

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



Desarrollo de un simulador MIPS en lenguaje C

Alumno: Kevin Arévalo Fernández
Curso: Organización de computadores
Sección L3
Profesor(a): Leo Medina
Ayudante: Alexander Palma Larraín

10 de Mayo de 2019

Tabla de contenidos

1. Introducción	1
1.1. Objetivo general	1
1.2. Objetivos específicos	1
2. Marco teórico	2
2.1. CPU	2
2.2. Memoria RAM	3
2.3. MIPS	3
3. Desarrollo	6
3.1. Entrada	6
3.2. Instrucciones	7
3.3. Registros	7
3.4. Memoria	8
3.5. Simulación del ciclo de ejecución de instrucciones	9
4. Conclusiones	10
Bibliografía	11

1. Introducción

En el presente informe se describe el proceso de elaboración de un simulador de un procesador monociclo que ejecuta un subconjunto de las instrucciones totales de MIPS, desarrollado en el lenguaje C.

Las instrucciones que es capaz de ejecutar el simulador desarrollado son las siguientes:

- lw
- sw
- add
- addi
- sub
- and
- or
- slt
- beq
- j

1.1. Objetivo general

- La comprensión del funcionamiento de un procesador que ejecuta instrucciones MIPS.

1.2. Objetivos específicos

- Comprender el camino de datos del procesador
- Conocer cómo almacena e interpreta el procesador, las instrucciones de los tres formatos (R, I, J)

2. Marco teórico

2.1. CPU

Para el correcto entendimiento de este trabajo, es conveniente introducir al lector en los componentes principales que posee un procesador (CPU), los cuales se muestran en la figura 1

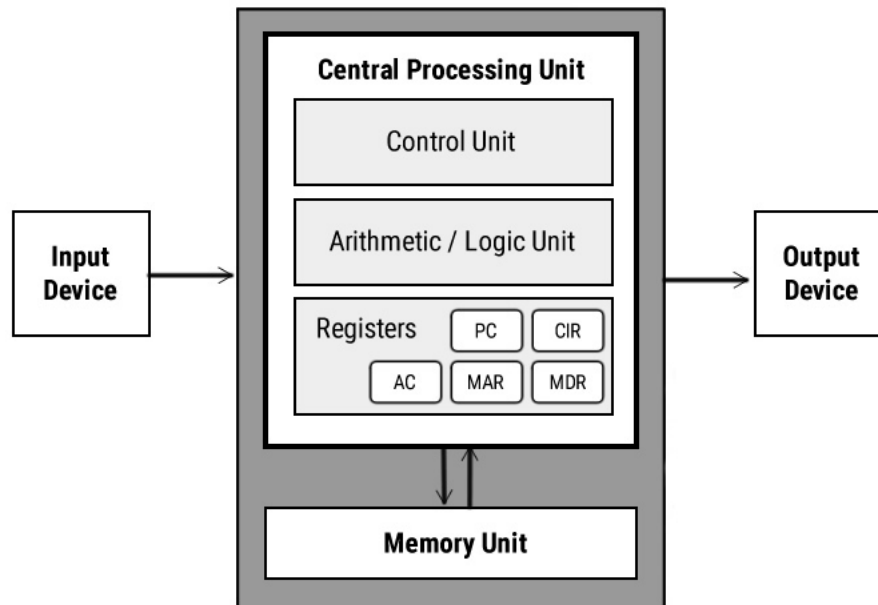


Figura 1: Componentes principales de un procesador. Arquitectura de Von Neumann. (Unknown, 2018).

A modo de funcionamiento muy simplificado, un procesador lee instrucciones y datos, los cuales son almacenados en la memoria RAM. A la medida que se ejecutan estas instrucciones, los datos son llevados mediante un camino de datos hacia los registros del procesador, esto permite que sean operados por la ALU (Arithmetic / Logic Unit). Una vez operados, los resultados son almacenados en los registros, y si el programador desea, puede almacenarlos en memoria RAM. Finalmente cuando el procesador termina de ejecutar todas las instrucciones que tenía almacenadas, se muestra una salida.

