CSci 2021: Review Lecture 1

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Quiz 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
  - 32-bit versus 64-bit
  - Buffer overflow attacks

Outline

Topics in number representation

Topics in machine code

Number representation problems

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 +/- 0, +/- ∞, NaN
- Ordering properties but fewer algebraic properties

Normalized and denormalized

- All but the smallest finite numbers are normalized
  - Represent as $1.x \cdot 2^e$
  - (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

Topics in machine code

Number representation problems

Machine code problems

Instructions and operands

- Assembly language $\leftrightarrow$ 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,scale)
  - Displacement is any constant, scale is 1, 2, 4 or 8
  - Base and index are registers
  - Formula: mem[disp + base + index · 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
- `%esp` points at the most recent in-use element ("top")
- Convenient instructions: `push` and `pop`
- Section for one run of a function: stack frame
- `%ebp` used to point at current frame

Calling conventions

- Handle that both *caller* and *callee* want to use registers
- Caller-saved: callee might modify, caller must save if using
  - `%eax`, `%ecx`, `%edx`, flags
- Callee-saved: caller might be using, callee must save before using
  - `%ebx`, `%esi`, `%edi`, (`%esp`, `%ebp`)
- Function arguments appear on stack below return address
- Return value is in `%eax`

Arrays

- Sequence of values of same size and type, next to each other
- Numbered starting from 0 in C
- To find location: start with base, add index times size
- C's pointer arithmetic is basically the same operation
- Multi-dimensional array
  - Needs more multiplying
- Array of pointers to arrays
  - Different, more flexible layout
  - Each access needs more loads

Structs and unions

- Struct groups objects of different types and sizes, in order
- Fields often accessed using displacement from a pointer
- Alignment requirements $\longrightarrow$ padding
  - Most primitive values aligned to their size
  - Pad between elements, when next needs more alignment
  - Pad at end, to round off total size
- Unions: "like structs where every offset is 0"
  - Used to save space if only one needed at a time
  - Can also reveal storage details

x86-64

- C `long` and pointers increase to 64-bits
- 32-bit registers widen to 64-bit ("r"), plus 8 more
  - 64-bit operations specified with `q` suffix
  - 32-bit operations still possible, usually zero-extend result
- Frame pointer usually not used
- First six (i.e., most) parameters passed in registers

Buffer overflows

- Local arrays stored on the stack
- C compilers usually do not check limits of array accesses
- Too much buffer data can overwrite a return address
  - Changes what code will execute
  - Various nefarious uses
- Various partial defenses:
  - Randomize stack location
  - Non-executable stack
  - Stack canary checking

Outline

Topics in number representation

Topics in machine code

Number representation problems

Machine code problems

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow:
  - Unsigned overflow, no signed overflow:
  - Unsigned overflow, positive overflow:
  - Unsigned overflow, negative overflow:
  - No unsigned overflow, positive overflow:
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow:
  - Unsigned overflow, positive overflow:
  - Unsigned overflow, negative overflow:
  - No unsigned overflow, positive overflow:
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow: `1111 + 0001 = 0000`
  - Unsigned overflow, positive overflow:
  - Unsigned overflow, negative overflow:
  - No unsigned overflow, positive overflow:
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow: `1111 + 0001 = 0000`
  - Unsigned overflow, positive overflow: can’t happen
  - Unsigned overflow, negative overflow:
  - No unsigned overflow, positive overflow:
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow: `1111 + 0001 = 0000`
  - Unsigned overflow, positive overflow: can't happen
  - Unsigned overflow, negative overflow: `1000 + 1000 = 0000`
  - No unsigned overflow, positive overflow:
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow: `1111 + 0001 = 0000`
  - Unsigned overflow, positive overflow: can't happen
  - Unsigned overflow, negative overflow: `1000 + 1000 = 0000`
  - No unsigned overflow, positive overflow: `0100 + 0100 = 1000`
  - No unsigned overflow, negative overflow:

Overflow

- Which of these combinations can describe the same additions?
  - No unsigned overflow, no signed overflow: `0000 + 0000 = 0000`
  - Unsigned overflow, no signed overflow: `1111 + 0001 = 0000`
  - Unsigned overflow, positive overflow: can't happen
  - Unsigned overflow, negative overflow: `1000 + 1000 = 0000`
  - No unsigned overflow, positive overflow: `0100 + 0100 = 1000`
  - No unsigned overflow, negative overflow: can't happen

Outline

Topics in number representation

Topics in machine code

Number representation problems

Machine code problems

Working with ordering

Which of these conditions are the same?

```
  x < y      x > y      x <= y      x >= y
  y < x      y > x      y <= x      y >= x
!(x < y)   !(x > y)   !(x <= y)   !(x >= y)
!(y < x)   !(y > x)   !(y <= x)   !(y >= x)
```

Working with ordering

Which of these conditions are the same?

```
A:x < y      B:x > y      C:x <= y      D:x >= y
B:y < x      A:y > x      D:y <= x      C:y >= x
D:!(x < y)   C:!(x > y)   B:!(x <= y)   A:!(x >= y)
C:!(y < x)   D:!(y > x)   A:!(y <= x)   B:!(y >= x)
```