

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA



Organización de Computadores
Laboratorio N°2

Alumno: Israel Arias
Sección: 0-L-1
Profesor(a): Leonel Medina

20 de Mayo de 2021

Tabla de contenidos

1. Introducción	1
1.1. Problemas	1
1.1.1. Parte 1: Uso de syscall	1
1.1.2. Parte 2: Subrutinas para multiplicación y división de números	1
1.2. Soluciones	2
1.3. Objetivos	2
2. Marco Teórico	3
3. Desarrollo de la solución	4
3.1. Solución Parte 1	4
3.2. Solución Parte 2	6
3.2.1. Solución Parte A)	6
3.2.2. Solución Parte B)	7
3.2.3. Solución Parte C)	8
4. Resultados	9
4.1. Resultados Parte 1	9
4.2. Resultados Parte 2	9
4.2.1. Resultados Parte A)	9
4.2.2. Resultados Parte B)	9
4.2.3. Resultados Parte C)	9
5. Conclusiones	10

1. Introducción

En la presente experiencia de laboratorio de la asignatura Organización de Computadores, se plantea una actividad en la cual se busca el aprendizaje de aprender como programar en lenguaje ensamblador MIPS, para lo cual la actividad esta dividida en dos partes, la primera busca el uso y aprendizaje de las llamadas de sistema o “syscall”, la segunda parte busca el uso y aprendizaje de subrutinas dentro de un programa. En el presente informe se detallarán los problemas abordados, su metodología de resolución, además se detallarán los objetivos específicos de la experiencia de laboratorio, se presentarán resultados para cada problema abordado y finalmente se efectuará una conclusión de acuerdo los resultados conseguidos.

1.1. Problemas

En la presente experiencia se plantean 4 problemas, divididos en dos secciones, las cuales serán detalladas a continuación:

1.1.1. Parte 1: Uso de syscall

En esta sección se plantea el siguiente problema: “Escribe un programa que lea dos enteros ingresados por el usuario, determine el máximo entre estos dos números y luego imprima este valor. Para ello, utiliza llamadas de sistema, “syscall”, para 1) imprimir en la consola de MARS los mensajes de interacción con el usuario, 2) permitir que el usuario ingrese los números en tiempo de ejecución, 3) imprimir el resultado en la consola, y 4) terminar el programa (exit)”. Además como restricción se solicita que “el cálculo del máximo debe ser realizado en una subrutina, utilizando los registros apropiados para argumentos y salida de un procedimiento”.

1.1.2. Parte 2: Subrutinas para multiplicación y división de números

En esta sección se plantean tres problemas: A) “Escribe un programa en MIPS que calcule la multiplicación de dos enteros mediante la implementación de subrutinas. No se pueden utilizar instrucciones de multiplicación, división y desplazamiento, se debe imple-

mentar una técnica de multiplicación basada en otras operaciones matemáticas y el uso de subrutinas”. Además, se solicita que los datos no sean solicitados por consola y estén directamente escritos en el código fuente, a fin de poder probarlos con facilidad con distintos valores.

B) “Utilizando tu programa de la parte A), escribe un programa que calcule la factorial de un número entero”.

C) “Escribe un programa en MIPS similar al de A) que calcule la división de dos enteros mediante la implementación de subrutinas. Al igual que en A), no se pueden utilizar instrucciones de multiplicación, división y desplazamiento, se debe implementar una técnica de división basada en otras operaciones matemáticas y el uso de subrutinas. Para el caso de divisiones no exactas (i.e., resto no nulo), el programa debe ser capaz de calcular hasta 2 decimales del cociente, sin atender a errores de precisión más allá del segundo decimal”.

1.2. Soluciones

Las soluciones de los problemas presentados en la sección anterior se encuentran programados en lenguaje ensamblador MIPS, los códigos fuente de las soluciones pueden encontrarse dentro de esta misma entrega de laboratorio con los nombres: “parte1.asm”, “parte2a.asm”, “parte2b.asm” y “parte2c.asm”.

En la sección de “Desarrollo de la solución” se explicará la lógica y el desarrollo de la solución a cada uno de los problemas presentados con más detalle.

