EE 352 Homework 1
Redekopp

Name: ___________________________________________  Score: ________

Note: Attach all work to receive full credit

Pseudo-Instructions
1.) (11 pts.) Expand/convert each of the following pseudo-instructions to an actual MIPS instruction sequence (you may use Section B.10 of the textbook for a list of instructions. Again you can’t use the ones marked “pseudo-instructions”. For certain pseudo-instructions you will need to use the $at ($1) register as a temporary value (in fact, we have started this for you).

Submission Instructions: For blanks, enter the register mnemonic like $t0, $at, etc. For immediate values write them as a 4-digit hex value (e.g. 0x0000). For multiple choice enter the appropriate letter of your select (e.g. a-d). For the beq/bne selection, type in your selection (e.g. beq).

<table>
<thead>
<tr>
<th>Pseudo-Instruction</th>
<th>Actual Instruction Seq. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.) li $t0,0x7fffabcd</td>
<td>lui $at,______ ori <strong><strong>,</strong></strong>,____</td>
</tr>
<tr>
<td>b.) clear $t0 (i.e. assign $t0=0)</td>
<td>Which of the following is not correct?</td>
</tr>
<tr>
<td></td>
<td>a) XOR $t0,$t0,$t0</td>
</tr>
<tr>
<td></td>
<td>b) SUB $t0,$t0,$t0</td>
</tr>
<tr>
<td></td>
<td>c) ADD $t0,$0,$0</td>
</tr>
<tr>
<td></td>
<td>d) None of the above (i.e. they all work)</td>
</tr>
<tr>
<td>c.) neg $t5,$t6 (i.e. $t5 = -$t6)</td>
<td>sub <strong><strong>,</strong></strong>,____</td>
</tr>
<tr>
<td>d.) bgt $t1,$t2,L1</td>
<td>slt $at,<strong><strong>,</strong></strong> beg/bne $at,$0,L1 (select one)</td>
</tr>
</tbody>
</table>
## Loads and Stores

2.) (18 pts.) Examine each of the following instructions. For the load instructions show the entire 32-bit content of $t1$ (in hex) after execution of the instruction.

**Submission Instructions:** Show the full 32-bit hex value of $t1$ after each load instructions. Precede the value with the hex modified ‘0x’ (e.g. 0x12345678).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) lw $t1,8($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>b) lbu $t1,0x13($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>c) lh $t1,-4($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>d) lw $t1,0xfffc($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>e) lb $t1,9($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>f) lhu $t1,-6($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>g) lb $t1,0xf($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>h) lhu $t1,-8($t0)</td>
<td>$t1 =</td>
</tr>
<tr>
<td>i) lb $t1,18($t0)</td>
<td>$t1 =</td>
</tr>
</tbody>
</table>
3.) (7 pts.) Assume the contents of all memory locations shown below are initially 0.
Perform each store instruction and then show the contents of all memory locations after all stores have been executed.

**Submission Instructions**: Show the full 32-bit hex value of each word of memory
Precede the value with the hex modified ‘0x’ (e.g. 0x12345678). Remember memory locations not changed by the store instructions should be filled with 0’s.

<table>
<thead>
<tr>
<th>$t0$</th>
<th>$t1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000071A8</td>
<td>AB125680</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) sw $t1,8($t0)</td>
<td>(Show in memory above)</td>
</tr>
<tr>
<td>b) sb $t1,0x0d($t0)</td>
<td></td>
</tr>
<tr>
<td>c) sh $t1,-2($t0)</td>
<td></td>
</tr>
<tr>
<td>d) sb $t1,0xffffa($t0)</td>
<td></td>
</tr>
<tr>
<td>e) sh $t1,4($t0)</td>
<td></td>
</tr>
</tbody>
</table>
MIPS Assembly

4.) (26 pts.) The following assembly instruction sequence implements an expression of a C-style high level language.

a. Fill in the indicated result of each instruction. Assume the following initial values are true at the beginning of execution of the first instruction. Further assume all registers contain 0’s initially.

Submission Instructions: Show the full 32-bit hex value of each register after execution of the instruction. Begin the value with the hex modified ‘0x’ (e.g. 0x12345678).

