Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte final

Software emulador de un microprocesador 8086 con fines didácticos

Proyecto Tecnológico

Trimestre 14 Primavera

Elaborado por

Francisco David Meneses Bautista

Matrícula: 210206171

Asesora

M. en C. María Antonieta García Galván

Departamento de Electrónica

29 de agosto de 2014

DECLARATORIA

Yo, M. en C. María Antonieta García Galván, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

M. en C. María Antonieta García Galván
Profesor Titular B
Departamento de Electrónica

Yo, Francisco David Meneses Bautista, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Francisco David Meneses Bautista

Tabla de contenido

Tabla de figuras

Introducción

Las instituciones educativas que imparten carreras del área de la computación, la electrónica y otras afines a estos campos, requieren de herramientas de software para facilitar la enseñanza de las asignaturas relacionadas con la arquitectura de computadoras. Algunas de las herramientas que se utilizan actualmente incluyen a los ensambladores, emuladores, paquetería para diseño lógico y software de lenguajes de descripción de hardware, entre otros. Sin embargo, existe el problema de que el uso de estos programas se encuentra muchas veces sujeto al pago de licencias, que pueden dificultar el acceso a ese software, sobre todo a las instituciones públicas.

Otro problema es que las características de los programas existentes suelen no corresponderse con los contenidos de los programas de estudio. Para solventar esta situación, se hace necesario emplear un conjunto de aplicaciones distintas para enseñar los diferentes temas que se cubren en los cursos de arquitectura de computadores. Por ejemplo, suelen emplearse ensambladores para enseñar la programación del lenguaje ensamblador, a la vez que se necesita usar paquetes de diseño lógico para mostrar la interacción entre los componentes del hardware.

Para cubrir estas necesidades, se propuso la creación de un software emulador del microprocesador 8086, que incorporara algunas de las características específicas del chip, haciéndolo ideal para usarse como apoyo para esos cursos.

La mayor prioridad en el diseño del emulador propuesto recaerá en la representación gráfica de los cambios de estado al interior del microprocesador. Esto permitirá visualizar claramente la transferencia de información entre los componentes internos del microprocesador y la interacción con los dispositivos periféricos externos.

Justificación

La mayoría de los programas que pueden aplicarse en la enseñanza de arquitectura de computadoras y microprocesadores no fueron concebidos para demostrar las características del funcionamiento interno de un microprocesador. Los ensambladores sólo convierten los programas fuente en programas ejecutables, pero la parte de la arquitectura del procesador ya debe ser conocida de antemano y el ensamblador no da ningún tipo de información al respecto. Por el contrario, los programas orientados al diseño lógico del hardware no muestran los detalles del comportamiento a nivel de software, su objetivo no es mostrar la interacción del software con el hardware.

En este proyecto se incluyó tanto la parte del ensamblado de programas fuente a código máquina, como la visualización de la ejecución del programa a nivel interno del microprocesador. De esta manera, el emulador podrá ser usado para facilitar el entendimiento del proceso de creación, carga y ejecución de los programas a bajo nivel en una computadora.

Existen actualmente proyectos de software con objetivos similares. Sin embargo, el diseño de este emulador pretende ser altamente adaptable a varias plataformas de hardware y software, para que pueda ser usado por un público muy amplio sin necesidad de adquirir computadoras muy poderosas u otra paquetería adicional.

Objetivos

Objetivo general

- Diseñar e implementar un software emulador del microprocesador 8086, con las características adecuadas para ser usado como un auxiliar didáctico en cursos de arquitectura de computadoras o microprocesadores.

Objetivos particulares

- Identificar las características y funcionalidades más importantes del microprocesador 8086 que deben ser implementadas en el emulador, desde el punto de vista didáctico.
- Diseñar la estructura principal del software, haciendo hincapié en la modularidad y la extensibilidad.
- Diseñar e implementar el módulo principal del emulador del 8086.
- Determinar los módulos auxiliares que resulten imprescindibles para conseguir la funcionalidad buscada para el software.
- Diseñar e implementar los módulos básicos de entrada y salida.
- Combinar los módulos implementados para obtener el software final.
- Realizar pruebas al software.

Marco teórico

Organización básica de una computadora

Una computadora sencilla es un sistema formado de tres componentes principales:

- *Microprocesador* o unidad central de proceso. Es el componente principal de la computadora. El microprocesador está diseñado para efectuar un conjunto de operaciones aritméticas, lógicas y de transferencia de información. Es el responsable de ejecutar los programas del usuario.
- *Memoria.* Es un circuito formado por una serie de registros, capaces de almacenar información. La memoria es el medio donde el microprocesador busca y deposita los datos que emplea durante la ejecución de un programa.

- *Dispositivos de entrada/salida*. Son dispositivos auxiliares conectados a la computadora. Estos dispositivos intercambian información con la computadora, ya sea suministrarla (entrada) u obtenerla (salida). A menudo estos dispositivos se diseñan para que los usuarios tengan una interacción más natural con la computadora.

La computadora IBM PC

En la década de los ochentas muchas compañías apostaron al desarrollo de las computadoras domésticas o microcomputadoras. Entre ellas, la empresa IBM presentó en agosto de 1981 una microcomputadora a la que llamó IBM PC (IBM Personal Computer). Esta computadora estaba diseñada sobre el microprocesador Intel 8088 de 16 bits, corría a 4.77MHz y contaba con capacidad en RAM desde 16Kb hasta 64Kb.

