Computer Science 210 tutorial 2

Data representation

Binary number system

- Last time we have learnt how to represent numbers (positive natural numbers) in binary.
- For example:
  - $17_{10} = 10001$
  - $1024_{10} = 10000000000_2$
- But… how do we represent negative numbers
- 4 possible ways:
  - Sign magnitude
  - Offset binary
  - 1's complement
  - 2's complement

Sign magnitude

- Assume we have N bits available to represent numbers.
- We use the most significant bit to represent sign (+ or –), '+' -> 0 and '–' -> 1
- Say to represent +12 using 8 bits
- +12 = (+) (12) = '0' '0001100' = 00001100
- And -12 is represented as:
- -12 = (–) (12) = '1' '0001100' = 10001100
- But… 00000000 = +0 = 0
- And… 10000000 = -0 = 0

Off set binary

- Assume we have N bits available to represent numbers.
- Those bits can represent 2^N different values
- ➔ can be from 0 to 2^N or from -2^N to 0 or from -2^{N-1} to 2^{N-1}, or also can be from A to B in any range as long as $B - A = 2^N$
- N bits: 000….000 -> 111….111
- **Set 000….000 = A and 111….111 = B**
- -> 000….001 = A+1
- -> 000….010 = A+2
- -> 000….011 = A+3
- This is offset Excess – A, it's only show one 0 but complicated in **calculation.**

1's complement

- Give convenience in calculation (addition)
- Positive numbers are represent the same as sign magnitude:
- +12 = 00001100
- Negative numbers are bit-inversed from positive number
- -12 = inverse(00001100) = 11110011
- But…
  - +0 = 00000000 = -0 = 11111111
  - Wasting expensive bits/memory

2's complement

- Develop from 1's complement
- Positive numbers are represent the same as sign magnitude:
- +12 = 00001100
- Negative numbers are bit-inversed from positive number and add 1
- -12 = inverse(00001100)+1 = 11110011+1 = 11110100
- This is widely used in all computer system nowadays

Table of signed binary numbers

Performing Arithmetic:

value	Sign Magnitude	Offset Binary	1's complement	2's complement
+7	0111	1111	0111	0111
+6	0110	1110	0110	0110
+5	0101	1101	0101	0101
+4	0100	1100	0100	0100
+3	0011	1011	0011	0011
+2	0010	1010	0010	0010
+1	0001	1001	0001	0001
0	0000	1000	0000	0000
-1	1001	0111	1110	1111
-2	1010	0110	1101	1110
-3	1011	0101	1100	1101
-4	1100	0100	1011	1100
-5	1101	0011	1010	1011
-6	1110	0010	1001	1010
-7	1111	0001	1000	1001
-8	//	0000	//	1000
-0	1000	//	1111	//

Over flow and under flow

Carry from MSB?	Carry into MSB?	overflow
no	no	no
no	yes	yes
yes	no	yes
yes	yes	no

Overflow and Underflow in addition:

- Adding two numbers with different signs can never produce an overflow or underflow.
- Adding two positive numbers produces an overflow if the sign of the result is negative.
- Adding two negative numbers produces an underflow if the sign of the result is positive.
- Note that in one case there is a carry out and in the other there is not

(+7) 0111	(-7) 1001
(+7) 0111	(-6) 1010
(+14) 1110	(-13) 0011

Overflow and Underflow in Subtraction:

- Subtracting two numbers with the same signs can never produce an overflow or underflow.
- Subtracting a negative number from a positive number produces an overflow if the sign of the result is negative.
- Subtracting a positive number from a negative number produces an underflow if the sign of the result is positive.

(+4)	0100	0100	-4	1100	1100
-(-5)	-1011	0101	-(+5)	-0101	1011
+9		1001	-9		0111

Shifting (<</>>/>>>)

- Shift operators move bits to the left or to the right.
- •Used to shift the bit patterns left and right.
- •Shift corresponds to division/multiplication by powers of 2 (no overflow problem)
- •Three shift instructions:
  - "sll" (shift left logical): <<
  - "sra" (shift right arithmetic): >>
  - "srl" (shift right logical): >>>
- •The shift logical instructions fill the vacated bits with 0
- •The shift right arithmetic instruction fills the vacated bits with the sign bit.
- •These instructions can be used to extract fields out of a bit pattern, and interpret them as either unsigned or signed numbers.

