CSCI 210: Computer Architecture
Lecture 6: Number Systems

Stephen Checkoway
Oberlin College
Slides from Cynthia Taylor

Announcements

- Problem Set 1 due TONIGHT 11:59 p.m.
- Problem Set 2 available now
- Lab 1 available now and due Sunday, February 23

Why we need to learn binary (and other number systems)

- Fundamental to how your computer works
  - Will need a good grasp of binary to understand things like logical operations
  - Will need it a lot when we get to logic gates and how the CPU works
  - Will need to translate to binary to work out examples

- Need to understand it to understand many things like network protocols (IP addresses), bit masking, etc.

Positional Notation

- The meaning of a digit depends on its position in a number.
- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ in base b represents the value

$$d_n * b^n + d_{n-1} * b^{n-1} + \ldots + d_2 * b^2 + d_1 * b^1 + d_0 * b^0$$

Consider 101

- In base 10, it represents the number 101 (one hundred one)
- In base 2, 101_2 =
- In base 8, 101_8 =

$101_5 = ?$

A. 26

B. 51

C. 126

D. 130

$122_{-3}=?$

A. 17

B. 5

C. 10

D. -30

CS History: Negabinary

- Early Polish computers the BINEG (1959) and UMC-1 (1962) used negative binary (base -2)
- Allowed for a natural representation of both negative and positive numbers
- Problem: Math is more complicated

Binary: Base 2

- Used by computers

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0, 1}, represents the value

$$d_n * 2^n + d_{n-1} * 2^{n-1} + \ldots + d_2 * 2^2 + d_1 * 2^1 + d_0 * 2^0$$

Binary to Decimal

- We have b = 2

10110_2 =

Decimal to Binary

- Convert 115 to binary
- We know

  $115 = d_n \cdot 2^n + \cdots + d_1 \cdot 2^1 + d_0 \cdot 2^0$

  $= 2(d_n \cdot 2^{n-1} + \cdots + d_1) + d_0$
- 115 is odd and $2(d_n \cdot 2^{n-1} + \cdots + d_1)$ is even so $d_0 = 1$
- Subtract 1, divide by 2, and repeat

  $57 = d_n \cdot 2^{n-1} + \cdots + d_2 \cdot 2^1 + d_1$

  $= 2(d_n \cdot 2^{n-2} + \cdots + d_2) + d_1$

Convert 115 to Binary

Decimal to Binary

- Repeatedly divide by 2, recording the remainders
- The remainders form the binary digits of the number from the least significant to the most significant
- Converting 25 to binary

$34_{10}=?_{2}$

A. 010001

B. 010010

C. 100010

D. 111110

E. None of the above

Hexadecimal: Base 16

- Like binary, but shorter!
- Each digit is a “nibble”, or half a byte (4 bits)
- Indicated by prefacing number with 0x (usually)

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, represents the value

$$d_n * 16^n + d_{n-1} * 16^{n-1} + \ldots + d_2 * 16^2 + d_1 * 16^1 + d_0 * 16^0$$

Hexadecimal to binary

- Each hex digit corresponds directly to four binary digits

- $35AE_{16}$ =

$$23C_{16} = ?_{2}$$

A. 0010 0000 1100

B. 0010 1111 0010

C. 0010 0011 1100

D. 1000 1101 1000

E. None of the above

Octal: Base 8

- Sometimes used to shorten binary
  - Used to specify UNIX permissions (remember CS 241?)

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7}, represents the value

$$d_n * 8^n + d_{n-1} * 8^{n-1} + \ldots + d_2 * 8^2 + d_1 * 8^1 + d_0 * 8^0$$

$$31_8 = ?_{10}$$

A. 24

B. 25

C. 200

D. 208

E. None of the above

If every hex digit corresponds to 4 binary digits, how many binary digits does an octal digit correspond to?

A. 2
B. 3
C. 4
D. 5

Addition

- Use the same place-by-place algorithm that you use for decimal numbers, but do the arithmetic in the appropriate base

$$2A5C_{16} + 38BE_{16} = ?$$

A. 586A

B. 631A

C. 6986

D. None of the above

How We Store Numbers

- Binary numbers in memory are stored using a finite, fixed number of bits (typically 8, 16, 32, or 64)
  - 8 bits = byte (usually and always in this class)
- Pad extra digits with leading 0s
- A byte representing 4_{10} = 00000100

A byte (8 bits) can store nonnegatives values from 0 up to

A. 127

B. 128

C. 255

D. 256

E. None of the above

Java

- A `byte` is 8 bits
- A `char` is 16 bits
- A `short` is 16 bits
- An `int` is 32 bits
- A `long` is 64 bits

Rust

- bools are 1 byte, chars are 4 bytes
- Specify size in type for ints
  - i8, i16, i32, etc
- isize or usize will be the size of an address on the architecture it's compiled for
  - 32 bits on 32 bit systems, 64 bits on 64 bit systems

In C, an int is

A. 8 bits

B. 16 bits

C. 32 bits

D. It depends

E. None of the above

C specifies a *minimum size* for types

- chars are 1 byte and must be at least 8 bits (but can be more!)
- shorts and ints must be at least 16 bits
- longs are at least 32 bits
- long longs are at least 64 bits
- sizeof(type) tells us how many bytes type is
- 1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)

So how do I know?

- Use sizeof(int) to check
- Or use C99 types like int16_t or int32_t

Reading

- Next lecture: Negatives in binary
  - Section 2.4

- Problem Set 1 due Today
- Lab 1 due a week from Monday