El impacto comercial que tuvo la IBM PC fue muy notable. Los consumidores utilizaban la IBM PC como una herramienta seria de trabajo, y tenían la confianza en la calidad que ofrecía IBM en todos sus productos. Poco a poco, la IBM PC superó las ventas de sus competidores de otras marcas.

IBM decidió desarrollar una arquitectura abierta para la PC, poniendo disponibles los detalles de la construcción y funcionamiento de la máquina a los desarrolladores de hardware. La arquitectura abierta fomentaba la creación de más dispositivos periféricos diseñados para la PC, sin cobrar patentes. Al tener esta información disponible, algunos fabricantes de computadoras pudieron diseñar máquinas compatibles con la IBM PC, convirtiendo su arquitectura en un estándar industrial que aún hoy perdura en muchas computadoras personales modernas, varias décadas después.

El microprocesador Intel 8086

Los procesadores Intel 8086 y 8088 son sucesores del procesador Intel 8080 de 8 bits. Los procesadores 8086 y 8088 son prácticamente idénticos, su principal diferencia es el tamaño de su bus de datos: en el 8086 es de 16 bits mientras que en el 8088 es de 8 bits.

El 8086 es un procesador de 16 bits, trabaja con operandos de 8 y 16 bits, tiene capacidad de direccionamiento de memoria de 20 bits (hasta 1 MB) y cuenta con 65536 puertos de entrada/salida (64K). Dispone de 92 tipos de instrucciones, con 7 modos de direccionamiento. Su frecuencia de reloj puede ser de 4.77 MHz, 8 MHz o 10 MHz, dependiendo del modelo del chip.

El procesador cuenta con tres grupos de registros de 16 bits:

- Registros de propósito general: acumulador (AX), base (BX), contador (CX), y dato (DX).
- Registros de propósito específico: apuntador de pila (SP), apuntador base (BP), índice fuente (SI) e índice destino (DI)

- Registros de segmento: segmento de código (CS), segmento de datos (DS), segmento de pila (SS) y segmento extra (ES)

Además, dispone de un registro apuntador de instrucción (IP) y un registro de estado (FLAGS).

La arquitectura interna del 8086 está divida en una unidad de ejecución (EU) y una unidad de interfaz de bus (BIU). La EU ejecuta las instrucciones. La BIU busca instrucciones y operandos, y escribe resultados. Dado que ambas unidades trabajan simultánea e independientemente, se consigue un incremento en la velocidad de la ejecución del microprocesador.

Para mejorar más la velocidad de ejecución, el 8086 incorpora una cola de instrucciones. En ella se almacena una pequeña porción de la memoria (6 bytes a partir de la dirección codificada en el registro apuntador de instrucción y el registro de segmento de código). Esta cola facilita la búsqueda posterior de operandos o de las instrucciones siguientes a la que se ejecuta en el microprocesador, al ya no requerirse tantas lecturas adicionales a la memoria RAM.

La figura 1 muestra la estructura interna del microprocesador. Allí se puede apreciar la interconexión entre los componentes a través de un bus compartido.

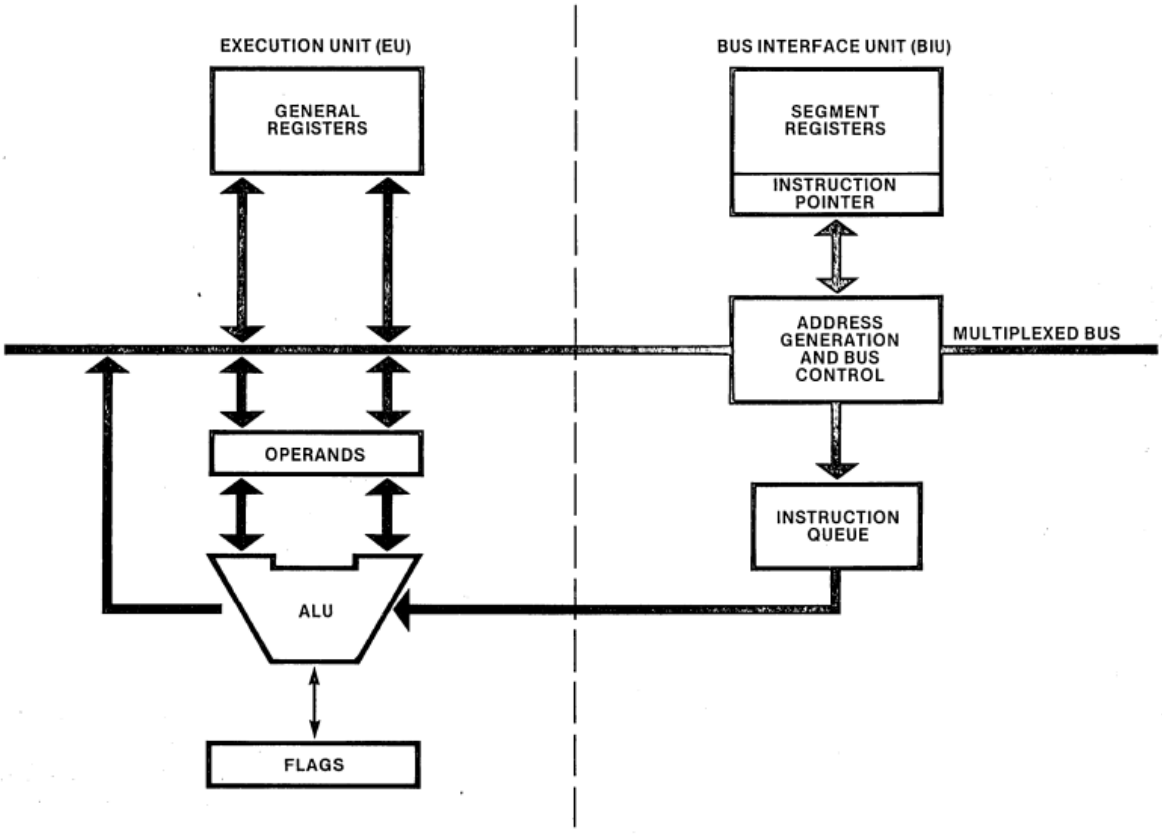

Figura 1. Arquitectura del microprocesador 8086.