2.2. Memoria RAM

En la sección anterior se menciona la memoria ram, a continuación, en la figura 2 se ilustra cómo los datos son almacenados.

En la arquitectura de estudio de este trabajo, los datos e instrucciones son almacenados como cadenas de 32 bits (4 bytes), estos datos e instrucciones son identificados por su posición en memoria, que al ser medida en bytes, provoca que estas direcciones sean múltiplos de 4.

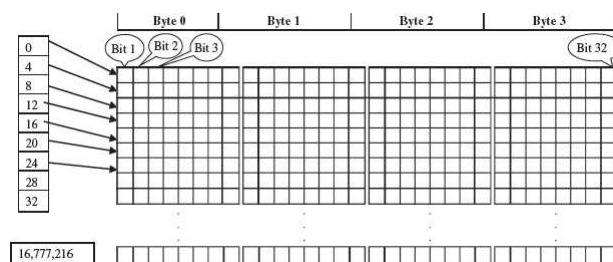


Figura 2: Representación de la memoria. (Unknown, 2012).

2.3. MIPS

Dado que ya se conoce el funcionamiento general de un procesador, están las condiciones necesarias para poder comprender qué es MIPS.

MIPS es una "ISA" (Instruction Set Architecture), que significa básicamente una arquitectura a seguir en el diseño de las instrucciones que puede ejecutar un procesador. Un procesador MIPS, es entonces, hardware dispuesto de manera tal que permita la ejecución del conjunto de instrucciones del ISA MIPS. (David A. Patterson, 1998).

La arquitectura MIPS soporta tres tipos de instrucciones (R, I, J), que se ilustran en la figura 3

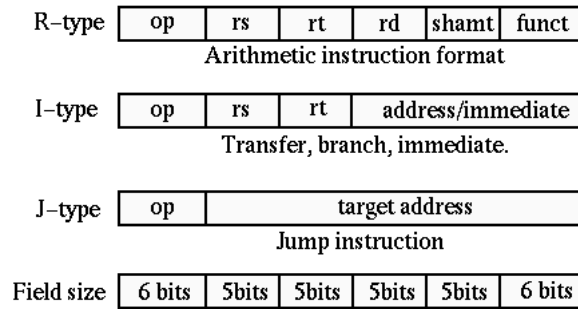


Figura 3: Tipos de instrucciones en MIPS. (Wang, 2003).

El procesador opera cada una de las instrucciones que le son cargadas, mediante el ciclo de la figura 4

