Midterm 1 topics (in one slide)

- **Number representation**
  - Bits and bitwise operators
  - Unsigned and signed integers
  - Floating point numbers

- **Machine-level code representation**
  - Instructions, operands, flags
  - Branches, jump tables, loops
  - Procedures and calling conventions
  - Arrays, structs, unions
  - Buffer overflow attacks

Outline

- **Topics in number representation**
- Exam logistics
- Number representation problem
- Topics in machine code
- Machine code problems

Bits and bitwise operations

- Base 2 (binary) and base 16 (hex) generalize from base 10 (decimal)
- And, or, xor, not
- Left shift, two kinds of right shift
  - Similarity to multiply/divide by $2^k$

Unsigned and signed integers

- Unsigned: plain base 2, non-negative
  - Overflow is like operations modulo $2^n$
- Signed: two's complement with a sign bit
  - Sign bit counts for negative place value
  - Overflow possible in both directions
- Comparing the two
  - Ranges partially overlap
  - $+ , -, *$ (same size output), $<<, ==$, narrowing are the same
  - $/ , \%, >>, <, *$ (high output bits), and widening are different
- Algebra properties exist despite overflow

Floating point numbers

- Represent fractions and larger numbers using binary scientific notation
- Fractions whose denominator is a power of two
  - All others must be rounded
  - Limited precision gradually loses information
- Rounding: examine thrown-away bits
- Special cases for $\pm 0$, $\pm \infty$, NaN
- Ordering properties but fewer algebraic properties
Normalized and denormalized

- All but the smallest finite numbers are normalized
  - Represent as $1 \times 2^e$
  - (Leading 1 is not stored)
- For smallest numbers, special denormalized form
  - Smallest exp encoding: same $E$ as smallest normal
  - Leading 0 is not stored

Outline

- Topics in number representation
- Exam logistics
- Number representation problem
- Topics in machine code
- Machine code problems

Exam rules

- Begins promptly at 3:35, ends promptly at 4:25
- Open-book, open-notes, any paper materials OK
- No electronics: no laptops, smartphones, calculators, etc.
  - Arithmetic will use easy numbers
- Sit in alternating seats as long as possible

Exam strategy suggestions

- Writing implement: mechanical pencil plus good eraser
- Make a summary sheet to save flipping through notes or textbook
- Show your work when possible
- Do the easiest questions first
- Allow time to answer every question

Outline

- Topics in number representation
- Exam logistics
- Number representation problem
- Topics in machine code
- Machine code problems

Overflow

- Which of these combinations can describe the addition of the same bits? If possible, give an example with 4-bit ints.
  - No unsigned OF, no signed OF:
  - Unsigned OF, no signed OF:
  - Unsigned OF, positive OF:
  - Unsigned OF, negative OF:
  - No unsigned OF, positive OF:
  - No unsigned OF, negative OF:
- https://chimein.cla.umn.edu/course/view/2021
Which of these combinations can describe the addition of the same bits? If possible, give an example with 4-bit ints.

- No unsigned OF, no signed OF: $\texttt{0000 + 0000 = 0000}$
- Unsigned OF, no signed OF: $\texttt{1111 + 0001 = 0000}$
- Unsigned OF, positive OF: can't happen
- Unsigned OF, negative OF: $\texttt{1000 + 1000 = 0000}$
- No unsigned OF, positive OF: $\texttt{0100 + 0100 = 1000}$
- No unsigned OF, negative OF: can't happen

https://chimein.cla.umn.edu/course/view/2021
Outline

- Topics in number representation
- Exam logistics
- Number representation problem
- Topics in machine code
- Machine code problems

Instructions and operands

- Assembly language ↔ machine code
- Sequence of instructions, encoded in bytes
- An instruction reads from or writes to operands
  - x86: usually at most one memory operand
  - AT&T: destination is last operand
  - AT&T shows operand size with b/w/l/q suffix

Addressing modes

- General form: disp(base,index,\text{scale})
  - Displacement is any constant, scale is 1, 2, 4 or 8
  - Base and index are registers
  - Formula: mem[disp + base + index \cdot scale]
- All but base are optional
  - Missing displacement or index: 0
  - Missing scale: 1
  - Drop trailing (but not leading) commas
- Do same computation, just put address in register: lea

Flags and branches

- Flags (aka condition codes) are set based on results of arithmetic
  - ZF: result is zero
  - SF: result is negative (highest bit set)
  - OF: signed overflow occurred
  - CF: unsigned overflow (“carry”) occurred
- Used for condition in:
  - setCC: store 1 or 0
  - cmovCC: copy or don’t copy
  - jCC: jump or don’t jump
- Just for setting flags: cmp (like sub), test (like and)

Jump tables

- Faster compilation for some switch statements
- Make table of code addresses for cases
- Read from that table like an array
- Fall-through implemented by ordering and/or jumps

Loops

- Simplest structure: conditional jump “at the bottom”, like a C do-while
- C while also checks at beginning
- C for e.g. initializes a variable and updates it on each iteration
- Assembly most like C with goto
Stack and frames

- "The" stack is used for data with a function lifetime
- `%rsp` points at the most recent in-use element ("top")
- Convenient instructions: `push` and `pop`
- Section for one run of a function: stack frame

Calling conventions

- Function arguments go in `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, and `%r9
- Return value is in `%rax`
- Handle that both caller and callee want to use registers
- Caller-saved: callee might modify, caller must save if using
  - `%rax`, `%rdi`, . . . , `%r10`, `%r11`, flags
- Callee-saved: caller might be using, callee must save before using
  - `%rbx`, `%r12`, . . . , `%rbp`, (%`rsp`)
Working with ordering

Which of these conditions are the same?

<table>
<thead>
<tr>
<th>Col. 1</th>
<th>Col. 2</th>
<th>Col. 3</th>
<th>Col. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: x &lt; y</td>
<td>B: x &gt; y</td>
<td>C: x &lt;= y</td>
<td>D: x &gt;= y</td>
</tr>
<tr>
<td>y &lt; x</td>
<td>y &gt; x</td>
<td>y &lt;= x</td>
<td>y &gt;= x</td>
</tr>
<tr>
<td>!(x &lt; y)</td>
<td>!(x &gt; y)</td>
<td>!(x &lt;= y)</td>
<td>!(x &gt;= y)</td>
</tr>
<tr>
<td>!(y &lt; x)</td>
<td>!(y &gt; x)</td>
<td>!(y &lt;= x)</td>
<td>!(y &gt;= x)</td>
</tr>
</tbody>
</table>

https://chimein.cla.umn.edu/course/view/2021