August 13, 2007

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Major Exam II

Summer Semester (063)

Time: 7:00-9:30 PM

Student Name : _KEY_____

Student ID. : _____

Question	Max Points	Score
Q1	20	
Q2	16	
Q3	16	
Q4	14	
Q5	8	
Q6	16	
Q7	10	
Total	100	

Dr. Aiman El-Maleh

[20 Points]

(Q1) Suppose that you are given a positive integer. You can add individual digits of this number to get another integer. If we repeat this procedure, eventually we will end up with a single digit. Here is an example:

7391928 = 7+3+9+1+9+2+8 = 39
39 = 3+9 = 12
12 = 1+2= 3

Write a procedure, **ToSDigit**, that receives a positive integer in $a0 and returns a single digit in register $v0 according to the method described above. It is required that the procedure **preserves the content of all used registers** by saving and restoring them on the stack. Then, write a program to read a positive integer from the user and display the **single digit** obtained by the above procedure.

A **sample execution** of the program is:

Enter a number: 7391928
Result is: 3

```
################## Data segment ####################
.data
MSG1: .asciiz "Enter a number: "
MSG2: .asciiz "Result is: "

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

# Getting the number
        li $v0, 4
        la $a0, MSG1
        syscall
        li $v0, 5
        syscall

# Calling the procedure
        move $a0, $v0
        jal ToSDigit

# Displaying the result
        move $t0, $v0
        li $v0, 4
        la $a0, MSG2
        syscall
```

```
                li $v0, 1
        move $a0, $t0
        syscall

# Exit program
        li $v0, 10
        syscall

# The procedure receives a positive integer in $a0 and returns a single digit in register $v0
ToSDigit:

# Saving used registers on the stack
        addi $sp, $sp, -4       #saving register $t0
        sw $t0, ($sp)
        addi $sp, $sp, -4       #saving register $t1
        sw $t1, ($sp)
        addi $sp, $sp, -4       #saving register $t2
        sw $t2, ($sp)
        addi $sp, $sp, -4       #saving register $t3
        sw $t3, ($sp)

        li $t0, 10
        move $t1, $a0
Next:
        xor $t3, $t3, $t3       # Holds the sum of the digits
# Compute the sum of the digits
Again:
        divu $t1, $t0
        mflo $t1                        # $t1 = quotient
        mfhi $t2                        # $t2 = remainder
        addu $t3, $t3, $t2
        bnez $t1, Again
# Check if the sum is greater than 9
        move $t1, $t3
        bge $t1, $t0, Next
        move $v0, $t1

# Restoring  registers from the stack
        lw $t3, ($sp)
        addi $sp, $sp, 4        #restoring register $t3
        lw $t2, ($sp)
        addi $sp, $sp, 4        #restoring register $t2
        lw $t1, ($sp)
        addi $sp, $sp, 4        #restoring register $t1
        lw $t0, ($sp)
        addi $sp, $sp, 4        #restoring register $t0

        jr $ra
```

[16 Points]

(Q2) Given that **Multiplicand=1010** and **Multiplier=1001**.

(i) Using the **refined unsigned multiplication** hardware, show the **unsigned** multiplication of **Multiplicand** by **Multiplier**. The result of the multiplication should be an 8 bit **unsigned** number in HI and LO registers. Show the steps of your work.

Iteration		**Multiplicand**	**Carry**	**Product = HI,LO**
0	Initialize (LO = Multiplier)	1010		0000 100**1**
1	LO[0] = 1 => ADD		0	1010 1001
	Shift Product = (HI, LO) right 1 bit	1010		0101 010**0**
2	LO[0] = 0 => Do Nothing		0	0101 0100
	Shift Product = (HI, LO) right 1 bit	1010		0010 101**0**
3	LO[0] = 0 => Do Nothing		0	0010 1010
	Shift Product = (HI, LO) right 1 bit	1010		0001 010**1**
4	LO[0] = 1 => ADD		0	1011 0101
	Shift Product = (HI, LO) right 1 bit			**0101 1010**

(ii) Using the **refined signed multiplication** hardware, show the **signed** multiplication of **Multiplicand** by **Multiplier**. The result of the multiplication should be an 8 bit **signed** number in HI and LO registers. Show the steps of your work.

Iteration		**Multiplicand**	**Sign**	**Product = HI,LO**
0	Initialize (LO = Multiplier)	1010		0000 100**1**
1	LO[0] = 1 => ADD		1	1010 1001
	Shift Product = (HI, LO) right 1 bit	1010		1101 010**0**
2	LO[0] = 0 => Do Nothing		1	1101 0100
	Shift Product = (HI, LO) right 1 bit	1010		1110 101**0**
3	LO[0] = 0 => Do Nothing		1	1110 1010
	Shift Product = (HI, LO) right 1 bit	1010		1111 010**1**
4	LO[0] = 1 => SUB (ADD 2's comp.)	0110	0	0101 0101
	Shift Product = (HI, LO) right 1 bit			**0010 1010**

[16 Points]

(Q3) Given that **Dividend=1010** and **Divisor=0100**.

(i) Using the **refined unsigned division** hardware, show the **unsigned** division of **Dividend** by **Divisor**. The result of division should be stored in the Remainder and Quotient registers. Show the steps of your work.

Iteration		**Remainder**	**Quotient**	**Divisor**	**Difference**
0	Initialize	0000	1010	0100	
1	1: SLL, Difference	0001	0100	0100	1101
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0010	1000	0100	1110
	2: Diff < 0 => Do Nothing				
3	1: SLL, Difference	0101	0000	0100	0001
	2: Rem = Diff, set lsb Quotient	0001	000**1**		
4	1: SLL, Difference	**0010**	**0010**	0100	1110
	2: Diff < 0 => Do Nothing				

(ii) Using the **refined unsigned division** hardware, show the **signed** division of **Dividend** by **Divisor**. The result of division should be stored in the Remainder and Quotient registers. Show the steps of your work.

