COE 301 / ICS 233
Computer Organization

Exam 1 – Spring 2017

Saturday, March 18, 2017
10 AM – 12 Noon

Computer Engineering Department
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals

Student Name: **SOLUTION**________________________

Student ID: ________________________________

Section: ________________________________

Q1	/ 20	Q2	/ 15
Q3	/ 15	Q4	/ 15
Q5	/ 20	Q6	/ 20
Total	/ 105		

Important Reminder on Academic Honesty

Using unauthorized information or notes on an exam, peeking at others work, or altering graded exams to claim more credit are severe violations of academic honesty. Detected cases will receive a failing grade in the course.

Prepared by Dr. Muhamed Mudawar and Dr. Marwan Abu-Amara

Question 1: Fill-in the Blanks

a) (2 pts) Imagine that you are working for a company that fabricates a certain IC chip. The cost per wafer is $3000, and each wafer has 2000 dies. If the cost of a good die is $2.50, then the yield of this manufacturing process is **($3000/$2.50)/2000 = 0.6 or 60%**.

b) (2 pts) Given that the instruction `j NEXT` is at address `0x004000F4`, and the label `NEXT` is at address `0x00402AEC`. Then, the 26-bit immediate stored in the jump instruction for the label `NEXT` is `0x00402AEC >> 2 = 0x0100ABB`.

c) (3 pts) Given the following data definitions, the address of the first variable `X` is given at `0x10010000` (hexadecimal), the hexadecimal addresses for **Y**, **Z**, and **S** will be:

```
.data
X: .half   1, 2, 3
Y: .byte   'A', 'B', 'C'
Z: .word   7, 8, 9
.ALIGN 3
S: .asciiz  "STRING"
```

Address of Y = `0x10010006`

Address of Z = `0x1001000C`

Address of S = `0x10010018`

d) (3 pts) Show the MIPS assembly language instruction that is equivalent to the following machine language instruction. Provide the immediate value in **decimal**. The MIPS Reference data sheet is attached at the end.

Machine language instruction	MIPS assembly language instruction
`0011 0001 0001 0001 1000 0111 0110 0101`	`Op=ANDI, Rs=$8, Rt=$17, I=34661` `ANDI $s1, $t0, 34661`

e) (5 pts) Each square in the table shown below represents one byte in memory and each row stores 8 bytes in memory. Starting at address **`0x10010000`** in the data segment, show the **byte content** in memory in **hexadecimal** for the following data definitions. If a byte is not used (or uninitialized) then leave it empty. Fill only the bytes that are initialized. For words and half words, the little endian byte ordering should be used.

```
.DATA
.WORD       -2
.HALF       0x1FFF
.ALIGN      2
.BYTE       11:3
.ALIGN      4
.BYTE       13, -1
```

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x10010000	0xFE	0xFF	0xFF	0xFF	0xFF	0x1F		
0x10010008	0x0B	0x0B	0x0B					
0x10010010	0x0D	0xFF						
0x10010018								

f) (5 pts) Given the following contents of memory, where each square represents only one byte in memory, show the values of registers **`$t0`** thru **`$t4`** in **hexadecimal** after executing each of the following MIPS assembly language instructions. The little endian byte ordering should be used. Assume **`$s0 = 0x10010020`**.

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x10010020	0xFA	0x20	0x10	0xC0	0xB0	0x5F	0x94	

`lw  $t0, 0($s0)`	$t0 = 0xC01020FA
`lh  $t1, 2($s0)`	$t1 = 0xFFFFC010
`lhu $t2, 4($s0)`	$t2 = 0x00005FB0
`lb  $t3, 5($s0)`	$t3 = 0x0000005F
`lbu $t4, 6($s0)`	$t4 = 0x00000094

Question 2: Pseudo-Instructions

For each of the following pseudo-instructions, produce a **minimal** sequence of basic MIPS instructions to accomplish the same thing. You may use the **`$at`** register only as a temporary register.

a) `abs $t1, $t2` `# absolute value` (4 pts)

```
    addu  $t1, $t2, $zero   # $t1 = $t2
    bgez  $t2, done
    subu  $t1, $zero, $t2   # $t1 = -$t2
done:

    # Solution 2: No branch
    sra   $at, $t2, 31       # $at = 0 or -1 (0xFFFFFFFF)
    xor   $t1, $t2, $at      # $t1 = $t2 or 1's complement
    subu  $t1, $t1, $at      # $t1 = $t2 or 2's complement
```

b) `addiu $t1, $t2, 0x1234abcd` `# 32-bit constant` (4 pts)

```
    lui   $at, $0x1234          # $at = 0x12340000
    ori   $at, $at, $0xabcd     # $at = 0x1234abcd
    addu  $t1, $t2, $at
```

