**HW 9 (SOLUTION)**

1. (a) 1 1101 11101

Bias = 2^(4-1)-1 = 2^(3)-1 = 7

(-1)^1 x (1 + (1 x 2^-1) + (1 x 2^-2) + (1 x 2^-3) + (0 x 2^-4) + (1 x 2^-5)) x 2^(13-7)

-1 x (1 + .5 + .25 + .125 + 0 + .03125) x 2^6

-(1.90625) x 64

-122

-1.11101\*2^6

0 0110 11010

(-1)^0 x (1 + (1 x 2^-1) + (1 x 2^-2) + (0 x 2^-3) + (1 x 2^-4) + (0 x 2^-5)) x 2^(6-7)

(1 + .5 + .25 + 0 + .0625 + 0) x 2^-1

1 x 1.8125 x .5

.90625

0.11101

1.11010\*2^-1

1. (b) Lets us consider, we have 6-bit of precision, i.e. we can use 6-bit for the significand

Step 1: Shift the smaller number to the right until its exponent would match the larger exponent

1.11010x2^-1

0.11101x2^0

.

.

.

0.00000x2^6

Step 2: Add significands

-1.11101 +0.00000 = -1.11101 x 2^6

Step 3: Normalize the sum

-1.11101 x 2^6 (already normalized)

Step 4: Round

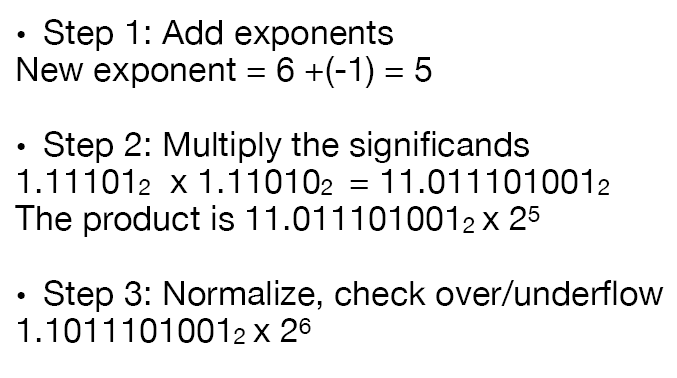
-1.11101 x 2^6 (already rounded)

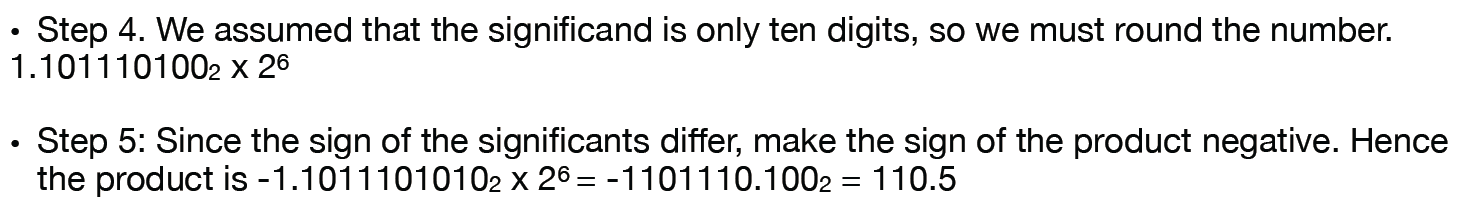
S=1; E=bias+true exponent=6+7=1101; F=11101

Final Result: 1 1101 11101

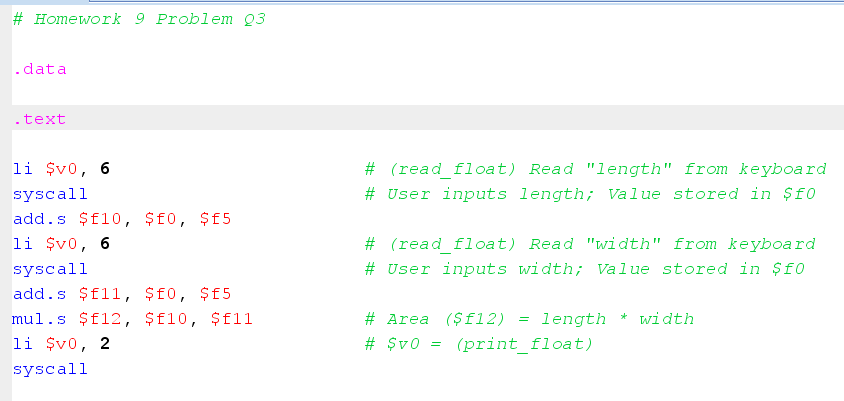
Comments: The adder circuit with 10-bit floating number system representation has a significant precision error. It completely ignores the effect of the smaller number.

