Nov. 13, 2008

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Major Exam I

First Semester (081)

Time: 1:00-3:00 PM

Student Name : _KEY_____

Student ID. : _____

Question	Max Points	Score
Q1	35	
Q2	20	
Q3	15	
Q4	30	
Total	100	

Dr. Aiman El-Maleh

[35 Points]

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

(1) The smallest (negative) number that can be represented using 16-bit 2`s complement in hexadecimal is 8000 and the largest positive number in hexadecimal is 7FFF.

(2) Assuming 8-bit representation of numbers, the hexadecimal number 8E is equal to 142 as unsigned number and -114 in 2`s complement representation.

(3) Assuming variable Array is defined as shown below:

Array: .word 0x000000A0, 0x000000B0

The content of register $t0 after executing the following sequence of instructions is 000000B0.

```
la $t0, Array
lw $t0, 4($t0)
```

(4) With a 32-bit address bus and 32-bit data bus, the maximum memory size than can be accessed by a processor is 2^{32}=4G Byte and the maximum number of bytes that can be read or written in a single cycle is 32/8=4 Bytes.

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

- Rotation speed = 8000 RPM (rotations per minute)
- Average seek = 7 ms, Sector = 1024 bytes, Track = 250 sectors

The average time to access a block of 200 consecutive sectors is 16.75 ms.

Average access time = Seek Time + Rotation Latency + Transfer Time
Rotations per second = 8000/60 =133.33 RPS
Rotation time in milliseconds = 1000/133.33 = 7.5 ms
Rotation Latency = 7.5/2 = 3.75 ms
Time to transfer 200 sectors = (200/250)* 7.5 = 6 ms
Average access time = 7 + 3.75 + 6 = 16.75 ms.

(6) 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 0x10010006 and 0x1001000C**.**

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

(7) Assume that the CPU has just read a 32-bit instruction from the address 0x00400000. Then, the address of the next instruction that this CPU is going to read is 0x00400004.

(8) Assume that the instruction j NEXT is at address 0x00400030 in the text segment, and the label NEXT is at address 0x004000a8. Then, the address stored in the assembled instruction for the label NEXT is 0x004000a8/4=0x010002a.

(9) Assume that the instruction bne $t0, $t1, NEXT is at address 0x00400030 in the text segment, and the label NEXT is at address 0x004000a8. Then, the address stored in the assembled instruction for the label NEXT is (0x004000a8-0x00400034)/4=0x00400074/4= 0x001d.

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

(11) Assume you are in a company that will market a certain IC chip. The cost per wafer is $2000, and each wafer can be diced into 200 dies. The die yield is 80%. Then the cost per good die is $2000/(200*0.8)=$2000/160=$12.5**.**

(12) Assembly language produces more compact and more efficient code than high-level language.

(13) Cache memory is faster than random access memory but it is slower than registers .

(14) The instruction set architecture of a processor consists of the instruction set, programmer-accessible registers and memory.

(15) The difference between *slt* and *sltu* instructions is that *slt* is used for signed comparison while *sltu* is used for unsigned comparison.

[20 Points]

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

1. *andi $t0, 0x12345678* #$t0 is anded with the 32-bit value 0x12345678

```
lui $at, 0x1234
ori $at, $at, 0x5678
and $t0, $t0, $at
```

2. *bge $t0, $t1, Next* # branch to Next if $t0 is greater than or equal to $t1

```
slt $at, $t0, $t1
beq $at, $0, Next
```

3. *bgt $t0, 100, Next* # branch to Next if $t0 is greater than 100

```
slti $at, $t0, 101
beq $at, $0, Next
```

4. *neg $t0, $t1* #$t0 is loaded with the negative value of $t1

```
sub $t0, $0, $t1
```

5. *rol $t0, $t0, 12* #$t0 is rotated to the left by 12 bits and stored in $t0

```
srl $at, $t0, 20
sll $t0, $t0, 12
or $t0, $t0, $at
```

[15 Points]

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

(i) Determine the content of register **$s1** after executing the following code:

```
ori $s1, $zero, 4
sll $t0, $s1, 4
sub $t0, $t0, $s1
sra $t1, $s1, 2
add $s1, $t0, $t1
```

The content of $s1=61=0x3d. The code computes the result of multiplying the content of $s1 by 15.25 = 4*15.25=61.

