

COE 301/ICS 233, Term 171

Computer Architecture & Assembly Language

HW# 4 Solution

Q.1. Write a MIPS assembly program to perform **signed multiplication** of 32-bit numbers using the algorithm studied in class. The program should ask the user to enter two integers and then display the result of multiplication. If the result cannot fit in 32-bit then the program should indicate that there is overflow. Test your program using the following numbers:

1. -1×-1
2. 100×-2
3. 0×10
4. 2147483647×2

A sample execution of the program is shown below:

```
Enter the multiplier: 100
Enter the multiplicand: -2
Result of multiplication = -200
```

```
# Description: A program to implement 32-bit signed multiplication
##### Data segment #####
.data
msg1: .asciiz "Enter the multiplier:"
msg2: .asciiz "Enter the multiplicand:"
msg3: .asciiz "Result of multiplication = "
msg4: .asciiz "\nThere is overflow"
##### Code segment #####
.text
.globl main
main:    # main program entry

# Getting the first integer
li $v0, 4
la $a0, msg1
syscall
li $v0, 5
syscall
move $s1, $v0
```

Getting the second integer

```
li $v0, 4
la $a0, msg2
syscall
li $v0, 5
syscall
move $s2, $v0
```

Performing signed multiplication

```
xor $s3, $s3, $s3      # HI=0
li $s4, 32             # Loop counter
li $s0, 1
```

Loop:

```
andi $t0, $s1, 1
beqz $t0, Shift
beq $s4, $s0, Subtract
addu $s3, $s3, $s2
j Shift
```

Subtract:

```
subu $s3, $s3, $s2
```

Shift:

```
andi $t1, $s3, 1
ror $t1, $t1, 1
srl $s1, $s1, 1
or $s1, $s1, $t1
sra $s3, $s3, 1
addi $s4, $s4, -1
bnez $s4, Loop
```

Displaying Result

```
li $v0, 4
la $a0, msg3
syscall
```

```
li $v0, 1
move $a0, $s1
syscall
```

Checking for overflow

```
bltz $s1, Negative
bnez $s3, Overflow
j Done
```

Negative:

```
li $t0, 0xffff
beq $s3, $t0, Done
```

```

Overflow:
    li $v0, 4
    la $a0, msg4
    syscall
Done:
    li $v0, 10    # Exit program
    syscall

```

Results of running the program on the given test cases:

```

Enter the multiplier:**** user input : -1
Enter the multiplicand:**** user input : -1
Result of multiplication = 1

```

```

Enter the multiplier:**** user input : 100
Enter the multiplicand:**** user input : -2
Result of multiplication = -200

```

```

Enter the multiplier:**** user input : 0
Enter the multiplicand:**** user input : 100
Result of multiplication = 0

```

```

Enter the multiplier:**** user input : 2147483647
Enter the multiplicand:**** user input : 2
Result of multiplication = -2
There is overflow

```

Q.2. Write a MIPS assembly program to perform **signed division** of 32-bit numbers using the algorithm studied in class. The program should ask the user to enter two integers and then display the result of division displaying both the quotient and remainder. Test your program using the following numbers:

1. $+17 \div +3$
2. $+17 \div -3$
3. $-17 \div +3$
4. $-17 \div -3$

A sample execution of the program is shown below:

```

Enter the dividend: 17
Enter the divisor: -3
Result of division: Quotient = -5 Remainder = 2

```

```

# Description: A program to implement 32-bit signed multiplication
##### Data segment #####
.data

```

```

msg1: .asciiz "Enter the dividend:"
msg2: .asciiz "Enter the divisor:"
msg3: .asciiz "Result of division: "
msg4: .asciiz "Quotient = "
msg5: .asciiz " Remainder = "
##### Code segment #####
.text
.globl main
main: # main program entry

# Getting the dividend
    li $v0, 4
    la $a0, msg1
    syscall
    li $v0, 5
    syscall
    move $s1, $v0
    xor $s5, $s5, $s5
    bgez $s1, Skip1
    li $s5, 0x80000000          # Storing the sign of the dividend
    neg $s1, $s1
Skip1:

# Getting the divisor
    li $v0, 4
    la $a0, msg2
    syscall
    li $v0, 5
    syscall
    move $s2, $v0
    xor $s6, $s6, $s6
    bgez $s2, Skip2
    li $s6, 0x80000000          # Storing the sign of the divisor
    neg $s2, $s2
Skip2:

# Performing signed division

    xor $s3, $s3, $s3          # Rem=0
    li $s4, 32                 # Loop counter
Loop:
    rol $t1, $s1, 1            # Shift (Remainder, Quotient) Left
    andi $t1, $t1, 1
    sll $s1, $s1, 1
    sll $s3, $s3, 1
    or $s3, $s3, $t1

    subu $t1, $s3, $s2          # Difference = Remainder – Divisor