Desarrollo del proyecto

Módulos principales del proyecto

El emulador desarrollado en este proyecto se encuentra conformado por tres módulos. En primer lugar, está el propio emulador del CPU 8086, encargado de cargar, decodificar y ejecutar los programas que reciba como entrada. Por otro lado, se implementaron dos programas auxiliares que sirven para emular un dispositivo de entrada y otro de salida. La arquitectura general de la aplicación puede observarse en la figura 2.

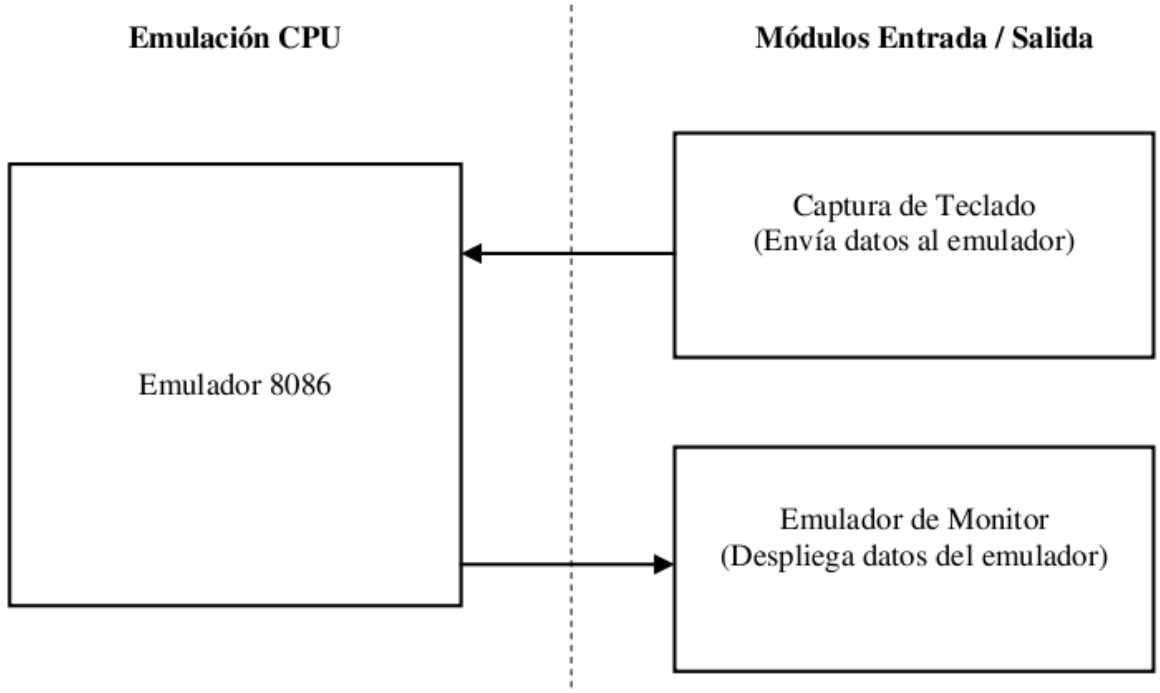

Figura 2. Arquitectura general del proyecto

Desarrollo e implementación del módulo emulador

El módulo emulador se implementó en una clase denominada CPU. Esta clase está compuesta a su vez por instancias de objetos de las clases Word, ALU, Memory, Ports, Operands y Registers. A continuación se describe el funcionamiento de cada una de estas clases:

- *Word.* Es una clase que permite disponer de un valor entero de 16 bits, como las palabras del 8086. Tiene métodos para consultar el valor de su parte alta (8 bits más significativos) como el de su parte baja (8 bits menos significativos).
- *ALU.* Contiene los métodos que definen las operaciones principales del microprocesador en la emulación. Contiene métodos para operaciones aritméticas, lógicas, de transferencia de datos, de manejo de pila y de banderas.
- *Memory.* Esta clase modela la memoria RAM del emulador. Tiene encapsulado un arreglo de byte y los métodos necesarios para leer el contenido del arreglo, en tamaño byte o palabra (dos bytes).

- *Ports.* Esta clase modela los puertos del emulador. Es similar a la clase Memory.
- *Operands.* Esta clase modela los operandos de la ALU del microprocesador. Un objeto de clase Operands contiene la información sobre el tipo de operando y su valor, tanto para el operando fuente como para el operando destino.
- *Registers.* Esta clase encapsula los registros disponibles en el 8086. Cada registro es un objeto de la clase `Word`.
- *CPU.* Es la clase principal del módulo de emulación. Su método principal, `init`, arranca la emulación. La clase CPU tiene otros métodos para controlar la emulación: pararla, cambiar a modo paso a paso y reiniciarla.

Una clase auxiliar de suma importancia en el desarrollo del módulo emulador es la clase `Const`. En esta clase se agruparon varias constantes que ayudan identificar objetos como los registros o los tipos de operandos, y a la vez, permite facilitar el proceso de ensamblado/desensamblado, porque se procuró utilizar los valores que utiliza el 8086 para codificar los registros en los códigos de operación.

