Exam 1 Review; bitwise operators

COSC 208, Introduction to Computer Systems, 2021-02-15

Announcements

- Exam 1: next week -- tutor hours 6:30 to 8:30 TW evenings
- Project 1 Part A due Thursday at 11pm

Outline

- Exam 1 review
- Bitwise in between vs. other operators

Binary arithmetic

Perform the following calculations. Operands are be encoded using two's complement encoding with 6 bits. For each calculation, express the result in binary and decimal, and indicate whether the result overflows, underflows, or neither.

- Q1: `0b110000 + 0b111111 = 0b101111 = -17`; neither
- `0b001111 + 0b000001 = 0b010000 = 16`; neither
- `0b101010 + 0b100100 = 0b001110 = 14`; underflow
- `0b001000 + 0b011000 = 0b100000 = -32`; overflow
- `0b110000 + 0b010000 = 0b000000 = 0`; neither

Number base conversions

Perform the following conversions

- `97` to 8-bit unsigned binary = `0b01100001`
- `-42` to 8-bit two's complement = `0b11010110`
- `0b11001100` to unsigned decimal = `128 + 64 + 8 + 4` = `204`
- `0b11001100` to signed decimal = `-52`
- `0x27` to unsigned decimal = `39`
- Q11: `0xDEAD` to 16-bit binary = `0b1101111010101101`

Bitwise operators

- Apply an operation to a single bit (not) or a pair of bits (and, or, xor)
- `~` (not)
    - Flips bits: if bit is 0, then result is 1; otherwise, result is 0
    - Example: `~0b101` = `0b010`
- `&` (and)
    - If both bits are 1, then result is 1; otherwise, result is 0
    - Example: `0b0101 & 0b0011` = `0b0001`
    - This is different than logical and (`&&`) which checks whether both operands are non-zero
        - 0b10 && 0b01 -> true (because both non-zero)
        - 0b11 && 0b00 -> false
- `|` (or)
    - If either or both bits are 1, then result is 1; otherwise, result is 0
    - Example: `0b0101 | 0b0011` = `0b0111`
    - This is different than logical or (`||`) which checks whether at least one operand is non-zero

- `^` (xor)
    - If either, but not both, bits are 1, then result is 1; otherwise, result is 0
    - Example: `0b0101 | 0b0011` = `0b0110`
- Q12: `0b1010 | 0b0101` = `0b1111`
- Q13: `0b1010 & 0b0101` = `0b0000`
- Q14: `~(0b1100 & 0b0110)` = `~0b0100` = `0b1011`
- Q15: `0b1000 >> 0b011` = `0b0001` (divide by 2^3)
- Q16: `0b0001 << 0b0010` = `0b0100` (multiply by 2^2)
- `<<` (left shift), `>>` (right shift)
    - Move bits to the left or the right and append or prepend zeros to keep the same number of bits
    - Example: `0b1111 << 0b0010` = `0b1100`
    - Example: `0b1111 >> 0b0001` = `0b0111`
    - Can use bit shifting to multiply or divide by powers of two
- Practice
    - Q16: `0b1111 & (~0b0010)` = `0b1111 & 0b1101` = `0b1101` (clear a bit)
    - Q18: `0b0000 | 0b0010` = `0b0010` (set a bit)

Logical & bitwise operators

For each of the following expressions, select all operators that make the expression evaluate to true. Operands are encoded using two's complement.

- `0b110000 __ 0b111111` — `&`, `&&`, `|`, `||`, `^`, `<`
- `0b011110 __ 0b000001` — `&&`, `|`, `||`, `^`, `>`
- `0b000000 __ 0b000000` — *none*
- `0b000111 __ 0b000111` — `&`, `&&`, `|`, `||`

Strings

- *The following program should ask the user to enter a word, then print the word's length and whether it is a palindrome (i.e., reads the same backward as forward). For example, if the user enters `"kayak"` the program should print `"The word is 5 characters long and is a palindrome."` However, the program contains several errors. Modify the program to correct the errors.*

```c
#include <stdio.h>

void palindrome(char word[]) {
    int i = 0;
    int j = strlen(word);
    while (i < j) {
        if (word[i] != word[j]) {
            return -1;
        }
        i++;
        j--;
    }
    return 1;
}

int main() {
    printf("Enter a word: ");
    char word[50];
    fgets(word, 50, stdin);
    word[strlen(word)-1] = '\0'; // Remove newline
    int len = strlen(word);
    printf("The word is %c characters long and is ", len);
    if (palindrome(word)) {
        printf("a palindrome.\n");
    } else {
        printf("not a palindrome.\n");
    }
}
```

  - `#include <string.h>`
  - Change return type of `palindrome` to `int`
  - Initialize `j` to `strlen(word) - 1`
  - Change `return -1` to `return 0`
  - Change `%c` in 2nd `printf` in `main` to `%d`

- *Write a function called `molecular_formula` that takes a string containing the constituent atoms of a molecule and updates the string to contain the molecular formula. For example, the string `"HHO"` should be changed to `"H2O"`, and the string `"HHSOOOO"` should be changed to `"H2SO4"`. You can assume:*
    - *Molecules will only contain elements that are represented by a single letter — e.g., a molecule may contain `'H'` but not `"Na"`*
    - *All atoms of the same element are listed consecutively — e.g., the constituent atoms may be provided as `"HHO"` but not `"HOH"`*
    - *The elements are listed in the order they should appear in the molecular formula — e.g., the constituent atoms `"HHO"` are changed to the molecular formula `"H2O"`, whereas the constituent atoms `"OHH"` are changed to the molecular formula `"OH2"`*
    - *There will be at most 9 atoms of each element — e.g., `"H9C9"` may occur, but `"H10C11"` will not occur*

```
void molecular_formula(char elements[]) {
    int count = 1;
    int j = 0;
    for (int i = 1; i <= strlen(elements); i++) {
        if (elements[i] != elements[i-1]) {
            elements[j] = elements[i-1];
            j++;
            if (count > 1) {
                elements[j] = count + '0';
                j++;
            }
            count = 0;
        }
        count++;
    }
    elements[j] = '\0';
}
```

Structs

- *Define a struct for representing a chemical element, which includes the element's:*
  - *Name*
  - *Chemical symbol*
  - *Atomic number*
  - *State (solid, liquid, or gas) at room temperature*

```c
struct element {
    char name[20];
    char symbol[3];
    int number;
    char state;
};
```

- *Write a function called `lookup` that takes a chemical symbol and an array containing a struct for each of the 118 elements in the periodic table. The function should return the specified element's atomic number. If the provided symbol does not correspond to a known element, the function should return -1.*

```c
int lookup(char symbol[], struct element table[]) {
    for (int i = 0; i < 118; i++) {
        if (strcmp(symbol, table[i].symbol) == 0) {
            return table[i].number;
        }
    }
    return -1;
}
```