ASHWIN ANAND - COMPUTER ARCHITECTURE LAB 5

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Course Name: Computer Architecture Lab

Course Number and Section: 14:332:333:01

Experiment: Lab # 5 – RISC-V functions and arrays

Lab Instructor: Mingbo Zhang

Date Performed: April 21st, 2021

Date Submitted: April 25th, 2021

Submitted by: Ashwin Anand - 192007894

ASHWIN ANAND - COMPUTER ARCHITECTURE LAB 5

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Computer Architecture and Assembly
Lab Spring 2021

Lab 5
RISC-V functions and arrays

Instructions

Please answer all the questions below. You need to use the Venus RISC-V simulator for running and testing your code. Note that the simulator is 32-bit, and we do not consider overflow here.
Upload your lab report with the department cover page and your source code using Sakai.

Exercises

1. [30 pts] Write a RISC-V program in Venus simulator that accepts an input integer x and uses two methods to compute a factorial:

- Recursive method: $f(x) = x * f(x - 1)$
- Iterative/loop method: $f(x) = x! = x * (x - 1) * (x - 2) * ... * 2 * 1$

You can assume x is always a positive number.

In the program, please have a **main** function that takes the value of the input, performs the factorial computations by the two methods, and prints the outputs of the two methods in the console.

- Verify your program for 3 different input values x_1, x_2, and x_3 such that $x_1 \neq x_2 \neq x_3 \neq 0$.

Note: you need to provide your own inputs and show screenshots of the outputs based on the given inputs. (Each method and function worth 10 pts.)

Code:

```
main:
addi x28, x28, 1
addi x29, x0, 2
beq x28, x29, printer
addi x18, x0, 1
#input
addi x16, x0, 8

#recursion method
recursion:
addi x2, x2, -16
sw x1, 0(x2)
sw x16, 8(x2)
bge x18,x16, Exit
addi x16, x16, -1
jal x1, recursion
add x18, x16, x0
lw x1, 0(x2)
lw x16, 8(x2)
addi x2, x2, 16
mul x16, x16, x18
jalr x0, 0(x1)

Exit:
jalr x0, 0(x1)
```

```
printer:



addi x10 x0 4
la x11 msg1
ecall

#print recursion output
add x11, x0, x16
addi x10, x0, 1
ecall
addi x11, x0, 32
addi x10, x0, 11
ecall
#input
addi x16, x0, 8

loop:
addi x18, x0, 0
addi x18, x18, 1
add x7, x16, x7

#iterative method
Iteration:
bge x18, x7, finishprint
addi x7, x7, -1
mul x16, x16, x7
jal x0, Iteration
```

