

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



Laboratorio 1 de Organización de Computadores

Integrantes: Cristian Espinoza
Curso: Organización de Computadores
Sección L1
Profesor: Daniel Wladdimiro
Ayudante: Sebastian Pinto-Agüero

11 de Septiembre de 2017

Tabla de contenidos

| | |
|---|-----------|
| 1. Introducción | 1 |
| 1.1. Objetivos Generales | 2 |
| 1.2. Objetivos Específicos | 2 |
| 1.3. Datos | 2 |
| 1.4. Herramientas | 2 |
| 2. Marco teórico | 3 |
| 2.1. Tipo de instrucciones | 3 |
| 2.2. Camino de datos | 3 |
| 2.3. Lineas de control | 4 |
| 3. Desarrollo | 5 |
| 3.1. Estructuras | 5 |
| 3.2. Funciones | 7 |
| 3.2.1. Funciones necesarias para el programa | 7 |
| 3.2.2. Funciones para cada instrucción de MIPS | 8 |
| 3.2.3. Funciones de escritura/lectura de los archivos | 8 |
| 4. Experimento | 9 |
| 4.1. Resultados obtenidos | 9 |
| 4.2. Análisis de resultados | 9 |
| 5. Conclusiones | 10 |
| Bibliografía | 11 |

Índice de figuras

| | | |
|----|---|---|
| 1. | Tipo de instrucciones | 3 |
| 2. | Registro de MIPS. | 5 |
| 3. | Lineas de control | 5 |
| 4. | Label | 5 |
| 5. | Instrucciones | 6 |
| 6. | Lista | 6 |
| 7. | Instrucciones | 7 |
| 8. | Resultado de ejemplo numero uno | 9 |

1. Introducción

En el presente documento se ha pedido a los alumnos de la asignatura Organización de computadores que presenten el trabajo realizado en el laboratorio numero uno, el cual al realizarlo conlleva el aprendizaje de los contenidos en forma practica y teórica simultáneamente.

En el presente laboratorio se usarán directamente conocimientos adquiridos en cátedra, tales como: lineas de control y camino de datos. Además, es necesario tener un conocimiento previo del lenguaje ensamblador.

En el presente laboratorio, se deberían leer dos archivos de entrada, el primero contiene las instrucciones de un programa de MIPS, mientras que el segundo, contiene las líneas de control de cada una de las instrucciones del programa. Dado los dos archivos de entrada, usted debe generar un archivo de salida donde entregue la traza de la ejecución del código propuesto en el primer archivo de entrada, indicando si las líneas de control que usted obtiene son las mismas que las líneas de control del segundo archivo, y en caso de existir un error, indicar cual de los bits es el erróneo. Guzmán et al. (2017a)

La solución del presente laboratorio se abarca con la herramienta de división en sub-problemas, con el fin de poder abarcar en su totalidad cada una de sus dificultades. Destacar que el alcance que se presenta es el cumplimiento del desarrollo del laboratorio.

Por ultimo, en este informe se presenta un marco teórico necesario para el entendimiento de todo el desarrollo del programa, además se cuenta con un desarrollo que explica cómo se realizo, las herramientas y técnicas utilizadas para el desarrollo de este.

1.1. Objetivos Generales

El objetivo general es realizar un programa capaz de solucionar el problema expuesto en el enunciado, desarrollado en lenguaje de programación C, ocupando el paradigma de programación imperativo.

1.2. Objetivos Específicos

Los objetivos específicos del programa son implementar lo siguiente:

- Entregar de manera correcta la solución del problema planteado.
- Aprender correctamente el camino de datos y sus respectivas líneas de control de las distintas instrucciones de MIPS.

1.3. Datos

- El archivo uno contiene las instrucciones del programa en MIPS.
- El archivo dos contiene las líneas de control de las instrucciones del programa en MIPS.

1.4. Herramientas

Las herramientas a utilizar para el desarrollo del laboratorio número uno es un editor de texto encarecido, es decir, funciona como una ayuda con la sintaxis propia del lenguaje destacando estructuras, funciones, declaraciones, etc.

Este programa se desarrolla en el entorno de Windows, sin embargo, no se requiere estrictamente, compilar y ejecutar en dicho sistema operativo.

2. Marco teórico

Tal como se informó anteriormente, varios conceptos son necesarios para entender el desarrollo de este laboratorio y evitar confusiones para el lector.

