Práctica 7

- Cuestión 8.

➢ Escribe el código que calcula la suma de los elementos de la diagonal principal de una matriz 4x4 de valores enteros introducida por teclado. Muestra la suma por pantalla.

Añadidos.
Decir al usuario que fila / columna se está introduciendo. Mostrar la matriz una vez completada, preguntar si es correcta, si es así, haz el calculo, si no, vuelve a pedir datos. Una vez mostrado el cálculo, preguntamos al usuario si quiere hacer alguna otra operación más.

Explicación: *La manera que he usado para hacer este problema fue el de usar un solo bucle iterando sobre todas las posiciones de la matriz, y sabiendo que la matriz es de 4x4, averiguo en cuál columna estamos dividiendo la posición en la que estamos de la matriz entre 4 y la fila multiplicando la columna por 4 y restandosela a la posición.*

Este ejercicio creo que fue el más complicado que he hecho hasta el momento, debido a la cantidad de datos que tengo que maniobrar y los cálculos que se hacen por cada iteración del bucle.

También en este ejercicio y en algunos siguientes he decidido separar las funciones reutilizables y las funciones específicas del programa llamandolas "Secciones".

```
# Escribe el código que calcula la suma de los elementos de la diagonal principal
# de una matriz 4x4 de valores enteros introducida por teclado. Muestra la suma por pantalla.

# Añadidos.
# Decir al usuario que fila / columna se esta introduciendo
# Mostrar la matriz una vez completada
# Preguntar si es correcta, si es asi, haz el calculo, si no, vuelve a pedir datos. (S/N)
# una vez mostrado el calculo, preguntamos al usuario si quiere hacer alguna otra operacion mas? (S/N).
###################################
```

```
###################################
#
# Primero, vamos a ver de que maneras podemos hacer la logica de la matriz, ya que es lo unico más "groso"
del ejercicio.
# Manera 1: 2 bucles, uno para iterar sobre las columnas y otro sobre filas.
# Manera 2: 1 bucle,  que itere sobre 16 posiciones, y separamos las columnas en multiplos de 4 empezando
desde 1
#
# La manera 2 parece ser la más optima, pero tambien la más tediosa, de todos modos, parece interesante, lo
hare
#    usando la manera 2.
# Ahora.. como lo implementamos? (20min más tarde) Pues de la siguiente manera.
#
#  0 / 4 = 0  <-  0 | [ 3][ 2][ 1][ 0]
#  4 / 4 = 1  <-  4 | [ 7][ 6][ 5][ 4]
#  8 / 4 = 2  <-  8 | [11][#Y][ 9][ 8] <- La matriz está representada como está en Memoria!!
# 12 / 4 = 3  <- 12 | [15][14][13][12]
#
# X = Columna  (desde 0 - 3)
# Y = Iterador (desde 0 - 15)
# Divisor = 4
#-----------------------------------------------------------------
# Ejemplo:
#
#    Posicion de la matriz: 10
#
#    Y = 10, entonces, X (desde de 0 - 3) = (Y / Divisor)
#    Entonces, Columna a mostrar = X + 1 = 3
#    Entonces, Fila a mostrar    = Y - (X * 4) + 1 = 3
#-----------------------------------------------------------------
# - Formulas que usarmeos:

# Contador de fila a mostrar al usuario    = X + 1
# Contador de columna a mostrar al usuario = Y - (X * 4) + 1
#
# Adicional, al hacer la matriz, si la hacemos de bytes, podemos usar el propio contador del bucle
# para realizar los saltos de registro a registro, y ahorrarnos más registros.
###################################

#########################
#         DATOS         #
#########################

.data           # Especificamos que queremos empezar a manipular datos para empezar desde el segmeto de
datos.
```

```
#   Nums  #
MAT: .space 16     # Reservamos la matriz.
DIVISOR: .byte 4   # Numero bytes de salto (para dividir el contador).
MATSIZE: .byte 16  # Numero total de posiciones.

# Strings #
# Strings para pedir y mostrar los datos
DEMANDNUM0: .asciiz "Inserte el numero de la fila ("
ROWMAT: .space 1
DEMANDNUM1: .asciiz ") columna ("
COLMAT: .space 1
DEMANDNUM2: .asciiz "): "

# String para pregunar si la matriz es la correcta
ISMATCORRECT: .asciiz "Es esta la matriz que esperabas?\n( [S]i o [Cualquier caracter] / [N]o ): "

# String para pregunar si quiere continuar o no.
CONTINUE: .asciiz "Quieres dar por finalizada esta tarea?\n( [S]i o [Cualquier caracter] / [N]o ): "

# Strings separadores (decorado para separar los numeros a la hora de mostrar los numeros a sumar).
SEPARATORNUM: .asciiz " + "
SEPARATORRES: .asciiz " = "



########################
#     INSTRUCCIONES    #
########################

.text         # Especificamos que queremos empezar a manipular instrucciones para empezar desde el segmeto de instrucciones.

# $s0 = @ROWMAT, $s1 = @COLMAT, $s2 = @MATSIZE, $s3 = *MATSIZE ..... Valores iniciales.
jal setInitialValues

# Inicio del programa.
main:

# Seteamos los contadores a 0.
jal resetCountersAndMatAccess

# Iniciamos el bucle para pedir los datos y ir guardandolos en memoria.
readMatLoop:
  jal updateUserCounters        # Actualizamos el contador que mostramos por pantala.
  bne $t9, $s4, skipNewLineR    # Si el valor del contador de columna del usuario es 1, imprime una linea para separar.
```