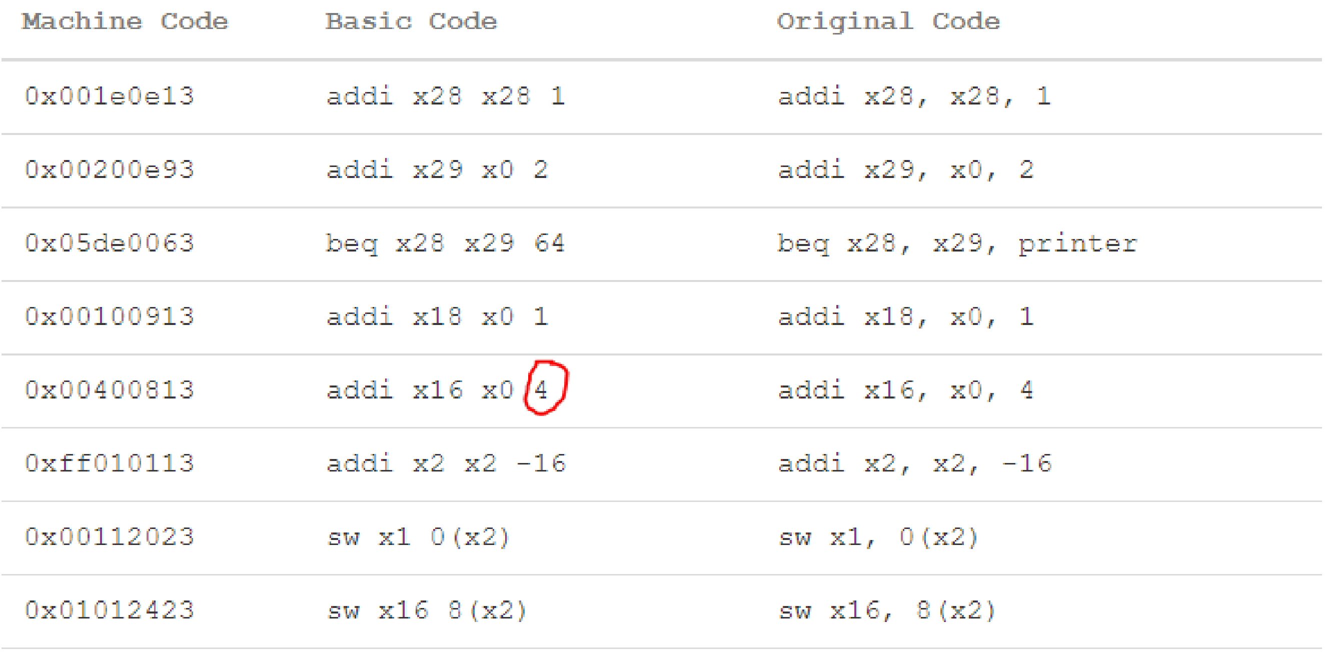
```
ve output
finishprint:
addi x10 x0 4



la x11 nline
ecall


#label print
addi x10 x0 4
la x11 msg2
ecall


add x11, x0, x16
addi x10, x0, 1
ecall




.data
nline:  .asciiz "\n"
msg1:         .asciiz "Recursive Output: "
msg2:         .asciiz "Iterative Output: "
```

Test Cases:

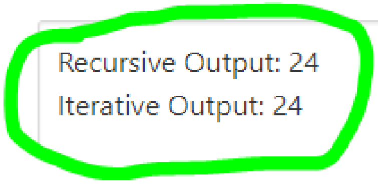
Test Case 1:

Input (red circle): 4

Output (green circle): recursive output = 24 iterative output = 24



Machine Code	Basic Code	Original Code
0x001e0e13	addi x28 x28 1	addi x28, x28, 1
0x00200e93	addi x29 x0 2	addi x29, x0, 2
0x05de0063	beq x28 x29 64	beq x28, x29, printer
0x00100913	addi x18 x0 1	addi x18, x0, 1
0x00400813	addi x16 x0 4	addi x16, x0, 4
0xff010113	addi x2 x2 -16	addi x2, x2, -16
0x00112023	sw x1 0(x2)	sw x1, 0(x2)
0x01012423	sw x16 8(x2)	sw x16, 8(x2)

