Assignment_2 Solution (Prof. N. Chen)

2.10
Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.

2.10.1
What is the value of $t0 for the following assembly code?
add $t0, $s0, $s1

2.10.1 0x50000000

(Prof. Chen's note: if you look at the most significant hex numbers, you have 0x8+0xd)

2.10.2
Is the result in `$t0` the desired result, or has there been overflow?

2.10.2 overflow

2.10.3
For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?
```
sub $t0, $s0, $s1
```

2.10.3 0xB0000000

Prof. Chen's note: This question is a bit tricky. The way to understand it is that the hardware uses the add mechanism to perform a subtraction. How? What we want here is to subtract $s1, which is equivalent to adding a negated $s1. Given that $s1 has a value of 0xd0000000. What is a negated $s1 value? Knowing questions is more important than knowing answers!

D -> 1101 0000 … 0000

Flip all bits -> 0010 1111 … 1111

Add one -> 0011 0000 … 0000

Now add 80000000 (hex) to 0011 0000 … 0000 -> 1000 0000 … 0000 + 0011 0000 … 0000 = 1011 0000 … 0000 -> B0000000 (hex)

2.10.4
Is the result in `$t0` the desired result, or has there been overflow?

2.10.4 no overflow

2.10.5
For the contents of registers `$s0` and `$s1`as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
add $t0, $t0, $s0
```

2.10.5

add $t0, $s0, $s1 gives us 0x5000000

add $t0, $t0, $s0 means 0x50000000 + 0x80000000

The result is

0xD0000000

2.10.6
Is the result in `$t0` the desired result, or has there been overflow?

2.10.6 overflow

2.12
Provide the type and assembly language instruction for the following binary value:

`0000 0010 0001 0000 1000 0000 0010 0000`$_{two}$

2.12 r-type, add $s0, $s0, $s0

Prof. Chen's note: need to learn how to read the green sheet

2.13
Provide the type and hexadecimal representation of following instruction:

`sw $t1, 32($t2)`
2.13

i-type, 0xAD490020

Prof. Chen's note: need to learn how to read the green sheet

2.22.2

Suppose the program counter (PC) is set to `0x2000 0000`.
What range of addresses can be reached using the MIPS branch if equal (beq) instructions? (In other words, what is the set of possible values for the PC after the branch instruction executes?)
2.22.2

To be provided in the group discussion.

Extra 1:

Write the MIPS assembly code that creates the 32-bit constant `0010 0000 0000 0001 0100 1001 0010 0100`$_{two}$ and stores that value to register `$t1`.

lui $t1, most_significant_16_bits

ori $t1, $t1, least_significant_16_bits

(Note: LUI – Load Upper Immediate; ORI – bitwise OR Immediate)

Extra 2:

If the current value of the PC is `0x00000000`, can you use a single jump instruction to get to the PC address as shown in Extra 1?

No. Reason: again, pay attention to the root cause of the problem (don't focus on how to fix the problem right from the beginning; if you don't understand the root cause, don't bother to fix it!)