

LABORATORIO 1

Organización de Computadores

Nombre: Leandro Pizarro
Profesor: Leonel Medina
Ayudante: Ricardo Álvarez
Fecha de Entrega: 21-04-2018

Santiago de Chile

1 - 2018

TABLA DE CONTENIDO

Capítulo 1.	Introducción.....	1
1.1	Enunciado del problema.....	1
1.2	Motivación	1
1.3	Objetivos	1
1.3.1	Objetivo general	1
1.3.2	Objetivos específicos	1
1.4	Herramientas	2
1.5	Estructura del informe.....	2
Capítulo 2.	Marco teórico.....	3
Capítulo 3.	Desarrollo	5
3.1	Descripción de la solución	5
3.2	Algoritmos y estructuras de datos	5
3.2.1	Estructura utilizada: Registro	5
3.2.2	Funciones y procedimientos más relevantes	6
Capítulo 4.	Experimentos a realizar	7
4.1	Resultados	7
4.2	Análisis de resultados.....	8
Capítulo 5.	Conclusiones.....	9
Capítulo 6.	Referencias	11

ÍNDICE DE FIGURAS

Ilustración 4.1 Archivo con instrucciones prueba.....	7
Ilustración 4.2 Archivo líneas de control prueba	7
Ilustración 4.3 Archivo salida registros prueba	8

CAPÍTULO 1. INTRODUCCIÓN

1.1 ENUNCIADO DEL PROBLEMA

El problema en esta ocasión es el siguiente: Una vez proporcionados dos archivos, uno con las instrucciones de un programa en MIPS y otro con el listado de línea de control, las cuales pueden presentar problemas de *stuck at zero*, se deben entregar otros dos archivos que contienen a su vez la traza del programa y el estado de los registros para cada instrucción. Para ello es debido el desarrollo de un programa bajo el lenguaje de programación C que permita la lectura de los archivos, emule el funcionamiento de un procesador, los 32 registros y la memoria ram y escriba los archivos de salida.

1.2 MOTIVACIÓN

Trabajar en un alto nivel de abstracción permite desentenderse en cierta manera del funcionamiento de niveles inferiores y entenderlos como una especie de caja negra. Lo anterior se ve reflejado a la hora de diseñar software independiente de cómo está implementado el hardware que llevará a cabo cada una de las instrucciones de un determinado programa. Si bien esto facilita el proceso de creación ya que no es necesario pensar en cómo los componentes de la máquina interactúan entre sí al momento de programar; se vuelve una ventaja comparativa en computación e informática el conocer cómo software y hardware trabajan en conjunto y más específicamente como este último ejecuta las instrucciones de un programa, ya que permite el desarrollo de programas más robustos, optimizados y que aprovechen al máximo el procesador.

1.3 OBJETIVOS

1.3.1 Objetivo general

Desarrollar un programa capaz de simular la ejecución de un programa en MIPS

1.3.2 Objetivos específicos

Entender la importancia de las líneas de control en el camino de datos y cómo un error en estas puede alterar drásticamente el resultado de una instrucción.

1.4 HERRAMIENTAS

Para dar solución al problema y tal como se señala anteriormente, se ha de utilizar C como lenguaje de programación, el cual provee de estructuras, punteros y manejo de memoria en tiempo de ejecución.

También que se incluirán diferentes librerías que facilitarán el trabajo enormemente, además de `stdio.h` y `stdlib.h` cuyo uso resulta obligatorio, como lo es `string.h`, la que se utilizará para comparar cadenas de caracteres de manera correcta y poco tediosa.

1.5 ESTRUCTURA DEL INFORME

El presente escrito presenta a continuación conceptos que fueron la base en la que se ergió la solución al problema, tales como punteros, memoria dinámica, registros, entre otros, con el fin exponer el material investigado, la descripción de la solución y los resultados obtenidos con su respectivo análisis y conclusiones.

