Ho Chi Minh City University of Technology
FACULTY OF COMPUTER SCIENCE&ENGINEERING

Laboratory Report
Computer Architecture
Assignment 1: Multiply or divide 2 integers without directly using the said MIPS' commands.
Student Name Student ID
Cao Minh Thông 1915351

Ho Chi Minh City,12/2020

1. Introduction:

Design and implement an Arithmetic/Logic Unit(ALU) with the ability to multiply or divide 2 integers input from user, with the ability to handle hexadecimal numbers or decimal numbers. All work must be done using the MIPS commands and cannot directly use available multiply or divide commands from the MIPS instructions set.

2. Analysis:

We need to analyze the requirements set by the assignment.
Firstly implement an translator which can translate hexadecimal input into binary in order for the controller to work on. Integer is supported natively on the microchip so we can forward it directly to the ALU without further processing.
Then, since the assignment requires implementing the multiplier/divider without directly involve said instructions given by the MIPS, we need to implement it based on the algorithms given in Chapter Arithmetics.
After getting the results from the above mentioned ALU, we need to turn it into the number base preferred by the user, so we need to implement a converter with the ability to turn back into hexadecimal output in case the user do choose hexadecimal mode.

3. Implementation:

Working mode decision block:

```
la $a0, input
li $v0, 4
syscall
li $v0, 5
syscall
move $s4,$v0
beq $s4, 1, Int
beq $s4,2, Hex
beq $s4,3, Float
j Exception
```

Hexadecimal to binary converter:

```
Hexconvert:
la $a0, mess # Message buffer address
```

```
li $v0, 4
syscall
la $a0, hexin #Input buffer address, will be our intermadiate
li $a1, 10 # Maximum accepted characters, derived from 32bits / 4bits each positions + null terminate
li $v0, 8 # Prompt for input string
syscall
#la baseaddress, hexin # Our intermediate buffer address
#move termaddress, baseaddress
la $t5, hexin
move $t6,$t5
li $a1, 0
li $t7,0
cont:
        lb $a0, 0($t6) #Load character at address
        beq $a0, 10, Hexcon #Look for NULL termination
        subu $a1, $t6, $t5
        addi $t6, $t6, 1#Offset 1 for every bytes we compare
        j cont
Hexcon:
lb $t9, ($t5)
beqz $t9, Hexout
li $t2, 0
sle $t2, $t9, 57
bnez $t2, numex
li $t2, 0
sge $t2, $t9, 65
bnez $t2, charex
j Exception
numex:
li $t2, 0
slti $t2, $t9, 47
bnez $t2, Exception
subi $t9, $t9, 48 #Substract the difference
j Hconti
charex:
li $t2, 0
sgt $t2, $t9, 70
bnez $t2, Exception
subi $t9, $t9, 55 #Substract the difference
j Hconti
Hconti:
addi $t5, $t5, 1
sll $t7, $t7, 4
add $t7,$t7, $t9
beq $t5,$t6, Hexout
j Hexcon
Hexout:
subi $a1,$a1,1
move $s5,$t7
jr $ra
```

Now that we have our input ready, we must calculate the output, which means implementing the multiplier/divider for 2 integers itself, without directly using the instructions given by MIPS.
We must implement them using the algorithms taught in chapter3: Arithmetic for Computers.
Our Multiplication algorithm:

```
Mul:
Mulstart:
move $s6, $a0
move $s7, $a1
move $t6, $s6
move $t7, $s7
li $t2, 0
sgt $t2, $t6, 0
bnez $t2, Mul1
sub $t6, $0,$t6
Mul1:
li $t2, 0
sgt $t2, $t7, 0
bnez $t2, Mul2
sub $t7, $0,$t7
Mul2:
li $t4, 0
li $a0, 0
Mulcon:
andi $t9,$t7,1 #test multiplier0
beqz $t9, Mul00
add $a0, $a0, $t6 #mul0 = 1
Mul00:
sll $t6,$t6, 1
srl $t7,$t7, 1
addi $t4,$t4,1
beq $t4,31,Mulexit
j Mulcon
Mulexit:
li $a1, 0
li $t5, 0
slt $t6, $s6, $0
slt $t7, $s7, $0
xor $t5, $t6, $t7
beqz $t5, ALUexit
sub $a0, $0, $a0
j ALUexit
```