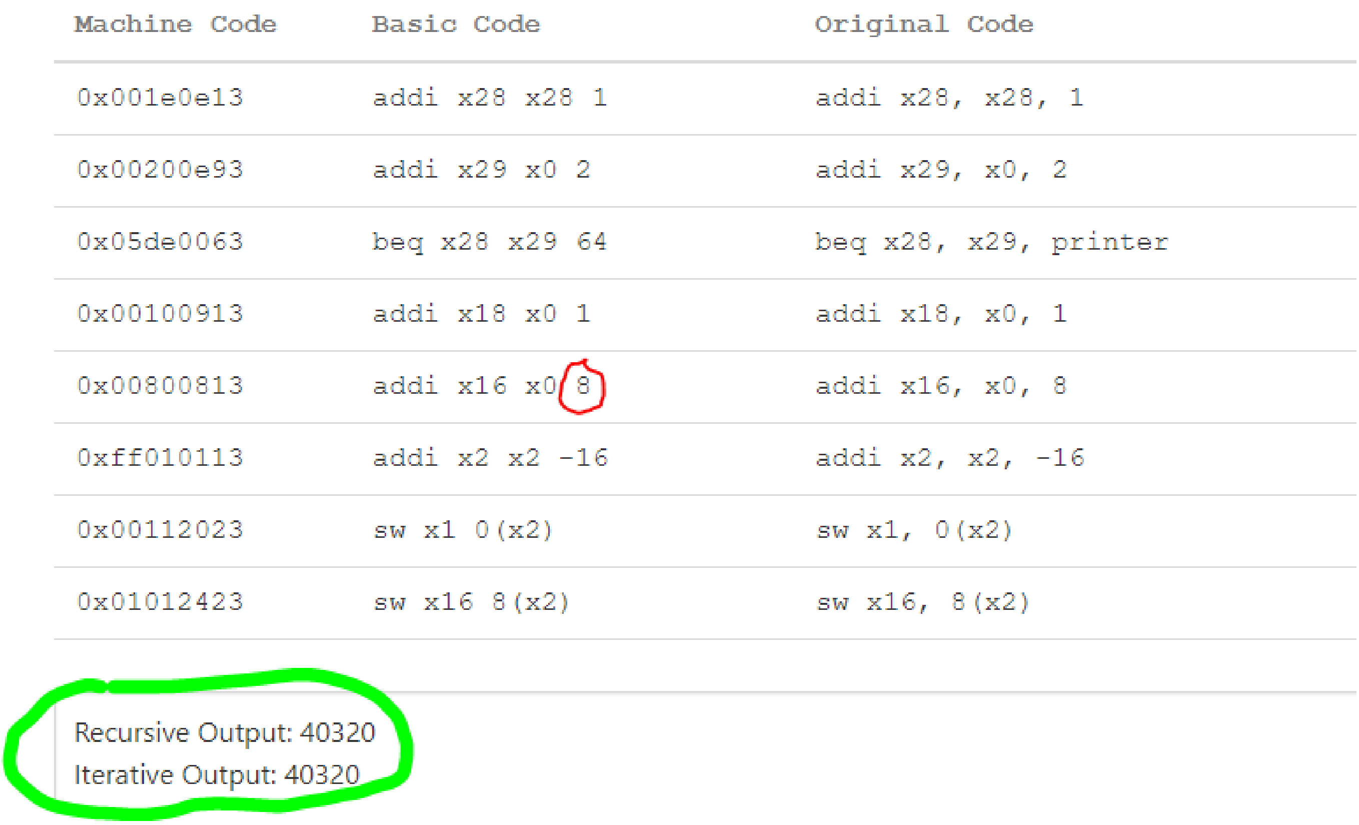


```
Recursive Output: 24
Iterative Output: 24
```

Test Case 2:

Input (red circle): 8

Output (green circle): recursive output = 40320 iterative output = 40320



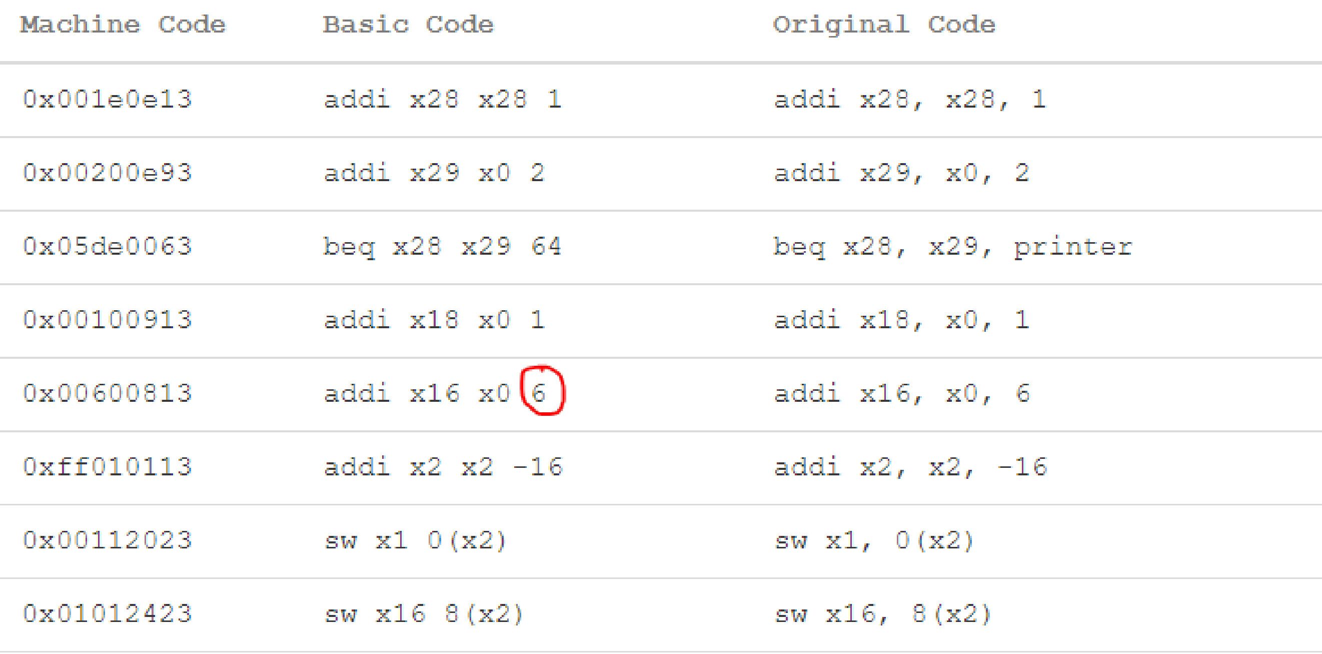
Machine Code	Basic Code	Original Code
0x001e0e13	addi x28 x28 1	addi x28, x28, 1
0x00200e93	addi x29 x0 2	addi x29, x0, 2
0x05de0063	beq x28 x29 64	beq x28, x29, printer
0x00100913	addi x18 x0 1	addi x18, x0, 1
0x00800813	addi x16 x0 8	addi x16, x0, 8
0xff010113	addi x2 x2 -16	addi x2, x2, -16
0x00112023	sw x1 0(x2)	sw x1, 0(x2)
0x01012423	sw x16 8(x2)	sw x16, 8(x2)

```
Recursive Output: 40320
Iterative Output: 40320
```

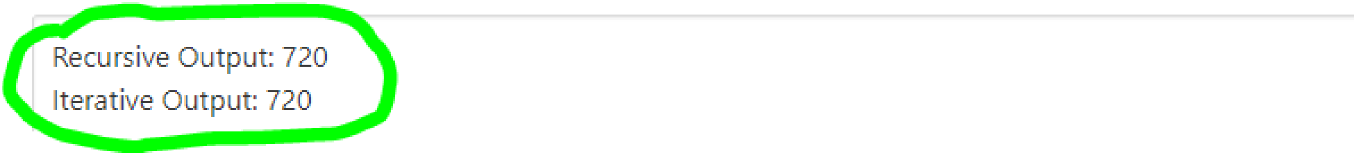
Test Case 3:

Input (red circle): 6

Output (green circle): recursive output = 720 iterative output = 720



Machine Code	Basic Code	Original Code
0x001e0e13	addi x28 x28 1	addi x28, x28, 1
0x00200e93	addi x29 x0 2	addi x29, x0, 2
0x05de0063	beq x28 x29 64	beq x28, x29, printer
0x00100913	addi x18 x0 1	addi x18, x0, 1
0x00600813	addi x16 x0 6	addi x16, x0, 6
0xff010113	addi x2 x2 -16	addi x2, x2, -16
0x00112023	sw x1 0(x2)	sw x1, 0(x2)
0x01012423	sw x16 8(x2)	sw x16, 8(x2)



```
Recursive Output: 720
Iterative Output: 720
```

2. [30 pts] Write a RISC-V program in Venus simulator that splits the given array {4, 37, 0, 12, 1, 0, 6} into the following three sub-arrays:

- Array 1: the elements are the odd positive numbers
- Array 2: the elements are the even positive numbers
- Array 3: the elements are all zeros

In the program, please have a **main** function that obtains the value of the input, splits the array, and prints the three sub-arrays in the console. Your implementation should work for arrays of different values as well.

Note: you need to show screenshots of the outputs based on the given input. (Each array worth 10 pts.)

