Ensamblador y simulador de microprocesador basado en el Intel 8086.

García García Jonathan Eduardo
jgarciag1404@alumno.ipn.mx

González Santiesteban Santiago
sgonzalezs1400@alumno.ipn.mx

Guzmán Olvera Jessica
jguzmano1701@alumno.ipn.mx

Abstracto—Este documento es el reporte técnico, explicativo de la estructura interna de este simulador basado en la arquitectura del Microprocesador INTEL 8086.

Palabras clave—component, formatting, style, styling, insert

I. Introducción

El 8086 fue diseñado para trabajar con lenguajes de alto nivel, disponiendo de un soporte hardware con el que los programas escritos en dichos lenguajes ocupan un pequeño espacio de código y pueden ejecutarse a gran velocidad. Esta concepción, orientada al uso de compiladores, se materializa en un conjunto de facilidades y recursos, y en unas instrucciones entre las que cabe destacar las que permiten efectuar operaciones aritméticas de multiplicar y dividir, con y sin signo; las que manejan En su momento, el 8086 junto con el 8088 fueron los microprocesadores más empleados dentro de su categoría, especialmente desde que IBM los adoptó para la En su momento, el 8086 junto con el 8088 fueron los microprocesadores más empleados dentro de su categoría, especialmente desde que IBM los adoptó para la cadenas de caracteres, etc.
En su momento, el 8086 junto con el 8088 fueron los microprocesadores más empleados dentro de su categoría, especialmente desde que IBM los adoptó para la construcción de su computadora personal.

II. Marco teórico

A. Microprocesador

Un microprocesador, también conocido como procesador, micro, chip o microchip, es un circuito lógico que responde y procesa las operaciones lógicas y aritméticas que hacen funcionar a nuestras computadoras. En definitiva, es su cerebro.
Pero un procesador no actúa por propia iniciativa, recibe constantemente órdenes de múltiples procedencias. Cuando encendemos nuestra computadora, lo primero que hace el micro es cumplir con las instrucciones de la BIOS (basic input/output system), que forma parte de la memoria de la computadora. Una vez funcionando, además de la BIOS, será el sistema operativo y los programas instalados los que seguirán haciéndose obedecer por el microprocesador.

B. Instrucciones

Las instrucciones serán las encargadas de decirle al procesador que hacer con esos datos, a veces los transformaran, otras se encargara de enviarlo a algún periferico.
El conjunto de instrucciones que un procesador soporta definirá que aplicaciones entiende y por tanto cuales puede llegar a ejecutar.

C. Unidad de Control

La unidad de control es el componente del procesador que dirige y coordina la mayoría de las operaciones en la computadora. La unidad de control se encarga de interpretar cada una de las instrucciones generadas por un programa y después inicia las acciones apropiadas para dirige incluyen la unidad lógico y aritmética, los registros, y los buses. llevar a cabo las instrucciones.
Los tipos de componentes internos que la unidad de control

D. Unidad Lógico Aritmética ALU

Una unidad aritmético-lógica (ALU) es la parte de un procesador de computadora (CPU) que realiza operaciones aritméticas y lógicas en los operandos en las palabras de instrucción de computadora . En algunos procesadores, la ALU se divide en dos unidades, una unidad aritmética (AU) y una unidad lógica (LU).
La finalidad primordial de la ALU consiste en aceptar datos binarios que están almacenados en la memoria y ejecutar operaciones aritméticas con estos datos, de acuerdo con instrucciones que provienen de la unidad de control.

E. Registros de datos

Los registros se encuentran dentro de cada microprocesador y su función es almacenar los valores de datos, comandos, instrucciones o estados binarios que ordenan qué dato debe procesarse, como la forma en la que se debe hacer.
Un registro no deja de ser una memoria de velocidad alta y con poca capacidad.

F. Banderas de estado

Sirven para guardar valores reales cuya función es determinar cuándo una instrucción debe ejecutarse o no. T ambién se le conoce como CCR (Condition Code Register).

