UNIVERSIDAD DE SANTIAGO DE CHILE

Acercándose al Hardware

Arquitectura de Computadores

Laboratorio: N°2

Curso: 13309

Nombre: -Cristóbal Marchant Osorio

Correo: cristobal.marchant@usach.cl

Profesores: -Leonel Medina

-Manuel Villalobos

-Maximiliano Orellana

Fecha: 6 de diciembre del 2021

UNIVERSIDAD DE SANTIAGO DE CHILE

índice

Introducción

En el siguiente informe se plantean una serie de problemas sobre el manejo de un lenguaje de bajo nivel de nombre “MIPS” (Microprocessor without Interlocked Pipeline Stages), se propone como objetivo darle una solución a cada uno de los problemas planteados y aprender estos para el posterior manejo sobre estos, estos problemas son; parte 1, el uso de las “syscall” para poder leer 2 números y poder imprimir el número mayor de estos; parte 2, encontrar el máximo común divisor entre 2 números a base del algoritmo de Euclides; parte 3, esta parte de divide en la parte “A” la que nos pide diseñar un método de multiplicación sin hacer uso de instrucciones de multiplicación, división y desplazamiento, a base de subrutinas y en la parte “B”, se debe diseñar un método de división sin hacer uso de instrucciones de multiplicación, división y desplazamiento, pero se puede implementar el método de multiplicación de la parte “A”, para finalizar en la parte 4, aplicando todo lo anteriormente enseñado debemos dar un resultado aproximado de las funciones logaritmo natural y seno con el método de expansiones de Taylor en torno a 0, posteriormente hacer un gráfico mostrando el error de distintos números. La solución a cada uno de estos problemas esta abordada a mayor profundidad en el desarrollo de este informe.

Marco teórico

MIPS es un lenguaje de programación de bajo nivel con las siglas de Microprocesador sin etapas de tubería interconectadas en español, en las cuales se trabajan un gran número de instrucciones simples de 4 bytes junto a direcciones de registros en donde las podemos usar para guardar valores temporales, resultados, números enteros, flotantes de carácter simple o doble entre otros, en el gran número de instrucciones que posee podemos encontrar funciones de salto, suma, resta, multiplicación, división, mover, asignar, condiciones, entre muchas otras, estas mismas funciones tienen variantes para el tipo de dato que se esté trabajando, por ejemplo, si se trabaja con 2 enteros seria “add”, si se trabajó con flotantes simples seria “add.s”, es de esta manera que denota un gran número de funciones.

Desarrollo

En el siguiente bloque se darán a ver los resultados obtenidos durante este proyecto además de una explicación sobre cómo se llegaron a estos y un análisis objetivo para cada uno de estos

PARTE 1: Máximo entre 2 números

```
        .data
p1: .asciiz "Por favor ingrese el primer entero: "
p2: .asciiz "Por favor ingrese el segundo entero: "
pm: .asciiz "el numero maximo es: "
        .text
#-------------------------------------------------------------------------------
li $v0, 4                       #syscall para imprimir un string
la $a0, p1                      #p1 sera la frase a imprimir en la consola
syscall

li $v0, 5                       #syscall para leer un entero
syscall

move $t0, $v0                   #mueve el numero de v0 a t0
#-------------------------------------------------------------------------------
li $v0, 4                       #syscall para imprimir string
la $a0, p2                      #p1 sera la frase a imprimir en la consola
syscall

li $v0, 5                       #syscall para leer un entero
syscall

move $t1, $v0                   #mueve el numero de v0 a t0
#-------------------------------------------------------------------------------
bge  $t0, $t1, numlmax          #condicional si t0 >= t1 ir a "numlmax"

move $t2, $t1                   #mueve t1 a t2
j result                        #salta a donde se imprimira el resultado

numlmax:
move $t2, $t0                   #mueve t0 a t2

result:
li $v0, 4                       #syscall para imprimir string
la $a0, pm                      #pm sera la frase a imprimir en la consola
syscall

li $v0, 1                       #syscall para imprimir entero
move $a0, $t2                   #mueve t2 (osea el mayor) a a0 para ser imprimido
syscall
#-------------------------------------------------------------------------------
end:
li $v0, 10                      #termino del programa
syscall
```

La primera sección (cuando se habla de secciones son las líneas verdes que separan una parte y la otra) en donde se guardan las palabras que serán imprimidas en la consola.

