Machine Representation/Numbers Lecture 3

CS 61C Machines Structures
Fall 00
David Patterson
U.C. Berkeley

http://www-inst.eecs.berkeley.edu/~cs61c/

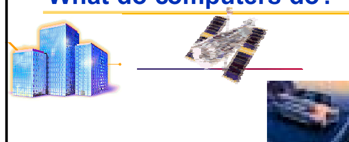
From last time: C v. Java

- C Designed for writing systems code, device drivers
- C is an efficient language, with little protection
  - Array bounds not checked
  - Variables not automatically initialized
- C v. Java: pointers and explicit memory allocation and deallocation
  - No garbage collection
  - Leads to memory leaks, funny pointers
  - Structure declaration does not allocate memory; use `malloc()` and `free()`

Overview

- Recap: C v. Java
- Computer representation of “things”
- Unsigned Numbers
- Administrivia
- Free Food 5PM Thursday, Sept. 7
- Computers at Work
- Signed Numbers: search for a good representation
- Shortcuts
- In Conclusion

What do computers do?



- Computers manipulate representations of things!
- What can you represent with N bits?
  - 2^N things!
- Which things?
  - Numbers! Characters! Pixels! Dollars! Position! Instructions! ...
  - Depends on what operations you do on them

Decimal Numbers: Base 10

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

3271 =
$(3x10^3) + (2x10^2) + (7x10^1) + (1x10^0)$

Numbers: positional notation

- Number Base B => B symbols per digit:
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    Base 2 (Binary): 0, 1
- Number representation:
  - $d_{31}d_{30}$ ... $d_2d_1d_0$ is a 32 digit number
  - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + ... + d_2 \times B^2 + d_1 \times B^1 + d_0 \times B^0$
- Binary: 0,1
  - $1011010 = 1x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 1x2 + 0x1 = 64 + 16 + 8 + 2 = 90$
  - Notice that 7 digit binary number turns into a 2 digit decimal number
  - 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: picked alphabet
- Conversion: Binary <-> Hex
  - 1 hex digit represents 16 decimal values
  - 4 binary digits represent 16 decimal values
  - => 1 hex digit replaces 4 binary digits
- Examples:
  - 1010 1100 0101 (binary) = ? (hex)
  - 10111 (binary) = 0001 0111 (binary) = ?
  - 3F9(hex) = ? (binary)

cs61c-f00 L3 9/6 7

Decimal vs. Hexadecimal vs.Binary

•Examples:

•1010 1100 0101 (binary) = ? (hex)

•10111 (binary) = 0001 0111 (binary) = ? (hex)

•3F9(hex) = ? (binary)

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

cs61c-f00 L3 9/6 8

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

```
        1   1
    1   0   1   0
+   0   1   1   1
------------------------
1   0   0   0   1
```

  - so simple to add in binary that we can build circuits to do it
  - subtraction also just as you would in decimal

cs61c-f00 L3 9/6 9

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; just say no
- Binary: what computers use; you learn how computers do +,-,*,/
  - To a computer, numbers always binary
  - Doesn't matter base in C, just the value: 32_{10} == 0x20 == 100000_2
  - Use subscripts "ten", "hex", "two" in book, slides when might be confusing

cs61c-f00 L3 9/6 10

Administrivia

- Grading: fixed scale, not on a curve
- To try to switch sections - email request to cs61c
- Viewing lectures again: tapes in 205 McLaughlin
- Read web page: Intro, FAQ, Schedule
  `www-inst.eecs.berkeley.edu/~cs61c`
  - TA assignments, Office Hours
  - Project 1 due Friday by Midnight

cs61c-f00 L3 9/6 11

Administrivia

- Tu/Th section 5-6PM; 18/118
- "Mark Chew" is most recent TA
- He quit, so lab/discussion in canceled

cs61c-f00 L3 9/6 12

Free Food 5PM Thursday, Sept. 7

- "The Importance of Graduate School"
  - Professor Katherine Yelick, UC Berkeley (Moderator)
  - Professor Mary Gray Baker, Stanford University
  - Dr. Serap Savari, Lucent Technology
  - Kris Hildrum, CS Current Graduate Student
  - 5:30 p.m. PANEL DISCUSSION, Hewlett-Packard Auditorium, 306 SODA
- 5:00 p.m. REFRESHMENTS in the Hall, Fourth Floor, Soda Hall

