Array in MIPS

```
.data
        myArray: .space 12              #12 bytes for 3 integers
.text
        addi    $s0, $zero, 4
        addi    $s1, $zero, 10
        addi    $s2, $zero, 12
        addi    $t0, $zero, 0           #index = $t0
        sw      $s0, myArray($t0)       #Store contents of s0 in first position of array
        addi    $t0, $t0, 4             #increment the index by 4
        sw      $s1, myArray($t0)       #Store contents of s2 in second position of array
        addi    $t0, $t0, 4             #increment the index by 4
        sw      $s2, myArray($t0)       #Store the contents of s3 in third location of array
        lw      $t6, myArray($zero)     #load the word in the first location of myArray into $t6

        li      $v0, 1               }
        addi    $a0, $t6, 0          }  #Print the value of t6
        syscall                      }
```

Array using *while* loops

```
.data
        myArray: .space 12              #declare an array of 3 elements
        newline  : .asciiz "\n"
.text
main:
        addi    $s0, $zero, 4        }
        addi    $s1, $zero, 10       }  #The three values are stored in 3 registers
        addi    $s2, $zero, 12       }
        addi    $t0, $zero, 10          #index is at t0
        sw      $s0, myArray($t0)
        addi    $t0, $t0, 4
        sw      $s1, myArray($t1)
        addi    $t0, $t0,4
        addi    $t0, $zero, 0
while:
        beq     $t0, 12, exit
        lw      $t6, myArray($t0)
        addi    $t0, $t0, 4
        li      $v0, 1               }
        addi    $a0, $t6, 0          }  #Print the no
        syscall                      }

        li      $v0,4                }
        la      $a0, newline         }  #Print a new line
        syscall                      }
        j       while
exit:
```

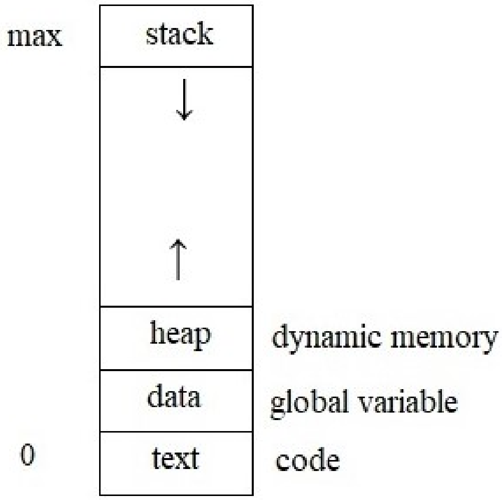
Array Initialization

```
.data
myArray: .word 100:3                    #Declare an array in RAM having 3 elements each initialized
                                                  to 100
newline  : .asciiz "\n"

.text
main:
        add     $t0, $zero, 0
while:
        beq     $t0, 12, exit
        lw      $t6, myArray($t0)
        addi    $t0, $t0, 4
        li      $v0, 1
        move    $a0, $t6
        syscall
#Print a new line
        li      $v0, 4
        la      $a0, newline
        syscall
        j       while
exit:
        li      $v0, 10
        syscall
```

Introduction to Recursion

Process: A program is in execution



Recursive Function: A function which call itself

factorial(n) = 1 if n=0
n*factorial(n-1) if n≥1

For example
4! = 4 x 3 x 2 x 1 = 24
3! = 3 x 2 x 1 = 6

Advantage:

- Efficient for Searching and Sorting
- Easier for complex problem, divides a problem into smaller problem until we reach base case

C program for factorial

```
# include<stdio.h>
int findFact(int number)
int main()
        {
                printf("The factorial of 3 is %d,  findFact(3) );
                return 0;
        }
Int findFact( int number )
{
        if (number == 0)
        return 1;
        else
        return number*findFact(number-1);
}
```

