The Hardware/Software Interface
CSE 351 Spring 2020

Instructor:
Porter Jones

Teaching Assistants:
Amy Xu
Callum Walker
Sam Wolfson
Tim Mandzyuk

http://xkcd.com/676/
Introductions: Course Staff

❖ Instructor: Porter Jones

❖ TAs:

- Learn more about me and the staff on the course website!
- Available in section, office hours, and on Piazza
- An invaluable source of information and help

❖ Get to know us

- We are here to help you succeed!
Introductions: You!

❖ ~60 students registered

❖ CSE majors, ECE 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
Welcome to Summer 2020!

❖ Thanks in advance for working with us to make this the best online experience we can!

❖ Help us figure out the best ways to handle:
  ▪ Lecture
  ▪ Office Hours (Schedule on Website & Zoom links coming soon!)
  ▪ Sections (Zoom links coming soon!)
  ▪ Students in different time zones
  ▪ Other challenges/opportunities!

❖ We’ll be experimenting with different formats/approaches to see what works best!
Aside: Lecture Questions

❖ Ask lots of questions, please!!!
❖ The Zoom chat box is convenient, but that means it can become cluttered fast
  ▪ Good for clarifying questions, asking me to repeat stuff, telling me that my Zoom cutout, etc.

❖ Each lecture will have a google doc for asking questions that TAs will monitor/answer
  ▪ Good for curiosity questions, tangential questions, etc.

❖ Today’s doc: https://tinyurl.com/CSE351-6-22
Welcome to CSE351!

Our goal is to teach you the key abstractions “under the hood”

- How does your source code become something that your computer understands?
- What happens as your computer is executing one or more processes (applications)?
Welcome to CSE351!

❖ This is an *introduction* that will:
  ▪ Profoundly change/augment your view of computers and programs
  ▪ Leave you impressed that computers ever work
Code in Many Forms

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

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

```
cmpl $0, -4(%ebp)
j e .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
10001011010001000010010000011000
10001011010001100010010100010100
100011010000010000000010
1000100111000010
110000011111101000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

### Machine Code
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

https://s-media-cache-ak0.pinimg.com/564x/91/37/23/91372375e2e6517f8af128aa655e3b4.jpg
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
How does your source code become something that your computer understands?

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

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
110000011111101000011111
```

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
What happens as your computer is executing one or more processes?

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

Machine code:
```
0111010000011000
100011010000010000000010
1000100111000010
11000001111111010000011111
```

Computer system:

Memory & data
Integers & floats
x86 assembly
Procedures & stacks
Executables
Arrays & structs
Processes
Virtual memory
Memory & caches
Memory allocation
Java vs. C

OS:
- Windows 10
- OS X Yosemite
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**
  ▪ https://courses.cs.washington.edu/courses/cse351/20su/syllabus/

❖ Binary
Bookmarks

❖ Course Website:  https://cs.uw.edu/351
  ▪ Schedule, policies, materials, videos, assignments, etc.
❖ Discussion:
  https://piazza.com/washington/summer2020/cse351
  ▪ Announcements made here
  ▪ Ask and answer questions – staff will monitor and contribute
❖ Gradescope:  https://www.gradescope.com/courses/140414
  ▪ Assignment submissions
❖ Canvas:  https://canvas.uw.edu/courses/1381167
  ▪ Few Quizzes, Zoom Links, and Gradebook
❖ Poll Everywhere:  http://pollev.com/pbjones
  ▪ In-lecture voting
Recommended 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)
    - [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)
  - Well done textbook
    - Students have found useful in 351 and beyond
    - Recommended lecture readings on website

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

❖ Lectures (26)
  ▪ Via Zoom, meant to introduce the concepts
  ▪ Slides posted before, recordings/ink posted after

❖ Sections (9)
  ▪ Via Zoom, short review then mainly group work
  ▪ Not recorded, but materials/helpful videos posted after

❖ Office Hours
  ▪ Via Zoom, schedule on the course calendar. Not recorded.
  ▪ Come ask questions! (course material or others)
  ▪ If things are busy we will use a queue for organization.
  https://tinyurl.com/351-queue
Course Components:

- Pre-quarter and Mid-quarter surveys (on Canvas)
  - Meant to check in and get to know you better
- Online Homework (23)
  - Solidify concept understanding, submitted via Gradescope
- Labs (6)
  - In depth applications/investigations of course material
  - Specs on website, submitted via Gradescope
- Unit Summaries (3)
  - Summaries/reflections on course material
  - Spec on website, submitted via Gradescope
- Can use up to 7 late days on labs and unit summaries (see syllabus for more details)
Grading:

- **Homework:** 25% total
  - Autograded; unlimited submission attempts
  - *Group work encouraged*

- **Labs:** 50% total
  - Graded by TAs; last submission graded
  - *Individual work only (high-level discussions OK)*

- **Unit Summaries:** 15% total
  - Meant to replace the review, summarizing, and reflecting that studying for exams provides. More info on these later.
  - *Individual work only (high-level discussions OK)*

- **Participation:** 10%
Lab Collaboration and Academic Integrity

- All submissions are expected to be yours and yours alone
- You are encouraged to discuss your assignments with other students (*high level 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
Course Environment and Culture

❖ It should go without saying we value your physical and mental health above the course material

❖ Simple course rules:
  ▪ Respect one another
  ▪ Ask questions
  ▪ Have fun!