La segunda y tercera sección es para imprimir el mensaje de ingresar número y guardar este número en $t0 y $t1 respectivamente.

La cuarta parte es donde existe una comparación entre $t0 y $t1 para ver quien es mayor si $t0 o $t1 es mayor este se guarda en $t2 el cual será imprimido en “result” con el string de “El número máximo es:”

Y la quinta parte es para el fin del programa

(figura 1, Código máximo entre 2 números)

Como se puede apreciar en este problema se planteó una comparación entre ambas, de igual manera se pensó en una comparación que fuera una resta entre ambas para saber cuál era mayor, pero indagando en la biblioteca de MARS fue posible encontrar esta solución más práctica.

PARTE 2: Máximo común divisor

```
        .data
pr: .asciiz "el maximo comun divisor es:  "
        .text
#-------------------------------------------------------------------------------
li $t6, 81                      #int a = NUMERO 1
li $t7, 153                     #int b = NUMERO 2
#-------------------------------------------------------------------------------
while:                          #Ciclo while ("Ciclo mientras pa los habla hispana")
bne $t6, $t7, A                 #si ($t6 != $t7) ir a A
#-------------------------------------------------------------------------------
                                #cuando se rompe el ciclo avanzara a imprimir el MCD y Finalizar el programa
result:
li $v0, 4                       #syscall para imprimir string
la $a0, pr                      #pr sera la frase a imprimir en la consola
syscall

li $v0, 1                       #syscall para imprimir entero
move $a0, $t6                   #mueve t6 (osea el MCD) a a0 para ser imprimido
syscall
#-------------------------------------------------------------------------------
END:                            #sino no lo es termina el programa
li $v0, 10
syscall
#-------------------------------------------------------------------------------
A:
bge $t6, $t7, B                 #Si $t6 >= $t7 ir a B

                                #Sino
sub $t7, $t7, $t6               #t7= t7-t6
j while                         #vuelve a evaluar en el while

B:
sub $t6, $t6, $t7               #t6= t6-t7
j while                         #vuelve a evaluar en el while
```

(figura 2, Código máximo común divisor)

La primera sección es el texto que se imprimirá, la segunda sección son los números escritos en “Duro” que se utilizaran

En la tercera sección se encuentra el While que dice mientras $t6 sea distinto de $t7 salta a “A”, en “A” hay otra condición que sea que si $t6 es mayor o igual a $t7 salte hacia “B”, en cada uno de los saltos se resta el número mayor con el menor, y así sucesivamente, de este modo cuando ambos números sean iguales se imprimirá el string del “el máximo común divisor es:” y se imprimirá $t6, para finalmente después de esto terminar la ejecución del programa. *(figura*

Esta idea fue hecha a base del algoritmo de Euclides de forma más práctica agregando ciclos.

