# CS 61c: Great Ideas in Computer Architecture

## Introduction and Number Representation

**Instructor:** Alan Christopher

June 23, 2014

# Introducing Your Instructor

- I'm not big on formalities, just call me Alan.
- **Upbringing:**
  - Born in Cleveland
  - Raised primarily in LA
- **Education:**
  - B.S. in EECS (Option IV)
  - B.A. in Applied Math (Probability Theory Cluster)
- **Teaching Experience:** Four semesters TAing 61c
- **Interests:**

# Outline

# What is CS 61c about?

- It's about the **hardware-software interface**

# What is CS 61c about?

- It's about the **hardware-software interface**
  - What is *actually happening* when programs execute?

# What is CS 61c about?

- It's about the **hardware-software interface**
  - What is *actually happening* when programs execute?
  - What does the programmer need to know to achieve the best possible performance?

# What is CS 61c about?

- It's about the **hardware-software interface**
  - What is *actually happening* when programs execute?
  - What does the programmer need to know to achieve the best possible performance?
- Using low-level programming languages (closer to the underlying hardware)

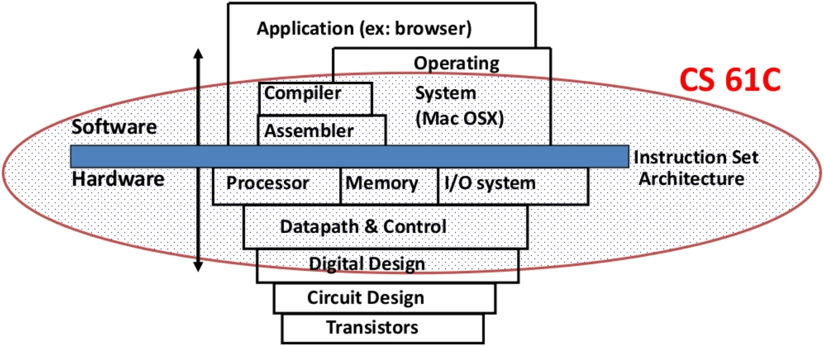
# What is CS 61c about?

- It's about the **hardware-software interface**
  - What is *actually happening* when programs execute?
  - What does the programmer need to know to achieve the best possible performance?
- Using low-level programming languages (closer to the underlying hardware)
  - Allows us to talk about key hardware features in higher-level terms.

# What is CS 61c about?

- It's about the **hardware-software interface**
  - What is *actually happening* when programs execute?
  - What does the programmer need to know to achieve the best possible performance?
- Using low-level programming languages (closer to the underlying hardware)
  - Allows us to talk about key hardware features in higher-level terms.
  - Allows programmers to harness underlying hardware for high performance.

# Old School 61c: Machine Structures



# New School 61c: Parallelism Too

**Software:**

- Parallel Requests

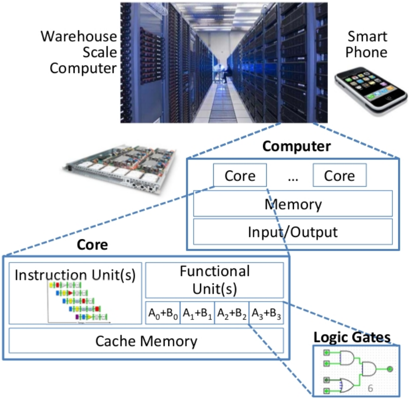
  Assigned to computer
  e.g. search "Garcia"
- Parallel Threads

  Assigned to core
  e.g. lookup, ads
- Parallel Instructions

  >1 instruction at a time
  e.g. pipelined instructions
- Hardware descriptions

  All gates functioning in parallel at same time

**Hardware:**

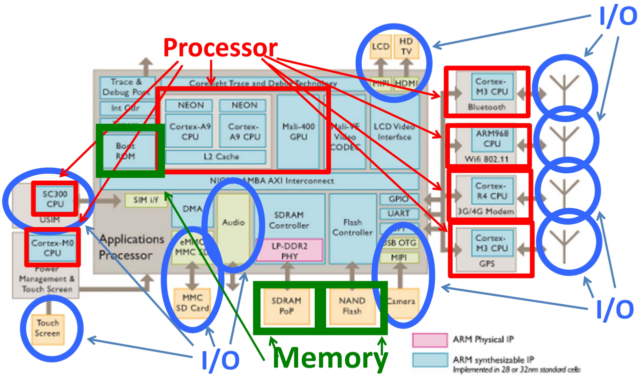


# Post-PC Era: Late 2000s - ???

Personal Mobile Devices (PMD):

**Enabling Tech:** Wireless networking, smartphones
**Big Players:** Apple, Nokia, . . .
**Cost:** ≈$500, **Target:** Consumers on the go
**Using:** Object C, Android OS, IOS

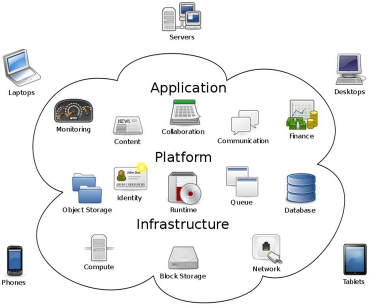
# iPhone Innards




You will learn about multiple processors, the memory hierarchy, and I/O in this course

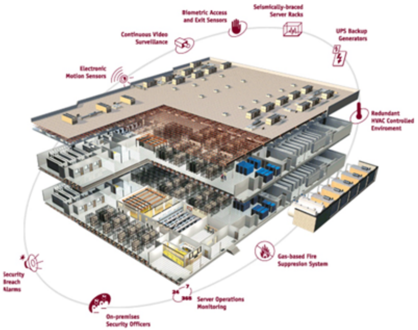
# Post-PC Era: Late 2000s - ???

