May 17, 2008

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Major Exam II

Second Semester (072)

Time: 8:00-10:00 PM

Student Name : _KEY_____

Student ID. : _____

Question	Max Points	Score
Q1	20	
Q2	15	
Q3	16	
Q4	10	
Q5	14	
Q6	10	
Q7	15	
Total	100	

Dr. Aiman El-Maleh

[20 Points]

(Q1) You are required to write a procedure that receives two parameters N and K in registers $a0 and $a1 and computes the result of multiplying NxN-1xN-2x…xK and stores the result in $v0. The procedure is described in a recursive way as follows:

RangeMul(N,K){ If N=K return N else return N*RangeMul(N-1,K) }

Implement the given recursive procedure, RangeMul, in MIPS assembly programming with the minimal number of instructions. Then, write a program to ask the user to enter two integers N and K and print RangeMul(N,K).

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

Enter first integer N: 5
Enter second integer K: 3
Result is: 60

A summary of syscall services you can use is given below:

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read

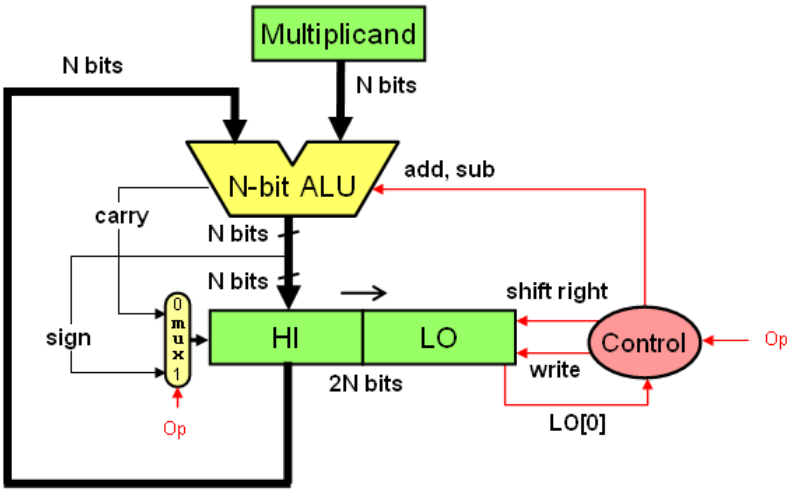
```
################## Data segment ####################
.data
msg1: .asciiz "Enter first integer N:"
msg2: .asciiz "Enter second integer K:"
msg3: .asciiz "Result is:"

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

li $v0, 4
la $a0, msg1
syscall
li $v0, 5
syscall
move $t0, $v0

li $v0, 4
la $a0, msg2
```

```
            syscall
li $v0, 5
syscall
move $a1, $v0

li $v0, 4
la $a0, msg3
syscall

move $a0, $t0

jal RangeMul

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


li $v0, 10        # Exit program
syscall

RangeMul:

bne $a0, $a1, Skip
move $v0, $a0
jr $ra
Skip:
addi $sp, $sp, -8
sw $a0, ($sp)
sw $ra, 4($sp)
addi $a0, $a0, -1
jal RangeMul
lw $t0, ($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $v0, $t0
jr $ra
```

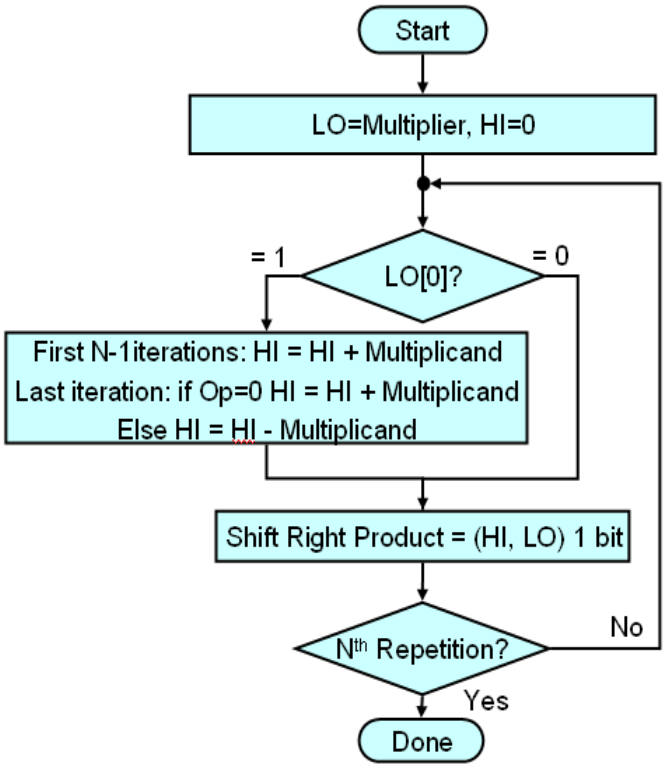
[15 Points]