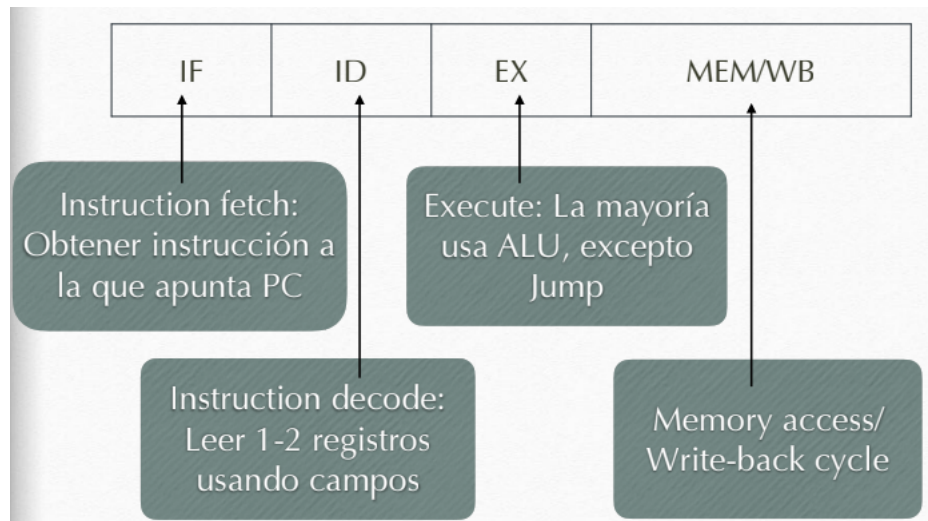


Figura 4: Ciclo de Instrucción

A nivel de hardware, un procesador MIPS luce como en la figura

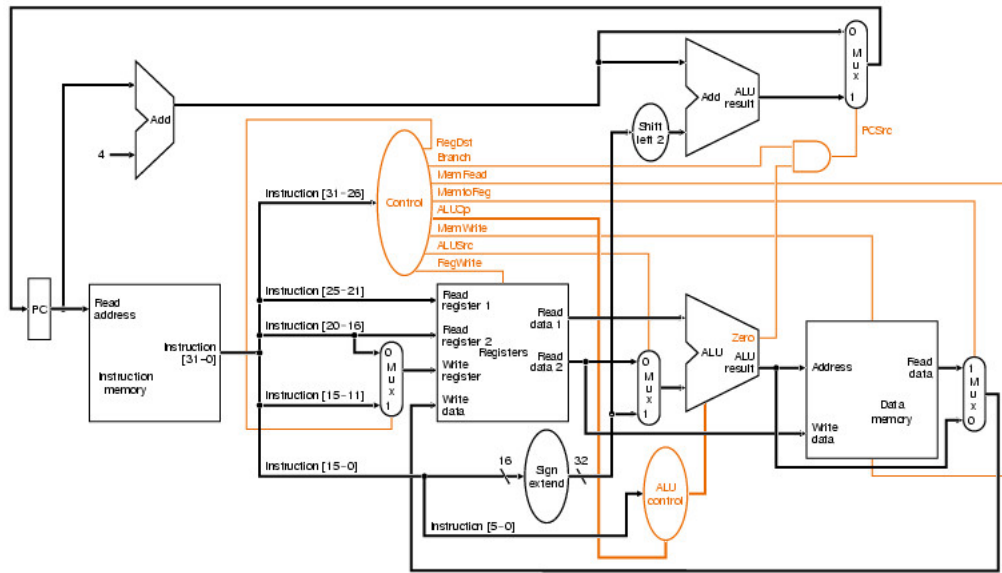
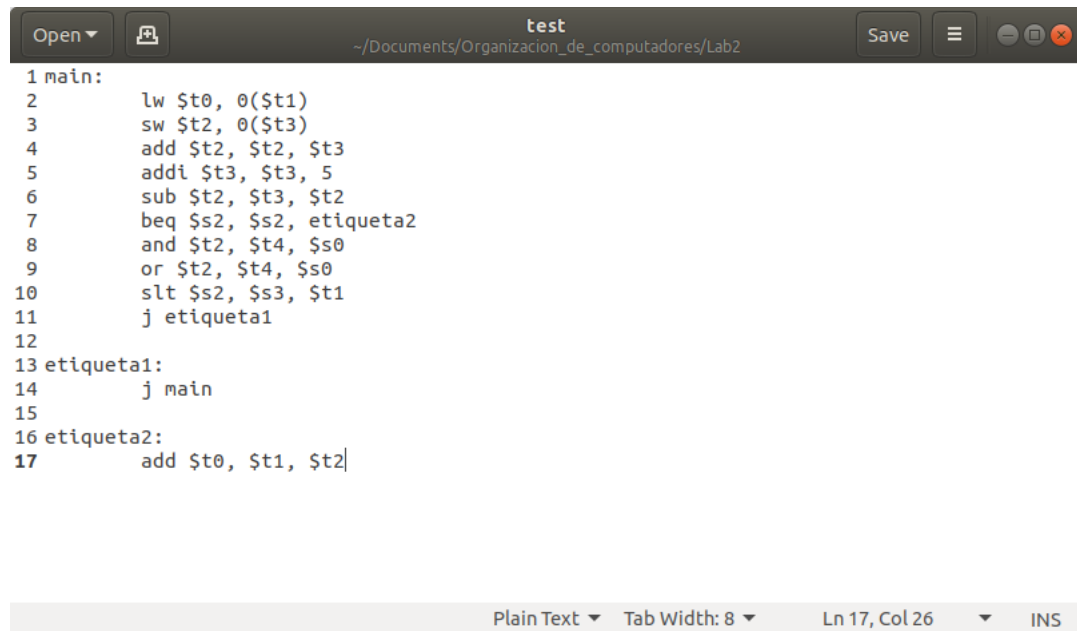


