CS 61C: Great Ideas in Computer Architecture (a.k.a. Machine Structures)
Lecture 1: Course Introduction

Instructors:
Krste Asanović, Randy H. Katz

http://inst.eecs.berkeley.edu/~cs61c/
Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What You Need to Know About This Class
• Everything is a Number
Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What You Need to Know About This Class
• Everything is a Number
Most Popular Programming Languages 2016-7

1. Python
2. C
3. Java
4. C++
5. C#
6. R
7. JavaScript
8. PHP
9. Go
10. Swift

What do you think is the most "popular" programming language in use today?
Why You Need to Learn C!
CS61C is NOT really about C Programming

• It is about the *hardware-software interface*
  – What does the programmer need to know to achieve the highest possible performance

• C is close to the underlying hardware, unlike languages like Python and Java!
  – Allows us to talk about key hardware features in higher level terms
  – Allows programmer to explicitly harness underlying hardware parallelism for higher *performance* and *power* efficiency
Old School CS61C
New-School Machine Structures

- **Parallel Requests**
  Assigned to computer
  e.g., Search “cats”

- **Parallel Threads**
  Assigned to core
  e.g., Lookup, Ads

- **Parallel Instructions**
  >1 instruction @ one time
  e.g., 5 pipelined instructions

- **Parallel Data**
  >1 data item @ one time
  e.g., Add of 4 pairs of words

- **Hardware descriptions**
  All gates functioning in parallel
  at same time
Agenda

- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- What You Need to Know About This Class
- Everything is a Number
Five Great Ideas
in Computer Architecture

1. Abstraction (Layers of Representation/Interpretation)
2. Moore’s Law (Designing through trends)
3. Principle of Locality (Memory Hierarchy)
4. Parallelism
5. Dependability via Redundancy
Great Idea #1: Abstraction
(Levels of Representation/Interpretation)

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Hardware Architecture Description

Architecture Implementation

Logic Circuit Description

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)

Anything can be represented as a number, i.e., data or instructions

0000 1001 1100 0110 1010 1111 0101 1000 1010 1111 0101 1000 0000 1001 0101 1000 0000 1001 0101 1000 0110 1100 0110 1010 1111 0101 1000 0000 1001 0101 1000 0000 1001

Register File

ALU

Logic Circuit Description

(Circuit Schematic Diagrams)
#2: Moore’s Law

Predicts: 2X Transistors / chip every 2 years

Gordon Moore
Intel Cofounder
B.S. Cal 1950!
Jim Gray’s Storage Latency Analogy: How Far Away is the Data?

- **Tape/Optical Robot**: $10^9$ (2,000 Years)
- **Disk**: $10^6$ (2 Years)
- **Main Memory**: 100
- **On Board Cache**: 10
- **On Chip Cache**: 2
- **Registers**: 1 (ns)

- **Sacramento**: 1.5 hr
- **This Campus**: 10 min
- **This Room**: 1 min
- **My Head**: 1 min

- **Andromeda**: 2,000 Years

Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!
Great Idea #3: Principle of Locality/ Memory Hierarchy
Great Idea #4: Parallelism
Great Idea #5: Dependability via Redundancy

- Redundancy so that a failing piece doesn’t make the whole system fail

Increasing transistor density reduces the cost of redundancy

1+1=2
1+1=2
1+1=1

2 of 3 agree

FAIL!
Great Idea #5: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory to instructors
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)
HOW TO WRITE GOOD CODE:

START PROJECT.

DO things right or do them fast?

FAST

CODE FAST

RIGHT

CODE WELL

DOES IT WORK YET?

NO

ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.

ARE YOU DONE YET?

NO

NO, AND THE REQUIREMENTS HAVE CHANGED.

THROW IT ALL OUT AND START OVER.

GOOD CODE
Why is Architecture Exciting Today?

Stuttering

- Transistors per chip, ‘000
- Clock speed (max), MHz
- Thermal design power*, W
- Chip introduction dates, selected

Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; The Economist

*Maximum safe power consumption
Old Conventional Wisdom

• Moore’s Law + Dennard Scaling = faster, cheaper, lower-power general-purpose computers each year

• In glory days, 1%/week performance improvement!