CAPÍTULO 2. MARCO TEÓRICO

MIPS (procesador): Con el nombre de MIPS se le conoce a la familia de microprocesadores de arquitectura RISC desarrollados por MIPS Thecnologies, diseñada para optimizar la segmentación en unidades de control y para facilitar la generación automática de código máquina por parte de los compiladores, sus diseños son utilizados productos informáticos de Silicon Graphics, consolas como la Nintendo 64 y Playstation y dispositivos para Windows CE. Un microprocesador sin bloqueos en las etapas de segmentación (MIPS) posee un repertorio de instrucciones, registros, modelo de excepciones, manejo de memoria virtual, mapa de direcciones físicas y otras características comunes.

Registro: Un registro corresponde al hardware que forma parte de un procesador y puede contener una determinada cantidad de bits. Estos ofrecen un nivel de memoria más rápido y acotado que la memoria principal, son empleados para controlar instrucciones en ejecución, manejar el direccionamiento de memoria y proporcionar capacidad aritmética.

Puntero: “Un puntero es una variable que contiene una dirección de memoria de otra. Los punteros se utilizan mucho en C, en parte debido a ellos son en ocasiones la única forma de expresar operaciones y a que por lo general llevan un código más compacto y eficiente de lo que se puede obtener en otras formas” (Kernighan, Ritchie, 1988, p.103). Cuando un puntero es declarado se reserva memoria para albergar una dirección de memoria, pero no para la información o valor a la que “apunta” el puntero. El espacio de memoria que se reserva para almacenar un puntero depende de la arquitectura del sistema operativo del tipo de dato al que apunte.

Memoria Dinámica: Las variables definidas de forma estática no pueden variar su tamaño durante la ejecución del programa, en cambio aquellas definidas de forma dinámica pueden modificar la cantidad de memoria que utilizan dependiendo de lo que se requiera a lo largo de la ejecución, incluso liberarla si es necesario; conllevando así a un programa más eficiente en términos de la memoria que utiliza.

CAPÍTULO 3. DESARROLLO

3.1 DESCRIPCIÓN DE LA SOLUCIÓN

Usar C como lenguaje de programación ofrece la posibilidad de subdividir el problema y encapsular cada una de las partes en un determinado proceso. Es así como el problema se separa de la siguiente manera:

1. Lectura de archivo con instrucciones y posterior inserción en arreglo
2. Lectura de archivo con líneas de control y posterior inserción en arreglo (solo aquellas que presenten error *stuck at zero*)
3. Lectura, ejecución de instrucciones y escritura de archivos de salida
 - 3.1. Actualizar la traza en el archivo de salida
 - 3.2. Identificar y ejecutar la
 - 3.3. Actualizar el estado de los registros para el ciclo actual en el archivo de salida
 - 3.4. Saltar a la siguiente instrucción

Cabe destacar que la memoria ram se ha de simular mediante una matriz de números enteros y no requerirá de una estructura especial a diferencia de los registros. La posición en la matriz a la que accederán las instrucciones *lw* y *sw* será de la siguiente manera:

1. `lw $t1, 8($t2) ⇒ $t1 = matriz[$t2][8/4]`
2. `sw $t1, 8($t2) ⇒ matriz[$t2][8/4]`

3.2 ALGORITMOS Y ESTRUCTURAS DE DATOS

3.2.1 Estructura utilizada: Registro

La estructura registro ha sido creada de tal manera que permita almacenar un valor numérico y su respectivo nombre dentro de un string (puntero a caracter). Esto facilitará identificarlos al momento de ejecutar una determinada instrucción por medio de la función `strcmp()` que proporciona la librería `string.h`.

3.2.2 Funciones y procedimientos más relevantes

3.2.2.1 *buscar_etiqueta*

Esta función recibe un arreglo de strings que contiene las instrucciones del programa, un token con la etiqueta a buscar dentro de las estas y un número entero que representa el largo del arreglo de instrucciones. Mediante una iteración determina en qué posición de dicho arreglo se encuentra la etiqueta. Esto permite al programa determinar a qué instrucción debe saltar luego de la instrucción *Jump* o *Branch on Equal*.