Cloud Computing:



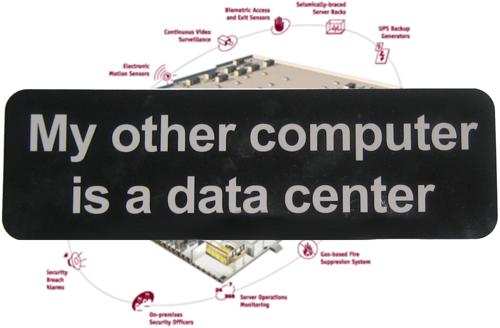

**Enabling Tech:** Local Area Networks, broadband Internet
**Big Players:** Amazon, Google, . . .
**Target:** Transient users or users who cannot afford high-end equipment

# Warehouse Scale Computers (WSC)




**Enabling Tech:** Local Area Networks, cheap servers
**Cost:** \$200M clusters + maintenance costs
**Target:** Internet services and PMDs
**Example Uses:** MapReduce, Web Search

# Warehouse Scale Computers (WSC)




**Enabling Tech:** Local Area Networks, cheap servers
**Cost:** \$200M clusters + maintenance costs
**Target:** Internet services and PMDs

# Six Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy

# Great Idea #1: Levels of Representation/Interpretation

High Level Language Program (e.g.C)

*Compiler*

Assembly Language Program (e.g. MIPS)

*Assembler*

Machine Language Program

*Machine Interpretation*

Hardware Architecture Description (e.g. block diagrams)

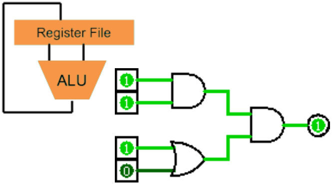
*Architecture Implementation*

Logic Circuit Description (Circuit Schematic Diagrams)

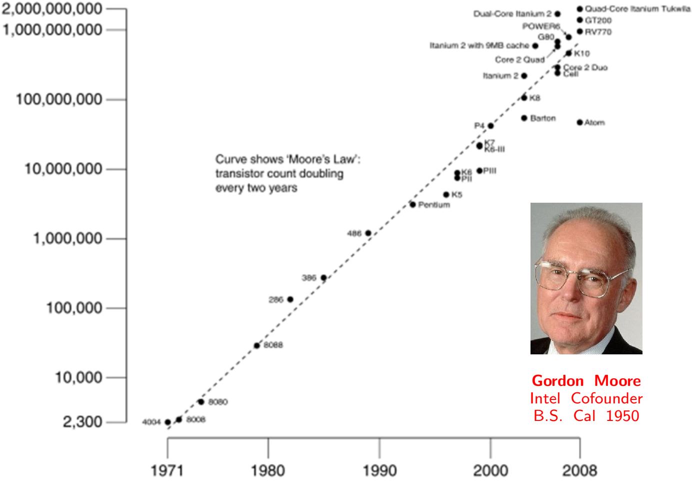
```
temp = v[k];
v[k] = v[k+1]
v[k+1] = temp;
```

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

```
1000 1100 0100 1000 0000 0000 0000 0000
1000 1100 0100 1001 0000 0000 0000 0100
1010 1100 0100 1001 0000 0000 0000 0000
1010 1100 0100 1000 0000 0000 0000 0100
```