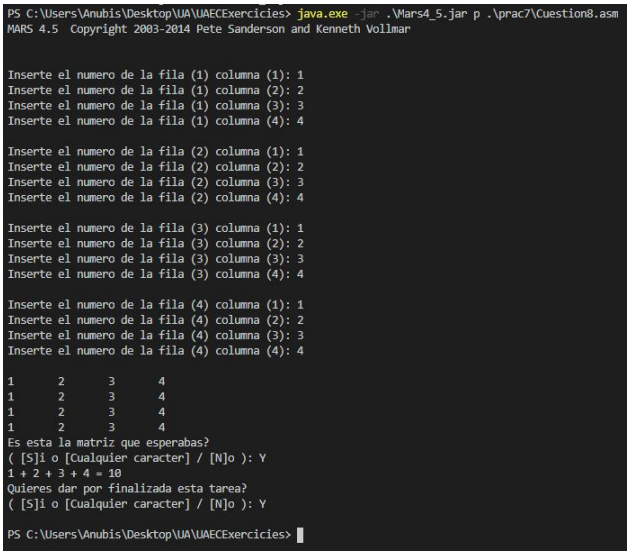
```
  jal printCharNewline          # Imprime una linea
  skipNewLineR:

  jal readInt                   # Pide el numero para una posicion de la matriz
  jal saveMatSlot               # Guarda el numero en la posicion de memoria.
  jal incrementMatCounterNormal # Incrementa el contador, y salta al siguiente registro.
blt $t0, $s3, readMatLoop       # Si el contador es menor que la cantidad de posicines de la matriz,
continua.

jal resetCountersAndMatAccess

# Iniciamos el bucle para mostrar la matriz desde memoria.
printMatLoop:
  jal updateUserCounters

  bne $t9, $s4, skipNewLineP
  jal printCharNewline
  skipNewLineP:

  jal loadMatSlot             # Pedimos el valor de la posicion $t0 (registro $t1) de la matriz desde
memoria.
  move $a0, $v0               # movemos el dato desde $v0 a $a0 para mostrarlo.
  jal printInt                # Imprimimos el dato.
  jal printCharTab            # Imprimimos un espaciado (tabulador).
  jal incrementMatCounterNormal
blt $t0, $s3, printMatLoop

jal resetCountersAndMatAccess

# Preguntamos si la matriz es la correcta
jal printCharNewline

la $a0, ISMATCORRECT          # Cargamos el mensaje
jal printStr                  # Mostramos el mensaje
jal readChar                  # Pedimos un caracter

beq $v0, $s5, main            # Si el caracter perdido es "N", entonces, vuleve a pedir la matriz, si es
cualquier otra cosa, continua.

# Iniciamos el bucle para sumar y mostrar la diagonal
jal printCharNewline
sumMatDiag:
  jal updateUserCounters
  jal loadMatSlot
  move $a0, $v0
```

