

# Bits, Ints and Floats, Vim

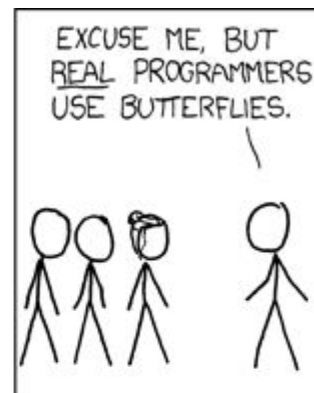
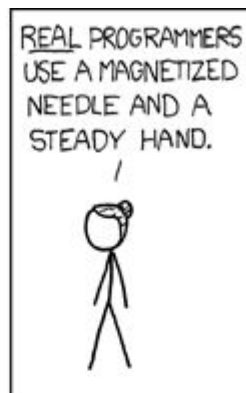
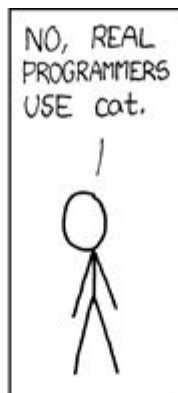
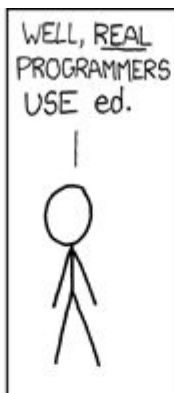
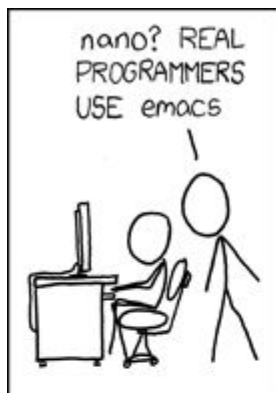
COMP201 Lab 2

Spring 2024



**KOÇ  
UNIVERSITY**

# Vi/Vim Reminder

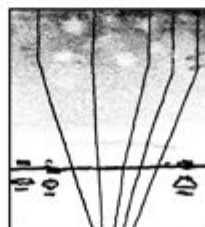


THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.



THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,

WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.





# Vi/Vim Reminder



The screenshot shows a terminal window with a dark background. At the top, there is a status bar with the text "...ce22@linuxpool.ku.edu.tr" on the left, "~ -- ~ -- -fish" in the center, and a "+" icon on the right. Below the status bar, the text "Lab 1. The Linux Shell" and "Lab 2. Manipulating Bi" is visible. The main area of the terminal contains several lines of tilde characters (~). At the bottom of the terminal, the text "-- INSERT --" is displayed on the left, "2,23" in the center, and "All" on the right.

- **Insert mode**
  - Every character you type is put to the file.
  - Cue the **--INSERT--** on the left bottom
  - To switch from normal mode to insert mode, type **i** in the normal mode.
  - To switch back to normal mode, press **esc**



# Basic Commands in Vi/Vim (in Normal Mode)

- **Basic navigation:** Arrow keys
- **Navigating 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)
- **Quitting a file without saving:** :q
- **Quitting a file by discarding modification:** :q!
- **Saving a file without quitting the file:** :w
- **Saving a file and quitting it:** :x

# **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.
  - $\&$   $\wedge$   $\gg$   $+$
  - Examples of bitwise operations:
    - **Getting least significant 2 bits of 1110:**
      - $1110 \& 0011 = 0010$
    - **Flipping least significant 2 bits of 1110:**
      - $1110 \wedge 0011 = 1101$
    - **Arithmetic right shifting 1010 by 2 bits:**
      - $1010 \gg 2 = 1110$
    - **Getting the most significant 2 bits of 1010:**
      - $(1010 \gg 2) \& 0011 = 1110 \& 0011 = 0010$



# 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**

$0x6e \wedge 0x0f = 01101110 \wedge 00001111 = 01100001 = 0x061$

- **Arithmetic right shifting 0xee by 4 bits**

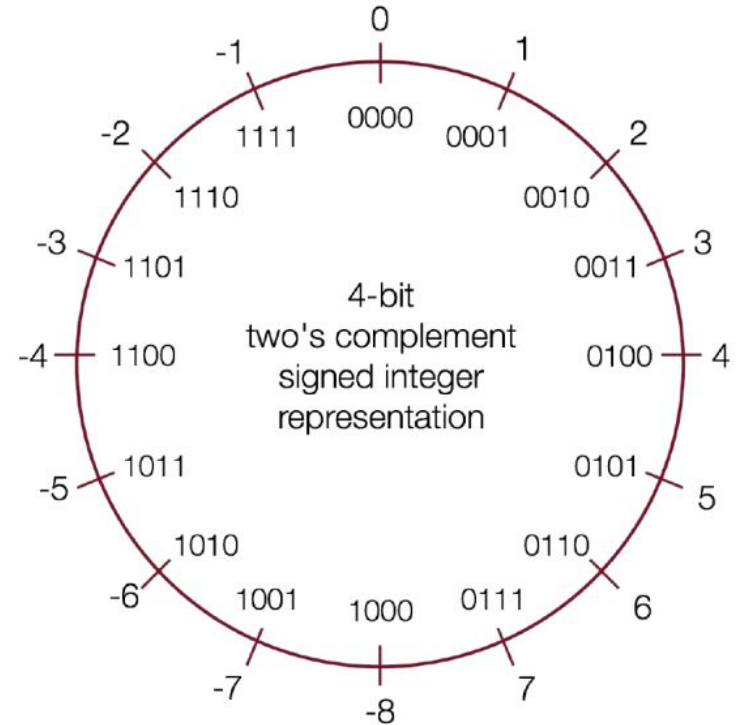
$0xee \gg 4 = 11101110 \gg 4 = 11111110 = 0xfe$