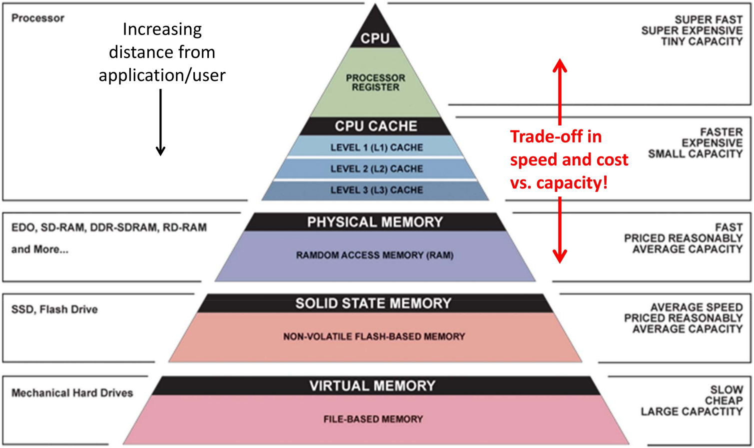


# Great Idea #2: Moore's Law

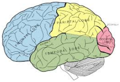



**Gordon Moore**
Intel Cofounder
B.S. Cal 1950

# Great Idea #3: Principle of Locality/Memory Hierarchy



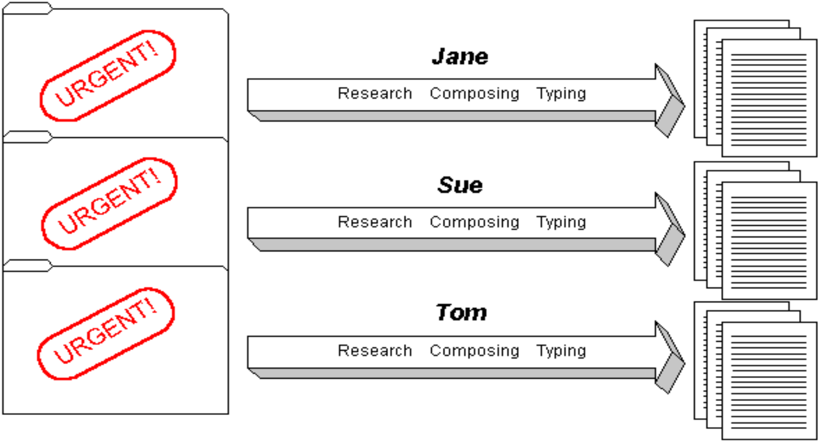
# Storage Latency Analogy: How Far Away is the Data?

| Access Time (ns) | Technology | Analagous Location | Analagous Time |
|---|---|---|---|
| 1 | Registers |  | 1 minute |
| 2 | On Chip Cache |  | 2 minutes |
| 10 | On Board Cache | | 10 minutes |

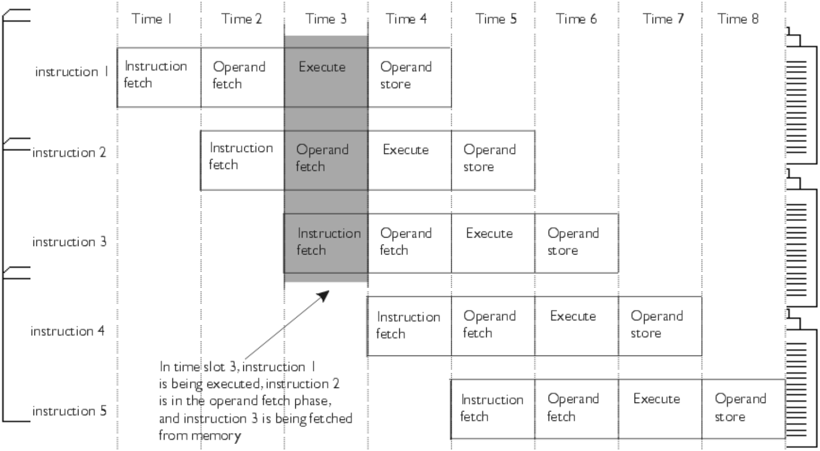
# Storage Latency Analogy: How Far Away is the Data?

| Access Time (ns) | Technology | Analagous Location | Analagous Time |
|---|---|---|---|
| 100 | DRAM |  | 1.5 hours |
| $10^6$ | Disk |  | 2 years |
| $10^9$ | Tape/Optical Robot |  | 2000 years |

