The Hardware/Software Interface
CSE 351 Autumn 2017

Instructor:
Justin Hsia

Teaching Assistants:
Lucas Wotton
Michael Zhang
Parker DeWilde
Ryan Wong
Sam Gehman
Sam Wolfson
Savanna Yee
Vinny Palaniappan

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?
  - And 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
Who: Course Staff

- Your Instructor: just call me Justin
  - From California (UC Berkeley and the Bay Area)
  - I like: teaching, the outdoors, board games, and ultimate
  - Excited to be teaching this course for the 2nd time!

- TAs:
  - Available in section, office hours, via email, on Piazza
  - An invaluable source of information and help

- Get to know us
  - We are here to help you succeed!
Who are You?

- ~185 students registered, split across two lectures
  - See me if you are interested in taking the class but are not yet registered
- 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
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;

Compiler

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:

Assembler

```
100000110111110001001000011100000000
0111010000011000
1000101111000100100001001000010100
10001011100011000100100010100100100
10011010100001100000000010
100010011100010
110000111111101000011111
111101111111111000011110100001111100
10010010100010000000100100000011100
```
C/Java, assembly, and machine code

- All 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

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

```
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:
```
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**

**Code Time**
- User program in C
- .c file

**Compile Time**
- C Compiler
- Assembler

**Run Time**
- Hardware
- .exe file

**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);
```

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

Machine code:
```
0111010000011000
1000110100000100000000101000100111000010110000011111101000011111
```

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

Computer system:
- Intel Core i5
- RAM
- SSD

Operating system:
- 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
Lecture Outline

- Course Introduction

- **Course Policies**

- Binary
Communication

❖ Website:  [http://cs.uw.edu/351](http://cs.uw.edu/351)
  ▪ Schedule, policies, materials, videos, assignments, etc.

  ▪ Announcements made here
  ▪ Ask and answer questions – staff will monitor and contribute

❖ Office Hours: spread throughout the week
  ▪ Can also e-mail to make individual appointments

❖ Anonymous feedback:
  ▪ Comments about anything related to the course where you would feel better not attaching your name
  ▪ Can send to individual staff member of whole staff
Textbooks

❖ **Computer Systems: A Programmer’s Perspective**
  - Randal E. Bryant and David R. O’Hallaron
  - Website: http://csapp.cs.cmu.edu
  - Must be 3rd edition
    - http://csapp.cs.cmu.edu/3e/changes3e.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)
Course Components

- **Lectures (29)**
  - Introduce the concepts; supplemented by textbook

- **Sections (10)**
  - Applied 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:** Monday, October 30, 5:00-6:30pm (joint)
  - **Final:** Wednesday, December 13, 12:30-2:20pm (joint)
Grading

- **Homework:** 20% total
  - Autograded; 20 submission attempts
  - *Group work okay*

- **Labs:** 30% total
  - Graded by TAs; last submission graded
  - *Individual work only*

- **Exams:** Midterm (15%) and Final (30%)
  - Many old exams on course website

- **EPA:** Effort, Participation, and Altruism (5%)

- More details on course website
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
EPA

- Encourage class-wide learning!

- Effort
  - Attending office hours, completing all assignments
  - Keeping up with Piazza activity

- Participation
  - Making the class more interactive by asking questions in lecture, section, office hours, and on Piazza
  - Peer instruction voting

- Altruism
  - Helping others in section, office hours, and on Piazza
Peer Instruction

- Increase real-time learning in lecture, test your understanding, increase student interactions
  - Lots of research supports its effectiveness

- Multiple choice question at end of lecture “segment”
  - 1 minute to decide on your own
  - 2-4 minutes in pairs to reach consensus
  - Learn through discussion

- Vote using Poll Everywhere
  - Use website (https://www.polleverywhere.com) or app
  - Linked to your UWNetID
Some fun topics that we will touch on

- Which of the following seems the most interesting to you? (vote at http://PollEv.com/justinh)
  
a) What is a GFLOP and why is it used in computer benchmarks?
  
b) How and why does running many programs for a long time eat into your memory (RAM)?
  
c) What is stack overflow and how does it happen?
  
d) Why does your computer slow down when you run out of disk space?
  
e) What was the flaw behind the original Internet worm, the Heartbleed bug, and the Cloudbleed bug?
  
f) What is the meaning behind the different CPU specifications? (e.g. # of cores, # and size of cache, supported memory types)
Tips for Success in 351

- Attend all lectures and sections
  - Only use Panopto for review
  - Avoid devices during lecture except for PollEverywhere

- Learn by doing
  - Can answer many questions by writing small programs

- Visit Piazza often
  - Ask questions and try to answer fellow students’ questions

- Go to office hours
  - Even if you don’t have specific questions in mind

- Find a study and homework group

- Start assignments early

- Don’t be afraid to ask questions
To-Do List

❖ Admin
  ▪ Explore/read website *thoroughly*: [http://cs.uw.edu/351](http://cs.uw.edu/351)
  ▪ Check that you are enrolled in Piazza
  ▪ Log in to Poll Everywhere
  ▪ Get your machine set up for this class (VM or attu) *as soon as possible*

❖ Assignments
  ▪ Pre-Course Survey due Friday (9/29)
  ▪ Lab 0 due Monday (10/2)
  ▪ HW 1 due Wednesday (10/4)
Lecture Outline

- Course Introduction
- Course Policies

- Binary
  - Decimal, Binary, and Hexadecimal
  - Base Conversion
  - Binary Encoding
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 7061₈ in base 10?