PARTE 3: A) Multiplicación

```
        .data
A: .float 6.0
B: .float 5.0
ONE: .float 1.0
ZERO: .float 0.0
ACUM: .float 0.0
        .text

l.s $f0, A
l.s $f1, B
l.s $f6, ACUM
l.s $f7, ONE
l.s $f8, ZERO
#--VERIFICACION-----------------------------------------------------------------
veri:
c.eq.s $f8, $f0
bclt end
c.eq.s $f8, $f1                 #verifica si son 0 o si son negativos
bclt end                        #simplemente es un paso de verificacion
c.lt.s $f8, $f0                 #para que se trabaje mejor
bclf veri2                      #en el problema 3.B tambien esta presente
c.lt.s $f8, $f1
bclf veri3
#-------------------------------------------------------------------------------
while:
c.le.s $f1, $f8                 #while principal
bclf operacion                  #ira comprobando hasta que f1 llegue a 0
#-------------------------------------------------------------------------------
c.eq.s $f15, $f7
bclf end                        #agrega negatividad si es que la poseen
sub.s $f2, $f8, $f2
#-------------------------------------------------------------------------------
end:
li $v0, 10                      #termino del programa
syscall
#-------------------------------------------------------------------------------
operacion:
add.s $f2, $f2, $f0             #operacion principal
sub.s $f1, $f1, $f7             #gurda en f2 la suma con f0 y f1 va disminuyendo
j while
#---CASO EN DONDE UNO O LOS 2 NUMEROS SON NEGATIVOS-----------------------------
veri2:
sub.s $f0, $f8, $f0
add.s $f15, $f15, $f7           #casos donde son negativos
j veri                          #se restan con 0 para quitar negatividad
veri3:                          #y se guarda un 1 para saber si el resultado final
sub.s $f1, $f8, $f1             #necesita ser negativo, si tiene 0 o 2, no necesita serlo
add.s $f15, $f15, $f7
j while
```

(figura 3, Código multiplicación)

En la primera sección están anotados los datos y datos auxiliares para este desarrollo, cabe destacar que se está trabajando en float de carácter simple.

En la segunda sección se verifica que si uno de los 2 número es igual a 0, el programa termine y el resultado estará en $f2 ósea 0, también que si una variante o ambas son menores que 0, se transforman a positivo para un mejor trabajo, además de que se guarda para el final una variante si corresponde a transformar el resultado final para que sea negativo o positivo, el desarrollo de esta verificación se hace en la parte final, en donde se resta por 0 para que se haga positivo el numero además al “acum” $f15 se le agrega un 1.

En la tercera parte tener el uso del While en el cual se compara el $f1 con el $f0, o en otras palabras el 0, para que este salte a la parte de “operación” en esta parte se le resta 1 a $f1 y se le agrega en $f2 el $f1 para que este se vaya acumulando

En la cuarta parte se es cuando el While de la multiplicación termino y se evalúa $f15, para ver cuando se haya que transformar a negativo el número, es decir, si $f15=0 no se hace nada porque ambos son positivos, si $f15=1 se transforma el resultado final a negativo y si $f15=2 no se hace nada.

Se pensó en esta solución en el uso exclusivos de enteros, pero para la parte B, tuve que expandir su uso para enteros y flotantes para que estos puedan funcionar, también cabe destacar que se desarrolló esta solución analizando casos especiales, es decir, cuando es negativo o 0, y también se pensó de la manera de cuantas 3x4 es lo mismo que 3+3+3+3, pero cada suma de 3 es por un ciclo, es decir, ciclo 1 = 3, ciclo 2 = 6, ciclo 3 = 9 y ciclo 4 = 12.