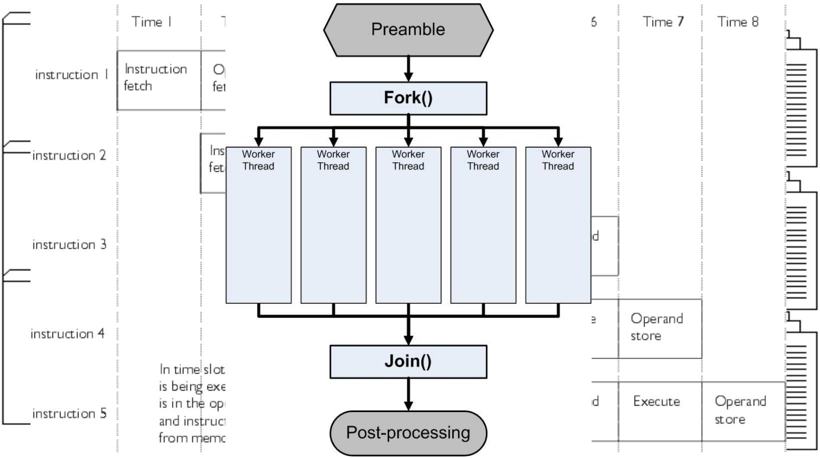
# Great Idea #4: Parallelism



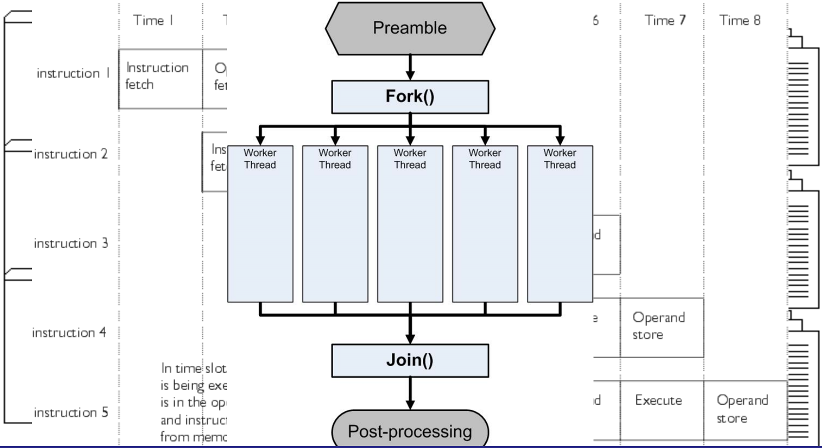
# Great Idea #4: Parallelism



# Great Idea #4: Parallelism



# Great Idea #4: Parallelism

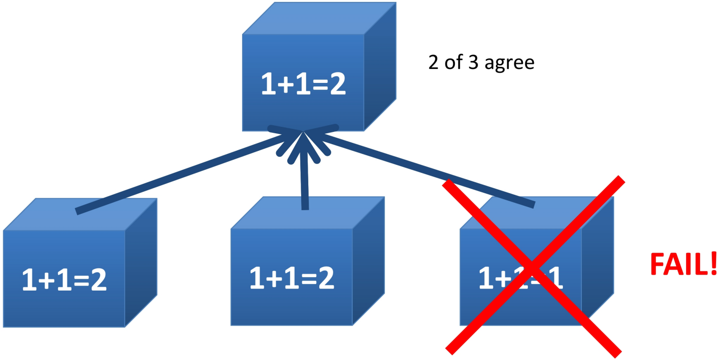


# Great Idea #5: Performance Measurement and Improvement

- Allows direct comparisons of architectures and quantification of improvements
- Most common measures are *time to finish* (latency) and *rate of execution* (throughput)
  - Latency includes both *setup* and *execution* time
- Match application to hardware to exploit:
  - Locality
  - parallelism
  - special hardware features

# Great Idea #6: Dependability via Redundancy

- Redundancy so that a single failing piece of doesn't compromise the whole system.



# Great Idea #6: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can 1 disk but not lose data (Redundant Arrays of Inexpensive Disks/RAID)
  - Redundant bits of memory so that can lose 1 bit but no data (Error Correcting Codes/ECC)

# Outline

# Course Information

This information, and more, is available on the course syllabus.

- **Website:** http://inst.eecs.berkeley.edu/~cs61c/su14
- **Instructor:** Alan Christopher
- **GSIs:** David Adams, Fred Hong, Hokeun Kim, Kevin Liston, Andrew Luo
- **Textbooks:** Average 15 pages of reading per week
  - Patterson & Hennessey, *Computer Organization and Design*, 5th Edition
  - Kernighan & Ritchie, *The C Programming Language*, 2nd Edition
  - Barroso & Holzle, *The Datacenter as a Computer*, 1st Edition
- **Piazza** (http://piazza.com/class#summer2014/cs61c) is the class forum
  - Announcements, general discussion, and clarifications happen there.

# Course Assignments and Grading

- **Homework** (5%) – 6 total, weighted equally
- **Labs** (6%) – 12 total, weighted equally
  - Done in partners
- **Projects** (24%) – 3 total, tentatively weighted equally
  1. LIFC Compiler
  2. Performance Optimization
  3. Processor Design
- **Midterm** (30%): Monday, July 21. Can be clobbered by final.
- **Final** (30%): Friday, August 15
- **EPA** (5%)

# EPA (Effort, Participation, Altruism)

- **Effort**
  - Go to office hours regularly
  - Attend all your discussions
  - Complete all assignments
- **Participation**
  - Ask and answer questions in lecture
  - Aks and answer questions in discussion
- **Altruism**
  - Help others in lab, discussion, and on Piazza
- EPA is calculated internally, and used to boost students on the borderline of grades.
  - Almost impossible to hurt your grade with EPA

# Peer Instruction

- Increase real-time learning in lecture, test understanding of concepts vs. details
  http://mazur.harvard.edu/education/pi_manual.php
- Multiple choice question at end of a "segment"
  - 1 minute to decide yourself
  - 2 minutes in pairs to reach consensus
  - Learn by teaching!
- Vote by flashing colored index card
  - Cup it in your hand if you don't want those around you to know what you voted (but make sure I can see it)
  - Or hold it out for all to see if you're feeling confident

**Question:** Which statement is **FALSE** about Great Ideas in Computer Architecture

(A) To offer a dependable system, you must use components that almost never fail

(B) The goal of a memory hierarchy is to look as fast as the most expensive memory and as big as the cheapest

(C) Moore's Law states that integrated circuits will fit twice as many transistors per chip $\approx$2 years

(D) Using different levels of represenation we can represent anything as 1s and 0s

# Late Policy – Slip Days

- Assignments are due at 23:59:59 (time-stamped)
- Everyone starts with 3 slip days
  - 1 slip day used for every day a project or hw is late (even by a second)
  - Slip days cannot be retroactively reassigned, so bear in mind how much different assignments are worth
- After all slip days are expended a 33% penalty accrues every day late
- Slip days are a 1 size fits all solution to extension requests, not a resource that students are entitled to.