**Q2.**





**Q3.**



**Q4. (a)**

# Matrix Addition and Multiplication

# the folowing program performs X=X+Y\*Z,

# Where X, Y, and Z are 8x8 Matrix and the

# elements are represented as double precision format

# Display the Results on the Mips Editor

.data

X: .double 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5

Y: .double 9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5

Z: .double 5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5

newLine: .asciiz "\n"

.text

la $a0, X #load the address of X

la $a1, Y #Load the address of Y

la $a2, Z #load the address of Z

li $t1, 8 # $t1 = 8 (row size/loop end)

li $s0, 0 # i = 0; initialize 1st for loop

L1: li $s1, 0 # j = 0; restart 2nd for loop

L2: li $s2, 0 # k = 0; restart 3rd for loop

sll $t2, $s0, 3 # $t2 = i \* 8 (size of row of x)

addu $t2, $t2, $s1 # $t2 = i \* size(row) + j

sll $t2, $t2, 3 # $t2 = byte offset of [i][j]

addu $t2, $a0, $t2 # $t2 = byte address of x[i][j]

l.d $f4, 0($t2) # $f4 = 8 bytes of x[i][j]

L3: sll $t0, $s2, 3 # $t0 = k \* 8 (size of row of z)

addu $t0, $t0, $s1 # $t0 = k \* size(row) + j

sll $t0, $t0, 3 # $t0 = byte offset of [k][j]

addu $t0, $a2, $t0 # $t0 = byte address of z[k][j]

l.d $f16, 0($t0) # $f16 = 8 bytes of z[k][j]

sll $t0, $s0, 3 # $t0 = i\*8 (size of row of y)

addu $t0, $t0, $s2 # $t0 = i\*size(row) + k

sll $t0, $t0, 3 # $t0 = byte offset of [i][k]

addu $t0, $a1, $t0 # $t0 = byte address of y[i][k]

l.d $f18, 0($t0) # $f18 = 8 bytes of y[i][k]

mul.d $f16, $f18, $f16 # $f16 = y[i][k] \* z[k][j]

add.d $f4, $f4, $f16 # f4=x[i][j] + y[i][k]\*z[k][j]

addiu $s2, $s2, 1 # $k k + 1

bne $s2, $t1, L3 # if (k != 8) go to L3

s.d $f4, 0($t2) # x[i][j] = $f4

mov.d $f12,$f4 #move the $f4 into $f12 for-

# -printing result with syscall

jal print #Call Print Function

addiu $s1, $s1, 1 # $j = j + 1

bne $s1, $t1, L2 # if (j != 8) go to L2

jal next\_row # Print Next\_row

addiu $s0, $s0, 1 # $i = i + 1

bne $s0, $t1, L1 # if (i != 8) go to L1

j Exit

print:

addi $sp, $sp, -8 # reserve space in the stack to store $a0,$ra

sw $ra, 0($sp) # save $ra into the stack

sw $a0, 4($sp) # Save $a0 into the stack

li $v0, 3 # print\_double

syscall # print result

# print space, 32 is ASCII code for space

li $a0, 32

li $v0, 11 # syscall number for printing blank space

syscall

syscall

lw $a0, 4($sp) #restore $a0 for the caller

lw $ra, 0($sp) #restore $ra for the caller

addi $sp, $sp, 8 #free-up stack space

jr $ra #jump back to the calling program

next\_row:

addi $sp, $sp, -8 # reserve space in the stack to store $a0, $ra

sw $ra, 0($sp) # save $ra into the stack

sw $a0, 4($sp) # Save $a0 into the stack

la $a0, newLine #get NewLine Command Charater

addi $v0, $0, 4 #Newline Function parameter

syscall

lw $a0, 4($sp) #restore $a0 for the caller

lw $ra, 0($sp) #restore $ra for the caller

addi $sp, $sp, 8 #free-up stack space

jr $ra

Exit:

nop

Q4. (b) ----Next Page

**Q4. (b):**

# Matrix Addition and Multiplication

# the folowing program performs Z=X\*Y-Z,

# Where X, Y, and Z are 8x8 Matrix and the

# elements are represented as double precision format

# Display the Results on the Mips Editor

.data

X: .double 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,

5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5

Y: .double 9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5

Z: .double 5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,

5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,

9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5

Zr: .double 0.0

newLine: .asciiz "\n"