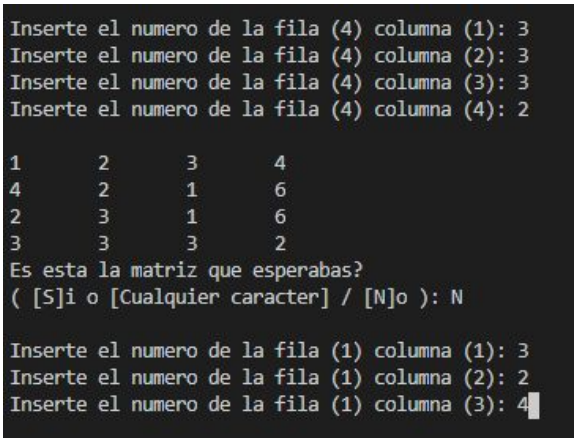
```
  add $t3, $t3, $a0           # Hacemos la suma mediante vamos iterando por el bucle
  jal printInt                # Y a la vez vamos mostrando el valor de cada diagonal.
  beq $t8, $s7, skipFancy

  fancy:
  la $a0, SEPARATORNUM        # Ponemos un separador entre los numeros para que quede mas bonito
  jal printStr

  skipFancy:
  jal incrementMatCounterX4add1
blt $t0, $s3, sumMatDiag

la $a0, SEPARATORRES          # Ponemos un separador entre los  y el resultado para que quede mas bonito.
jal printStr

move $a0, $t3                 # Mostramos la suma.
jal printInt

jal printCharNewline

la $a0, CONTINUE              # Preguntamos al usuario si quiere continuar.
jal printStr
jal readChar
beq $v0, $s5, main            # Si el caracter perdido es "N", entonces, vuleve a pedir la matriz, si es
cualquier otra cosa, continua.

jal exit                      # Salimos del programa

############################
#        SECCIONES         #
############################

# Las secciones serán funciones que tendrán influencia directa sobre la lógica principal del programa.

# Nos guardamos la direccion / valores de los registros que usaremos después etc etc.
setInitialValues:
la $s0, ROWMAT  # Direccion de memoria del contador de fila para mostrar al usuario.
la $s1, COLMAT  # Direccion de memoria del contador de columna para mostrar al usuario.
la $s2, MATSIZE # Direccion de memoria de inicio de la matriz.
lb $s3, 0($s2)  # Valor del tamaño de la matriz.
li $s4, 1       # Valor que usaremos para comparar si estamos en la columna 1. ($9 == $s1).
li $s5, 'N'     # Valor para comprar si ha escrito N.
la $s6, DIVISOR # Direccion de memoria del Divisor.
lb $s7, 0($s6)  # Valor del divisor (4).
li $t3, 0       # Regisitro que usaremos para hacer la suma de la diagonal.
jr $ra          # Vuelve al programa principal.
```