Bicycle Computer (Embedded)

- P. Brain
  - wireless heart monitor strap
  - record 5 measures: speed, time, current distance, elevation and heart rate
  - Every 10 to 60 sec.
  - 8KB data => 33 hours
  - Stores information so can be uploaded through a serial port into PC to be analyzed

Limits of Computer Numbers

- Bits can represent anything!
- Characters?
  - 26 letter => 5 bits
  - upper/lower case + punctuation => 7 bits (in 8)
  - rest of the world's languages => 16 bits (unicode)
- Logical values?
  - 0 -> False, 1 => True
- colors ?
- locations / addresses? commands?
- but N bits => only 2^N things

Comparison

- How do you tell if X > Y ?
- See if X - Y > 0

How to Represent Negative Numbers?

- So far, unsigned numbers
- Obvious solution: define leftmost bit to be sign!
  - 0 => +, 1 => -
  - Rest of bits can be numerical value of number
- Representation called sign and magnitude
- MIPS uses 32-bit integers. $+1_{ten}$ would be:

0000 0000 0000 0000 0000 0000 0000 0001

- And - 1_{ten} in sign and magnitude would be:

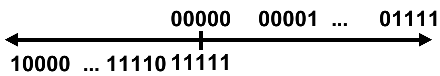
1000 0000 0000 0000 0000 0000 0000 0001

Shortcomings of sign and magnitude?

- Arithmetic circuit more complicated
  - Special steps depending whether signs are the same or not
- Also, Two zeros
  - 0x00000000 = $+0_{ten}$
  - 0x80000000 = -0_{ten}
  - What would it mean for programming?
- Sign and magnitude abandoned

Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called one's Complement
- Note: postive numbers have leading 0s, negative numbers have leadings 1s.



- What is -00000 ?
- How many positive numbers in N bits?
- How many negative ones?

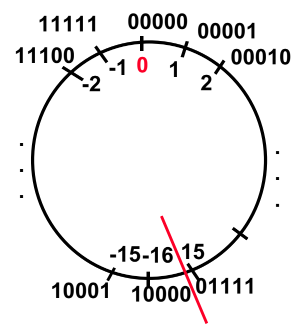
Shortcomings of ones complement?

- Arithmetic not too hard
- Still two zeros
  - `0x00000000` = $+0_{ten}$
  - `0xFFFFFFFF` = -0_{ten}
  - What would it mean for programming?
- One's complement eventually abandoned because another solution was better

Search for Negative Number Representation

- Obvious solution didn't work, find another
- 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
  - With no obvious better alternative, pick representation that made the hardware simple: leading 0s ⇒ positive, leading 1s ⇒ negative
  - 000000...xxx is >=0, 111111...xxx is < 0
- This representation called two's complement

2's Complement Number line



- 2^{N-1} non-negatives
- 2^{N-1} negatives
- one zero
- how many positives?
- comparison?
- overflow?

Two'sComplement

$0000 \ldots 0000\ 0000\ 0000\ 0000_{two}$ =	0_{ten}
$0000 \ldots 0000\ 0000\ 0000\ 0001_{two}$ =	1_{ten}
$0000 \ldots 0000\ 0000\ 0000\ 0010_{two}$ =	2_{ten}
. . .	
$0111 \ldots 1111\ 1111\ 1111\ 1101_{two}$ =	$2{,}147{,}483{,}645_{ten}$
$0111 \ldots 1111\ 1111\ 1111\ 1110_{two}$ =	$2{,}147{,}483{,}646_{ten}$
$0111 \ldots 1111\ 1111\ 1111\ 1111_{two}$ =	$2{,}147{,}483{,}647_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0000_{two}$ =	$-2{,}147{,}483{,}648_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0001_{two}$ =	$-2{,}147{,}483{,}647_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0010_{two}$ =	$-2{,}147{,}483{,}646_{ten}$
. . .	
$1111 \ldots 1111\ 1111\ 1111\ 1101_{two}$ =	-3_{ten}
$1111 \ldots 1111\ 1111\ 1111\ 1110_{two}$ =	-2_{ten}
$1111 \ldots 1111\ 1111\ 1111\ 1111_{two}$ =	-1_{ten}

