



Proyecto final Arquitectura de computadores

Miguel Ángel Naranjo Joya 20201020037
Camilo Andrés García Zambrano 20201020047
Laura Andrea Riobueno Rincón 20201020040

Universidad Distrital Francisco José De Caldas
Facultad de Ingeniería
Ingeniería de Sistemas
Bogotá
2022

1. Objetivo General

Simular una máquina de computo virtual la cuál pueda procesar códigos de operación e implemente registros y memorias.

1.1. Objetivos específicos

1. Entender el funcionamiento tanto individual como en conjunto de cada uno de los componentes del sistema SAP.
2. Ampliar la capacidad de la maquina de computo virtual brindada a una que cumpla con los requerimientos.

2. Planteamiento del problema

Elaborar un software que simule el funcionamiento de una máquina de cómputo, usando las siguientes especificaciones mínimas:

- El bus de datos será de 16 bits.
- La longitud del OpCode será de 6 bits (Para un máximo de 64 instrucciones de máquina).
- Se podrá direccionar hasta 16 MiB de RAM (24 bits).
- Tendrá un registro de pila, la cual será de 1 KiB, y empezará en la dirección de memoria 0xffffbff (Osea, las últimas posiciones de memoria).
- El diseño tendrá en cuenta el formato de almacenamiento Big Endian.

Se recomienda usar un lenguaje de programación orientado a objetos para construir una solución tipo monolito con GUI, donde se puedan pasar programas en el código ensamblador con los opcodes diseñados (tener 2 programas de prueba listos).

3. Marco teórico

3.1. SAP

Como sus siglas en inglés lo indican (Simple As Possible), se trata de un modelo computacional o más precisamente una arquitectura a nivel máquina propuesto por Albert Paul Malvino y Jerald A. Brown, bajo una propuesta educativa ya que propone enseñar el funcionamiento a nivel máquina de lo que hoy en día se percibe en términos generales como Computador o PC; para proponer este diseño implemento las siguientes partes:

1. MAR (Memory Address Register).
2. Registros A y B.
3. ALU (Arithmetic Logic Unit).
4. RAM (Random Access Memory).

5. PC (Program Counter).
6. IR (Instruction Register).
7. OUT y CONTROL.

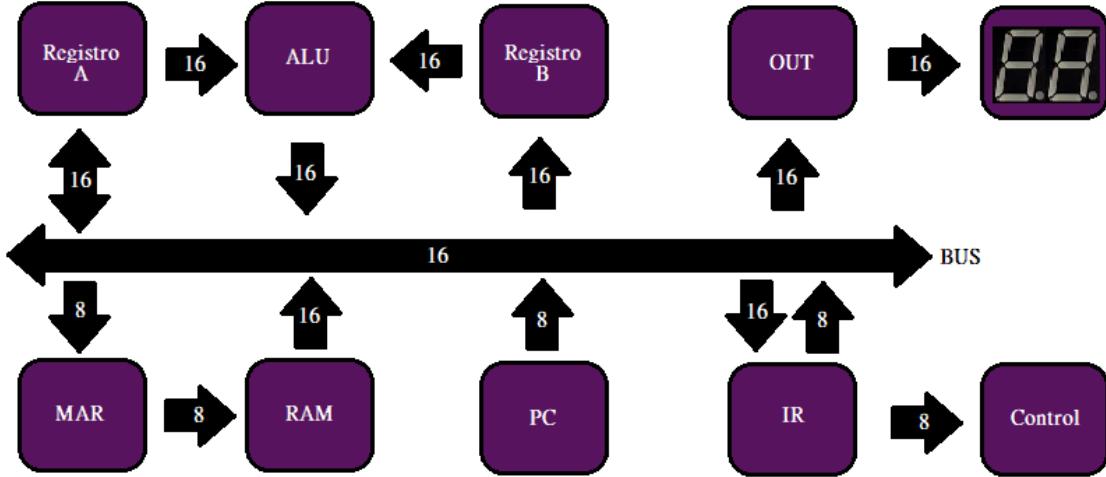


Figura 1. Esquematización de sistema SAP

3.2. Direcciones de memoria

Cuando se habla a nivel hardware de una dirección de memoria se hace referencia al contenido alojado en un registro, que nos indica en qué posición de la memoria RAM debemos ir a buscar determinado valor.

