

CPS104
Computer Organization
Lecture 2
Data representations, Data Types
August 26 , 2009
Gershon Kedem

Remindar! Homework-0

- Send me (kedem@cs.duke.edu) email message with: your name, year, major and a short description of your computer science / Engineering background.
- Readings:
- Chapter 2.4 pages 80-94,
- Chapter 3.1-3.2, 3.5 pages 224-229, 242-253
- Chapter 1

Data Representations

Review: Data Representations

- **Question: How do computers store numbers?**

When you write in a program:

```
int x; float y;  
x= 10;  
y = 0.34;
```

- **What do the variables x and y actually hold?**

Review: Number Systems

We use 10 symbols (0,1, ... 9) to represent numbers.

☞ Can one use fewer symbols?

☞ Can one use more?

☞ What is the “best” number of symbols to use inside a digital computer?

- **Humans use decimal (base 10)**

- ◆ digits 0-9 are composed to make larger numbers

Example: $341 = 3 \cdot 10^2 + 4 \cdot 10^1 + 1 \cdot 10^0$

- ◆ **Weighted Positional Notation!!**

- **Addition and Subtraction are straightforward**

- ◆ carry and borrow (today called regrouping)

- **Multiplication and Division less so**

- ◆ can use logarithms and then do adds and subtracts

Review: Number Systems for Computers

- Today's computers are built from transistors (Electronic Switches)
- A switch is either **off** or **on**. (Voltage across a switch high or low)
- Need to represent numbers using only **off-on** or **high-low**
 - Use only two symbols!
- **off** and **on** can represent the digits **0** and **1**
 - ♦ A bit can have a value of 0 or 1
- Computer memory is **organize in words**.
- Each computer word has a fixed number of **binary bits**.
- Typical desktop computer uses 32-bit words.
- 32 bit computers operate on **Bytes** (8 bits), **half words** (16 bits), **words** (32-bits) and **double words** (64 bits).

• **Bottom Line: Everything inside the computer is Binary!!**
That is; all data (and instructions) are represented as bit strings!!

Data Representations

- ★ **All Data (built-in or otherwise) inside a computer is represented as a finite collection of bytes, half-words, single words, or double words (bit strings)!!!**

Binary representation

- weighted positional notation using base 2

$$11_{10} = 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1011_2$$

$$11_{10} = 8 + 2 + 1$$

What is largest number that can be represented, given 4 bits?

Conversion from Decimal to Binary

- **N** is a positive Integer (in decimal representation)
- b_i $i=0,\dots,k$ are the bits (binary digits) for the binary representation of **N**
- $N = b_k * 2^k + \dots + b_2 * 2^2 + b_1 * 2 + b_0$
- binary representation: $b_k \dots b_3 b_2 b_1 b_0$
- How do I compute b_0 ?

Compute binary representation of 11?

Conversion from Decimal

```
i=0;
```

```
while N > 0 do
```

```
     $b_i = N \% 2;$  //  $b_i = \text{remainder}; N \bmod 2$ 
```

```
     $N = N / 2;$  //  $N$  becomes quotient of division
```

```
    i++;
```

```
end while
```

- Replace 2 by **A** and you have an algorithm that computes the **base A** representation for N

Powers of 2

<u>N</u>	<u>2ⁿ</u>	<u>Binary</u>
0	1	0000000001
1	2	0000000010
2	4	0000000100
3	8	0000001000
4	16	0000010000
5	32	00000100000
6	64	00001000000
7	128	00010000000
8	256	00100000000
9	512	01000000000
10	1024 (1K)	10000000000

Binary, Octal and Hexidecimal numbers

- **Computers can input and output decimal numbers but they convert them to internal binary representation.**
- **Binary is good for computers, hard for us to read**
 - ◆ **Use numbers easily computed from binary**
- **Binary numbers use only two different digits: {0,1}**
 - ◆ **Example: $1200_{10} = 0000010010110000_2$**
- **Octal numbers use 8 digits: {0 - 7}**
 - ◆ **Example: $1200_{10} = 04260_8$**
- **Hexidecimal numbers use 16 digits: {0-9, A-F}**
 - ◆ **Example: $1200_{10} = 04B0_{16} = 0x04B0$**
 - ◆ **One does not distinguish between upper and lower case**

Binary and Octal

- Easy to convert Binary numbers To/From Octal.
- Group the binary digits in groups of three bits and convert each group to an Octal digit.

• $2^3 = 8$

Bin.	Oct.
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Example:

11 000 010 011 001 110 100 111 101 010 101₂
3 0 2 3 1 6 4 7 5 2 5₈

Binary and Hex

- To convert to and from hex: group binary digits in groups of four and convert according to table
- $2^4 = 16$

Hex	Bin	Hex	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Example:

1100 0010 0110 0111 0100 1111 1101 0101₂
C 2 6 7 4 F D 5₁₆

Issues for Binary Representation

- **Complexity of arithmetic operations**
- **Negative numbers**
- **Maximum representable number**

Binary Integers

- **Unsigned Integers:**

- ♦ $i = 100101_2$; $i = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

- **4 bits => max number is 15**

- **What about representing negative numbers?**

Sign-Magnitude Representation for Integers

- Add a sign bit

 - ◆ Example: $010110_2 = 22_{10}$; $110110_2 = -22_{10}$

- Advantages:

 - ◆ Simple extension of unsigned numbers.
 - ◆ Same number of positive and negative numbers.

