CSE 410 Assignment 1

Spring 2008

Due: Midnight, Monday 4/14/2008

Use electronic submission via the Catalyst tool:
https://catalysttools.washington.edu/collectit/dropbox/telmas/2218

Reading: Chapter 2 in P&H 3rd edition, or Chapter 3 in P&H 2nd edition; Appendix A, particularly the sections on MIPS assembly language (on the CD or downloadable from the spim web site).

Directions: In problems 0-2, resist the temptation to use a calculator that does the conversion between binary, decimal, and hex automatically.

(0) For the following base 10 values, convert the number to both binary and hexadecimal.
a. 10
b. 1011
c. 396
d. 4928
e. What two symbols have a shared meaning across binary, hex, and decimal? (hint: what values are common to the bases 2, 10, and 16?)

(1) For the following words, convert the binary number to both decimal and hexadecimal.
a. 1111 0000 1010 0101 0011 1100 0001 1000
b. 0000 0100 0010 1001 0110 0010 1110 0111

(2) Convert the following from hex to decimal and binary
a. 0x10110111
b. 0xBA5EBA11

(3) Show the MIPS instruction or instructions for the following C statements, assuming that 'a' corresponds to register $t0, 'b' corresponds to $t1, and the base address of the array 'd' is 4,000,000 (base 10), and that each variable is an integer (4 bytes) or array of ints.

(3a) a = a + b;

(3b) d[0] = d[1] + b;

(4) Show how the following MIPS instructions are encoded in memory (i.e., the binary word equivalent, one for each instruction):

```
addi  $s1, $t1, 5
add   $s0, $t0, $t2
lw    $t3, 32($t2)
```

(5) Add a comment per line to the following MIPS code and describe in one sentence what it computes. Assume that $a0 is used for the input and initially contains n, a positive integer. Assume that $v0 is used for the output. (note: slte is set on less than or equal to, and is a pseudoinstruction)

```
 begin:  addi  $t0, $zero, 0
         addi  $t1, $zero, 2
 loop:   slte  $t2, $a0, $t1
         bne   $t2, $zero, finish
         add   $t0, $t0, $t1
         addi  $t1, $t1, 2
         j loop
finish:  add   $v0, $t0, $zero
```

(6) Using the following MIPS assembly, fill out the corresponding register table and memory table below. The memory has been initialized for you, before the MIPS code has been executed. Your task is to trace through the code and each time you hit the "j start" label, record a snapshot of what is currently in memory (addresses 200-216) and in the registers ($s0,$s1,$s2,$t0,$t1) by filling in the tables. Note that you might have to add more columns to both tables depending on the number of iterations executed. (Note: the "addi" instruction is just like "add" except it allows you to add in a small constant directly rather than specifying a second register. For example, if $s0 contains 3, then after 'add $s2, $s0, 5' executes $s2 will contain 8.)
(6a) In one sentence, explain what this algorithm does?

```
start:   beq   $s1, $s2, exit
         add   $t0, $s2, $zero
         sll   $t0, $t0, 2
         add   $t0, $t0, $s0
         lw    $t1, 0($t0)
         srl   $t1, $t1, 1
         sw    $t1, 0($t0)
         addi  $s2, $s2, 1
         j     start
exit:
```

Here are the contents of a portion of memory before this snippet runs:

Address	**Initial contents**	**First time at "j start"**	**Second time at "j start"**	**...**
200	50			
204	128			
208	1024			
212	111			

216	2			

Here are the initial values of the registers before this snippet runs: (Note $t0 and $t1 are not in the list because they are used only as temporary registers, and their initial values do not matter.)

Register	**Initial contents**	**First time at "j start"**	**Second time at "j start"**	**...**
$s0	200			
$s1	5			
$s2	0			
$t0	N/A			
$t1	N/A			

(6b) Extra Credit: Both a conditional branch ("beq $s1, $s2, exit") and an unconditional jump ("j start") are executed each time through the loop. Only poor compilers would produce code with this loop overhead. Rewrite the assembly code so that it uses at most one branch or jump each time through the code. How many instructions are executed before and after the optimization? (Note: You may find it necessary to put one or more instructions before the "start" label.)

(7) The following program is analogous to an array copy (of unsigned ints) from one memory location to another. While copying, we want to keep track of the number of words copied (ie, the size of the array) and the sum of the individual elements in the array. The arrays are terminated with null (a completely unset word - all zeroes), and so this program should also copy the null over, but not include it in the count of items copied or the summation of the items copied.

The following code has many bugs, but tries to copy words from the address in register $a0 to the address in register $a1, counting the number of words copied in register $v0 and tallying the sum of those words copied in $v2. The program stops copying when it finds a word equal to zero, and you do not need to preserve the contents of $v0, $a0, and $a1.

```
       add $v2, $zero, $zero   #initialize to zero
       or $v0, $v0, $zero      #another way to initialize to zero
loop:  lw   $v1, 0($a0)        #read next word from source
       addi $v0, $v0, 1        #increment count words copied
       add  $v2, $v2, $v1      #add the value to our running total in v2
       sw   $v1, 0($a1)        #write to destination
       addi $a0,$a0,32         #advance ptr to next source
       addi $a1, $a1, 32       #advance ptr to next dest
       bne  $v1, $zero,loop    #loop if word copied not zero
```

(7a) There are multiple bugs in this program; identify the bugs and describe each in one sentence, then fix the bugs and turn in a bug-free version.