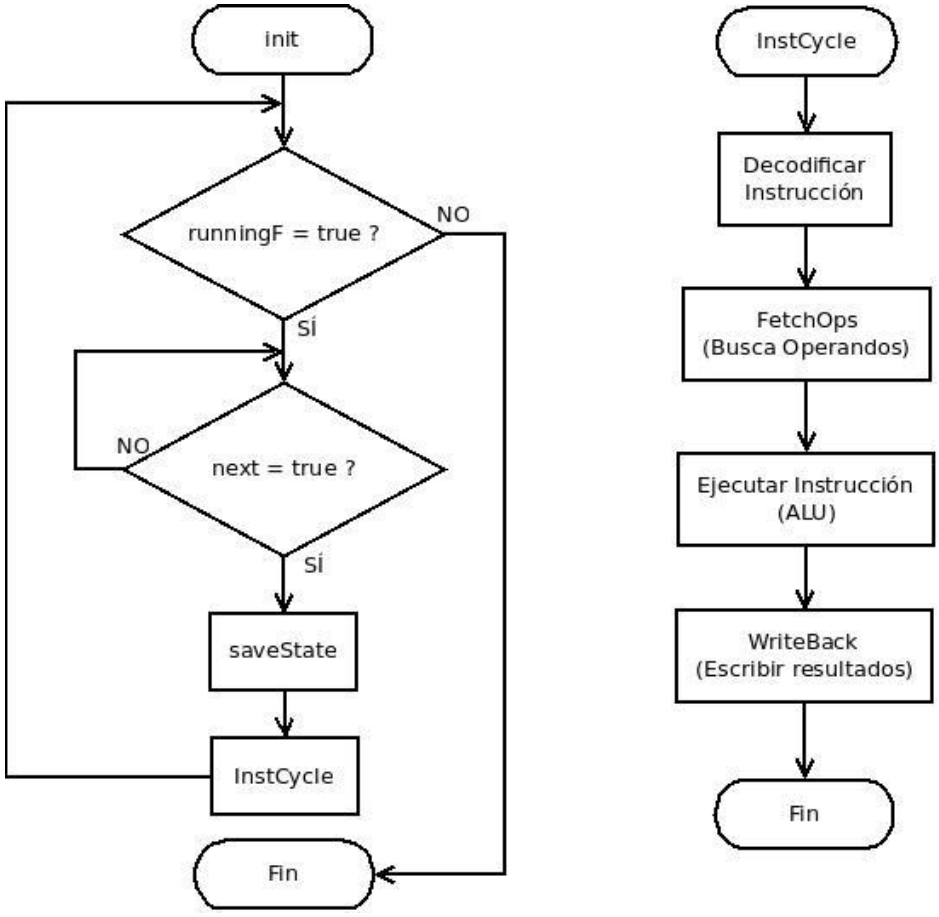

Figura 3. Métodos init y InstCycle

Al instanciar un objeto de la clase `CPU`, su constructor inicializa sus objetos miembro. Crea una memoria, registros, ALU, y puertos. En ese momento el objeto CPU está en condiciones de iniciar la ejecución de un programa. Sin embargo, se implementó el método

denominado `reset`, que establece los valores iniciales de los registros como ocurre al encender una PC.

Como ya se mencionó, el método más importante de la clase CPU es `init`. Este método pone en marcha la emulación. Al llamar este método se crea un hilo que está construido como un ciclo controlado por las variables booleanas `runningF` (que indica si el CPU está en funcionamiento) y `next` (que alterna entre el modo de ejecución paso a paso y continuo). Al interior de este ciclo, `init` hace un respaldo del estado actual del CPU, respaldando los registros a través del método `saveState`. Después, llama al método `InstCycle`, que se encarga de ejecutar la siguiente instrucción.

`InstCycle` se compone de cuatro etapas principales: búsqueda y decodificación de la siguiente instrucción, búsqueda de operandos (a través del método `FetchOps`), ejecución de la instrucción en la ALU y escritura de resultados (a través del método `WriteBack`). El proceso de ejecución se repetirá dependiendo de los valores de `runningF` y `next`. La figura 3 muestra un diagrama que permite apreciar gráficamente la operación de estos ciclos.

La búsqueda de la instrucción se implementó revisando el primer byte de la siguiente instrucción. Ese primer byte contiene el código de la operación. En algunos casos, las instrucciones también contienen codificada una parte en el segundo byte, pero basta conocer el primero para determinar de cuál instrucción se trata. Una vez que se sabe cuál es la instrucción, `InstCycle` establece el tipo de operandos que utilizará.

Para la búsqueda de operandos, el método `FetchOps` decodifica el modo de direccionamiento que emplea la instrucción y después adquiere los valores de los operandos de la memoria o de los registros.

La clase `ALU` tiene diferentes métodos que implementan algunas de las operaciones del procesador. Otras operaciones se efectúan directamente en el método `InstCycle`.

El método `WriteBack` escribe los resultados en el operando destino, que puede ser un registro o una posición de memoria.

También se implementó un método auxiliar, `DumpRegs`, que permite conocer el valor de los registros del CPU en un momento dado.

Módulos de entrada/salida

Los módulos de entrada/salida se diseñaron para comunicarse con el emulador a través de sockets. A grandes rasgos, estos módulos se componen de dos partes: un submódulo de comunicación y un módulo de tratamiento de los datos.