• Dumb to compete by designing parallel or specialized computers

• By time you’ve finished design, next generation of general-purpose will beat you
New Conventional Wisdom

Google TPU2
Specialized Engine for NN training
Deployed in cloud
45 TFLOPS/chip

Serious heatsinks!
Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What You Need to Know About This Class
• Everything is a Number
Course Information

- Instructors: Krste Asanović and Randy H. Katz
- Teaching Staff:
  - Co-Head TAs: Steven Ho, Steven Chen, and Peijei Li
  - TAs: Nikhil Athreya, Connor Brennan, Irene Dea, Dylan Dreyer, Julian Early, Derek Feng, Haoan Jing, Tejas Kannan, Daniel Ho, Lisa Lee, Ehimare Okayomon, Srinivasa Pranav, Nicholas Riasanovsky, William Sheu
  - Tutors: Ryan Hayes, Anna Li, Sameer Suresh, Morgan Reschenberg, Keyhan Vakil, Nicolas Zoghb
- Textbooks: Average 15 pages of reading/week (can rent!)
- Piazza:
  - Every announcement, discussion, clarification happens there!
CS61c House Rules in a Nutshell

• Webcast? Yes!
• Labs and discussion after NEXT Tuesday’s lecture
• Wait listed? Enroll in any available section/lab, swap later
• Excused Absences: Let us know by second week
• Midterms are in class 26 September and 31 October; Final is 14 December at 7-10 PM
• Labs and Projects (4+1 Extra Credit) are partnered
  – Discussion is Good, but Co-Developing/Sharing/Borrowing Project Code or Circuits is Bad
  – No Public Repos Please: Don’t Look, Don’t Publish
• Join Piazza for more details ... see http://inst.eecs.berkeley.edu/~cs61c/fa17/
EPA!

• **Effort**
  – Attending prof and TA office hours, completing all assignments, turning in HW, doing reading quizzes

• **Participation**
  – Attending discussion and asking great questions

• **Altruism**
  – Helping others in lab or on Piazza

• **EPA! points have the potential to bump students up to the next grade level!** (but actual EPA! scores are internal)
Peer Instruction

• Increase real-time learning in lecture, test understanding of concepts vs. details

• As complete a “segment” ask multiple-choice question
  – 1-2 minutes to decide yourself
  – 2 minutes in pairs/triples to reach consensus.
  – Teach others!
  – 2 minute discussion of answers, questions, clarifications

• No need for iClickers; we will distribute color cards for you to use!
Collaboration in Black and White

• **Good Collaboration**
  – High level discussion and brainstorming, stopping short of code snippets

• **Bad Collaboration**
  – Sitting together and co-writing code, inspecting each other’s code, taking (or giving) code whether in exchange or wholesale copying

• *This should be obvious, but …*
  – Don’t hire someone to do your assignments
  – Don’t ask someone outside of the class (a parent, a student from a previous semester, sourceforge) to help you
  – Don’t use search engines to look for solutions on-line or in someone’s unprotected GitHub (and don’t put your course project solutions in unprotected GitHubs please!)
  – It is supposed to be your own work after all!
Architecture of a typical Lecture

Attention

Full

Time (minutes)

10 35 60 78 90

Administrivia  Fun/News  “And in conclusion…”

9/6/17  Fall 2017 - Lecture #1
An unmatched left parenthesis creates an unresolved tension that will stay with you all day.
Agenda

• Thinking about Machine Structures
• Great Ideas in Computer Architecture
• What You Need to Know About This Class
• Everything is a Number
Computer Data

• Computers represent data as binary values
• Unit element: *bit*
  – Just two possible values, 0 or 1
  – Can be efficiently stored/communicated/manipulated in hardware
• Use many bits to store more complex information, e.g.
  – Byte: 8 bits, can represent $2^8 = 256$ different values
  – Word, e.g. 4 bytes (32 bits) to represent $2^{32}$ different values
  – 64-bit floating point numbers
  – Text files, databases, ... (many bytes)
  – *Computer program*
Binary Number Conversion

