

INSTITUTO POLITÉCNICO NACIONAL



CENTRO DE INVESTIGACIÓN Y  
DESARROLLO TECNOLÓGICO EN  
CÓMPUTO

SIMULADOR A NIVEL MICRO ARQUITECTURA DE  
UN PROCESADOR RISC DE 32-BITS

DIPLOMADO

PROGRAMACIÓN ORIENTADA A OBJETOS

*P R E S E N T A*  
*MIGUEL ANGEL RODRÍGUEZ FUENTES*

MÉXICO, D.F., JUNIO 2010

# Índice

---

## Índice

Resumen	ii
Introducción	1
Capítulo 1 La Ruta de Datos	3
1.1 Fundamentos de una micro arquitectura	4
1.2 Ruta de Datos	4
1.2.1 Sistema de Registros	4
1.2.2 Buses de Entrada y Salida	5
1.2.3 Diagrama de clases del Sistema de Registro	6
1.2.4 Código en java del Sistema de Registro	6
1.2.5 Unidad Lógico Aritmética (ALU)	10
1.2.6 Diagrama de clases de la ALU	11
1.2.7 Código en java de la ALU	11
Capítulo 2 La Unidad de Control	13
2.1 Registro de Control de micro instrucciones (MIR)	13
2.2 Memoria de Control	15
2.3 Diagrama de clases de la MIR	16
2.4 Código en java de la MIR y la Memoria de Control	17
Capítulo 3 La Memoria Principal y la integración del procesador	23
3.1 La Memoria Principal	24
3.1.1 Configuración <i>Big-Endian</i>	24
3.1.2 Operaciones de Lectura y Escritura	24
3.1.3 Diagrama de clases de la Memoria Principal	25
3.1.4 Código en java de la Memoria Principal	25
3.2 Integración y pruebas	27
3.3 Diagrama de clases del simulador	29
3.4 ¿Qué sucede cuando se oprime el botón Ejecutar Microinstrucción	30
3.5 Código en java del botón Ejecutar Microinstrucción	30
3.6 Programa para pruebas del simulador	35
Conclusiones y recomendaciones	37
Referencias	38
Anexo	39

# Resumen

---

## Resumen

El objetivo principal de la tesina es el desarrollo de un simulador a nivel micro arquitectura de un microprocesador tipo RISC de 32-bits, basado en la arquitectura de los procesadores SPARC de *Sun MicroSystems*. El trabajo incluye el desarrollo del simulador en la plataforma java, y cuyo propósito es su uso como herramienta didáctica de apoyo a la asignatura de Arquitectura de Computadoras. El simulador se basa en el procesador ARC (A RISC Computer) presentado en el libro de Principios de Arquitectura de Computadoras de Miles Murdocca y Vincen Heuring.

El programa fue desarrollado utilizando la versión gratuita de NetBeans IDE 6.8. Se presenta las partes que constituyen a un procesador a nivel micro arquitectura, los programas en java de dichas partes, así como la forma en cómo utilizar el simulador.

Finalmente se exponen las conclusiones y recomendaciones.

El trabajo se presenta como proyecto final del Diplomado en Programación Orientada a Objetos cursado dentro del programa de año sabático.

# Introducción

---

## Introducción

La tesina se presenta como un ejemplo de aplicación de los conocimientos adquiridos en el diplomado en Programación Orientada a Objetos, para desarrollar un simulador didáctico de un procesador a nivel micro arquitectura.

Se parte de un procesador de 32-bits denominado ARC, cuyo set de instrucciones (ensamblador), formato de instrucciones y arquitectura interna se explican detalladamente en el libro de Principios de Arquitectura de Computadoras de Miles Murdocca y Vincent Heuring.

La importancia del trabajo realizado, es contar con una herramienta de simulación para explicar dentro de la Asignatura de Arquitectura de Computadoras a nivel superior del Instituto Politécnico Nacional, el tema correspondiente al nivel micro arquitectura de un procesador. Es sumamente importante que el alumno pueda comprender que el procesador no es una caja negra, sino que es un circuito lógico muy grande, con muchas partes, pero que trabajan de manera coordinada y controlada en la ejecución de un programa para la solución de un problema numérico.

Si bien no existe un procesador universal, y por lo tanto cada fabricante tiene su propio lenguaje y arquitectura, es imposible estudiar todos dentro de una asignatura, pero al estudiar a detalle un procesador ejemplo y entenderlo a diversos niveles de abstracción, el alumno podrá extraer sus conocimientos en el entendimiento y aplicación de cualquier otro procesador, incluso podrá descubrir que puede diseñar su propio procesador, con su propio lenguaje y su propia arquitectura, al igual que lo han hecho innumerables compañías a nivel mundial.

La experiencia como docente de ésta asignatura en la UPIITA del IPN, aunado a los conocimientos adquiridos, permitieron el desarrollo del presente simulador.

Para el desarrollo del programa se siguieron 3 etapas:

1. Desarrollo de los elementos internos del procesador. Esto implicó desarrollar los programas de las partes fundamentales del procesador, lo que corresponde a la Ruta de Datos, donde encontramos a los registros y a la Unidad Lógico Aritmética (ALU), y el desarrollo de la Unidad de Control.
2. Desarrollo de la memoria principal. El procesador se basa en el concepto de programa almacenado, por lo cual la memoria es el elemento fundamental con el cual interactúa el procesador.
3. La integración de todas las partes del procesador y la memoria, y la transferencia de información entre las mismas.

Para describir lo anterior, el contenido de la tesina se presenta en los siguientes Capítulos:

El Capítulo 1, denominado **La Ruta de los Datos**, presenta los fundamentos de la micro arquitectura, como lo es el ciclo búsqueda-ejecución, el sistema de registros, los buses de entrada y salida asociados al sistema de registros, y la ALU. Se incluye el diagrama de clases y el código en java de los elementos.

## Introducción

---

El Capítulo 2, denominado **La Unidad de Control**, presenta dos elementos fundamentales de cualquier procesador, el registro de control de micro instrucciones (MIR) y la memoria de control. Se agrega el diagrama de clase y el código en java.

El Capítulo 3, denominado **La Memoria Principal y la integración del procesador**, presenta el desarrollo de la memoria principal, las operaciones de lectura y escritura, y como se guardan los datos en memoria al ser un procesador de 32-bits. Se muestra el diagrama de clases y código en java de la memoria. La segunda parte corresponde a la integración de todos los elementos anteriormente descritos para llegar al programa final del simulador.

En la parte final, se presentan las **Conclusiones** del trabajo y **recomendaciones**, así como las **Referencias** consultadas. En la parte denominada **Anexo**, se presenta un ejemplo del uso del simulador.

# Capítulo 1

# La Ruta de los Datos

# Capítulo 1. La Ruta de los Datos

## 1. La Ruta de los Datos

### 1.1 Fundamentos de una micro arquitectura

El funcionamiento de una micro arquitectura se basa en la realización del *llamado ciclo búsqueda – ejecución*, consistente en 5 pasos que ejecuta la Unidad de Control:

1. Busca en la memoria principal la siguiente instrucción a ejecutarse.
2. Decodifica el código de operación.
3. Si es necesario, lee de la memoria principal el operando u operandos y los almacena en los registros.
4. Ejecuta la instrucción y almacena los resultados.
5. Regresa al paso 1.

La micro arquitectura del procesador se constituye por dos partes fundamentales, la Ruta de los Datos (*Datapath*), y la Unidad de Control (*Control Unit*).

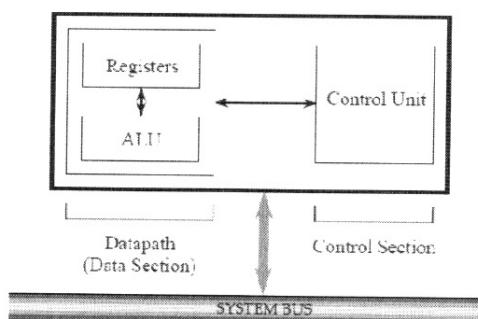


Figura 1.1 Vista general de un procesador.

## 1.2 Ruta de Datos

### 1.2.1 Sistema de Registros

El sistema de registros está constituido por 38 registros de 32-bits cada uno, los registros %r0 a %r31 son visibles al usuario y son de propósito general. El registro %r0 tiene la peculiaridad de contener siempre 0. El registro 32 (%pc – *program counter*), se utiliza como una apuntador a la localidad de memoria principal donde se encuentran la instrucción a ejecutarse. Los registros temporales (%tmp 0 a %tmp3, registros 33 a 36) los utiliza el procesador como auxiliares en las tareas de ejecución de las microinstrucciones. El último registro (%ir- *instruction register*, registro 37) se emplea para almacenar la instrucción a ejecutar por parte del procesador. El registro %ir se conecta con la Unidad de Control, es por eso que aparecen varios campos correspondientes a este registro, como son op, rd, op3, rs1, rs2, simm13.

# Capítulo 1. La Ruta de los Datos

%r0= 0	%r1= 00000000	%r2= 00000000	%r3= 00000000					
%r4= 00000000	%r5= 00000000	%r6= 00000000	%r7= 00000000					
%r8= 00000000	%r9= 00000000	%r10= 00000000	%r11= 00000000					
%r12= 00000000	%r13= 00000000	%r14= 00000000	%r15= 00000000					
%r16= 00000000	%r17= 00000000	%r18= 00000000	%r19= 00000000					
%r20= 00000000	%r21= 00000000	%r22= 00000000	%r23= 00000000					
%r24= 00000000	%r25= 00000000	%r26= 00000000	%r27= 00000000					
%r28= 00000000	%r29= 00000000	%r30= 00000000	%r31= 00000000					
%tm0= 00000000	%tm1= 00000000	%tm2= 00000000	%tm3= 00000000					
%pc= 00000000	%ir= 00000000							
Area de trabajo		op	rd	op3	rs1	i	rs2	simm13
		0	0	0	0	0	0	

Figura 1.2 Imagen del sistema de registros del procesador.

## 1.2.2 Buses de entrada y salida

El procesador cuenta con 2 buses de salida, denominados Bus A (color verde) y el Bus B (color azul), ambos de 32-bits. Cualquiera de los registros puede colocar su contenido en cualquiera de los dos buses. La información del Bus A puede ser utilizada para dos tareas diferentes, una es servir como dato de entrada de la ALU o bien, representar una dirección de la memoria principal donde se desea leer o escribir un dato.

El Bus B también puede servir como entrada de la ALU o bien puede contener la información que se desea escribir en la memoria principal.

La información del Bus C (color rojo) es de entrada, es decir, la información del bus puede guardarse en cualquiera de los 38 registros. Como se ve en la imagen, la información del Bus C puede tener dos orígenes, uno que sea el resultado de una operación lógica o aritmética cuyo valor se va a guardar en un registro, o bien, es el contenido de una localidad de memoria que se lee y se desea guardar en registro.

La selección de un registro en particular que coloque sus datos en el Bus A o el Bus B, o la información del Bus C se almacene en algún registro está determinado por la Unidad de Control, como más adelante se explicará.

