IT5002 Computer Organization
Tutorial 2
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. MIPS Bitwise Operations

Implement the following in MIPS assembly. Assume that integer variables **a**, **b** and **c** are mapped to registers $s0, $s1 and $s2 respectively. Each part is independent of all the other parts. **For bitwise instructions (e.g. ori, andi, etc),** any immediate values you use should be written in binary for this question. This is optional for non-bitwise instructions (e.g. addi, etc).

Note that bit 31 is the most significant bit (MSB) on the left, and bit 0 is the least significant bit (LSB) on the right, i.e.:

MSB					LSB
Bit 31	Bit 30	Bit 29	…	Bit 1	Bit 0

a. Set bits 2, 8, 9, 14 and 16 of **b** to 1. Leave all other bits unchanged.
To set bits, we create a "mask" with 1's in the bit positions we want to set. Since bit 16 is in the upper 16 bits of the register, we need to use lui to set it.

```
lui $t0, 1      # Sets bit 16 of $t0.
ori $t0, $t0, 0b0100001100000100 # Set bits 14, 9, 8
and 2. or $s1, $s1, $t0
```

b. Copy over bits 1, 3 and 7 of **b** into **a**, without changing any other bits of **a**. # We use the property that x AND 1 = x to copy out the values of
bits 7, 3 and 1 of b into $t0. Note that we zero all other bits
so that they don't change anything in $s0 when we OR
later on. andi $t0, $s1, 0b0000000010001010

```
# We use the property of x OR 0 = x to copy in
# the bits into a, so we prepare a by zero-ing bits 7, 3 and 1.
# To do this we need the mask 1111111111111111
1111111101110101 lui      $t1, 0b1111111111111111
ori   $t1, $t1, 0b1111111101110101
and  $s0, $s0, $t1

# Now OR together a and $t0 to copy over
the bits or $s0, $s0, $t0
```

c. Make bits 2, 4 and 8 of **c** the inverse of bits 1, 3 and 7 of **b** (i.e. if bit 1 of **b** is 0, then bit 2 of **c** should be 1; if bit 1 of **b** is 1, then bit 2 of **c** should be 0), without changing any other bits of **c**. # We use the property that x XOR 1 = ~x to flip the values of bits 7, 3 and 1.
xori $t0, $s1, 0b10001010

```
# Zero every bit except 7, 3
and 1. andi $t0, $t0,
0b10001010

# Shift left one position: bit 1 becomes 0, bit 1 becomes bit 2, bit 3 becomes bit 4, and
bit 7
become
s bit 8.
sll $t0,
```

```
$t0, 1

# Now, to clear bits 8, 4 and 2 of c,
# we need the mask 0b1111111111111111
1111111011101010 lui      $t1, 0b1111111111111111
ori   $t1, $t1, 0b1111111011101011
and $s2, $s2, $t1

# and we OR the new c
with $t0 or $s2, $s2, $t0
```

3. MIPS Arithmetic

Write the following in MIPS Assembly, using as few instructions as possible. You may rewrite the equations if necessary to minimize instructions.

In all parts you can assume that integer variables **a**, **b**, **c** and **d** are mapped to registers $s0, $s1, $s2 and $s3 respectively. Each part is independent of the others.

a. $c = a + b$

```
add $s2, $s0, $s1
```

b. $d = a + b - c$

```
add $s3, $s0, $s1     # d = a + b
sub $s3, $s3, $s2     # d = (a + b) - c
```

c. $c = 2b + (a - 2)$

```
add $s2, $s1, $s1     # c = 2b (alternatively, can do a shift
left 1 bit) addi $t0, $s0, -2  # $t0 = a - 2
add $s2, $s2, $t0     # c = 2b + (a – 2)
```

d. $d = 6a + 3(b - 2c)$

Note to TAs: Students may find better solutions than this. Check to ensure that they achieve the equation above.

```
Rewrite:
d = 6a +
3b – 6c
Factoriz
e out 3

d = 3(2a + b – 2c)
= 3(2a – 2c + b)
= 3(2(a – c) + b)
```

```
sub $t0, $s0, $s2      # t0 = a – c
sll $t0, $t0, 1        # t0 =
2(a – c) add $t0, $t0, $s1   #
t0 = 2(a – c) + b
sll $t1, $t0, 2        # t1 = 4(2(a
– c) + b) sub $s3, $t1, $t0   # d =
3(2(a – c) + b)
```

4. [AY2013/14 Semester 2 Exam]
The mysterious MIPS code below assumes that **$s0 is a 31-bit binary sequence**, i.e. the MSB (most significant bit) of **$s0** is assumed to be zero at the start of the code.

```
      add   $t0, $s0, $zero    # make a copy of $s0 in $t0
      lui   $t1, 0x8000
lp:   beq   $t0, $zero, e
      andi  $t2, $t0, 1
      beq   $t2, $zero, s
      xor   $s0, $s0, $t1
s:    srl   $t0, $t0, 1
      j     lp
e:
```

a) For each of the following initial values in register **$s0** at the beginning of the code, give the hexadecimal value of the content in register **$s0** at the end of the code.

i. Decimal value **31**.
ii. Hexadecimal value **0x0AAAAAAA**.

b) Explain the purpose of the code in one sentence.

Answers:

a. i. $s0 = 0x8000 001F

ii. **$s0 = 0x0AAA AAAA**

b. The code sets bit 31 of $s0 to 1 if there are odd number of '1' in $s0 initially, or 0 if there are even number of '1'. (This is called the even parity bit.)