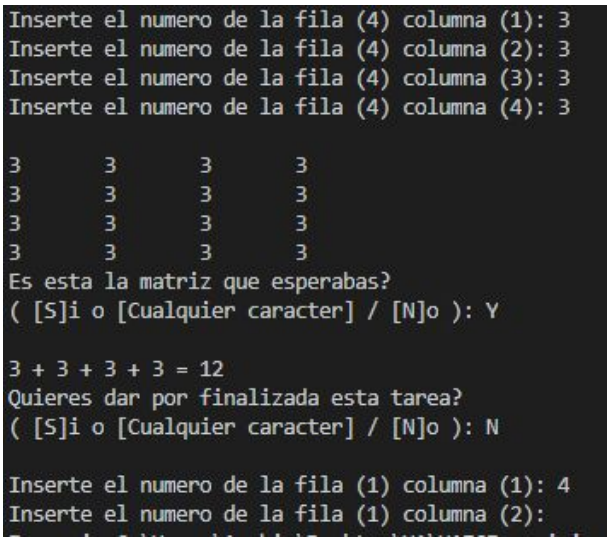
```
# Funcion que resetea los contadores del bucle y el registro origen de la matriz.
resetCountersAndMatAccess:
li $t0, 0
la $t1, MAT
jr $ra            # Vuelve al programa principal

# Incrementa el contador $t0 y registro $t1
incrementMatCounterNormal:
add $t0, $t0, 1
la  $t1, MAT      # $t1 = Posicion de registro.
add $t1, $t1, $t0 # $t1 = Posicion de registro + salto del contador.
jr  $ra           # Vuelve al programa principal

# Incrementa el contador y el registro de 4 + 1 en 4 + 1 para acceder justo a la diagonal.
incrementMatCounterX4add1:
add $t0, $t0, $s7
add $t0, $t0, 1
la  $t1, MAT      # $t1 = Posicion de registro.
add $t1, $t1, $t0 # $t1 = Posicion de registro + salto del contador.
jr  $ra           # Vuelve al programa principal

# Guarda en memoria el valor leeido en la posicion $t1.
saveMatSlot:
sb $v0, 0($t1)
jr $ra            # Vuelve al programa principal

# Retorna un valor de la matriz desde memoria de la posicion $t1.
loadMatSlot:
lb $v0, 0($t1)
jr $ra            # Vuelve al programa principal

# Actualiza el valor del contador para ROWMAT y COLMAT.
updateUserCounters:
div   $t0, $s7 # Y / 4 = X
mflo  $t8      # val de X

mult  $t8, $s7 # X * 4
mflo  $t9      # Val de Y sin procesar

sub   $t9, $t0, $t9 # Y = Y - (X * 4)

addi  $t8, $t8, 1 # X = X + 1
addi  $t9, $t9, 1 # Y = Y + 1

sb    $t8, 0($s0) # ROWMAT
```

```
sb    $t9, 0($s1) # COLMAT

jr $ra      # Vuelve al programa principal

############################
#        FUNCIONES         #
############################

# Funcion lee un entero mostrando la petición.
readInt:
la $a0, DEMANDNUM0  # Mostramos la primera parte del mensaje
li $v0, 4
syscall

lb $a0, 0($s0)      # Mostramos de la fila del contador de usuario
li $v0, 1
syscall

la $a0, DEMANDNUM1  # Mostramos la segunda parte del mensaje
li $v0, 4
syscall

lb $a0, 0($s1)      # Mostramos de la columna del contador de usuario
li $v0, 1
syscall

la $a0, DEMANDNUM2  # Mostamos el final del mensaje
li $v0, 4
syscall

li $v0, 5  # Pedimos un entero
syscall

jr $ra     # Vuelve al programa principal

# Funcion, lee un caracter
readChar:
li $v0, 12 # funcion leer caracter
syscall    # Lee un caracter a $v0

jr $ra     # Vuelve al programa principal

# Funcion, imprime un string
printStr:
li $v0, 4  # funcion imprime un string
syscall    # Lee un caracter a $v0
```

```
jr $ra     # Vuelve al programa principal

# Funcion, imprime un entenro
printInt:
li $v0, 1  # Función imprimir
syscall    # Escribe el valor de $a0

jr $ra     # Vuelve al programa principal

# Funcion, imprime un espaciador (tabulador)
printCharTab:
li $a0, '\t'
li $v0, 11
syscall

jr $ra

# Funcion, imprime una nueva linea
printCharNewline:
li $a0, '\n'
li $v0, 11
syscall

jr $ra

# Sale del programa
exit:
li $v0, 10
syscall
```







Práctica 8

- Cuestión 7.

➢ Completa el siguiente código de partida que pide el radio por teclado y tiene que calcular y mostrar en la consola la longitud de la circunferencia y el área del círculo.