```

```

    sgtu $t2, $t1, $s3          # Check if Difference < 0
    bnez $t2, Negative
    move $s3, $t1              # Remainder = Difference
    ori $s1, $s1, 0x0001      # Set least significant bit of Quotient
Negative:
    addi $s4, $s4, -1
    bnez $s4, Loop

# Setting the sign of the result
    bgez $s5, Skip3           # Setting remainder sign
    neg $s3, $s3

Skip3:
    xor $s6, $s6, $s5         # Setting quotient sign
    bgez $s6, Skip4
    neg $s1, $s1

Skip4:

# Displaying Result
    li $v0, 4
    la $a0, msg3
    syscall

    li $v0, 4                 # Display quotient
    la $a0, msg4
    syscall
    li $v0, 1
    move $a0, $s1
    syscall

    li $v0, 4                 # Display remainder
    la $a0, msg5
    syscall
    li $v0, 1
    move $a0, $s3
    syscall

    li $v0, 10                # Exit program
    syscall

```

Results of running the program on the given test cases:

Enter the dividend:**** user input : 17

Enter the divisor:**** user input : 3

Result of division: Quotient = 5 Remainder = 2

Enter the dividend:**** user input : 17
 Enter the divisor:**** user input : -3
 Result of division: Quotient = -5 Remainder = 2

Enter the dividend:**** user input : -17
 Enter the divisor:**** user input : 3
 Result of division: Quotient = -5 Remainder = -2

Enter the dividend:**** user input : -17
 Enter the divisor:**** user input : -3
 Result of division: Quotient = 5 Remainder = -2

Q.3. What is the decimal value of the following single-precision floating-point numbers?

(i) 0010 0000 0001 1100 0000 0000 0000 0000

$$= + (1.0011100\dots 0)_2 * 2^{(64-127)}$$

$$= + 1.21875 * 2^{-63}$$

(ii) 1100 1111 1110 1000 0000 0000 0000 0000

$$= - (1.110100\dots 0)_2 * 2^{(159-127)}$$

$$= - 1.8125 * 2^{32}$$

Q.4. Show the IEEE 754 binary representation for: -24.0625 in ...

(i) Single Precision

1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

(ii) Double precision

1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Q.5. Perform the following floating-point operations rounding the result to the nearest even. Perform the operation assuming both infinite precision and using only guard, round and sticky bits. Compare your solution in both cases.

(i) 0100 0011 1000 0000 0000 0000 0000 0000

-0100 0001 1000 0000 0000 0000 0000 1100

With Infinite Precision:

$$\begin{array}{r}
 1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
 - 1.000\ 0000\ 0000\ 0000\ 0000\ 1100 \\
 \hline
 = 1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
 - 0.000\ 1000\ 0000\ 0000\ 0000\ 0000\ 1100 \\
 \hline
 = 01.000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
 + 11.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 0100 \\
 \hline
 = 00.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 0100 \\
 = +0.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 0100 \\
 = +1.110\ 1111\ 1111\ 1111\ 1111\ 1110\ 1000
 \end{array}
 \begin{array}{l}
 \times 2^8 \\
 \times 2^4 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^7
 \end{array}$$

Then, we round to the nearest even and we do not add a 1 since the least significant bit is 0. Thus, the result will be:

$$+1.110\ 1111\ 1111\ 1111\ 1111\ 1110 \quad \times 2^7$$

With Guard, Round and Sticky Bits:

We add three bits for each operand representing G, R, S bits as follows.

$$\begin{array}{r}
 1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \\
 - 1.000\ 0000\ 0000\ 0000\ 0000\ 1100\ 000 \\
 \hline
 = 1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \\
 - 0.000\ 1000\ 0000\ 0000\ 0000\ 0000\ 110 \\
 \hline
 = 01.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \\
 + 11.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 010 \\
 \hline
 = 00.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 010 \\
 = +0.111\ 0111\ 1111\ 1111\ 1111\ 1111\ 010 \\
 = +1.110\ 1111\ 1111\ 1111\ 1111\ 1110\ 100
 \end{array}
 \begin{array}{l}
 \times 2^8 \\
 \times 2^4 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^8 \\
 \times 2^7
 \end{array}$$

