COMPSCI 210 S1T Computer Systems

Data Representation

Negative Number Representation

Agenda & Reading

- Agenda:
  - Negative Number Representation
    - Sign and Magnitude
    - Excess (Biased)
    - Two's Complement
  - Range
  - Overflow
  - Subtraction by Complement Addition
- Java Example:
  - 03\OverflowInt.java
- Exercise:
  - 03

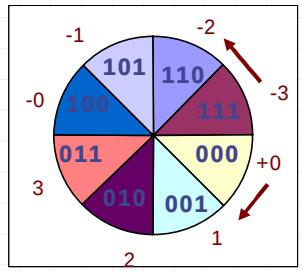
Unsigned & Signed Integers

- Unsigned Representation
  - Represent positive numbers only
  - Example: 8-bit
    - Range : 0 <= x <= 255
    - 11111011 = 251
    - 00000101 = 5
- Signed Representation
  - Represent both positive and negative numbers
  - Three important representations:
    - Sign and Magnitude
    - Excess (biased)
    - Two's complement

Sign And Magnitude

- One bit to represent whether a number is positive or negative
  - Left-most bit as the sign bit
    - 0 – positive, 1 – negative
  - Represent numbers between $-(2^{n-1} -1)$ and $+(2^{n-1} -1)$
- Example: 8-bit
  - Example
    - 10000101 = -5
    - 00000101 = +5
  - Range:
    - -127 <= x <= 127
  - Disadvantage:
    - Two representations for zero
      - 00000000 = +0
      - 10000000 = -0

Example: 3-bit

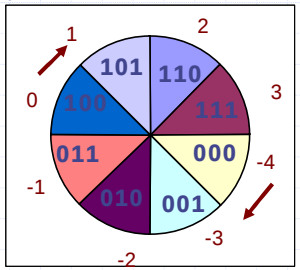


(3-bit)	Value
000	0
001	1
010	2
011	3
100	-0
101	-1
110	-2
111	-3

Excess (biased)

- Represent number by adding the absolute value of the most negative number to the value
- Represent numbers between $-(2^{n-1})$ and $+(2^{n-1} -1)$ in excess 2^{n-1} representation
  - A positive number x is represented as (1 << (n-1)) + x. (i.e. $2^{n-1} + x$).
  - 0 is represented as 1 << n-1. (i.e. 2^{n-1}).
  - A negative number -x is represented as ((1 <<(n-1)) - 1) - x + 1. (i.e. (2^{n-1} - 1) - x + 1).
- Example: 8-bit
  - Example
    - 00000101 = 5 – 128 = -123
    - 10000100 = 4
  - Range:
    - -128 <= x <= 127
- Advantage:
  - Good for ordering purpose
  - Used in the floating point

Example:3-bit



(3-bit)	Value
000	-4
001	-3
010	-2
011	-1
100	0
101	1
110	2
111	3

Excess (Biased) Conversion

- Examples (8-bit)
  - Binary -> Decimal (Formula: Sum – 128)
    - 00000101
      - Sum of all bits = 5
      - Answer = 5 – 128 = -123
    - 10000001
      - Sum of all bits = 129
      - Answer = 129-128 = 1
  - Decimal to Binary
    - Positive number: Formula = 10000000 + x
    - Negative number: Formula = 01111111 - x + 1
    - 30
      - 10000000 + 00011110 = 10011110
    - -30
      - 01111111 - 00011110 + 1= 01100001 + 1
      - = 01100010

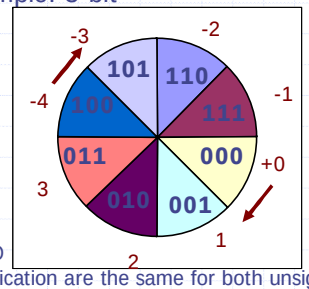
01111111
-00011110
01100001

Invert all bits (except for the most significant bit

Two's Complement

- Used to represent signed integers on most machines
- Represent numbers between $-(2^{n-1})$ and $+(2^{n-1} -1)$
  - A positive number x is represented as itself, x.
  - 0 is represented as itself, 0.
  - A negative number -x is represented as ((1 << n) - 1) - x + 1 (i.e. (2^n -1) - x + 1, or ~x + 1)
    - Two's complement: 1->0 or 0->1, +1
- Example: 8-bit
  - Example:
    - 00000101 = +5
    - 11111011 = -5 (11111010 + 1)
  - Range:
    - -128 <= x <= 127
  - Advantage:
    - Only one representation for ZERO
    - Addition, subtraction, and multiplication are the same for both unsigned and two's complement numbers

Example: 3-bit



(3-bit)	Value
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Two's complement Conversion

- Binary -> Decimal
  - Left most bit: 0 – the rest of the 7 bits represent normal binary numbers
  - Left most bit: 1 – then this represents -ve number. Complement it and add 1
  - **0**0000101
    - +ve => Total of the last seven bits = 5, Answer = 5
  - **1**0000001
    - -ve => Invert all bits + 1=> 01111110 + 1
    - = 01111111 = 127 => -127
- Decimal to Binary
  - Positive number: Formula = x
  - Negative number: Formula = 11111111 - x + 1 (Invert all bits and add 1)
  - 30
    - = 00011110
  - -30
    - 11111111 - 00011110 + 1= 11100001 + 1
    - = 11100010