PARTE 3: B) división

```
        .data
A: .float 4.0
B: .float 4.0
TEN: .float 10.0
ZERO: .float 0.0
ACUM: .float 0.0
ONE: .float 1.0
DEC: .float 0.1
DEC2: .float 0.01
        .text
#resultado final en $f2
# numero A / numero B
l.s $f0, A                      #int numero 1
l.s $f1, B                      #int numero 2
l.s $f2, ACUM                   #SUMA FINAL
l.s $f3, ACUM                   #RESULTADO MULTI 1
l.s $f6, ACUM                   #RESULTADO DIVI 1
l.s $f7, ONE
l.s $f8, ZERO
l.s $f9, DEC
l.s $f5, TEN
l.s $f10, ACUM
l.s $f11, ACUM
l.s $f12, ACUM
#$f15 indica si el resutado es negativo o positivo
#---CASO EN DONDE UNO O LOS 2 NUMEROS SON NEGATIVOS-----------------------------------
vericero:
c.eq.s $f8, $f0
bc1t end                                #caso donde la multiplicacion debe dar 0
vericero2:
c.eq.s $f8, $f1
bc1t end
#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
veri:
c.lt.s $f8, $f0
bc1f veri2
c.lt.s $f8, $f1                 #verifica que el numero sea mayor o menor que cero
bc1f veri3
j primer
#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
veri2:
sub.s $f0, $f8, $f0
add.s $f15, $f15, $f7
j veri
veri3:                          #si es menor que cero lo resta con cero para obtener un positivo
sub.s $f1, $f8, $f1             #al final se le volvera a cambiar el signo para obtner el resultado correcto
add.s $f15, $f15, $f7
j primer
#--PASO DE LA DIVISION-----------------------------------------------------------------
primer:                          #inicial con el valor de A y B
while:
c.lt.s $f0, $f1                 #CASO CUANDO F0 SEA MENOR QUE F2 Y CUANDO ES ASI SALTA A FINAL
bc1f operacion                  #es decir, a f0 se le ira restando f1, hasta quedar con el resto
#cuando sea $f0 sea 0
c.eq.s $f8, $f0                 #caso cunado el resto sea 0, ej: 4/2 o 10/2
bc1t end1

j while2                        #salta a while2 si quedan resto

operacion:
sub.s $f0, $f0, $f1             #operacion de esta para obtener el resto
add.s $f6, $f6, $f7             #suma para saber cuantas veces cae el f1 en el f0
j while
#-------------------------------------------------------------------------------------
PasoDecimal:                    #paso que sera utilizado cuando estemos con el 0.01
l.s $f5, TEN
l.s $f9, DEC2                   #se restablecen algunas variantes con el fin
l.s $f0, ZERO                   #de volver a tenerlas en su estado inical para
add.s $f0, $f0, $f3             #tranjarlas con el 0.001
l.s $f3, ZERO
add.s $f11, $f11, $f7
#--PASO DE LA MULTIPLICACION-----------------------------------------------------------
while2:
c.le.s $f5, $f8                 #practicamente multiplica al resto x 10
bc1f operacion2                 #con el fin que este sea divido para sacar
                                #el primer decimal
j while3                        #salta a while 3 para dividir este numero

operacion2:
add.s $f3, $f3, $f0
sub.s $f5, $f5, $f7
j while2
#-----DIVIDE AL DECIMAL-----------------------------------------------------------------
PARTE2:
while3:                         #aqui se divide al numero multiplicado x10
c.lt.s $f3, $f1                 #CASO CUANDO F0 SEA MENOR QUE F2 Y CUANDO ES ASI SALTA A FINAL
bc1f operacion3
#cuando sea $f0 sea 0
c.eq.s $f8, $f0
bc1t end

j PARTE3                        #si queda resto pasa a la parte 3, sino termina el proceso

operacion3:
sub.s $f3, $f3, $f1
add.s $f10, $f10, $f7
j while3
```

(figura 4 y 5, código división)

En este problema fue más extenso que el anterior, ya que en el anterior se tenían por entrada y salidas números enteros, en cambio este tiene de entrada enteros y de salida números enteros y decimales.

En la primera parte se encuntran los datos de entra en duro y numeros axuliares que nos ayudaran en el desarrollo.

En la segunda parte como en el caso anterior se encuntra el paso de verificar si es negativo, esto no seria problema de explicacar yaque se explico anteriormente.

En la tercera parte, “EL PASO DE LA DIVISIÓN”, en esta parte tenemos un While el cual nos permite ver cuantas veces cae el segundo número en el primer número, si el resto es 0 el programa termina, sino avanza al segundo While.

En la cuarta parte agregamos números auxiliares y cambiamos el valor de otros, cabe destacar que esto es solo cuando el resultado tiene más de un decimal.

En la quinta parte, “PASO DE LA MULTIPLICACIÓN” tenemos que el resto se multiplica por 10 (este se encuentra en el primer número) y avanzamos al siguiente paso.