**Binary → Decimal**

\[ 1001010_{\text{two}} = ?_{\text{ten}} \]

<table>
<thead>
<tr>
<th>Binary Digit</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 x 2^0 = 0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2^1 = 2</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^2 = 0</td>
</tr>
<tr>
<td>1</td>
<td>0 x 2^3 = 8</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^4 = 0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2^5 = 0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2^6 = 64</td>
</tr>
</tbody>
</table>

\[ \Sigma = 74_{\text{ten}} \]

**Decimal → Binary**

\[ 74_{\text{ten}} = ?_{\text{two}} \]

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (odd?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 37</td>
<td>1</td>
</tr>
<tr>
<td>/2 = 18</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 9</td>
<td>1</td>
</tr>
<tr>
<td>/2 = 4</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 2</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Collect \[ 1001010_{\text{two}} \]
Hexadecimal

• Problem: many digits
  – e.g. $7643_{\text{ten}} = 1110111011011_{\text{two}}$

• Solutions:
  – Grouping: $1\ 1101\ 1101\ 1011_{\text{two}}$
  – Hexadecimal: $1\text{DDB}_{\text{hex}}$
  – Octal: $1\ 110\ 111\ 011\ 011_{\text{two}}$
    $16733_{\text{oct}}$

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
The Computer Knows it, too

```c
#include <stdio.h>

int main() {
    const int N = 1234;

    printf("Decimal: %d\n", N);
    printf("Hex:   %x\n", N);
    printf("Octal: %o\n", N);

    printf("Literals (not supported by all compilers):\n")
    printf("0x4d2 = %d (hex)\n", 0x4d2);
    printf("0b10011010010 = %d (binary)\n", 0b10011010010);
    printf("02322 = %d (octal, prefix 0 - zero)\n", 02322);
}
```

**Output**

- Decimal: 1234
- Hex: 4d2
- Octal: 2322
- Literals (not supported by all compilers):
  - 0x4d2 = 1234 (hex)
  - 0b10011010010 = 1234 (binary)
  - 02322 = 1234 (octal, prefix 0 - zero)
Large Numbers

- Decimal

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Multiplier</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>$10^3$</td>
<td>1000</td>
</tr>
<tr>
<td>M</td>
<td>$10^6$</td>
<td>1000,000</td>
</tr>
<tr>
<td>G</td>
<td>$10^9$</td>
<td>1000,000,000</td>
</tr>
<tr>
<td>T</td>
<td>$10^{12}$</td>
<td>1000,000,000,000</td>
</tr>
</tbody>
</table>

- Binary (IEC)

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Multiplier</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ki</td>
<td>$2^{10}$</td>
<td>1024</td>
</tr>
<tr>
<td>Mi</td>
<td>$2^{20}$</td>
<td>1048,576</td>
</tr>
<tr>
<td>Gi</td>
<td>$2^{30}$</td>
<td>1073,741,824</td>
</tr>
<tr>
<td>Ti</td>
<td>$2^{40}$</td>
<td>1099,511,627,776</td>
</tr>
</tbody>
</table>

E.g. 1GiByte disk versus 1GByte disk

*Marketing exploits this: 1TB disk $\rightarrow$ 100GB less than 1TiB*

https://en.wikipedia.org/wiki/Byte
Signed Integer Representation

Sign & magnitude (8-bit example):

<table>
<thead>
<tr>
<th>sign</th>
<th>7-bit magnitude (0 … 127)</th>
</tr>
</thead>
</table>

Rules for addition, $a + b$:

- If $(a > 0 \text{ and } b > 0)$: add
- If $(a > 0 \text{ and } b < 0)$: subtract
- ...
- +0, -0 → are they equal? comparator must handle special case!

