CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Course Introduction, Number Representation

Instructor:
Michael Greenbaum
http://inst.eecs.berkeley.edu/~cs61c/su11

My HW0

- Grew up in the San Fernando Valley ("The Valley"). Moved to nearby Agoura Hills during middle school.
- BS, EECS – UC Berkeley, 2008. Entered PhD program studying AI.
- TA’d 5 times. Cs61c twice.
- Enjoy golfing, playing “roguelike” computer games (nethack, Dungeon Crawl).
- I’m not a professor. Call me Michael.
- Teaching this class will be consuming all my time this summer!

Agenda

- Great Ideas in Computer Architecture
- Administrivia
- Break
- Number Representation

Old School CS61c

New School CS61c
Old-School Machine Structures

New-School Machine Structures (It’s a bit more complicated!)
- Parallel Requests
  Assigned to computer
e. g., Search “Katz”
- 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

6 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 (MIPS)
  Machine Interpretation

#2: Moore’s Law

Predicts: 2X Transistors / chip every 2 years
- Gordon Moore
  Intel Cofounder
  B.S. Cal 1950!
Great Idea #3: Principle of Locality/Memory Hierarchy

Great Idea #4: Parallelism

Great Idea #5: Performance Measurement and Improvement
- Matching application to underlying hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g., matrix manipulation)
- Latency
  - How long to set the problem up
  - How much faster does it execute once it gets going
  - It is all about time to finish

Great Idea #6: Dependability via Redundancy
- Redundancy so that a failing piece doesn’t make the whole system fail

Fig 3. Amdahl's Law as Obstacle to Improved Performance. Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel's planned and actual technology. Core count assumed to double for each rule generation.
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 lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  – Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Agenda

• Great Ideas in Computer Architecture
• Administrivia
• Break
• Number Representation

Course Information

• Course Website: [http://inst.eecs.berkeley.edu/~cs61c/su11](http://inst.eecs.berkeley.edu/~cs61c/su11)
• Textbooks: Average 15 pages of reading/week
  – Barroso & Holzle, *The Datacenter as a Computer*, 1st Edition
• Piazza for main discussion forum.

Your Course Staff

• TAs
  – Justin Hsia
  – Sean Soleyman
  – Alvin Yuan
• 3 Readers
• ~5 Lab Assistants

Course Organization

• Grading
  – Effort, Participation, Altruism (EPA) (5%)
  – Homework (5%)
  – Labs (15%)
  – Projects (30%)
    1. Computer Instruction Set Simulator (C)
    2. Performance Tuning of a Parallel Application/Matrix Multiply using cache blocking, SIMD, MIMD (OpenMP, due with partner)
    3. Computer Processor Design (Logisim)
  – Midterm (20%): end of 4th week
  – Final (25%): last day of class

EPA

• Effort
  – Attending Michael's and TA’s office hours, completing all assignments, turning in HW0
• Participation
  – Attending lecture and discussion sections.
  – Asking great questions in lecture and discussion and making it more interactive
• Altruism
  – Helping others in lab or on Piazza
• EPA! extra credit points have the potential to bump students up to the next grade level! (but actual EPA! scores are internal)
Exam Clobbering Policy

• A higher score on the “midterm” portion of your final than on your midterm will replace your midterm score!
  – Note: The “midterm” portion of the final still counts toward your score on the final, regardless

Late Policy

• Assignments mostly due Sundays at 11:59:59 PM
• Late homeworks not accepted (100% penalty)
• Late projects get 20% penalty, accepted up to two full days later.
  – No credit if more than 48 hours late
  – No “slip days”
  • Only three projects – Don’t want late projects to be the norm.

Policy on Assignments and Independent Work

• With the exception of laboratories and assignments that explicitly permit you to work in groups, all homeworks and projects are to be YOUR work and your work ALONE.
• You are encouraged to discuss your assignments with other students, and extra credit will be assigned to students who help others, particularly by answering questions on Piazza, but we expect that what you hand is yours.
• It is NOT acceptable to copy solutions from other students.
• It is NOT acceptable to copy (or start your) solutions from the Web.
• We have tools and methods, developed over many years, for detecting this. You WILL be caught, and the penalties WILL be severe.
• At the minimum a ZERO for the assignment, possibly an F in the course, and a letter to your university record documenting the incidence of cheating.
• (We caught people last semester!)

Comments on Summer Version

• Summer is EXTREMELY hectic!
  – Double the standard pace.
• Please stay on top of your reading and assignments
• You won’t learn just from lecture. Engaging the material outside of class and lab is critical to your success
• If the course begins to overwhelm you, don’t wait, contact me or your TA immediately!

Discussion Sections Moved!

• As part of a Berkeley Institute of Design research project, all sections will be held in 360 Heart Mining Building (at their usual times).
• Your enrollment in CS61C and your class grades will NOT be affected by your decision to participate in the study (sections without full consent will be held in the location without any of the equipment or experiment proceedings)

Other notes

• My OH will be held Monday, Wednesday from 1-2 pm in the Soda Alcoves.
Attention Time (minutes)

0 20 25 50 55 78 80

Administration Break “And in conclusion...”

Agenda

• Great Ideas in Computer Architecture
• Administrivia
• Break
• Number Representation

Number Representation

• Inside a computer, everything stored as a fixed sequence of 0’s and 1’s.

• How do we represent integers in this format?