(Q2) You are required to design a circuit that can be used to perform either signed or unsigned multiplication of two N-bit operands A and B depending on an input signal OP. When OP=0, the circuit will perform unsigned multiplication. Otherwise, it will perform signed multiplication. Show the algorithm that will be used with the circuit in performing the multiplication operations.

Circuit:



Algorithm:



[15 Points]

(Q3) Given that **Dividend=1001** 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	1001	0100	
1	1: SLL, Difference	0001	0010	0100	1101
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0010	0100	0100	1110
	2: Diff < 0 => Do Nothing				
3	1: SLL, Difference	0100	1000	0100	0000
	2: Rem = Diff, set lsb Quotient	0000	100**1**		
4	1: SLL, Difference	**0001**	**0010**	0100	1101
	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 0111. Then, we perform unsigned division as shown below.

Iteration		**Remainder**	**Quotient**	**Divisor**	**Difference**
0	Initialize	0000	0111	0100	
1	1: SLL, Difference	0000	1110	0100	1100
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0001	1100	0100	1101
	2: Diff < 0 => Do Nothing				
3	1: SLL, Difference	0011	1000	0100	1111
	2: Diff < 0 => Do Nothing				
4	1: SLL, Difference	0111	0000	0100	0011
	2: Rem = Diff, set lsb Quotient	**0011**	**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=1101.

[10 Points]

(Q4) Consider a simplified 8-bit floating point representation following the general guidelines of the IEEE format in representing normalized, denormalized, Nan, infinity and 0. Suppose that the number of bits used for the exponent is 3 and for the fraction is 4 bits.

(i) Determine the smallest and largest positive values of normalized numbers.

(ii) Determine the smallest and largest positive values of denormalized numbers.

(iii) Determine the representation used for +0 and +∞.

(iv) What is the largest and smallest error in this representation?

(i)

Number	S	Exp	Fraction	E	Value
Smallest normalize number	0	001	0000	1-3=-2	1*1/4=1/4
Largest normalize number	0	110	1111	6-3=3	31/2=15.5

(ii)

Number	S	Exp	Fraction	E	Value
Smallest denormalize number	0	000	0001	-2	1/16*1/4=1/64
Largest denormalize number	0	000	1111	-2	15/64≈1/4

(iii)

Number	S	Exp	Fraction
+0	0	000	0000
+∞	0	111	0000

(iii)

If we consider the largest values with E=110=3, we can see that these values range from 8 to 15.5. The values in this range are: 8, 8.5, 9, 9.5 …., 15.5. Thus, the largest error is 0.5/2=0.25.

If we consider the smallest normalized values with E=001=-2, we can see that these values range from 1/4 to 31/16. The values in this range are: 1/4=16/64, 17/64, 18/64, 19/64 …., 31/16. Thus, the smallest error is 1/64*2=1/128.

Note that the same magnitude of error occurs in the representation of denormalized numbers as they are in the range of 1/64, 2/64,..,15/64.

[14 Points]

(Q5) Given the following two floating-point numbers in single-precision format:

X=0100 0011 1000 0000 0000 0000 0000 0000
Y=1011 1000 0000 1100 0000 0000 0000 0000

(i) Determine the decimal value of the two numbers X and Y.

(ii) 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.

