Homework Buddy :

Lab 02: Binary Arithmetic and MIPS

An Instruction to Assembly Programming for the MIPS Architecture

You may collaborate on this homework with at most one person, an optional "homework buddy". You must turn in your own version of this lab, but you may discuss answers and approaches in detail. Projects should be turned in using Gradescope as a PDF (please do not change the formatting of the PDF, just add your answers). Due Friday at 5 pm.

Binary Subtraction

The following problems ask you to subtract one binary number from another. While this can be done directly, it is typically easier to negate the second number (using the 2's complement method), and then add the two together. You must express the answer in 8 bits. There will be several steps but only show the answer. You will also need to specify the overflow (V) and carry (C) bits.

1. (2 pts) What is

```
     10001101
   - 01011010
```

00110011, C=1, V=1

2. (2 pts) What is

```
     11001001
   - 00110101
```

10010100, C=1, V=0

3. (2 pts) What is

```
   10110111
 - 10000101
```

```
   00110010, C=1, V=0
```

4. (2 pts) What is

```
   01010111
 - 10100001
```

```
   10110110, C=0, V=1
```

More Bitwise Operations

Recall that bitwise AND is specified with `&`, bitwise OR with `|`, and bitwise XOR with `^`.

5. (2 pts) What is

```
   10011001
 & 01011100
```

```
   00011000
```

6. (2 pts) What is

```
   11001100
 & 01010101
```

```
   01000100
```

7. (2 pts) What is

```
    10101111
  | 11110101
```

```
    11111111
```

8. (2 pts) What is

```
    11011011
  ^ 10100001
```

```
    01111010
```

Give your answers for the following set in 2-digit hexadecimal. Recall that bitwise AND is specified with `&`, bitwise OR with `|`, and bitwise XOR with `^`.

9. (2 pts) Calculate 0xF7 & 0x10

11111111 & 00010000

Answer: 0x10

10. (2 pts) Calculate 0xAF & 0xFD

10101111 & 11111101

Answer: 0xAD

11. (2 pts) Calculate 0x55 | 0xFF

01010101 | 11111111

Answer: 0xFF

12. (2 pts) Calculate 0xCA | 0xAA

11001010 | 10101010

Answer: 0xEA

13. (2 pts) Calculate 0x6C ^ 0xF9

01101100 ^ 11111001

Answer: 0x95

14. (2 pts) Calculate 0xAD ^ 0xBB

10101101 ^ 10111011

Answer: 0x16

Shift Left and Shift Right Operations

Give your answers for the following set in **8-bit binary**.

15. (2 pts) What is 10000110 >> 3 for logical shift right?

00010000

16. (2 pts) What is 10110110 >> 1 for arithmetic shift right?

11011011

17. (2 pts) What is 01100001 >> 2 for logical shift right?

00011000

18. (2 pts) What is 01101011 >> 2 for arithmetic shift right?

00011010

19. (2 pts) What is 10011101 << 2 for shift left?

01110100

20. (2 pts) What is 01101000 << 1 for shift left?

11010000

Bit Masking

For the following three questions, remember that bit numbers start numbering from 0. Assume that these hexadecimal numbers are always unsigned. In addition to showing the mask you would use, also show the operation you would use, like: & **0x0020**. (i.e., AND the original number with the hexadecimal mask 0x0020.)

(2 pts) Specify the operation and mask you would need to inspect **bit 9** of the unknown number. Express the mask as a 4-digit hexadecimal number (i.e. in the form of 0xhhhh).

& 0x0200

(2 pts) Specify the operation and mask you would need to set just **bit 9** of the unknown number to zero. That is, the result of this operation results in a new number, which the unknown number should be subsequently set to. Again, express the mask as a 4-digit hexadecimal number.

& 0xFDFF

(2 pts) Specify the operation and mask you would need to set **bit 9** of the unknown number to one. That is, the result of this operation results in a new number, which the unknown number should be subsequently set to. Express it as a 4-digit hexadecimal number.

| 0x0200

Bit Manipulation in C/C++

(5 pts) In 1 sentence, explain how you would change **bit 12** in a 32-bit number to be 0? Based on your explanation, complete this C/C++ function that takes an integer (i.e. signed 32-bits), changes its bit 12 (remember that the least significant bit is bit 0) to 0, and returns the new integer. **You can ONLY use a single return statement, declare auxiliary variables, use assignment (=), use constants, and use bitwise operations (>>, <<, &, |, ^, ~).**

Explanation:

Start with an integer 1, and move that bit over 12 times (to the twelfth position), and then xor with -1. Lastly, I take that integer and use the AND operator on it with the original integer.

```
int setBit12to0(int v){
    int mask = ~(1 << 12);
    return v & mask;
}
```

(5 pts) In 1 sentence, explain how you would change **bit 7** in a 32-bit number to be the inverse of what it was originally (i.e. 1➜0 and 0➜1). Based on your explanation, complete this C/C++ function that takes an integer (i.e. signed 32-bits), sets its bit 7 its inverse, and returns the new integer. **You can ONLY use a single return statement, declare auxiliary variables, use assignment (=), use constants, and use bitwise operations (>>, <<, &, |, ^, ~).**

Explanation:

Start with an integer 1, and move that bit over 7 times (to the seventh position), and then xor with v.

```
int flipBit7(int v){
    int mask = 1 << 7;
    return v ^ mask;
}
```

Bitwise Operations in MIPS Assembly

(5 pts) Complete this C/C++ function that takes an integer and returns a 1 if **bit 9** is a 1, or otherwise returns a 0. **You can ONLY use a single return statement, declare auxiliary variables, use assignment (=), use constants, and use bitwise operations (>>, <<, &, |, ^, ~).**

```
int ifBit9is1(int v){
```

```
int mask = 1 << 9;
return (v & mask) >> 9;
}
```

(5 pts) Complete this C/C++ function that takes an integer and returns its bits 10 through 15 as an integer value. For example, if bits 10 through 15 is 110010 in binary, you should return the corresponding integer value, which is 50. **You can ONLY use a single return statement, declare auxiliary variables, use assignment (=), use constants, and use bitwise operations (>>, <<, &, |, ^, ~).**

```
int signedBits10through15(int v){
int mask = (1 << 6) - 1;
mask = mask << 10;
return (v & mask)>> 10;
}
```

(8 pts) Complete the MIPS assembly language program below that is supposed to take the values in **$t0, $t1, and $t2** and first calculate the value of **($t2 - $t1 + $t0)**, then logic **XOR** it to the value of **($t1 + $t0)**. The entire calculated value should be placed in **$t3**. Finally, the program should print the value of **$t3** to standard output and exit correctly. You should try it out and test it in SPIM to be sure it has the correct syntax.

.text

```
main:
     li $t0, 42
     li $t1, 83
     li $t2, 214
 # To-Do: the rest of the program here (don't forget to exit properly)!
```

```
sub $t4, $t2, $t1
add $t5, $t4, $t0
add $t6, $t1, $t0
xor $t3, $t5, $t6
li $v0, 1
move $a0, $t3
syscall
li $v0, 10
syscall
```

(2pts) What is the value of **$t3** (in 32-bit hexadecimal) after the above program is executed? Show it with a hand-calculation (i.e. not by running the program only, but by calculating it yourself).

214-83+42 = 173 = 10101101
83+42 = 125 = 01111101

173 ^ 125 = 11010000

$t3 = 0x000000D0