Integers: 0→0b0, 1→0b1, 2→0b10, etc.
  - English Letters: CSE→0x435345, yay→0x796179
  - Emoticons: 😃 0x0, 😞 0x1, 😎 0x2, 😁 0x3, 😈 0x4, 🙁 0x5
Binary Encoding

- With N binary digits, how many “things” can you represent?
  - Need N binary digits to represent \( n \) things, where \( 2^N \geq n \)
  - Example: 5 binary digits for alphabet because \( 2^5 = 32 > 26 \)
  - Example: < 300 binary digits for every atom in the universe

- A binary digit is known as a **bit**
- A group of 4 bits (1 hex digit) is called a **nibble**
- A group of 8 bits (2 hex digits) is called a **byte**
  - 1 bit → 2 things, 1 nibble → 16 things, 1 byte → 256 things
So What’s It Mean?

❖ A sequence of bits can have many meanings!

❖ Consider the hex sequence 0x4E6F21
  ▪ Common interpretations include:
    • The decimal number 5140257
    • The characters “No!”
    • The background color of this slide
    • The fractional number \(7.203034 \times 10^{-39}\)

❖ It is up to the program/programmer to decide how to interpret the sequence of bits
Binary Encoding – Colors

- **RGB – Red, Green, Blue**
  - Additive color model (light): byte (8 bits) for each color
  - Commonly seen in hex (in HTML, photo editing, etc.)
  - Examples: Blue → 0x0000FF, Gold → 0xFFD700, White → 0xFFFFFFFF, Deep Pink → 0xFF1493
Binary Encoding – Characters/Text

- ASCII Encoding ([www.asciitable.com](http://www.asciitable.com))
  - American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>040</td>
<td>#32</td>
<td>Space</td>
<td>64</td>
<td>100</td>
<td>#64</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>041</td>
<td>#33</td>
<td>!</td>
<td>65</td>
<td>101</td>
<td>#65</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>042</td>
<td>#34</td>
<td>&quot;</td>
<td>66</td>
<td>102</td>
<td>#66</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>043</td>
<td>#35</td>
<td>#</td>
<td>67</td>
<td>103</td>
<td>#67</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>044</td>
<td>#36</td>
<td>$</td>
<td>68</td>
<td>104</td>
<td>#68</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>045</td>
<td>#37</td>
<td>%</td>
<td>69</td>
<td>105</td>
<td>#69</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>046</td>
<td>#38</td>
<td>+</td>
<td>70</td>
<td>106</td>
<td>#70</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>047</td>
<td>#39</td>
<td>'</td>
<td>71</td>
<td>107</td>
<td>#71</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>050</td>
<td>#40</td>
<td>(</td>
<td>72</td>
<td>110</td>
<td>#72</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41</td>
<td>051</td>
<td>#41</td>
<td>)</td>
<td>73</td>
<td>111</td>
<td>#73</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>A12</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>052</td>
<td>#42</td>
<td>*</td>
<td>74</td>
<td>112</td>
<td>#74</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B13</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>053</td>
<td>#43</td>
<td>+</td>
<td>75</td>
<td>113</td>
<td>#75</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>C14</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>054</td>
<td>#44</td>
<td>-</td>
<td>76</td>
<td>114</td>
<td>#76</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>D15</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>055</td>
<td>#45</td>
<td>(</td>
<td>77</td>
<td>115</td>
<td>#77</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>E16</td>
<td>SO (shift out)</td>
<td>46</td>
<td>056</td>
<td>#46</td>
<td>)</td>
<td>78</td>
<td>116</td>
<td>#78</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>F17</td>
<td>SI (shift in)</td>
<td>47</td>
<td>057</td>
<td>#47</td>
<td>/</td>
<td>79</td>
<td>117</td>
<td>#79</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>102</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>060</td>
<td>#48</td>
<td>0</td>
<td>80</td>
<td>120</td>
<td>#80</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>101</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>061</td>
<td>#49</td>
<td>1</td>
<td>81</td>
<td>121</td>
<td>#81</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>102</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>062</td>
<td>#50</td>
<td>2</td>
<td>82</td>
<td>122</td>
<td>#82</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>103</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>063</td>
<td>#51</td>
<td>3</td>
<td>83</td>
<td>123</td>
<td>#83</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>104</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>064</td>
<td>#52</td>
<td>4</td>
<td>84</td>
<td>124</td>
<td>#84</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>105</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>065</td>
<td>#53</td>
<td>5</td>
<td>85</td>
<td>125</td>
<td>#85</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>106</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>066</td>
<td>#54</td>
<td>6</td>
<td>86</td>
<td>126</td>
<td>#86</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>107</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>067</td>
<td>#55</td>
<td>7</td>
<td>87</td>
<td>127</td>
<td>#87</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>108</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>070</td>
<td>#56</td>
<td>8</td>
<td>88</td>
<td>130</td>
<td>#88</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>109</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>071</td>
<td>#57</td>
<td>9</td>
<td>89</td>
<td>131</td>
<td>#89</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>110</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>072</td>
<td>#58</td>
<td>0</td>
<td>90</td>
<td>132</td>
<td>#90</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>111</td>
<td>ESC (escape)</td>
<td>59</td>
<td>073</td>
<td>#59</td>
<td>1</td>
<td>91</td>
<td>133</td>
<td>#91</td>
<td>A</td>
</tr>
<tr>
<td>28</td>
<td>112</td>
<td>FS (file separator)</td>
<td>60</td>
<td>074</td>
<td>#60</td>
<td>$</td>
<td>92</td>
<td>134</td>
<td>#92</td>
<td>B</td>
</tr>
<tr>
<td>29</td>
<td>113</td>
<td>GS (group separator)</td>
<td>61</td>
<td>075</td>
<td>#61</td>
<td>%</td>
<td>93</td>
<td>135</td>
<td>#93</td>
<td>C</td>
</tr>
<tr>
<td>30</td>
<td>114</td>
<td>RS (record separator)</td>
<td>62</td>
<td>076</td>
<td>#62</td>
<td>+</td>
<td>94</td>
<td>136</td>
<td>#94</td>
<td>D</td>
</tr>
<tr>
<td>31</td>
<td>115</td>
<td>US (unit separator)</td>
<td>63</td>
<td>077</td>
<td>#63</td>
<td>)</td>
<td>95</td>
<td>137</td>
<td>#95</td>
<td>E</td>
</tr>
</tbody>
</table>

Source: [www.LookupTables.com](http://www.LookupTables.com)
Binary Encoding – Files and Programs

- At the lowest level, all digital data is stored as bits!

- Layers of abstraction keep everything comprehensible
  - Data/files are groups of bits interpreted by program
  - Program is actually groups of bits being interpreted by your CPU

- Computer Memory Demo (if time)
  - From vim: `%!xxd`
  - From emacs: `M-x hexl-mode`
Summary

- Humans think about numbers in decimal; computers think about numbers in binary
  - Base conversion to go between them
  - Hexadecimal is more human-readable than binary

- All information on a computer is binary
  - For physical-world engineering reasons!

- Binary encoding can represent *anything*!
  - Computer/program needs to know how to interpret the bits