Módulo monitor

El módulo monitor es un ejemplo de módulo de salida. La clase monitor implementa una interfaz gráfica que pretende simular un monitor de texto a colores, como se ve en la figura 4. Este módulo puede considerarse una aplicación auxiliar independiente del emulador. Su funcionamiento básico consiste en hacer solicitudes de lectura de memoria al emulador,

consultando los 4000 bytes a partir de la dirección 0xB8000, que es donde la PC iene la memoria de video en modo texto. Cuando el emulador envía los datos al monitor, este los va desplegando. Los caracteres y colores que muestra el monitor se obtienen de archivos de imagen previamente guardados.

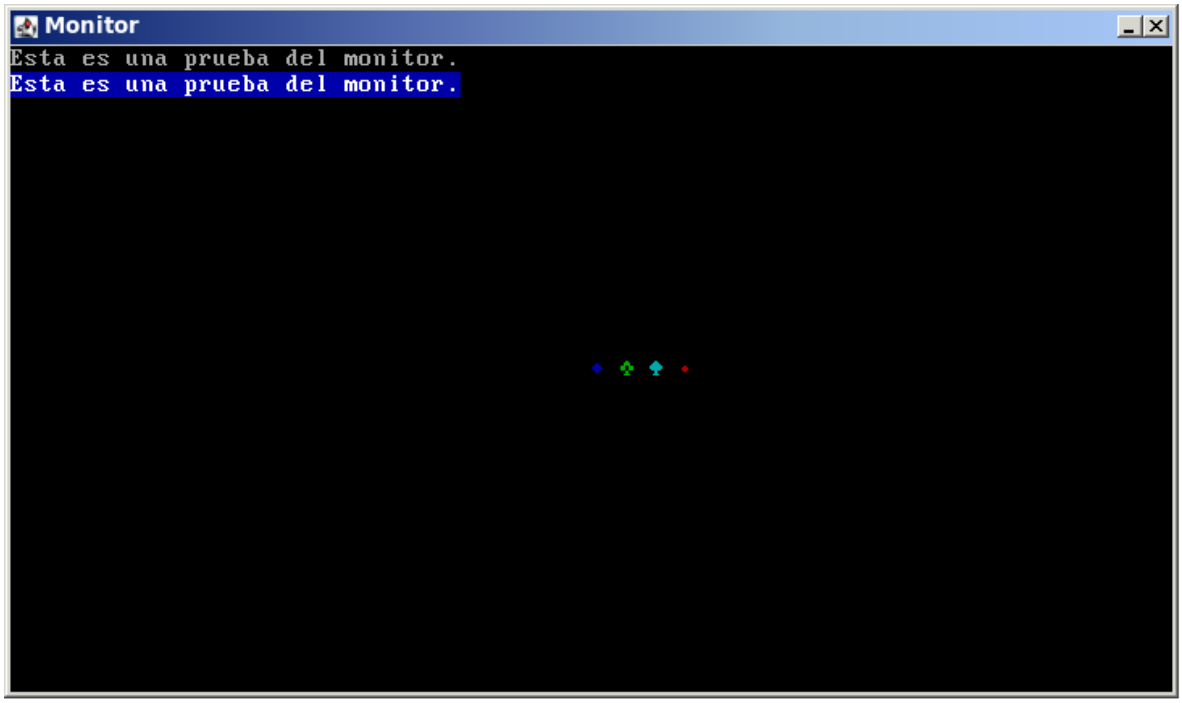

Figura 4. Monitor

Interfaces de usuario

El emulador implementa varias interfaces de usuario. La interfaz principal (Figura 5) permite acceder a todas las funcionalidades de la aplicación. Despliega un esquema del microprocesador, donde se puede observar el estado de sus registros así como la cola de instrucciones y la siguiente instrucción a ejecutarse.

Esta interfaz principal tiene las opciones para controlar la emulación, iniciar el servidor de conexión para aplicaciones de entrada/salida y abrir la interfaz de gestión de memoria.

La interfaz principal es la responsable de mostrar el estado de la ejecución de un programa tras cada instrucción. Está implementada para actualizar los valores que se despliegan cada vez que hay una modificación en los registros.

Otra interfaz es la interfaz de gestión de memoria. En esta interfaz se puede desplegar el contenido de la memoria. Se puede visualizar en modo lineal o por bancos. También es posible elegir la dirección de memoria que se quiere desplegar, y modificar su contenido. Otra función que se implementó en esta interfaz fue la de cargar y guardar el contenido de la memoria en archivos (Figura 6).

