Solutions to Homework 2

Problems from the text.

2.3

```
sub  $t1, $s3, $s4     # $t1 holds index
sll  $t1, $t1, 2       # mult by 4 to compute offset
add  $s9, $s6, $t1     # $s9 now holds base + offset
lw   $t4, 0 ($s9)      # get the item from the array
sw   $t4, 32 ($s7)     # store the item into new array at index 8
```

2.4

```
B[g] = A[f] + A[f+1];
```

2.5

Byte:	0	1	2	3
Big-endian:	ab	cd	ef	12
Little-endian:	12	ef	cd	ab

2.10.1

The first thing to note is that the values are represented in *64* bit registers (remember there are 4 bits/hex digit)

In hexadecimal:

```
 0x8000 0000 0000 0000
+0xD000 0000 0000 0000
----------------------
 0x5000 0000 0000 0000
```

There is a 1 that is carried out.

In binary:

```
 1000 0000 0000 ------- 0
+1101 0000 0000 ------- 0
-------------------------
 0101 0000 0000 ------- 0
```

As before, a 1 is carried out. Also, in binary, you can see the change in sign (positive) compared to the operands (both negative).

2.10.2

Since a 1 was carried out and/or there is a change in sign, there is overflow.

2.10.3

You really have to do this one in binary, because you have to use two's complement:

```
 1000 0000 0000 ------- 0   =   1000 0000 0000 ------ 0
-1101 0000 0000 ------- 0   =  +0011 0000 0000 ------ 0
-------------------------      ------------------------
                                1011 0000 0000 ------ 0

                              0x  B    0    0  ------ 0
```

2.10.4

No overflow.

2.19

```
nor  $t1, $t2, $t2
```

or

```
nor  $t1, $t2, $zero
```

2.20

```
lw  $t1, 0($s0)
sll $t1, $t1, 4
```

Additional problems.

1. Show MIPS for `x = A [ B [x] ];`

```
lw  $t0, x
sll $t0, $t0, 2
lw  $t1, B ($t0)
sll $t1, $t1, 2
lw  $t2, A ($t1)
sw  $t2, x
```

2. Show MIPS for `A[4] = A[5] + a;`, where the address of A is $18,875,010_{10}$ and `a` is stored in `$t3`.

18,875,010 does not fit into the 16 bit format field for `lw`. Thus, the address has to be stored in a register (`$t2`, in this case). Note that `la` (load address) could work, but it is a pseudoinstruction, and thus not available in MIPS. If you convert the number to binary, the upper (left) 16 bits work out to 288_{10}, while the lower (right) 16 bits work out to 642_{10}. Now you have to get each half into the same register.

```
lui $t2, 288            # load upper "half" of the number
ori $t2, $t2, 642       # "add in" the "lower" half of the number
                        # $t2 now holds base address
lw  $t1, 20 ($t2)       # now you can do offset normally; t1 is A[5]
add $t1, $t1, $t3       # add A[5] + a
sw  $t1, 16 ($t2)       # store in A[4]
```

3. Show the MIPS instruction for `b = 45 & a;`

   This is a bitwise `or` operator.

```
lw   $t1, a
ori  $t2, $t1, 45
sw   $t2, b
```

4. Give the binary instruction for `sw $s1, number($t2)`, where the address of `number` is 132.

op code	rs	rt	offset
0x2b	$t2	$s1	132
43	10	17	132
101011	01010	10001	0000000010000100

5. Write the binary instruction for `sllv $t1, $s4, $t7`.

op code	rs	rt	rd	shamt	funct
0x0	$t7	$s4	$t1	0	0x4
0	15	20	9	0	4
000000	01111	10100	01001	00000	000100

6. $\bar{A}C + AB + A\bar{C}$ or $\bar{A}C + BC + A\bar{C}$

7. Easy.

8. A truth table shows that the two expressions are not equivalent.

   Prove: $\bar{B}\bar{C} \neq \bar{B} + \bar{C}$

   We can make a truth table showing all possible inputs for B and C:

B	C	$\bar{B}\bar{C}$	$\bar{B} + \bar{C}$
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

   Thus, since the third and fourth columns do not match in every case, the two expressions are not equivalent.