# Policy on Independent work

- All assignments are individual work, unless explicitly stated otherwise
- You are encouraged to discuss ideas with other students, but what you hand in should be your work and your work alone
- It is **NOT** acceptable to copy solutions from other students
- It is **NOT** acceptable to take solutions from the internet.
- We have industrial strength tools for catching cheaters, and we will use them. You will be caught if you cheat, and the consequences will be severe.
  - Cheating is a serious crime, regardless of the size of the assignment. The penalties will be severe whether we catch you cheating on a project, or on one of the homeworks (which are worth practically nothing, and graded on effort).

## Comments on the Summer Variant

- Summer is *incredibly* frantic
  - Double the standard pace, in a course which already covers more material than is entirely reasonable
  - Less time to grok material
  - Falling behind just a little can be disastrous
    - If the course begins to overwhelm you, don't wait, contact me or your TA immediately
- No MapReduce project
- Starts deceptively slowly (first two weeks)

# Pedagogical Philosophy

- Real learning does not happen in lecture
  - Lecture's mostly just for the initial information dump
  - Learn by doing: labs, discussions, and assignments

# Pedagogical Philosophy

- Real learning does not happen in lecture
  - Lecture's mostly just for the initial information dump
  - Learn by doing: labs, discussions, and assignments
- You are all seriously capable individuals
  - I will not hesitate to throw you into the deep end of the pool

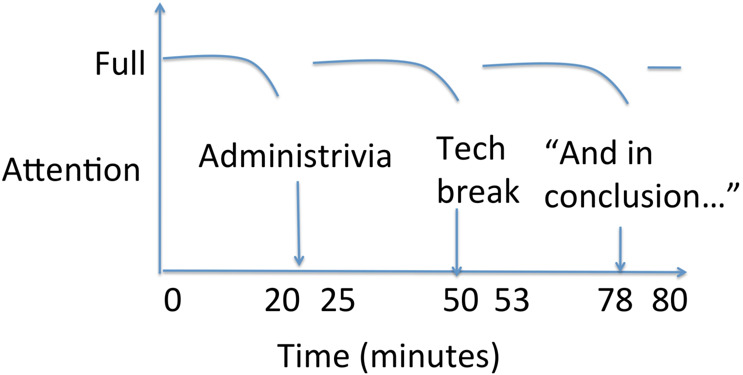
# Pedagogical Philosophy

- Real learning does not happen in lecture
  - Lecture's mostly just for the initial information dump
  - Learn by doing: labs, discussions, and assignments
- You are all seriously capable individuals
  - I will not hesitate to throw you into the deep end of the pool
- The best way to succeed is to fail, but quickly (i.e. before the exam)
  - There is no penalty for being wrong in lecture, in fact it helps me to know what's confusing students.
  - There is a penalty for being silently wrong in lecture though, as you'll probably take a hit on the exam for doing so.

# Pedagogical Philosophy

- Real learning does not happen in lecture
  - Lecture's mostly just for the initial information dump
  - Learn by doing: labs, discussions, and assignments
- You are all seriously capable individuals
  - I will not hesitate to throw you into the deep end of the pool
- The best way to succeed is to fail, but quickly (i.e. before the exam)
  - There is no penalty for being wrong in lecture, in fact it helps me to know what's confusing students.
  - There is a penalty for being silently wrong in lecture though, as you'll probably take a hit on the exam for doing so.
- Group study is incredibly valuable
  - The class is curved, but the effect of improving one person's score is negligible. Don't be afraid to collaborate
  - Plus, employers don't look all that closely at GPA, since a lot of universities have incredible grade inflation – it's about what you know and what you can do.

# Architecture of a Lecture



# Last Things...

- Discussions, labs, and OHs start immediately
  - Yes, that means today
  - Switching sections: if you find another 61c student willing to swap lab, talk to your TAs. There's no need to go through telebears
  - Attend whichever discussion(s) you want, so long as there's enough room to fit everyone
- HW0 is due next Tuesday in lab
- HW1 is due this Sunday

# Get to know your instructor

- Here are the rules
  - You say your name, your question for me, and your answer to that question.
  - Then I answer your question and the next person goes.

# Get to know your instructor

- Here are the rules
  - You say your name, your question for me, and your answer to that question.
  - Then I answer your question and the next person goes.
- Who's first?

# Outline

# Number Representation

- Numbers are an abstract concept
  - What we recognize as numbers (e.g. 23) are abitrary symbols, and are more properly called numerals.
- Inside a computer, everything is stored as a sequence of 1s and 0s (bits)
  - Actually, we build up and abstraction so that everything in a computer can be thought of as a sequence of bits.
- How does one represent numbers using only 1s and 0s?
  - Let's start with nonnegative integers.

# Radices

- **Key terminology:** *digit* ($d$) and *radix* ($r$). Another common term for radix is *base*.
- Value of $i$-th digit is $d \times r^i$, where $i$ starts at 0 and increases from right to left
  - $n$ digit numeral $d_{n-1}d_{n-2}\dots d_1 d_0$
  - value $= \sum_{i=0}^{n-1} d_i r^i = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \cdots + d_1 r + d_0$
- In radix $r$, each digit is one of $r$ possible symbols, with values in the range $[0, r)$.
- A numeral's radix is usually indicated either using a prefix or a subscript.