Figura 5: Hardware de un procesador MIPS. (Wang, 2005).

3. Desarrollo

3.1. Entrada

El simulador lee un archivo de texto plano con código escrito en MIPS, y limitado a las instrucciones listadas en la introducción.



```
1 main:
2     lw $t0, 0($t1)
3     sw $t2, 0($t3)
4     add $t2, $t2, $t3
5     addi $t3, $t3, 5
6     sub $t2, $t3, $t2
7     beq $s2, $s2, etiqueta2
8     and $t2, $t4, $s0
9     or $t2, $t4, $s0
10    slt $s2, $s3, $t1
11    j etiqueta1
12
13 etiqueta1:
14    j main
15
16 etiqueta2:
17    add $t0, $t1, $t2
```

Plain Text ▾ Tab Width: 8 ▾ Ln 17, Col 26 ▾ INS

Figura 6: Ejemplo de entrada.

Las instrucciones escritas en el archivo son guardadas de forma conveniente y respetando la estructura interna de MIPS, es decir, se representa cada aspecto de la instrucción en código binario, dependiendo de su tipo.

Para simular el comportamiento de un procesador MIPS en C, se identifican los componentes principales, los cuales son: instrucciones, registros, memoria. Cada uno de ellos es representado mediante estructuras adecuadas explicadas a continuación.

3.2. Instrucciones

La figura 8 ilustra la representación en C de las instrucciones a ejecutar en el simulador MIPS. La estructura `instruction` puede ser utilizada para almacenar cualquiera de los tres tipos de instrucciones (R, I, J). Aquellos campos no requeridos para algún tipo de instrucción se dejan en valor `"NULL"`. Por ejemplo, si se está almacenando una instrucción de tipo J, no es necesario utilizar el campo `immediate`, ya que por el propio formato de la instrucción, este campo es inexistente.

```
struct instruction{
    char* address; // 32 Bits
    char* name;
    char type;
    char* op; // 6 Bits
    char* rs; // 5 Bits
    char* rt; // 5 Bits
    char* rd; // Tipo R, 5 Bits
    char* shamt; // Tipo R, 5 Bits
    char* func; // Tipo R, 6 Bits
    char* immediate; // Tipo I, 16 Bits
    char* j_address; // Tipo j, 26 Bits
    char* label;
    struct instruction* next;
};
typedef struct instruction Instruction;

struct instr_memory{
    char* programCounter;
    Instruction* first;
    Instruction* last;
    int length;
};
typedef struct instr_memory Instr_memory;
```

Figura 7: Estructuras empleadas para la representación de instrucciones.

Para almacenar todas las instrucciones necesarias, se emplea una lista enlazada, encapsulada en la estructura `instrmemory`, que además de tener punteros a la primera y última instrucción, cuenta con la cantidad de instrucciones almacenadas y el program counter, que indica la instrucción en ejecución en el ciclo del programa.

3.3. Registros

Para el caso de los registros se utiliza una estrategia similar. Se cuenta con dos estructuras, una que representa a un registro como tal, y otra estructura encapsuladora y con datos de interés relacionados, como los registros que se están leyendo en el ciclo actual y los datos leídos desde estos registros. En este caso, se almacenan los registros en un arreglo y no en una lista enlazada, ya que son una cantidad definida (32).

```
struct regist{
    char* name;|
    char* number; // 32 Bits
    char* data; // 32 Bits
};
typedef struct regist Register;

struct reg_file{
    char* readRegister1; // 5 Bits
    char* readRegister2; // 5 Bits
    char* readData1; // 32 Bits
    char* readData2; // 32 Bits
    Register** registers;
};
typedef struct reg_file Reg_file;
```

Figura 8: Estructuras empleadas para la representación de registros.