(i)

$$X = +(1.0)_2 \times 2^8 = 256$$
$$Y = -(1.00011)_2 \times 2^{-15} = -1.09375 \times 2^{-15} = -3.34 \times 10^{-5}$$

(ii)

$$\begin{array}{lll} & 1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 & \times 2^8 \\ - & 1.000\ 1100\ 0000\ 0000\ 0000\ 0000\ 000 & \times 2^{-15} \\ \hline \end{array}$$

First, we align by shifting it right by 23 bits and then we perform a subtraction operation:

$$\begin{array}{lll} & & \text{GRS} \\ = & 1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 & \times 2^8 \\ - & 0.000\ 0000\ 0000\ 0000\ 0000\ 0001\ 001 & \times 2^8 \\ \hline = & 01.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 & \times 2^8 \\ + & 11.111\ 1111\ 1111\ 1111\ 1111\ 1110\ 111 & \times 2^8 \\ \hline = & 00.111\ 1111\ 1111\ 1111\ 1111\ 1110\ 111 & \times 2^8 \\ = & +0.111\ 1111\ 1111\ 1111\ 1111\ 1110\ 111 & \times 2^8 \end{array}$$

Then, we normalize the result by shifting by one bit to the left and subtracting 1 from the exponent:

$$= +1.111\ 1111\ 1111\ 1111\ 1111\ 1101\ 111 \times 2^7$$

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

$$= +1.111\ 1111\ 1111\ 1111\ 1111\ 1110\ 111 \times 2^7$$

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

0100 0011 0111 1111 1111 1111 1111 1110

[10 Points]

(Q6) You are going to enhance a computer, and there are two possible improvements: either make multiply instructions run four times faster than before, or make memory access instructions run two times faster than before. You repeatedly run a program that takes 100 seconds to execute. Of this time, 20% is used for multiplication, 50% for memory access instructions, and 30% for other tasks. What will the speedup be if you improve only multiplication? What will the speedup be if you improve only memory access? What will the speedup be if both improvements are made?

Speedup =1 / (f/s + (1-f))

1. Speedup due to improving multiplier alone = 1 / (0.2/4+0.8)=1/(0.05+0.8)=1/0.85=1.18

2. Speedup due to improving memory alone = 1 / (0.5/2+0.5)=1/(0.25+0.5)=1/0.75=1.33

3. Speedup due to improving multiplier & memory =
1 / (0.2/4+0.5/2+0.3)=1/(0.05+0.25+0.3)=1/0.6=1.67

[15 Points]

(Q7) You are the lead designer of a new processor. The processor design and compiler are complete, and now you must decide whether to produce the current design as it stands or spend additional time to improve it. You discuss this problem with your hardware engineering team and arrive at the following options:

a. *Leave the design as it stands.* Call this base computer *Mbase*. It has a clock rate of 500 MHz, and the following measurements have been made using a simulator:

Instruction Class	CPI	Frequency
A	2	40%
B	3	25%
C	3	25%
D	5	10%

b. *Optimize the hardware.* The hardware team claims that it can improve the processor design to give it a clock rate of 600 MHz. Call this computer *Mopt*. The following measurements were made using a simulator for *Mopt*:

Instruction Class	CPI	Frequency
A	2	40%
B	2	25%
C	3	25%
D	4	10%

(i) What is the average CPI for each computer?

(ii) What are the MIPS ratings for *Mbase* and *Mopt*?

(iii) How much faster is *Mopt* more than *Mbase* ?

(i) Mbase CPI = 2*0.4+3*0.25+3*0.25+5*0.1=0.8+0.75+0.75+0.5=2.8
Mopt CPI = 2*0.4+2*0.25+3*0.25+4*0.1=0.8+0.5+0.75+0.4=2.45

(ii) MIPS Rate = #instructions/(Execution Time * 10^6)= Clk Rate/(CPI * 10^6)
Mbase MIPS = (500*0^6)/(2.8*0^6)=178.57
Mopt MIPS = (600*0^6)/(2.45*0^6)=244.89

(iii) Speedup = 244.89/178.57=1.37