Bits, Ints and Floats, Vim

COMP201 Lab 2

Spring 2022

KOÇ UNIVERSITY

Vi/Vim Reminder

Vi/Vim Reminder

- Insert mode
  - The one on the left picture.
  - To switch from normal mode to insert mode, type ‘i’ in the normal mode.
  - Every character you type is put to the file.
  - To switch back to normal mode, press <Esc>

Vi/Vim Reminder

- Visual mode
  - To switch from normal mode to visual mode, type ‘v’.
  - You can select blocks of text.
  - Type d to delete the block, c to delete the block and switch to insert mode to replace the deleted block with another string.
  - To switch back to normal mode, type <Esc>.

Basic Commands in Vi/Vim (in Normal Mode)

- *Basic movements:* h (left), j (down), k (up), l (right)

- *Moving across words:* w (next word), b (beginning of word), e (end of word)

- *Jumping in a line:* 0 (beginning of line), $ (end of line)

- *Jumping in a file:* gg (beginning of file), G (end of file), :{num}<Enter> (moving to line number num)

- *Searching for a string:* /{regex}, n (moving forward to find the next match), N (moving backward to find a previous match)

- :q (*quitting a file without saving*), :q! (*quitting a file by discarding modification*), :w (*saving a file without quitting the file*), :x (*saving a file and quitting it*)

Vi/Vim Examples

```
Today, we will start with a couple of vi/vim examples.

For the first example, let's go into insertion mode to fix the next sentence:
"This is Comp201-LabX and my name is Y."

For the second example, let's go into visual mode to replace "hate" with "love"
in the next sentence:
"I hate vi/vim!"

That's all for vi/vim examples. Thank you!
~
~
~
~
~
~
~
~
~
~
~
~
~
"vi-examples.txt" 9 lines, 342 characters
```

NOTE: The initial file is available as vi-examples.txt

Bitwise Operations
and
Bit Representation of Integers & Floats

KOÇ UNIVERSITY

Bitwise Operations

- In today's lab practice, you are going to use some bitwise operators.
  - & ^ >> +
  - Examples of bitwise operations:
    - *getting least significant 2 bits of 1110:*
      - 1110 & 0011 = 0010
    - *flipping least significant 2 bits of 1110:*
      - 1110 ^ 0011 = 1101
    - *arithmetic right shifting 1010 by 2 bits:*
      - 1010 >> 2 = 1110
    - *getting the most significant 2 bits of 1010:*
      - (1010 >> 2) & 0011 = 1110 & 0011 = 0010

Bitwise Operations at Byte Level

- *getting the least 4-bits of 0x6e*

  0x6e & 0x0f = 01101110 & 00001111 = 00001110 = 0x0e

- *flipping the least significant 4-bits of 0x6e*

  0x6e ^ 0x0f = 01101110 ^ 00001111 = 01100001 = 0x061

- *arithmetic right shifting 0xee by 4 bits:*

  0xee >> 4 = 11101110 >> 4 = 11111110 = 0xfe

- *getting the most significant 4 bits of 0xe5*

  (0xe5 >> 4) & 0x0f = (11100101 >> 4) & 00001111 = 11111110 & 00001111 = 00001110 = 0x0e

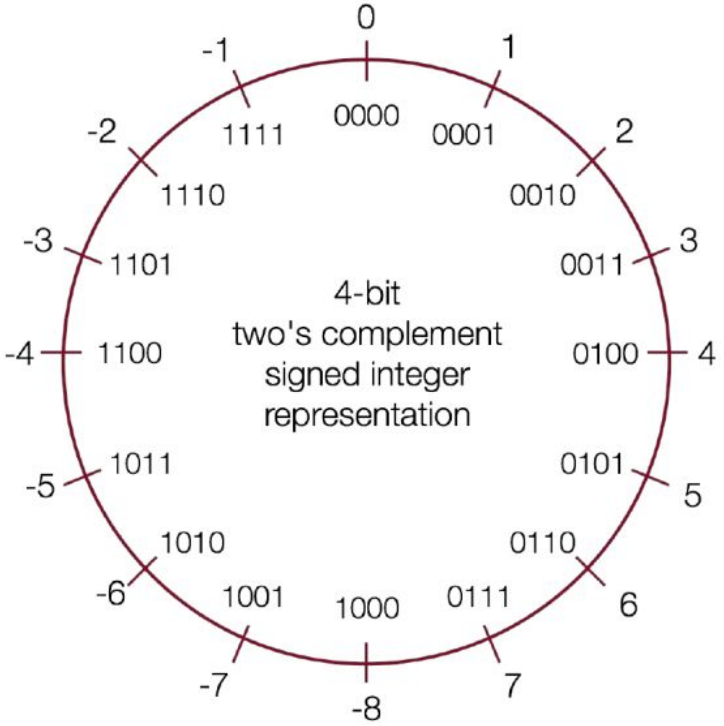
Bitwise Exercise

- **allEvenBits** - return 1 if all even-numbered bits in word set to 1
    - Examples: allEvenBits(0xFFFFFFFE) = 0, allEvenBits(0x55555555) = 1
    - Legal ops: ! ~ & ^ | + << >>

