Universidad de Santiago de Chile

Facultad de Ingeniería

Departamento de Ingeniería Informática

Laboratorio 2 acercándose al Hardware: Programación en Lenguaje Ensamblador

Alumno: Ignacio Villarroel E.

Profesor: Carlos González C.

Asignatura: Organización de computadores

Sección: L – 3

Fecha de entrega: 18/12/2021

Tabla de contenidos

1. Introducción

Al igual que muchos lenguajes de programación, Mips tiene una gran variedad en las formas de cómo interpretar y escribir un código, sin embargo, la interpretación del ensamblador al ejecutarse no es siempre muy cómoda, debido a que Mips tiene un desarrollo de instrucciones lineal, es decir en simples palabras, va de corrido hacia abajo, es por eso que se estudiará y aplicara el uso de subrutinas. Estas se utilizarán para el desarrollo de problemáticas de baja complejidad, que luego se estudiarán en su implementación para notar que cosas pueden mejorarse.

1.1 Objetivos

Para el desarrollo óptimo de la actividad, se plantean los siguientes objetivos a realizarse

- Utilizar Mips en desarrollos aritméticos, de salto y memoria para la implementación en códigos.
- Comprender y ejecutar el uso de subrutinas.
- Utilización de syscall para llamadas necesarias.
- Implementación de algoritmos para resolver problemas.

2. Marco teórico

En el ámbito global general de Mips, está el uso de registros, debido a que toda acción realizada, hace referencia al uso de los datos contenidos en estos, es por ello, que en el desarrollo de la implementación de códigos, se usan números de carácter entero “int” los cuales irán directamente en los registros de Mips, mientras que el uso de números de coma flotante (float) irán en el co-procesador, los cuales pueden ser de precisión simple (single) o de doble precisión (double), estos últimos utilizan 2 registros, puesto que su tamaño es el equivalente a dos words (8 bits).

Los saltos, corresponden a la base para el libre desarrollo de subrutinas, puesto que permiten avanzar o devolverse a ciertas partes del código por medio de la mención de una etiqueta, la cual es una subrutina, estas últimas corresponden a espacios, donde se necesita que se desarrolle cierta parte del código. Gracias a estos, es posible la incorporación de algoritmos

recursivos, puesto que en los códigos presentados representan gran parte del uso para poder encontrar la solución al problema.

El uso de syscall es muy diverso, aunque sea solo una palabra, acompañada del número de la opción que quieres realizar se puede pedir un número, hasta mostrar por pantalla un mensaje para que el usuario esté al tanto de lo que va ocurriendo con el programa.

3. Desarrollo y resultados

En primer lugar se destaca que todos los códigos fueron realizados exitosamente, a continuación se para a detallar el desarrollo de cada uno de ellos.

3.1 Usos syscall

El objetivo de este programa es encontrar la diferencia entre dos números. Por medio del uso syscall para el llamado de mensajes y para adjudicar en la sección de registros de los valores otorgados por el usuario, luego se crea una subrutina, donde se verá que número es mayor para luego proceder a realizar la resta para obtener la diferencia, luego de esto, se buscar para ver si es par o impar, esto a través de una división por dos, que luego con la ayuda de la función “mfhi” permitirá tener la información del ultimo bit si termina en 0 o 1, de esta forma se podrá saber si corresponde a un número par o a uno impar, por lo que dependiendo el resultado se saltará a la subrutina llamada: *par* o *impar* para colocar el mensaje por pantalla. Sobre los números a implementar, se pueden todos los números que puedan caber en una Word (4 bytes) ya que representan a los números enteros, por lo que no muestra ningún problema.