1.3. Objetivos

Los objetivos específicos para esta experiencia de laboratorio son:

- Usar MARS (un IDE para MIPS) para escribir, ensamblar y depurar programas MIPS.
- Escribir programas MIPS incluyendo instrucciones aritméticas, de salto y memoria.
- Comprender el uso de subrutinas en MIPS, incluyendo el manejo del stack.
- Realizar llamadas de sistema en MIPS mediante “syscall”
- Implementar algoritmos en MIPS para resolver problemas matemáticos de baja complejidad

2. Marco Teórico

En esta sección se definirán distintos conceptos que se presentaran a lo largo del informe que se consideran relevantes para la comprensión del trabajo desarrollado:

- IDE: Integrated Development Environment, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- Lenguaje Ensamblador: Consiste en un conjunto de instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables.
- Instrucciones: Son el conjunto de funcionalidades que una unidad central de procesamiento puede entender y ejecutar
- MARS: Es un IDE que permite programar en MIPS
- MIPS: Son un conjunto de instrucciones que leen y escriben en memoria utilizando registros.
- Registro: Es una memoria de alta velocidad y poca capacidad, integrada en el microprocesador, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas.
- Subrutina: Porción de código que realiza una operación en base a un conjunto de valores dados como parámetros de forma independiente al resto del programa y que puede ser invocado desde cualquier lugar del código, incluso desde dentro de ella misma.
- Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problema.

3. Desarrollo de la solución

En esa sección se detallará el proceso y la lógica utilizada para resolver cada uno de los problemas planteados en esta experiencia de laboratorio.

3.1. Solución Parte 1

En el problema de la parte 1 se solicitó calcular el máximo entre dos enteros ingresados por el usuario, para ello se crearon las dos siguientes subrutinas:

```
determinarMaximo:
    bgt $a1, $a2, primerNumeroMaximo #¿Es a1 mayor a a2?
    #Si no se efectuo el salto, entonces $a2 es mayor o igual a $a0
    #Se traspasa el valor de $a2 al registro de resultados $v1
    add $v1, $a2, $zero
    #Se regresa al proceso principal
    jr $ra

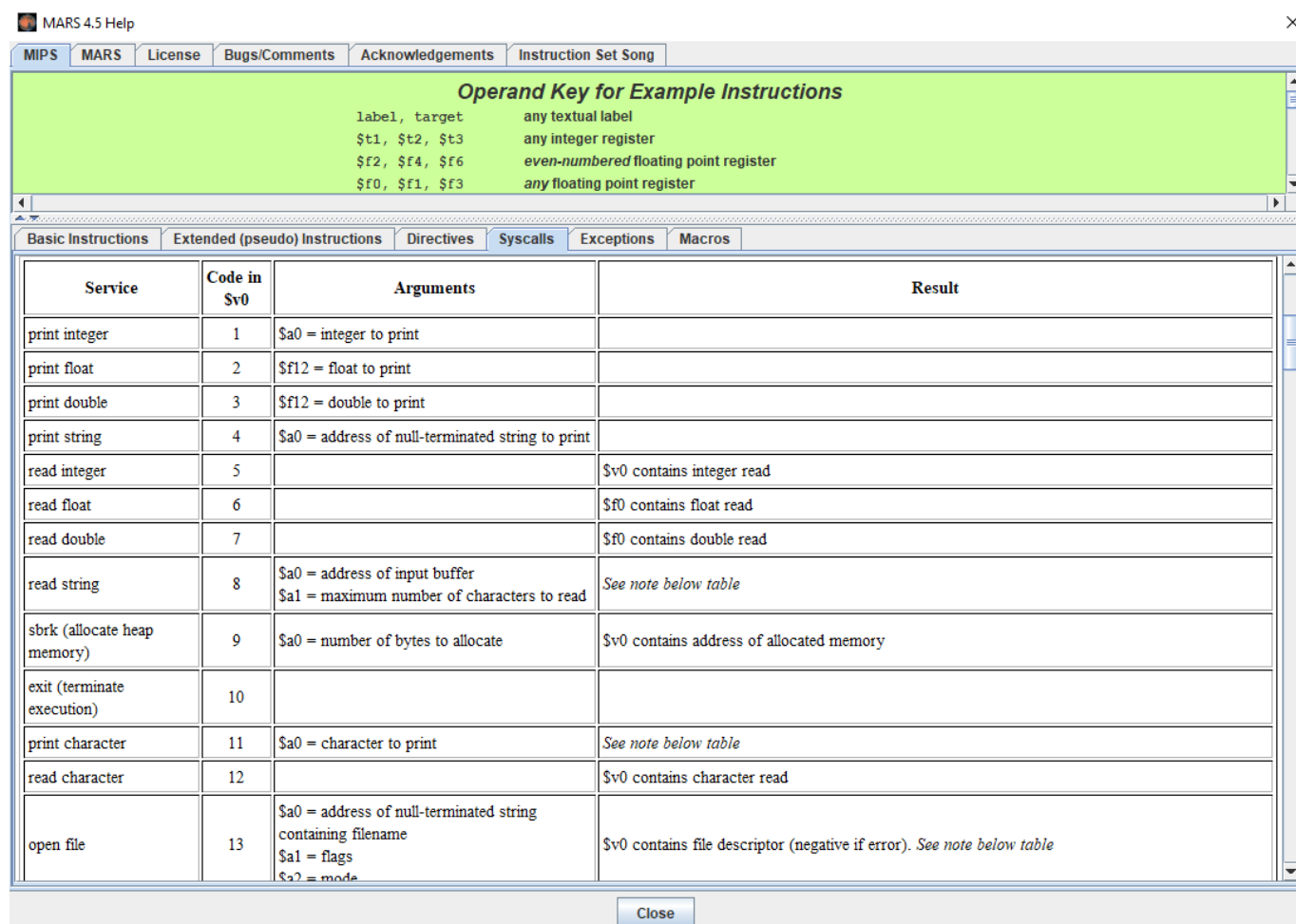
primerNumeroMaximo:
    #Se traspasa el valor de $a1 al registro de resultados $v1
    add $v1, $a1, $zero
    #Se regresa al proceso principal
    jr $ra
```