- Disadvantages:

 - ◆ Two representations for 0: $0=000000$; $-0=100000$.
 - ◆ Algorithm (circuit) for addition depends on the arguments' signs.

2's Complement Representation for Integers

- Key idea is to use largest positive binary numbers to represent negative numbers
- Obtain negative number by subtracting large constant
- $i = -a_{n-1} * 2^{n-1} + a_{n-2} * 2^{n-2} + \dots + a_0 * 2^0$

6-bit examples:

$$010110_2 = 22_{10} ; 101010_2 = -22_{10}$$

$$0_{10} = 000000_2 ; 1_{10} = 000001_2 ; -1_{10} = 111111_2$$

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

2's Complement

- **Advantages:**

- ◆ Only one representation for 0: $0 = 000000$
- ◆ Addition algorithm independent of sign bits.

- **Disadvantage:**

- ◆ One more negative number than positive :
Example: 6-bit 2's complement number.
 $100000_2 = -32_{10}$; but 32_{10} could not be represented

2's Complement Negation and Addition

- ★ To negate a number do:
 - ◆ Step 1. complement the digits
 - ◆ Step 2. add 1

Examples

$$\begin{array}{r} 14_{10} = 001110_2 \\ -14_{10} = 110001_2 \\ \quad \quad \quad + 1 \\ \hline 110010_2 \end{array}$$

$$\begin{array}{r} 010010_2 \\ +110010_2 \\ \hline 000100_2 \end{array}$$

- ★ To add signed numbers use regular addition but disregard carry out
 - ◆ Example $18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}$

2's Complement (cont.)

* Example: $A = 0x0ABC$; $B = 0x0FEB$.

* Compute: $A + B$ and $A - B$ in 16-bit 2's complement arithmetic.

2's Complement Precision Extension

- ✱ Most computers today support 32-bit (int) or 64-bit integers
 - ◆ 64-bit using `gcc` is `long long`
 - ◆ 64-bit using Digital/Compaq compiler is `long`
- ✱ To extend precision do `sign bit extension`
 - ◆ precision is number of bits used to represent a number

Example

$14_{10} = 001110_2$ in 6-bit representation.

$14_{10} = 000000001110_2$ in 12-bit representation

$-14_{10} = 110010_2$ in 6-bit representation

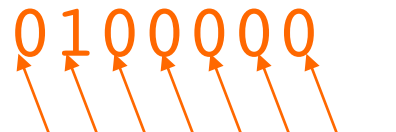
$-14_{10} = 111111110010_2$ in 12-bit representation.

Overflow

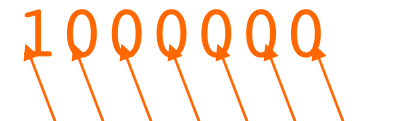
- Two's Complement **representation** is **finite**.
 - ◆ For any fixed precision there is a maximum number that can be represented. (The largest positive number).
 - ◆ There is also the smallest number (The largest negative number).
- **What do they** (the largest and smallest numbers) **look like?**
- **What happened when you add (subtract) two numbers and the result is too big (or too small) to be represented?**

Overflow

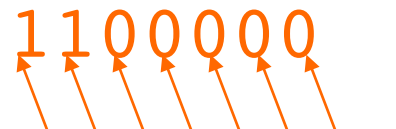
Example1:


$$\begin{array}{r} 0110101_2 \quad (= 53_{10}) \\ +0101010_2 \quad (= 42_{10}) \\ \hline 1011111_2 \quad (= -33_{10}) \end{array}$$

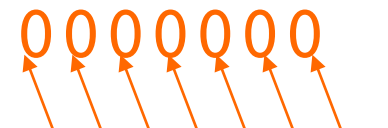
Example2:


$$\begin{array}{r} 1010101_2 \quad (= -43_{10}) \\ +1001010_2 \quad (= -54_{10}) \\ \hline 0011111_2 \quad (= 31_{10}) \end{array}$$

Example3:


$$\begin{array}{r} 0110101_2 \quad (= 53_{10}) \\ +1101010_2 \quad (= -22_{10}) \\ \hline 0011111_2 \quad (= 31_{10}) \end{array}$$

Example4:


$$\begin{array}{r} 0010101_2 \quad (= 21_{10}) \\ +0101010_2 \quad (= 42_{10}) \\ \hline 0111111_2 \quad (= 63_{10}) \end{array}$$

What About Non-integer Numbers?

