The Hardware/Software Interface
CSE 351 Spring 2020

Instructor:
Ruth Anderson

Teaching Assistants:
Alex Olshanskyy
Rehaan Bhimani
Callum Walker
Chin Yeoh
Diya Joy
Eric Fan
Edan Sneh
Jonathan Chen
Jeffery Tian
Millicent Li
Melissa Birchfield
Porter Jones
Joseph Schafer
Connie Wang
Eddy (Tianyi) Zhou

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

- **Instructor:** Ruth Anderson

- **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!

- ~220 students registered across 2 lecture sections

- 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
Welcome to Spring 2020!

- Thanks in advance for working with us to make this the best on-line experience we can!
- Help us figure out the best ways to handle:
  - Lecture
  - Office Hours (Schedule & 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!
- Feedback Survey for today: (please fill this out after lecture today)
  - [https://catalyst.uw.edu/webq/survey/rea2000/387634](https://catalyst.uw.edu/webq/survey/rea2000/387634)
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?
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)
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)

.assembler

1000001101111100001001001000001110000000000
0111010100001100
100010110100010001001000010100
1001101000010100011000000010
1000100110000100
1100001111111011000011111
1111011110111110000100100000001100
100100101000100001001000011000
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/91/37/23/91372375e2e6517f8af128aa b655e3b4.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

[Diagram showing a stick figure interacting with hardware through an interface]
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
**Roadmap**

How does your source code become something that your computer understands?