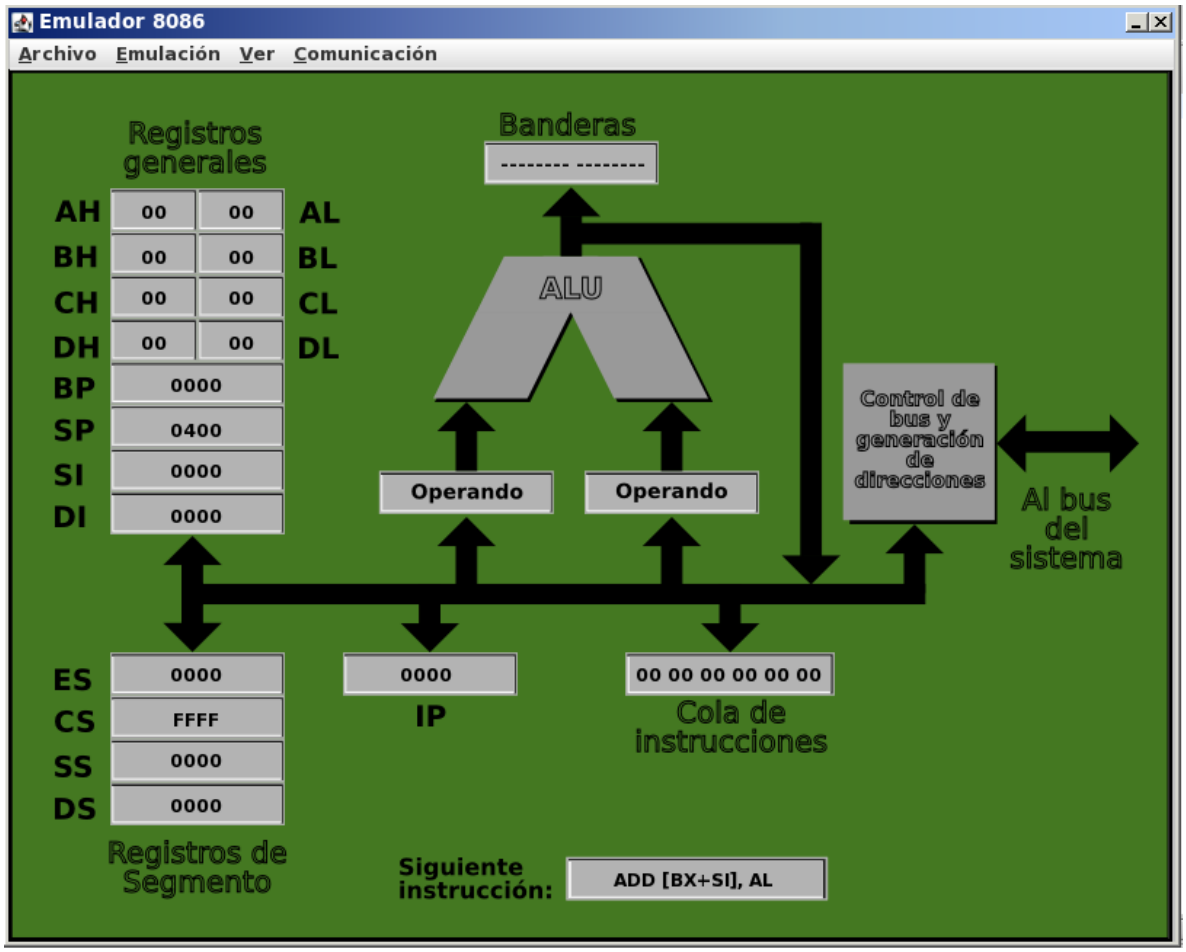

Figura 5. Interfaz principal

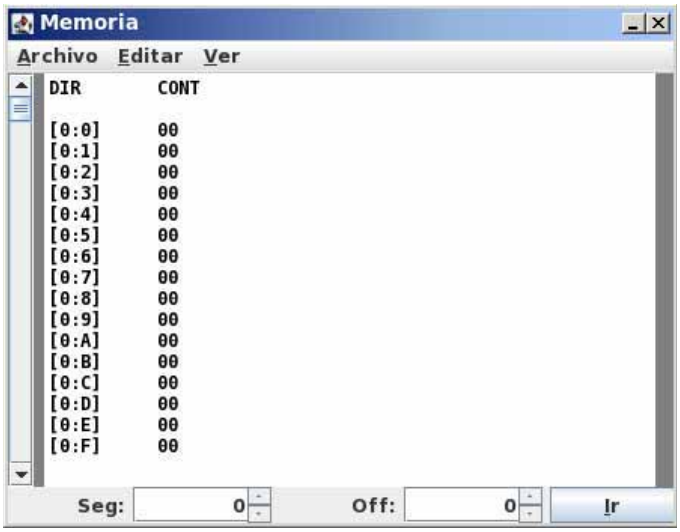

Figura 6. Interfaz de memoria

Resultados

Tras efectuarse varias pruebas al programa, se fueron detectando algunos errores que permitieron corregir el funcionamiento del emulador.

Algunas de las instrucciones del 8086 no se implementaron en el emulador. Por ejemplo, no se implementaron las operaciones de cadena. En caso de presentarse alguna de estas instrucciones en un programa, alteraría la decodificación de las instrucciones siguientes, puesto que el emulador simplemente salta esos códigos de operación que no reconoce y continúa en la siguiente posición de memoria.

A continuación se muestra un ejemplo de ejecución de una secuencia de instrucciones en el emulador. El programa de prueba fue el siguiente:

```
MOV AX, 0FF00H

MOV BX, 00FFH

ADD AX, BX
```

En la figura 7 se muestra el estado del emulador antes de ejecutar la primera instrucción.