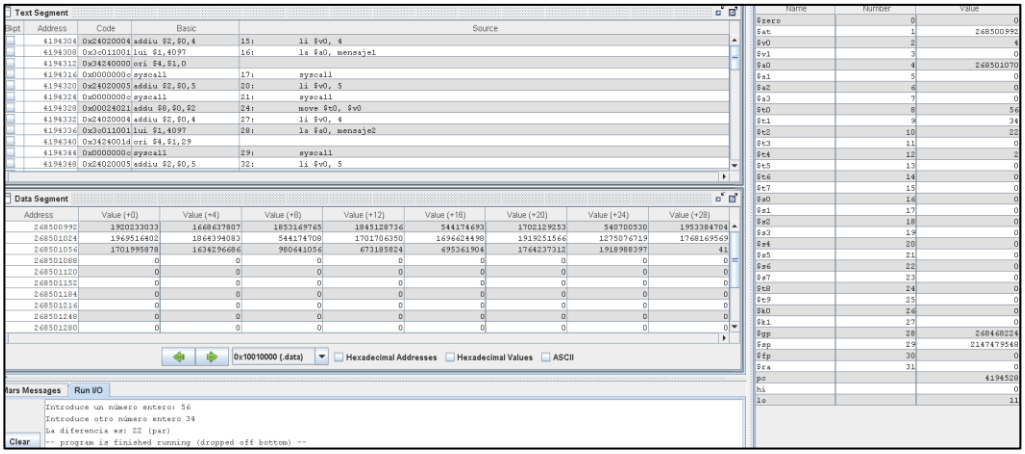
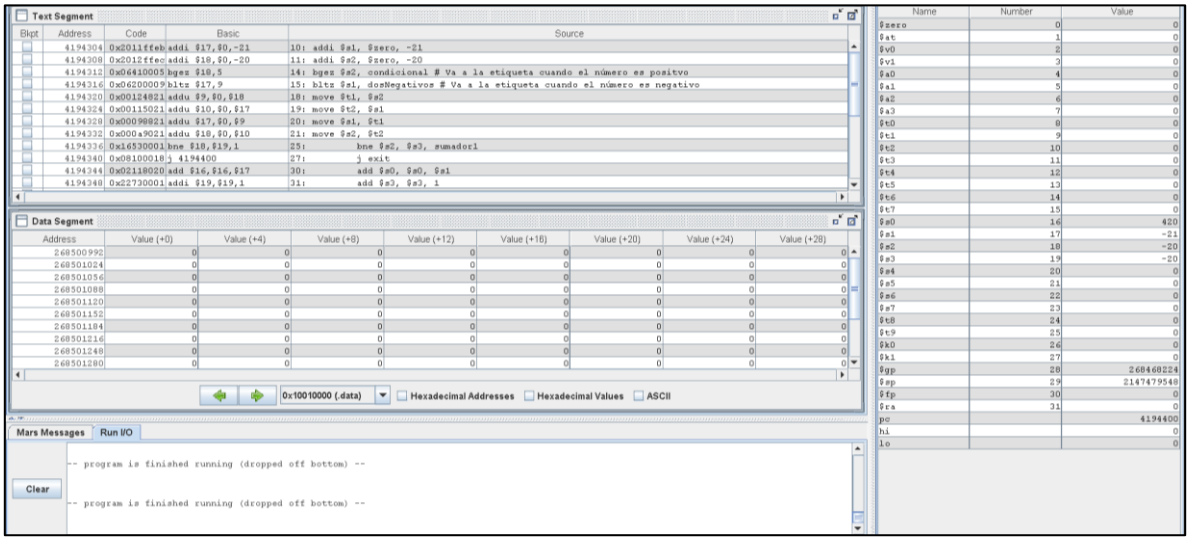


Figura 1. Ejemplo uso de syscall

3.2 Multiplicación

Se destaca que al utilizar los datos en duro, los registros que deben cambiarse para la utilización rápida del programa son los “$s1” y “$s2”, que están al principio del código, además que el resultado final de la multiplicación se encontrará en “$s0” en la sección de registros.

La complicación más difícil de esta parte fue la incorporación para los números negativos como resultado cuando el segundo factor era el negativo, puesto que el sistema consiste en



que por medio de un acumulador “$s3” que se irá comparando con el “$s2” (segundo factor) hasta que se iguales, por cada vez que no sea igual se sumará a si mismo lo que estará en “$s0 con “$s1” y a su ves el acumulador se le sumaria 1, el programa finalizaría hasta que el acumulador con el segundo factor fueran iguales, esta idea se podía implementar para cuando ambos factores eran positivos o negativos, con la diferencia que en la de los negativos, el resultado se traspasaba a positivo, por medio de un conteo que consiste en copiar el registro del resultado en otro lado “s4” y crear una recursión que vaya sumando 2 al resultado en “$s0” y un 1 al registro copia, hasta que este último llegue a cero.

