CSCI 4593 Spring 2014 **Vince Coghlan**

Homework #1

2.3)

```
sub $t5, $s3, $s4;
add $t5, $t5, $t5;
add $t5, $t5, $t5;
lw $t0, $t5($s6);
addi $t5, $zero, 8;
add $t5, $t5, $t5;
add $t5, $t5, $t5;
sw $t5($s7), $t0;
```

2.7) From low memory to high memory, the value would look like:

AB	CD	EF	12

Big Endian

12	EF	CD	AB

Little Endian

2.11)

inst	opcode	rs	rt	rd	immed
addi	0x8	0x16	0x8	-	0x4
add	0x0	0x16	0x0	0x9	-
sw	0x2b	0x8	0x9	-	0x0
lw	0x23	0x8	0x8	-	0x0
add	0x0	0x9	0x8	0x10	-

2.12.3) The answer should be positive, since we are taking a negative number, and adding something larger to it, like so:

$$\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

Which is equivalent to 0xB0000000.

2.12.4) Yes, there has been overflow, since the answer we are looking for should be positive, and this is negative.

2.14) This is equivalent to add $s0, $s0, $s0, basically double the contents of $s0. Its an R type instruction.

2.15) This is equivalent to 0xAD490020, an I type instruction.

2.17) This is equivalent to lw $v0, 4($at), an I typer instruction. also can be 0x8C220004

2.19.2) The value will be the same as $t1, or 0x12345678, this is because a shift by 44 will clear $t0 to 0, when you logical or, you just get $t1.

2.23) The first instruction tests if $t0 is greater than 0, which it is so $t2 becomes 1. bne will break to ELSE. The final result will be $t2+2, or 3.

2.24) j can take an immidiate value of up to 26 bits, we multiply this value by 4, and then concatonate the first four bits from the PC counter, this means that we cannot change the first four bits of the PC, and cannot jump that far. beq will allow us to use an offset value of 16 bits. we will multiply this by four, and add to PC+4. This does not even come close to a 32 bit offset, and once again we can not change the PC by this much.

2.26.1) $s2 acts like a counter inside a for loop, this code can be translated to the following C code:

```
$s2 = 0;
for($t1 = 10;$t1 > 0;$t1--) {
  $s2 = $s2 + 2;
}
```

When ran this code outputs a value of 20 for s2.

2.38) lbu is going to load the value of 0c11223344 into register $t0. That value is then stored at location 0x10000010. it is then read from $t2, we would read the exact same value, 0x11223344, since our memory looks like:

11	22	11	22	33	44

2.40) No, since we only have 26 bits, and that is a 30 bit value. The first 4 bits of the address will always be 0 because that's how big our PC is.