# Capítulo 1. La Ruta de los Datos

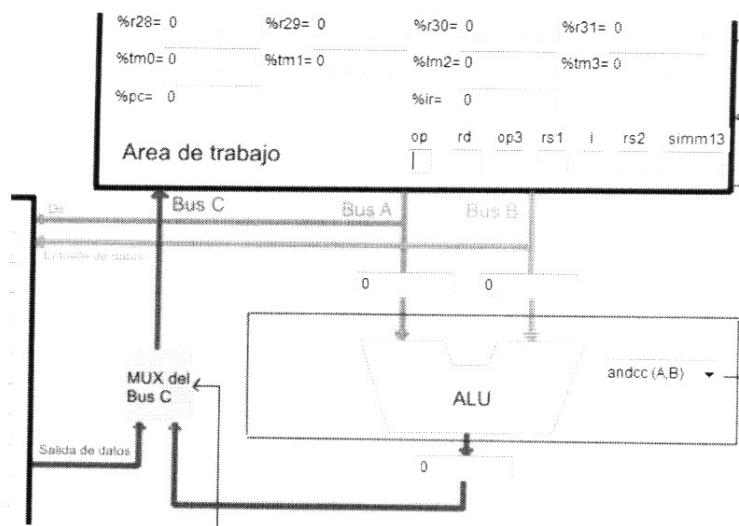


Figura 1.3 Imagen del sistema de buses del procesador.

## 1.2.3 Diagrama de clases del sistema de registros



Figura 1.4 Diagrama del clases del sistema de registros

## 1.2.4 Código Java del sistema de registros

```
//Objetos
private TextField[] registro;
private Label[] etiqueta;
private TextField irop, irrd, irop3, irrs1, iri, irrs2, irsimm13;
private Label labir;

//Inicialización
public void init() {
    registro = new TextField[38];
    etiqueta = new Label[38];
    irop = new TextField(2);
    irrd = new TextField(2);
    irop3 = new TextField(2);
    irrs1 = new TextField(2);
    iri = new TextField(1);
    irrs2 = new TextField(2);
    irsimm13 = new TextField(4);
```

## Capítulo 1. La Ruta de los Datos

```
labir = new Label();

for (int i = 0; i < 38; i++) {
    registro[i] = new TextField(8);
    registro[i].setSize(80, 20);
    registro[i].addKeyListener(this);
    etiqueta[i] = new Label();
    etiqueta[i].setSize(40, 20);
}

int x = 300;
int y = 10;
registro[0].setLocation(x + 0, y + 0);
registro[1].setLocation(x + 120, y + 0);
registro[2].setLocation(x + 240, y + 0);
registro[3].setLocation(x + 360, y + 0);

etiqueta[0].setText("%r0=");
etiqueta[0].setLocation(x - 35, y + 0);
etiqueta[1].setText("%r1=");
etiqueta[1].setLocation(x + 85, y + 0);
etiqueta[2].setText("%r2=");
etiqueta[2].setLocation(x + 205, y + 0);
etiqueta[3].setText("%r3=");
etiqueta[3].setLocation(x + 325, y + 0);

registro[4].setLocation(x + 0, y + 30);
registro[5].setLocation(x + 120, y + 30);
registro[6].setLocation(x + 240, y + 30);
registro[7].setLocation(x + 360, y + 30);

etiqueta[4].setText("%r4=");
etiqueta[4].setLocation(x - 35, y + 30);
etiqueta[5].setText("%r5=");
etiqueta[5].setLocation(x + 85, y + 30);
etiqueta[6].setText("%r6=");
etiqueta[6].setLocation(x + 205, y + 30);
etiqueta[7].setText("%r7=");
etiqueta[7].setLocation(x + 325, y + 30);

registro[8].setLocation(x + 0, y + 60);
registro[9].setLocation(x + 120, y + 60);
registro[10].setLocation(x + 240, y + 60);
registro[11].setLocation(x + 360, y + 60);

etiqueta[8].setText("%r8=");
etiqueta[8].setLocation(x - 40, y + 60);
etiqueta[9].setText("%r9=");
etiqueta[9].setLocation(x + 80, y + 60);
etiqueta[10].setText("%r10=");
etiqueta[10].setLocation(x + 200, y + 60);
etiqueta[11].setText("%r11=");
etiqueta[11].setLocation(x + 320, y + 60);

registro[12].setLocation(x + 0, y + 90);
registro[13].setLocation(x + 120, y + 90);
registro[14].setLocation(x + 240, y + 90);
registro[15].setLocation(x + 360, y + 90);

etiqueta[12].setText("%r12=");
etiqueta[12].setLocation(x - 40, y + 90);
etiqueta[13].setText("%r13=");
etiqueta[13].setLocation(x + 80, y + 90);
etiqueta[14].setText("%r14=");
etiqueta[14].setLocation(x + 200, y + 90);
etiqueta[15].setText("%r15=");
etiqueta[15].setLocation(x + 320, y + 90);

registro[16].setLocation(x + 0, y + 120);
registro[17].setLocation(x + 120, y + 120);
```

## Capítulo 1. La Ruta de los Datos

```
registro[18].setLocation(x + 240, y + 120);
registro[19].setLocation(x + 360, y + 120);

etiqueta[16].setText("%r16=");
etiqueta[16].setLocation(x - 40, y + 120);
etiqueta[17].setText("%r17=");
etiqueta[17].setLocation(x + 80, y + 120);
etiqueta[18].setText("%r18=");
etiqueta[18].setLocation(x + 200, y + 120);
etiqueta[19].setText("%r19=");
etiqueta[19].setLocation(x + 320, y + 120);

registro[20].setLocation(x + 0, y + 150);
registro[21].setLocation(x + 120, y + 150);
registro[22].setLocation(x + 240, y + 150);
registro[23].setLocation(x + 360, y + 150);

etiqueta[20].setText("%r20=");
etiqueta[20].setLocation(x - 40, y + 150);
etiqueta[21].setText("%r21=");
etiqueta[21].setLocation(x + 80, y + 150);
etiqueta[22].setText("%r22=");
etiqueta[22].setLocation(x + 200, y + 150);
etiqueta[23].setText("%r23=");
etiqueta[23].setLocation(x + 320, y + 150);

registro[24].setLocation(x + 0, y + 180);
registro[25].setLocation(x + 120, y + 180);
registro[26].setLocation(x + 240, y + 180);
registro[27].setLocation(x + 360, y + 180);

etiqueta[24].setText("%r24=");
etiqueta[24].setLocation(x - 40, y + 180);
etiqueta[25].setText("%r25=");
etiqueta[25].setLocation(x + 80, y + 180);
etiqueta[26].setText("%r26=");
etiqueta[26].setLocation(x + 200, y + 180);
etiqueta[27].setText("%r27=");
etiqueta[27].setLocation(x + 320, y + 180);

registro[28].setLocation(x + 0, y + 210);
registro[29].setLocation(x + 120, y + 210);
registro[30].setLocation(x + 240, y + 210);
registro[31].setLocation(x + 360, y + 210);

etiqueta[28].setText("%r28=");
etiqueta[28].setLocation(x - 40, y + 210);
etiqueta[29].setText("%r29=");
etiqueta[29].setLocation(x + 80, y + 210);
etiqueta[30].setText("%r30=");
etiqueta[30].setLocation(x + 200, y + 210);
etiqueta[31].setText("%r31=");
etiqueta[31].setLocation(x + 320, y + 210);

registro[32].setLocation(x + 0, y + 270); //%pc
registro[33].setLocation(x + 0, y + 240);
registro[34].setLocation(x + 120, y + 240);
registro[35].setLocation(x + 240, y + 240);

etiqueta[32].setText("%tm0=");
etiqueta[32].setLocation(x - 40, y + 240);
etiqueta[33].setText("%tm1=");
etiqueta[33].setLocation(x + 80, y + 240);
etiqueta[34].setText("%tm2=");
etiqueta[34].setLocation(x + 200, y + 240);
etiqueta[35].setText("%tm3=");
etiqueta[35].setLocation(x + 320, y + 240);

registro[36].setLocation(x + 360, y + 240);
registro[37].setLocation(x + 240, y + 270);
```

## Capítulo 1. La Ruta de los Datos

```
etiqueta[36].setText("%pc=");
etiqueta[36].setLocation(x - 40, y + 270);
etiqueta[37].setText("%ir=");
etiqueta[37].setLocation(x + 200, y + 270);

irop.setSize(20, 20);
irop.setLocation(x + 200, y + 320);
add(irop);
irop.addKeyListener(this);
irrd.setSize(25, 20);
irrd.setLocation(x + 235, y + 320);
add(irrd);
irrd.addKeyListener(this);
irop3.setSize(25, 20);
irop3.setLocation(x + 270, y + 320);
add(irop3);
irop3.addKeyListener(this);
irrs1.setSize(25, 20);
irrs1.setLocation(x + 305, y + 320);
add(irrs1);
irrs1.addKeyListener(this);
iri.setSize(20, 20);
iri.setLocation(x + 340, y + 320);
add(iri);
iri.addKeyListener(this);
irrs2.setSize(25, 20);
irrs2.setLocation(x + 370, y + 320);
add(irrs2);
irrs2.addKeyListener(this);
irsimm13.setSize(50, 20);
irsimm13.setLocation(x + 405, y + 320);
add(irsimm13);
irsimm13.addKeyListener(this);

labir.setLocation(x + 200, y + 300);
labir.setSize(260, 20);
labir.setText("op      rd      op3      rs1      i      rs2      simm13");
add(labir);

for (int i = 0; i < 38; i++) {
    registro[i].setText("0");
    add(registro[i]);
    add(etiqueta[i]);
}
}
```

## Capítulo 1. La Ruta de los Datos

### 1.2.5 ALU (Unidad Lógica Aritmética)

La ALU del procesador tiene 16 operaciones. La selección de una operación en particular la determina la Unidad de Control.

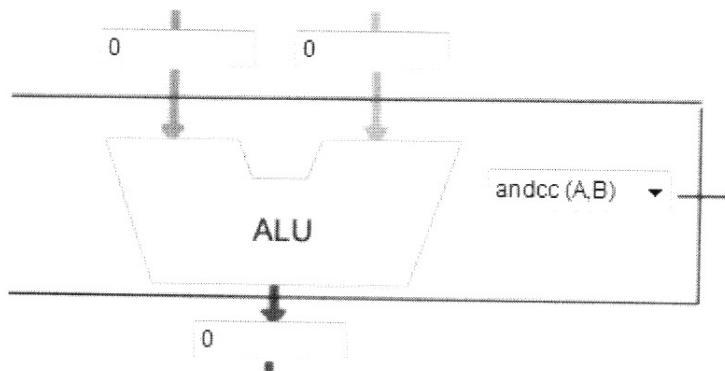


Figura 1.5 Imagen de la ALU del procesador y la operación que realiza.

Las 16 operaciones de la ALU se muestran en la Tabla 1.1.

Número	Operación	Descripción
0	ANDCC (A, B)*	AND lógica entre el Bus A y el Bus B.
1	ORCC (A, B)*	OR lógica entre el Bus A y el Bus B.
2	NORCC (A, B)*	NOR lógica entre el Bus A y el Bus B.
3	ADDCC (A, B)*	Suma el dato del Bus A mas el del Bus B.
4	SRL (A, B)	Corrimiento del Bus A a la derecha el número de veces especificado por el Bus B.
5	AND (A, B)	AND lógica entre el Bus A y el Bus B.
6	OR (A, B)	OR lógica entre el Bus A y el Bus B.
7	NOR (A, B)	NOR lógica entre el Bus A y el Bus B.
8	ADD (A, B)	Suma el dato del Bus A más el del Bus B.
9	LSHIFT2 (A)	Corrimiento del Bus A dos bits a la izquierda.
10	LSHIFT10 (A)	Corrimiento del Bus A diez bits a la izquierda.
11	SIMM13 (A)	Recupera los 13 bits menos significativos del Bus A y coloca ceros en los 19 bits más significativos.
12	SEXT13 (A)	Extensión de signo de los 13 bits menos significativos del Bus A. Si el bit 13 es cero, coloca ceros en los 19 bits más significativos, si es uno coloca unos en los 19 bits más significativos.
13	INC (A)	Incrementa el valor del Bus A en uno.
14	INCPC (A)	Incrementa el valor del Bus A en cuatro.
15	RSHIFT5 (A)	Corrimiento del valor del Bus A 5 bits a la derecha.

(\*) el sufijo CC corresponde a Condition Codes (Códigos de Condición), es decir, estas instrucciones modifican registro del estado de las banderas, aunque en el caso de la versión del simulador, no se han utilizado el registro de banderas.

Tabla 1.1 Operaciones de la ALU.

## Capítulo 1. La Ruta de los Datos

### 1.2.6 Diagrama de clases de la ALU



Figura 1.6 Diagrama de clases de la ALU.

### 1.2.7 Código en Java de la ALU

```
//Objetos
private TextField busa, busb, busc;
private Choice op;
private Image aluimg;

//Inicialización
public void init(){
busa = new TextField(10);
busa.setSize(80, 20);
busa.setLocation(100 + 360, 30 + 400);
add(busa);
busa.addKeyListener(this);
busa.setText("0");

busb = new TextField(10);
busb.setSize(80, 20);
busb.setLocation(200 + 360, 30 + 400);
add(busb);
busb.addKeyListener(this);
busb.setText("0");

busc = new TextField(10);
busc.setSize(80, 20);
busc.setLocation(150 + 360, 180 + 400);
add(busc);
busc.addKeyListener(this);
busc.setText("0");

op = new Choice();
op.setLocation(300 + 360, 100 + 400);
op.addItem("andcc (A,B)");
op.addItem("orcc (A,B)");
op.addItem("norcc (A,B)");
op.addItem("addcc (A,B) ");
op.addItem("srl (A,B) ");
op.addItem("and (A,B) ");
op.addItem("or (A,B) ");
op.addItem("nor (A,B) ");
op.addItem("add (A,B) ");
op.addItem("lshift2 (A) ");
op.addItem("lshift10 (A) ");
```

## Capítulo 1. La Ruta de los Datos

```
op.addItem("simm13 (A) ");
op.addItem("sext13 (A) ");
op.addItem("inc (A) ");
op.addItem("incpc (A) ");
op.addItem("rshift5 (A) ");
add(op);
op.addItemListener(this);
}

class operacion { //alu

    long alua, alub, aluc, aux, aux1;

    operacion() {

        //Lee valores del bus A y del bus B
        alua = Long.parseLong(busa.getText());
        alub = Long.parseLong(busb.getText());


        //operacion alu
        op.select(Integer.parseInt(alu.getText()));
        aux = op.getSelectedIndex();
        if (aux == 0) { //andcc
            aluc = alua & alub;
        } else if (aux == 1) { //orc
            aluc = alua | alub;
        } else if (aux == 2) { //norcc
            aluc = ~(alua | alub);
        } else if (aux == 3) { //addcc
            aluc = alua + alub;
        } else if (aux == 4) { //srl
            aluc = alua >> alub;
        } else if (aux == 5) { //and
            aluc = alua & alub;
        } else if (aux == 6) { //or
            aluc = alua | alub;
        } else if (aux == 7) { //nor
            aluc = ~(alua + alub);
        } else if (aux == 8) { //add
            aluc = alua + alub;
        } else if (aux == 9) { //lshift2
            aluc = alua << 2;
        } else if (aux == 10) { //lshift10
            aluc = alua << 10;
        } else if (aux == 11) { //simm13
            aluc = alua & 0x00001FFF;
        } else if (aux == 12) { //sext13
            aluc = alua & 0x00001FFF;
            aux1 = aluc & 0x00001000;
            if (aux1 == 0x00001000) //si es negativo llena con ls a la izq
                aluc = aluc | 0x1111E000;
        }
        } else if (aux == 13) { //inc
            aluc = alua + 1;
        } else if (aux == 14) { //incpc
            aluc = alua + 4;
        } else if (aux == 15) { //rshift5
            aluc = alua >> 5;
        }

        //impression de resultados alu
        busa.setText(Long.toString(alua));
        busb.setText(Long.toString(alub));
        busc.setText(Long.toString(aluc));
    }
}
```

# **Capítulo 2**

# **La Unidad de Control**

## Capítulo 2. La Unidad de Control

### 2. La Unidad de Control

#### 2.1 Registro de control de microinstrucciones (MIR)

El MIR está formado por 44 bits, y es el encargado de determinar la configuración de todo el procesador para poder ejecutar una micro instrucción.

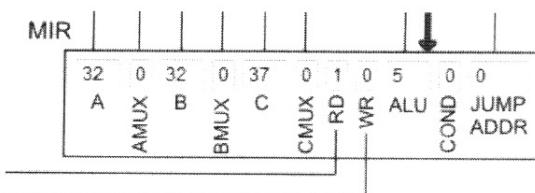


Figura 2.1 Vista del registro MIR del procesador.

Tiene 11 campos cuyas funciones son las siguientes:

Nombre	N. de Bits	Función
A	6	Determina cual registro coloca sus datos en el Bus A, siempre y cuando AMUX=0.
AMUX	1	Si AMUX =0, el valor definido en el campo A del MIR determina cual de los registros coloca su dato en el Bus A, si AMUX = 1, el campo rs1 de %ir determina cual de los registros coloca su dato en el Bus A.
B	6	Determina cual registro coloca sus datos en el Bus B, siempre y cuando BMUX=0.
BMUX	1	Si BMUX =0, el valor definido en el campo B del MIR determina cual de los registros coloca su dato en el Bus B, si BMUX = 1, el campo rs2 de %ir determina cual de los registros coloca su dato en el Bus B.
C	6	Determina en cual registro se guarda la información del Bus C, siempre y cuando CMUX=0.
CMUX	1	Si CMUX=0, la información contenida por el Bus C se guarda en el registro especificado por el campo C de MIR, si CMUX=1, entonces la información se guarda en el registro especificado por el campo rd de %ir.
RD	1	Si RD=1, se realiza una lectura a la memoria principal. La dirección de la localidad de memoria a leer se especifica por el dato contenido en el Bus A.
WR	1	Si WR=1, se realiza una escritura en la memoria principal. La dirección a escribir la determina el dato contenido en el Bus A, y en el Bus B se encuentra el dato por escribir en dicha localidad.
ALU	4	Especifica una de las dieciséis operaciones que puede realizar la ALU.
COND	3	Determina cual es la siguiente localidad de la memoria de control por ejecutar, es decir, controla el MUX de la memoria de control. Tiene varias posibilidades:

## Capítulo 2. La Unidad de Control

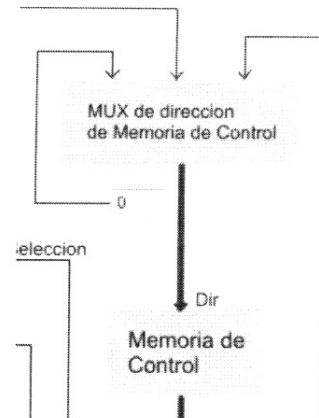


Figura 2.2 Determinación de la dirección de la memoria de control.

Si COND=0, se utiliza la siguiente dirección.

Si COND=1 a 4, la siguiente localidad la especifica el campo JUMP ADDRESS del MIR.

Si COND=5 y el campo  $i=1$  de  $\%ir$  la siguiente localidad la especifica el campo JUMP ADDRESS del MIR, si el campo  $i=0$  de  $\%ir$ , se utiliza la siguiente localidad.

Si COND=6, la siguiente localidad la especifica el campo JUMP ADDRESS del MIR.

Si COND=7, la siguiente localidad se especifica al llenar los siguientes campos del DECODE con valores del registro  $\%ir$  y determinar su valor numérico

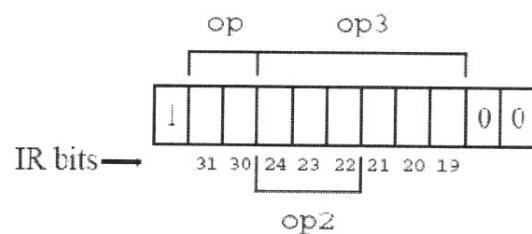


Figura 2.3 Operación DECODE.

JUMP ADDRESS	11	Especifica una localidad de la memoria de control.
--------------	----	--

Tabla 2.1 Campos del MIR.

## Capítulo 2. La Unidad de Control

---

### 2.2 Memoria de Control

Contiene en su interior la información con la cual se llenan los campos del MIR. Para el caso del simulador, se han implementado solamente las siguientes operaciones:

Localidades	Instrucción
0	Lee instrucción de memoria principal.
1	DECODE
1600 - 1603	addcc
1604 - 1607	andcc
1608 - 1611	orcc
1624 - 1627	norcc
1688 - 1691	srl
1792 - 1795	ld
1808 - 1811	st
2047	Incpcc

Tabla 2.2 Localidades de las micro instrucciones.

El detalle de cada una de las instrucciones implementadas el simulador se muestra en la tabla 2.3. Nótese que dependiendo de cada instrucción se requiere una o más micro instrucciones para llevarla a cabo.

LOC	A	AMUX	B	BMUX	C	CMUX	RD	WR	ALU	COND	JUMPADDR
0	32	0	32	0	37	0	1	0	5	0	0
1	0	0	0	0	0	0	0	0	5	7	0
1600	0	0	0	0	0	0	0	0	5	5	1602
1601	0	1	0	1	0	1	0	0	3	6	2047
1602	37	0	0	0	33	0	0	0	12	0	0
1603	0	1	33	0	0	1	0	0	3	6	2047
1604	0	0	0	0	0	0	0	0	5	5	1606
1605	0	1	0	1	0	1	0	0	0	6	2047
1606	37	0	0	0	33	0	0	0	11	0	0
1607	0	1	33	0	0	1	0	0	0	6	2047
1608	0	0	0	0	0	0	0	0	5	5	1610
1609	0	1	0	1	0	1	0	0	1	6	2047
1610	37	0	0	0	33	0	0	0	11	0	0
1611	0	1	33	0	0	1	0	0	1	6	2047
1624	0	0	0	0	0	0	0	0	5	5	1626
1625	0	1	0	1	0	1	0	0	2	6	2047
1626	37	0	0	0	33	0	0	0	11	0	0
1627	0	1	33	0	0	1	0	0	2	6	2047
1688	0	0	0	0	0	0	0	0	5	0	1690
1689	0	1	0	1	0	1	0	0	4	6	2047

## Capítulo 2. La Unidad de Control

1690	37	0	0	0	33	0	0	0	11	0	0
1691	0	1	33	0	0	1	0	0	4	6	2047
1792	0	1	0	1	33	0	0	0	8	5	1794
1793	33	0	33	0	0	1	1	0	5	6	2047
1794	37	0	0	0	33	0	0	0	12	0	0
1795	0	1	33	0	33	0	0	0	8	6	1793
1808	0	1	0	1	33	0	0	0	8	5	1810
1809	37	0	0	0	37	0	0	0	15	6	40
40	37	0	0	0	37	0	0	0	15	0	0
41	37	0	0	0	37	0	0	0	15	0	0
42	37	0	0	0	37	0	0	0	15	0	0
43	37	0	0	0	37	0	0	0	15	0	0
44	33	1	0	1	0	0	0	1	5	6	2047
1810	37	0	0	0	33	0	0	0	12	0	0
1811	0	1	33	0	33	0	0	0	8	6	1809
2047	32	0	0	0	32	0	0	0	14	6	0

Tabla 2.3 Contenido de la memoria de control.

### 2.3 Diagrama de clases del MIR



Figura 2.4 Diagrama de clases del MIR y la memoria de control.

## Capítulo 2. La Unidad de Control

### 2.4 Código en Java del MIR y la memoria de control

```
//mir y memoria control
private Button calc;
private TextField a, amux, b, bmux, c, cmux, rd, wr, alu, cond, jumpaddr, loc, dirmca;
private int[][] cs = new int[2048][11]; //memoria de control

//mir y mem control
public void init(){
    //loc 0 lee inst de mem
    cs[0][0] = 32;
    cs[0][1] = 0;
    cs[0][2] = 32;
    cs[0][3] = 0;
    cs[0][4] = 37;
    cs[0][5] = 0;
    cs[0][6] = 1;
    cs[0][7] = 0;
    cs[0][8] = 5;
    cs[0][9] = 0;
    cs[0][10] = 0;
    //loc 1 decode
    cs[1][0] = 0;
    cs[1][1] = 0;
    cs[1][2] = 0;
    cs[1][3] = 0;
    cs[1][4] = 0;
    cs[1][5] = 0;
    cs[1][6] = 0;
    cs[1][7] = 0;
    cs[1][8] = 5;
    cs[1][9] = 7;
    cs[1][10] = 0;
    //addcc
    cs[1600][0] = 0;
    cs[1600][1] = 0;
    cs[1600][2] = 0;
    cs[1600][3] = 0;
    cs[1600][4] = 0;
    cs[1600][5] = 0;
    cs[1600][6] = 0;
    cs[1600][7] = 0;
    cs[1600][8] = 5;
    cs[1600][9] = 5;
    cs[1600][10] = 1602;
    cs[1601][0] = 0;
    cs[1601][1] = 1;
    cs[1601][2] = 0;
    cs[1601][3] = 1;
    cs[1601][4] = 0;
    cs[1601][5] = 1;
    cs[1601][6] = 0;
    cs[1601][7] = 0;
    cs[1601][8] = 3;
    cs[1601][9] = 6;
    cs[1601][10] = 2047;
    cs[1602][0] = 37;
    cs[1602][1] = 0;
    cs[1602][2] = 0;
    cs[1602][3] = 0;
    cs[1602][4] = 33;
    cs[1602][5] = 0;
    cs[1602][6] = 0;
    cs[1602][7] = 0;
    cs[1602][8] = 12;
    cs[1602][9] = 0;
    cs[1602][10] = 0;
    cs[1603][0] = 0;
    cs[1603][1] = 1;
    cs[1603][2] = 33;
```

## Capítulo 2. La Unidad de Control

---

```
cs[1603][3] = 0;
cs[1603][4] = 0;
cs[1603][5] = 1;
cs[1603][6] = 0;
cs[1603][7] = 0;
cs[1603][8] = 3;
cs[1603][9] = 6;
cs[1603][10] = 2047;
//andcc
cs[1604][0] = 0;
cs[1604][1] = 0;
cs[1604][2] = 0;
cs[1604][3] = 0;
cs[1604][4] = 0;
cs[1604][5] = 0;
cs[1604][6] = 0;
cs[1604][7] = 0;
cs[1604][8] = 5;
cs[1604][9] = 5;
cs[1604][10] = 1606;
cs[1605][0] = 0;
cs[1605][1] = 1;
cs[1605][2] = 0;
cs[1605][3] = 1;
cs[1605][4] = 0;
cs[1605][5] = 1;
cs[1605][6] = 0;
cs[1605][7] = 0;
cs[1605][8] = 0;
cs[1605][9] = 6;
cs[1605][10] = 2047;
cs[1606][0] = 37;
cs[1606][1] = 0;
cs[1606][2] = 0;
cs[1606][3] = 0;
cs[1606][4] = 33;
cs[1606][5] = 0;
cs[1606][6] = 0;
cs[1606][7] = 0;
cs[1606][8] = 11;
cs[1606][9] = 0;
cs[1606][10] = 0;
cs[1607][0] = 0;
cs[1607][1] = 1;
cs[1607][2] = 33;
cs[1607][3] = 0;
cs[1607][4] = 0;
cs[1607][5] = 1;
cs[1607][6] = 0;
cs[1607][7] = 0;
cs[1607][8] = 0;
cs[1607][9] = 6;
cs[1607][10] = 2047;
//or
cs[1608][0] = 0;
cs[1608][1] = 0;
cs[1608][2] = 0;
cs[1608][3] = 0;
cs[1608][4] = 0;
cs[1608][5] = 0;
cs[1608][6] = 0;
cs[1608][7] = 0;
cs[1608][8] = 5;
cs[1608][9] = 5;
cs[1608][10] = 1610;
cs[1609][0] = 0;
cs[1609][1] = 1;
cs[1609][2] = 0;
cs[1609][3] = 1;
cs[1609][4] = 0;
cs[1609][5] = 1;
```

## Capítulo 2. La Unidad de Control

```
cs[1609][6] = 0;
cs[1609][7] = 0;
cs[1609][8] = 1;
cs[1609][9] = 6;
cs[1609][10] = 2047;
cs[1610][0] = 37;
cs[1610][1] = 0;
cs[1610][2] = 0;
cs[1610][3] = 0;
cs[1610][4] = 33;
cs[1610][5] = 0;
cs[1610][6] = 0;
cs[1610][7] = 0;
cs[1610][8] = 11;
cs[1610][9] = 0;
cs[1610][10] = 0;
cs[1611][0] = 0;
cs[1611][1] = 1;
cs[1611][2] = 33;
cs[1611][3] = 0;
cs[1611][4] = 0;
cs[1611][5] = 1;
cs[1611][6] = 0;
cs[1611][7] = 0;
cs[1611][8] = 1;
cs[1611][9] = 6;
cs[1611][10] = 2047;
//nor
cs[1624][0] = 0;
cs[1624][1] = 0;
cs[1624][2] = 0;
cs[1624][3] = 0;
cs[1624][4] = 0;
cs[1624][5] = 0;
cs[1624][6] = 0;
cs[1624][7] = 0;
cs[1624][8] = 5;
cs[1624][9] = 5;
cs[1624][10] = 1626;
cs[1625][0] = 0;
cs[1625][1] = 1;
cs[1625][2] = 0;
cs[1625][3] = 1;
cs[1625][4] = 0;
cs[1625][5] = 1;
cs[1625][6] = 0;
cs[1625][7] = 0;
cs[1625][8] = 2;
cs[1625][9] = 6;
cs[1625][10] = 2047;
cs[1626][0] = 37;
cs[1626][1] = 0;
cs[1626][2] = 0;
cs[1626][3] = 0;
cs[1626][4] = 33;
cs[1626][5] = 0;
cs[1626][6] = 0;
cs[1626][7] = 0;
cs[1626][8] = 11;
cs[1626][9] = 0;
cs[1626][10] = 0;
cs[1627][0] = 0;
cs[1627][1] = 1;
cs[1627][2] = 33;
cs[1627][3] = 0;
cs[1627][4] = 0;
cs[1627][5] = 1;
cs[1627][6] = 0;
cs[1627][7] = 0;
cs[1627][8] = 2;
cs[1627][9] = 6;
```

## Capítulo 2. La Unidad de Control

```
cs[1627][10] = 2047;
//srl
cs[1688][0] = 0;
cs[1688][1] = 0;
cs[1688][2] = 0;
cs[1688][3] = 0;
cs[1688][4] = 0;
cs[1688][5] = 0;
cs[1688][6] = 0;
cs[1688][7] = 0;
cs[1688][8] = 5;
cs[1688][9] = 5;
cs[1688][10] = 1690;
cs[1689][0] = 0;
cs[1689][1] = 1;
cs[1689][2] = 0;
cs[1689][3] = 1;
cs[1689][4] = 0;
cs[1689][5] = 1;
cs[1689][6] = 0;
cs[1689][7] = 0;
cs[1689][8] = 4;
cs[1689][9] = 6;
cs[1689][10] = 2047;
cs[1690][0] = 37;
cs[1690][1] = 0;
cs[1690][2] = 0;
cs[1690][3] = 0;
cs[1690][4] = 33;
cs[1690][5] = 0;
cs[1690][6] = 0;
cs[1690][7] = 0;
cs[1690][8] = 11;
cs[1690][9] = 0;
cs[1690][10] = 0;
cs[1691][0] = 0;
cs[1691][1] = 1;
cs[1691][2] = 33;
cs[1691][3] = 0;
cs[1691][4] = 0;
cs[1691][5] = 1;
cs[1691][6] = 0;
cs[1691][7] = 0;
cs[1691][8] = 4;
cs[1691][9] = 6;
cs[1691][10] = 2047;

//ld
cs[1792][0] = 0;
cs[1792][1] = 1;
cs[1792][2] = 0;
cs[1792][3] = 1;
cs[1792][4] = 33;
cs[1792][5] = 0;
cs[1792][6] = 0;
cs[1792][7] = 0;
cs[1792][8] = 8;
cs[1792][9] = 5;
cs[1792][10] = 1794;
cs[1793][0] = 33;
cs[1793][1] = 0;
cs[1793][2] = 33;
cs[1793][3] = 0;
cs[1793][4] = 0;
cs[1793][5] = 1;
cs[1793][6] = 1;
cs[1793][7] = 0;
cs[1793][8] = 5;
cs[1793][9] = 6;
cs[1793][10] = 2047;
cs[1794][0] = 37;
```

## Capítulo 2. La Unidad de Control

---

```
cs[1794][1] = 0;
cs[1794][2] = 0;
cs[1794][3] = 0;
cs[1794][4] = 33;
cs[1794][5] = 0;
cs[1794][6] = 0;
cs[1794][7] = 0;
cs[1794][8] = 12;
cs[1794][9] = 0;
cs[1794][10] = 0;
cs[1795][0] = 0;
cs[1795][1] = 1;
cs[1795][2] = 33;
cs[1795][3] = 0;
cs[1795][4] = 33;
cs[1795][5] = 0;
cs[1795][6] = 0;
cs[1795][7] = 0;
cs[1795][8] = 8;
cs[1795][9] = 6;
cs[1795][10] = 1793;

//st
cs[1808][0] = 0;
cs[1808][1] = 1;
cs[1808][2] = 0;
cs[1808][3] = 1;
cs[1808][4] = 33;
cs[1808][5] = 0;
cs[1808][6] = 0;
cs[1808][7] = 0;
cs[1808][8] = 8;
cs[1808][9] = 5;
cs[1808][10] = 1810;
cs[1809][0] = 37;
cs[1809][1] = 0;
cs[1809][2] = 0;
cs[1809][3] = 0;
cs[1809][4] = 37;
cs[1809][5] = 0;
cs[1809][6] = 0;
cs[1809][7] = 0;
cs[1809][8] = 15;
cs[1809][9] = 6;
cs[1809][10] = 40;
cs[40][0] = 37;
cs[40][1] = 0;
cs[40][2] = 0;
cs[40][3] = 0;
cs[40][4] = 37;
cs[40][5] = 0;
cs[40][6] = 0;
cs[40][7] = 0;
cs[40][8] = 15;
cs[40][9] = 0;
cs[40][10] = 0;
cs[41][0] = 37;
cs[41][1] = 0;
cs[41][2] = 0;
cs[41][3] = 0;
cs[41][4] = 37;
cs[41][5] = 0;
cs[41][6] = 0;
cs[41][7] = 0;
cs[41][8] = 15;
cs[41][9] = 0;
cs[41][10] = 0;
cs[42][0] = 37;
cs[42][1] = 0;
cs[42][2] = 0;
cs[42][3] = 0;
```

## Capítulo 2. La Unidad de Control

```
cs[42][4] = 37;
cs[42][5] = 0;
cs[42][6] = 0;
cs[42][7] = 0;
cs[42][8] = 15;
cs[42][9] = 0;
cs[42][10] = 0;
cs[43][0] = 37;
cs[43][1] = 0;
cs[43][2] = 0;
cs[43][3] = 0;
cs[43][4] = 37;
cs[43][5] = 0;
cs[43][6] = 0;
cs[43][7] = 0;
cs[43][8] = 15;
cs[43][9] = 0;
cs[43][10] = 0;
cs[44][0] = 33;
cs[44][1] = 0;
cs[44][2] = 0;
cs[44][3] = 1;
cs[44][4] = 0;
cs[44][5] = 0;
cs[44][6] = 0;
cs[44][7] = 1;
cs[44][8] = 5;
cs[44][9] = 6;
cs[44][10] = 2047;
cs[1810][0] = 37;
cs[1810][1] = 0;
cs[1810][2] = 0;
cs[1810][3] = 0;
cs[1810][4] = 33;
cs[1810][5] = 0;
cs[1810][6] = 0;
cs[1810][7] = 0;
cs[1810][8] = 12;
cs[1810][9] = 0;
cs[1810][10] = 0;
cs[1811][0] = 0;
cs[1811][1] = 1;
cs[1811][2] = 33;
cs[1811][3] = 0;
cs[1811][4] = 33;
cs[1811][5] = 0;
cs[1811][6] = 0;
cs[1811][7] = 0;
cs[1811][8] = 8;
cs[1811][9] = 6;
cs[1811][10] = 1809;
//inc pc
cs[2047][0] = 32;
cs[2047][1] = 0;
cs[2047][2] = 0;
cs[2047][3] = 0;
cs[2047][4] = 32;
cs[2047][5] = 0;
cs[2047][6] = 0;
cs[2047][7] = 0;
cs[2047][8] = 14;
cs[2047][9] = 6;
cs[2047][10] = 0;

a = new TextField(10);
a.setSize(30, 20);
a.setLocation(100 + 800, 100 + 480);
add(a);

amux = new TextField(1);
amux.setSize(15, 20);
```

## Capítulo 2. La Unidad de Control

```
amux.setLocation(135 + 800, 100 + 480);
add(amux);

b = new TextField(10);
b.setSize(30, 20);
b.setLocation(155 + 800, 100 + 480);
add(b);

bmux = new TextField(1);
bmux.setSize(15, 20);
bmux.setLocation(190 + 800, 100 + 480);
add(bmux);

c = new TextField(10);
c.setSize(30, 20);
c.setLocation(210 + 800, 100 + 480);
add(c);

cmux = new TextField(1);
cmux.setSize(15, 20);
cmux.setLocation(245 + 800, 100 + 480);
add(cmux);

rd = new TextField(10);
rd.setSize(15, 20);
rd.setLocation(265 + 800, 100 + 480);
add(rd);

wr = new TextField(10);
wr.setSize(15, 20);
wr.setLocation(285 + 800, 100 + 480);
add(wr);

alu = new TextField(10);
alu.setSize(30, 20);
alu.setLocation(305 + 800, 100 + 480);
add(alu);

cond = new TextField(10);
cond.setSize(15, 20);
cond.setLocation(340 + 800, 100 + 480);
add(cond);

jumpaddr = new TextField(4);
jumpaddr.setSize(40, 20);
jumpaddr.setLocation(360 + 800, 100 + 480);
add(jumpaddr);

loc = new TextField(4);
loc.setSize(40, 20);
loc.setLocation(1114, 252);
add(loc);
loc.addKeyListener(this);
loc.setText("0");

dirmca = new TextField(4);
dirmca.setSize(40, 20);
dirmca.setLocation(830, 602);
add(dirmca);
dirmca.addKeyListener(this);

calc = new Button("Ejecuta microinstruccion");
calc.setSize(150, 40);
calc.setLocation(300 + 800, 80 + 580);
add(calc);
calc.addActionListener(this);

}
```

# **Capítulo 3**

# **La Memoria Principal y**

# **la integración del**

# **procesador**

# Capítulo 3. La Memoria Principal y la integración del procesador

## 3.1 Memoria Principal

### 3.1.1 Configuración *Big-Endian*

Al ser la ARC un procesador de 32-bits, tanto las instrucciones como los datos son de 32-bits, sin embargo la memoria se direcciona a nivel de byte, por lo cual se utilizan 4 localidades continuas de la memoria principal para formar una instrucción o un dato. El simulador utiliza la configuración *Big-Endian*, en el cual el byte más significativo se coloca en la dirección más baja.

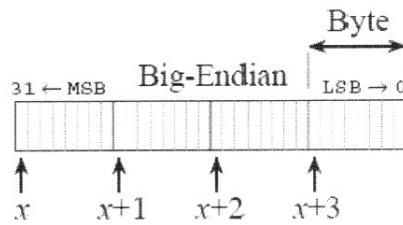


Figura 3.1 Configuración *Big-Endian*.

### 3.1.2 Operaciones de lectura y escritura

La memoria del simulador cuenta con 4096 localidades, en todas el usuario puede realizar operaciones de lectura y escritura, ya sea manualmente, o automáticamente cuando las controla el MIR, como ya antes se mencionó. Los botones de escritura y lectura facilitan al usuario poder cargar a nivel de bytes el código de las instrucciones y los datos de un determinado programa previamente ensamblado, y cuya ejecución a nivel micro arquitectura se desea verificar. En el simulador aparecen 10 localidades continuas y sus contenidos. Para realizar la lectura de 10 localidades continuas, se coloca en la primera casilla el número de la localidad de inicio, y al apretar el botón R (*Read – Lectura*), se despliega el contenido de las localidades. Para escribir, se coloca en la casillas *Contenido* los valores que se deseen guardar, y para su escritura se oprime el botón W (*Write – Escritura*).

Memoria Principal	
Dirección	Contenido
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Figura 3.2 Imagen de la memoria principal del procesador.

## Capítulo 3. La Memoria Principal y la integración del procesador

### 3.1.3 Diagrama de clases de la Memoria Principal

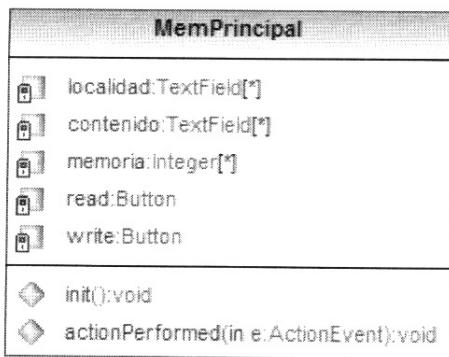


Figura 3.3 Diagrama de clases de la Memoria Principal.

### 3.1.4 Código en Java de la memoria principal

```
//memoria principal
private TextField[] localidad, contenido;
private Integer memoria[];
private Button read, write;

//memoria principal
localidad = new TextField[10];
contenido = new TextField[10];
memoria = new Integer[4096];

public void init(){
    setLayout(null);

    for (int i = 0; i < 10; i++) {    //textfields de mem principal
        localidad[i] = new TextField(8);
        localidad[i].setSize(80, 20);
        contenido[i] = new TextField(8);
        contenido[i].setSize(80, 20);
    }

    for (int i = 0; i < 10; i++) {    //textfields de mem principal
        localidad[i].setLocation(15, 400 + (i * 25));
        contenido[i].setLocation(105, 400 + (i * 25));
        add(localidad[i]);
        add(contenido[i]);
        contenido[i].addKeyListener(this);
    }
}

for (int i = 0; i < 4096; i++) {    //inicializacion memoria
    memoria[i] = 0;
}

for (int i = 0; i < 10; i++) {    //despliegue de memoria principal
    localidad[i].setText(Integer.toString(i * 1));
    contenido[i].setText(Integer.toString(memoria[i * 1]));
}
```

## Capítulo 3. La Memoria Principal y la integración del procesador

```
read = new Button("R"); //boton lectura memoria principal
read.setSize(80, 20);
read.setLocation(15, 650);
add(read);
read.addActionListener(this);

write = new Button("W"); //boton escritura memoria principal
write.setSize(80, 20);
write.setLocation(105, 650);
add(write);
write.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
    //mir y mem control
    int dirmc;

    //memoria
    if (e.getSource() == read) { //lec a partir de loc especificada
        int auxloc = Integer.valueOf(localidad[0].getText());
        for (int i = 0; i < 10; i++) {
            localidad[i].setText(Integer.toString((i * 1) + auxloc));
            contenido[i].setText(Integer.toString(memoria[(i * 1) + auxloc]));
        }
    } else if (e.getSource() == write) {
        for (int i = 0; i < 10; i++) { //lee todos textfiels y guarda en memoria
            memoria[Integer.valueOf(localidad[i].getText())] =
                Integer.valueOf(contenido[i].getText());
        }
    }
}
```

# Capítulo 3. La Memoria Principal y la integración del procesador

## 3.2 Integración y Pruebas

La imagen del simulador se muestra en la Figura 3.4, a la izquierda se encuentra la memoria principal, donde el usuario escribe las instrucciones y datos de un programa que consiste solamente en números decimales que representan bytes, y en la parte de la derecha se encuentra el procesador, conteniendo como ya se explicó anteriormente, la Ruta de los Datos (Área de trabajo) y la Unidad de Control.

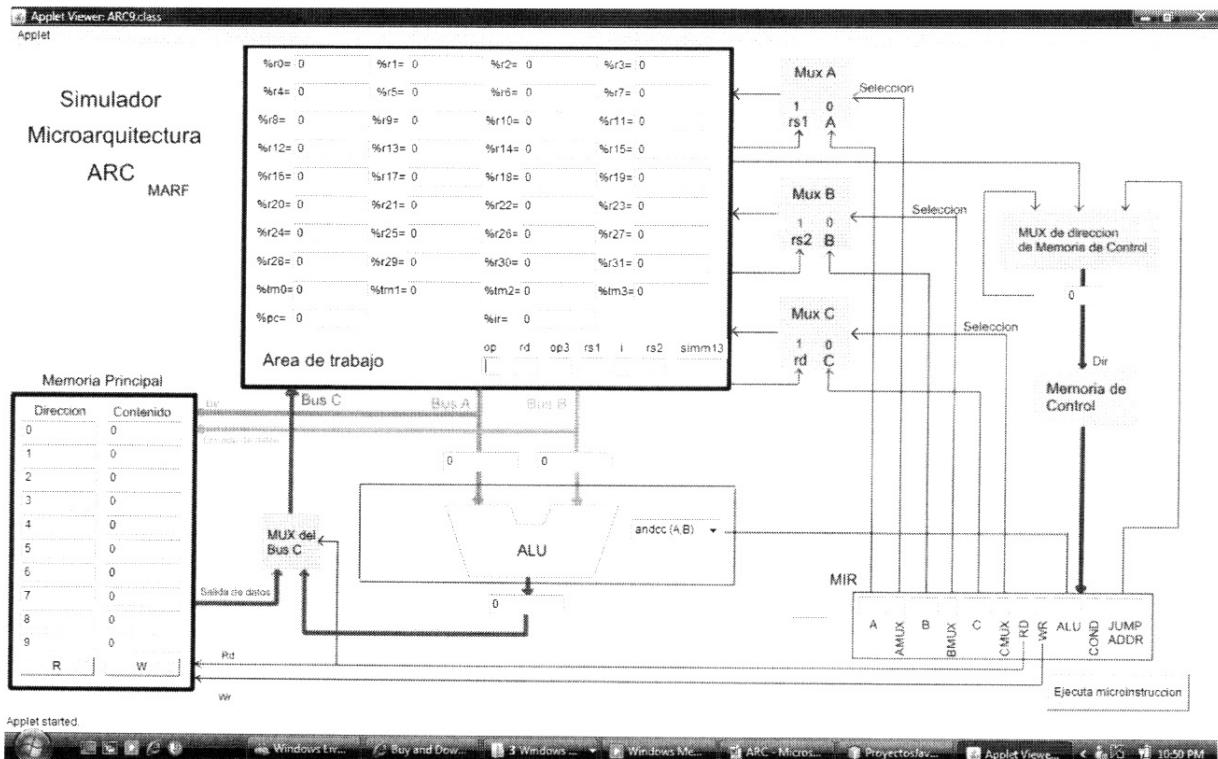


Figura 3.4 Imagen del simulador del procesador.

El simulador trabaja en esta primera versión en decimal todos sus campos, por lo cual no pueden utilizarse ningún carácter que no sea numérico, y los números son enteros, logrando esto con el siguiente código en java.

```
public void keyTyped(KeyEvent e) {
    if ((e.getKeyChar() < '0' || e.getKeyChar() > '9') && e.getKeyChar() != 8) {
        e.setKeyChar('\0');
    }
}
```

El simulador tiene una imagen de fondo (Figura 3.5), la cual se realizó en *CorelPhoto Paint X3*, sobre ésta se colocan los objetos programados en java. La imagen es del tipo jpg, y se carga en el procedimiento de inicialización.

## Capítulo 3. La Memoria Principal y la integración del procesador

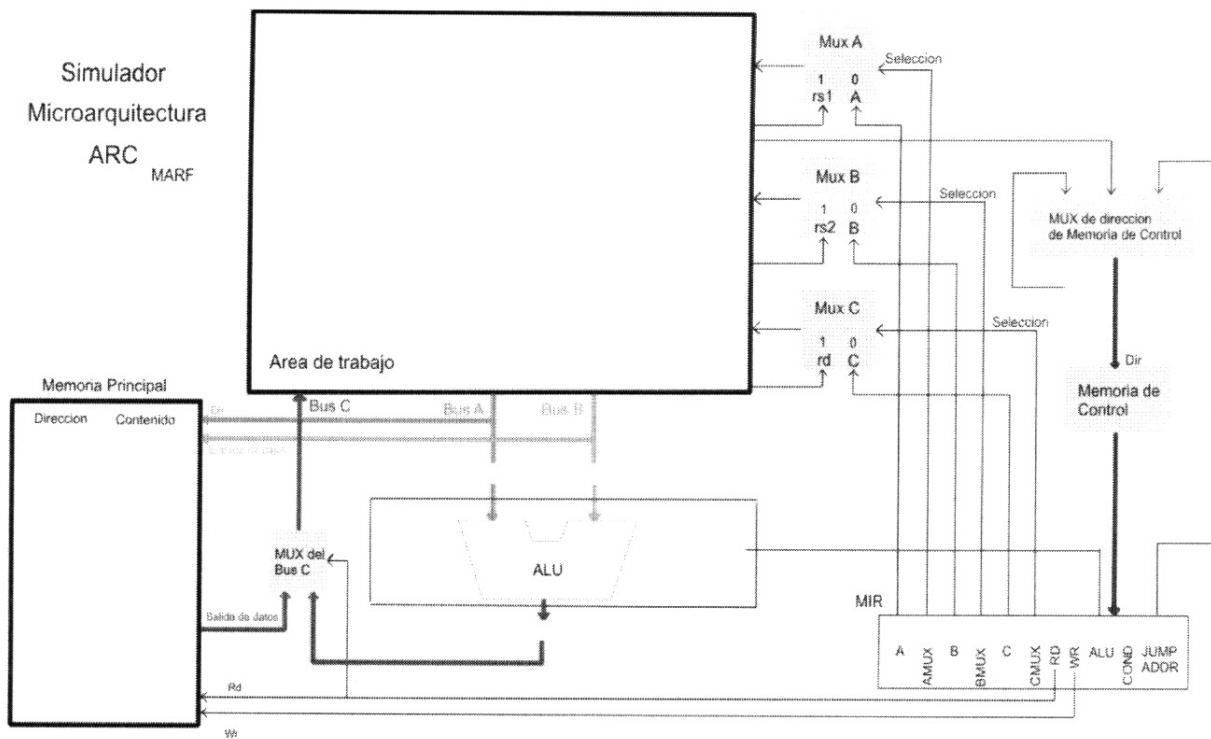


Figura 3.5 Imagen de fondo para el simulador.

```

private Image aluimg;

public void init(){
aluimg = getImage(getDocumentBase(), "arc3.jpg");
}

public void paint(Graphics g) { //dibujo mascara
g.drawImage(aluimg, 2, -1, this);
}

```

## Capítulo 3. La Memoria Principal y la integración del procesador

### 3.3 Diagrama de clases del simulador



Figura 3.6 Diagrama de clases del simulador.

## Capítulo 3. La Memoria Principal y la integración del procesador

### 3.4 ¿Qué sucede cuando se oprime el botón de Ejecutar Microinstrucción?

Cuando se oprime el botón de Ejecutar Microinstrucción se ejecutan de manera secuencial las siguientes operaciones:

1. Se direcciona la localidad 0 de la Memoria de Control, y se colocan los valores correspondientes en el MIR.
2. Se crea un objeto de la clase **opbusa**, en cuyo constructor se determina cual de los registros debe colocar su contenido en el Bus A. Recuérdese que esto depende del campo AMUX del MIR.
3. Se crea un objeto de la clase **opbusb**, en cuyo constructor se determina cual de los registros debe colocar su contenido en el Bus B. Recuérdese que esto depende del campo BMUX del MIR.
4. Se crea un objeto de la clase **operacion**, en cuyo constructor se realiza la operación aritmética o lógica especificada por el campo ALU del MIR, y coloca el resultado en el Bus C del procesador.
5. Se crea un objeto de la clase **opbusc**, en cuyo constructor se determina en cuál de los registros se almacena la información contenida por el Bus C. Esta información puede provenir del resultado de la ALU o bien ser una localidad de memoria que se ha leído.
6. Se crea un objeto de la clase **optarea**, la cual se utiliza para realizar escritura en la memoria principal cuando el campo WR del MIR es igual a 1.
7. Se crea un objeto de la clase **opcondicion**, en cuyo constructor se tiene la lógica para determinar cuál es la siguiente microinstrucción a ejecutar, existiendo tres posibilidades como se explico en el capítulo anterior, utilizar la dirección siguiente (NEXT), utilizar como dirección el campo JUMP ADDR del MIR, o realizar la operación DECODE.
8. Se crea un objeto de la clase **opir**, en cuyo constructor se lee el registro %ir, y se obtienen los campos op, rd, rs1, rs2, i, op3 y simm13.

### 3.5 Código en Java de lo que sucede cuando se oprime el botón Ejecutar Microinsrucción

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource() == calc) {  
        dirmc = Integer.parseInt(loc.getText());  
        dirmca.setText(Integer.toString(dirmc));  
        a.setText(Integer.toString(cs[dirmc][0]));  
        amux.setText(Integer.toString(cs[dirmc][1]));  
        b.setText(Integer.toString(cs[dirmc][2]));  
        bmux.setText(Integer.toString(cs[dirmc][3]));  
        c.setText(Integer.toString(cs[dirmc][4]));  
        cmux.setText(Integer.toString(cs[dirmc][5]));  
        rd.setText(Integer.toString(cs[dirmc][6]));  
        wr.setText(Integer.toString(cs[dirmc][7]));  
        alu.setText(Integer.toString(cs[dirmc][8]));  
        cond.setText(Integer.toString(cs[dirmc][9]));  
        jumpaddr.setText(Integer.toString(cs[dirmc][10]));  
  
        limpia limpial;  
        limpial = new limpia();  
  
        opbusa opbusal;  
        opbusal = new opbusa();  
  
        opbusb opbusb1;  
        opbusb1 = new opbusb();  
  
        operacion operacion1;
```

## Capítulo 3. La Memoria Principal y la integración del procesador

```
operacion1 = new operacion();

opbusc opbusc1;
opbusc1 = new opbusc();

optarea optareal;
optareal = new optarea();

opcondicion opcondicion1;
opcondicion1 = new opcondicion();

opir opir2;
opir2 = new opir();
}

}

class operacion { //alu

//int alua, alub, aluc, aux, aux1;
long alua, alub, aluc, aux, aux1;

operacion() {

    alua = Long.parseLong(busa.getText());
    alub = Long.parseLong(busb.getText());


    //operacion alu
    op.select(Integer.parseInt(alu.getText()));
    aux = op.getSelectedIndex();
    if (aux == 0) { //andcc
        aluc = alua & alub;
    } else if (aux == 1) { //orcc
        aluc = alua | alub;
    } else if (aux == 2) { //norcc
        aluc = ~(alua | alub);
    } else if (aux == 3) { //addcc
        aluc = alua + alub;
    } else if (aux == 4) { //srl
        aluc = alua >> alub;
    } else if (aux == 5) { //and
        aluc = alua & alub;
    } else if (aux == 6) { //or
        aluc = alua | alub;
    } else if (aux == 7) { //nor
        aluc = ~(alua + alub);
    } else if (aux == 8) { //add
        aluc = alua + alub;
    } else if (aux == 9) { //lshift2
        aluc = alua << 2;
    } else if (aux == 10) { //lshift10
        aluc = alua << 10;
    } else if (aux == 11) { //simm13
        aluc = alua & 0x00001FFF;
    } else if (aux == 12) { //sext13
        aluc = alua & 0x00001FFF;
        aux1 = aluc & 0x00001000;
        if (aux1 == 0x00001000) { //si es negativo llena con 1s a la izq
            aluc = aluc | 0x1111E000;
        }
    } else if (aux == 13) { //inc
        aluc = alua + 1;
    } else if (aux == 14) { //incpc
        aluc = alua + 4;
    } else if (aux == 15) { //rshift5
        aluc = alua >> 5;
    }
}
```

## Capítulo 3. La Memoria Principal y la integración del procesador

```
// impresion de resultados alu

busa.setText(Long.toString(alua));
busb.setText(Long.toString(alub));
busc.setText(Long.toString(aluc));

}

}

class opir {

long aux2;

opir() {
//campo op
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0xc0000000;
aux2 = aux2 >> 30;
irop.setText(Long.toString(aux2));
//campo rd
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x3e000000;
aux2 = aux2 >> 25;
irrd.setText(Long.toString(aux2));
//campo op3
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x01f80000;
aux2 = aux2 >> 19;
irop3.setText(Long.toString(aux2));
//campo rs1
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x0007c000;
aux2 = aux2 >> 14;
irrs1.setText(Long.toString(aux2));
//campo i
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x00002000;
aux2 = aux2 >> 13;
iri.setText(Long.toString(aux2));
//campo rs2
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x0000001f;
irrs2.setText(Long.toString(aux2));
//campo simm13
aux2 = Long.parseLong(registro[37].getText());
aux2 = aux2 & 0x00001fff;
irsimm13.setText(Long.toString(aux2));
}
}

class limpia {

limpia() {
for (int i = 0; i < 38; i++) {
registro[i].setBackground(Color.WHITE);
}
}
}

class opbusa {

opbusa() {

// verifica amux y coloca valor en busa
if (amux.getText().equals("0")) {
//a de MIR
busa.setText(registro[Integer.parseInt(a.getText())].getText());
registro[Integer.parseInt(a.getText())].setBackground(Color.GREEN);
} else {
//a de rsl de %ir
}
}
}
```

## Capítulo 3. La Memoria Principal y la integración del procesador

```
busa.setText(registro[Integer.parseInt(irrs1.getText())].getText());
registro[Integer.parseInt(irrs1.getText())].setBackground(Color.green);
}

}

class opbusb {

opbusb() {
//verifica bmux y coloca valor en busb
if (bmux.getText().equals("0")) {
//b de MIR
busb.setText(registro[Integer.parseInt(b.getText())].getText());
registro[Integer.parseInt(b.getText())].setBackground(Color.CYAN);
} else {
//b de rs2 de %ir
busb.setText(registro[Integer.parseInt(irrs2.getText())].getText());
registro[Integer.parseInt(irrs2.getText())].setBackground(Color.CYAN);
}
}

}

class opbusc {

long valb1, valb2, valb3, valb4, valtot;

opbusc() {

if (cmux.getText().equals("0")) {
//c de MIR
if (rd.getText().equals("0")) { //registro <- alu
registro[Integer.parseInt(c.getText())].setText(busc.getText());
registro[Integer.parseInt(c.getText())].setBackground(Color.red);
} else { //registro <- mem
System.out.println("lee");
valb1 = (memoria[Integer.parseInt(busa.getText())]) * 16777216L;
valb2 = (memoria[Integer.parseInt(busa.getText()) + 1]) * 65536L;
valb3 = (memoria[Integer.parseInt(busa.getText()) + 2]) * 256L;
valb4 = memoria[Integer.parseInt(busa.getText()) + 3];
System.out.println(memoria[2048]);
System.out.println(valb1);
valtot = valb1 + valb2 + valb3 + valb4;
registro[Integer.parseInt(c.getText())].setText(Long.toString(valtot));
registro[Integer.parseInt(c.getText())].setBackground(Color.red);
}

} else {
//c de rd de %ir
if (rd.getText().equals("0")) { //registro <- alu
registro[Integer.parseInt(irrd.getText())].setText(busc.getText());
registro[Integer.parseInt(irrd.getText())].setBackground(Color.red);
} else { //registro <- mem
valb1 = (memoria[Integer.parseInt(busa.getText())]) << 24;
valb2 = (memoria[Integer.parseInt(busa.getText()) + 1]) << 16;
valb3 = (memoria[Integer.parseInt(busa.getText()) + 2]) << 8;
valb4 = memoria[Integer.parseInt(busa.getText()) + 3];
valtot = valb1 + valb2 + valb3 + valb4;
registro[Integer.parseInt(irrd.getText())].setText(Long.toString(valtot));
registro[Integer.parseInt(irrd.getText())].setBackground(Color.red);
}
}
}

}

class optarea {

optarea() {
if (wr.getText().equals("1")) { //escribe en memoria busa=dir, busb=dato
memoria[Integer.parseInt(busa.getText()) + 0] = (Integer.parseInt(busb.getText()))
}
}
```

## Capítulo 3. La Memoria Principal y la integración del procesador

```
& 0xFF000000) >> 24;
        memoria[Integer.parseInt(busa.getText()) + 1] = (Integer.parseInt(busb.getText()))
& 0x00FF0000) >> 16;
        memoria[Integer.parseInt(busa.getText()) + 2] = (Integer.parseInt(busb.getText()))
& 0x0000FF00) >> 8;
        memoria[Integer.parseInt(busa.getText()) + 3] = (Integer.parseInt(busb.getText()))
& 0x000000FF) >> 0;
    }
}

class opcondicion {

    int ayuda;

    opcondicion() {
        if (cond.getText().equals("0")) { //use next address loc=loc+1
            loc.setText(Integer.toString(Integer.parseInt(loc.getText()) + 1));
        } else if (cond.getText().equals("1")) { //use jump addr
            loc.setText(jumpaddr.getText());
        } else if (cond.getText().equals("2")) { //use jump addr
            loc.setText(jumpaddr.getText());
        } else if (cond.getText().equals("3")) { //use jump addr
            loc.setText(jumpaddr.getText());
        } else if (cond.getText().equals("4")) { //use jump addr
            loc.setText(jumpaddr.getText());
        } else if (cond.getText().equals("5")) { //use jump addr si if[13]=1
            if (iri.getText().equals("1")) {
                loc.setText(jumpaddr.getText());
            } else {
                loc.setText(Integer.toString(Integer.parseInt(loc.getText()) + 1));
            }
        } else if (cond.getText().equals("6")) { //use jump addr
            loc.setText(jumpaddr.getText());
        } else if (cond.getText().equals("7")) { //decode
            ayuda = 1024 + (Integer.parseInt(rop.getText())) * 256 +
(Integer.parseInt(rop3.getText())) * 4;
            loc.setText(Integer.toString(ayuda));
        }
    }
}
```

## Capítulo 3. La Memoria Principal y la integración del procesador

---

### 3.6 Programa para pruebas del simulador

Para su verificación se ejecutó a nivel micro arquitectura un programa que suma 2 números de la memoria, y almacena el resultado nuevamente en la memoria. El programa en ensamblador y su equivalencia a lenguaje de máquina (unos y ceros) se muestra a continuación.

```
.begin          !Inicio del programa  
.org 2048      !Dirección de inicio de donde inician las instrucciones  
ld [x],%r1      !Carga el valor de x en el registro %r1, por lo tanto %r1=15  
ld [y],%r2      !Carga el valor de y en el registro %r2, por lo tanto %r2=3  
addcc %r1,%r2,%r3  !Suma %r1 mas %r2, y almacena el resultado en %r3, %r3=18  
st %r3,[z]      !Almacena el valor del registro %r3 en la localidad z, [z]=18  
  
