CS2100 Computer Organization
Tutorial 3
ANSWERS

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.

To tutors: It may help if you can project the programs in Q1-3 on the whiteboard. For Q2, you may also get students to write their answers overlay with the projected program on the whiteboard.
For room with a lot of whiteboards, you may consider getting students to write their answers for different on the whiteboards concurrently to save time.

Answers:

a.

```
      addi $s0, $zero, 0     # low = 0
      addi $s1, $s5, -1      # high = size-1
      addi $s3, $zero, 1     # matched = 1
loop: slt  $t0, $s0, $s1     # (low < high)?
      beq  $t0, $zero, exit  # exit if (low >= high)
      beq  $s3, $zero, exit  # exit if (matched == 0)
      add  $t1, $s4, $s0     # address of string[low]
      lb   $t2, 0($t1)       # t2 = string[low]
      add  $t3, $s4, $s1     # address of string[high]
      lb   $t4, 0($t3)       # t4 = string[high]
      beq  $t2, $t4, else
      addi $s3, $zero, 0     # matched = 0
      j endW                 # can be "j loop"
else: addi $s0, $s0, 1       # low++
      addi $s1, $s1, -1      # high–
endW: j loop                 # end of while
exit:                        # outside of while
```

b.

```
      addi $s0, $zero, 0     # low = 0
      addi $s1, $s5, -1      # high = size-1
      addi $s3, $zero, 1     # matched = 1
      add  $t1, $s4, $s0     # address of string[low]
      add  $t3, $s4, $s1     # address of string[high]
loop: slt  $t0, $t1, $t3     # compare low and high addr
      beq  $t0 $zero, exit
      beq  $s3, $zero, exit  # exit if (matched == 0)
      lb   $t2, 0($t1)       # t2 = string[low]
      lb   $t4, 0($t3)       # t4 = string[high]
      beq  $t2, $t4, else
      addi $s3, $zero, 0     # matched = 0
      j endW                 # can be "j loop"
else: addi $t1, $t1, 1       # low address increases
      addi $t3, $t3, -1      # high address decreases
endW: j loop                 # end of while
exit:                        # outside of while
```

2. Answer omitted.

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`
`    [`**`srl $s4, $s4, 1`**` ]`
	`#    mid = (low + high)/ 2`

`    sll $t0, $s4, 2`
`    add $t0, $s0, $t0`
`    [`**`lw  $t1, 0($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:`
`    [`**`slt $t9, $t1, $s1`**`]`
`    [`**`beq $t9, $zero, equal`**`]`
	`#    else if (target > array[mid])`

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

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

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

`end:`	

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

Answer: Immediate value = **16_{10}**

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")?

Answer: Binary encoding for "j loopEnd": **0x0B FF FF D1**

Workings:

"loopEnd" is the 18th instruction.

So, offset from start = 17 instructions × 4 = 68_{10} = 44_{16}

Address of loopEnd = 0xFFFFFFFF**44**

j loopEnd = 000010 **1111…….. 1101 0001** = **0x0B FF FF D1**

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

Answer: No. Jump specifies the target “directly”. So, two jumps to the same target will give the same encoding.