Figura 1: Subrutinas para determinar el máximo entre dos enteros

Los valores ingresados por el usuario se encuentran en los registros \$a1 y \$a2 antes de entrar a la subrutina “determinarMaximo”, una vez dentro de la subrutina se usa la instrucción bgt (Branch if Greater than) bifurcar si es más grande que, lo que se traduce a decir if $\$a1 > \$a2$, en caso que esa condición se cumpla y el valor del registro \$a1 sea mayor al de \$a2, se saltara a la subrutina “primerNumeroMaximo” en la cual se traspasara el valor del registro \$a1 al registro \$v1, luego se volverá al proceso principal. En caso de que la condición $\$a1 > \$a2$ no se cumpla, significa que el valor contenido en el registro \$a2 es mayor al del registro \$a1, por lo que se traspasara el valor de \$a2 al registro \$v1 y se volverá al registro principal.

Para cumplir con las condiciones solicitadas como: interactuar con el usuario por consola, permitirle la entrada de datos por teclado, mostrar el resultado por consola y terminar la

ejecución del programa, se hizo uso de las distintas funciones que ofrecen las llamadas de sistema o “syscall”, gracias a la documentación que se encuentra dentro del IDE MARS, se pudo determinar que las funciones a utilizar de syscall son la 1, 4 y 5.



The screenshot shows the MARS 4.5 Help window. At the top, there's a green box titled "Operand Key for Example Instructions" with the following content:

label, target	any textual label
\$t1, \$t2, \$t3	any integer register
\$f2, \$f4, \$f6	even-numbered floating point register
\$f0, \$f1, \$f3	any floating point register

Below this, there's a tabbed interface with tabs for "Basic Instructions", "Extended (pseudo) Instructions", "Directives", "Syscalls", "Exceptions", and "Macros". The "Syscalls" tab is selected, showing a table of system calls.

Service	Code in Sv0	Arguments	Result
print integer	1	\$a0 = integer to print	
print float	2	\$f12 = float to print	
print double	3	\$f12 = double to print	
print string	4	\$a0 = address of null-terminated string to print	
read integer	5		\$v0 contains integer read
read float	6		\$f0 contains float read
read double	7		\$f0 contains double read
read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	See note below table
sbrk (allocate heap memory)	9	\$a0 = number of bytes to allocate	\$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	\$a0 = character to print	See note below table
read character	12		\$v0 contains character read
open file	13	\$a0 = address of null-terminated string containing filename \$a1 = flags \$a2 = mode	\$v0 contains file descriptor (negative if error). See note below table

At the bottom of the window, there is a "Close" button.

Figura 2: Tabla de funciones que puede realizar syscall

El código 1 de syscall permite imprimir por consola un numero entero, en este caso se usó para imprimir el resultado final que era el máximo entre ambos enteros ingresados.

El código 4 permite imprimir un string por consola, el cual se usó para mostrar por consola el texto para indicarle al usuario que debía ingresar un número y para luego indicarle al usuario que se imprimiría el número máximo.

El código 5 permite que el usuario pueda ingresar por consola un numero entero, se uso en el programa para hacer que el usuario ingrese los dos operandos a los cuales se determinara cual es el máximo entre ambos. Para ingresar y salir de las subrutinas, se hizo uso de la

instrucción jal (Jump and link) el cual permite luego volver a la rutina principal mediante el uso de jr (Jump register), gracias a que la instrucción jal guarda la dirección del contador de programas en el registro \$ra.