3.4. Memoria

La representación de la memoria, naturalmente sigue el mismo esquema. También se utilizan dos estructuras para representarla. Siendo una de ellas la que encapsula una lista enlazada.

```
struct mem_data{
    char* address; // 32 Bits
    char* data; // 32 Bits
    struct mem_data* next;
};
typedef struct mem_data Mem_data;

struct ram_mem{
    char* current_addr;
    Mem_data* first;
    Mem_data* last;
    Mem_data* length;
};
typedef struct ram_mem Ram_mem;
```

Figura 9: Estructuras empleadas para la representación de la memoria.

3.5. Simulación del ciclo de ejecución de instrucciones

Para simular el proceso de ejecución de las instrucciones, se utiliza un algoritmo bastante sencillo, que consiste en buscar en la memoria de instrucciones, aquella, cuya dirección coincida con la indicada en el Program Counter. Una vez encontrada esta instrucción, se incrementa automáticamente el Program Counter y se procede de manera diferenciada, según la instrucción particular, ésta puede ocasionar escrituras en memoria y en registros, cuyos valores son trazados en un archivo de texto que constituye la salida del programa.

```
1 mientras((Instruccion = leerInstruccionDesdeMemoria(PC)) != NULL):  
2     PC = PC+4  
3     Leer los registros involucrados en la instruccion  
4     Operar dichos registros según el funcionamiento de la instruccion|
```

Figura 10: Pseudocódigo que representa la ejecución principal del simulador.

Así por ejemplo si la instrucción a ejecutar es:

add \$s0, \$t1, \$t2

Se extraen los registros $rs = \$t1$, $rt = \$t2$ y $rd = \$s0$ cuya dirección está disponible en la estructura de la instrucción leída. Luego se escribe el resultado en el registro correspondiente $\$s0 = \$t1 + \$t2$ y se continúa con el ciclo (recordemos que el Program Counter ya se ha incrementado, por lo tanto, en la próxima iteración del ciclo, se leerá la instrucción siguiente).

4. Conclusiones

Mediante la programación del simulador, se comprende de mejor manera cómo funciona de forma interna un procesador MIPS monociclo.

Se dan cuenta detalles como por ejemplo, que los accesos a memoria son más complejos que aquellas instrucciones que operan sólo utilizando los registros internos del procesador. Estos aspectos son relevantes, y cada programador debería por lo menos saber los conceptos básicos aquí expuestos, para desarrollar código más eficiente.

Si bien los procesadores MIPS no están presentes en los dispositivos más comunes de hoy en día (smartphones o computadores personales), el estudio de esta arquitectura significa un aprendizaje de los conceptos básicos que rigen el comportamiento de la computación, y dichos conceptos son aplicables a otros tipos de arquitecturas (como intel x86x64) en el caso de que se desee aprenderlas.

Bibliografía

David A. Patterson, J. L. (1998). *Computer Organization and Design*, volume 5. Morgan Kaufman.

Unknown (2012). Computer memory representation. [Online] <http://generalnote.com/Computer-Fundamental/Computer-Memory/Memory-Representation.php>.

Unknown (2018). Von-neumann-architecture-diagram. [Online] <https://codefluegel.com/en/die-zukunft-des-prozessors/von-neumann-architecture-diagram/>.

Wang, R. (2003). Instruction set of mips processor. [Online] http://fourier.eng.hmc.edu/e85_old/lectures/instruction/node7.html.

Wang, R. (2005). Mips processor (single cycle). [Online] http://fourier.eng.hmc.edu/e85_old/lectures/processor/node4.html.