# Common Bases

- **Decimal** (10)
  - Symbols: 0,1,2,3,4,5,6,7,8,9
  - Notation: $9472_{\text{ten}} = 9472$
- **Binary** (2)
  - Symbols: 0,1
  - Notation: $1011110_{\text{two}} = \text{0b1011110}$
- **Hexadecimal** (16)
  - Symbols: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Notation: $\text{DEAD}_{\text{hex}} = \text{0xDEAD}$

| Decimal | Binary | Hex |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# Some Examples

- Examples (grab your calculators):
  - Apply the definition of unsigned numbers to 9472

# Some Examples

- Examples (grab your calculators):
  - Apply the definition of unsigned numbers to 9472

$$
\begin{aligned}
9472_{10} &= 9000 && + 400 && + 70 && + 2 \\
&= 9 \times 1000 && + 4 \times 100 && + 7 \times 10 && + 2 \times 1 \\
&= 9 \times 10^3 && + 4 \times 10^2 && + 7 \times 10^1 && + 2 \times 10^0
\end{aligned}
$$

  - Convert 9472 to hex

# Some Examples

- Examples (grab your calculators):
  - Apply the definition of unsigned numbers to 9472

$$
\begin{aligned}
9472_{10} &= 9000 &&+ 400 &&+ 70 &&+ 2 \\
&= 9 \times 1000 &&+ 4 \times 100 &&+ 7 \times 10 &&+ 2 \times 1 \\
&= 9 \times 10^3 &&+ 4 \times 10^2 &&+ 7 \times 10^1 &&+ 2 \times 10^0
\end{aligned}
$$

  - Convert 9472 to hex

$$
\begin{aligned}
9472_{10} &= 2 \times 16^3 + 5 \times 16^2 + 0 + 0 \\
&= \text{0x2500}
\end{aligned}
$$

  - Convert 0xA15 to binary

# Some Examples

- Examples (grab your calculators):
  - Apply the definition of unsigned numbers to 9472
    $$\begin{aligned} 9472_{10} &= 9000 + 400 + 70 + 2 \\ &= 9 \times 1000 + 4 \times 100 + 7 \times 10 + 2 \times 1 \\ &= 9 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 \end{aligned}$$
  - Convert 9472 to hex
    $$\begin{aligned} 9472_{10} &= 2 \times 16^3 + 5 \times 16^2 + 0 + 0 \\ &= \text{0x2500} \end{aligned}$$
  - Convert 0xA15 to binary
    0xA15 = 0b1010 0001 0101

# Bits can represent anything

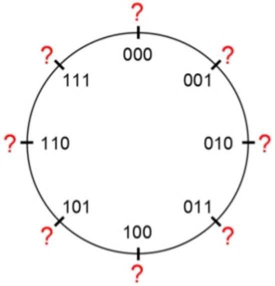
- $n$ digits in radix $r$ can represent at most $r^n$ things
  - Need to represent more things? Add more digits!
- Example: Logical values (1 bit) – 0 is False, 1 is True
- Example: Characters
  - 26 letters require at least 5 bits ($2^4 < 26 < 2^5$)
- Example: Students in this class (8 bits)
- For ease of reading, often group bits into *nybbles* (4 bits) or *bytes* (8 bits).

# What's in a numeral

- We are using binary bit patterns to *represent* numbers
  - These bit strings are more properly called numerals, and have no meaning until they are *interpretted*.
  - Is CAB a word (taxi), or a number (3243)?
  - Is 0xE0E0E0 a number, or a color (RGB)?
- The same bit pattern can mean many different things, depending on how it is interpretted. The context in which a numeral is encountered will usually tell you how to deal with it.

# Numbers in a Computer

- Numbers really have $\infty$ digits, but hardware can only store a finite number of them
  - Ignore leading zeroes
  - Leftmost bit is *most significant bit* (MSB)
  - Rightmost bit is *least significant bit* (LSB)
- "Circle" of 3-bit numerals:



# Unsigned Integers

Represent only non-negative (unsigned) integers:

$0000_2 = 0$ **Zero?**

$0001_2 = 1$

...

$0110_2 = 6$

$0111_2 = 7$

$1000_2 = 8$

$1001_2 = 9$

...

$1110_2 = 14$

$1111_2 = 15$

# Unsigned Integers

Represent only non-negative (unsigned) integers:

$$0000_2 = 0$$
$$0001_2 = 1$$
$$\ldots$$
$$0110_2 = 6$$
$$0111_2 = 7$$
$$1000_2 = 8$$
$$1001_2 = 9$$
$$\ldots$$
$$1110_2 = 14$$
$$1111_2 = 15$$

**Zero?**

$$0 \ldots 0_2 = 0$$

**Most negative number?**

# Unsigned Integers

Represent only non-negative (unsigned) integers:

$0000_2 = 0$
$0001_2 = 1$
...
$0110_2 = 6$
$0111_2 = 7$
$1000_2 = 8$
$1001_2 = 9$
...
$1110_2 = 14$
$1111_2 = 15$

**Zero?**

$0 \ldots 0_2 = 0$

**Most negative number?**

$0 \ldots 0_2 = 0$

**Most positive number?**

# Unsigned Integers

Represent only non-negative (unsigned) integers:

$0000_2 = 0$
$0001_2 = 1$
...
$0110_2 = 6$
$0111_2 = 7$
$1000_2 = 8$
$1001_2 = 9$
...
$1110_2 = 14$
$1111_2 = 15$

**Zero?**

$0 \ldots 0_2 = 0$

**Most negative number?**

$0 \ldots 0_2 = 0$

**Most positive number?**

$1 \ldots 1_2 = 2^n - 1$

**Increment Behavior?**

# Signed Integers

- $n$ bits can represent $2^n$ different things
  - Ideally, want to split that range evenly between positive and negative
- But how do we deal with zero?
- Can we encode signed integers in such a way as allows us to reuse the same hardware, regardless of signed-ness?