3.2. Solución Parte 2

3.2.1. Solución Parte A)

En la parte A) de este problema se solicitó la creación de un programa en MIPS que sea capaz de multiplicar dos enteros mediante el uso de subrutinas y sin ocupar instrucciones de multiplicación, división y desplazamiento. Para lo cual se implementaron las siguientes dos subrutinas:

```
multiplicar:
    #Se verifica condicion de termino: si $t0 es igual al segundo operando se procede a escribir el resultado
    beq $t0, $a2, escribirResultado

    #En caso que aun se este multiplicando
    add $t1, $t1, $a1 # Se acumula la suma del primer operando en el registro $t1
    addi $t0, $t0, 1 #Se suma uno al valor actual de $t0, el que es usado como contador
    j multiplicar #Se vuelve al ciclo

escribirResultado:
    #Se traspasa el valor de $t1 a $v1
    add $v1, $t1, $zero
    #Se regresa al proceso principal
    jr $ra
```

Figura 3: Subrutinas que efectúan multiplicación

Teniendo los dos operandos de la multiplicación en los registros \$a1 y \$a2 se usa el registro \$t0 para usarlo como contador de veces que se ha realizado el ciclo, el ciclo suma el primer operando la misma cantidad de veces del segundo operando, acumulando el resultado de la suma en el registro \$t1, por ejemplo si se ingresaran los operandos 2 y 3, se efectuaría la multiplicación de 2x3 sumando tres veces el numero dos, o sea $2 + 2 + 2 = 6$ y ese es el resultado de la multiplicación que luego se guarda en el registro \$v1.

El programa también es capaz de efectuar multiplicaciones para números enteros negativos, para ello antes de entrar a la subrutina que efectúa el proceso de multiplicación antes descrito verifica si alguno de sus operandos es negativo y en caso de serlo, los convierte a positivo

y se registra que fue convertido, luego antes de mostrar por pantalla el resultado de la multiplicación, se verifica cuantos operandos fueron convertidos a positivo, en caso que haya sido 1 el resultado se convertirá a negativo, en caso de que 0 o los 2 operandos hayan sido convertidos a positivo, no se convertirá el resultado, debido a que la multiplicación de dos negativos es positiva y la multiplicación de dos positivos es positiva. Este proceso ocurre en 8 subrutinas que se encuentran disponibles dentro del código fuente.

3.2.2. Solución Parte B)

Para realizar el cálculo de la factorial solicitada en la parte B), se utilizó la siguiente subrutina:

```

calcularFactorial:
    #Se verifica la condicion de termino: si $t2 es igual al numero cuyo factorial esta siendo calculado se procede a escribir el resultado
    beq $t2, $a3, escribirResultadoFactorial

    #Si aun no acaba el ciclo, se calcula la multiplicacion del valor actual de $t2 con la acumulacion
    add $a1, $t3, $zero #Se traspasa el valor acumulado en $t3 al registro $a1 ahora es el primer operando de la multiplicacion
    add $a2, $t2, $zero #Se traspasa el valor actual de $t2 a $a2, sera el segundo operando de la multiplicacion
    add $t0, $zero, $zero #Se fija el registro $t0 al valor cero
    add $t1, $zero, $zero #Se fija el registro $t1 al valor cero

    # Se respalda el valor del registro $ra para poder volver a donde se llamó a "calcularFactorial"
    addi $sp, $sp, -4 # Se da espacio en el stack
    sw $ra, 0($sp) # Se guarda el valor de $ra en el stack

    jal multiplicar #Se efectua la multiplicacion en una subrutina

    # Se restablece el valor del registro $ra
    lw $ra, 0($sp) # Se carga el valor del registro $ra anteriormente almacenado
    addi $sp, $sp, 4 # Se restablece el valor de $sp (puntero)

    #Luego de efectuar la multiplicacion se actualizan los acumuladores
    add $t3, $t3, $v1 # $t3 contiene la suma de las multiplicaciones
    addi $t2, $t2, 1
    j calcularFactorial

```

Figura 4: Subrutina que calcula la factorial de un número

La subrutina empieza con el valor 1 en el registro \$t3 y con el valor 0 en el registro \$t2, el registro \$t2 avanza en uno por cada ciclo y por cada ciclo se efectúa la multiplicación del valor de \$t3 al de \$t2 y el resultado se almacena en el registro \$t3, el ciclo para cuando \$t2 tenga el mismo valor que el número al cual se está calculando su factorial, de manera que por ejemplo si se calcula la factorial de 4 la operación resultante es: $1 \times 2 \times 3 \times 4 = 24$. Debido a que la factorial de un numero negativo no esta definida, el programa solo calcula factoriales para números enteros positivos. Se hace uso del stack pointer para guardar los valores del registro \$ra y restaurarlos cuando sea necesario de forma de poder saltar a las etiquetas necesarias

mediante el uso de jal.