<table>
<thead>
<tr>
<th>C Variable Name</th>
<th>Memory Contents</th>
<th>Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>7E12 048A</td>
<td>0x1000E010</td>
</tr>
<tr>
<td>D</td>
<td>0000 0008</td>
<td>0x1000E00c</td>
</tr>
<tr>
<td>C</td>
<td>4FFD 3447</td>
<td>0x1000E008</td>
</tr>
<tr>
<td>B</td>
<td>0000 0005</td>
<td>0x1000E004</td>
</tr>
<tr>
<td>A</td>
<td>68CE 8932</td>
<td>0x1000E000</td>
</tr>
</tbody>
</table>

i) lui $s0,0x1000 $s0 = __________________
ii) ori $s0,$s0,0xe000 $s0 = __________________
iii) lw $s1,0($s0) $s1 = __________________
iv) lw $s2,4($s0) $s2 = __________________
v) lw $s3,8($s0) $s3 = __________________
vi) addi $s0,$s0,16 $s0 = __________________
vii) lw $s4,-4($s0) $s4 = __________________
viii) mul $t0,$s2,$s4 $t0 = __________________
ix) addi $t0,$t0,8 $t0 = __________________
x) sub $s1,$s2,$s4 $s1 = __________________
xi) sll $s1,$s1,1 $s1 = __________________
xii) add $s5,$t0,$s1 $s5 = __________________
xiii) sw $s5,0($s0)

b. Assume that the memory addresses correspond to C variables (shown above) of type int. Now use your understanding of the above instruction sequence to translate the MIPS instructions into a corresponding C style assignment statement. Select the appropriate answer below.

Submission Instructions: Enter the appropriate letter representing your selection.

a) Z = B*(Z+8) – (A-C)/2;
b) Z = B*(Z+8) – 2*(A-C);
c) Z = B*D + 8 + (A-C)/2;
d) Z = B*D + 8 + 2*(A-C);
Assembler Directives
5.) (8 pts.) Examine the following C program variable declarations and translate them to the appropriate directives.

**Submission Instructions:** Enter the number of your desired selection.

a. short int x = 6;
   1. x: .word 6
   2. x: .half 6
   3. x: .space 6
   4. x: .align 6

b. unsigned char msg[8] = {1,4,9,7,3,6,8,2};
   1. msg: .unsigned 1,4,9,7,3,6,8,2
   2. msg: .byte 0x14973682
   3. msg: .byte 1,4,9,7,3,6,8,2
   4. msg: .half 1,4,9,7,3,6,8,2

c. int data[100];
   1. data: .word 100
   2. data: .space 100
   3. data: .space 200
   4. data: .space 0x190

d. char str[] = “hello\n”
   1. str: .ascii “hello\n”
   2. str: .ascii “hello\n”
   3. str: .byte “hello\n”
   4. str: .ascii “hello”
## HLL to Assembly Translation

6.) (30 pts.) Translate the following C code statements to an equivalent assembly language implementation.

**Submission Instructions:** For constants/numbers enter your values in **decimal**. For registers, use the descriptive mnemonic (i.e. $t0, $s1). For labels, just enter it as shown. For opcodes, enter the full opcode (e.g. b__ => enter ‘bgt’).

```
short data[20];
...
for(int i=0; i < 20; i++){
    data[i] = data[i] + 5;
}
// let i be stored in $t0
```

```
data .space   ________  .text...
  la $s0,data
  add $t0,____,____
  li $t4,______
L0:  slt $t1,___,$t4
L1:  b__ $t1,$zero,____
L2:  sll $t2,$t0,____
L3:  add $s1,___,$t2
L4:  lh $t3,________
L5:  addi $t3,___,5
L6:  sh $t3,_______
L7:  addi $t0,_____,____
L8:  b _________
L9:  ...
```

```
int *x, *y;
...
if(*x >= 10 && *y < 5)  
code A
else if (*x < 10 || *y > 5)  
code B
else  
code C
...
```

```
x .space 4
y .space 4
...
  la $s0,x
  la $s1,y
  addi $t0,____,10
  addi $t1,____,5
  lw $s2,0($s0)
  lw $s3,0($s1)
  lw $t2,_____
  lw $t3,_____
  blt __,$t0,____
  b__ $t3,___
L1:  Code A instructions
L2:  b _______
L3:  b__ $t2,$t0,____
L4:  b__ $t3,$t1,____
L5:  Code B instructions
L6:  b _______
L7:  Code C instructions
L8:  ...
```