x: 15  
y: 3  
z: 0  
  
.end          !Fin del programa
```

Cuando este programa es ensamblado, se obtiene los códigos binarios que representan a las instrucciones y a los datos en el lenguaje que puede entender el procesador, que son simplemente unos y ceros. Recordando que la ARC es un procesador tipo RISC de 32 bits, las instrucciones y los datos se representan por 32 bits, es decir 4 bytes continuos de la memoria.

Dirección	Contenido (hex)	Instrucción o dato
2048	C2002810	ld [x],%r1
2052	C4002814	ld [y],%r2
2056	86804002	addcc %r1,%r2,%r3
2060	C6202818	st %r3,[z]
2064	0000000F	15
2068	00000003	3
2072	00000000	0

Al utilizar 4 bytes por cada instrucción o dato, y al utilizar la configuración *big-endian*, se tendría que escribir en la memoria principal del simulador, los siguientes valores.

## Capítulo 3. La Memoria Principal y la integración del procesador

---

Dirección	Contenido (hex)	Contenido (dec)
2048	C2	194
2049	00	0
2050	28	40
2051	10	16
2052	C4	196
2053	00	0
2054	28	40
2055	14	20
2056	86	134
2057	80	128
2058	40	64
2059	02	2
2060	C6	198
2061	20	32
2062	28	40
2063	18	24
2064	00	0
2065	00	0
2066	00	0
2067	0F	15
2068	00	0
2069	00	0
2070	00	0
2071	03	3
2072	00	0
2073	00	0
2074	00	0
2075	00	0

## Conclusiones y Recomendaciones

---

### Conclusiones y Recomendaciones

Existen temas en las asignaturas a nivel superior del Instituto, cuyo nivel de abstracción es tal que dificulta al docente su explicación, y al alumno su entendimiento, por lo tanto el desarrollo de programas de simulación son una herramienta de ayuda en el proceso de enseñanza – aprendizaje.

Al estar basado en un procesador comercial (SPARC, *Sun Microsystems*) y ser una versión didáctica y simplificada del mismo (el caso del ARC), existe una gran cantidad de información y aplicaciones desarrolladas por el autor Miles Murdocca en su libro, permite al alumno tener esta información y ayudarle en su proceso de aprendizaje.

Si bien es una primera versión del simulador, existen muchas mejoras por implementarse en el futuro, como será el poder trabajar y alternar entre el sistema hexadecimal y decimal, cuando el usuario así lo desee.

En una versión más completa, es importante poder implementar un mayor número de instrucciones que las que ya se han implementado, para que la herramienta sea más flexible.

La ALU no está trabajando con el registro del estado del procesador (%psr), el cual almacena el estado de las banderas. La versión de la ARC, trabaja con solo cuatro banderas, acarreo, sobreflujo, negativo y cero. Es importante implementarlas y realizar las modificaciones necesarias para la correcta secuencia de las condiciones que involucran el registro de las banderas.

Se puede implementar nuevos métodos para cargar inicialmente un programa de un mayor número de instrucciones, ya sea por lectura de un archivo de texto, o trabajando palabras de 4 bytes en hexadecimal.

El simulador desarrollado puede utilizarse como base en la realización de otros simuladores de arquitecturas diferentes, que sumen esfuerzos en el proceso de entendimiento de los mismos.

## Referencias

---

### Referencias

BELL, Douglas, PARR, Mike. JAVA para estudiantes, Prentice Hall, 2003.

HORSTMANN, Cay. Java 2, Volumen I, Fundamentos, Prentice Hall, 2006.

MURDOCCA, Miles, HEURING, Vincent. Computer Architecture and Organization. Wiley, 2007.

SCHMULLER, Joseph. Aprendiendo UML en 24 horas, Prentice Hall, 2005.

SCHILD'T, Herbert. Java 2, Manual de referencia, Mc Graw Hill, 2001.

## Anexo. Manual de Usuario

### Ejemplo

**Planteamiento:** Se realizará la ejecución de un programa que ejecuta la suma de 2 números en la memoria y almacenando el resultado. (Ver Capítulo 3.6)

**Paso 1.** Cargar en programa del simulador utilizando NetBeans IDE 6.8. Colocar el mouse sobre el area de código y apretando el botón derecho, dar click en Run File.

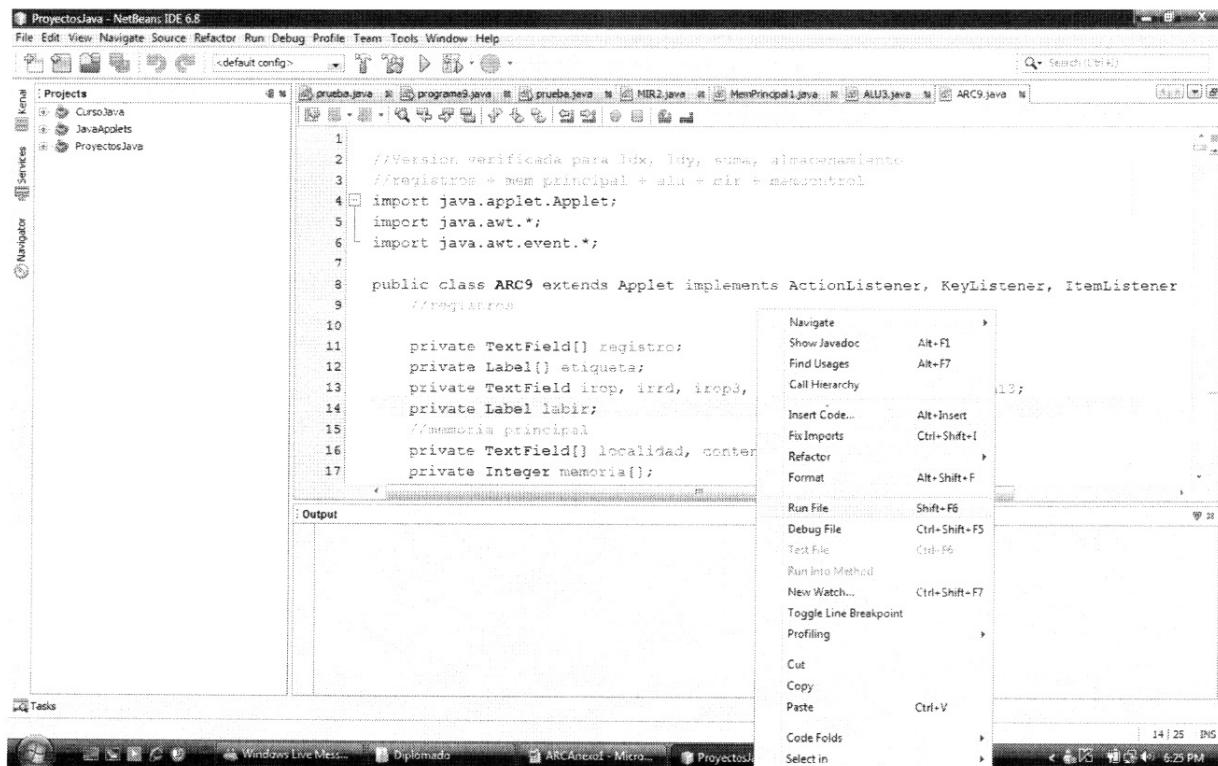


Figura A.1 Carga del programa del simulador.

## Anexo. Manual de Usuario

**Paso 2.** Deberá aparecer la imagen del simulador.

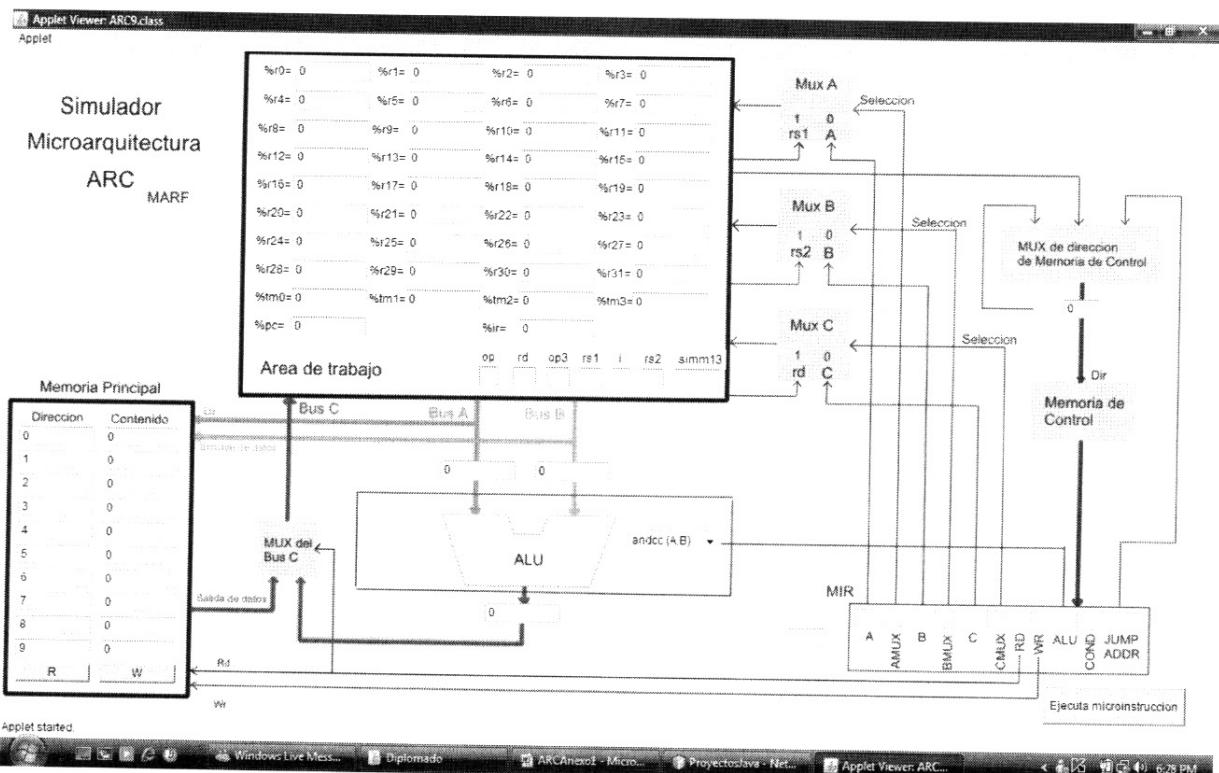


Figura A.2 Ventana de inicio del simulador.

**Paso 3.** Lo primero que se debe realizar, es cargar el programa en la memoria principal. Recordemos que se utilizará el programa ejemplo del Capítulo 3, como el programa empieza en la localidad 2048, se debe colorar este valor en la primera casilla de dirección, y dar un click en el botón R, al hacer esto aparecerán las localidades de memoria de la 2048 a la 2057.

Memoria Principal	
Direccion	Contenido
2048	0
2049	0
2050	0
2051	0
2052	0
2053	0
2054	0
2055	0
2056	0
2057	0
R	W

Figura A.3 Lectura de las localidades de memoria en donde se colocará el programa.

## Anexo. Manual de Usuario

Inicialmente la memoria contiene puros ceros, pero se necesita que este aquí almacenado el programa y los datos, para esto, se va llenando las localidades de memoria con los datos necesarios y para guardarlos en la memoria, se oprime el botón W, que guarda el contenido de las 10 localidades que se visualizan.

Memoria Principal	
Direccion	Contenido
2048	192
2049	0
2050	40
2051	16
2052	196
2053	0
2054	40
2055	20
2056	134
2057	128

Figura A.4 Escritura en la memoria principal de los primeras instrucciones del programa.

Para continuar escribiendo el programa y los datos, se repite el paso 3 las veces que sea necesario, colocando como dirección las localidades siguientes.

**Paso 4.** Colocar en el registro %pc, el valor de la localidad donde inicia el programa, en este ejemplo, 2048.

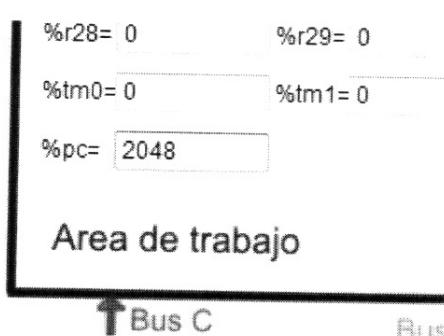


Figura A.5 Inicialización del contador del programa.

## Anexo. Manual de Usuario

**Paso 5.** Se oprime el botón de **Ejecutar Microinstrucción** para ir realizando la ejecución del programa, y se puede ir analizando que sucede en el procesador por cada micro instrucción.

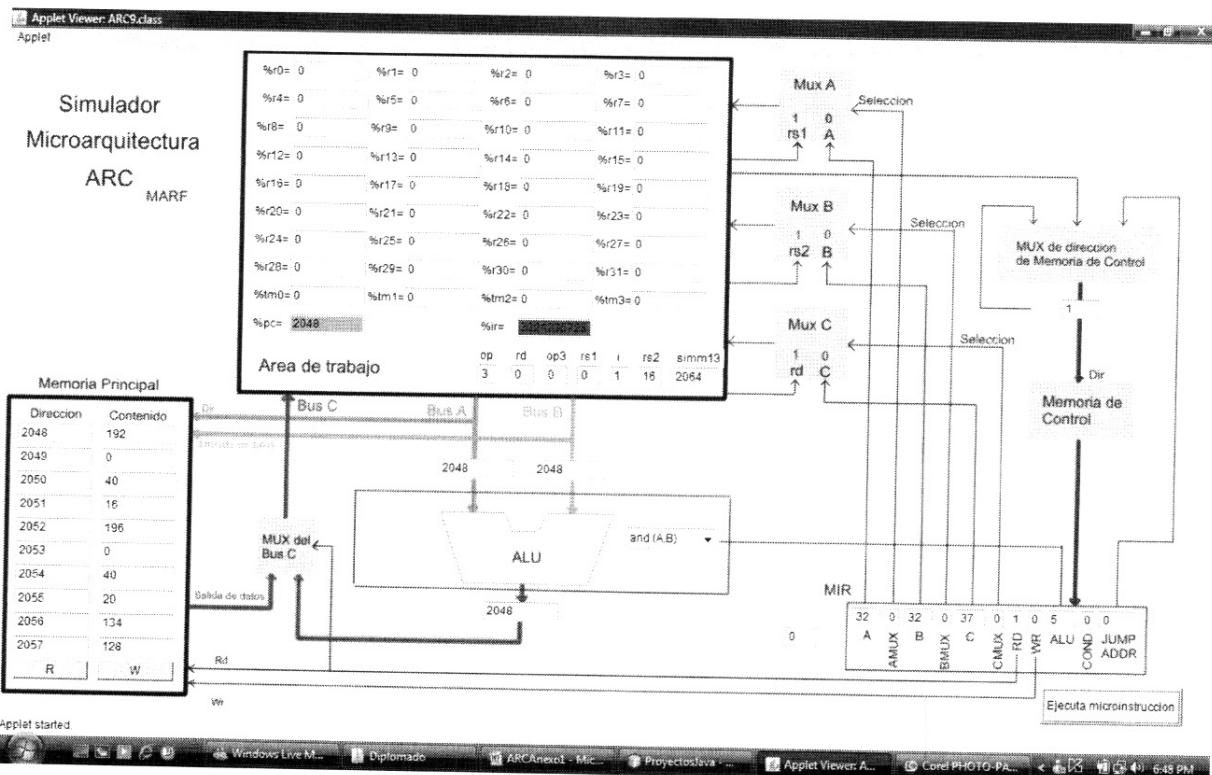


Figura A.6 Ejecución de una micro instrucción.

De la Figura A.6 se puede obtener mucha información. Por ejemplo, el MIR está lleno con la información de la localidad 0 de la Memoria de Control, y por lo campos que tiene el MIR. Como ya se había mencionado, vemos que el registro %pc que vale 2048 coloca este valor tanto en el Bus A, como en Bus B. La ALU hace la operación and entre estos valores obteniéndose el mismo resultado. Lo más importante de esta micro instrucción es la operación de lectura, vemos que el dato 2048 que está en el Bus A representa la dirección de memoria que se desea leer de la memoria principal, y es aquí donde está el inicio del programa, recordando la configuración *Big-endian*, se leen 4 casillas continuas (2048 a 2052) y esta es la primera instrucción del programa. Al leer la memoria, el dato leído (4 bytes) se coloca en registro %ir (%ir=3221235728), y esto sucede debido a que el campo C del MIR apunta al registro 37, que es %ir. Se aprecia también en la parte superior de la memoria de control, que la siguiente micro instrucción es la localidad 1. Y se continúa oprimiendo el botón de Ejecutar Micro instrucción hasta haber terminado con la ejecución de todas las instrucciones del programa principal.