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

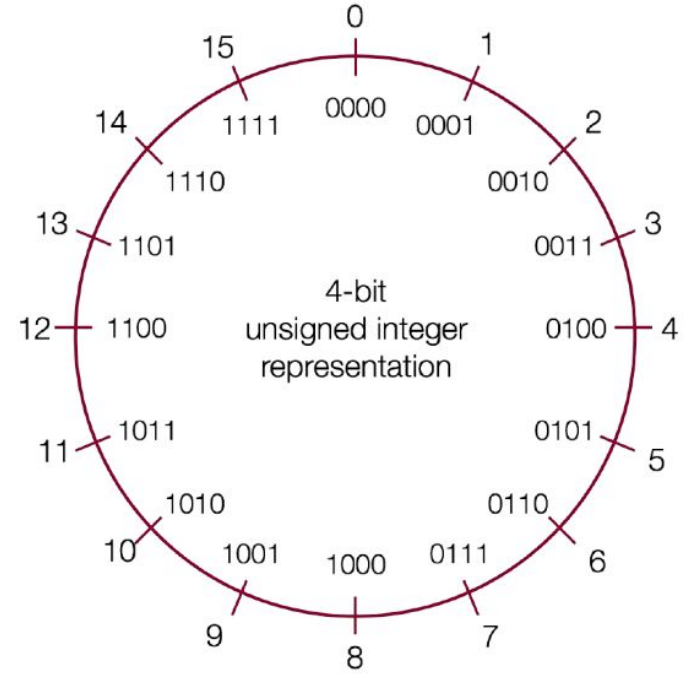
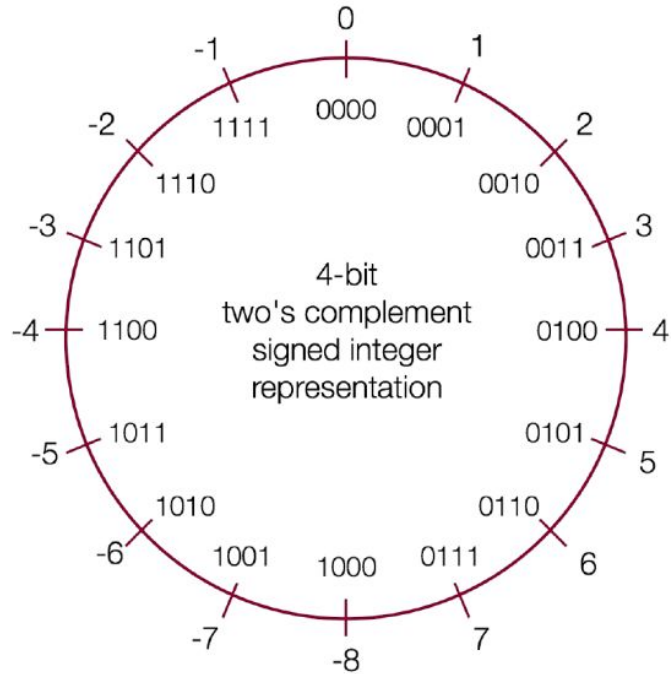
$(0xe5 \gg 4) \& 0x0f = (11100101 \gg 4) \& 00001111 = 11111110 \& 00001111 = 00001110 = 0x0e$

# 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)



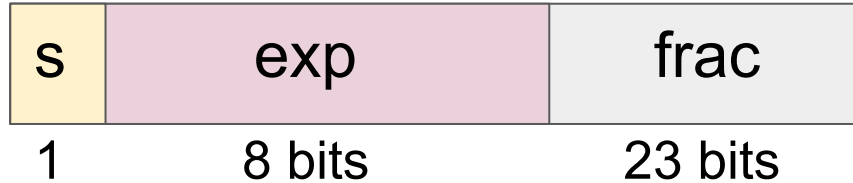
# Signed vs Unsigned



# Two's Complement Exercises

- **minusOne** - return a value of -1
  - Example: `minusOne()` = -1
  - Legal ops: `! ~ & ^ | + << >>`
- **negate** - return -x given x
  - Example: `negate(5)` = -5, `negate(-4)` = 4
  - 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: `! ~ & ^ | + << >>`

# Bit Representation of Floating Point Numbers (32-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  $\text{exp} > 0$  (normalized), floating point number =  $(s ? -1 : 1) * (1.\text{frac}) * 2^{(\text{exp} - 127)}$
  - If  $\text{exp} = 0$  (denormalized), floating point number =  $(s ? -1 : 1) * (0.\text{frac}) * 2^{-126}$

# Bit Representation of Floating Point Numbers (32-bits)

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

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

- **$\pm$  Infinity ( $\pm \infty$ ):**

Sign	Exponent	Fraction
any	All ones	All zeros

- **Zero (0):**

Sign	Exponent	Fraction
any	All zeros	All zeros

**Now, the in lab assignment :)**