# First Pass: Sign and Magnitude

MSB, determines the sign, rest treated as unsigned (magnitude):

$0000_2 = +0$

$0001_2 = +1$

...

$0110_2 = +6$

$0111_2 = +7$

$1000_2 = -0$

$1001_2 = -1$

...

$1110_2 = -6$

$1111_2 = -7$

**Zero?**

# First Pass: Sign and Magnitude

MSB, determines the sign, rest treated as unsigned (magnitude):

$0000_2 = +0$
$0001_2 = +1$
$\ldots$
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -0$
$1001_2 = -1$
$\ldots$
$1110_2 = -6$
$1111_2 = -7$

**Zero?**

$0\ldots0_2 = +0$

$1\ldots0_2 = -0$

**Most negative number?**

# First Pass: Sign and Magnitude

MSB, determines the sign, rest treated as unsigned (magnitude):

$$0000_2 = +0$$
$$0001_2 = +1$$
$$\dots$$
$$0110_2 = +6$$
$$0111_2 = +7$$
$$1000_2 = -0$$
$$1001_2 = -1$$
$$\dots$$
$$1110_2 = -6$$
$$1111_2 = -7$$

**Zero?**

$$0\dots0_2 = +0$$
$$1\dots0_2 = -0$$

**Most negative number?**

$$1\dots1_2 = -(2^{n-1} - 1)$$

**Most positive number?**

# First Pass: Sign and Magnitude

MSB, determines the sign, rest treated as unsigned (magnitude):

$0000_2 = +0$
$0001_2 = +1$
$\ldots$
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -0$
$1001_2 = -1$
$\ldots$
$1110_2 = -6$
$1111_2 = -7$

**Zero?**

$0\ldots0_2 = +0$

$1\ldots0_2 = -0$

**Most negative number?**

$1\ldots1_2 = -(2^{n-1} - 1)$

**Most positive number?**

$01\ldots1_2 = 2^{n-1} - 1$

**Increment Behavior?**

# Second Pass: Bias

Like unsigned, but "shifted" so zero is in the middle:

$0000_2 = -7$

$0001_2 = -6$

...

$0110_2 = -1$

$0111_2 = 0$

$1000_2 = +1$

$1001_2 = +2$

...

$1110_2 = +7$

$1111_2 = +8$

**Zero?**

# Second Pass: Bias

Like unsigned, but "shifted" so zero is in the middle:

$0000_2 = -7$
$0001_2 = -6$
$\ldots$
$0110_2 = -1$
$0111_2 = 0$
$1000_2 = +1$
$1001_2 = +2$
$\ldots$
$1110_2 = +7$
$1111_2 = +8$

**Zero?**

$01\ldots1_2 = 0$

**Most negative number?**

# Second Pass: Bias

Like unsigned, but "shifted" so zero is in the middle:

$0000_2 = -7$
$0001_2 = -6$
...
$0110_2 = -1$
$0111_2 = 0$
$1000_2 = +1$
$1001_2 = +2$
...
$1110_2 = +7$
$1111_2 = +8$

**Zero?**

$01\ldots1_2 = 0$

**Most negative number?**

$0\ldots0_2 = -(2^{n-1} - 1)$

**Most positive number?**

# Second Pass: Bias

Like unsigned, but "shifted" so zero is in the middle:

$0000_2 = -7$
$0001_2 = -6$
$\dots$
$0110_2 = -1$
$0111_2 = 0$
$1000_2 = +1$
$1001_2 = +2$
$\dots$
$1110_2 = +7$
$1111_2 = +8$

**Zero?**

$01\dots1_2 = 0$

**Most negative number?**

$0\dots0_2 = -(2^{n-1} - 1)$

**Most positive number?**

$1\dots1_2 = 2^{n-1}$

**Increment Behavior?**

# A Crazy Idea: One's Complement

New negation procedure – flip *all* the bits:

$$0000_2 = +0$$
$$0001_2 = +1$$
$$\ldots$$
$$0110_2 = +6$$
$$0111_2 = +7$$
$$1000_2 = -7$$
$$1001_2 = -6$$
$$\ldots$$
$$1110_2 = -1$$
$$1111_2 = -0$$

**Zero?**

# A Crazy Idea: One's Complement

New negation procedure – flip *all* the bits:

$0000_2 = +0$
$0001_2 = +1$
$\ldots$
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -7$
$1001_2 = -6$
$\ldots$
$1110_2 = -1$
$1111_2 = -0$