Equivalent Mips program for above factorial function

```
.data
        promptMessage: .asciiz  "enter a number to find factorial"
        resultMessage: .asciiz "\n the factorial of the number is "
        theNumber: .word 0
        theAnswer: .word 0
.text
        .global main
main:
        # Read the number from the user
        li $v0,4
        la $a0,promptMessage
        syscall
        li $v0,5
        syscall
        sw $v0,theNumber
        # Call the factorial function
        lw $a0,theNumber
        jal findfactorial
        sw $v0,theAnswer
        # Display the results
        li $v0,4
        la $a0,resultMessage
        syscall
        li $v0,1
        lw $a0, theAnswer
        syscall
        # Tell the OS that this is the end of program
        li $v0,10
        syscall

.globl findFactorial
findFactorial:
                subu $sp,$sp,8
                sw $ra,($sp)   # storing the value of ra to stack
                sw $s0,4($sp)
        # Base Case
                li $vo,1
                beq $a0,0,factorialDone        # Factorial will rewind
                move $so,$a0                   # findFactorial(number-1)
                sub $ao,$ao,1
                jal findFactorial                        # it will execute when
                mult $v0,$s0,$v0               # recursion will be unwinding
factorialDone:
                lw $ra,($sp)                   # Restoring ra from stack
                lw $s0,4($sp)
                addi $sp,Sp,8
                jr $ra
```

Multi-Dimensional Array in MIPS

In reality memory is a single dimensional entity.
Ways to represent multi-dimensional array- Row major and Column major

int array[3][4]

[2][0]	[2][1]	[2][2]	[2][3]
[1][0]	[1][1]	[1][2]	[1][3]
[0][0]	[0][1]	[0][2]	[0][3]

Row Major: Place all rows sequentially (one after the another)

8	9	10	11
4	5	6	7
0	1	2	3

11	Arr[2][3]
10	Arr[2][2]
9	Arr[2][1]
8	Arr[2][0]
7	Arr[1][3]
6	Arr[1][2]
5	Arr[1][1]
4	Arr[1][0]
3	Arr[0][3]
2	Arr[0][2]
1	Arr[0][1]
0	Arr[0][0]

Address = baseAddress + (rowIndex * columnSize + columnIndex) * datasize

Column Major: Placing column sequentially

2	5	8	11
1	4	7	10
0	3	6	9

11	Arr[2][3]
10	Arr[1][3]
9	Arr[0][3]
8	Arr[2][2]
7	Arr[1][2]
6	Arr[0][2]
5	Arr[2][1]
4	Arr[1][1]
3	Arr[0][1]
2	Arr[2][0]
1	Arr[1][0]
0	Arr[0][0]

Address = baseAddress + (columnIndex * rowSize + rowIndex) * datasize

Implementing 2D Array

```
.data
        mdArray:  .word 2,5        #Square Matrix
                  .word 3,7
        size:  .word 2
        .eq  DATA_SIZE  4     #constant
.text
        main:
        la $a0,mdArray      #Reg a0 has base address of mdArray
        lw $a1,size         #a1 has size
        jal sumDiagonal    #add elements of diagonal takes 2 arguments which are in a0 and a1
        move $a0,$v0   #when I returns the sum will be in v0 & then I move it to a0 as I want to print it on screen
        li $v0,1
        syscall       #display arguments in a0 to screen
        li $v0,10
        syscall      #End of the program
sumDiagonal:
        li $v0,0       #sum=0
        li $t0,0       #Reg t0 as the index
sumloop:
        mul $t1,$t0,$a1   #t1 ← t0*a1(Rowindex*Columnsize)
        add $t1,$t1,$t0   # (Rowindex*Columnsize) + Columnindex(it is equal to row index since we are
                                                              interested in the diagonal)
        mul $t1,$t1,DATA_SIZE
        add $t1,$t1,$a0     #adding base address
        lw $t2,($t1)        #contents of        pointed by t1 is moved to t2
#if i < size then loop again
        addi $t0,$t0,1
        blt $t0,$a1,sumloop
        jr $ra
```