Code:

```
main:

.data #static data section

array:      .word 4 37 0 12 1 0 6

oddarr:                 .word 0 0 0 0 0 0 0

evenarr:    .word 0 0 0 0 0 0 0

zeros:                  .word 0 0 0 0 0 0 0

size:       .word 7

numodd:     .asciiz "positive odd numbers: "

numeven:    .asciiz "positive even numbers: "

zeromsg:    .asciiz "Zeros: "

.text


#load initial values

la x17, array

la x18, oddarr
la x19, evenarr
la x21, zeros
lw x14, size


addi x26, x26, 2
addi x28, x28, 1


#store in the odd, even and zero arrays
checkandsort:
bge x16, x14, EXIT
addi x16, x16, 1
lw x27, 0(x17)
rem x25, x27, x26
addi x17, x17, 4
bge x25, x28, sortOdd
div x29, x27, x26
blt x29, x28, sortZero
sw x27, 0(x19)
addi x19, x19, 4
addi x5, x5, 1
jal x0, checkandsort


#store in the zero array
sortZero:
sw x27, 0(x21)
addi x21, x21, 4
addi x6, x6, 1
jal x0, checkandsort

#store in the odd array
sortOdd:
sw x27, 0(x18)
addi x18, x18, 4
addi x8, x8, 1
jal x0, checkandsort

EXIT:
addi x27, x0, 0
addi x28, x0, 0
addi x10, x0, 1
la x18, oddarr
la x19, evenarr
la x21, zeros

#print odd text message
printOdd:
addi x10 x0 1
addi x10 x0 4
la x11 numodd
ecall

#loop thru odd array to print
```

```
printOddstart:
bge x24, x8, loop1

addi x10, x0, 1
lw x11, 0(x18)
addi x18, x18, 4
ecall
addi x11, x0, 32
addi x10, x0, 11
ecall
addi x24, x24, 1
jal x0, printOddstart

loop1:
addi x11, x0, 10
addi x10, x0, 11
ecall
addi x27, x0, 0

#print odd text message
printEven:
addi x10 x0 4
la x11 numeven
ecall

#loop thru even array to print
```

```
printEvenstart:
bge x27, x5, loop2

addi x10, x0, 1
lw x11, 0(x19)
addi x19, x19, 4
ecall
addi x11, x0, 32
addi x10, x0, 11
ecall
addi x27, x27, 1
jal x0, printEvenstart

loop2:
addi x11, x0, 10
addi x10, x0, 11
ecall

#print zero text message
printZeros:
addi x10 x0 4
la x11 zeromsg
ecall

#loop thru zero array to print
printZerosstart:
```

```
bge x28, x6, DONE


addi x10, x0, 1
lw x11, 0(x21)
addi x21, x21, 4
ecall
addi x11, x0, 32
addi x10, x0, 11
ecall
addi x28, x28, 1
jal x0, printZerosstart

DONE:
```

Test Cases:

Test Case 1:

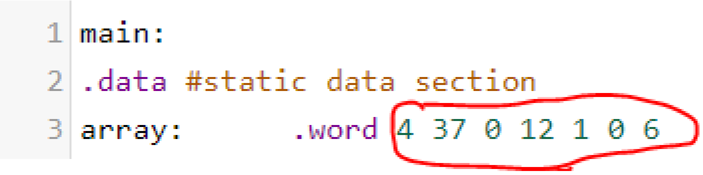
Input (red circle): {4,37,0,12,1,0,6}

Output (green circle):

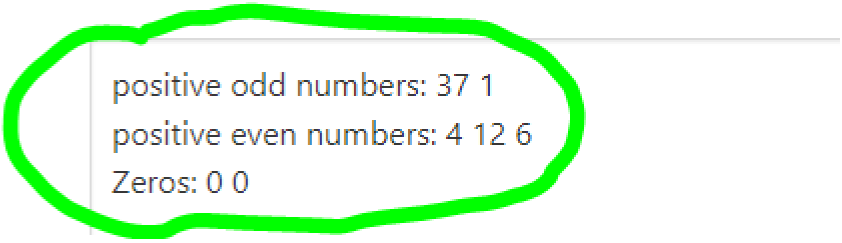
positive odd numbers: 37 1

positive even numbers: 4 12 6

Zeros: 0 0



```
main:
.data #static data section
array:      .word 4 37 0 12 1 0 6
```



```
positive odd numbers: 37 1
positive even numbers: 4 12 6
Zeros: 0 0
```