Cumbersome
- "Complicated" hardware: reduced speed / increased power
- *Is there a better way?*
### 4-bit Example

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>+ -3</td>
<td>+ 1101</td>
</tr>
<tr>
<td>4</td>
<td>10100</td>
</tr>
<tr>
<td>+ 16</td>
<td>0100</td>
</tr>
<tr>
<td>4 + 16</td>
<td>+ 10000</td>
</tr>
<tr>
<td>16</td>
<td>0000</td>
</tr>
</tbody>
</table>

- **Map negative → positive numbers**
  - Example for N=4-bit: 
    - $-3 \rightarrow 2^4 - 3 = 13$
  - “Two’s complement”
  - No special rules for adding positive and negative numbers

\[-8 \quad -7 \quad \ldots \quad -1\quad 0 \quad 1 \quad \ldots \quad 7\]

\[0 \quad 1 \quad \ldots \quad 7\]

\[8 \quad 9 \quad \ldots \quad 15\]

\[+ 2^4 = 16\]
## Two’s Complement

<table>
<thead>
<tr>
<th>Signed Decimal</th>
<th>Unsigned Decimal</th>
<th>Binary Two’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-128</td>
<td>128</td>
<td>1000 0000</td>
</tr>
<tr>
<td>-127</td>
<td>129</td>
<td>1000 0001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-2</td>
<td>254</td>
<td>1111 1110</td>
</tr>
<tr>
<td>-1</td>
<td>255</td>
<td>1111 1111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>+0</td>
<td>0000 0001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>127</td>
<td>127</td>
<td>0111 1111</td>
</tr>
</tbody>
</table>

**Note:** Most significant bit (MSB) equals sign
Unary Negation (Two’s Complement)
4-bit Example (-8\text{ten} \ldots +7\text{ten})

Brute Force & Tedious

| \(16\text{ten} - 3\text{ten} = \) | \(10000\text{two} - 0011\text{two} = \) |
| 13\text{ten} | 1101\text{two} |

"largest" 4-bit number + 1

Clever & Elegant

| \(15\text{ten} - 3\text{ten} = \) | \(01111\text{two} - 0011\text{two} = \) |
| 12\text{ten} | 1100\text{two} |
+ 1\text{ten}

| \(13\text{ten} + 0001\text{two} = \) | |
| 1101\text{two} |

invert
Your Turn

• What is the decimal value of the following binary 8-bit 2’s complement number?

1110 0001_{two}

<table>
<thead>
<tr>
<th>Answer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>33_{ten}</td>
</tr>
<tr>
<td>B</td>
<td>-31_{ten}</td>
</tr>
<tr>
<td>C</td>
<td>225_{ten}</td>
</tr>
<tr>
<td>D</td>
<td>-33_{ten}</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
</tr>
</tbody>
</table>
Addition
4-bit Example

Unsigned

\[
\begin{array}{c}
3_{\text{ten}} \\
+ 4_{\text{ten}} \\
\hline
7_{\text{ten}}
\end{array} + \begin{array}{c}
0011_{\text{two}} \\
+ 0100_{\text{two}} \\
\hline
0111_{\text{two}}
\end{array} + \begin{array}{c}
3_{\text{ten}} \\
+ 11_{\text{ten}} \\
\hline
14_{\text{ten}}
\end{array} + \begin{array}{c}
0011_{\text{two}} \\
+ 1011_{\text{two}} \\
\hline
1110_{\text{two}}
\end{array}
\]

Signed (Two’s Complement)

\[
\begin{array}{c}
3_{\text{ten}} \\
+ 4_{\text{ten}} \\
\hline
7_{\text{ten}}
\end{array} + \begin{array}{c}
0011_{\text{two}} \\
+ 0100_{\text{two}} \\
\hline
0111_{\text{two}}
\end{array} + \begin{array}{c}
3_{\text{ten}} \\
+ 11_{\text{ten}} \\
\hline
14_{\text{ten}}
\end{array} + \begin{array}{c}
0011_{\text{two}} \\
+ 1011_{\text{two}} \\
\hline
1110_{\text{two}}
\end{array}
\]