G. Assembler

Cuando programamos en un lenguaje distinto del lenguaje máquina, nuestro código debe ser traducido a binario para que el ordenador pueda entenderlo y ejecutarlo. Se llaman ensambladores los programas encargados de traducir los programas escritos en ensamblador a código binario

- Tanto los compiladores como los Ensambladores caen en la categoría de programas que llamamos traductores.
- Un traductor es un programa que acepta archivos en código fuente comprensibles para el humano y genera alguna clase de archivo binario.
- El archivo binario puede ser un archivo de programa ejecutable que el CPU puede comprender, o podría ser un archivo font, o un archivo de datos binarios comprimido, o alguno de los cientos de otros tipos de archivos binarios.
- Los programas traductores generan instrucciones de maquina que el CPU puede comprender.
- Un programa traductor lee un archivo en código fuente línea por línea, y escribe un archivo binario de instrucciones de maquina que realiza las acciones de computadora que el archivo de código fuente describe. Este archivo binario es llamado archivo de código objeto

III. Desarrollo

Tanto el ensamblador como el simulador están escritos en C# "C Sharp" debido a que este lenguaje de programación implementa los pilares de POO.

- Abstracción
  Permite diseñar cada uno de los modulos de un microprocesador a nivel de objetos.
- Polimorfismo
  Permite sobrecargar métodos teniendo un único patrón en conmún.
- Encapsulamiento
  Protege los datos almacenados en la memoria limitando la lectura ó escritura de los mismos a menos que seán autorizados explicitamente.
- Herencia
  Permite compartir fragmentos de código entre clases heredando métodos y atributos evitando la redundacia en el código.

A. Llevando la escala de instrucciones a nivel de bits en alto nivel.

En C-sharp como tál no existe un tipo de dato primitivo que reciba la denominación de bit, sin embargo existe el tipo de dato "Booleano" que unicamente permite almacenar dos valores al igual que un bit de información:

- Verdadero, cuyo correspondiente en bit seria 1
- Falso interpretado como 0

B. Generalidades

El programa simula un microprocesador basado en la arquitectura de los procesadores Intel 8086.
Las unidades con las que cuenta esta procesador son:

- Unidad Lógica Artimetica.
- Unidad de Control.
- Registros de propostito General.
- Memoria de programa.
- Memoria del programador.

C. Set de instrucciones

Cada instrucción tiene una longuitud de 32 bits.
Formato de instrucción:

- Código de operación: 6 bits.
- Modificador (Indican el tipo de direccionamiento y loguitud de la instrucción): 4 bits.
- Identificador de registro: 4 bits.
- Número, valor binario a 32 bits.

#	INSTRUCCIONES	CÓDIGO OPERACIÓN
1	MOV	000001
2	ADD	000010
3	SUB	000011
4	DIV	000100
5	MUL	000101
6	NOT	000110
7	OR	000111
8	NOR	001000
9	XOR	001001
10	XNOR	001010
11	AND	001011
12	NAND	001100
13	CMP	001101
14	JMP	001110
15	JZ	001111
16	JE	010000
17	JNZ	010001
18	JNE	010010
19	JC	010011
20	JA	010100
21	JAE	010101
22	JLE	010110
23	JO	010111
24	JNS	011000
25	JNO	011001
26	ETIQUETA	011010
27	JL	011011
28	begin	011100
29	LOOP	011101
30	DB	011110
31	RET	011111

MODIFICADOR	TIPO	EJEMPLO
0001	POR REGISTRO	MOV AX,AX
0010	DIRECTO	MOV AX,[00H]
0011	INMEDIATO	MOV AX,09H
0100	INDIRECTO	MOV AX,[SI/DI]
0101	INDEXADO	MOV AX[BX+SI/DI]
0110	SIMPLE	MUL AX
0111	SALTO	JUMP ETIQUETA
1000	FIN DE PROGRAMA	RET
1001	DIRECTO INVERSO	MOV [00H],AX
1010	INDIRECTO INVERSO	MOV [DI/SI],AX
1011	INDEXADO	MOV [BX+SI/DI],AX

D. Registros de proposito general

#	REGISTROS	IDENTIFICADOR
1	AX	0001
2	AH	0010
3	AL	0011
4	BX	0100
5	BH	0101
6	BL	0110
7	CX	0111
8	CH	1000
9	CL	1001
10	DX	1010
11	DH	1011
12	DL	1100
13	SI	1101
14	DI	1110

- IP
  Contador del Programa.
  (índice de programa) almacena el desplazamiento dentro del segmento de código. Este registro junto al registro CS apunta a la dirección de la próxima instrucción. No puede ser usado como operando en operaciones aritmético/lógicas.
- IR:
  Registro de Instrucción. Es un espacio temporal de memoria para almacenar la instrucción que se está ejecutando en ese momento.
- IA
  Registro de Trabajo. Es al que se podrá cargar datos directamente, cargarlos de la memoria o guardarlos a la memoria, y donde se guardaran los resultados de la ALU.