Then, we round to the nearest even and we do not add a 1 since the least significant bit is 0. Thus, the result will be:

$$+1.110\ 1111\ 1111\ 1111\ 1111\ 1110 \quad \times 2^7$$

(ii) $0011\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000$
 $-0011\ 1111\ 1000\ 0100\ 0000\ 0000\ 0000\ 0000$

With Infinite Precision:

$$\begin{array}{r}
 1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
 - 1.000\ 0100\ 0000\ 0000\ 0000\ 0000 \\
 \hline
 = 01.000\ 0000\ 0000\ 0000\ 0000\ 0000
 \end{array}
 \begin{array}{l}
 \times 2^0 \\
 \times 2^0 \\
 \times 2^0
 \end{array}$$

$$\begin{array}{r}
+ \quad 10.111 \ 1100 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^0 \\
\hline
= \quad 11.111 \ 1100 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^0 \\
= \quad -0.000 \ 0100 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^0 \\
= \quad -1.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^{-5}
\end{array}$$

No rounding is necessary in this case.

The case using Guard, Round and Sticky bits will produce identical result since there is no right shifting.

$$\begin{array}{r}
\text{(iii)} \quad 0011 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1110 \\
+0011 \ 0100 \ 0100 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000
\end{array}$$

With Infinite Precision:

$$\begin{array}{r}
\quad \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1110 \quad \times 2^0 \\
+ \quad 1.100 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^{-23} \\
\hline
= \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1110 \quad \times 2^0 \\
+ \quad 0.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001 \ 1000 \ 0000\dots000 \times 2^0 \\
\hline
= \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1000 \ 0000\dots000 \times 2^0
\end{array}$$

Then, we round to the nearest even and we add a 1 since the least significant bit is 1. Thus, the result will be:

$$\begin{array}{r}
= \quad 10.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^0 \\
= \quad 1.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^1
\end{array}$$

With Guard, Round and Sticky Bits:

$$\begin{array}{r}
\quad \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1110 \ 000 \quad \times 2^0 \\
+ \quad 1.100 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 000 \quad \times 2^{-23} \\
\hline
= \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1110 \quad \times 2^0 \\
+ \quad 0.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001 \ 100 \quad \times 2^0 \\
\hline
= \quad 1.111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 100 \quad \times 2^0
\end{array}$$

Then, we round to the nearest even and we add a 1 since the least significant bit is 1. Thus, the result will be:

$$= \quad 10.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \quad \times 2^0$$

$$= 1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^1$$

Q.6. Given that $x = 0101\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000$
 $y = 0011\ 1111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000$
 $z = 1101\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000$

represent single precision IEEE 754 floating-point numbers. Perform the following operations showing all work:

(i) $x + y$

We are going to solve the question using guard, round and sticky bits.

$$\begin{array}{r}
 \\
 + \quad 1.011\ 1110\ 0100\ 0000\ 0000\ 0000\ 000 \\
 \hline
 = \quad 1.011\ 1110\ 0100\ 0000\ 0000\ 0000\ 000 \\
 + \quad 0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 001 \\
 \hline
 \end{array}
 \begin{array}{l}
 \times 2^{64} \\
 \times 2^0 \\
 \times 2^{64} \\
 \times 2^{64}
 \end{array}$$

$$= 1.011\ 1110\ 0100\ 0000\ 0000\ 0000\ 001 \times 2^{64}$$

Rounding the result to the nearest even gives the result:

$$= +1.011\ 1110\ 0100\ 0000\ 0000\ 0000 \times 2^{64}$$

(ii) Result of (i) + z

$$\begin{array}{r}
 \\
 - \quad 1.011\ 1110\ 0100\ 0000\ 0000\ 0000\ 000 \\
 \hline
 = \quad 01.011\ 1110\ 0100\ 0000\ 0000\ 0000\ 000 \\
 + \quad 10.100\ 0001\ 1100\ 0000\ 0000\ 0000\ 000 \\
 \hline
 \end{array}
 \begin{array}{l}
 \times 2^{64} \\
 \times 2^{64} \\
 \times 2^{64} \\
 \times 2^{64}
 \end{array}$$

$$\begin{array}{r}
 = \quad 00.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \\
 = \quad +0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000 \\
 \hline
 \end{array}
 \begin{array}{l}
 \times 2^{64} \\
 \times 2^{64}
 \end{array}$$

Rounding the result to the nearest even gives the result:

$$= +0.000\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{64}$$

Note that this number is equivalent to 0.

(iii) Why is the result of (ii) counterintuitive?

The result is counterintuitive because $z = -x$ and one expects that the result we will obtain will be equal to y . However, we got the result as 0. The main reason is that when we added x and y the result was x due to rounding. Thus, when we subtracted $z = -x$, we got 0.