Decimal Numbers: Base 10

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Example:

\[ 3271 = (3 \times 10^3) + (2 \times 10^2) + (7 \times 10^1) + (1 \times 10^0) \]
Numbers: positional notation

- Number Base $B \Rightarrow B$ symbols per digit:
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Base 2 (Binary): 0, 1

- Number representation:
  - $d_{31}d_{30} \ldots d_1d_0$ is a 32 digit number
  - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + \ldots + d_1 \times B^{1} + d_0 \times B^{0}$

- Binary: 0, 1 (In binary digits called “bits”)
  - $0b11010 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - $= 16 + 8 + 2$
  - $= 26$

  #s often written 0b…
  - Here 5 digit binary # turns into a 2 digit decimal #
  - Can we find a base that converts to binary easily?
Hexadecimal Numbers: Base 16

- Hexadecimal:
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Normal digits + 6 more from the alphabet
  - In C, written as 0x… (e.g., 0xFAB5)

- Conversion: Binary ⇔ Hex
  - 1 hex digit represents 16 decimal values
  - 4 binary digits represent 16 decimal values
  ⇒ 1 hex digit replaces 4 binary digits

- Example:
  - 1010 1100 0011 (binary) = 0x_____ ?
Decimal vs. Hexadecimal vs. Binary

Examples:

1010 1100 0011 (binary) = 0xAC3

10111 (binary) = 0001 0111 (binary) = 0x17

0x3F9 = 11 1111 1001 (binary)

How do we convert between hex and Decimal?

MEMORIZE!
What to do with representations of numbers?

• Just what we do with numbers!
  • Add them
  • Subtract them
  • Multiply them
  • Divide them
  • Compare them

• Example: $10 + 7 = 17$
  • …so simple to add in binary that we can build circuits to do it!
  • subtraction just as you would in decimal
  • Comparison: How do you tell if $X > Y$?
Which base do we use?

- **Decimal**: great for humans, especially when doing arithmetic

- **Hex**: if human looking at long strings of binary numbers, its much easier to convert to hex and look 4 bits/symbol
  - Terrible for arithmetic on paper

- **Binary**: what computers use; you will learn how computers do +, -, *, /
  - To a computer, numbers always binary
  - Regardless of how number is written:
    \[32_{\text{ten}} = 32_{10} = 0x20 = 100000_{2} = 0b100000\]
  - Use subscripts “ten”, “hex”, “two” in book, slides when might be confusing
BIG IDEA: Bits can represent anything!!

- **Characters?**
  - 26 letters ⇒ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation ⇒ 7 bits (in 8) (“ASCII”)
  - standard code to cover all the world’s languages ⇒ 16 bits (“Unicode”)

- **Logical values?**
  - 0 ⇒ False, 1 ⇒ True

- **colors?** Ex: **Red (00)**  **Green (01)**  **Blue (11)**

- **locations / addresses? commands?**

- **MEMORIZE:** N bits ⇒ at most $2^N$ things
How to Represent Negative Numbers?

• So far, **unsigned numbers**

• Obvious solution: define leftmost bit to be sign!
  • $0 \Rightarrow +$, $1 \Rightarrow -$  
  • Rest of bits can be numerical value of number

• Representation called **sign and magnitude**

• MIPS uses 32-bit integers. $+1_{\text{ten}}$ would be: 
  \[0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\]

• And $-1_{\text{ten}}$ in sign and magnitude would be: 
  \[1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\]
Shortcomings of sign and magnitude?

• Arithmetic circuit complicated
  • Special steps depending whether signs are the same or not

• Also, two zeros
  • $0x00000000 = +0_{\text{ten}}$
  • $0x80000000 = -0_{\text{ten}}$
  • What would two 0s mean for programming?

• Therefore sign and magnitude abandoned
Administrivia

• Look at class website often!

• Reading for next Mon: K&R Ch1-4
  • Reading quiz www up soon, check email

• Unlike 61B, labs need to be completed in that lab session unless “officially extended” by your TA

• We will DROP your lowest HW, Lab!

• Homework #1 up now, due next Wed @ 11:59pm

• Homework #2 up soon, due following Mon @ 11:59pm
Another try: complement the bits

• Example: \( 7_{10} = 00111_2 \) \(-7_{10} = 11000_2 \)

• Called **One’s Complement**

• Note: positive numbers have leading 0s, negative numbers have leading 1s.

\[
\begin{array}{ccccccc}
00000 & 00001 & & \cdots & 01111 \\
10000 & \cdots & 11110 & 11111 \\
\end{array}
\]

• What is \(-00000\)? Answer: 11111

• How many positive numbers in N bits?

• How many negative ones?
Shortcomings of One’s complement?

- Arithmetic still a somewhat complicated.
- Still two zeros
  - $0x00000000 = +0_{\text{ten}}$
  - $0xFFFFFFFF = -0_{\text{ten}}$
- Although used for awhile on some computer products, one’s complement was eventually abandoned because another solution was better.
Standard Negative Number Representation

• What is result for unsigned numbers if tried to subtract large number from a small one?
  • Would try to borrow from string of leading 0s, so result would have a string of leading 1s
    - $3 - 4 \Rightarrow 00...0011 - 00...0100 = 11...1111$
  • With no obvious better alternative, pick representation that made the hardware simple
  • As with sign and magnitude, leading 0s $\Rightarrow$ positive, leading 1s $\Rightarrow$ negative
    - $000000...xxx$ is $\geq 0$, $111111...xxx$ is $< 0$
    - except $1...1111$ is -1, not -0 (as in sign & mag.)

