Midterm 1 topics (in one slide)

- The C language
  - Functions, variables, and types
  - Branches and loops
  - Arrays, pointers, and structures
- Number representation
  - Bits and bitwise operators
  - Unsigned and signed integers
  - Floating point numbers
- Machine-level code representation
  - Instructions, operands
  - Arithmetic and addressing modes

Outline

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

C compared to other languages

- Predecessor of C++, Java, other more modern languages
- No objects, for instance functions and no methods
- Most features have a direct translation to machine code

C numeric types

- char, short, int, and long are 8, 16, 32, or 64 bits on x86-64
- Unsigned integers are $\geq 0$
- Mixed operands upgraded to larger size and unsigned
- float and double are 32-bit and 64-bit floating point

Kinds of variables and allocation

- Local variables exist in one function execution, and go away when it is over
  - Even if you think you have a pointer to it!
- Global variables can be accessed from any function, and last for the whole program
- For more control, allocate memory with malloc and get a pointer
### C strings
- Instead of a real string type, C programs pass pointers to characters.
- Usually, length of string is indicated by a `\0` terminator.
- Transform strings by writing loops over characters.
- Programmer needs to be explicit about allocation and sharing.

### C pointers
- Pointers hold addresses, and the compiler knows their type.
- Create a pointer to a variable with `&`.
- Dereference a pointer with `*`.
- Pointer arithmetic uses the element size, like an array.
- In fact, `a[x]` is the same as `*(a + x)`.

### More about pointers
- Pointer parameters implement pass by reference.
- The null pointer doesn't point at anything.
  - So don't dereference it.
- When using pointers, pay attention to data lifetime and sharing.

### C structures
- A `struct` groups several related values together.
  - Similar to objects with features removed.
- Commonly structs are accessed with pointers, fields with `->` for instance, to implement linked lists and trees.
- `malloc` with the structure size is like `new`.

### For instance, HA1 search tree
- Every search tree node is a `struct`.
  - Each allocated with `malloc`.
- Choices for string storage:
  - Struct has char pointer, can reuse slurped storage.
  - Struct has char array, use `strcpy`.
  - Struct has char pointer, use `strdup`.
  - Optionally, remember string length.

### Outline
- **C language topics**
- **Exam logistics**
  - Topics in number representation
  - 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.
- No arithmetic on big numbers needed
- Leave at least one seat between students

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

C language topics
Exam logistics
Topics in number representation
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
  - $+,-,\times$ (same size output), $\ll$, $\gg$, $\langle$, $\rangle$ (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, +/- $\infty$, NaN
- Ordering properties but fewer algebraic properties
Normalized and denormalized

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

C language topics
Exam logistics
Topics in number representation
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:

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

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

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: $\text{disp(base,index,\_scale)}$
- Displacement is any constant, scale is 1, 2, 4 or 8
- Base and index are registers
- Formula: $\text{mem[disp + base + index \cdot scale]}$
- 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: $\text{lea}$

Outline

C language topics
Exam logistics
Topics in number representation
Number representation problem
Topics in machine code
Machine code problems
Working with ordering

Which of these conditions are the same?

\[
\begin{align*}
\text{Col. 1} & \quad \text{Col. 2} & \quad \text{Col. 3} & \quad \text{Col. 4} \\
A: x < y & \quad B: x > y & \quad C: x \leq y & \quad D: x \geq y \\
B: y < x & \quad A: y > x & \quad D: y \leq x & \quad C: y \geq x \\
D!: (x < y) & \quad C!: (x > y) & \quad B!: (x \leq y) & \quad A!: (x \geq y) \\
C!: (y < x) & \quad D!: (y > x) & \quad A!: (y \leq x) & \quad B!: (y \geq x)
\end{align*}
\]