3.3. Bus de memoria

Un bus de memoria hace referencia al puente o camino sobre el cual se van a comunicar las diferentes partes o componentes del sistema, siendo este un conjunto de líneas o caminos sobre los cuales se da paso a la información y órdenes de control, etc.

3.4. Registro Pila

Trayendo el concepto de pila de programación, este posee un funcionamiento similar al de una pila de platos en donde el primero en llegar es el último en salir (LIFO); a nivel hardware no es más que una memoria que recibirá el nombre de pila y dos registros que almacenarán los punteros que indican la dirección de memoria del primer y último elemento almacenado de manera tal que se pueda acceder al primer valor y se pueda agregar.

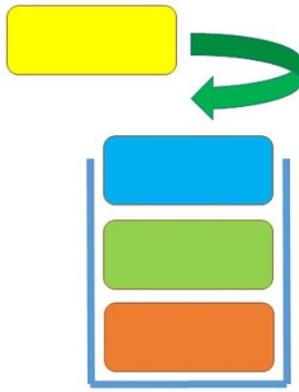


Figura 2. Esquematización de registro PILA

3.5. OPCODES

En español, códigos de operaciones, son los encargados de dar el paso a paso, y el comportamiento de cada uno de los componentes en el sistema, desde registros, hasta el resultado de la ALU; en términos generales, es el intermediario entre el usuario (programador) y el lenguaje máquina, permitiendo cargar, añadir y demás instrucciones acorde a la necesidad del usuario y la limitación de la máquina.

4. Problemas encontrados

Al realizar el programa con los 16 MiB de RAM obtenemos el siguiente error:

```
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "main"
[...]
C:\Users\lain-\AppData\Local\NetBeans\Cache\14\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\lain-\AppData\Local\NetBeans\Cache\14\executor-snippets\run.xml:94: Java returned: 1
BUILD FAILED (total time: 34 seconds)
```

Figura 3. Error arrojado por Java

Como se sabe en el lenguaje Java se debe inicializar campo a campo de un array para no obtener el famoso *java.lang.NullPointerException*; en una primera instancia al intentar inicializar cada una de las posiciones del array sobre el cual se iba a manejar la memoria RAM, se observa como de repente el programa toma un tiempo en responder (gráficamente) así como también se ve un uso elevado del procesador, debido a la cantidad de operaciones que debe emplear; en consecuencia resultamos en un segundo escenario, en donde se encuentra con un error de memoria debido a que la JVM no puede manejar la cantidad memoria a emplear.

5. Nuevo planteamiento

5.1. Cambio en planteamiento inicial

Debido a este error y buscando que se lograse aumentar la cantidad de memoria RAM, se optó por aumentar la memoria a 64B de RAM (6bits), debido a que este valor nos permite mantener un rendimiento óptimo y eficiente; por otra los demás requerimientos se mantienen sin cambio alguno.

5.2. ¿Qué se agrego nuevo al código y como se implemento?

1. Se crearon dos nuevos registros de 6 bits y 16 bits respectivamente.
 2. Se aumento la capacidad de memoria a 64 bits.

```
public Memoria(IRegistro MAR) {  
    capacidadMemoria = 64;  
    this.data = new int[capacidadMemoria];  
    this.MAR = MAR;  
    this.observers = new ArrayList<IRAMObserver>();
```

Figura 4. Aumento de memoria RAM

3. Se realizaron cambios en la ALU para hacerla compatible con los cambios.

```
// Calcula es estado de Zero
int resultado;
if (sub) {
    resultado = (0b0000000000000000000000000000000011111111111111111111111111111111 & this.registroA.getValor());
    ...
    - (0b0000000000000000000000000000000011111111111111111111111111111111 & this.registroB.getValor());
} else {
    resultado = (0b0000000000000000000000000000000011111111111111111111111111111111 & this.registroA.getValor());
    ...
    + (0b0000000000000000000000000000000011111111111111111111111111111111 & this.registroB.getValor());
}

if ((resultado & 0b1111111111111111) == 0) {
    zF = true;
} else {
    zF = false;
}
```