Test Case 2:

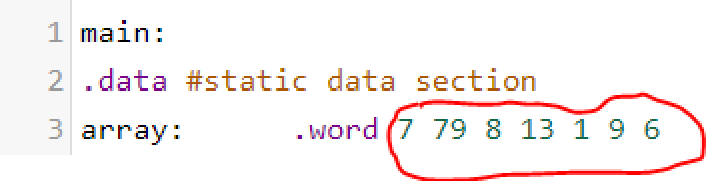
Input (red circle):{7,79,8,13,1,9,6}

Output (green circle):

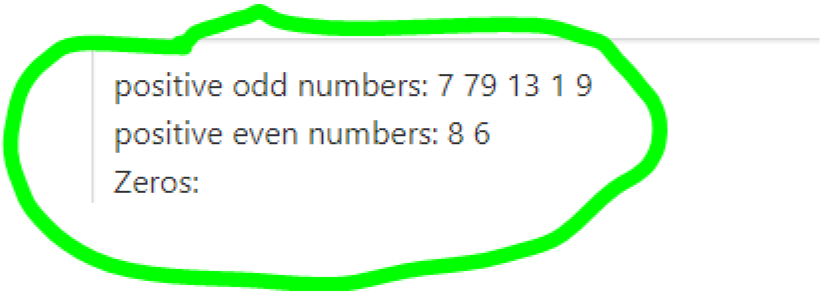
positive odd numbers: 7 79 13 1 9

positive even numbers: 8 6

Zeros:



```
main:
.data #static data section
array:      .word 7 79 8 13 1 9 6
```



```
positive odd numbers: 7 79 13 1 9
positive even numbers: 8 6
Zeros:
```

Test Case 3:

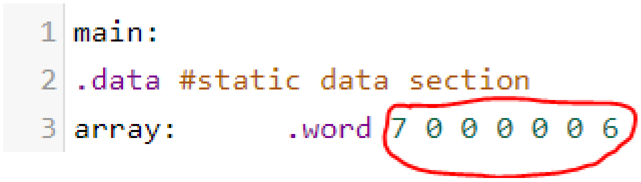
Input (red circle): {7,0,0,0,0,0,6}

Output (green circle):

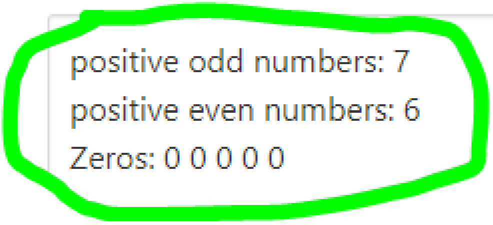
positive odd numbers: 7

positive even numbers: 6

Zeros: 0 0 0 0 0



```
main:
.data #static data section
array:      .word 7 0 0 0 0 0 6
```



```
positive odd numbers: 7
positive even numbers: 6
Zeros: 0 0 0 0 0
```

3. [40 pts] Write a RISC-V program in Venus simulator that computes matrix multiplication. You can assume all the entries in each matrix are positive numbers. In the program, you need to consider the following cases:

- When the number of rows for matrix A and the number of columns for matrix B are equal, compute the multiplication and obtain the product matrix C: $C = A * B$.
- When the number of rows for matrix A and the number of columns for matrix are not equal, return the error code 99. Hint: please refer to the Ecall wiki page about returning the error code (https://github.com/kvakil/venus/wiki/Environmental-Calls).

In the program, please have a **main** function that checks the condition, perform the appropriate computations, and print the matrix C or the error code in the console.

- Verify your program for 3 input matrices $m_1, m_2,$ and m_3 such that the dimensions of $m_1, m_2,$ and m_3 are all unique.
- Demonstrate each possible case at least once.
- Your implementation should work for matrices of different values as well.

Note: you need to provide your own inputs and show screenshots of the outputs based on the given inputs. (The **main** function worth 15 pts, and the multiplication function worth 25 pts.)

