Slide 1

Administrivia

- Reminder: Quiz 2 Wednesday.

Slide 2

Minute Essay From Last Lecture

- Convert 34 (base 10) to binary (base 2).
- Convert 34 (base 10) to hexadecimal (base 16).
- Convert 19 (base 16) to decimal (base 10).
- Convert -34 (base 10) to binary, using 16-bit two's complement notation.

Slide 3

Arrays and Pointers

- Recall (or observe) that in C (and C++) arrays and pointers are "the same": `void foo(char msg[])` same as `void foo(char *msg)`.
- Consider two ways of setting all elements of an array to 0:

```
void clear1(int a[], int n) {
    for (int i = 0; i < n; ++i)
            a[i] = 0;
}

void clear2(int *a, int n) {
    for (int *p = a; p < a+n, ++p)
            *p = 0;
}
```

Once upon a time, people interested in writing fast code were told to use `clear2`. Why?

Slide 4

Arrays and Pointers, Continued

- Compare code for `clear1` (left) and `clear2` (right):

```
        add   $t0, $zero, $zero # i                 add   $t0, $a0, $zero    # p
loop:   add   $t1, $t0, $t0     # 2*i       loop:   sw    $zero, 0($t0)
        add   $t1, $t1, $t1     # 4*i               addi  $t0, $t0, 4
        add   $t2, $a0, $t1     # addr(a[i])        add   $t1, $a1, $a1      # 2*n (*)
        sw    $zero, 0($t2)                         add   $t1, $t1, $t1      # 4*n (*)
        addi  $t0, $t0, 1                           add   $t2, $a0, $t1      # a+n (*)
        slt   $t3, $t0, $a1                         slt   $t3, $t0, $t2
        bne   $t3, $zero, loop                      bne   $t3, $zero, loop
```

- Which is faster? (Look at the instructions marked with *.)

Slide 5

Binary Versus Decimal

- In decimal (base 10) notation, each digit is multiplied by a power of 10. Same idea for binary (base 2), but using powers of 2.
- So, converting from binary to decimal is easy (if tedious), working from definition. Example?
- Converting from decimal to binary? Repeatedly divide by 2 and record remainders . . .

  We could describe this as a recursive algorithm for computing $bits(n)$:
  - Base case is $n < 2$; trivial.
  - For recursive step, divide n by 2 to get quotient q and remainder r. Then $n = 2q + r$, and:

    The last bit of $bits(n)$ should be r.

    The remaining bits are $bits(q)$, left-shifted by 1.
- Terminology: "Least significant" and "most significant" bits.

Slide 6

Binary Versus Hexadecimal

- Binary is useful for showing real internal state but not very compact. Decimal is compact but not so easy to convert to/from binary.
- We might notice — easy to convert to/from a base that's a power of 2. Hence the use of "octal" (base 8) and "hexadecimal" (base 16). For the latter, we need more than 10 digits, so we use "A" through "F".

  Examples?
- Notice that we can also convert directly to/from decimal, much as we did for binary.

Slide 7

Representing Integers

- Representing non-negative integers is easy — convert to binary and pad on the left with zeros.
- What about negative integers?
- Could try using one bit for sign, but then you have +0 and -0, and there are other complications.
- Or ... consider a car odometer — in effect, representable numbers form a circle, since adding 1 to largest number yields 0.

Slide 8

Representing Integers, Continued

- We could implement the car-odometer idea in binary, and then choose where to "cut the circle" (between smallest and largest):
  - Between 0 and all ones — unsigned integers.
  - Between largest number with 0 as the MSB and smallest number with 1 as MSB — "two's complement" signed integers.
- Notice that with the two's complement scheme, +1/-1 moves us "around the circle" — nothing special needed for negative numbers.
- Notice that if we have n bits, adding 2^n to x gives us x again. This leads to an easy way to compute $-x$: Compute $2^n - x$, and notice that

  $$2^n - x \;=\; (2^n - 1) - x + 1$$

  which is very easy to compute ...

  Examples?

Minute Essay

- Convert 30_{10} to binary and then to hexadecimal.
- Convert -30_{10} to 16-bit two's complement notation; show in binary and hexadecimal.
- Convert $2a_{16}$ to decimal.

Slide 9