CPE390 Worksheet: Base Arithmetic, and Boolean Operations

| decimal | hex (16) | octal (8) | binary |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0000 |
| 1 | 1 | 1 | 0001 |
| 2 | 2 | 2 | 0010 |
| 3 | 3 | 3 | 0011 |
| 4 | 4 | 4 | 0100 |
| 5 | 5 | 5 | 0101 |
| 6 | 6 | 6 | 0110 |
| 7 | 7 | 7 | 0111 |
| 8 | 8 | 10 | 1000 |
| 9 | 9 | 11 | 1001 |
| 10 | A | 12 | 1010 |
| 11 | B | 13 | 1011 |
| 12 | C | 14 | 1100 |
| 13 | D | 15 | 1101 |
| 14 | E | 16 | 1110 |
| 15 | F | 17 | 1111 |

Convert from binary to decimal

Example: Binary to decimal

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

=128+16+8+1=153

1. convert 11010110 to decimal \_\_\_\_\_
2. convert 10110101 to decimal \_\_\_\_\_

Convert from decimal to binary

subtract highest power of 2 from the number

compute remainder

write 0 any time power of 2 does not fit

Example: decimal to binary  
236 = 128 + 64 + 32 + (12 remain) 0 + 8 +4 + 0 + 0  
= 1 1 1 0 1 1 0 0  
3. convert 255 to binary \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. convert 126 to binary \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. convert 73 to binary \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5b. (knowing 65536=216) convert 65535 to binary

C++ base notation

| decimal | hex | octal | binary |
| --- | --- | --- | --- |
| 123 | 0x7B | 0776 | 0b10110101011 |

Converting hex and octal to binary

12b. Each digit -> 3 or 4 binary digits (see table)

0xFE2C109A = 1111 1110 0010 1100 0001 0000 1001 1010  
0776 = 111111110  
  
mnemonics: A is 10 = 1010 B is 11 = 1011

6. Convert 0xDEADBEEF = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Convert 0xF001EAC4 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. A multiple of 4 in hex will always end in digits \_\_\_\_\_\_

9. A multiple of 8 in hex will always end in digits \_\_\_\_\_\_

Boolean Operations

Note: &&, || are LOGICAL and and or (different)

| AND | & | and x0, x1,x2 |
| --- | --- | --- |
| OR | | | orr x19, x2, x7 |
| XOR | ^ | eor x20, x2, x5 |
| NOT | ~ | movn x0, x1 |
| unsigned left shift | a << 3 | lsl x0, 3 |
| unsigned right shift | a >> 4 | lsr x0, 4 |
| signed left shift | b << 7 | asl x0, 7 |
| signed right shift | b >> 5 | asr x0, 5 |
| rotate right |  | ror x0, 3 |

10. 0xFEED9008 & 0x49A7C5D2 =

11. 0x12345678 | 0x7C43AE2C

12. 0x12345678 ^ 0x7C43AE2C =

Identify binary in the registers

mov x0, 136 // x0=10001000  
mov x1, 215 // x1=11010111  
eor x2, x0, x1 // x2=01011111 = 0x5F  
orr x3, x0, x1 // x3=11011111 = 0xDF  
eor x4, x2, x2 // x4=0  
and x5, x0, x1 // x5=10000000 = 0x80  
movn x6, x0 // x6=0xFFFFFFFFFFFFFF77

Bit Masking

It is possible to surgically modify individual bits in words using bitwise operations

set means = 1, clear means = 0

| set bit  OR with 1 | 010x0101  00010000 | a | 0x10 |
| --- | --- | --- |
| clear bit  AND with 0 | 011x1001 11101111 | a & 0xEF |
| toggle bit  XOR with 1 | 1010x101 00001000 | a ^ 0x08 |
| test is bit 1? AND with 1 | 1x001100 01000000 | a & 0x40 |
| mult bits all 1? AND | 10xx01x0 00110010 | a & 0x32 == 0x32 |
| mult bits any 1? AND | 10xx01x0 00110010 | (a & 0x32) != 0 |

if (a & mask)

tst x0, MASK

bne ???

13. Write C++ to set all bits marked x.

1. a=100x 0101 1001 1100
2. a=10x0 xx00 011x 0000
3. a=xxxx xx00 1011 1010

14. Write C++ to clear all bits marked x.

1. a=1010 0xx0 x000 0101
2. a=1100 11xx 0x0x x000

15. Write C++ to toggle all bits x.

1. a=1010 0xx0 x000 0101
2. a=1100 11xx 0x0x x000

16. Write the C++ hex constant for all 1 bits:  
a. 32 bits 0x\_\_\_\_\_\_\_\_\_\_

b. 64 bits 0x\_\_\_\_\_\_\_\_\_\_

How can we clear the low 3 bits of a number, rounding down to the nearest multiple of 8? This is slow:  
x = x - x%8

17. Write ARM assembler sequences to compute the examples from the masking table

1. set bit
2. clear bit
3. toggle bit
4. test if bit is true and branch to f

18. Write C++ functions to

1. int countbits(uint64\_t a);
2. uint64\_t setbit(uint64\_t v, uint32\_t pos);
3. uint64\_t clearbit(uint64\_t v, uint32\_t pos);

Future reference

1. Write a function to use rubberbanding (xor mode) to draw a horizontal line:  
     
   horiz\_line\_xor(uint32\_t x1, uint32\_t x2, uint32\_t y, uint32\_t color);