Code:

```
# setup data

# update row1pos, row2pos, col1pos, col2pos based on setup input array matrix

.data

array1:     .word 2 0

            .word 3 4

row1pos:    .word 2

col1pos:    .word 2
```

```
array2:         .word 2 4 3

                .word 1 2 0

row2pos:        .word 2

col2pos:        .word 3

CMatrix:        .word 0

        .word 0

.text


main:

# load input array row and col

la x17, array1

lw x21, row1pos

lw x19, col1pos

la x29, CMatrix

la x22, array2

lw x24, row2pos

lw x20, col2pos

addi x5, x5, 4


#matrix 2 col size

mul x31, x5, x20
```

```
size



mul x13, x5, x19

mul x23, x20, x5

add x23, x22, x23

mul x18, x19, x5

add x18, x17, x19

beq x19, x24, traverseandmultiply

addi x10, x0, 17


#set error code

addi x11, x0, 99

ecall


#traversing col and row for matrix1 and matrix2

traverseandmultiply:

beq x6, x19, outoflayer

lw x26, 0(x17)

lw x25, 0(x22)

mul x27, x26, x25

add x28, x28, x27

addi x17, x17, 4

add x22, x22, x31
```

```
a

jal x0, traverseandmultiply



#going thru multiple layers and loop

outoflayer:

addi x6, x0, 0

sw x28, 0(x29)

addi x29, x29, 4

addi x28, x0, 0

addi x12, x12, 1

addi x8, x8, 1

mul x9, x21, x20

beq x12, x20, next

la x17, array1

la x22, array2

mul x30, x12, x5

add x22, x30, x22

mul x30, x11, x13

add x17, x30, x17

jal x0, traverseandmultiply


next:

beq x11, x21, done
```

```
, 1
la x17, array1
la x22, array2



mul x30, x13, x11
add x17, x17, x30
addi x12, x0, 0
jal x0, traverseandmultiply

#loading final matrix
done:
la x29, CMatrix
addi x6, x0, 0

#print out the matrix 3 values
doprint:
bge x6, x9, exit
addi x10, x0, 1
lw x11, 0(x29)
addi x29, x29, 4
ecall
addi x11, x0, 32
addi x10, x0, 11
```

addi x6, x6, 1

jal x1, doprint

exit:

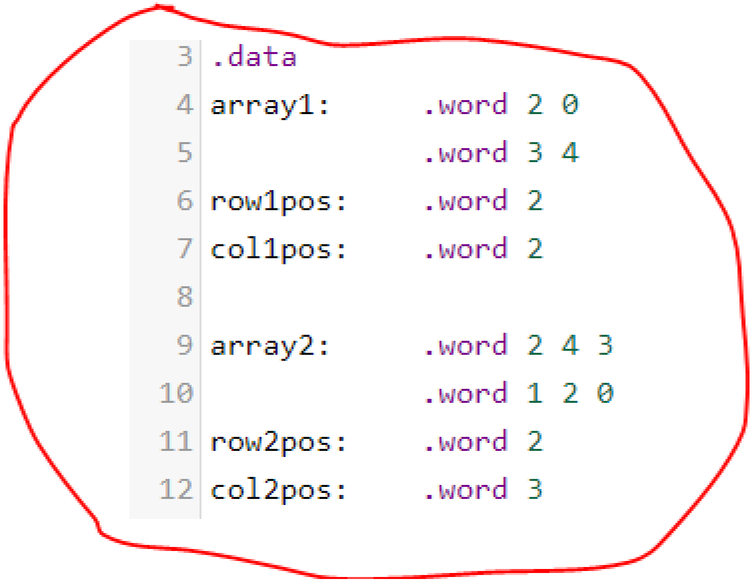
Test Cases:

Test Case 1:

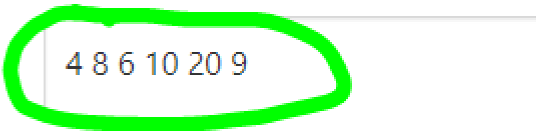
Input (red circle): Matrix 1: 2 0 Matrix 2: 2 4 3