En la quinta parte se divide el resto multiplicado por 10 por el segundo número, y avanza al paso 3

```
#-------------------------------------------------------------------------------
PARTE3:
while4:
c.le.s $f10, $f8                #multiplica el resto que queda x0.1
bc1f operacion4                 #si hay 2 decimales volvera a este paso pero
                                #con 0.01
j casual

operacion4:
add.s $f12, $f9, $f12
sub.s $f10, $f10, $f7
j while4
#-------------------------------------------------------------------------------
casual:
add.s $f2, $f6, $f12
c.eq.s $f3, $f8                 #f3 osea el resto == 0 termina el programa
bc1t end

c.eq.s $f11, $f7                #aqui comprueba si necesita un segundo decimal
bc1t end                        #sino lo necesita temrina el programa
j PasoDecimal                   #y si lo necesita sube y cambia variables para poder
                                #lograrse (se explica en el "PasoDecimal")

end1:
add.s $f2, $f6, $f12            #suma los valores, osea la parte entera y decimal
c.eq.s $f15, $f7                #si es que se acaba el programa
bc1f endfinal
add.s $f2, $f2, $f10            #agrega la negatividad si es que la debe poseer
sub.s $f2, $f8, $f2
j endfinal

end:
c.eq.s $f15, $f7
bc1f endfinal
add.s $f2, $f2, $f10            #agrega la negatividad si es que la debe poseer
sub.s $f2, $f8, $f2
endfinal:
li $v0, 10                      #termino del programa
syscall
```

(figura 6, código división)

En esta sexta parte el resultado obtenido de la división se multiplica por 0.1 y posteriormente se suma al resultado entero.

En esta séptima parte se evalúa si el resto sigue siendo 0, si es así se termina el programa, sino el programa vuelve al “paso decimal” para repetir el proceso y obtener el 2do decimal, en el “paso decimal” como se dijo este cambia algunos valores auxiliares en el que destaca que ya no es 0.1 sino 0.01, de esta manera obtener un resultado de 2 decimales.

En los pasos de end1 y “end” se evalúa si el numero tiene que ser negativo o no (paso idéntico al explicado en la “parte 3 A”) para finalmente guardar la respuesta en $f2.

Este problema se pensó en un principio como el caso contrario a la multiplicación, pero posteriormente se implementó el uso de decimales a este, se tenía planeado en usar solo una sección para la división y otra para la multiplicación, pero me fue más factible hacerlo de esta manera sin tantos enredos.

PARTE 4: A) Funciones logaritmo natural y seno

```
        .data
p1: .asciiz "Por favor ingrese el numero entero: "
ps: .asciiz "El logaritmo natural por expansiones de Taylor (ln(1+x)) es: "
TEN: .float 10.0
ZERO: .float 0.0
ACUM: .float 0.0
ONE: .float 1.0
DEC: .float 0.1
DEC2: .float 0.01
TWO: .float 2.0
TRHEE: .float 3.0
FOUR: .float 4.0
FIVE: .float 5.0
SIX: .float 6.0
SEVEN: .float 7.0
        .text

l.s $f6, ACUM
l.s $f7, ONE
l.s $f8, ZERO
l.s $f31, TWO
l.s $f29, TRHEE
l.s $f27, FOUR
l.s $f28, FIVE
l.s $f26, SIX
#resultado de multiplicacion y division se guardaran en f2, tener ojo
#-------------------------------------------------------------------------------
li $v0, 4                       #syscall para imprimir string
la $a0, p1                      #p1 sera la frase a imprimir en la consola
syscall

li $v0, 6                       #syscall para leer un float simple
syscall
add.s $f19, $f0, $f19
#-------------------------------------------------------------------------------
#-IMPLEMENTACION DE LA MULTIPLICACION-------LOGARITMO NATURAL-------------------
veri:
c.eq.s $f8, $f0                 #verifica si son 0 o si son negativos
bc1t end                        #simplemente es un paso de verificacion
c.lt.s $f8, $f0                 #para que se trabaje mejor
bc1f veri2                      #en el problema 3.B tambien esta presente
#-------------------------------------------------------------------------------
```

(figura 7, código función logaritmo y seno)

En esta parte al igual que las demás se tiene una definición de todas las variantes auxiliares que se usaran en el desarrollo de este código.