### 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
1000110100000100
1000100111000010
1100001111110100001111
```

### OS:

- Memory & data
- Integers & floats
- x86 assembly
- Procedures & stacks
- Executables
- Arrays & structs
- Memory & caches
- Processes
- Virtual memory
- Memory allocation

### Computer system:

- Windows 10
- OS X Yosemite

---

**How does your source code become something that your computer understands?**

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

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

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

4. **Machine code**
   - `0111010000011000
    1000110100000100
    1000100111000010
    1100001111110100001111`

5. **OS**
   - Memory & data
   - Integers & floats
   - x86 assembly
   - Procedures & stacks
   - Executables
   - Arrays & structs
   - Memory & caches
   - Processes
   - Virtual memory
   - Memory allocation

6. **Computer system**
   - Windows 10
   - OS X Yosemite
Roadmap

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

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:
```
011101010000011000
100011010000010000000010
1000110111000010
1100001011111010000001111
```

OS:
- Windows 10
- OS X Yosemite

Computer system:

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**
  - https://courses.cs.washington.edu/courses/cse351/20sp/syllabus/
- Binary
Bookmarks

- Course Website:  http://cs.uw.edu/351
  - Schedule, policies, materials, videos, assignments, etc.
- Discussion:  http://piazza.com/washington/spring2020/cse351
  - Announcements made here
  - Ask and answer questions – staff will monitor and contribute
- Gradescope:  https://www.gradescope.com/courses/106949
  - Assignment submissions
- Canvas:  https://canvas.uw.edu/courses/1371942
  - Gradebook
- Poll Everywhere:  http://pollev.com/rea
  - In-lecture voting
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 (North American) **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!
    - Lecture readings
    - 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)
Grading:

- **Homework**: 30% total
  - Autograded; unlimited submission attempts
  - *Group work encouraged*

- **Labs**: 45% total
  - Graded by TAs; last submission graded
  - *Individual work only*

- **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*

- **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 (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
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) 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/rea)

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 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
  - Avoid devices during lecture except for Poll Everywhere
- Do the textbook readings ahead of time
- 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! (EEs included)
    • TOMORROW, Tuesday 1:30-2:20pm
    • [https://courses.cs.washington.edu/courses/cse391/20sp/](https://courses.cs.washington.edu/courses/cse391/20sp/)

❖ Assignments – Nothing **Due** this week!
  ▪ Pre-Course Survey, hw0, hw1, hw2 due Monday (4/06) – 11:59pm
  ▪ Lab 0 due Tuesday (4/07) – 11:59pm
  ▪ hw3 due Wednesday (4/08) – 11am
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. $35_{10}$

$$3 \times 8^1 + 4 \times 8^0$$

$$= 24 + 4$$
Binary and Hexadecimal

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

- **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: \(16_{10} = 10_{16} = \text{ox}10\)
  - **Example**: What is \(\text{ox}A5\) in base 10?
    - \(\text{ox}A5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}\)
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/rea
Converting to Base 10

- Can convert from any base \textit{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 (\textit{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 $(\text{power of 2}) \leq N$?
     - 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>$2^4=16$</th>
<th>$2^3=8$</th>
<th>$2^2=4$</th>
<th>$2^1=2$</th>
<th>$2^0=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{array}{r}
13 - 8 = 5 \\
5 - 4 = 1 \\
1 - 1 = 0 \\
0 - 0 = 0 \\
\end{array}
\]
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

  \[
  \begin{array}{c|c|c}
  16^2=256 & 16^1=16 & 16^0=1 \\
  \hline
  0 & 10 & 5 \\
  \end{array}
  \]

  $0 \times A5$
## 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 0b00101101
    - 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? 15
  - Pad: \(0b\ 0100\ 1101\ 1010\ 1101\)
  - Substitute: \(0\times4DAD\)
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 \texttt{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>Decimal</th>
<th>Hex</th>
<th>Octal</th>
<th>HTML</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 000</td>
<td>0x32</td>
<td>040</td>
<td>$\text{&amp;32}$</td>
<td>Space</td>
</tr>
<tr>
<td>1 001</td>
<td>0x96</td>
<td>0141</td>
<td>$\text{&amp;96}$</td>
<td>(</td>
</tr>
<tr>
<td>2 002</td>
<td>0x65</td>
<td>0102</td>
<td>$\text{&amp;65}$</td>
<td>)</td>
</tr>
<tr>
<td>3 003</td>
<td>0x77</td>
<td>0103</td>
<td>$\text{&amp;77}$</td>
<td>:</td>
</tr>
<tr>
<td>4 004</td>
<td>0x6b</td>
<td>0104</td>
<td>$\text{&amp;6b}$</td>
<td>;</td>
</tr>
<tr>
<td>5 005</td>
<td>0x6c</td>
<td>0105</td>
<td>$\text{&amp;6c}$</td>
<td>&lt;</td>
</tr>
<tr>
<td>6 006</td>
<td>0x6d</td>
<td>0106</td>
<td>$\text{&amp;6d}$</td>
<td>=</td>
</tr>
<tr>
<td>7 007</td>
<td>0x6e</td>
<td>0107</td>
<td>$\text{&amp;6e}$</td>
<td>&gt;</td>
</tr>
<tr>
<td>8 100</td>
<td>0x6f</td>
<td>0108</td>
<td>$\text{&amp;6f}$</td>
<td>?</td>
</tr>
<tr>
<td>9 101</td>
<td>0x70</td>
<td>0109</td>
<td>$\text{&amp;70}$</td>
<td>`</td>
</tr>
<tr>
<td>A 102</td>
<td>0x71</td>
<td>0110</td>
<td>$\text{&amp;71}$</td>
<td>a</td>
</tr>
<tr>
<td>B 103</td>
<td>0x72</td>
<td>0111</td>
<td>$\text{&amp;72}$</td>
<td>b</td>
</tr>
<tr>
<td>C 104</td>
<td>0x73</td>
<td>0112</td>
<td>$\text{&amp;73}$</td>
<td>c</td>
</tr>
<tr>
<td>D 105</td>
<td>0x74</td>
<td>0113</td>
<td>$\text{&amp;74}$</td>
<td>d</td>
</tr>
<tr>
<td>E 106</td>
<td>0x75</td>
<td>0114</td>
<td>$\text{&amp;75}$</td>
<td>e</td>
</tr>
<tr>
<td>F 107</td>
<td>0x76</td>
<td>0115</td>
<td>$\text{&amp;76}$</td>
<td>f</td>
</tr>
<tr>
<td>10 020</td>
<td>0x77</td>
<td>0116</td>
<td>$\text{&amp;77}$</td>
<td>g</td>
</tr>
<tr>
<td>11 021</td>
<td>0x78</td>
<td>0117</td>
<td>$\text{&amp;78}$</td>
<td>h</td>
</tr>
<tr>
<td>12 022</td>
<td>0x79</td>
<td>0118</td>
<td>$\text{&amp;79}$</td>
<td>i</td>
</tr>
<tr>
<td>13 023</td>
<td>0x7a</td>
<td>0119</td>
<td>$\text{&amp;7a}$</td>
<td>j</td>
</tr>
<tr>
<td>14 024</td>
<td>0x7b</td>
<td>0120</td>
<td>$\text{&amp;7b}$</td>
<td>k</td>
</tr>
<tr>
<td>15 025</td>
<td>0x7c</td>
<td>0121</td>
<td>$\text{&amp;7c}$</td>
<td>l</td>
</tr>
<tr>
<td>16 026</td>
<td>0x7d</td>
<td>0122</td>
<td>$\text{&amp;7d}$</td>
<td>m</td>
</tr>
<tr>
<td>17 027</td>
<td>0x7e</td>
<td>0123</td>
<td>$\text{&amp;7e}$</td>
<td>n</td>
</tr>
<tr>
<td>18 028</td>
<td>0x7f</td>
<td>0124</td>
<td>$\text{&amp;7f}$</td>
<td>o</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 (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