Examples

- **3. Example Binary Computation:**
- **3.1 3 – 2 or 3 + (–2)**
- Answer:
- Sign Magnitude: 0011 – 0010 = 0001
- Offset Binary: 1011 + 0110 = (1) 0001
- 1's Complement: 0011 + 1101 = (1) 0000 = 0001
- 2's Complement: 0011 + 1110 = 0001
- **3.2 Given the following bit pattern; 0101101_2**
- **Work out the value of it based on the following assumption:**
- Unsigned 7-bit binary: (45)
- 7-bit sign magnitude (MSB is sign bit): (45)
- 7-bit 2's complement: (45)
- XS-33 (Excess-K, K = 33): (12)
- Unsigned fixed point (assume a 3 bit fraction, 0101.101): (5.625)

Examples

3.3 What is result for 17 Add 19 in binary?
And check is overflow or not in 5 bits(unsigned).

$17_{10} = 10001_2$ $19_{10} = 10011_2$

1 11 <--- Carry bits

(Showing sign bits) 010001

\+ 010011

Discard extra bit → *1*00100

That will be overflowing just use 5 bits binary, but not overflow in 6 bit binary.

3.4 What is result for -17 Add -19 in binary?
And check is overflow or not in 6 bits.

$-17_{10} = 101111_2$ $-19_{10} = 101101_2$

1 1111 <--- Carry bits

(Showing sign bits) 101111

\+ 101101

Discard extra bit → *1*011100

FINAL ANSWER: $011100_2 = +28_{10}$

If we use 8 bits to represent the result, that will be 11011100(-36).

Floating point numbers IEEE 754 standard

- Pre-IEEE754
  - Fixed point Faction: 1001.1111
  - Limitation on range of representation
- IEEE754 standard – 2 types:
  - Single precision (floating) uses 32 bits
  - Double precision (double) uses 64 bits
- The most popular way to represent floating points numbers.
  - Why?
  - Compare to fixed point N bit fractions
- Include: sign bit, magnitude bits and mantissa bits.
- Single precision: 1 + 8 + 23 = 32 bits
- Double precision: 1 + 11 + 52 = 64 bits

Single IEEE 754 example

- Represent -3.25 in IEEE 754 single precision
- $3.25 = 11.01_2 = 11.01 \times 2^0 = 1.101 \times 2^1$
- Read the result:
  - Sign bit = negative(-) = 1_2
  - Magnitude = 1+127 = 128_{10} = 1000,0000
  - Mantissa = 10100000… (fill up 23 bits)
- Concat them together:
  - 1 10000000 10100000000…
  - 1100,0000,0 101,0000,000,…
  - **C0500000 is the answer**

Single IEEE 754 example 2

- C0A40000 in IEEE 754 single precision to decimal floating point representation
- 1100,0000,1010,0100,0000,0000,00000000
- 1/10000001/01001000000000000000000
- Calculation:
  - Sign bit = 1 -> negative number
  - Magnitude = 2^7+1 = 128+1 = 129 - 127 = 2
  - Mantissa = 1.01001
  - Together = - 1.01001 x 2^2 = -101.001
  - = -(5+2^-3) = -5.125

Exercises

- *Question 1:*
- Please answer why there are only 15 values when you use `sign magnitude' in **table of signed binary numbers** on page 7, but 16 when there is no sign bit?

- *Question 2:*
- What are the decimal values of the following binary number if they are **Unsigned 7-bit binary, 7-bit sign magnitude (MSB is sign bit), 7-bit 1's complement, 7-bit 2's complement, XS-13 (Excess-K, K = 13), Unsigned fixed point (assume a 3 bit fraction):**
- 0110101
- 0001110
- 1101001
- 1000101

Exercises

- *Question 3:*
- Computing following equations show the result in 16 bits 2's complement binary number:
- -112+63=
- 78-13=
- -333+111 =
- -123-14 =
- *Question 4:*
- Answer the follow computing is overflow or not in 6 bit binary:
- 001010 + 010100
- 010101 + 100010
- 110110 + 001111
- 110010 + 110011

Exercises

- *Question 5:*
- Answer the follow computing (6 bits used):
- 001010 & 010100
- 010101 | 100010
- 110110 ^ 001111
- NOT(110010 ^ 110011)
- 000100 >> 2
- 110110 << 1
- 100110 >>> 3

Exercises

- *Exercise 6: Convert $C2100000_{16}$ from IEEE 754 Floating Point (Single Precision) to decimal*

- *Exercise 7: Convert 2.25 from Decimal to IEEE 754 Floating Point (Single Precision)*