En la segunda sección se hace el llamado a la “syscall” para imprimir la oración y leer el numero a utilizar

En la tercera sección se utiliza para verificar si son negativos o un 0, si es 0 termina el programa y si es negativo avanza a “veri2” en donde cambia de signo.

```
add.s $f1, $f0, $f1
casos:
caso2:                          #cuando se eleva por 2
c.eq.s $f30, $f7                #compureba que sea el acum($f30) sea igual a 1
bclf caso3
add.s $f20, $f20, $f2

caso3:                          #cuando se eleva por 3
c.eq.s $f30, $f31               #el elevado a 4 estara en el end
bclf caso4                      #yaque ese sera el ultimo(se nos pidio trabajar con numeros
add.s $f21, $f21, $f2           #cercanos al 0, por lo que entendi)

caso4:                          #cuando se eleva por 4
c.eq.s $f30, $f29
bclf caso5
add.s $f22, $f22, $f2

caso5:                          #cuando se eleva por 5
c.eq.s $f30, $f27
bclf caso6
add.s $f23, $f23, $f2

caso6:                          #cuando se eleva por 6
c.eq.s $f30, $f28
bclf operaciontraspaso
add.s $f24, $f24, $f2

operaciontraspaso:
add.s $f30, $f30, $f7           #aqui se suma 1 al acum
add.s $f1, $f1, $f2             #aqui $f1 toma el valor de $f2
l.s $f2, ZERO
#-------------------------------------------------------------------------------------
while:
c.le.s $f1, $f8                 #while principal
bclf operacion                  #ira comprobando hasta que f1 llegue a 0
#-------------------------------------------------------------------------------------
end:
c.lt.s $f30, $f26               #este es el caso para 4
bclt casos                      #volvera a casos y si esta listo
add.s $f25, $f25, $f2           #se guardara el valor de f2 en f22
j paso2
#-------------------------------------------------------------------------------------
operacion:
add.s $f2, $f2, $f0             #operacion principal
sub.s $f1, $f1, $f7             #gurda en f2 la suma con f0 y f1 va disminuyendo
j while

#---CASO EN DONDE UNO O LOS 2 NUMEROS SON NEGATIVOS-------------------------------------
veri2:
sub.s $f0, $f8, $f0
add.s $f15, $f15, $f7           #casos donde son negativos
j veri                          #se restan con 0 para quitar negatiridad
#-------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------
paso2:
add.s $f24, $f21, $f24
l.s $f0 ZERO
l.s $f30, ZERO
add.s $f0, $f20, $f0
l.s $f1, TWO
#--PASO DE LA DIVISION-----------------------------------------------------------------
#$f15 indica si el resutado es negativo o positivo

#--PASO DE LA DIVISION-----------------------------------------------------------------
primerm:
l.s $f2, ACUM                   #SUMA FINAL
l.s $f3, ACUM                   #RESULTADO MULTI 1
l.s $f6, ACUM                   #RESULTADO DIVI 1
l.s $f7, ONE
l.s $f8, ZERO
l.s $f9, DEC
l.s $f5, TEN
l.s $f10, ACUM
l.s $f11, ACUM
l.s $f12, ACUM                          #inicial con el valor de A y B
add.s $f30, $f30, $f7
whilem:
c.lt.s $f0, $f1                 #CASO CUANDO F0 SEA MENOR QUE F2 Y CUANDO ES ASI SALTA A FINAL
bclf operacionm                 #es decir, a f0 se le ira restando f1, hasta quedar con el resto
#cuando sea $f0 sea 0
c.eq.s $f8, $f0                 #caso cunado el resto sea 0, ej: 4/2 o 10/2
bclt endml

j whilem2                       #salta a while2 si quedan resto

operacionm:
sub.s $f0, $f0, $f1             #operacion de esta para obtener el resto
add.s $f6, $f6, $f7             #suma para saber cuantas veces cae el f1 en el f0
j whilem
```

(figura 8 y 9, código función logaritmo y seno)

En esta sección se encuentran los números que serán elevados, ósea serán multiplicados por si mismos, de esta manera tendremos desde $19 hasta $f25 el elevado del numero ingresado.