.text

la $a0, X #load the address of X

la $a1, Y #Load the address of Y

la $a2, Z #load the address of Z

# Initialize $f20 = 0.0;

la $t7,Zr

l.d $f20, 0($t7)

#######################(Start Looping)###################

li $t1, 8 # $t1 = 8 (row size/loop end)

li $s0, 0 # i = 0; initialize 1st for loop

L1: li $s1, 0 # j = 0; restart 2nd for loop

L2: li $s2, 0 # k = 0; restart 3rd for loop

sll $t2, $s0, 3 # $t2 = i \* 8 (size of row of Z)

addu $t2, $t2, $s1 # $t2 = i \* size(row) + j

sll $t2, $t2, 3 # $t2 = byte offset of [i][j]

addu $t2, $a2, $t2 # $t2 = byte address of Z[i][j]

l.d $f4, 0($t2) # $f4 = 8 bytes of Z[i][j]

L3: sll $t0, $s0, 3 # $t0 = i \* 8 (size of row of X)

addu $t0, $t0, $s2 # $t0 = i \* size(row) + k

sll $t0, $t0, 3 # $t0 = byte offset of [i][k]

addu $t0, $a0, $t0 # $t0 = byte address of X[i][k]

l.d $f16, 0($t0) # $f16 = 8 bytes of X[i][k]

sll $t0, $s2, 3 # $t0 = k\*8 (size of row of Y)

addu $t0, $t0, $s1 # $t0 = k\*size(row) + j

sll $t0, $t0, 3 # $t0 = byte offset of [k][j]

addu $t0, $a1, $t0 # $t0 = byte address of Y[k][j]

l.d $f18, 0($t0) # $f18 = 8 bytes of Y[k][j]

mul.d $f16, $f16, $f18 # $f16 = X[i][k] \* Y[k][j]

add.d $f20, $f16, $f20 # f20=x[i][k]\*y[k][j]

addiu $s2, $s2, 1 # $k k + 1

bne $s2, $t1, L3 # if (k != 8) go to L3

sub.d $f4, $f20,$f4 # Z=X\*Y-Z ($f4 = x[i][k]\*y[k][j]-Z[i][j]

l.d $f20,0($t7) #Initialize $f20 to zero

s.d $f4, 0($t2) # Z[i][j] = $f4

mov.d $f12,$f4 #move the $f4 into $f12 for-

# -printing result with syscall

jal print #Call Print Function

addiu $s1, $s1, 1 # $j = j + 1

bne $s1, $t1, L2 # if (j != 8) go to L2

jal next\_row # Print Next\_row

addiu $s0, $s0, 1 # $i = i + 1

bne $s0, $t1, L1 # if (i != 8) go to L1

j Exit

print:

addi $sp, $sp, -8 # reserve space in the stack to store $a0,$ra

sw $ra, 0($sp) # save $ra into the stack

sw $a0, 4($sp) # Save $a0 into the stack

li $v0, 3 # print\_double

syscall # print result

# print space, 32 is ASCII code for space

li $a0, 32

li $v0, 11 # syscall number for printing blank space

syscall

syscall

lw $a0, 4($sp) #restore $a0 for the caller

lw $ra, 0($sp) #restore $ra for the caller

addi $sp, $sp, 8 #free-up stack space

jr $ra #jump back to the calling program

next\_row:

addi $sp, $sp, -8 # reserve space in the stack to store $a0, $ra

sw $ra, 0($sp) # save $ra into the stack

sw $a0, 4($sp) # Save $a0 into the stack

la $a0, newLine #get NewLine Command Charater

addi $v0, $0, 4 #Newline Function parameter

syscall

lw $a0, 4($sp) #restore $a0 for the caller

lw $ra, 0($sp) #restore $ra for the caller

addi $sp, $sp, 8 #free-up stack space

jr $ra

Exit:

nop