A continuación, se dejarán en claro los conceptos que se deben manejar para una adecuada lectura del informe.

2.1. Tipo de instrucciones

MIPS posee tres tipos de instrucciones, que se separan en las siguientes clases:

- Tipo R: Operaciones aritmético-lógicas.
- Tipo I: Operaciones de memoria.
- Tipo J: Operaciones de salto.

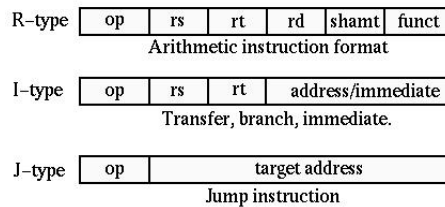


Figura 1: Tipo de instrucciones

2.2. Camino de datos

Es la sección del computador encargada de manipular y transformar los datos procedentes de la memoria o los registros internos, para obtener los resultados correspondientes.

En el camino de datos existen 5 etapas las cuales, son activas de cierta forma, por medio de una unidad de control. Estas etapas son las siguientes:

- IF (Instruction Fetch): Esta etapa se encarga de enviar el contador de programa (PC) a la memoria que contiene el código y búsqueda de la instrucción desde la memoria de instrucciones. Guzmán et al. (2017b)

- ID (Instruction Decode): Leer uno o dos registros (dependiendo de la instrucción) usando los campos de la instrucción. Guzmán et al. (2017b)
- EX (Execute): Se utiliza la ALU, para realizar una operación aritmético-lógica o calcular una dirección. Guzmán et al. (2017b)
- MEM (Data Memory Access): Se accede a memoria para leer o escribir. Guzmán et al. (2017b)
- WB (Write Back): Se escribe en el registro correspondiente. Guzmán et al. (2017b)

2.3. Líneas de control

Es la sección del computador encargada de interpretar las instrucciones del programa y gobernar la ejecución de las mismas.

Una señal de control es utilizada para la selección de un multiplexor o para direccionar la operación de una unidad funcional.

Para efectos de entendimientos esta unidad de control tiene las siguientes líneas de control:

- Líneas de un bit las cuales pueden tomar el valor de '1' y '0'.
 - RegDst.
 - Jump.
 - Branch.
 - MemRead.
 - MemToReg.
 - MemWrite.
 - ALUSrc.
 - RegWrite.
- Línea de dos bits la cual puede tomar el valor de '00', '01' y '10'.
 - ALUOp.

3. Desarrollo

Para efectos de facilitar las funcionalidades del programa, se crea un arreglo constante, el cual contiene el nombre de todo los registros que existen, el cual se puede observar en la figura 2.

```
const char *registroMars[32] = {"$zero", "$at", "$v0", "$v1", "$a0", "$a1", "$a2", "$a3", "$t0", "$t1",  
"$t2", "$t3", "$t4", "$t5", "$t6", "$t7", "$s0", "$s1", "$s2", "$s3", "$s4", "$s5", "$s6", "$s7", "$t8", "$t9",  
"$k0", "$k1", "$gp", "$sp", "$fp", "$ra"};
```

Figura 2: Registro de MIPS.

3.1. Estructuras

- Lineas de control: Esta estructura se encarga de almacenar los valores que van obteniendo cada una de las instrucciones durante transcurso de la traza del programa.

```
typedef struct LineaDeControl  
{  
    char* Estado;  
    int PC;  
    char RegDst;  
    char Jump;  
    char Branch;  
    char MemRead;  
    char MemToReg;  
    char* ALUOp;  
    char MemWrite;  
    char ALUSrc;  
    char RegWrite;  
}LineaDeControl;
```

Figura 3: Lineas de control

- Label: Esta estructura es la que se encarga de almacenar las etiquetas del programa MIPS, por lo tanto, lleva un arreglo de caracteres que corresponde a la etiqueta y, además un entero , que guarda el PC (program counter) de la instrucción a la cual debe saltar, en caso de ser jump o Branch. La estructura está compuesta por:

```
typedef struct Label  
{  
    char *label;  
    int PC;  
}Label;
```