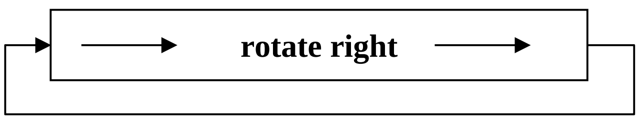
c) `bgt $t1, 100, Label` `# branch if greater than 100` (3 pts)

```
    ori   $at, $zero, 100
    slt   $at, $at, $t1
    bne   $at, $zero, Label

    # Better solution
    slti  $at, $t1, 101
    beq   $at, $zero, Label
```

d) `ror $t1, $t2, 15` `# rotate right value of $t2 15 bits` (4 pts)

```
    sll   $t1, $t2, 17
    srl   $at, $t2, 15
    or    $t1, $t1, $at
```



Question 3: Trace the Execution of the following Code

a) (7 pts) Given that **`Array`** is defined as shown below, determine the content of registers **`$v0`** and **`$v1`** after executing the following code. **<u>Explain what the program does</u>**.

```
Array: .word 15, -19, 17, 20, -10, 12, 100, -5

        la   $a0, Array # $a0 = 0x10010000
        addi $a1, $a0, 28
        move $v0, $a0
        lw   $v1, 0($v0)
        move $t0, $a0
loop:   addi $t0, $t0, 4
        lw   $t1, 0($t0)
        bge  $t1, $v1, skip
        move $v0, $t0
        move $v1, $t1
skip:   bne  $t0, $a1, loop
```

```
$v0 = 0x10010004 (address of minimum element)
$v1 = -19 (minimum value)
```

The program is determining the minimum element in the array and its address in memory.

b) (8 pts) Given that **`Array`** is defined as shown below, determine the content of **`Array`** after executing the following code. **<u>Explain what the program does.</u>**

```
Array: .half 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

       la   $a0, Array
       li   $a1, 6
       move $t0, $a0
       addi $t1, $a0, 12

loop:  lh   $t3, ($t0)
       lh   $t4, ($t1)
       sh   $t3, ($t1)
       sh   $t4, ($t0)
       addi $t0, $t0, 2
       addi $t1, $t1, 2
       addi $a1, $a1, -1
       bne  $a1, $zero, loop
```

```
New Array Content:

7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6
```

The program is swapping the first six array elements with the last six

Question 4: Writing MIPS code

(15 pts) Write a MIPS loop that converts a string to lower case. The address of the string exists in register **`$a0`**. The string is terminated with a null character. The string should be read, converted, and stored in memory. Check each character if it is an upper case letter (range **'A'** to **'Z'**) before converting it to lower case. Recall that **'A' = 0x41** and **'a' = 0x61**.

```
        move $t0, $a0           # $t0 = string pointer

loop:   lb   $t1, 0($t0)        # load $t1 = character
        blt  $t1, 'A', skip     # not a capital letter
        bgt  $t1, 'Z', skip     # not a capital letter
        addi $t1, $t1, 0x20     # convert to lowercase
        sb   $t1, 0($t0)        # store lowercase letter
skip:   addi $t0, $t0, 1        # advance pointer
        bne  $t1, $zero, loop   # loop if not null character
```

Question 5: Translating Nested Loops into MIPS Assembly Language

(20 pts) Translate the following nested loops into MIPS assembly language. Register **`$a0`** stores the number **`n`** of elements in all arrays, **`$a1`** = address of the array **`a[]`**, **`$a2`** = address of the array **`b[]`**, and **`$a3`** = address of the array **`c[]`**. Each array element is a 32-bit signed integer. Insert comments to clarify the meaning of instructions and the use of registers.

```
for (i=0; i != n; i++) {
   int cnt = 0;
   for (j=0; j != n; j++) {
      if (a[i] == b[j]) cnt = cnt + 1;
   }
   c[i] = cnt;
}
```