3.2.3. Solución Parte C)

En la parte C) se solicita el realizar un programa que realice división entre dos números enteros tomando en consideración una precisión de dos decimales y al igual que la parte A) esta prohibido el uso de instrucciones de multiplicación, división y desplazamiento, por lo que de una manera muy similar a la solución de la parte A) se realizo algo similar: En primer lugar se convierten los operandos a positivos en caso de ser negativos, luego con la misma lógica de la parte A) se resta al primer operando el segundo operando y se registra la cantidad de veces que se le resta hasta que el primer operando sea menor al segundo operando, por ejemplo $20/5$ el 20 es el primer operando, el cual dentro del ciclo iría variando así: $15 \rightarrow 10 \rightarrow 5 \rightarrow 0$ un total de 4 restas efectuadas, que es justamente el resultado de la división. En caso de que quede un resto, se efectuara el calculo de los dos decimales, para esto se multiplicara el resto por 10, para ello se utilizaran las subrutinas del programa A) que multiplican mediante sumas, luego ese $\text{resto} \times 10$ volverá a entrar al subproceso de división antes descrito, con el cual se calculara el primer decimal, en caso que aún siga existiendo resto, se calculara el segundo decimal con el mismo proceso descrito, por ejemplo la división $1/2$ se efectuaría de la siguiente manera: $1/2 = 0 \rightarrow \text{resto} = 1$ el cual se multiplica $\times 10 \rightarrow 1 \times 10 = 10$ luego se divide por 2 $\rightarrow 10/2 = 5$ resultado del primer decimal, no queda resto, entonces el resultado de la división es 0.50. Este programa implementa el uso del stack pointer al igual que el programa de B) para almacenar los valores de \$ra y mantener el flujo del programa de forma correcta, además también posee los subprocesos del programa A) para poder convertir números positivos a negativos y poder dividir números negativos, además de las subrutinas que permiten multiplicar que acá son usadas para calcular los dos decimales en caso de divisiones no exactas.

4. Resultados

En esta sección se presentarán los resultados de cada problema.

4.1. Resultados Parte 1

Para todos los casos probados, ya sea multiplicación de números positivos o negativos el programa cumple su función sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

4.2. Resultados Parte 2

4.2.1. Resultados Parte A)

Para todos los casos probados, el programa cumple su función sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

4.2.2. Resultados Parte B)

Para todos los casos probados, el programa cumple su función calculando la factorial del número sin incurrir en errores, por lo que se concluye que cumple su función a cabalidad.

4.2.3. Resultados Parte C)

Para todos los casos probados, el programa cumple su función calculando la división entre los dos operandos sin incurrir en errores, tanto para números positivos como negativos, calculando con precisión los primeros dos decimales cuando la división no es entera, por lo que se concluye que cumple su función a cabalidad.

5. Conclusiones

Es posible concluir que esta experiencia de laboratorio fue satisfactoria, debido a que se cumplieron todos los objetivos planteados inicialmente:

- Se utilizó MARS para escribir, ensamblar y depurar cuatro programas MIPS.
- Se escribieron programas MIPS que incluían instrucciones aritméticas, de salto y uso de memoria.
- Se comprendió el uso de subrutinas en MIPS, siendo estas utilizadas en todos los programas de esta experiencia de laboratorio, además se hizo un manejo del stack en algunos programas para almacenar valores del registro \$ra.
- Se realizaron llamadas del sistema mediante “syscall” que permitieron interactuar con el usuario o mostrar los resultados de una manera más comprensible por consola.
- Se implementaron algoritmos en MIPS que resolvían problemas matemáticos de baja complejidad, como multiplicar, calcular una factorial o la división de números sin ocupar operaciones de multiplicación, división o desplazamiento.

Para finalizar, esta experiencia de laboratorio fue muy útil para interiorizarse en la programación en lenguaje ensamblador en el IDE MARS, permitiendo entender y aplicar conceptos como las llamadas de sistema, uso del stack pointer y su asignación de memoria, saltos, bifurcaciones, entre otros.