Examples

- Decimal -> Binary
  - Convert 23_{10} to binary numbers using
    - 8-bit, Unsigned representation
      - 00010111
    - 8-bit, Excess representation :
      - 10000000 + 00010111 = 10010111
    - 8-bit, Two's complement representation :
      - 00010111
  - Convert -23_{10} to binary numbers using
    - 8-bit, Unsigned representation
      - NA
    - 8-bit, Excess representation :
      - 01111111 - x + 1 = 01101000 + 1 = 01101001
    - 8-bit, Two's complement representation :
      - 23 -> Binary: 00010111
      - Complement it + 1 = 11101000 + 1 = 11101001

Examples

- Binary -> Decimal
  - Convert 10101010 to Decimal if the number is represented as
    - Unsigned 8-bit binary numbers
      - 10101010 = 170
    - Signed 8-bit binary numbers in Excess
      - Sum = 170 – 128 =42
    - Signed 8-bit binary numbers in two's complement
      - 10101010 -> Negative number
      - => Complement it + 1 = 01010101 + 1 = 86 => Answer = -86
  - Convert 01010101 to Decimal if the number is represented as
    - Unsigned 8-bit binary numbers
      - 01010101 = 85
    - Signed 8-bit binary numbers in Excess
      - Sum = 85 – 128 = -43
    - Signed 8-bit binary numbers in two's complement
      - 01010101 -> +ve number
      - =85

Exercise

- Convert Decimal –17 to 8-bit binary numbers using
  - Excess
  - Two's complement:
- Convert 10110011 to Decimal if the number is represented as
  - Unsigned 8-bit
  - Signed 8-bit Excess
  - Signed 8-bit Two's complement

Range

- 4-bit
  - Range of unsigned: 0 <= x <= 15
  - Excess: - 8 <= x <= 7
  - Two's complement: - 8 <= x <= 7
- 8-bit
  - Range of unsigned: $0 <= x <= 2^8 - 1$
  - Excess - $(2^7) <= x <= (2^7 – 1)$
  - Two's complement - $(2^7) <= x <= (2^7 – 1)$
- 32-bit
  - Range of unsigned: $0 <= x <= 2^{32} - 1$
  - Excess - $(2^{31}) <= x <= (2^{31} – 1)$
  - Two's complement - $(2^{31}) <= x <= (2^{31} – 1)$

Overflow

- Example: 4-bit
  - Two's complement - 8 <= x <= 7
  - After calculation, if value represents is outside the range, leading to an overflow
  - Examples:

```
0110  6
+ 0111  7
01100 carries
01101 (4-bit) => 1101
```

6 + 7 = 13 (outside the range)

Answer = -3 Invalid answer

```
1010  -6
+ 1001  -7
10000 carries
10011 (4-bit) = 0011
```

-6 + -7 = -13 (outside the range)

Answer = 3 Invalid answer

```
1110  -2
+ 0011  3
11100 carries
10001 (4-bit)= 0001
```

-2 + 3 = 1

Answer = 1 valid answer

```
0010  2
+0011  3
0100 carries
0101
```

2 + 3 = 5

Answer = 5 Valid answer

Valid or invalid?

- Overflow occurs if
  - +ve number add +ve number => -ve number
  - -ve number add –ve number => +ve number
- A better way:
  - look at the **carries into** and **out of the sign bit**. If they are not equal, there is an overflow.
    - Not equal: -> overflow

```
  0110
+ 0111
01100 carries
01101
```

Carry into the sign bit

```
  1010
+ 1001
10000 carries
10011
```

Carry out of the sign bit

    - Equal = OK

```
  1110
+ 0011
11100 carries
10001
```

Carry into & Carry out

```
  0010
+ 0011
00100 carries
0101
```

No carry

Exercise

```
  0111
+ 0101
```

```
  1001
+ 1100
```

```
  1100
+ 0111
```

```
  1001
+ 0011
```


Java Example

- Java uses 4 bytes (32 bits) to store integers
  - Range: Two's complement - (2^{31}) <= x <= $(2^{31} – 1)$
  - Range: –2147483648 to 2147483647
- Example:

```
int num = 2000000000;
System.out.println( num * 2);
```

  - Answer = -294967296
  - Why?
    - An int is 32-bit, the result is 4, 000, 000, 000 which needs >32-bit to store the value.
    - Use 64 bits to store the value =0000000EE6B2800 (hex)
    - Use 32 bits = EE6B2800 (negative, overflow occurs)

Subtraction by Complement Addition

- Perform subtraction by adding the complement of subtrahend
- X – Y = X + (-Y)
- Example:
  - 18 – 11 (18 = 00010010, 11 = 00001011)
  - Two's complement of 11 is 11110101

    00010010 (18)

    + 11110101 (-11)

    111100000 carries

    100000111

    Carry into = carry out => valid (no overflow)

    Answer = 00000111

Subtraction by Complement Addition

- 01110011 – 00001110
  - Two's complement of 00001110 is 11110010

    01110011

    + 11110010

    111100100 carries

    101100101
  - Carry into and out are equal => valid
- 00001010 – 10000011
  - Two's complement of 10000011 is 01111101

    00001010

    + 01111101

    011110000 carries

    10000111
  - Carry into and out are not equal => overflow, invalid answer

Exercises

- 01101011 – 00010110
- 00101100 – 01101001