  - \(7061₈ = (7 \times 8^3) + (0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 3633_{10}\)
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)
  
  ▪ No voting for this question
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?
  - $0xA5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}$
Peer Instruction Question

Which of the following orderings is correct?

A. $0xC < 0b1010 < 11$
B. $0xC < 11 < 0b1010$
C. $11 < 0b1010 < 0xC$
D. $0b1010 < 11 < 0xC$
E. $0b1010 < 0xC < 11$

Think on your own for a minute, then discuss with your neighbor(s)

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)
  - No voting for this question
Converting from Decimal to Binary

Given a decimal number N:
- List increasing powers of 2 from right to left until ≥ N
- Then from left to right, ask is that (power of 2) ≤ 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=16</th>
<th>2^3=8</th>
<th>2^2=4</th>
<th>2^1=2</th>
<th>2^0=1</th>
</tr>
</thead>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></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 ↔ 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, 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 $\rightarrow$ 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 $2^5 = 32 > 26$

- 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 $\rightarrow$ 2 things, 1 nibble $\rightarrow$ 16 things, 1 byte $\rightarrow$ 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 real 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 → 0xFFFFFF, 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</th>
<th>Oct</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32: 20 040</td>
<td>64: 40 100</td>
<td>96: 60 140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33: 21 041</td>
<td>65: 41 101</td>
<td>97: 61 141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34: 22 042</td>
<td>66: 42 102</td>
<td>98: 62 142</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35: 23 043</td>
<td>67: 43 103</td>
<td>99: 63 143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36: 24 044</td>
<td>68: 44 104</td>
<td>100: 64 144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37: 25 045</td>
<td>69: 45 105</td>
<td>101: 65 145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38: 26 046</td>
<td>70: 46 106</td>
<td>102: 66 146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39: 27 047</td>
<td>71: 47 107</td>
<td>103: 67 147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40: 28 050</td>
<td>72: 48 110</td>
<td>104: 68 150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>HT (horizontal tab)</td>
<td>41: 29 051</td>
<td>73: 49 111</td>
<td>105: 69 151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A12</td>
<td>LF (NL line feed, new line)</td>
<td>42: 2A 052</td>
<td>74: 4A 112</td>
<td>106: 6A 152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B13</td>
<td>VT (vertical tab)</td>
<td>43: 2B 053</td>
<td>75: 4B 113</td>
<td>107: 6B 153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C14</td>
<td>FF (NP form feed, new page)</td>
<td>44: 2C 054</td>
<td>76: 4C 114</td>
<td>108: 6C 154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D15</td>
<td>CR (carriage return)</td>
<td>45: 2D 055</td>
<td>77: 4D 115</td>
<td>109: 6D 155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>E16</td>
<td>SO (shift out)</td>
<td>46: 2E 056</td>
<td>78: 4E 116</td>
<td>110: 6E 156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F17</td>
<td>ST (shift in)</td>
<td>47: 2F 057</td>
<td>79: 4F 117</td>
<td>111: 6F 157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>102</td>
<td>DLE (data link escape)</td>
<td>48: 30 060</td>
<td>80: 50 120</td>
<td>112: 70 160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>101</td>
<td>DC1 (device control 1)</td>
<td>49: 31 061</td>
<td>81: 51 121</td>
<td>113: 71 161</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>122</td>
<td>DC2 (device control 2)</td>
<td>50: 32 062</td>
<td>82: 52 122</td>
<td>114: 72 162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>123</td>
<td>DC3 (device control 3)</td>
<td>51: 33 063</td>
<td>83: 53 123</td>
<td>115: 73 163</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>124</td>
<td>DC4 (device control 4)</td>
<td>52: 34 064</td>
<td>84: 54 124</td>
<td>116: 74 164</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>125</td>
<td>NAK (negative acknowledge)</td>
<td>53: 35 065</td>
<td>85: 55 125</td>
<td>117: 75 165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>162</td>
<td>SYN (synchronous idle)</td>
<td>54: 36 066</td>
<td>86: 56 126</td>
<td>118: 76 166</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>172</td>
<td>ETB (end of trans. block)</td>
<td>55: 37 067</td>
<td>87: 57 127</td>
<td>119: 77 167</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>103</td>
<td>CAN (cancel)</td>
<td>56: 38 070</td>
<td>88: 58 130</td>
<td>120: 78 170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>193</td>
<td>EM (end of medium)</td>
<td>57: 39 071</td>
<td>89: 59 131</td>
<td>121: 79 171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1A3</td>
<td>SUB (substitute)</td>
<td>58: 3A 072</td>
<td>90: 5A 132</td>
<td>122: 7A 172</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>1B3</td>
<td>ESC (escape)</td>
<td>59: 3B 073</td>
<td>91: 5B 133</td>
<td>123: 7B 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1C3</td>
<td>FS (file separator)</td>
<td>60: 3C 074</td>
<td>92: 5C 134</td>
<td>124: 7C 174</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D3</td>
<td>GS (group separator)</td>
<td>61: 3D 075</td>
<td>93: 5D 135</td>
<td>125: 7D 175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1E3</td>
<td>RS (record separator)</td>
<td>62: 3E 076</td>
<td>94: 5E 136</td>
<td>126: 7E 176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1F3</td>
<td>US (unit separator)</td>
<td>63: 3F 077</td>
<td>95: 5F 137</td>
<td>127: 7F 177</td>
<td></td>
<td></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
  - From vim: `% !xxd`
  - From emacs: `M-x hexl-mode`
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

Binary encoding can represent anything!

- Computer/program needs to know how to interpret the bits