Figura 5. Cambios en la ALU

4. Se crea la PILA con la capacidad necesaria (1 KiB) y sus propios métodos para mover, eliminar, etc.

```
public class Pila {  
    private int capacidadPila;  
    private int[] data;  
    private RegistroPunteroI punteroInicio; //guarda referencia direccion en memoria de primera posicion.  
    private RegistroPunteroF punteroFinal; //guarda referencia direccion en memoria de primera posicion.  
  
    public Pila() {  
        capacidadPila = 0x400;  
        this.data = new int[capacidadPila];  
        this.punteroInicio.setValor((int) 0x400);  
        this.punteroFinal.setValor((int) 0x400);  
    }  
}
```

Figura 6. Clase PILA

5. Se aumentó el espacio del Program Counter (PC) a 64 bits.

```
// Incrementa el contador del programa
public void activarConteo() {
    if (this.valor == 63) {
        // Si el PC ya está en 64, reinicielo ya que eso representa un desbordamiento de 6 bits
        this.valor = 0;
    } else {
        this.valor++;
    }
}
```

Figura 7. Cambios en el PC

6. Se crearon dos clases nuevas, registro puntero final y registro puntero inicial para hacer funcionar la lógica de la PILA.

```
public class RegistroPunteroF implements IRegistro{
    private int valor;

    public RegistroPunteroF() {
        this.valor = 0;
    }

    @Override
    public void setValor(int v) {
        this.valor = v;
    }

    @Override
    public int getValor() {
        return this.valor;
    }

    @Override
    public void clear() {
        this.valor = 0;
    }
}
```

Figura 8. Registro Puntero Final

```

public class RegistroPunteroI implements IRegistro{

    private int valor;

    public RegistroPunteroI() {
        this.valor = 0;
    }

    @Override
    public void setValor(int v) {
        this.valor = v;
    }

    @Override
    public int getValor() {
        return this.valor;
    }

    @Override
    public void clear() {
        this.valor = 0;
    }

}

```

Figura 9. Registro Puntero Inicial

7. Se realizaron cambios en la clase SistemaSAP para ser compatible con la lógica de la PILA.
8. Se agregaron dos nuevas instrucciones (OpCode), para ingresar valores al PILA (INP) y para sacar valores de la PILA (OTP).

```

if (instrucionActual == TipoInstruccion.INP) {
    if (this.stepCount == 3) {
        this.resetTodasLineasControl();
        this.lineasControl[IO] = true;
        this.lineasControl[MI] = true;
        notificarCambioLineasControl();
        EventLog.getEventLog().addEntrada("INP => IO, MI activadas");
    }
    if (this.stepCount == 4) {
        this.resetTodasLineasControl();
        this.lineasControl[INP] = true;
        notificarCambioLineasControl();
        EventLog.getEventLog().addEntrada("INP => RO, AI activadas");
    }
    if (this.stepCount == 5) {
        this.lineasControl[INP] = true;
        EventLog.getEventLog().addEntrada("INP => No hace nada");
    }
}

```

Figura 10. OpCode INP

```

if (instruccionActual == TipoInstruccion.OTP) {
    if (this.stepCount == 3) {
        this.resetTodasLineasControl();
        this.lineasControl[IO] = true;
        this.lineasControl[MI] = true;
        notificarCambioLineasControl();
        EventLog.getEventLog().addEntrada("OTP => IO, MI activadas");
    }
    if (this.stepCount == 4) {
        this.resetTodasLineasControl();
        this.lineasControl[OTP] = true;
        notificarCambioLineasControl();
        EventLog.getEventLog().addEntrada("OTP => RO, AI activadas");
    }
    if (this.stepCount == 5) {
        this.lineasControl[OTP] = true;
        EventLog.getEventLog().addEntrada("OTP => No hace nada");
    }
}

```

Figura 11. OpCode OTP

9. Se aumentó la longitud del OpCode para 64 posiciones de instrucciones de máquina.

```

public SistemaSAP() {
    this.registroA = new Registro16Bit();
    this.registroB = new Registro16Bit();
    this.registroSalida = new Registro16Bit();
    this.registroIR = new Registro16Bit();
    this.registroMAR = new Registro6Bit();
    this.programCounter = new ProgramCounter();
    this.stepCount = 0;
    this.RAM = new Memoria(this.registroMAR);
    this.log = EventLog.getEventLog();
    this.alu = new ALU(this.registroA, this.registroB);
    this.lineasControl = new boolean[64];

    this.bus = new Registro16Bit();
}

```

Figura 12. Aumento en la capacidad de lineas de control.