Figura 2. Ejemplo de multiplicación -21 x -20

Como se mencionó en el párrafo anterior, para cuando un número (cualquiera de los dos) fuera negativo, era estrictamente necesario que este estuviera en el primer registro, pues se evitaba el problema de igualdad con el acumulador para la comparativa con el segunda factor, es por esto que cuando esto ocurriera, al principio se hizo una etapa de evaluación para saber los signos de los números, para que cuando el segundo factor fuera negativo y el primero positivo, se intercambiaran las posiciones, de esta forma se ahorraba código permitiendo re usar la idea de multiplicación hecha.

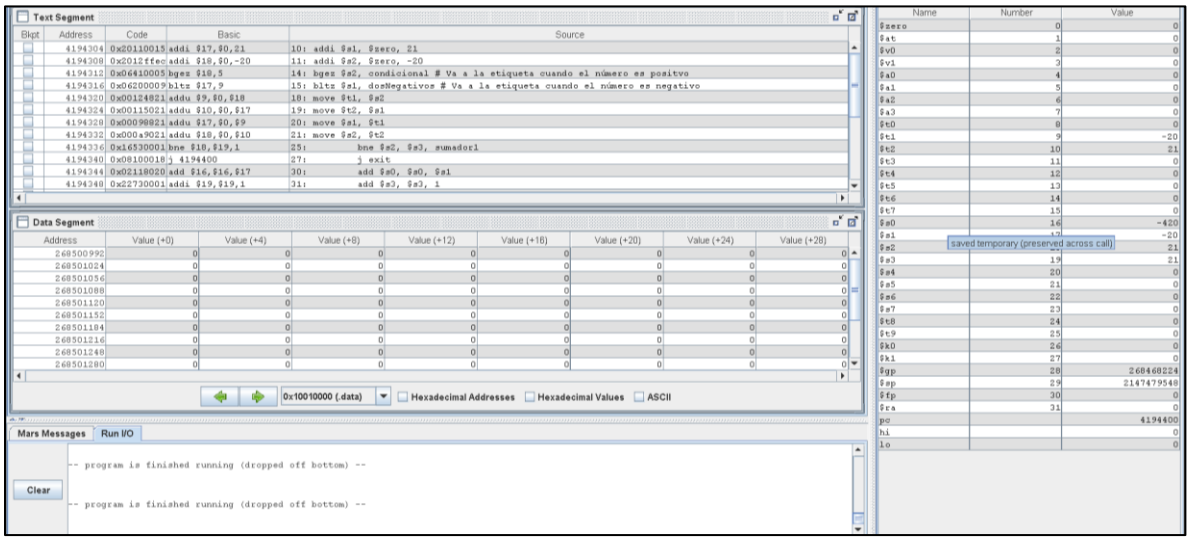


Figura 3. Ejemplo de multiplicación 21 x -20

3.3 Factorial

Para este problema se reutilizó la idea del ejercicio anterior, con la simplificación solamente se realizará para números mayores o iguales que cero, esto porque el factorial no tiene desarrollo para números negativos. El resultado final estará en el registro "$s1", además que también se colocará el número a calcularle el factorial en esa posición. Luego en el siguiente registro "$s2" se colocará el antecesor del número en el registro superior, puesto que el programa multiplicará el resultado, los guardará en "$s0" utilizando el acumulador en "$s4", sin embargo, cuando terminé existirá la subrutina "regulador" que reordenará dejando el resultado de la multiplicación inicial en "$s1" y se resta una unidad a los alojado en el segundo factor, además de resetear el acumulador y eliminar el resultado final en "$s0". Se repite este ciclo hasta que el segundo factor llegue a 0, mostrando por pantalla el resultado. Como detalle final, hay 2 excepciones para cero y uno, puesto que el resultado de estos, es 1, se salta todo el proceso para mostrarlos por pantalla.

En el ensamblador vemos que la muestra del resultado no tiene problemas hasta el número doce (12), mas allá, se calcula el resultado con el factorial con éxito, con la diferencia que se muestra por pantalla lo siguiente: "*Go: execution terminated with errors."*, se infiere que el resultado es demasiado grande por lo que el programa sufre deficiencias ya que el número podría no caber en una word por su tamaño.

