March 27, 2008

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Major Exam I

Second Semester (072)

Time: 1:00-3:30 PM

Student Name : _KEY____

Student ID. : ____

Question	Max Points	Score
Q1	30	
Q2	15	
Q3	15	
Q4	20	
Q5	20	
Total	100	

Dr. Aiman El-Maleh

[30 Points]

(Q1) Fill in the blank in each of the following questions:

(1) The smallest (negative) number that can be represented using 32-bit 2`s complement in hexadecimal is 80000000 and the largest positive number in hexadecimal is 7FFFFFFF.

(2) Assuming 8-bit representation of numbers, the binary number 10101110 is equal to -46 in sign-magnitude representation, -81 in 1`s complement representation, and -82 in 2`s complement representation.

(3) Two advantages of programming in assembly language are accessibility to hardware resources and space and time efficiency.

(4) The advantage of dynamic RAM over static RAM is that it is denser and cheaper but the disadvantage is that it is slower as it requires refreshing.

(5) Cache memory is used to bridge the widening speed gap between CPU and main memory.

(6) Memory hierarchy consists of the following from highest speed to lowest speed: Registers, L1 Cache, L2 Cache, Main Memory (RAM), and Hard Disk.

(7) The following assembler directive allocates 20 words initialized by 5.

```
X: .word 5:20
```

(8) With a 36-bit address bus and 64-bit data bus, the maximum memory size than can be accessed by a processor is $\underline{2^{36}=64G}$ Byte and the maximum number of bytes that can be read or written in a single cycle is $\underline{64/8=8}$ Bytes.

(9) Given a magnetic disk with the following properties:

- Rotation speed = 7200 RPM (rotations per minute)
- Average seek = 8 ms, Sector = 512 bytes, Track = 200 sectors

The average time to access a block of 100 consecutive sectors is $\underline{16.34}$ ms.

Average access time= Seek Time + Rotation Latency + Transfer Time
Rotations per second=7200/60 =120 RPS
Rotation time in milliseconds=1000/120=8.33 ms
Time to transfer 100 sectors=(100/200)* 8.33=4.17 ms
Average access time=8 + 4.17 + 4.17=**16.34 ms**.

(10) Assuming the following data segment, and assuming that the first variable X is given the address **0x10010000**, then the addresses for variable Y and Z will be **0x10010004** and **0x1001000C.**

```
.data
X:      .byte  1, 2, 3
Y:      .half   4, 5, 6
Z:      .word 7, 8, 9
```

(11) The code given below prints the statement: <u>*ICS 233is so easy!!*</u>. *Note that the ASCII code for the line feed character is 10 and the ASCII code for the carriage return character is 13.*

```
MSG: .ascii  "Exam1" ,13
     .ascii " ICS 233"
     .ascii "is so easy !!",0

     li $v0, 4
     la $a0, MSG
     syscall
```

(12) Assume that the instruction j NEXT is at address 0x00401FC4 in the text segment, and the label NEXT is at address 0x0040003C. Then, the address stored in the assembled instruction for the label NEXT is <u>0x010000F</u>.

0x0040003C/4=0x010000F.

(13) Assume that the instruction beq $t0, $t1, NEXT is at address 0x00401FC4 in the text segment, and the label NEXT is at address 0x0040003C. Then, the address stored in the assembled instruction for the label NEXT is <u>0xF81D</u>.

(Next-(PC+4))/4=(0x0040003C -0x00401FC8)/4=0xFFFFE074/4=0xF81D.

(14) Assuming that $a0 contains an Alphabetic character, the instruction *ori $a0, $a0, 0x20* will guarantee that the character in $a0 is <u>lower case character</u>. Note that the ASCII code of character ‘A’ is 0x41 while that of character ‘a’ is 0x61.

(15) Assume that you are in a company that will market a certain IC chip. The cost per wafer is $3000, and each wafer can be diced into 2000 dies. The cost per good die is $3. Then, the yield of this manufacturing process is <u>50%</u>.

Cost per die = $3 = 3000 / (Y *2000). Thus, Y =3000 / 3 * 2000 = 50%.

[15 Points]

(Q2) Using only basic MIPS instructions, write the shortest sequence of instructions to implement each of the following pseudo instructions:

1. *sgt $t0, $t1, $t2* #$t0 is set to 1 if $t1 is greater than $t2

   slt $t0, $t2, $t1

2. *move $t0, $t1* # $t0 = $t1

   addu $t0, $0, $t1

3. *ble $t0, 5, Next* # branch to Next if $t0 is less than or equal 5

   slti $at, $t0, 6
   bne $at, $0, Next

4. *abs $t0, $t1* #$t0 is loaded with the absolute value of $t1

   sra $at, $t1, 31
   xor $t0, $at, $t1
   subu $t0, $0, $at

5. *ror $t0, $t0, 8* #$t0 is rotated to the right by 8 bits and stored in $t0

   sll $at, $t0, 24
   srl $t0, $t0, 8
   or $t0, $t0, $at

[15 Points]

(Q3) Answer the following questions. Show how you obtained your answer:

(i) Given that **TABLE** is defined as: **TABLE: .word 1, -1, 2, 50, -20, 16**

Determine the content of register**s $v0** and **$v1** after executing the following code:

```
        la      $a0, TABLE
        addi    $a1, $a0, 20
        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
```

This program finds the minimum value in TABLE and stores it in $v1 along with its address in $v0. Thus, $v1=-20 and $v0=address of TABLE+16.