Explicación: *No hay mucho que explicar, ya que las fórmulas son simples, y la lógica para implementarlo sencilla. Lo que si que se podría mencionar es que en esta práctica, al no haberse mencionado el convenio de $f12 y $f14, no se están usando correctamente.*

```
#####################################
#                                   #
# Código de partida de la cuestión 7 #
# Cálculo longitud circunferencia    #
# y el área del círculo              #
#                                   #
#####################################

#########################
#         DATOS         #
#########################
.data

demanaPi:  .asciiz "Dame el valor de pi: "
pideRadio: .asciiz "Dame el radio: "
long:   .asciiz "Longitud de la circunferencia = "
super:  .asciiz "Area del circulo = "

#########################
#     INSTRUCCIONES     #
#########################
.text

jal defConsts   # Definimos constantes.

jal pideP       # Pide pi y lo guarda en $f0

mov.s $f1, $f0  # Mueve $f0 (valor de Pi) a $f1.

jal pideR       # Pide el radio y lo guarda en $f0

mov.s $f2, $f0  # Mueve $f0 (valor del radio) a $f2.

jal muestraL    # Calcula y muestra la longitud.

jal muestraA    # Calcula y muestra el area.
```

```
jal exit        # Sale del programa.

############################
#        SECCIONES         #
############################
defConsts:
li $s0, 2           # Constante entero, valor del num 2
mtc1 $s0, $f20      # Movemos $s0 a $f20, Constante flotante simple, valor del num 2
cvt.s.w $f20, $f20  # El valor sigue siendo entero, hay que pasarlo a flotante simple.
jr   $ra            # Vuelve al programa principal.

# Pedimos el numero Pi
pideP:
li $v0, 4
la $a0, demanaPi
syscall

li $v0,6
syscall
jr   $ra          # Vuelve al programa principal.

# Pedimos el radio.
pideR:
li $v0, 4
la $a0, pideRadio
syscall

li $v0, 6
syscall
jr   $ra           # Vuelve al programa principal.


# Calcula y Muestra la longitud
muestraL:
li $v0, 4
la $a0, long
syscall

# $f1 = pi en flotante simple.
# $f2 = radio en flotante simple.
# $f20 = valor del num 2 en flotante simple.
```

```
# Formula de la longitud: 2 * Pi * R.
# Procuramos siempre guardar el resultado final en $f12 para mostrar directamente el resultado.
mul.s $f12, $f1, $f2     # $f12 = pi * R
mul.s $f12, $f12, $f20   # $f12 = (pi * R) * 2

li $v0, 2
syscall

li $a0, '\n'    # Imprimimos un salto de linea
li $v0, 11
syscall

jr   $ra          # Vuelve al programa principal.

# Calcula y Muestra el Area
muestraA:
li $v0, 4
la $a0, super
syscall

# $f1 = pi en flotante simple.
# $f2 = radio en flotante simple.
# $f20 = valor del num 2 en flotante simple.

# Formula del area de un circulo: Pi * R^2.
# Procuramos siempre guardar el resultado final en $f12 para mostrar directamente el resultado.
mov.s $f3, $f2    # $f3 = $f2
mul.s $f3, $f3, $f3 # $f2 = $f2^2
mul.s $f12, $f3, $f1  # $f12 = $f2^2 * $f1

li $v0, 2
syscall
jr   $ra         # Vuelve al programa principal.


############################
#        FUNCIONES         #
############################


# Sale del programa.
```

```
exit:
li $v0, 10
syscall
```

```
Dame el valor de pi: 3.14
Dame el radio: 5
Longitud de la circunferencia = 31.400002
Area del circulo = 78.5
-- program is finished running --
```

```
Dame el valor de pi: 3.14
Dame el radio: 2
Longitud de la circunferencia = 12.56
Area del circulo = 12.56
-- program is finished running --
```

- Cuestión 8.

➢ Haz el código que suma los elementos del vector y calcula el valor medio en coma flotante. Muestra el resultado por la consola.