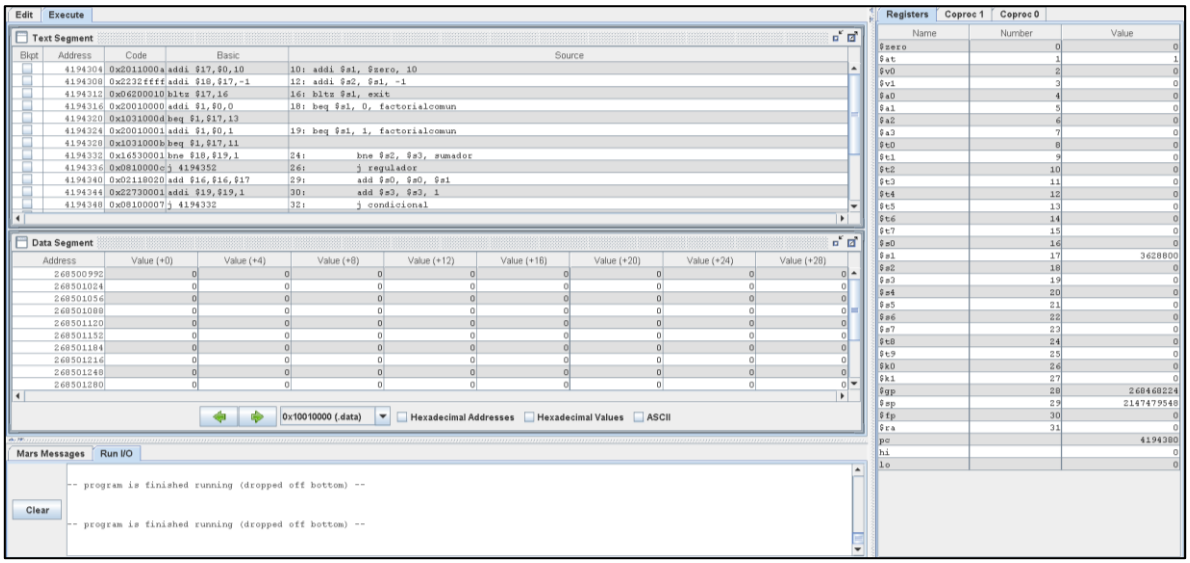


Figura 4. Ejemplo factorial de 10 (10!)

3.4 División

Es la mas extensa y complicada, utilizando reutilización del código de la multiplicación, el código se basó en las sumas de números de coma flotante de doble precisión (double), en primer lugar se pasaban los número a positivos para efectos prácticos a la hora de la división, luego de esto se condicionaba el dividendo hasta que fuera menor que el divisor, hasta que eso sucediera, se le restaba al dividendo una vez lo contenido en el divisor, a su vez se sumaba un 1.00 al registro del coprocesador 1 ("$f0") y se registraba en "$t3" cada vez que la acción se repetía (+ 1) , hasta que divisor fuera mayor, se verificaba si había resto, si no lo hay, pasa a una etapa para ver su signo final, si lo hay, se procede a multiplicar ese resto por 10, es decir, se le agrega un 0 al número y se repite el mismo proceso que el anterior, con la diferencia que en el registro del coprocesador 1 se suma un 0.1, y la cantidad de veces que esta ejecución se realice se sumará 1 en el registro "$t4", nuevamente, si no hay resto, pasa a la etapa para ver su signo, por el contrario, si hay resto, se multiplica por 10 y este resultado se dividirá nuevamente por el divisor, haciendo que al registro del coprocesador 1 se sume

un 0.01 y que la cantidad de veces que se haga esto quede registrada en el “$t5”, ahora si o si, pasamos a la etapa de obtención de signo, que se encarga de comparar analizar los datos del dividendo y divisor que se copiaron en “$t1” y “$t2”, con esto se incorporan reemplazando los datos que se sumaron en el coprocesador, pasando a usar 2.00 , 0.20 y 0.02, si los 2 números son positivos o negativos, no se realiza acción alguna, dejando el resultado intacto, por el contrario, si existe una disparidad de signo, el resultado pasará a negativo, descontando 2.00 al entero por cada vez que se reste 1 al registro “$t3”, que recordemos, era la cantidad de operaciones que se usó en el número entero, hasta que este registro llega a 0. Igualmente, con 0.2, que afectará al resultado tantas veces, haste que el “$t4” llegue a 0 y por último con 0.02 que hará lo mismo con la diferencia que será hasta que lo contenido en el “$t5” llego al número nulo, llegando al fin del programa.

