Arquitectura de Computadores - Laboratorio 1

Sebastián Cassone González (21.286.604-0)
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
sebastian.gonzalez.cass@usach.cl

I. Introducción

En la reciente experiencia de laboratorio, se presenta el problema de un grupo de arqueólogos descubren una tumba antigua que contenía inscripciones extrañas. Luego, se descubre que dichas inscripciones eran en realidad operaciones aritméticas como la suma, resta, multiplicación y división.
Para esto se pide que se desarrolle un código en ensamblador para la arquitectura Mips para resolver dichas operaciones. La solución considera que dichos números son operados de manera inversa, que hace más complejo la operación entre los números.
Los objetivos de esta experiencia estuvieron enfocadas principalmente aprender programar en ensamblador para la arquitectura Mips, además de resolver el problema presentado con sus limitaciones en caso de que así lo requiera.

II. Marco Teórico

Debido a las limitaciones expresadas en el enunciado del Laboratorio 1, se debió utilizar las definiciones básicas de multiplicación y división.

II-A. Ecuaciones

La ecuación utilizada y pensada para la solución fue una sumatorio de un i hasta un n, siendo este i el primer número a multiplicar y n el segundo número.

$$m = \sum_{i=1}^{n} x \tag{1}$$

II-B. Algoritmos

Los algoritmos que se diseñaron para obtener la suma en el formato escrito en el problema el presentado en el algoritmo 1 en pseudocódigo.

III. Desarrollo de la Solución

La solución está enfocada en primero en almacenar los números que se iban a operar en la memoria, luego de esto se le pide al usuario que seleccione una operación (+, -, *, /). Para la suma se hizo un algortimo presentado en algoritmo 1, esa misma lógica para hacer sumas y restas. Para la multiplicación se utilizó la ecuación para calcular la multiplicación, primero obtiene los números enteros, los multiplica y luego los vuelve a separar. Finalmente se almacena el resultado en el segmento de datos específico para eso pedido en el enunciado del Laboratorio y se muetra por consola el resultado.

III-A. Limitaciones

Para la resta, es necesario modificar manualmente la variable signo que está en .data, reemplazando el signo más por el menos.
Si bien, el algoritmo de multiplicación está correcto, se crashea.
También, no se logró desarrollar una división por su alta complejidad. Al limitarse el uso de mul o div, complica muchisimo una solución eficiente y sencilla.

IV. Resultados

Los resultados en la suma y en la resta fueron los esperados aunque con algunos resultados tiene un problema con el acarreo.
Sin embargo, con la multiplicación y la división no fueron satisfactorias debido a que por el lenguaje ensamblador y para el problema en si mismo era más dificil implementarlas en ensamblador.

V. Conclusiones

Luego de haber hecho este trabajo, se logró profundizar mucho más en como funciona el lenguaje ensamblador en la arquitectura Mips.
No obstante, los resultados quedaron al debe debido a las limitaciones del enunciado. Se segurá profundizando en este lenguaje debido a lo atractivo que es en el uso eficiente de la memoria.
Bajo mi opinión, este problema si hubiera sido un problema real, no debería haber sido necesario tanta gestión de memoria, pudiendo resolverse el problema con algún lenguaje de programación de alto nivel como C o Golang y seguramente la diferencia de tiempo no hubiera sido mucha o hasta imperceptible.

Algoritmo 1 Suma de listas

```
function SUMA(lista1, lista2, signos, acarreos = [0, 0, 0, 0])
    resultado ← []
    for i in 0 to len(lista1) − 1 do
        if signos[0] == " + " then
            if lista1[i] + lista2[i] ≥ 10 then
                if i + 1 == len(lista1) then
                    resultado.append(lista1[i] + lista2[i])
                else
                    resultado.append(lista1[i] + lista2[i] − 10)
                    acarreos[i + 1] += 1
                end if
            else
                resultado.append(lista1[i] + lista2[i] + acarreos[i])
            end if
        else if signos[0] == " − " and signos[1] == " + " then
            if lista2[i] − lista1[i] ≥ 10 then
                if i + 1 == len(lista1) then
                    resultado.append(lista2[i] − lista1[i])
                else
                    resultado.append(lista2[i] − lista1[i] − 10)
                    acarreos[i + 1] += 1
                end if
            else if lista2[i] − lista1[i] < 0 then
                resultado.append(lista2[i] − lista1[i] + 10)
            else
                resultado.append(lista2[i] − lista1[i] + acarreos[i])
            end if
        else if signos[1] == " − " and signos[0] == " + " then
            if lista1[i] − lista2[i] ≥ 10 then
                if i + 1 == len(lista1) then
                    resultado.append(lista1[i] − lista2[i])
                else
                    resultado.append(lista1[i] − lista2[i] − 10)
                    acarreos[i + 1] += 1
                end if
            else if lista1[i] − lista2[i] < 0 then
                resultado.append(lista1[i] − lista2[i] + 10)
            else
                resultado.append(lista1[i] − lista2[i] + acarreos[i])
            end if
        end if
    end for
    return resultado
end function
```