Proyecto 1. Máquina Virtual MIPS Ligero

1. Objetivo General

Desarrollar una máquina virtual MVML (Máquina Virtual MIPS Ligero) capaz de ejecutar programas escritos con un conjunto reducido de instrucciones MIPS (MIPSLigero). Esta máquina virtual será desarrollada en lenguaje ensamblador MIPS.

Los programas que ejecutará la MVML estarán ensamblados en MIPSLigero. Cada instrucción es de 32 bits y contiene el código de la operación a llevar a cabo y los operandos sobre los cuales actúa siguiendo los formatos de instrucción que serán especificados más adelante.

La MVML tiene un conjunto reducido de instrucciones para:

- Carga y almacenamiento en memoria
- Operaciones Aritméticas
- Transferencia de Control (saltos)

La máquina virtual ejecuta cada instrucción de un programa almacenado en memoria siguiendo el siguiente ciclo de ejecución.

2. Ciclo de Ejecución

Se define como el ciclo de ejecución de un procesador a los pasos internos que sigue para ejecutar una instrucción. El número de pasos y duración de este ciclo varían de procesador a procesador y depende de su arquitectura. A modo de simplificación para el desarrollo de nuestra máquina virtual, el ciclo de ejecución consta de cinco etapas:

- Búsqueda de la instrucción (IF),
- Decodificación y Búsqueda de los Registros(ID),
- Ejecución (E)
- Acceso a Memoria(Mem)
- Escritura del resultado (WB)

2.1 Búsqueda de la Instrucción

En esta fase, el procesador obtiene de memoria la siguiente instrucción a ejecutar. El procesador recibe los bytes de la instrucción (4 Bytes) y los almacena en un registro de instrucción (IR) para proceder a su decodificación. Una vez obtenidos los bytes de la instrucción se incrementa el contador de programa (PC) para que apunte a la siguiente instrucción. Como todas las instrucciones en MIPSLigero son de 32 bits, el contador de programa se incrementa en 4 unidades.

2.2 Decodificación y Búsqueda de los Registros

En el proceso de decodificación se extrae el código de operación. Dependiendo de este valor se procede a obtener el resto de los elementos de la instrucción tomando en cuenta los formatos de instrucción de MIPSLigero. Al terminar esta fase ya se sabe con exactitud qué operación se deberá realizar y la forma en que se obtendrán los operandos.

Formatos de Instrucción

Tipo R

(shamt: *shift amount* en instrucciones de desplazamiento)

Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino		Funct
xxxxxx	rs	rt	rd	shamt	funct
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0

Tipo I

(carga o almacenamiento, ramificación condicional, operaciones con inmediatos)

Cód. Op.	Registro base	Registro destino	Desplazamiento
xxxxxx	rs	rt	Offset
6	5	5	16
31-26	25-21	20-16	15-0

Tipo J

(salto incondicional)

Cód. Op.	Dirección destino
xxxxxx	target
6	26
31-26	25-0

2.3 Ejecución

En esta fase se realiza el cálculo aritmético/lógico especificado con la operación obtenida en el paso anterior. Si se trata de una operación de transferencia de control, se lleva a cabo la comparación de los operandos y el cálculo de la dirección de salto. Para el caso de las operaciones de transferencia se calcula la dirección del operando en memoria.

2.4 Acceso a Memoria

Para el caso de las operaciones de transferencia (Load / Store) se accede a memoria tomando en cuenta la dirección del operando calculada en la etapa anterior.

2.5 Escritura del resultado

Una vez terminada la operación aritmético/lógica o load codificada en la instrucción, el procesador guarda el resultado en el registro destino especificado en la instrucción.

Al terminar esta fase, el procesador tiene en el contador de programa la dirección de memoria en la que está almacenada la siguiente instrucción a ser ejecutada. Se repite este ciclo de instrucción con la búsqueda de la siguiente instrucción a ejecutar.

3. Primera Fase de MLMV:
Lectura y Carga en memoria de un programa ensamblado

3.1 Objetivos Específicos

- Adquirir destrezas en programación con el lenguaje ensamblador MIPS
- Adquirir destrezas en el uso de llamadas al sistema para la lectura de archivos
- Adquirir destrezas en la manipulación de bits a nivel de lenguaje ensamblador

3.2 Detalles de Implementación

Para la primera fase se desea que implemente un programa en MIPS para la lectura de un archivo que contiene un programa ensamblado en MIPSLigero y que posteriormente será ejecutado por la máquina virtual a desarrollar a lo largo del taller. Su programa deberà solicitar al usuario el nombre del archivo de texto que coniene el programa ensamblado.

MIPS ofrece una serie de llamadas al sistema para la Entrada y Salida a través de archivos.

Llamada	$v0	Argumentos	Resultado
open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error).
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error).
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error).
close file	16	$a0 = file descriptor	

La llamada al sistema 14 (read from file) permite la lectura de un cierto número de caracteres que se especifica en el registro $a2.

Un programa ensamblado en Hexadecimal almacenado en un archivo de texto lucirá como:

```
2402000d
3c011001
34240000
24050000
24060000
0000000c
0002b021
2402000e
00162021
3c011001
```

Su programa en MIPS debe leer cada una de las líneas del archivo de texto y transformarlas para que sean almacenadas correctamente en una palabra de la memoria para su posterior ejecución.

Cada línea del archivo de entrada consta de 10 caracteres:

- Los 8 primeros caracteres corresponden al ensamblaje de la instrucción y los caracteres que pueden aparecer son ´0´, ´1´, ´2´, ´3´,´4´,´5´,´6´,´7´,´8´,´9´,´a´,´b´,´c´,´d´,´e´ y ´f´
- Los siguientes 2 caracteres corresponden al salto de lìnea (puede variar si MARS ejecuta en Linux o Mac)

Tenga en cuenta que

´0´ = 48_{10} = 0x30
´1´ = 49_{10} = 0x31
´2´ = 50_{10} = 0x32
´3´ = 51_{10} = 0x33
´4´ = 52_{10} = 0x34
´5´ = 53_{10} = 0x35
´6´ = 54_{10} = 0x36
´7´ = 55_{10} = 0x37
´8´ = 56_{10} = 0x38
´9´ = 57_{10} = 0x39
´a´ = 97_{10} = 0x61
´b´ = 98_{10} = 0x62
´c´ = 99_{10} = 0x63
´d´ = 100_{10} = 0x64
´e´ = 101_{10} = 0x65
´f´ = 102_{10} = 0x66

Por ejemplo, al leer la línea que contiene 2402000d (obviando el salto de línea), ésta será almacenada en memoria como

← 32 bits →			
32	30	34	32
64	30	30	30

Y su programa deberá transformarla a

← 32 bits →			
24	02	00	00

← 32 bits →																															
0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
2				4				0				2				0				0				0				d			

3.3 Resultado

El resultado de esta fase deberá ser almacenado en el área de datos. Para ello se sugiere que reserve de forma estática un espacio de 100 palabras, es decir 400 bytes. Asumiremos que los programas a ser ejecutados por MVML tienen menos de 100 líneas de código. Para facilitar la corrección y poder continuar con las siguientes fases del proyecto, identifique este espacio con la etiqueta programa.

```
.data
.align 2        # en caso de que antes de programa tenga otras
                reservas de espacio de memoria
programa: .space 400
```

4. Segunda Fase de MLMV:
Decodificación de la Instrucción

4.1 Objetivos Específicos

- Adquirir destrezas en programación con el lenguaje ensamblador MIPS
- Adquirir destrezas en la manipulación de bits a nivel de lenguaje ensamblador
- Adquirir destrezas en el uso de estructuras de datos a nivel de lenguaje ensamblador

4.2 Detalles de Implementación

En esta fase deberá implementar un programa en MIPS (en las siguientes fases deberá transformarlo en función) que obtenga cada una de las instrucciones almacenadas a partir de la dirección **programa** y la decodifique con la finalidad de obtener el código de la operación, el tipo de formato asociado con la misma y la identificación de los operando involucrados.

Por ejemplo:

```
.data
programa: .word 0x80a4500  0x3464000a
```

El resultado esperado es:

```
0x80a45000    R   add $10 $5 $4
0x3464000a    I    ori $4 $3 10
```

Cómo se obtiene?

8				0				a				4				5				0				0				0			
1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

co-op	rs	rt	rd		

Instrucción		Código operación		Syntax	Operation
add	R	32	100000	add $rd, $rs, $rt	$rd = $rs + $rt
addi	I	8	001000	add $rt, $rs, Offset	$rt= $rs + ExtSigno(Offset)
and	R	40	101000	and $rd, $rs, $rt	$rd = $rs & $rt
andi	I	12	001100	andi $rs, $rt, Offset	$rt = $s & ExtCero(Offset)
mult	R	24	011000	mult $rd, $rs, $rt	$rd = $rs * $rt
or	R	37	100101	or $rd, $rs, $rt	$rd = $rs | $rt
ori	I	13	001101	ori $rt, $rs, Offset	$rt = $rs | ExtCero(Offset)
sllv	R	4	000100	sllv $rd, $rt, $rs	$rd = $rt << $rs
sub	R	34	100010	sub $rd, $rs, $rt	$rd = $rs - $rt
lw	I	35	100011	lw $rt, Offset($rs)	$rt = Mem[$rs+ExtSigno(Offset)
sw	I	43	101011	sw $rt, Offset($rs)	Mem[$rs+ExtSigno(Offset)= $rt
bne	I	5	000101	bne $rs, $rt, Offset	
beq	I	6	000110	beq $rs, $rt, Offset	
halt	R	0	000000	halt	Detiene ejecución