Number representation: base conversion; signed integers

COSC 208, Introduction to Computer Systems, 2021-09-10

Announcements

- Project 1 Part A due Thursday at 11pm
- Movie tonight 208L

Outline

- Warm-up
- Hexadecimal
- Binary <-> hex conversion
- Decimal -> binary conversion
- Signed integers

Warm-up

- Q1: *List the powers of two from 2^1 through 2^10*
  - 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
- Q2: *Convert `0b100111` to decimal*
  - 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 32 + 4 + 2 + 1 = `39`

Hexadecimal (i.e., base 16)

- *How many values can be represented with one hexadecimal digit?* - 16
- *How do you count to 13 in hexadecimal?* — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
- Powers of sixteen: 16, 256, 4096
- *Convert these hexadecimal numbers to decimal (i.e., base 10):*
  - Q3: `0x9` = 9 * 16^0 = 9
  - Q4: `0xB` = 11 * 16^0 = 11
  - Q5: `0xF` = 15 * 16^0 = 15
  - Q6: `0x11` = 1 * 16^1 + 1 * 16^0 = 17
  - Q7: `0x248` = 2 * 16^2 + 4 * 16^1 + 8 * 16^0 = 512 + 64 + 8 = 584

Binary <-> Hex Conversion

- *How do you convert from binary to hexadecimal?* — convert each group of four bits to its corresponding hex digit
- *What if the number of binary digits is not a multiple of four?* — pad the front of the binary number with zeros: e.g., `0b11` => `0b0011`
- *How do you convert from hexadecimal to binary?* — convert each hex digit to its corresponding four bits
- *Convert these binary numbers to hexadecimal:*
  - Q8: `0b1010` = `0xA`

    - Q9: `0b1111` = `0xF`
    - Q10: `0b11001100` = `0xCC`
    - Q11: `0b11100111` = `0xE7`
- *Convert these hexadecimal numbers to binary:*
    - Q12: `0x5` = `0b101`
    - Q13: `0x8` = `0b1000`
    - Q14: `0xB` = `0b1011`
    - Q15: `0x37` = `0b00110111`

Decimal -> Binary Conversion

- Why do we care about converting between binary and decimal?
    - We are used to working with decimal numbers, but computers represent numbers in binary
    - Computers allocate a fixed number of bits for different types of variables; mathematical operations whose result exceeds the number of available bits will return unexpected results — we'll talk about overflow (and underflow) on Monday
- Repeated division method
    - Check if number is even or odd: even => 0, odd => 1
    - Build binary number from right to left
    - Divide by two, dropping the fractional part: e.g., 5/2 = 2, 1/2 = 0
    - Repeat, until reach 0
- *What is alternative way to convert from decimal to binary?* — subtract powers of two
- *Convert these decimal numbers to binary:*
    - Q16: `10` = `0b1010`
    - Q17: `15` = `0b1111`
    - Q18: `42` = `0b101010`
    - Q19: `192` = `0b11000000`

Signed integers

- *How can we distinguish between positive values, zero, and negative values?*
- Use a bit to encode the sign --- called signed magnitude
    - *What is an advantage of signed magnitude?*
        - Easy to convert between negative and positive values
    - *What is a disadvantage of signed magnitude?*
        - Positive zero and negative zero
        - Discontinuity between positive and negative values
- Have the highest order bit contribute a negative value to the sum --- called two's complement
    - Example unsigned conversion: `0b0101`
        - 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 1 = 5
    - Example signed conversion: `0b0101`
        - -0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 1 = 5
    - Another example signed conversion: `0b1011`
        - -1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = -8 + 2 + 1 = -5
    - *How many values can be represented using 4 bits?* -- 2^4 = 16
        - One of these values is zero (`0b0000`)
        - *How many positive values an be represented using 4 bits?* 2^4 / 2 - 1 = 7

    - *How many negative values can be represented using 4 bits?* 2^4 /2 = 8
- *Express these decimal numbers using 8-bit two's complement:*
    - Q20: `13` = `0b00001101`
    - Q21: `-128` = `0b10000000`
    - Q22: `-64` = `0b11000000`
    - Q23: `-1` = `0b11111111`
    - Q24: `-13` = `0b11110011`
    - Q25: `127` = `0b01111111`

Extra practice

- *Convert these binary numbers to decimal:*
    - Q26: `0b1111` = 15
    - Q27: `0b10100` = 20
    - Q28: `0b101000` = 40
- *Convert these hexadecimal numbers to decimal:*
    - Q29: `0xC` = 12
    - Q30: `0x18` = 24
    - Q31: `0x30` = 48
- Q32: *Write a function called* `abbreviate` *that takes a string and modifies the string in place to include only the first letter of each word. For example,* `"Talk To You Later"` *is converted to* `TTYL`.

```
void abbreviate(char str[]) {
    int store = 1;
    int check = 1;
    while (check < strlen(str)) {
        if (str[check-1] == ' ') {
            str[store] = str[check];
            store++;
        }
        check++;
    }
    str[store] = '\0';
}
```

- Q33: *Write a function called* `check_password` *that returns 1 if a password is at least 8 characters long and contains at least one uppercase letter, at least one lowercase letter, and at least one digit. Otherwise, the function returns 0. You may want to use the functions* `isupper`, `islower`, *and* `isdigit`. *They take a character as a parameter and return 1 if the character is an uppercase letter, lowercase letter, or digit, respectively; otherwise, they return 0.*

```
int check_password(char passwd[]) {
    if (strlen(passwd) < 8) {
        return 0;
    }
    int lower = 0;
    int upper = 0;
```

```
    int digit = 0;
    for (int i = 0; i < strlen(passwd); i++) {
        if (islower(passwd[i])) {
            lower++;
        }
        else if (isupper(passwd[i])) {
            upper++;
        }
        else if (isdigit(passwd[i])) {
            digit++;
        }
    }
    if (lower == 0 || upper == 0 || digit == 0) {
        return 0;
    }
    return 1;
}
```