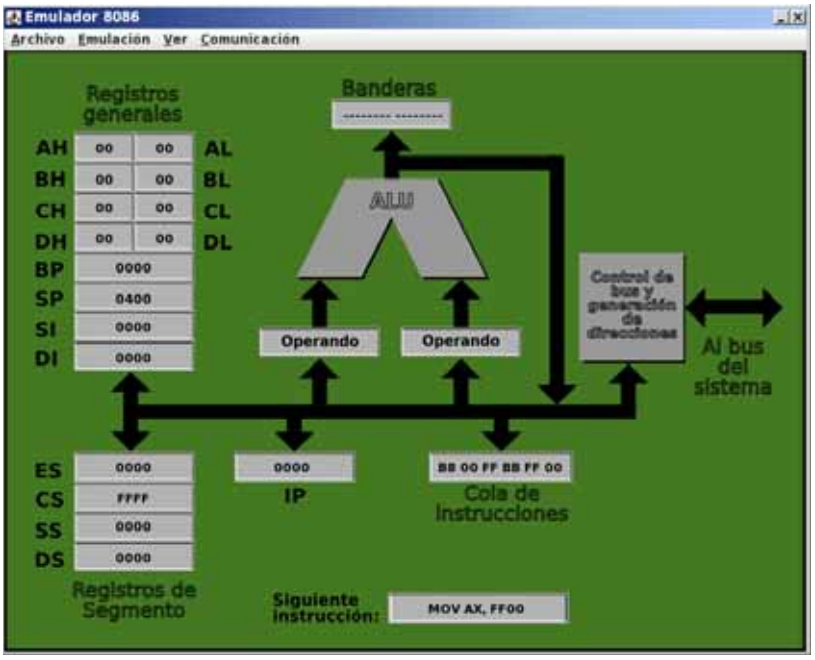

Figura 7. Estado inicial de la emulación

La figura 8 muestra el estado tras ejecutarse la primera instrucción. Nótese que se ha actualizado el valor de los registros AH y AL.

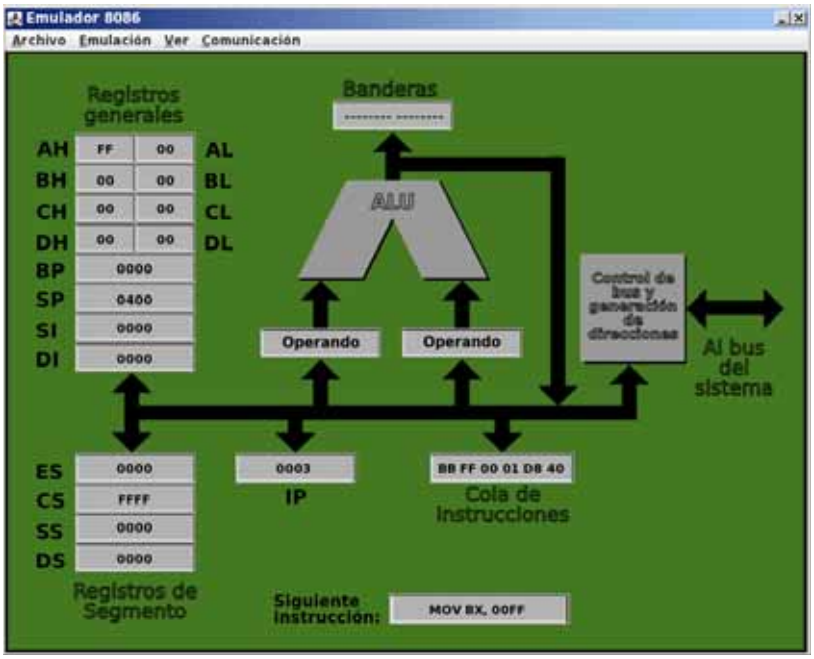

Figura 8. Resultado de la operación MOV AX, FF00h

Al ejecutarse la segunda instrucción, también se ha actualizado el valor de los registros BH y BL (figura 9).

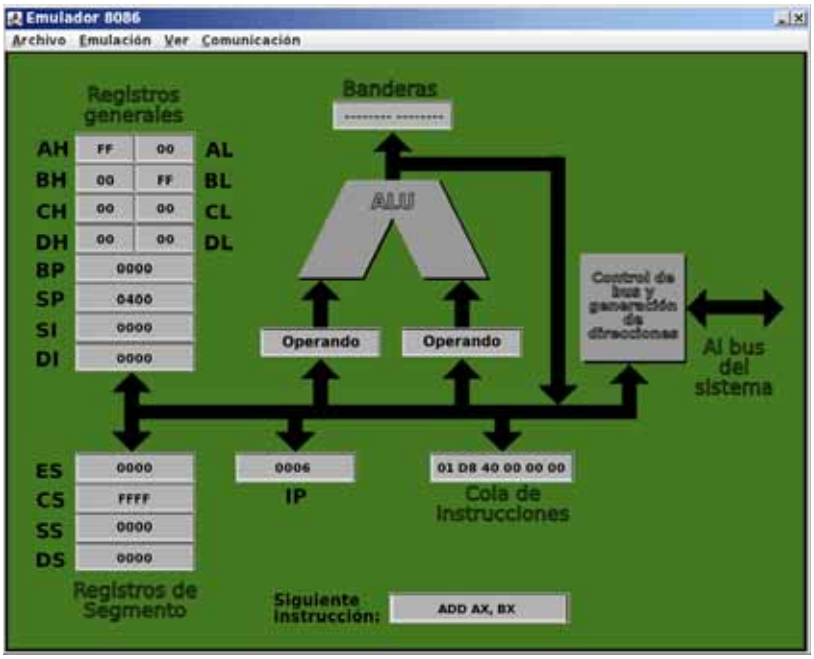

Figura 9. Resultado de la operación MOV BX, 00FFh

Por último, al ejecutarse la instrucción ADD AX, BX, los registros AH y AL se actualizan mostrando el resultado de la suma. Así mismo, se activa la bandera de paridad en el registro Banderas (Figura 10).