Figura 4: Label

- Instrucción: Esta estructura se encarga de almacenar toda la información de las instrucciones que se leen desde el archivo de texto, por lo tanto, esta estructura, la cual tiene 3 variables de tipo char, las cuales representan a los registros que utilizaría la instrucción (rs, rt y rd), cabe destacar, que algunas de las instrucciones no requieren de todo los registros mencionados. Además, se tienen dos variables de tipo int, la primera corresponde al valor inmediato, el cual es utilizado por las instrucciones de tipo I y la segunda variable int corresponde al PC que indica la posición de la instrucción. Por otro lado, se tiene una variable de tipo char contiene el nombre de la instrucción leída. Por ultimo, contiene una variable de tipo *Linea de control* la cual cumple el rol de guardar la linea de control de la instrucción.

```
typedef struct Instruccion
{
    char *instruccion;
    char *rt;
    char *rs;
    char *rd;
    int inmediato;
    int PC;
    LineaDeControl* lineaDeControl;
}Instruccion;
```

Figura 5: Instrucciones

- Lista: Esta estructura esta encargada de almacenar la totalidad de lineas de control que se generar al realizar la traza del programa de MIPS, por lo tanto, tiene como atributo una variable int que se encarga de mantener actualizado el largo de la lista y, una variable de tipo Lineas de control que contiene la información de las instrucciones.

```
typedef struct Lista
{
    int largo;
    LineaDeControl* lineaTraza;
}Lista;
```

Figura 6: Lista

- Información: Esta estructura es la mas relevante, porque es la que contiene a las anteriores, y además almacena todo que se necesita teóricamente, como también de manera practica. Contiene una variable de tipo int que contiene la cantidad de instrucciones que contiene el programa de MIPS y por otro lado, contiene un arreglo de int que se

encarga de guardar los valores que toma cada uno de los registros en el transcurso del programa. Por ultimo, un arreglo de int que se encarga de representar la memoria del programa que se utiliza para las funciones 'lw' y 'sw' (load word y store word).

```
typedef struct Informacion
{
    Instruccion* instrucciones;
    int registros[32];
    int cantidadDeInstrucciones;
    Label etiqueta[100];
    int memoria[1000];
    Lista* lineasDeControl;
}Informacion;
```

Figura 7: Instrucciones

3.2. Funciones

Las funciones se separan en tres partes, primero entre las que son necesarias para el desarrollo del programa, las que actúan directamente a las instrucciones del programa MIPS y por ultimo las funciones que se encargan de lectura y escritura de los archivos.

3.2.1. Funciones necesarias para el programa

Estas funciones son las que se implementaron antes de abordar el objetivo principal de la solución que es encontrar errores en las lineas del control al realizar la traza del programa.

- *int obtenerPosicionReg(char* registro) y int buscarPosicionEtiqueta(char* etiqueta, Informacion *info)*: Se encarga de obtener la posición de un registro por su nombre o la de una etiqueta respectivamente. El retorno de estas funciones son el valor del registro o el PC de la etiqueta.
- *void traza(Informacion* info, Instruccion* instrucciones, char* nombreArchivoSalida, LineaDeControl* lineasControlDadas)*: Esta función se encarga de recorrer el arreglo de instrucciones, en donde mediante un ciclo comienza a realizar la traza del programa, existen condiciones que conllevan a determinar la instrucción siguiente, de esta forma se determina lo que se debe realizar. Esta función utiliza a la funciones *obtenerPosicionReg*

y *buscarPosicionEtiqueta* con el objetivo de poder ir obteniendo los valores de los saltos que debe realizar dicha traza. El retorno final de la función es el arreglo de líneas de control que despliega la traza realizada por el programa, con el fin de poder comparar con las líneas de control entregadas.

3.2.2. Funciones para cada instrucción de MIPS

Estas funciones cumplen con la operación que realiza la instrucción de MIPS que se lee desde el archivo uno.

Las instrucciones implementadas son: add, sub, addi, subi, div, mul, beq, jump, lw, sw. La mayoría de las funciones tiene retorno void, excepto de las funciones jump y beq, que retorna el PC de la función donde debe saltar y retorna '1' en caso de cumplirse la condición y '0' en caso contrario, respectivamente.

3.2.3. Funciones de escritura/lectura de los archivos

Estas funciones están encargadas de leer la información presente en los archivos de entrada, además de escribir la salida del programa.