3.2.2.2 *validar_control*

La función *validar_control* recibe un arreglo con las líneas de control que presentan el error de stuck at zero , una línea de control que se quiere validar y el largo del arreglo de líneas de control en un entero. Al igual que *buscar_etiqueta* realiza una iteración comprobando si la línea de control ingresada se encuentra dentro del arreglo. Retorna 0 de ser falso o 1 de lo contrario. Esto facilita la identificación de aquellas instrucciones que aunque sean ejecutadas su resultado dentro de los registros no es efectivo.

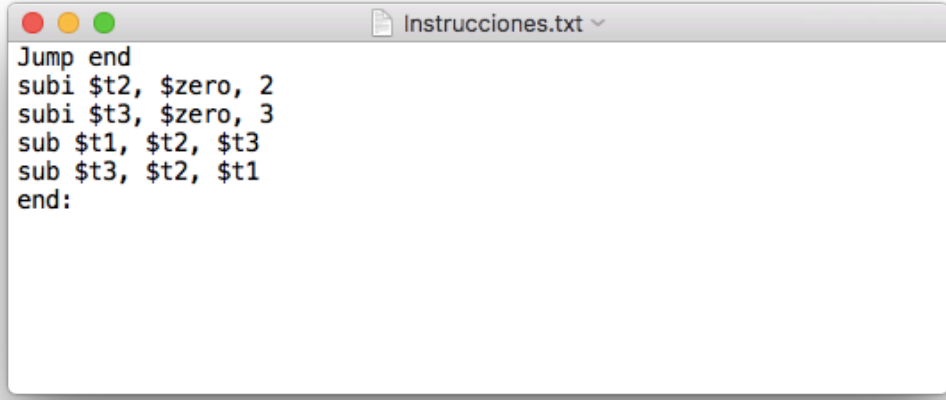
3.2.2.3 *contador_de_programa*

Este procedimiento por una parte simula al *Program Counter*, registro dentro del procesador encargado de indicar en qué posición de la secuencia de instrucciones se encuentra. Además, realiza cada una de las instrucciones que el programa debe soportar, las cuales son: *add*, *addi*, *sub*, *subbi*, *mul*, *div*, *beq*, *lw* , *sw* y *jump*.

CAPÍTULO 4. EXPERIMENTOS A REALIZAR

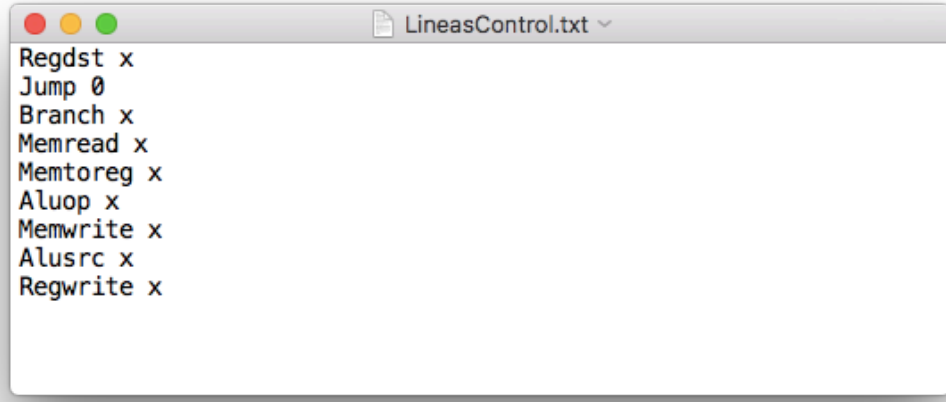
4.1 RESULTADOS

A continuación se presentan los resultados obtenidos de un caso de prueba y sus respectivos archivos de salida.