Decimal Numbers: Base 10

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

Example:

\[ 9472 = 9000 + 400 + 70 + 2 \]

\[ (9 \times 10^3) + (4 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) \]

Number Bases

• 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:
  - \( d_n \ldots d_2 \ldots d_1 \ldots d_0 \) is an n digit number
  - Value: \( d_n \times B^{n-1} + d_{n-1} \times B^{n-2} + \ldots + d_1 \times B^1 + d_0 \times B^0 \)
Number Bases

- Commonly deal with base 2, base 10, base 16.
  - Will refer to these here as 01101\textsubscript{two}, 12345\textsubscript{ten}, and 12345\textsubscript{hex} for clarity
  - Binary often signified with prefix 0b (eg 0b10101) and hex with 0x.

- An n digit, base B number can represent $B^n$ different things.

Hexadecimal

- Base 16 (hexadecimal) digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- $\text{FFE}_{\text{hex}} = 15_{\text{ten}} \times 16_{\text{ten}}^1 + 15_{\text{ten}} \times 16_{\text{ten}}^0 = 3840_{\text{ten}} + 240_{\text{ten}} + 14_{\text{ten}} = 4094_{\text{ten}}$
- Why hex? Because four sequential bits map to one hex digit. Easier to read a long string of binary if converted to hex.
  - $0111\ 1111\ 1110_{\text{two}} = \text{FFE}_{\text{hex}} = 4094_{\text{ten}}$

Unsigned Integers

Represent non-negative (unsigned) integers using base 2!

<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>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0</td>
</tr>
<tr>
<td>00001</td>
<td>1</td>
</tr>
<tr>
<td>00010</td>
<td>2</td>
</tr>
<tr>
<td>11110</td>
<td>111</td>
</tr>
<tr>
<td>11111</td>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0</td>
</tr>
<tr>
<td>00001</td>
<td>1</td>
</tr>
<tr>
<td>00010</td>
<td>2</td>
</tr>
<tr>
<td>00011</td>
<td>3</td>
</tr>
<tr>
<td>00100</td>
<td>4</td>
</tr>
<tr>
<td>00101</td>
<td>5</td>
</tr>
<tr>
<td>00110</td>
<td>6</td>
</tr>
<tr>
<td>00111</td>
<td>7</td>
</tr>
<tr>
<td>01000</td>
<td>8</td>
</tr>
<tr>
<td>01001</td>
<td>9</td>
</tr>
<tr>
<td>01010</td>
<td>A</td>
</tr>
<tr>
<td>01011</td>
<td>B</td>
</tr>
<tr>
<td>01100</td>
<td>C</td>
</tr>
<tr>
<td>01101</td>
<td>D</td>
</tr>
<tr>
<td>01110</td>
<td>E</td>
</tr>
<tr>
<td>01111</td>
<td>F</td>
</tr>
<tr>
<td>10000</td>
<td>10</td>
</tr>
<tr>
<td>10001</td>
<td>11</td>
</tr>
<tr>
<td>10010</td>
<td>12</td>
</tr>
<tr>
<td>10011</td>
<td>13</td>
</tr>
<tr>
<td>10100</td>
<td>14</td>
</tr>
<tr>
<td>10101</td>
<td>15</td>
</tr>
<tr>
<td>10110</td>
<td>16</td>
</tr>
<tr>
<td>10111</td>
<td>17</td>
</tr>
<tr>
<td>11000</td>
<td>18</td>
</tr>
<tr>
<td>11001</td>
<td>19</td>
</tr>
<tr>
<td>11010</td>
<td>20</td>
</tr>
<tr>
<td>11011</td>
<td>21</td>
</tr>
<tr>
<td>11100</td>
<td>22</td>
</tr>
<tr>
<td>11101</td>
<td>23</td>
</tr>
<tr>
<td>11110</td>
<td>24</td>
</tr>
<tr>
<td>11111</td>
<td>25</td>
</tr>
</tbody>
</table>

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) except for a few of the rightmost digits
- If result of add (or -, *, /) cannot be represented by these rightmost hardware bits, overflow is said to have occurred.

Signed Integers

- Programs often need to deal with negative numbers.
- How do we encode these?
- A sequence of n bits can represent $2^n$ different things.
  - Ideally, want the range evenly split between positive and negative.
  - Can we encode them in such a way that we can use the same hardware regardless of whether the numbers are signed or unsigned?

Sign and Magnitude

- Leading (leftmost) bit says whether positive or negative. 0 is positive, 1 is negative.
- Rest of bits treated as usual.
- 8 bit examples:
  - $0000\ 0010 = -2_{\text{ten}}$
  - $0000\ 0111 = 7_{\text{ten}}$
- What’s wrong with this?
Sign and Magnitude

• Two zeros! 000...0 and 100...0 are +0 and -0.
• Cannot reuse unsigned hardware:

One's Complement

• To negate – complement the bits:
  – +7 => 0000 0111. -7 => 1111 1000.
• Positive numbers always have leading 0's, negative numbers always have leading 1's

Two's Complement

• Minor modification of one's compliment.
  – “Shift” representation of negative numbers down by one to remove duplicate zero
• Using this representation, incrementing an unsigned int will always increment a signed int.
In Conclusion...

- CS61c: Learn 6 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 and Improvement
  6. Dependability via Redundancy
- Number Representation – how to represent positive and negative integers using binary.
  - Unsigned – Use straight up base 2.
  - Signed – Two’s Complement.