Division algorithm:

```
Div:
Divstart:
#move $s6, $a0
#move $s7, $a1
move $t6, $s6
move $t7, $s7
bgtz $t6, Div1
sub $t6, $0,$t6
```

```
Div1:
bgtz $t7, Div2
sub $t7, $0,$t7
Div2:
li $a0, 0
li $a1, 0
#beqz $s6, DivZero
beqz $s7, DivZero
Div00:
sub $t6, $t6, $t7
bltz $t6, Divdone
addi $a1, $a1, 1
j Div00
Divdone:
li $t5, 0
add $t6, $t6, $t7
add $a0,$t6, $0
slt $t6, $s6, $0
slt $t7, $s7, $0
xor $t5, $t6, $t7
beqz $t6, Dends
sub $a0, $0, $a0
Dends:
beqz $t5, ALUexit
sub $a1, $0, $a1
j ALUexit
DivZero:
la $a0, ecpt
li $v0, 4
syscall
j exit
```

After we have implemented our Arithmetic Unit, we now need to output the results back to the user, which means implementing a hexadecimal converter to turn our binary into hex base.

```
HEX:
move $a0,$s6
la $a1, text1
jal Hexoutput
la $a0, D1
li $v0,4
syscall
la $a1, text1
li $v0, 4
syscall
move $a0,$s7
la $a0, text1
jal Hexoutput
la $a0, D2
li $v0, 4
syscall
la $a0, text1
li $v0, 4
```

```
syscall
j exit
Hexoutput:
move $t0, $a0
li $t4,0
Hexoutcont:
andi $t8, $t0, 15
srl $t0, $t0, 4
# Store the least 4 bits, shift them away
bltz $t8, Exception
bgt $t8, 15, Exception
bge $t8, 10, tenover
bgez $t8, nineunder
tenover:
addi $t8, $t8, 55
j here
nineunder:
addi $t8, $t8, 48
j here
here:
sb $t8, ($a1)
#sb $0, 4($a1)
addi $a1, $a1,1
addi $t4, $t4,1
beq $t4, 8, return
j Hexoutcont
Exception:
la $a0, expt
li $v0, 4
syscall
exit:
```

4. Documentation:

Register nameFunction:

$V0 Used for calling system calls and intermediate when asking for user input

$A0 Argument 0, used for passing values to function, used as multiplicand or dividend or message buffer address

$A1 Argument 1, used for passing values to function,used as multiplier or divisor.

$T2 Branch register, used as intermediate for non standard branch

$T4 Used as a counter for multiplication limit and hexconverter limit

$T5 Temporary register for storing input string used by the hexconverter

$T6 Temporary register for direct number manipulation used by the functions, contains multiplicand or dividend

$T7 Temporary register for direct number manipulation used by the functions, contains multiplier or divisor

$S4 Number base mode memory, 1 for int, 2 for hex

$S6 Main storage for the first user inputted number

$S7 Main storage for the second user inputted number

$RA Return address used for jumping back to function call

5. Testcases:

Floating point:

Explanation: Floating isn't required to implement in the assignment. So it isn't implemented here.

Decimal cases:
Divide by zero:
Explanation: Divide by zero gives us an answer of infinity so a warning message is displayed.
Zero divide other numbers:

Explanation: Zero is divisible to every number and gives us the answer of 0, this is also applicable to multiply by zero.
Signed decimals division:

Signed decimals multiplication:

Explanation: 2147583647 is nearing the limit of 32 bit allowable on the MIPS so in this case it is an overflow example.
Hexadecimal cases:
Divide by zero:

Explanation: Similar to division by zero in decimal case, but the increase in memory instructions is the result of storing user input and output as text.
Multiplication:

0ABCDEFF divided by 3:

In case the user input something inappropriate :

Assuming every instructions has the CPI of 1, the program is loaded and executed onto a MIPS microcontroller with a clock of 2GHz, we can now calculate the time it takes for the program to execute with a corresponding output.
From above, we can see that divide by zero uses the least instructions, we can then base our calculations upon the increase in instructions to deduct the hex converter's number of instructions.

We can see that the converter block takes 10 times as much as the normal input block.

Type	ALU mode	Input 1	Input 2	Cycles
Float	X	X	X	18
Dec	Div	X	0	49
Dec	Div	0	X	78
Dec	Div	-500	-9	300

Dec	Div	-500	9	301	
Dec	Mul	-100	5	328	
Dec	Mul	-2000	-5	327	
Hex	Div	X	0	490	
Hex	Div	ABCDEFF	3	1206	
Hex	Mul	ABCD	F	1113	

From the above table we can assume that the program always stay below 1500 instructions, by then we can calculate that the program will stays below 0.75 microsec.