(ii) Determine the content of register **$t2** after executing the following code:

```
        li $s1, 0x1b
        and $t2, $zero, $t2
Next:
        andi $t1, $s1, 1
        add $t2, $t2, $t1
        srl  $s1, $s1, 1
        bne $s1, $0, Next
```

The content of $t2=4=0x4. The code counts the number of 1's in register $s1.

(iii) Given that **TABLE** is defined as: **TABLE: .word 1, 10, -4, 5, 20, 3**

Determine the content of register**s $t2** after executing the following code:

```
        la      $t0, TABLE
        li      $t1, 6
        lw      $t2, ($t0)
loop:   addi    $t0, $t0, 4
        lw      $t3, ($t0)
        ble     $t3, $t2, skip
        move    $t2, $t3
skip:   addi    $t1, $t1, -1
        bne     $t1, $0, loop
```

The content of $t2=20=0x14. The code finds the maximum value in Table and stores it in $t2.

[30 Points]

(Q4) Merge sort is a technique to combine two sorted arrays. Merge sort takes two sorted input arrays X and Y, say of size m and n, and produces a sorted array Z of size m+n that contains all elements of the two input arrays. The pseudo code of merge sort is as follows:

```
MergeSort (X, Y, Z, m, n)
    i:=0    {index variables for arrays X, Y, and Z}
    j:=0
    k:=0
    while (i<m)AND (j<n)
        if (X[i] ≤ Y[j]) then
            Z[k]:=X[i]
            k:=k+1
            i:=i+1
        else
            Z[k]:=Y[j]
            k:=k+1
            j:=j+1
        end if
    end while
    if (i<m) then
        while (i<m)
            Z[k]:=X[i]
            k:=k+1
            i:=i+1
        end while
    else
        while (j<n)
            Z[k]:=Y[j]
            k:=k+1
            j:=j+1
        end while
    end if
end MergeSort
```

Write a MIPS assembly program to implement **MergeSort** to merge sort two arrays of integers (i.e. 32-bit signed numbers) in an **ascending** order. Assume that the address of X array, Y array and Z array are stored in registers $s0, $s1 and $s2, respectively. Also, assume that m, the size of array X, is stored in register $s3 and n, the size of array Y, is stored in register $s4. Your code should not change the content of registers $s0-$s4 after execution.

```
    li $t0, 0                   # i =0
    li $t1, 0                   # j =0
    li $t2, 0                   # k =0
While1:
    bgeu $t0, $s3, Endwhile1    # while (i<m) AND (j<n)
```

```
        bgeu $t1, $s4, Endwhile1

        sll $t3, $t0, 2                 # get X[i] and store in $t3
        add $t3, $t3, $s0
        lw $t3, ($t3)
        sll $t4, $t1, 2                 # get Y[j] and store in $t4
        add $t4, $t4, $s1
        lw $t4, ($t4)
        sll $t5, $t2, 2                 # compute address of Z[k]
        add $t5, $t5, $s2
        bgt $t3, $t4, Else1             # if (X[i] <= Y[j]) then
        sw $t3, ($t5)                   # Z[k]:=X[i]
        addi $t2, $t2, 1                # k:=k+1
        addi $t0, $t0, 1                # i:=i+1
        j Endif1
Else1:
        sw $t4, ($t5)                   # Z[k]:=Y[j]
        addi $t2, $t2, 1                # k:=k+1
        addi $t1, $t1, 1                # j:=j+1
Endif1:
        j While1
Endwhile1:
        bgeu $t0, $s3, Else2            # if (i<m) then
While2:
        bgeu $t0, $s3, Endwhile2        # while (i<m)
        sll $t3, $t0, 2                 # get X[i] and store in $t3
        add $t3, $t3, $s0
        lw $t3, ($t3)
        sll $t5, $t2, 2                 # Z[k]:=X[i]
        add $t5, $t5, $s2
        sw $t3, ($t5)
        addi $t2, $t2, 1                # k:=k+1
        addi $t0, $t0, 1                # i:=i+1
        j While2
Endwhile2:
        j Endif2
Else2:
While3:
        bgeu $t1, $s4, Endwhile3        # while (j<n)
        sll $t4, $t1, 2                 # get Y[j] and store in $t4
        add $t4, $t4, $s1
        lw $t4, ($t4)
        sll $t5, $t2, 2                 # Z[k]:=Y[j]
        add $t5, $t5, $s2
        sw $t4, ($t5)
        addi $t2, $t2, 1                # k:=k+1
        addi $t1, $t1, 1                # j:=j+1
        j While3
Endwhile3:
Endif2:
```