En la sección del While y las 2 secciones siguientes, juntas conformaran el paso de la multiplicación en donde vuelven a la sección anterior para tomar otro elevado y así hasta terminar.

En esta sección esta el paso de la verificación en donde se quita la negatividad y se guarda un 1 en f15

Paso 2 son variables auxiliares que serán utilizadas en los pasos siguientes

En el paso de la división para el logaritmo natural volvemos a definir variables auxiliares, estas por cada división que hagan sumaran 1 a f30 para elegir el numero a dividir.

El paso el whilem volvemos a hacer uso de la división como se explico anteriormente y esta cuando termina con un resto salta a whilem2, sino termina.

```
#--PASO DE LA MULTIPLICACION-----------------------------------------------------------
whilem2:
c.le.s $f5, $f8                 #practicamente multiplica al resto x 10
bclf operacionm2                        #con el fin que este sea divido para sacar
                                #el primer decimal
j whilem3                       #salta a while 3 para dividir este numero

operacionm2:
add.s $f3, $f3, $f0
sub.s $f5, $f5, $f7
j whilem2
#-----DIVIDE AL DECIMAL-----------------------------------------------------------------
PARTEm2:
whilem3:                                #aqui se divide al numero multiplicado x10
c.lt.s $f3, $f1                 #CASO CUANDO F0 SEA MENOR QUE F2 Y CUANDO ES ASI SALTA A FINAL
bclf operacionm3
#cuando sea $f0 sea 0
c.eq.s $f8, $f0
bclt endm

j PARTEm3                       #si queda resto pasa a la parte 3, sino termina el proceso

operacionm3:
sub.s $f3, $f3, $f1
add.s $f10, $f10, $f7
j whilem3
#-------------------------------------------------------------------------------------
PARTEm3:
whilem4:
c.le.s $f10, $f8                #multiplica el resto que queda x0.1
bclf operacionm4                        #si hay 2 decimales volvera a este paso pero
                                #con 0.01
j casualm

operacionm4:
add.s $f12, $f9, $f12
sub.s $f10, $f10, $f7
j whilem4
#-------------------------------------------------------------------------------------
casualm:
add.s $f2, $f6, $f12
c.eq.s $f3, $f8                 #f3 osea el resto == 0 termina el programa
bclt endm

c.eq.s $f11, $f7                #aqui comprueba si necesita un segundo decimal
bclt endm                       #sino lo necesita temrina el programa
j PasoDecimalm                  #y si lo necesita sube y cambia variables para poder
                                #lograrse (se explica en el "PasoDecimal")

endm1:
add.s $f2, $f6, $f12            #suma los valores, osea la parte entera y decimal
c.eq.s $f15, $f7                #si es que se acaba el programa
bclf endfinalm
add.s $f2, $f2, $f10            #agrega la negatividad si es que la debe poseer
sub.s $f2, $f8, $f2
j endfinalm

endm:
c.eq.s $f15, $f7
bclf endfinalm
add.s $f2, $f2, $f10            #agrega la negatividad si es que la debe poseer
sub.s $f2, $f8, $f2
endfinalm:
c.eq.s $f30, $f7
bclf to2
l.s $f20, ZERO                  #en esta lista se guardan los elevados al numero ingresado
add.s $f20, $f20, $f2           #se divide el numero elevado a 2

l.s $f0, ZERO
add.s $f0, $f0, $f21
l.s $f1 TRHEE
to2:
c.eq.s $f30, $f31               #se divide el numero elevado a 3
bclf to3
l.s $f21, ZERO
add.s $f21, $f21, $f2

l.s $f0, ZERO
add.s $f0, $f0, $f22
l.s $f1, FOUR
to3:
c.eq.s $f30, $f29               #se divide el numero elevado a 4
bclf primerm
l.s $f22, ZERO
add.s $f22, $f22, $f2

finish:
l.s $f2, ZERO                   #se opera para obtner el resultado final en f2
sub.s $f2, $f19, $f20
add.s $f2, $f2, $f21
sub.s $f2, $f2, $f22

li $v0, 4                       #syscall para imprimir string
la $a0, ps                      #pm sera la frase a imprimir en la consola
syscall

li $v0, 2                       #syscall para imprimir entero
mov.s $f12, $f2                 #mueve t2 (osea el mayor) a a0 para ser imprimido
syscall
#-------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------
li $v0, 10                      #termino del programa
syscall
```