**Zero?**

$0 \ldots 0_2 = +0$
$1 \ldots 1_2 = -0$

**Most negative number?**

# A Crazy Idea: One's Complement

New negation procedure – flip *all* the bits:

$$0000_2 = +0$$
$$0001_2 = +1$$
$$\dots$$
$$0110_2 = +6$$
$$0111_2 = +7$$
$$1000_2 = -7$$
$$1001_2 = -6$$
$$\dots$$
$$1110_2 = -1$$
$$1111_2 = -0$$

**Zero?**

$$0 \dots 0_2 = +0$$
$$1 \dots 1_2 = -0$$

**Most negative number?**

$$10 \dots 0_2 = -(2^{n-1} - 1)$$

**Most positive number?**

# A Crazy Idea: One's Complement

New negation procedure – flip *all* the bits:

$0000_2 = +0$
$0001_2 = +1$
...
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -7$
$1001_2 = -6$
...
$1110_2 = -1$
$1111_2 = -0$

**Zero?**

$0\ldots0_2 = +0$

$1\ldots1_2 = -0$

**Most negative number?**

$10\ldots0_2 = -(2^{n-1} - 1)$

**Most positive number?**

$01\ldots1_2 = 2^{n-1} - 1$

**Increment Behavior?**

# The Golden Child: Two's Complement

Like one's complement, but negative numbers shifted to the left by 1.

$0000_2 = +0$
$0001_2 = +1$
$\ldots$
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -8$
$1001_2 = -7$
$\ldots$
$1110_2 = -2$
$1111_2 = -1$

**Zero?**

# The Golden Child: Two's Complement

Like one's complement, but negative numbers shifted to the left by 1.

$$0000_2 = +0$$
$$0001_2 = +1$$
$$\ldots$$
$$0110_2 = +6$$
$$0111_2 = +7$$
$$1000_2 = -8$$
$$1001_2 = -7$$
$$\ldots$$
$$1110_2 = -2$$
$$1111_2 = -1$$

**Zero?**

$$0 \ldots 0_2 = 0$$

**Most negative number?**

# The Golden Child: Two's Complement

Like one's complement, but negative numbers shifted to the left by 1.

$$0000_2 = +0$$
$$0001_2 = +1$$
$$\dots$$
$$0110_2 = +6$$
$$0111_2 = +7$$
$$1000_2 = -8$$
$$1001_2 = -7$$
$$\dots$$
$$1110_2 = -2$$
$$1111_2 = -1$$

**Zero?**

$$0 \dots 0_2 = 0$$

**Most negative number?**

$$10 \dots 0_2 = -2^{n-1}$$

**Most positive number?**

# The Golden Child: Two's Complement

Like one's complement, but negative numbers shifted to the left by 1.

$0000_2 = +0$
$0001_2 = +1$
$\ldots$
$0110_2 = +6$
$0111_2 = +7$
$1000_2 = -8$
$1001_2 = -7$
$\ldots$
$1110_2 = -2$
$1111_2 = -1$

**Zero?**

$0 \ldots 0_2 = 0$

**Most negative number?**

$10 \ldots 0_2 = -2^{n-1}$

**Most positive number?**

$01 \ldots 1_2 = 2^{n-1} - 1$

**Increment Behavior?**

# Two's Complement Summary

- Used by all modern hardware
- Roughly evenly split between positive and negative numbers
- 1 more negative number than positive
- MSB still acts as sign bit
- To negate flip all the bits, and add 1
  - Example: 0b1101 = -(0b0010 + 0b1) = -0b0011 = -3

# Two's Complement Review

**Question:** Suppose we had 5 bits. What integers can be represented in two's complement?

(A) -31 to 31

(B) -15 to 15

(C) 0 to +31

(D) -16 to +15

(E) -32 to +31

# Overflow

- *Overflow* is when the result of an arithmetic operation can't be represented by the number of bits used to perform the calculation
  - i.e. the result is mathematically incorrect (at least in the usual sense)
- Examples:
  - Unsigned: $0b1...1 + 1 = 0b0...0 = 0?$

# Overflow

- *Overflow* is when the result of an arithmetic operation can't be represented by the number of bits used to perform the calculation
  - i.e. the result is mathematically incorrect (at least in the usual sense)
- Examples:
  - Unsigned: $0\text{b}1...1 + 1 = 0\text{b}0...0 = 0$?
  - Two's comp: $0\text{b}01...1 + 1 = 0\text{b}10...0 = -2^{n-1}$?

# Sign Extension

- Want to represent the same number, but using more bits than before
  - Easy for non-negative numbers, just add more leading 0s
  - Sign and magnitude: Add 0s after the sign bit
  - One's complement: "smear" MSB
  - Two's complement: "smear" MSB
- Examples:
  - Sign and magnitude: 0b11 = 0b1001
  - Two's comp: 0b11 = 0b1111

# Summary (1/2)

- CS61c: Learn 6 Great Ideas in Computer Architecture to enable performance programming via parallelism
  1. Layers of Representation/Interpretation
  2. Moore's Law
  3. Principle of Locality/Memory Hierarchy
  4. Parallelism
  5. Performance Measurement & Improvement
  6. Dependability via Redundancy

# Summary (2/2)

- Number Representation: How to represent positive and negative integers using binary
  - Unsigned: Interpret numeral in base 2
  - Signed: Two's Complement
  - Biased: Subtract bias
  - Sign Extension: Extending numerals while preserving their values