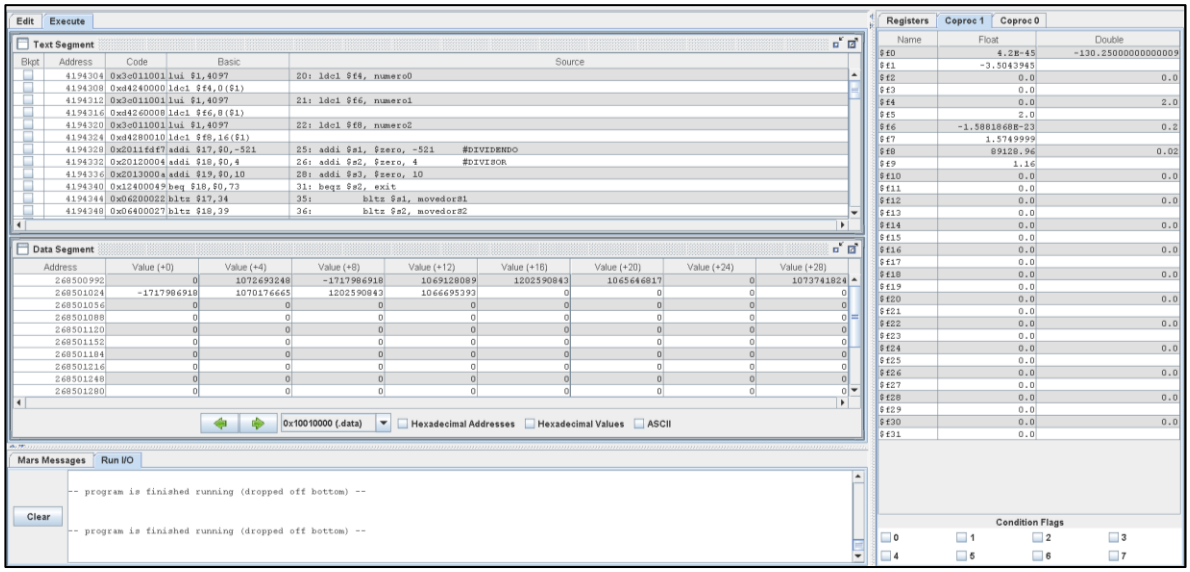


Figura 5. Ejemplo de división: -521 / 4

4. Conclusiones

Los objetivos se cumplieron totalmente en lo realizado, puesto que se dominó cada punto de los temas especificados, además de cubrir todos los casos posibles que hayan existido en cada problema que se propuso del laboratorio, sin embargo, el último punto se no se realizó puesto a que no hubo una idea de como interpretar en código las series de Taylor.

Pese a que las realización de los códigos fue de gran éxito, eso no implica a que las implementaciones fueran las mejores, puesto que al estar recién conociendo el lenguaje ensamblador, alguno de los códigos puede tener creaciones innecesarias, que solo crean demoras dentro de este mismo haciendo menos eficiente el código creado.

5. Referencia

- Amell Peralta. (2014, 26 diciembre). *MIPS Tutorial 7 Printing a Double* [Vídeo]. YouTube. https://www.youtube.com/watch?v=HpxHhCM_gP0&t=1s
- Amell Peralta. (2014a, diciembre 26). *MIPS Tutorial 6 Printing a Float* [Vídeo]. YouTube. https://www.youtube.com/watch?v=m_iQTl9lbQE
- Serrano Sanchez de Leon, A. (2006). *Programación en Ensablador MIPS*. Universidad Rey Juan Carlos. https://www.cartagena99.com/recursos/electronica/apuntes/tema12_ensamblador_MIPS.pdf