(figura 10, 11 y 12, código función logaritmo y seno)

En esta parte el resto se multiplica por 10 para poder dividirlo por el numero de la posición de la función de expansión de Taylor del logaritmo natural.

En esta sección se divide el resto aumentado 10 veces por el numero y se evalúa si el nuevo resto es 0 o aun queda resto, si es 0 avanza al siguiente paso a ser multiplicado por 0.1 para sumarse con el resultado de entero, sino vuelve a empezar el proceso, pero en vez de 0.1 es 0.01, de esta manera también se agrega este segundo decimal a la suma con el entero.

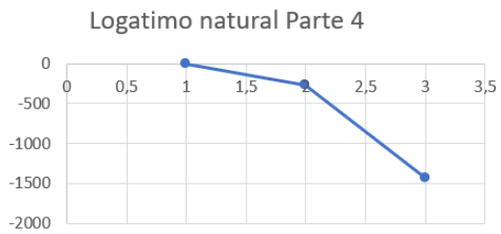
En esta sección se evalúa si necesita el 2do decimal antes mencionado.

En esta parte son los finales en donde se agrega a f2 los resultados anteriores y se verifica negatividad y se la aplica si la necesita.

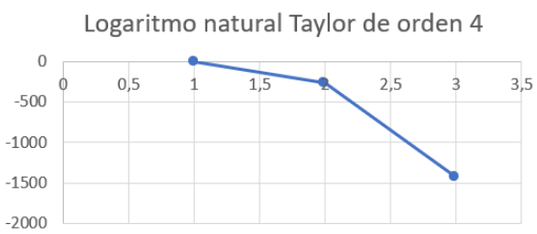
En esta sección encontramos los casos de que numero dividir cuando el numero esta elevado a cierta cifra, de este modo obtendremos el resultado para cada parte de la suma del logaritmo natural por expansiones de Taylor.

En esta sección se realiza la suma final de los valores obtenidos para obtener el valor aproximado del logaritmo natural, también se imprime el resultado por la consola y finalmente se termina el programa.

Parte 4: B)



(grafico 1, función logaritmo natural)



(grafico 2, función logaritmo natural Taylor 4)

		Parte 4: A)	Taylor
ln(1+x)	ln(1+2) x=2	-1,34	-1,33
	ln(1+6) x=6	-264	-264
	ln(1+9) x=9	-1428,75	-1428,75

(tabla 1, función logaritmo natural parte 4 A y Taylor 4)

A base de la información recopilada de los gráficos y la tabla se puede apreciar un error mínimo en el número más bajo, el cual este error puede cuantificarse siendo del 0.01, a lo cual podemos decir que existe un error bastante mínimo que no afecta al resultado en números más grandes.

Conclusión

Como conclusión de este informe se puede decir que se cumplió con la mayor parte de los objetivos a excepción del seno de la parte 4 B, el cual fue difícil de diseñar y aplicar sobre el código, no obstante, el resto de los códigos fueron completados con éxito con lo solicitado, pero en un análisis general podemos decir que se pudo haber hecho una mejor optimización de estos códigos mejorando su rendimiento por muy mínima que sea su labor, también al trabajar los códigos con flotantes se hizo mas largo el uso de instrucciones, al tener que definir algunas cosas antes que todo, además se aprendieron un gran numero de instrucciones de las cuales resultaron un mejor desarrollo y escritura para los próximos códigos, en una mirada objetiva al desarrollo de este trabajo en un futuro se planea resolver todos problemas de manera óptima, trabajando siempre con honestidad y completar con lo solicitado.