E. Banderas de estado

BANDERAS		
ACARREO	SIGNO	ZERO
OVERFLOW		

- ACARREO
  (bit de acarreo) vale 1 si se produce acarreo en una operación de suma, o acarreo negativo en una operación de resta. Contiene el bit que ha sido desplazado o rotado fuera de un registro o posición de memoria. Refleja el resultado de una comparación.
- SIGNO:
  (indicador de signo) solo tiene sentido en las operaciones con signo. Vale 1 cuando en una de estas operaciones el signo del resultado es negativo.
- ZERO
  (indicador de cero) vale 1 cuando el resultado de una operación es cero.

IV. Ensamblador (Compilador)

Se describen las fases del ensamblador:

A. Analizador sintactico

En esta primera etapa se clasifica cada palabra del código fuente en ensamblador en tokens, en caso de no identificar algún token el compilador envia el mensaje de ”Sentencia no reconocida”.

B. Token

Es la parte minima de cada instrucciones , las instrucciones están compuestas por uno ó varios token, como resultado del analisís anterior.

C. Analizador léxico

Valida que cada linea este escrita en un orden coherente y las clasifica en objetos del tipo ”Lina léxica”.

D. Linea léxica

Representa a una instrucción completa que ya ha sido,clasificada, validada y debidamente estructurada.

E. Sintesís

Fase de la compilación donde se extra información de todas las instrucciones en el programa preparandolas para su traducción final en binario. Cabe destacar que esta etapa se realiza en dos fases debido a que en la primera se debe calcular la dirección de memoria de cada etiqueta para poder apuntar correctamente al segmento de memoria que contiene la instrucción de salto.

F. Código máquina

Fase final donde se traduce el resultado de todas las etapas en un único archivo de código binario utilizando las definiciones de código de operación,Modificador y longuitud de instrucción.

Fig. 1: Diagrama de clases de assambler/compilador de ensamblador.

V. Proceso de Ejecución

Se obtiene la primera instrucción y se le da la Unidad de Control. Para esto el Registro de Instrucción (IR) deberá de tener la instrucción, ya que es ahí donde la Unidad de Control la busca.

La Unidad de Control se ayuda del Analizador lexico para saber el tipo de instrucción que ha recibido y el operando, y con ello moverá datos a diferentes registros, y le dirá a la ALU que haga una determinada operación.

Para finalizar, la Unidad de Control aumentara a IP, el contador de programa, para poder seguir con la siguiente instrucción, lo cual devuelve al programa al paso 2 para procesar la siguiente instrucción.

El programa terminara cuando se ejecute la última instrucción del programa.

A. Clases del Programa

Cada clase representa un componente diferente en el microprocesador, y contiene sus métodos para cumplir con la función.

- Registros
  Almacena los registros de proposito general definidos en la arquitectura.
- Registro
  Abstracción de un registro de propostito general.
  Permite el acceso a la parte alta y baja de cada registro así como habilitar/deshabilitar la lectura y escritura de los mismos
- CPU
  Unidad de control central
  Clase principal desde donde se llaman a todos los módulos para ejecutar cada una de las instrucciones contenidas en la pila de instrucciones.
- Memoria
  Pila de instruciones y memoria del programador.
  Se almacenan el programa a ejecutar.
- ALU
  Aqui es donde se realizan las operaciones lógicas y aritméticas a nivel de bits. (Micro-código)
- Banderas: En esta clase se guarda el estado de las banderas durante la ejecución del código