No special rules for two’s complement signed addition
Overflow
4-bit Example

Unsigned

\[
\begin{array}{c}
13_{\text{ten}} + 14_{\text{ten}} \\
\hline
27_{\text{ten}}
\end{array}
\]
\[
\begin{array}{c}
1101_{\text{two}} + 1110_{\text{two}} \\
\hline
11011_{\text{two}}
\end{array}
\]
carry-out and overflow

\[
\begin{array}{c}
7_{\text{ten}} + 1_{\text{ten}} \\
\hline
8_{\text{ten}}
\end{array}
\]
\[
\begin{array}{c}
0111_{\text{two}} + 0001_{\text{two}} \\
\hline
10100_{\text{two}}
\end{array}
\]
no carry-out and no overflow

Signed (Two’s Complement)

\[
\begin{array}{c}
-3_{\text{ten}} + -2_{\text{ten}} \\
\hline
-5_{\text{ten}}
\end{array}
\]
\[
\begin{array}{c}
1101_{\text{two}} + 1110_{\text{two}} \\
\hline
11011_{\text{two}}
\end{array}
\]
carry-out but no overflow

\[
\begin{array}{c}
7_{\text{ten}} + 1_{\text{ten}} \\
\hline
8_{\text{ten}}
\end{array}
\]
\[
\begin{array}{c}
0111_{\text{two}} + 0001_{\text{two}} \\
\hline
10100_{\text{two}}
\end{array}
\]
no carry-out but overflow

Carry-out → Overflow

Carry-out → Overflow
Overflow Detection

4-bit Example

Unsigned
• Carry-out indicates overflow

Signed (Two’s Complement)
• Overflow if
  – Signs of operands are equal
  \[ AND \]
  – Sign of result differs from sign of operands
• No overflow when signs of operands differ

Overflow rules depend on operands (signed vs unsigned)
## Sign Extension

<table>
<thead>
<tr>
<th>Decimal</th>
<th>4-bit</th>
<th>8-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3\text{_ten}</td>
<td>0011_{two}</td>
<td>0000 0011_{two}</td>
<td>0000 0000 0000 0011_{two}</td>
</tr>
<tr>
<td>-3\text{_ten}</td>
<td>1101_{two}</td>
<td>1111 1101_{two}</td>
<td>1111 1111 1111 1101_{two}</td>
</tr>
</tbody>
</table>

- Why is this relevant?
- Assignment differs for signed (above) and unsigned numbers
  - Compiler knows (from type declaration)
  - Different assembly instructions for copying signed/unsigned data
Your Turn

- Which range of decimals can be expressed with a 6-bit two’s complement number?

<table>
<thead>
<tr>
<th>Answer</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-32 ... 32</td>
</tr>
<tr>
<td>B</td>
<td>-64 ... 63</td>
</tr>
<tr>
<td>C</td>
<td>-31 ... 32</td>
</tr>
<tr>
<td>D</td>
<td>-16 ... 15</td>
</tr>
<tr>
<td>E</td>
<td>-32 ... 31</td>
</tr>
</tbody>
</table>
Answer

• Which range of decimals can be expressed with a 6-bit two’s complement number?

<table>
<thead>
<tr>
<th>Answer</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-32 ... 32</td>
</tr>
<tr>
<td>B</td>
<td>-64 ... 63</td>
</tr>
<tr>
<td>C</td>
<td>-31 ... 32</td>
</tr>
<tr>
<td>D</td>
<td>-16 ... 15</td>
</tr>
<tr>
<td>E</td>
<td>-32 ... 31</td>
</tr>
</tbody>
</table>
And In Conclusion ... (1/2)

- **CS61C:**
  - Higher speed performance + better energy efficiency by leveraging computer architecture:
    - Strength and weaknesses (e.g. cache)
    - Performance features (e.g. parallel instructions)
  - Learn C and assembly facilitate access to machine features
- **Basis:** five great ideas in computer architecture
  1. Abstraction: Layers of Representation/Interpretation
  2. Moore’s Law
  3. Principle of Locality/Memory Hierarchy
  4. Parallelism
  5. Dependability via Redundancy
- **Performance Measurement and Improvement**
And In Conclusion ... (2/2)

• Everything is a Number!
  – Collections of bits can store and communicate arbitrary digital data
  – Even programs are represented by bits

• Two’s complement representation avoids special rules for addition of negative numbers