• This representation is Two’s Complement
2’s Complement Number “line”: $N = 5$

- $2^{N-1}$ non-negatives
- $2^{N-1}$ negatives
- one zero
- how many positives?
Two’s Complement for N=32

<table>
<thead>
<tr>
<th>Binary</th>
<th>Two’s Complement</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 ... 0000 0000 0000 0000₀two</td>
<td>0₀ten</td>
<td></td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0001₀two</td>
<td>1₀ten</td>
<td></td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0010₀two</td>
<td>2₀ten</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1101₀two</td>
<td>2,147,483,645₀ten</td>
<td></td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1110₀two</td>
<td>2,147,483,646₀ten</td>
<td></td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1111₀two</td>
<td>2,147,483,647₀ten</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0000₀two</td>
<td>–2,147,483,648₀ten</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0001₀two</td>
<td>–2,147,483,647₀ten</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0010₀two</td>
<td>–2,147,483,646₀ten</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1101₀two</td>
<td>–3₀ten</td>
<td></td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1110₀two</td>
<td>–2₀ten</td>
<td></td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1111₀two</td>
<td>–1₀ten</td>
<td></td>
</tr>
</tbody>
</table>

- One zero; 1st bit called **sign bit**
- 1 “extra” negative: no positive 2,147,483,648₀ten
Two’s Complement Formula

• Can represent positive and negative numbers in terms of the bit value times a power of 2:
  \[ d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0 \]

• Example: \(1101_{\text{two}}\)
  \[= 1 \times -(2^3) + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\]
  \[= -2^3 + 2^2 + 0 + 2^0\]
  \[= -8 + 4 + 0 + 1\]
  \[= -8 + 5\]
  \[= -3_{\text{ten}}\]
Two’s Complement shortcut: Negation

- Change every 0 to 1 and 1 to 0 (invert or complement), then add 1 to the result

- Proof: Sum of number and its (one’s) complement must be $111...111_{\text{two}}$
  
  However, $111...111_{\text{two}} = -1_{\text{ten}}$

  Let $x' \Rightarrow$ one’s complement representation of $x$

  Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x$

- Example: -3 to +3 to -3

<table>
<thead>
<tr>
<th>$x$</th>
<th>1111 1111 1111 1111 1111 1111 1111 1101</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'$</td>
<td>0000 0000 0000 0000 0000 0000 0000 0010</td>
</tr>
<tr>
<td>$+1$</td>
<td>0000 0000 0000 0000 0000 0000 0000 0011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$()'</th>
<th>1111 1111 1111 1111 1111 1111 1111 1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1'$</td>
<td>1111 1111 1111 1111 1111 1111 1111 1101</td>
</tr>
</tbody>
</table>

You should be able to do this in your head...
Two’s comp. shortcut: Sign extension

- Convert 2’s complement number rep. using n bits to more than n bits

- Simply replicate the most significant bit (sign bit) of smaller to fill new bits
  - 2’s comp. positive number has infinite 0s
  - 2’s comp. negative number has infinite 1s

- Binary representation hides leading bits; sign extension restores some of them
  - 16-bit $-4_{\text{ten}}$ to 32-bit:
What if too big?

- Binary bit patterns above are simply *representatives* of numbers. Strictly speaking they are called “numerals”.

- Numbers really have an $\infty$ number of digits
  - with almost all being same (00...0 or 11...1) except for a few of the rightmost digits
  - Just don’t normally show leading digits

- If result of add (or -, *, /) cannot be represented by these rightmost HW bits, *overflow* is said to have occurred.
Peer Instruction Question

A. $X > Y$ (if signed)

B. $X > Y$ (if unsigned)

C. An encoding for Babylonians could have $2^N$ non-zero numbers w/N bits!
And in Conclusion...

• We represent “things” in computers as particular bit patterns: \( N \text{ bits } \Rightarrow 2^N \)

• Decimal for human calculations, binary for computers, hex to write binary more easily

• 1’s complement - mostly abandoned

• 2’s complement universal in computing: cannot avoid, so learn

• Overflow: numbers \( \infty \); computers finite, errors!
Bonus Slides

• Peer instruction let’s us skip example slides since you are expected to read book and lecture notes beforehand, but we include them for your review

• Slides shown in logical sequence order
BONUS: Numbers represented in memory

- Memory is a place to store bits

- A *word* is a fixed number of bits (e.g., 32) at an address

- **Addresses** are naturally represented as unsigned numbers in C
BONUS: Signed vs. Unsigned Variables

• Java just declares integers `int`
  • Uses two’s complement

• C has declaration `int` also
  • Declares variable as a signed integer
  • Uses two’s complement

• Also, C declaration `unsigned int`
  • Declares a unsigned integer
  • Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit