

## HW#5 (CSC390); Turn in your code on the Blackboard by 03/25/2016 by 5:00PM

**Q1.** Write a MIPS assembly program to calculate the area of a rectangle using the floating point data (length and width of the rectangle) supplied from the keyboard. Your program should also display the results on the MIPS editor.

Hints: use “syscall” function discussed in our last class and example codes posted on the blackboard.

**Q2.** The following program (as shown in fig. 4. The code is also posted on the blackboard) performs the matrix operation,  $X=X+Y*Z$ , and display the results on the MIPS editor as shown in Figure 5. Please note that X, Y, and Z are square matrixes of size **4x4** and the elements of the matrixes are double precision floating point number.

(a) Now, modify the program such that it would perform:

$X = X+Y*Z$ ; where X, Y, and Z are **8x8 matrixes** (as shown in fig 1) and each element is represented by double precision number. Your program should also display the result as shown in figure 2.

(Note that results are verified using Matlab)

```
7  .data
8  X: .double 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,
9           5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,
10          9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,
11          5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5,
12          1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,
13          5.5, 6.5, 7.5, 8.5, 9.5, 2.5, 3.5, 4.5,
14          9.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5,
15          5.5, 6.5, 7.5, 8.5, 1.5, 2.5, 3.5, 4.5
16
17  Y: .double 9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,
18          5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
19          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
20          5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
21          9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,
22          5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
23          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
24          5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5
25
26  Z: .double 5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
27          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
28          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
29          9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,
30          5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
31          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
32          1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
33          9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5
34  ..
```

Figure 1

Mars Messages	Run I/O
	209.5 178.5 219.5 260.5 213.5 182.5 223.5 264.5
	269.5 190.5 247.5 304.5 273.5 186.5 243.5 300.5
	129.5 74.5 99.5 124.5 125.5 78.5 103.5 128.5
	269.5 190.5 247.5 304.5 265.5 186.5 243.5 300.5
	209.5 178.5 219.5 260.5 213.5 182.5 223.5 264.5
Clear	269.5 190.5 247.5 304.5 273.5 186.5 243.5 300.5
	129.5 74.5 99.5 124.5 125.5 78.5 103.5 128.5
	269.5 190.5 247.5 304.5 265.5 186.5 243.5 300.5
	-- program is finished running (dropped off bottom) --

Figure 2

(b) Also, modify the program such that it would perform:

$Z = X*Y-Z$ ; where X, Y, and Z are 8x8 matrixes (as shown in fig 1) and each element is represented by double precision number. Your program should also display the result as shown in figure 3.

(Note that this part would be more challenging than the first part. Please go through the pages 215-217 of your book. Pay attention to the Indexed operation and how the multiplication is done. You may need to use an extra instruction sub.d in addition to the add.d instruction. Add.d would be used to find each element of  $X*Y$  and sub.d would be used for  $X*Y-Z$ . Recall the Matrix multiplication operation. You may also need to use an extra floating point register to hold the  $X*Y$  value temporarily)

Mars Messages	Run I/O
	198.5 181.5 220.5 259.5 198.5 181.5 220.5 259.5
	278.5 205.5 252.5 299.5 278.5 205.5 252.5 299.5
	278.5 205.5 252.5 299.5 278.5 205.5 252.5 299.5
	194.5 185.5 224.5 263.5 194.5 185.5 224.5 263.5
	198.5 181.5 220.5 259.5 198.5 181.5 220.5 259.5
Clear	278.5 205.5 252.5 299.5 278.5 205.5 252.5 299.5
	278.5 205.5 252.5 299.5 278.5 205.5 252.5 299.5
	194.5 185.5 224.5 263.5 194.5 185.5 224.5 263.5
	-- program is finished running (dropped off bottom) --

Figure 3

## Matrix\_Operation

```

1  # Matrix Addition and Multiplication
2  # the folowing program performs X=X+Y*Z,
3  # Where X, Y, and Z are 4x4 Matrix and the
4  # elements are represented as double precision format
5  # Display the Results on the Mips Editor
6
7  .data
8  X: .double 1.5, 2.5, 3.5, 4.5,
9           5.5, 6.5, 7.5, 8.5,
10          9.5, 2.5, 3.5, 4.5,
11          5.5, 6.5, 7.5, 8.5
12
13  Y: .double 9.5, 2.5, 3.5, 4.5,
14          5.5, 6.5, 7.5, 8.5,
15          1.5, 2.5, 3.5, 4.5,
16          5.5, 6.5, 7.5, 8.5
17
18  Z: .double 5.5, 6.5, 7.5, 8.5,
19          1.5, 2.5, 3.5, 4.5,
20          1.5, 2.5, 3.5, 4.5,
21          9.5, 2.5, 3.5, 4.5
22
23  newLine: .asciiz "\n"
24
25  .text
26  la $a0, X      #load the address of X
27  la $a1, Y      #Load the address of Y
28  la $a2, Z      #load the address of Z
29  li  $t1, 4      # $t1 = 4 (row size/loop end)
30  li  $s0, 0      # i = 0; initialize 1st for loop
31  L1: li  $s1, 0   # j = 0; restart 2nd for loop
32  L2: li  $s2, 0   # k = 0; restart 3rd for loop
33  sll  $t2, $s0, 2 # $t2 = i * 4 (size of row of x)
34  addu $t2, $t2, $s1 # $t2 = i * size(row) + j
35  sll  $t2, $t2, 3 # $t2 = byte offset of [i][j]
36  addu $t2, $a0, $t2 # $t2 = byte address of x[i][j]