- 👉 How can one represent very large and very small numbers?
 - 👉 What are the problems with finite representation?
 - 👉 Can we represent every number exactly?
-
- ✳️ There are infinitely many real numbers between any two integers.
 - ✳️ Many important numbers are real
 - ◆ speed of light $\approx 3 \times 10^8$
 - ◆ $\pi = 3.1415\dots$
 - ✳️ Fixed number of bits limits range of integers
 - ◆ Can't represent some important numbers
 - ✳️ Humans use Scientific Notation
 - ◆ 1.3×10^4

Floating Point Representation

Numbers are represented by:

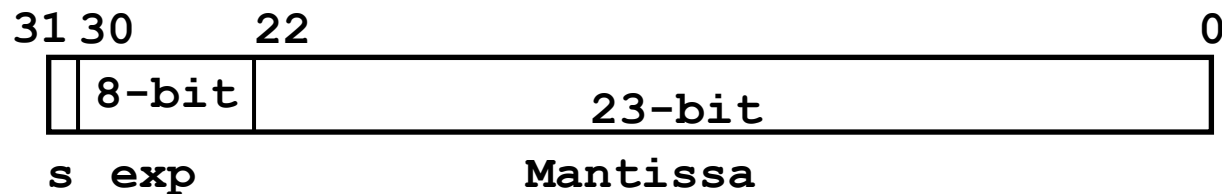
$$Z = (-1)^s \times 2^{E-127} \times 1.M$$

S := 1-bit field ; Sign bit

E := 8-bit field; Exponent: Biased integer, $0 \leq E \leq 255$.

M := 23-bit field; Mantissa: Normalized fraction with hidden 1
(don't actually store it)

Single precision floating point number
uses 32-bits for representation



Floating Point Representation

- The mantissa represents a fraction using binary notation:

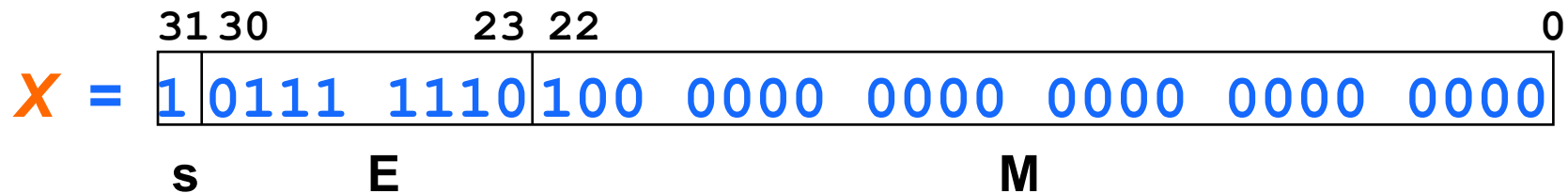
$$M = .s_1, s_2, s_3 \dots = 1.0 + s_1 * 2^{-1} + s_2 * 2^{-2} + s_3 * 2^{-3} + \dots$$

- Example: $X = -0.75_{10}$ in single precision $(-(1/2 + 1/4))$

$$-0.75_{10} = -0.11_2 = (-1) \times 1.1_2 \times 2^{-1} = (-1) \times 1.1_2 \times 2^{126-127}$$

$$S = 1 ; \text{Exp} = 126_{10} = 0111\ 1110_2 ;$$

$$M = 100\ 0000\ 0000\ 0000\ 0000\ 0000_2$$



Floating Point Representation

Example:

What floating-point number is:

0xC1580000?

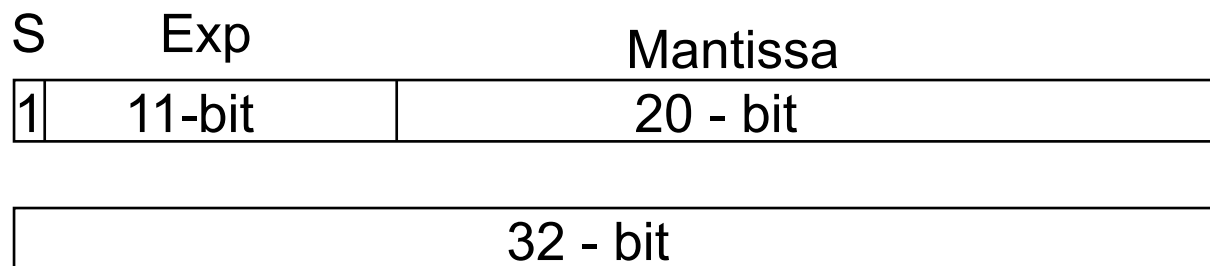
Floating Point Representation

- **Double Precision Floating point:**

64-bit representation: 1-bit **sign**, 11-bit (biased) **exponent**; 52-bit **mantissa** (with hidden 1).

$$X = (-1)^s \times 2^{E-1023} \times 1.M$$

Double precision floating point number



Reading Assignment

Reading:

- Chapter 2.4 pages 80-94,
- Chapter 3.1-3.2, 3.5 pages 224-229, 242-253
- Chapter 1