❖ If at any point you feel uncomfortable, disrespected, excluded, etc. by any staff member or students, please report the incident so we can address the issue and maintain a supportive and inclusive environment
  ▪ Contact: staff (direct or anonymous), CSE undergraduate advising, UW Office of the Ombud
Lecture Polling

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

- Multiple choice question during lecture
  - 1 minute to decide on your own
  - 2-4 minutes in pairs to reach consensus
  - Learn through discussion & teaching

- Vote using **Poll Everywhere**
  - Use website ([https://www.polleverywhere.com](https://www.polleverywhere.com)) or app
  - Linked to your UWNetID
Lecture Polling Credit

- You receive participation credit for answering poll everywhere questions
  - 1 point for any answer, .05 extra points for correct answer
  - 80% of available points is full credit for participation

- Can makeup lecture credit with Canvas quiz
  - Don’t submit these if you answered in lecture
  - Released after lecture, due 10:30am before next lecture
  - Must include explanation to simulate in-lecture discussion

- First week does **not** count for credit
Some fun topics that we will touch on

- Which of the following seems the most interesting to you? (vote at [http://pollev.com/pbjones](http://pollev.com/pbjones))
  
a) What are some ways in which hackers exploit your machine?
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) How is your computer able to seemingly run hundreds of applications all at once?
Tips for Success in 351

- Attend all lectures and sections
  - Avoid devices during lecture except for Poll Everywhere
- Review the slides and/or readings before and after lecture
- 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
- Start assignments early
- Don’t be afraid to ask questions
- Give us feedback on how things are going this quarter
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; read posts
▪ Log in to Poll Everywhere
▪ **Get your machine set up for this class (VM or attu) as soon as possible**
▪ Make sure you’re also enrolled in CSE391!
  • TOMORROW, Tuesday 10:50-11:50am
  • [https://courses.cs.washington.edu/courses/cse391/20su/](https://courses.cs.washington.edu/courses/cse391/20su/)

❖ Assignments – Some this week!

▪ Pre-Course Survey (Canvas), hw0 due Wednesday (6/24) – 11:59pm
▪ hw1 due Friday (6/26) – 10:30am
▪ Lab 0 due Friday (6/26) – 11:59pm
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 $7061_8$ in base 10?
  - $7061_8 = (7 \times 8^3) + (0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 3633_{10}$
Warmup Question

❖ What is $34_8$ in base 10?
  ▪ No voting for this question

A. $32_{10}$
B. $34_{10}$
C. $7_{10}$
D. $28_{10}$
E. $35_{10}$
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?
   ▪ \(0xA5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}\)
Polling 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)

- Vote at http://pollev.com/pbjones
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!
  - No voting for this question
Converting from Decimal to Binary

- Given a decimal number N:
  1. List increasing powers of 2 from right to left until $\geq N$
  2. 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></th>
<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></td>
<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:
  1. List increasing powers of B from right to left until \( \geq N \)
  2. 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></th>
<th>16²=256</th>
<th>16¹=16</th>
<th>16⁰=1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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 → 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>
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 → 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 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 → 0xFFFFFFFF, Deep Pink → 0xFF1493
Binary Encoding – Characters/Text

❖ ASCII Encoding (www.asciitable.com)

▪ American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>04</td>
<td>#32</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>#64</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>#33</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>#65</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>02</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>#34</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>#66</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>03</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#35</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>#67</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>04</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>#36</td>
<td>\</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>#68</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>05</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>#37</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>#69</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>06</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>#38</td>
<td>$</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>#70</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>07</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>#39</td>
<td></td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>#71</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>08</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>#40</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>#72</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>09</td>
<td>HT (vertical tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>#41</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>#73</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>10</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>#42</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>#74</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>11</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>#43</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>#75</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>12</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>#44</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>#76</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>13</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>#45</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>#77</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>14</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>#46</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>#78</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>15</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>#47</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>#79</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>01</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>#48</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>#80</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>02</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>#49</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>#81</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>03</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>#50</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>#82</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>04</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>#51</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>#83</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>05</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>#52</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>#84</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>06</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>#53</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>#85</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>07</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>#54</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>#86</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>08</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>#55</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>#87</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>09</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>#56</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>#88</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>0A</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>#57</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>#89</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>0B</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>40</td>
<td>072</td>
<td>#58</td>
<td>:</td>
<td>90</td>
<td>6A</td>
<td>132</td>
<td>#90</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>0C</td>
<td>ESC (escape)</td>
<td>59</td>
<td>41</td>
<td>073</td>
<td>#59</td>
<td>;</td>
<td>91</td>
<td>6B</td>
<td>133</td>
<td>#91</td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>0D</td>
<td>FS (file separator)</td>
<td>60</td>
<td>42</td>
<td>074</td>
<td>#60</td>
<td>&lt;</td>
<td>92</td>
<td>6C</td>
<td>134</td>
<td>#92</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>0E</td>
<td>GS (group separator)</td>
<td>61</td>
<td>43</td>
<td>075</td>
<td>#61</td>
<td>=</td>
<td>93</td>
<td>6D</td>
<td>135</td>
<td>#93</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>0F</td>
<td>RS (record separator)</td>
<td>62</td>
<td>44</td>
<td>076</td>
<td>#62</td>
<td>&gt;</td>
<td>94</td>
<td>6E</td>
<td>136</td>
<td>#94</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>00</td>
<td>US (unit separator)</td>
<td>63</td>
<td>45</td>
<td>077</td>
<td>#63</td>
<td>?</td>
<td>95</td>
<td>6F</td>
<td>137</td>
<td>#95</td>
<td>_</td>
</tr>
</tbody>
</table>

Source: 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 (try it!)
  ▪ 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

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