- One zero, 1st bit => >=0 or <0, called sign bit
  - but one negative with no positive $-2{,}147{,}483{,}648_{ten}$

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
  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$
  $= 1 \times -2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
  $= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0$
  $= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$
  $= -4_{ten}$
- Note: need to specify width: we use 32 bits

Two's complement shortcut: Negation

- Invert every 0 to 1 and every 1 to 0, then add 1 to the result
  - Sum of number and its one's complement must be $111...111_{two}$
  - $111...111_{two} = -1_{ten}$
  - Let x' mean the inverted representation of x
  - Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x$
- Example: -4 to +4 to -4

x : $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$
x': $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two}$
+1: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{two}$
()': $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{two}$
+1: $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

cs61c-f00 L3 9/6 25

Signed vs. Unsigned Numbers

- C declaration `int`
  - Declares a signed number
  - Uses two's complement
- C declaration `unsigned int`
  - Declares a unsigned number
  - Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit

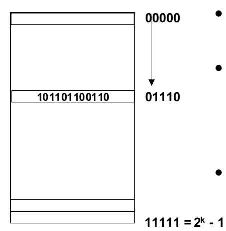
cs61c-f00 L3 9/6 26

Signed v. Unsigned Comparisons

- X = $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$
- Y = $0011\ 1011\ 1001\ 1010\ 1000\ 1010\ 0000\ 0000_{two}$
- Is X > Y?
  - unsigned: YES
  - signed: NO
- Converting to decimal to check
  - Signed comparison: $-4_{ten} < 1{,}000{,}000{,}000_{ten}$?
  - Unsigned comparison: $-4{,}294{,}967{,}292_{ten} < 1{,}000{,}000{,}000_{ten}$?

cs61c-f00 L3 9/6 27

Numbers are stored at addresses

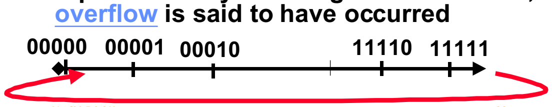


- Memory is a place to store bits
- A word is a fixed number of bits (eg, 32) at an address
  - also fixed no. of bits
- Addresses are naturally represented as unsigned numbers

cs61c-f00 L3 9/6 28

What if too big?

- Binary bit patterns above are simply representatives of numbers
- Numbers really have an infinite number of digits
  - with almost all being zero except for a few of the rightmost digits
  - Just don't normally show leading zeros
- If result of add (or -,*/) cannot be represented by these rightmost HW bits, overflow is said to have occurred



cs61c-f00 L3 9/6 29

Two's comp. shortcut: Sign extension

- Convert 2's complement number 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
  - Bit representation hides leading bits; sign extension restores some of them
  - 16-bit -4_{ten} to 32-bit:

$1111\ 1111\ 1111\ 1100_{two}$
$\boxed{1111\ 1111\ 1111\ 1111}\ 1111\ 1111\ 1111\ 1100_{two}$

cs61c-f00 L3 9/6 30

And in Conclusion...

- We represent "things" in computers as particular bit patterns: N bits =>2^N
  - numbers, characters, ... **(data)**
- Decimal for human calculations, binary to undertstand computers, hex to understand binary
- 2's complement universal in computing: cannot avoid, so learn
- Computer operations on the representation correspond to real operations on the real thing
- Overflow: numbers infinite but computers finite, so errors occur

cs61c-f00 L3 9/6 31