3 4 1 2 0

Output (green circle):



```
.data
array1:      .word 2 0
             .word 3 4
row1pos:     .word 2
col1pos:     .word 2

array2:      .word 2 4 3
             .word 1 2 0
row2pos:     .word 2
col2pos:     .word 3
```



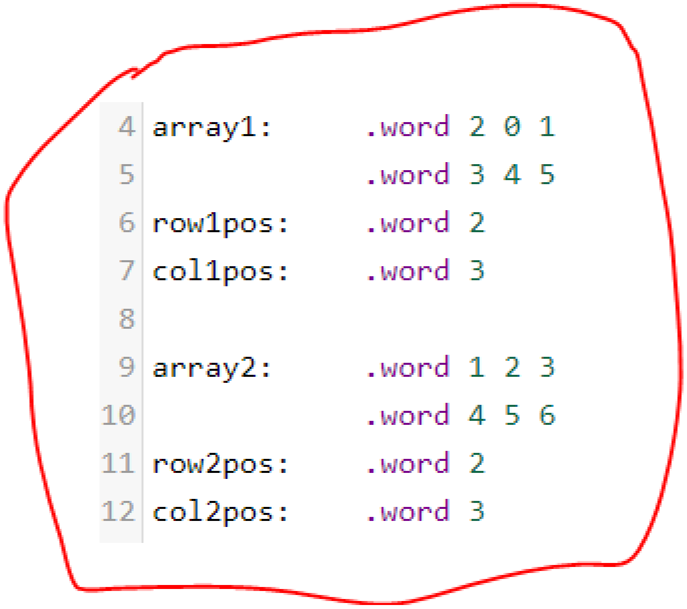
```
4 8 6 10 20 9
```

4	8	6
10	20	9

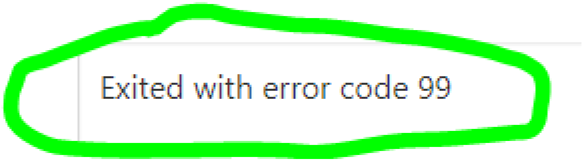
Test Case 2:

Input (red circle): Matrix 1: 2 0 1 Matrix 2: 1 2 3
3 4 5 4 5 6

Output (green circle):



```
array1:      .word 2 0 1
             .word 3 4 5
row1pos:     .word 2
col1pos:     .word 3

array2:      .word 1 2 3
             .word 4 5 6
row2pos:     .word 2
col2pos:     .word 3
```

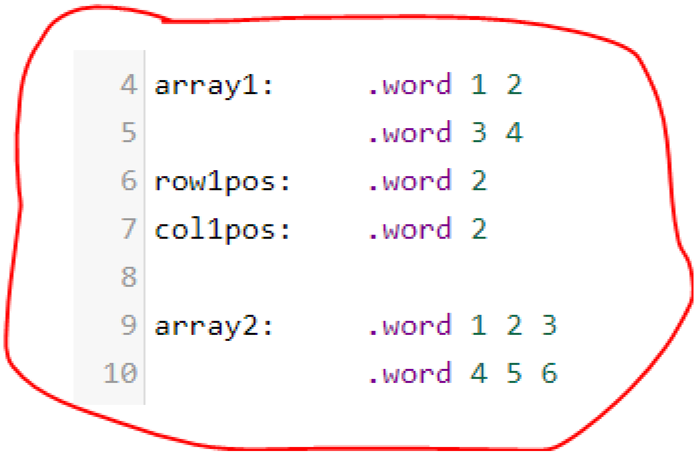


```
Exited with error code 99
```

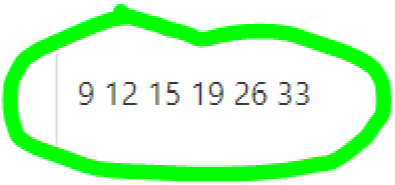
Test Case 3:

Input (red circle): Matrix 1: 1 2 Matrix 2: 1 2 3
3 4 4 5 6

Output (green circle):



```
array1:      .word 1 2
             .word 3 4
row1pos:     .word 2
col1pos:     .word 2

array2:      .word 1 2 3
             .word 4 5 6
```



```
9 12 15 19 26 33
```

9	12	15
19	26	33