10. La capacidad de direccionamiento de la memoria RAM se aumentó a 6 bits.

Dado que un requerimiento era aumentar la memoria RAM, se optó por subirla a 64 bits en vez que a 16 millones de bits, esto significa que los bits que direccionan esa capacidad de memoria, se aumentó a 6 bits, en vez de a 24 bits

11. Se realizó el cambio de la paleta de colores, además fue necesario redimensionar elementos como las casillas del contenido de la memoria, y agregar más Labels en el widget SAP.

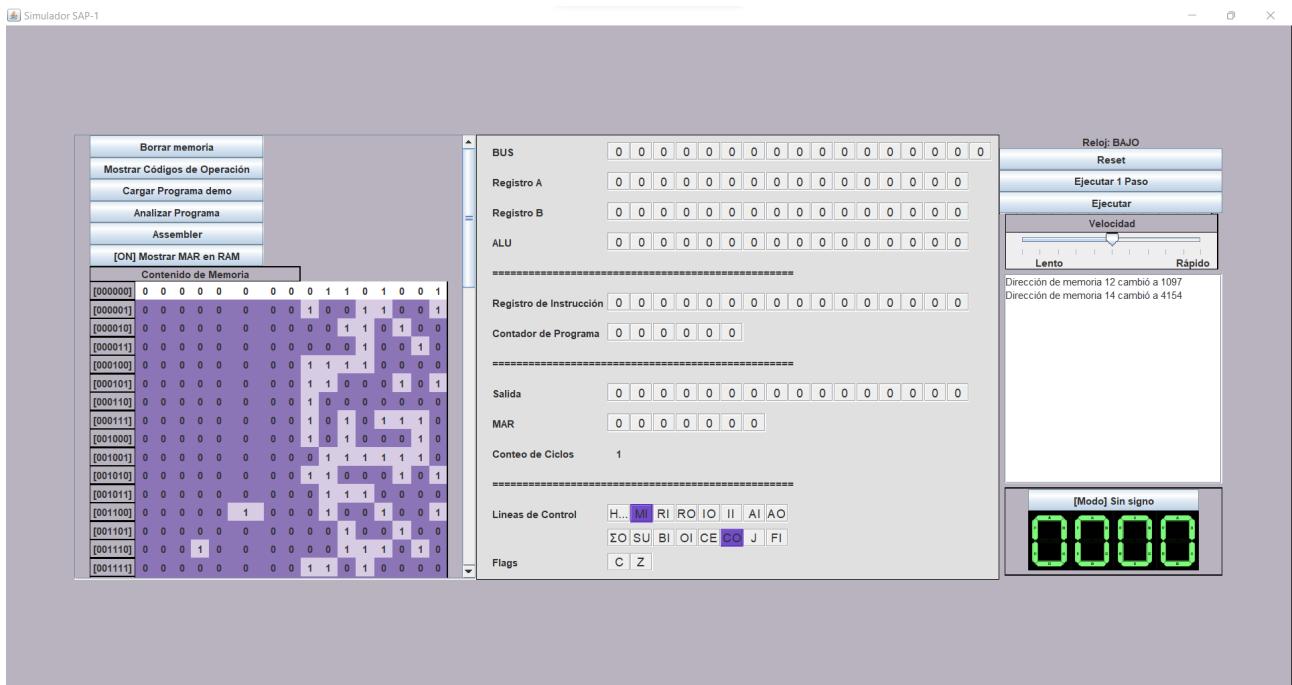


Figura 13. Vista del programa al correrlo.

6. Conclusiones

1. No se logró llevar a cabalidad cada uno de los requerimientos propuestos inicialmente; no obstante, se pudo reformular una parte del requerimiento acerca de la RAM y conseguir que se adaptasen al programa modificado.
 2. Debido al contra tiempo encontrado y modificaciones realizadas en base a este problema, no se logró obtener un programa 100% funcional, pero si se logró realizar una que otra funcionalidad.
 3. A pesar de los inconvenientes y de no poder darle solución a la problemática de manera totalmente correcta, realizando este laboratorio se pudieron reforzar todos los conocimientos vistos en el curso, logrando observar de manera más explícita el comportamiento y funcionamiento del sistema SAP y logrando comprenderse mejor; esto se ve reflejado en los diferentes cambios realizados en el código que se intentan acercar a los requerimientos solicitados.