- *Informacion* leerInstrucciones(char nombre[], int numeroDeLineas)*: Esta función se encarga de leer las instrucciones que contiene el archivo de MIPS, guardando la información de las etiquetas e instrucciones en la estructura *Instrucciones* que se menciono anteriormente, además se encarga de inicializar los registros en '0'. Esta función retorna un puntero a *Información*.
- *LineaDeControl* leerLineaDeControl(char nombre[],int numeroDeLineas)*: Esta función se encarga de leer las líneas de control que le pertenecen a las instrucciones que se leen desde el archivo numero uno.
- *void realizarSalida(Lista* lista,LineaDeControl* lineasControlDadas)*: Esta función se encarga de realizar la salida del archivo, que contendrá las líneas de control de la traza del programa, indicando si dichas líneas de control están correctas o erróneas (Indicando el error correspondiente).

4. Experimento

4.1. Resultados obtenidos

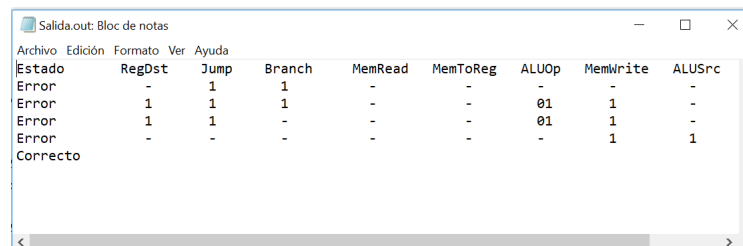
El programa consta con una interfaz, en la cual el usuario debe ingresar la opción a realizar para poner en curso el programa.

Al escoger la opción numero uno, debe ingresar el nombre del archivo uno con su respectiva extensión *'asm'*, que contiene las instrucciones en MIPS.

Después de ingresar el nombre del archivo uno, debe ingresar a la opción numero dos, donde debe ingresar el nombre del archivo dos con su respectiva extensión *'txt'*, que contiene las lineas de control de las instrucciones que son leídas desde el archivo uno.

Por ultimo, debe ingresar la opción numero tres, con el fin de realizar la traza del programa, entregando como resultado un archivo de salida con extensión *'out'*, donde se encontrara el análisis de las lineas de control de la traza realizada por el programa.

En la figura 8, se muestra la salida que entrega el programa al probar el archivo de prueba numero uno:



| Estado | RegDst | Jump | Branch | MemRead | MemToReg | ALUOp | MemWrite | ALUSrc |
|----------|--------|------|--------|---------|----------|-------|----------|--------|
| Error | - | 1 | 1 | - | - | - | - | - |
| Error | 1 | 1 | 1 | - | - | 01 | 1 | - |
| Error | 1 | 1 | - | - | - | 01 | 1 | - |
| Error | - | - | - | - | - | - | 1 | 1 |
| Correcto | | | | | | | | |

Figura 8: Resultado de ejemplo numero uno

4.2. Análisis de resultados

En la figura 8 se puede apreciar que el resultado del programa es correcto, ya que comprueba el valor de las lineas de control, indicando si es correcto o contiene algún error.

La solución propuesta es escalable, ya que se puede agregar las instrucciones restantes al código fuente.

5. Conclusiones

En este presente Laboratorio se pudo llevar a practica la teoría que se aprendió en la cátedra, pudiendo tener una visión mas integral del cómo llevar a la práctica cada uno de los elementos vistos y estudiados.

Como se ha mencionado anteriormente, se ha podido lograr la implementación en cada uno de los objetivos tanto como generales como específicos. Además, destacar que la mayor dificultad dentro del desarrollo del programa fue la organización del código para obtener la solución requerida.

Finalmente, se debe destacar que los resultados del programa fueron exitosos en las pruebas realizadas, además se implementó un algoritmo de $O(n)$, sin embargo, se debe dejar abierta la posibilidad de posibles mejoras para dicho algoritmo. Es por esto que se espera una retroalimentación, para evitar errores durante el segundo semestre del 2017.

Bibliografía

Guzmán, A., Wladdimiro, D., Alvarez, M., Pinto-Aguero, S., and Undurraga, A. (2017a). Organizacion de computadores laboratorio 1. [Online] http://www.udesantiagoovirtual.cl/moodle2/pluginfile.php?file=%2F111799%2Fmod_resource%2Fcontent%2F8%2F0C%20Lab%201.pdf.

Guzmán, A., Wladdimiro, D., and Rosas, E. (2017b). Organizacion de computadores camino de datos y unidad de control. [Online] http://www.udesantiagoovirtual.cl/moodle2/pluginfile.php?file=%2F111752%2Fmod_resource%2Fcontent%2F3%2F0C_Procesador.pdf.