(ii) Given that **TABLE** is defined as shown below, determine what will be printed by the following program:

TABLE: .ascii "0123456789ABCDEF"

```
        li $t0, 0x12EF67DC
        li $t3, 8
loop:
        rol  $t0, $t0, 4
        andi $a0,$t0, 15
        la $t1, TABLE
        addu $t1, $t1, $a0
        lb $t1, 0($t1)
        move $a0, $t1
        li $v0, 11
        syscall
        sub $t3, $t3, 1
        bne $t3, $zero, loop
```

This program prints the hexadecimal content of register $t0. Thus, it will print 12EF67DC.

(iii) Given that **Array** is defined as shown below, determine the content of **Array** after executing the following code:

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

```
    la $a0, Array
    li $a1, 4
    li $a2, 0
    li $a3, 2

    mul $t0, $a1, $a2
    add $t0, $t0, $a0
    mul $t1, $a1, $a3
    add $t1, $t1, $a0
Next:
    lb $t3, ($t0)
    lb $t4, ($t1)
    sb $t3, ($t1)
    sb $t4, ($t0)
    addi $t0, $t0, 1
    addi $t1, $t1, 1
    addi $a1, $a1, -1
    bnez $a1, Next
```

This program swaps the two rows in $a2 and $a3. $a1 contains the number of columns. Thus, the content of Array becomes:
9, 10, 11, 12, 5, 6, 7, 8, 1, 2, 3, 4

[20 Points]

(Q4) Write separate MIPS assembly programs to do each of the following using the smallest possible number of instructions.

(i) Multiply the content of register $s1 by 15.25.

```
sll $t0, $s1, 4          #t0=$s1*16
sub $t0, $t0, $s1        # $t0=$s1*15
sra $t1, $s1, 2          #$t1=$s1*1/4=$s1*0.25
add $t1, $t0, $t1        # $t1=$s1*15.25
```

(iii) Count the number of 1's in register $s1.

```
        xor $t2, $t2, $t2        #$t2=0 will hold the number of 1's
Next:
        andi $t1, $s1, 1
        add $t2, $t2, $t1
        srl  $s1, $s1, 1
        bne $s1, $0, Next
```

(iii) Ask the user to enter a character, c1. Then, in a new line ask the user to enter another character, c2, greater than the first character. Then, in a new line print the characters from character c1 until character c2 as shown in the format below. If the entered character is smaller than the first character ask the user to reenter the second character.

```
Enter a character: B
Enter another character greater than B: A
Enter another character greater than B: G
The range of entered characters is: B C D E F G
```

```
################# Data segment #####################
.data
msg1: .asciiz "Enter a character:"
msg2: .ascii "Enter another character greater than "
char: .byte 0,':',0
msg3: .asciiz "The range of entered characters is: "
################# Code segment #####################
.text
.globl main
main:       # main program entry

# Print msg1 asking the user to enter a character
    la $a0, msg1
    li $v0, 4
    syscall
```

```
# Read character & Store it
    li $v0, 12
    syscall
    move $s0, $a0
   la $t0, char
   sb $a0, ($t0)

# Print msg2 asking the user to enter another character
Again:
    la $a0, msg2
    li $v0, 4
    syscall

# Read 2nd character & Store it
    li $v0, 12
    syscall
    move $s1, $a0

# Check if 2nd character is greater than 1st character
    ble $s1, $s0, Again

# Print msg3 to print the range of entered character
    la $a0, msg3
    li $v0, 4
    syscall

# Print the characters in the range
    li $v0, 11
Next:
    move $a0, $s0
    syscall
    li $a0, ' '
    syscall
    addi $s0, $s0, 1
    ble $s0, $s1, Next
```

[20 Points]

(Q5) Write a MIPS assembly program, **BinarySearch,** to search an array which has been previously sorted in an ascending order. Each element in the array is a 32-bit signed integer. Assume that the address of the array to be searched in stored in $a0, the size (number of elements) of the array is stored in $a1, and the number to be searched is stored in $a2. If the number is found then the program returns in $v0 register the position of the number in the array. Otherwise, 0 is returned in $v0.

The pseudocode for the **BinarySearch** algorithm is given below:

```
BinarySearch (array, size, number) {
    lower = 0;
    upper = size-1;
    while (lower <= upper) {
        middle = (lower + upper)/2;
        if (number == array[middle])
            return middle;
        else if (number < array[middle])
            upper = middle–1;
        else
            lower = middle+1;
    }
    return 0;
}
```

```
################# Data segment #####################
.data
Array: .word 1, 2, 3, 4, 5, 6, 7, 8

################# Code segment #####################
.text
.globl main
main:   # main program entry

        la $a0, Array
        li $a1, 8                 # Number of elements in the array
        li $a2, 6                 # Number to be searched

        xor $t0, $t0, $t0         # lower=0
        addi $t1, $a1, -1         # upper=size-1
While:
        bgt $t0, $t1, EndWhile
        addu $t2, $t0, $t1        #  middle = (lower + upper)/2;
        srl     $t2, $t2, 1
        sll $t3, $t2, 2       # compute address of middle
        add $t3, $t3, $a0
        lw $t4, ($t3)         # array[middle]
```

```
        bne $a2, $t4, Elseif     # if (number == array[middle])
        move $v0, $t2            # return middle;
        j Done

Elseif:
        bge $a2, $t4, Else               # else if (number < array[middle])
        addi $t1, $t2, -1                # upper = middle–1
        j While
Else:
        addi $t0, $t2, 1                 #  lower = middle+1
        j While
EndWhile:
        li $v0, 0
Done:
```