CS2100 Computer Organization
Tutorial 3

D1. Given two integer arrays *A* and *B* with unknown number of elements, and their base addresses stored in registers **$s0** and **$s1** respectively, study the MIPS code below. Note that an integer takes up 32 bits of memory.

```
      addi $t0, $s0, 0
      addi $t1, $s1, 0
loop: lw   $t3, 0($t0)
      lw   $t4, 0($t1)
      slt  $t5, $t4, $t3      # line A
      beq  $t5, $zero, skip   # line B
      sw   $t4, 0($t0)
      sw   $t3, 0($t1)
skip: addi $t0, $t0, 4
      addi $t1, $t1, 4
      bne  $t3, $zero, loop
```

a. What is the purpose of register **$t1** in this code?

b. If array *A* = {7, 4, 1, 6, 0, 5, 9, 0} and
array *B* = {3, 4, 5, 2, 1, 0, 0, 9},
give the final content of these two arrays.

c. How many **store word** operations are performed given the contents of the arrays in part (b)?

d. What is the value (in decimal) of the immediate field in the machine code representation of the **bne** instruction?

e. The two lines indicated as “line A” and “line B” represent the translation of a MIPS pseudo-instruction. Give the corresponding pseudo-instruction.

Tutorial Questions:

1. Below is a C code that performs palindrome checking. A palindrome is a sequence of characters that reads the same backward or forward. For example, "madam" and "rotator" are palindromes.

```
char string[size] = { ... }; // some string
int low, high, matched;

// Translate to MIPS from this point onwards
low = 0;
high = size-1;
matched = 1;          // assume this is a palindrome
                      // In C, 1 means true and 0 means false
while ((low < high) && matched) {
    if (string[low] != string[high])
        matched = 0;  // found a mismatch
    else {
        low++;
        high--;
    }
}
// "matched" = 1 (palindrome) or 0 (not palindrome)
```

Given the following variable mappings:

low ➔ $s0;
high➔ $s1;
matched ➔ $s3;
base address of string[] ➔ $s4;
size ➔ $s5

a. Translate the C code into MIPS code by keeping track of the indices.

b. Translate the C code into MIPS code by using the idea of "array pointer". Basically, we keep track of the actual addresses of the elements to be accessed, rather than the indices. Refer to lecture set #8, slide 34 for an example.

Note: Recall the "short circuit" logical AND operation in C. Given condition (A && B), condition B will not be checked if A is found to be false.

2. a. You accidentally spilled coffee on your best friend's MIPS assembly code printout. Fortunately, there are enough hints for you to reconstruct the code. Fill in the missing lines (shaded cells) below to save your friendship.

Answer:

Instruction Encoding	MIPS Code
	# **$s1** stores the result, **$t0** stores a non-negative number
	addi $s1, $zero, 0 #Inst. address is 0x00400028
0x00084042	**loop: srl $t0, $t0, 1**
0x11000002	
0x22310001	
	j loop
	exit:

b. Give a simple mathematic expression for the relationship between **$s1** and **$t0** as calculated in the code.

3. [AY2012/13 Semester 2 Assignment 3]
   Your friend Alko just learned **binary search** in CS1020 and could not wait to impress you. As a friendly gesture, show Alko that you can do the same, but in MIPS! ☺

   Complete the following MIPS code. To simplify your tasks, some instructions have already been written for you, so you only need to fill in the missing parts in **[ ]**. Please translate as close as possible to the original code given in the comment column. You can assume registers $s0 to $s5 are properly initialized to the correct values before the code below.

   a.

Variable Mappings	Comments
address of array[] ➔ $s0 target ➔ $s1 // value to look for in array low ➔ $s2 // lower bound of the subarray high ➔ $s3 // upper bound of the subarray mid ➔ $s4 // middle index of the subarray ans ➔ $s5 // index of the target if found, -1 otherwise. Initialized to -1.	

loop:
  slt $t9, $s3, $s2
  bne $t9, $zero, end
	#while (low <= high) {

add $s4, $s2, $s3 [ ]	# mid = (low + high)/ 2

sll $t0, $s4, 2
  add $t0, $s0, $t0
  [ ]
	# t0 = mid*4
	# t0 = &array[mid] in bytes
	# t1 = array[mid]

slt $t9, $s1, $t1 beq $t9, $zero, bigger	# if (target < array[mid])
addi $s3, $s4, -1 j loopEnd	# high = mid – 1

bigger:
  [ ]
  [ ]
	# else if (target > array[mid])

addi $s2, $s4, 1 j loopEnd	# low = mid + 1

equal:
  add $s5, $s4, $zero
  [ ]
	# else {
	  ans = mid
	  break
	# }

loopEnd: [ ]	#} //end of while-loop
end:	

b. What is the immediate value in decimal for the "**bne $t9, $zero, end**" instruction? You should count only the instructions; labels are not included in the machine code.

c. If the first instruction is placed in memory address at 0xFFFFFF00, what is the **hexadecimal representation** of the instruction "**j loopEnd**" (for "high = mid – 1")?

d. Is the encoding for the second "**j loopEnd**" different from part (c)? If yes, give the new encoding, otherwise briefly explain the reason.