```

```

37  l.d  $f4, 0($t2)    # $f4 = 8 bytes of x[i][j]
38  L3: sll  $t0, $s2, 2    # $t0 = k * 4 (size of row of z)
39  addu $t0, $t0, $s1 # $t0 = k * size(row) + j
40  sll  $t0, $t0, 3    # $t0 = byte offset of [k][j]
41  addu $t0, $a2, $t0 # $t0 = byte address of z[k][j]
42  l.d  $f16, 0($t0)   # $f16 = 8 bytes of z[k][j]
43  sll  $t0, $s0, 2      # $t0 = i*4 (size of row of y)
44  addu $t0, $t0, $s2    # $t0 = i*size(row) + k
45  sll  $t0, $t0, 3      # $t0 = byte offset of [i][k]
46  addu $t0, $a1, $t0    # $t0 = byte address of y[i][k]
47  l.d  $f18, 0($t0)   # $f18 = 8 bytes of y[i][k]
48  mul.d $f16, $f18, $f16 # $f16 = y[i][k] * z[k][j]
49  add.d $f4, $f4, $f16 # f4=x[i][j] + y[i][k]*z[k][j]
50  addiu $s2, $s2, 1     # $k k + 1
51  bne  $s2, $t1, L3     # if (k != 4) go to L3
52  s.d  $f4, 0($t2)     # x[i][j] = $f4
53  mov.d $f12, $f4      #move the $f4 into $f12 for-
54                          # -printing result with syscall
55  jal  print            #Call Print Function
56
57  addiu $s1, $s1, 1     # $j = j + 1
58  bne  $s1, $t1, L2     # if (j != 4) go to L2
59
60  jal  next_row         # Print Next_row
61
62  addiu $s0, $s0, 1     # $i = i + 1
63  bne  $s0, $t1, L1     # if (i != 4) go to L1

65  j  Exit
66  print:
67  addi $sp, $sp, -8    # reserve space in the stack to store $a0,$ra
68  sw   $ra, 0($sp)    # save $ra into the stack
69  sw   $a0, 4($sp)    # Save $a0 into the stack
70  li   $v0, 3         # print_double
71  syscall              # print result
72  # print space, 32 is ASCII code for space
73  li   $a0, 32
74  li   $v0, 11        # syscall number for printing blank space
75  syscall
76  syscall

```

```

77 lw    $a0, 4($sp)  #restore $a0 for the caller
78 lw    $ra, 0($sp)  #restore $ra for the caller
79 addi   $sp, $sp, 8  #free-up stack space
80 jr     $ra         #jump back to the calling program
81
82 next_row:
83 addi   $sp, $sp, -8  # reserve space in the stack to store $a0, $ra
84 sw     $ra, 0($sp)  # save $ra into the stack
85 sw     $a0, 4($sp)  # Save $a0 into the stack
86 la     $a0, newLine #get NewLine Command Charater
87 addi   $v0, $0, 4    #Newline Function parameter
88 syscall
89 lw     $a0, 4($sp)  #restore $a0 for the caller
90 lw     $ra, 0($sp)  #restore $ra for the caller
91 addi   $sp, $sp, 8  #free-up stack space
92 jr     $ra
93
94 Exit:
95 nop

```

Figure 4

Results:

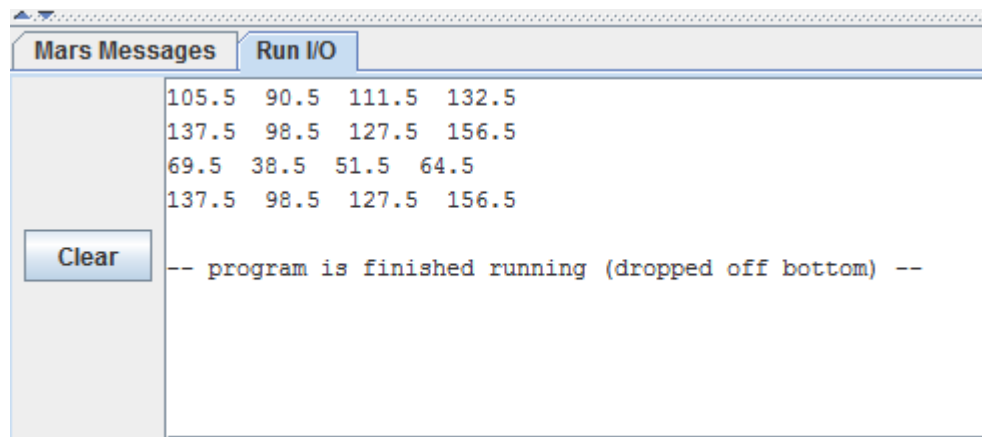


Figure 5