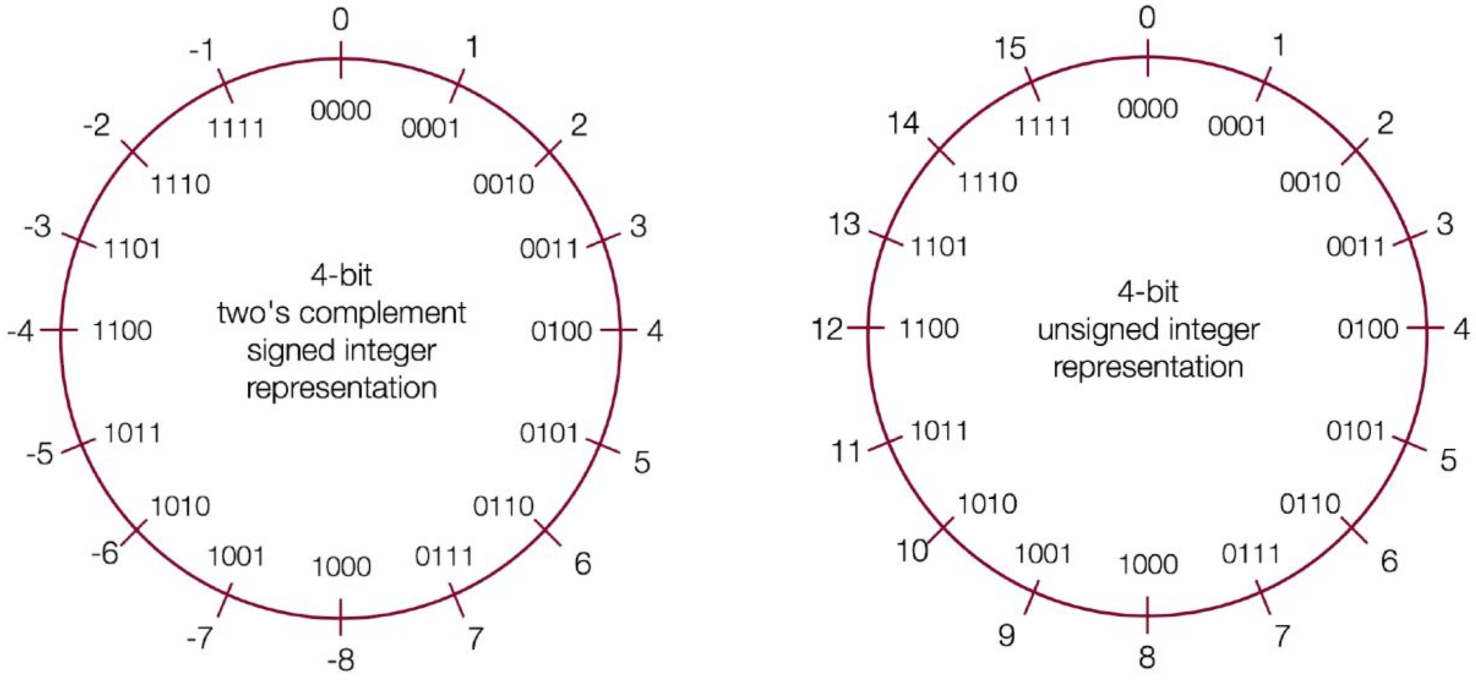
NOTE: The initial code is provided in bits-examples/bits.c. Solutions are available in bits-examples/bits.c-solutions. Testing with “./driver.pl” as Assignment 1.

Two's Complement (Bit Representation of Integers)

- We represent a positive number by itself and a negative number by the two's complement of the corresponding positive number
- The two's complement of a number is the binary digits inverted, plus 1.
  - e.g. -0001 (1) = 1111 (-1)
- Standard addition works
  - E.g. 1111 (-1) + 0001 (1) = 0000 (0)
- All bits are used to represent as many numbers as possible (efficient)



Signed vs Unsigned



Two’s Complement Exercises

- **minusOne** - return a value of -1
  - Example: minusOne() = -1
  - Legal ops: ! ~ & ^ | + << >>
- **fitsShort** - return 1 if x can be represented as a 16-bit, two's complement integer.
  - Examples: fitsShort(33000) = 0, fitsShort(-32768) = 1
  - Legal ops: ! ~ & ^ | + << >>

NOTE: The initial code is provided in bits-examples/bits.c. Solutions are available in bits-examples/bits.c-solutions. Testing with “./driver.pl” as Assignment 1.

Bit Representation of Floating Point Numbers (32-bits)

s	exp	frac
1	8 bits	23 bits

- 1 bit is for sign
- 8 bits are for exponent
- 23 bits are for fraction
- Bias = $2^{(8 - 1)} - 1 = 127$
- How to read:
  - If exp > 0 (normalized), floating point number = (s ? -1 : 1) * (1.frac) * $2^{(exp - 127)}$
  - If exp = 0 (denormalized), floating point number = (s ? -1 : 1) * (0.frac) * 2^{-126}

Bit Representation of Floating Point Numbers (32-bits)

- **Not A Number (NaN):**

Sign	Exponent						Fraction
any	1	...	...	...	...	1	Any nonzero

- **± Infinity (± ∞):**

Sign	Exponent	Fraction
any	All ones	All zeros

- **Zero (0):**

Sign	Exponent	Fraction
any	All zeros	All zeros

Floating Point Exercise

- **float_abs** - Return bit-level equivalent of absolute value of f for floating point argument f.
  - Both the argument and result are passed as unsigned int's, but they are to be interpreted as the bit-level representations of single-precision floating point values.
  - When argument is NaN, return argument.

NOTE: The initial code is provided in bits-examples/bits.c. Solutions are available in bits-examples/bits.c-solutions. Testing with “./driver.pl” as Assignment 1.

InLab Assignment