First, we convert the dividend into a positive number by taking its 2's complement. Thus, the dividend becomes 0110. Then, we perform unsigned division as shown below.

Iteration		**Remainder**	**Quotient**	**Divisor**	**Difference**
0	Initialize	0000	0110	0100	
1	1: SLL, Difference	0000	1100	0100	1100
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0001	1000	0100	1101
	2: Diff < 0 => Do Nothing				
3	1: SLL, Difference	0011	0000	0100	1111
	2: Diff < 0 => Do Nothing				
4	1: SLL, Difference	0110	0000	0100	0010
	2: Rem = Diff, set lsb Quotient	**0010**	**0001**		

Since Quotient sign should be negative we take its 2's complement. Thus, Quotient=1111.
Also, since remainder sign is negative, we take its 2's complement. Thus, Remainder=1110.

[14 Points]

(Q4) Given the following two floating-point numbers in single-precision format:

X=1100 0011 1100 0000 0000 0000 0000 0001
Y=0011 1110 1000 0000 0000 0010 0011 0001

(i) Perform the floating-point operation **X-Y** rounding the result to the nearest even, using guard, round and sticky bits. Represent the result in single-precision format.

Since the sign of X is negative and the operation is subtraction, then the effective operation is addition and the sign of the result is negative.

```
      1.100 0000 0000 0000 0000 0001 000      x 2^8
+     1.000 0000 0000 0010 0011 0001 000      x 2^-2
=     1.100 0000 0000 0000 0000 0001 000      x 2^8
+     0.000 0000 0010 0000 0000 0000 101      x 2^8

=    -1.100 0000 0010 0000 0000 0001 101      x 2^8
```

Then, we round to the nearest even and we add a 1. Thus, the result will be:

```
     -1.100 0000 0010 0000 0000 0010          x 2^8
```

The result represented in single-precision format is given below:

```
1100 0011 1100 0000 0010 0000 0000 0010
```

(ii) Perform the floating-point operation **X*Y** rounding the result to the nearest even. Represent the result in single-precision format.

```
     -1.100 0000 0000 0000 0000 0001          x 2^8
*    +1.000 0000 0000 0010 0011 0001          x 2^-2

=    +1.000 0000 0000 0010 0011 0001          x 2^-2
*    -1.100 0000 0000 0000 0000 0001          x 2^8
```

The result for the exponent will be $2^{8-2}=2^6$
To obtain the result of the fraction, we multiply the two fractions as shown below:

```
                                 100000000000001000110001
+       100000000000001000110001
+     100000000000001000110001
      1100000000000011010010101000000000001000110001
```

Thus, the fraction result will be equal to the following since we will have 46 bits in the fraction part:

1.1000000000000110100101010000000000001000110001

Rounding the result to the nearest even makes us add 1 since the bit next to the least significant bit is 1 and the remaining bits are not 0. Thus, the result of multiplication will be:

-1.100 0000 0000 0011 0100 1011 x 2^6

The result expressed in single-precision format is:

1100 0010 1100 0000 0000 0011 0100 1011

[8 Points]

(Q5) Suppose that a program runs in 150 seconds on a machine, with multiply operations responsible for 40 seconds of this time, divide operations responsible for 60 seconds of this time. The remaining time is taken by the remaining operations. Suppose that a new implementation of the machine has improved the execution time of the multiplier by a factor of 3 and the execution time of the divider by a factor of 2. Determine the new execution time and the speedup of the program on the new implementation.

Execution time of new implementation = 40/3 + 60/2 + 50 = 93.33 seconds

Speedup = 150/93.33 = 1.607

[16 Points]

(Q6) Given the following instruction mix of a program on a RISC processor:

Class	CPI	Frequency
ALU	3	20%
Branch	2	30%
Jump	1	25%
Load	5	15%
Store	4	10%

(i) What is the average CPI?

Average CPI=3*0.2 + 2*0.3+1*0.25+5*0.15+4*0.10=2.6

(ii) Assuming that the processor has a clock rate of 3 GHz, determine MIPS.

MIPS= Clock Rate / (CPI*10^6)=3* 10^9/ 2.6*10^6=1.154*10^3

(iii) What is the percent of time used by each instruction class?

Class	Percentage of Time
ALU	3*0.2/2.6=0.6/2.6=23.07%
Branch	2*0.3=0.6/2.6=23.07%
Jump	1*0.25/2.6=0.25/2.6=9.62%
Load	5*0.15/2.6=0.75/2.6=28.85%
Store	4*0.10/2.6=0.4/2.6=15.38%

(iv) How much faster would the program run if load and store time are reduced to 3 cycles, and two ALU instructions could be executed at once, assuming that the cycle time has increased by 10% and the instruction count has increased by 15%?

New Average CPI=**1.5***0.2 + 2*0.3+1*0.25+**3***0.15+**3***0.10=1.9

Speedup = Old Execution Time / New Execution Time
= IC * 2.6 * Cycle Time / 1.1* IC * 1.9 * 1.15 * Cycle Time
= 2.6 / 1.1*1.9*1.15 = 2.6/2.4035=1.082

[10 Points]

(Q7) Assume that a processor has four 8-bit registers: R0, R1, R2, and R3. You are required to design a register file that allows reading the value of one of the registers specified by the field RS[1:0] and writing into one of the registers specified by the field RD[1:0]. Show all required control signals for the register file.