VI. Código de las clases principales

A. CPU

CPU

```
    public static class CPU
    {
        public static Alu.Alu Alu { get; private set; }
        public static Banderas Banderas { get; private set; }
        public static Memoria Memoria { get; private set; }
        static CPU()
        {
            CPU.Alu = new Alu.Alu();
            CPU.Banderas = new Banderas();
            CPU.Memoria = new Memoria();
            Reset();
        }
        public static void Reset()
        {
            CPU.Banderas.Clear();
            CPU.Memoria.Clear();
            Registros.Registros.Reset();
        }
        public static void Ejecutar(bool[] Operacion, bool[] Modificador,
         bool[] Operador1, bool[] Operador2){...}
    }
```

B. ALU

ALU

```
    public class Alu
    {   public const int Byte = 16;
        public bool[] Resultado = new bool[Byte * 2 + 1];
        public void ADD(bool[] Operador1, bool[] Operador2){ ... }
        private bool HALF_ADD(bool A, bool B){ ... }
        private bool FULL_ADD(bool A, bool B){ ... }
        public void SUB(bool[] Operador1, bool[] Operador2){ ... }
        public bool[] COMPLEMENTO_2(bool[] Operador1){ ... }
        private bool AND(bool A, bool B){ ... }
        public void AND(bool[] Operador1, bool[] Operador2){ ... }
        public void OR(bool[] Operador1, bool[] Operador2){ ... }
        public void NAND(bool[] Operador1, bool[] Operador2){ ... }
        public void NOR(bool[] Operador1, bool[] Operador2){ ... }
        public void MUL(bool[] Operador2){ ... }
        public void NOT(bool[] Operador1){ ... }
        private bool XOR(bool Operador1, bool Operador2){ ... }
        public void XOR(bool[] Operador1, bool[] Operador2){ ... }
        public void XNOR(bool[] Operador1, bool[] Operador2){ ... }
        public void DIV(bool[] Divisor){ ... }
    }
```

C. Registros

Registros

```
1     public static class Registros
2     {
3         public static Registro AX { get; private set; }
4         public static Registro BX { get; private set; }
5         public static Registro CX { get; private set; }
6         public static Registro DX { get; private set; }
7         public static Registro SI { get; private set; }
8         public static Registro DI { get; private set; }
9         public static Registro IP { get; private set; }
10        public static Registro IA { get; private set; }
11        public static Registro IR { get; private set; }
12        static Registros()
13        {
14            Registros.AX = new Registro("AX");
15            Registros.BX = new Registro("BX");
16            Registros.CX = new Registro("CX");
17            Registros.DX = new Registro("DX");
18            Registros.SI = new Registro("SI");
19            Registros.DI = new Registro("DI");
20            Registros.IP = new Registro("IP");
21            Registros.IA = new Registro("IA");
22            Registros.IR = new Registro("IR");
23        }
24        internal static void Reset()
25        {
26            Registros.AX.Clear();
27            Registros.BX.Clear();
28            Registros.CX.Clear();
29            Registros.DX.Clear();
30            Registros.SI.Clear();
31            Registros.DI.Clear();
32            Registros.IP.Clear();
33        }
34    }
```

D. Registro

Registro

```
1     public class Registro : Localidad
2     {
3         public string Nombre { get; private set; }
4         private ParteRegistro High;
5         public ParteRegistro Low;
6         public void SetHigh(bool[] High){ ... }
7         public void SetLow(bool[] Low){ ... }
8     }
```

E. Memoria

Memoria

```
public class Memoria
{
    public bool[] this[bool[] direccion]
    {
        set
        {
            Escribir(direccion, value);
        }
    }
    private ObservableCollection<Celda> Real;

    public void Cargar(string CodigoMaquina) { ... }
    public void Cargar(bool[][] programa) { ... }
    public static bool[] CalcularDireccion(bool[] Numero) { ... }
    internal void Clear() { ... }
    public bool[] Leer(bool[] direccion) { ... }
    public void Escribir(bool[] direccion, bool[] Valor){ ... }
}
```

F. Banderas

Banderas

```
public class Banderas :
{
    private bool Carry;
    private bool Signo;
    private bool Zero;
    private bool OverFlow;
    internal void Clear()
    {
        Carry = false;
        Signo = false;
        Zero = false;
        OverFlow = false;
    }
}
```