16.482 / 16.561: Computer Architecture and Design

Summer 2015

Homework #1 Solution

For each instruction sequence below, assume the following initial state. Note that your answers to each part should use the values below—your answer to part (2), for example, should not affect your answer to part (1). However, please note that each part is a sequence of instructions—the result of the `sub` *in part (1) will affect the* `add` *in part (1).*

- *`$s0 = 0x16482000, $t0 = 0x0000000C, $t1 = 0x00000003`*
- *Contents of memory (all values are in hexadecimal)*

Address	*Lo*			*Hi*
0x16482000	*AA*	*BB*	*11*	*22*
0x16482004	*33*	*44*	*09*	*FF*

Please note that in the (corrected) figure above, "Lo" refers to the lowest address offset within the line (i.e., 0), while "Hi" refers to the highest offset (i.e., 3). In other words, the byte at address 0x16482000 is 0xAA, while the byte at address 0x16482003 is 0x22.

*For each sequence of instructions below, list **all** changed registers or memory locations and their new values. When listing memory values, list the entire word—for example, if a byte is written to 0x00100000, show the values at addresses 0x00100000-0x00100003.*

1. *(8 points)*

   `sub  $t3, $t0, $t1`

   $t3 = $t0 - $t1 = 0x0000000C – 0x00000003 = 0x00000009

   `addi  $t4, $t0, 8`

   $t4 = $t0 + 8 = 0x0000000C + 8 = 0x00000014

   `add  $t5, $t3, $t4`

   $t5 = $t3 + $t4 = 0x00000009 + 0x00000014 = 0x0000001D

2. *(12 points)*

   `addi  $s1, $zero, 0xFFFF`

   $s1 = $zero + 0xFFFFFFFF (sign-extended immediate)
   = 0 + 0xFFFFFFFF = 0xFFFFFFFF

   `xor  $s2, $t0, $s1`

   $s2 = $t0 XOR $s1 = 0x0000000C XOR 0xFFFFFFFF
   = 0xFFFFFFF3

   `srl  $s3, $s2, 4`

   $s3 = $s2 >> 4 (logical right shift)
   = 0xFFFFFFF3 >> 4 = 0x0FFFFFFF

   `and  $s4, $s3, $s2`

   $s4 = $s3 AND $s2 = 0x0FFFFFFF AND 0xFFFFFFF3
   = 0x0FFFFFF3

3. *(18 points)*

lh $t2, 0($s0)

$t2 = sign-extended halfword at mem[0x16482000]
= 0xFFFFAABB

lhu $t3, 6($s0)

$t3 = zero-extended halfword at mem[0x16482006]
= 0x000009FF

sra $t4, $t2, 8

$t4 = $t2 >> 8 (arithmetic right shift—keep sign)
= 0xFFFFAABB >> 8 = 0xFFFFFFAA

sb $t3, 3($s0)

mem[0x16482003] = lowest byte of $t3 = 0xFF
➔ mem[0x16482000] = 0xAABB11FF (changed byte underlined)

sw $t4, 4($s0)

mem[0x16482004] = $t4 = 0xFFFFFFAA

4. *(12 points)*

slti $s0, $t1, 11

$s0 = 1 if ($t1 < 11)
➔ Since $t1 = 0x00000003, $t1 < 11 ➔ $s0 = 0x00000001

bne $s0, $zero, L

Branch to L if $s0 is not equal to $zero
$s0 = 1, $zero = 0 ➔ Branch is taken

or $t0, $t0, $t1

Instruction is skipped

L: sh $t0, 2($s0)

mem[0x00000003] = lowest halfword of $t0 = 0x000C
➔ mem[0x00000003] = 0x00, mem[0x00000004] = 0x0C

The original solution (shown below) is actually incorrect because the first instruction changes $s0, which is used in the address calculation.

mem[0x16482002] = lowest halfword of $t0 = 0x000C
➔ mem[0x16482000] = 0xAABB000C (changed bytes underlined)