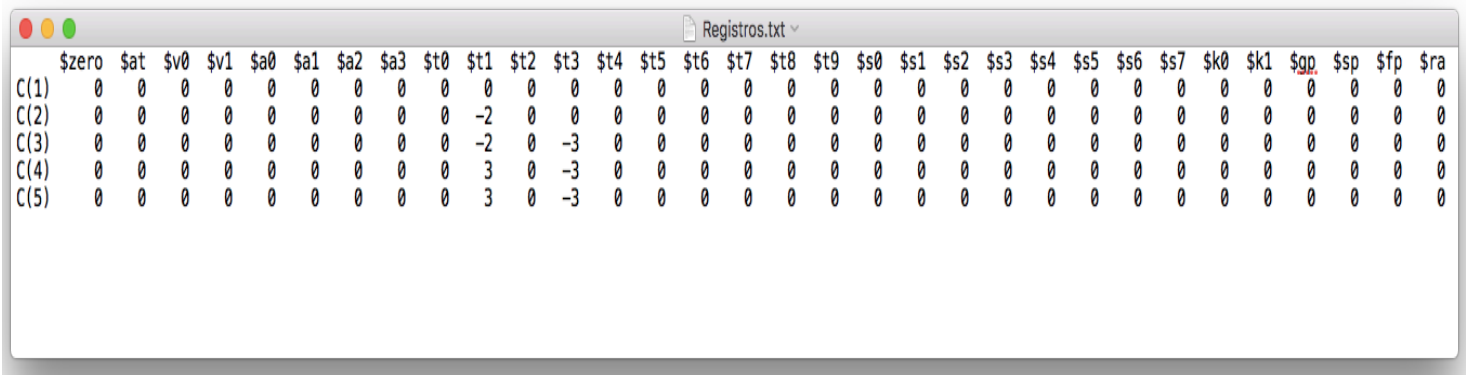
```
Jump end
subi $t2, $zero, 2
subi $t3, $zero, 3
sub $t1, $t2, $t3
sub $t3, $t2, $t1
end:
```

Ilustración 4.1 Archivo con instrucciones prueba



```
Regdst x
Jump 0
Branch x
Memread x
Mentoreg x
Aluop x
Memwrite x
Alusrc x
Regwrite x
```

Ilustración 4.2 Archivo líneas de control prueba



	\$zero	\$at	\$v0	\$v1	\$a0	\$a1	\$a2	\$a3	\$t0	\$t1	\$t2	\$t3	\$t4	\$t5	\$t6	\$t7	\$t8	\$t9	\$s0	\$s1	\$s2	\$s3	\$s4	\$s5	\$s6	\$s7	\$k0	\$k1	\$gp	\$sp	\$fp	\$ra
C(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C(2)	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C(3)	0	0	0	0	0	0	0	0	0	-2	0	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C(4)	0	0	0	0	0	0	0	0	0	3	0	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C(5)	0	0	0	0	0	0	0	0	0	3	0	-3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ilustración 4.3 Archivo salida registros prueba

4.2 ANÁLISIS DE RESULTADOS

Se ha logrado que el programa sea capaz de leer las instrucciones y las líneas de control que presentan problemas de los archivos de texto plano, para luego simular su ejecución alterando el estado de los registros involucrados en el programa en MIPS. Aún así para su correcto funcionamiento se debe conocer de manera anticipada cómo funciona la memoria en la simulación, ya que se cuenta con una cantidad mucho menor al común de los computadores actuales y el acceso a un valor dentro de esta se realiza por medio de una coordenada que indica la posición de la matriz en donde dicho valor se encuentra.

Además puede ocurrir una sobrescritura de la memoria si el offset es demasiado alto, superando los límites de la matriz, lo que provocaría una incongruencia en los resultados.

CAPÍTULO 5. CONCLUSIONES

Los objetivos indicados a inicios de este informe se han cumplido a cabalidad . El funcionamiento del programa es el correcto bajo el estándar ANSI C , siempre y cuando se respete la implementación de la memoria ram dentro del programa.

Se está conforme con los resultados obtenidos y los conocimientos que este laboratorio a entregado, esperando mejorar en la investigación previa a la realización del código y la creación de funciones más compactas e intuitivas, que resulten más fáciles de entender.

Haber realizado este laboratorio resultó en innumerables frutos, sobretodo del punto de vista de los conocimientos asimilados. Tener idea de cómo afectan las líneas de control en el proceso de ejecución de una instrucción es una herramienta muy eficaz a la hora de analizar el camino de datos bajo una determinada arquitectura.

CAPÍTULO 6. REFERENCIAS

Kernighan, B., Ritchie, D. (1988). *El lenguaje de programación C*. México: Pearson.