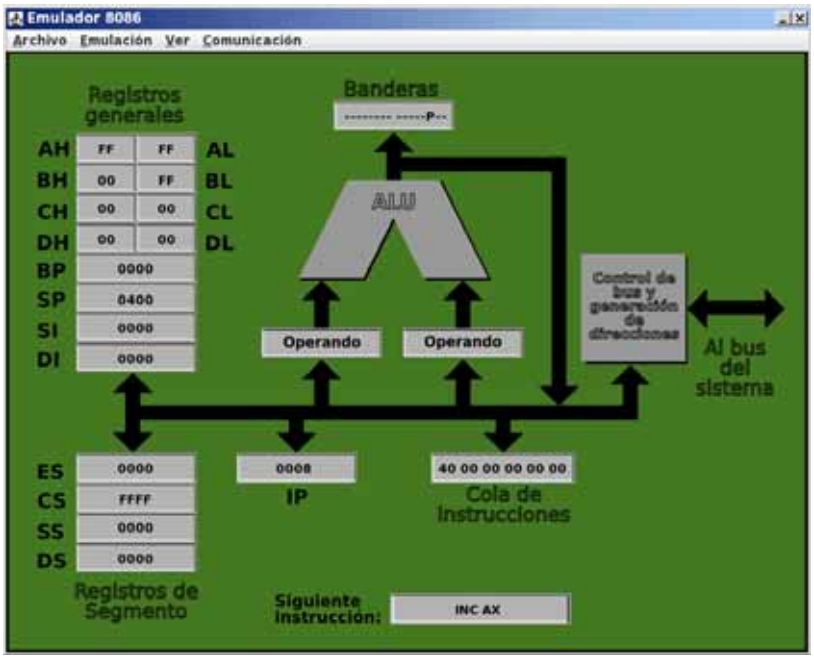

Figura 10. Resultado de la operación ADD AX, BX

Conclusiones

Los resultados de la implementación de los diferentes módulos del programa son satisfactorios. Se logró implementar la mayor parte de las instrucciones del microprocesador 8086 en la emulación. La comunicación con los módulos de entrada y salida también funciona de manera satisfactoria.

Se logró encontrar la forma de decodificar y ejecutar las instrucciones de acuerdo al modelo de clases implementado en el proyecto. Este modelo de clases facilitó también la implementación de las interfaces de usuario.

A pesar de que se encontraron ciertas dificultades a la hora de programar los módulos de comunicación, al final se implementaron exitosamente.

Con la implementación del proyecto en el lenguaje Java, además se consiguió la meta de hacer de este un programa portable en varios sistemas operativos.

Sin embargo, es necesario tener presente que este proyecto necesita ser ampliado para llegar a alcanzar su propósito original de ser usado como un apoyo para los estudiantes. Se debe complementar este trabajo terminando de implementar el conjunto completo de las instrucciones del 8086 y diseñando otros programas de entrada y salida que permitan comunicar al emulador con dispositivos de hardware real. También es posible que se necesite modificar algunas partes del programa para facilitar su uso por parte del usuario.

Bibliografía

- Intel Corporation, *"iA86, 868User's Manual"*, Intel Corporation Literature, California, Estados Unidos, 1981
- García de Celis, Ciriaco, "*El universo digital del IBM PC, AT Y PS/2"*, Grupo Universitario de Informática UVA, Valladolid, España, 1992-1997

Apéndices

Fragmento del código fuente del método InstCycle de la clase CPU

```
private void InstCycle() throws MemoryException {

            int ip_prev, ip, cs;

            Operands ops = new Operands();

            byte b, modregrm = 0;

            boolean executing = true; //true, para instruciones que
requieran más de un ciclo

            boolean wback;


            while(executing) {

                  executing=false;

                  wback = true;

                  cs = regs.getSeg(registers.CS).getIntValue();

                  ip_prev = regs.getIP().getIntValue();


                  b = mem.readB(getEAddress(cs, ip_prev));

                  if (verbose)

                        print("\nCPU.InstCycle(): Leyendo CS:IP ["  +
intHex(cs) + ":" + intHex(ip_prev) + "] = "+ byteHex(b) + "(InstFetch)");

                  // Incrementa IP

                  regs.incIP(1);

                  ip = regs.getIP().getIntValue();


                  switch (unsignedByte(b)) {

                  case 0x00: // ADD r/m8, r8

                        ops.setDstType(operands.OP_rm8);

                        ops.setSrcType(operands.OP_r8);

                        modregrm=FetchOps(ops);

                        regs = alu.ADD(regs, ops);
```

```
                break;
            case 0x01: // ADD r/m16, r16
                ops.setDstType(operands.OP_rm16);
                ops.setSrcType(operands.OP_r16);
                modregrm=FetchOps(ops);
                regs = alu.ADD(regs, ops);
                break;
            case 0x02: // ADD r8, r/m8
                ops.setDstType(operands.OP_r8);
                ops.setSrcType(operands.OP_rm8);
                modregrm=FetchOps(ops);
                regs = alu.ADD(regs, ops);
                break;
```

Fragmento de las constantes en la clase Const

```
public class Const {
    public class registers{
        /**** Constantes que identifican a los registros del CPU ***/

        /* Registros de 16 bits */
        public static final int AX=0;
        public static final int CX=1;
        public static final int DX=2;
        public static final int BX=3;
        public static final int SP=4;
        public static final int BP=5;
        public static final int SI=6;
        public static final int DI=7;

        public static final int IP=32;
```

```
/* Registros de 8 bits */
public static final int AL=0;
public static final int CL=1;
public static final int DL=2;
public static final int BL=3;
public static final int AH=4;
public static final int CH=5;
public static final int DH=6;
public static final int BH=7;

/* Registros de segmento */
public static final int ES=0;
public static final int CS=1;
public static final int SS=2;
public static final int DS=3;

/* Identificadores especiales */
public static final int FLAGS=33;
```