Solution:

```
      li      $t0, 0            # $t0 = i = 0
for1:                           # outer for loop
      li      $t1, 0            # cnt = 0
      li      $t2, 0            # $t2 = j = 0
      lw      $t3, 0($a1)       # load $t3 = a[i]
      move    $t4, $a2          # $t4 = address of b

for2:                           # inner for loop
      lw      $t5, 0($t4)       # load $t5 = b[j]
      bne     $t3, $t5, skip    # skip if a[i] != b[j]
      addiu   $t1, $t1, 1       # cnt = cnt + 1
skip:
      addiu   $t4, $t4, 4       # point to next b[j]
      addiu   $t2, $t2, 1       # j++
      bne     $t2, $a0, for2    # loop back if j!=n

      sw      $t1, 0($a3)       # store c[i] = cnt
      addiu   $a1, $a1, 4       # point to next a[i]
      addiu   $a3, $a3, 4       # point to next c[i]
      addiu   $t0, $t0, 1       # i++
      bne     $t0, $a0, for1    # loop back if i!=n
```

Question 6: The Transposition of a Matrix

(20 pts) Transposition is an important matrix operation. Given that matrix **A** is a square matrix of integers with dimensions **n×n**, the transposition is accomplished by swapping matrix element **A[i][j]** with element **A[j][i]**, as shown in the following nested loops. Given that register **$a0=n**, and register **$a1** = **address** of matrix **A**, translate the following nested loops into MIPS assembly language code.

```
for (i=0; i != n; i++) {
   for (j=i+1; j != n; j++) {
      temp1 = A[i][j];
      temp2 = A[j][i];
      A[i][j] = temp2;
      A[j][i] = temp1;
   }
}
```

Outer for loop has only (n – 1) iterates, because when i is (n – 1) the inner for loop will have zero iterates.

Solution:

```
        li      $t0, 0              # $t0 = i = 0
        addiu   $t9, $a0, -1        # $t9 = n-1 (iterates outer for)
  for1:                             # outer for loop
        addiu   $t1, $t0, 1         # $t1 = j = i+1
  for2:                             # inner for loop
        mul     $t2, $t0, $a0       # $t2 = i*n
        addu    $t2, $t2, $t1       # $t2 = i*n + j
        sll     $t2, $t2, 2         # $t2 = (i*n + j) * 4
        addu    $t2, $a1, $t2       # $t2 = &A[i][j]

        mul     $t3, $t1, $a0       # $t3 = j*n
        addu    $t3, $t3, $t0       # $t3 = j*n + i
        sll     $t3, $t3, 2         # $t3 = (j*n + i) * 4
        addu    $t3, $a1, $t3       # $t3 = &A[j][i]

        lw      $t4, 0($t2)         # $t4 = A[i][j]
        lw      $t5, 0($t3)         # $t5 = A[j][i]
        sw      $t5, 0($t2)         # A[i][j] = $t5
        sw      $t4, 0($t3)         # A[j][i] = $t4

        addiu   $t1, $t1, 1         # j++
        bne     $t1, $a0, for2      # loop back if j!=n

        addiu   $t0, $t0, 1         # i++
        bne     $t0, $t9, for1      # loop back if i!=(n-1)
```

Better Solution: Faster Traversal of Matrix by Rows and by Columns

```
        li      $t0, 0              # $t0 = i = 0
        sll     $t7, $a0, 2         # $t7 = n*4 (bytes per row)
        addiu   $t8, $t7, 4         # $t8 = n*4 + 4 (bytes)
        addiu   $t9, $a0, -1        # $t9 = n-1 (iterates outer for)

for1:                               # outer for loop
        addiu   $t1, $t0, 1         # $t1 = j = i+1
        addiu   $t2, $a1, 4         # $t2 = &A[i][j]
        addu    $t3, $a1, $t7       # $t3 = &A[j][i]

for2:                               # inner for loop
        lw      $t4, 0($t2)         # $t4 = A[i][j]
        lw      $t5, 0($t3)         # $t5 = A[j][i]
        sw      $t5, 0($t2)         # A[i][j] = $t5
        sw      $t4, 0($t3)         # A[j][i] = $t4

        addiu   $t2, $t2, 4         # $t2 = &A[i][j] (by row)
        addu    $t3, $t3, $t7       # $t3 = &A[j][i] (by column)

        addiu   $t1, $t1, 1         # j++
        bne     $t1, $a0, for2      # loop back if j!=n

        addu    $a1, $a1, $t8       # $a1 = &A[i][i] (main diagonal)
        addiu   $t0, $t0, 1         # i++
        bne     $t0, $t9, for1      # loop back if i!=(n-1)
```

Smaller inner loop: 8 instructions per inner loop iterate versus 14 used in first solution. No multiply instruction is used for address calculation in the second solution.

Any solution that works is acceptable.