UNSW

Sydney

School of Computer Science and Engineering

Summary of COMP1521 - Computer System Fundamentals

Term 3 - 2020

September, 2020

Written by Zheng Luo
z5206267@ad.unsw.edu.au

Contents

Chapter 1

Integer and Bitwise Operation

1.1 Week1 Notes

1. Binary Representation
   - Base 2
   - $1011 == 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 == 11$
   - Each digit represents 1 bit in binary
2. Hexadecimal Representation
   - Base 16
   - Digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
   - Each digit represents 4 bits in binary
   - 0x4A == 0100 1010 == 74
3. Bits and Bytes Conversion
   - 1 byte == 8 bits
   - char == 1 bytes == 8 bits
   - int == 4 bytes == 32 bits
4. Bitwise Operation
   - AND&: Only two '1' will result '1'
   - OR|: Any '1' will result '1'
   - NEG$\sim$: Opposite
   - XOR$\wedge$: Same number result '0', different number result '1'

- Left shift $<<$
- Right shift $>>$

5. Two's Complement
    - How to represent negative numerical value by binary number via human brain (No need to apply this logic in the program, as computer knows).
    - First, obtain the binary expression based on its position numerical value
    - Second, perform NEG operation to the binary expression
    - Last, plus(if positive to negative)/minus(if negative to positive) binary 1 to the binary expression
6. Mask
    - In computer science, a mask or bitmask is data that is used for bitwise operations, particularly in a bit field. Using a mask, multiple bits in a byte, nibble, word etc. can be set either on, off or inverted from on to off (or vice versa) in a single bitwise operation.
    - If we have a 32 bits binary number, we only want to see last 3 digits or the digits between 3 to 5.
    - Then we could initial a variable(mask) to be 0000111(first case)/ 0011100(second case) and do AND operation with our 32 bits number.

1.2 Week1 Coding Sample

1. Size, min and max values of different integer types. (int, char etc.) (integer_types.c)
2. Converting numerical expression into binary expression. (print_bits.c)
    (a) Looping from maximum bit value until zero
    (b) Right shift counter amount, shifting the most LHS bit to the most RHS
    (c) Do AND operation with 1, to determine the bit value at most RHS
    (d) Print out the determined value one by one in the loop
3. The detailed list of corresponding binary numbers from -128 to 127. (eight_bit_twos_complement.c)
4. Mask: shift operators and subtraction to obtain a bit pattern of n 1s. (set_low_bits.c) (set_bit_range.c) (extract_bit_range.c)
    (a) Initiate mask as 1
    (b) Left shift the mask for desired amount

(c) minus 1 to the mask

(d) (left shift again if asking range mask)

(e) Then do the AND operation for the rest

5. Converting numerical expression into hexadecimal expression via putchar.

   (a) Looping from max bit value to zero

   (b) But bitShifted is counter * 4, since 4 binary bits equal to 1 hex bit.

   (c) mask off (zero) all bits but the bottom 4 bites by AND 0xF(00001111).

   (d) Convert 0-16 in terms of hex format:
   int hex_digit_ascii = "0123456789ABCDEF"[hex_digit];

   (e) putchar().

6. Converting numerical expression into hexadecimal expression via storing into a string. (int_to_hex_string.c)

   (a) Allocate memory to hold the hex digits + a terminating 0.

   (b) Looping from zero to max bit value.

   (c) Then same process as in putchar().

   (d) Store hex_digit_ascii into string[counter].

   (e) After the end of the loop, add 0 to terminate the array.

   (f) Print string and free allocated memory.

7. Converting hexadecimal expression in string into numerical expression. (hex_string_to_int.c)

   (a) Looping through inside of string until string[counter] != NULL.

   (b) Convert each character from ASCII value to binary value.

   (c) assign this 4 bits value into a final value via OR operation.

   (d) left shift 4 places each time when processing each character.