Explicación: *Paso cada valor del array a un registro flotante, lo convierto, y lo sumo a un sumador, hago esto para todos los valores del array, y después lo divido entre su longitud, y muestro el resultado.*

```
#####################################
#                                   #
# Código de partida de la cuestión 8  #
#                                   #
#####################################
# Haz el código que suma los elementos del vector y calcula el valor medio en coma
#  flotante. Muestra el resultado por la consola.

#########################
#         DATOS         #
#########################
.data

array: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

#########################
#     INSTRUCCIONES     #
#########################
.text

jal setInitialValues

# Bucle que trae de memoria, convierte a flotante y suma en flotante.
addAllAndConvert:

 jal moveAndConvert

 jal addAll

 jal inctrementRegAndCounter

blt $t2, $s0, addAllAndConvert
```

```
jal showMed

jal printF

jal exit

############################
#        SECCIONES         #
############################
# Define unos valores iniciales.
setInitialValues:
la $t0, array   # Dirección de memoria del contador del array.
la $t2, 0       # Contador del bucle.

# Constante 10 en flotante para dividir después.
li $s0, 10          # Longitud del array.
mtc1 $s0, $f10      # Movemos $s0 a $10, Constante flotante simple, valor del num 10.
cvt.s.w $f10, $f10  # El valor sigue siendo entero, hay que pasarlo a flotante simple.

# Inicializamos $f2 (registro que usaremos para sumar) a 0.
li $s1, 0           # 0.
mtc1 $s1, $f2       # Movemos $s1 a $2, Constante flotante simple, valor del num 0
cvt.s.w $f2, $f2    # El valor sigue siendo entero, hay que pasarlo a flotante simple.

jr $ra          # Vuelve al programa principal.

# Incrementa la direccion y el contador.
inctrementRegAndCounter:
add $t2, $t2, 1  # Contador.
add $t0, $t0, 4  # Registro.
jr   $ra         # Vuelve al programa principal.


# Mueve el valor del array, despues lo muveve a un registro flotante, y despues lo convierte.
moveAndConvert:
lw $t3, 0($t0)      # Sacamos el valor de cada dir de memoria.
mtc1 $t3, $f1       # $t3 -> $f1.
cvt.s.w $f1, $f1    # Convertimos el valor entero a simple precision Float
jr   $ra            # Vuelve al programa principal.
```

```
# Sumamos por cada iteración.
addAll:
add.s $f2, $f2, $f1
jr   $ra            # Vuelve al programa principal.


# Mostramos la media de la división por la suma.
showMed:
div.s $f12, $f2, $f10
jr   $ra            # Vuelve al programa principal.

############################
#        FUNCIONES         #
############################

# Imprime el valor flotante en $f12.
printF:
li $v0, 2
syscall
jr   $ra          # Vuelve al programa principal.


# Sale del programa
exit:
li $v0, 10
syscall
```

```
5.5
-- program is finished running --
```

Práctica 9.

- Cuestión 5.

➢ Implementar la función float pow(float x;int n) que calcula la potencia n-ésima de x. Los argumentos y los valores se pasan según convenio: x en $f12, n en $a0. El resultado se devuelve en $f0.

Explicación: *He decidido modificar un poco el código de partida, y estructurarlo de la manera que siempre he realizado los ejercicios, con DATOS, INSTRUCCIONES, FUNCIONES, ya que me es mucho más cómodo, y me parece más ordenado y legible. La lógica del programa es simple, multiplicar dicho número sobre sí mismo tantas veces como el número de la potencia. Obteniendo un efecto idéntico al de la funcion Pow(x, p).*

```
#######################################
#                                     #
# Código de partida de la cuestión 5  #
#                                     #
#######################################

#########################
#         DATOS         #
#########################
.data
Xpide: .asciiz "X = "
Npide: .asciiz "n = "
powRes: .asciiz "X^n = "
```

```
#########################
#     INSTRUCCIONES     #
#########################
.text
jal xpide
jal npide

mov.s $f12, $f0
mov.s $f14, $f0
move  $a0, $v0

jal pow

mov.s $f12, $f0

jal printResult

jal exit

############################
#        FUNCIONES         #
############################
# Funcion pow(num, exp)
pow:
  # C O M P L E T A R
  # Usamos el valor de la potencia recibida como contador, y multiplicamos por cada
iteraci□n.
  subi $a0, $a0, 1
  beqz $a0, powCondionalExitOne   # En caso de que la potencia sea 1, o haya llegado a 1,
salimos.
  bltz $a0, powCondionalExitZero  # En caso de que la potencia se 0, devolvemos 1.

  mul.s $f14, $f14, $f12    # Hacemos la multiplicación de x*x
bnez $a0, pow               # Mientras que no sea 0, continua multiplicando.

powCondionalExitZero:
li $s0, 1             # Cargamos un 1 en nuestro registro.
mtc1 $s0, $f14        # Movemos el valor a $f14
cvt.s.w $f14, $f14
```

```
powCondionalExitOne:  # Fin condicional, si la potencia a procesar era 1.
mov.s $f0, $f14       # Movemos el resultado a $f0.

jr $ra

# Pide el numero flotante.
xpide:
la $a0, Xpide
li $v0, 4
syscall
li $v0, 6
syscall

jr $ra

# Pide la potencia.
npide:
la $a0, Npide
li $v0, 4
syscall
li $v0, 5
syscall

jr $ra

# Imprime el resultado.
printResult:
la $a0, powRes
li $v0, 4
syscall

li $v0, 2
syscall

jr $ra

# Cierra el programa.
exit: li $v0, 10
syscall
```

```
X = 2.5
n = 2
X^n = 6.25
-- program is finished running --
```

```
X = 2.0
n = 3
X^n = 8.0
-- program is finished running --
```

```
X = 2.0
n = 0
X^n = 1.0
-- program is finished running --
```

```
X = 2.0
n = 1
X^n = 2.0
-- program is finished running --
```

- Cuestión 6.

Implementar la función max que nos devuelve el valor mayor de dos números en coma flotante. Los argumentos se pasan según convenio en $f12 y $f14 y el resultado se devuelve en $f0.

Explicación: *Compruebo cual de ambos valores es mayor, y usando* `bc1t` *salto a una etiqueta si el valor del flag 1 es positivo, este flag se modificó al hacer la comparación usando* `c.lt.s` *(Compare-Less Than-Simple). En caso contrario, saltó manualmente a otra etiqueta que cambia el valor devuelto al mayor.*

```
#####################################
#                                   #
# Código de partida de la cuestión 6 #
#                                   #
#####################################
```

```
#########################
#         DATOS         #
#########################
.data
Xpide: .asciiz "X = "
Ypide: .asciiz "Y = "
MaxRes: .asciiz "El mayor es "

#########################
#     INSTRUCCIONES     #
#########################
.text
jal pidex

mov.s $f12, $f0

jal pidey

mov.s $f14, $f0

jal max

mov.s $f12, $f0

jal showResults

jal exit


############################
#        FUNCIONES         #
############################

# Funcion Max(float, float).
max:
# C O M P L E T A R

c.lt.s $f14, $f12
bc1t  maxConditionalAssinament1    # Si $f12 es mayor, asignamos $f12 a $f0
```

```
j maxConditionalAssinament2       # Si no, entonces significa que $f14 es mayor, asignamos
$f14 a $f0

maxConditionalAssinament1:        # Asigna $f12 a $f0
mov.s $f0, $f12
j maxConditionalExitNormal        #    y salta al final de la funcion

maxConditionalAssinament2:        # Asigna $f14 a $f0
mov.s $f0, $f14

maxConditionalExitNormal:         # final de la funcion.

jr $ra

# Pide X
pidex:
la $a0, Xpide
li $v0, 4
syscall

li $v0, 6
syscall

jr $ra

# Pide Y
pidey:

la $a0, Ypide
li $v0, 4
syscall

li $v0, 6
syscall

jr $ra

# Muestra el resultado.
showResults:
la $a0, MaxRes
```

```
li $v0, 4
syscall

li $v0, 2
syscall

jr $ra

exit:
li $v0, 10
syscall
```

```
X = 12.4
Y = 22.1
El mayor es 22.1
-- program is finished running --


Reset: reset completed.

X = 22.1
Y = 12.4
El mayor es 22.1
-- program is finished running --
```

```
X = 0
Y = 0
El mayor es 0.0
-- program is finished running --


Reset: reset completed.

X = 4
Y = 4
El mayor es 4.0
-- program is finished running --
```

```
X = -12.3
Y = 999
El mayor es 999.0
-- program is finished running --
```