

Z80 Documentacion

Marcel Julian Martinez Vanegas and Jose David Salazar Moreno

Universidad Nacional de Colombia

Facultad de Ingenieria

Departamento de Ingenieria de sistemas e industrial

Estudiantes de pregrado de Ingenieria de sistemas y computacion

27 de Junio del 2019

Contents

1	Especificaciones del Usuario	3
1.1	Sintaxis y ejecucion	3
1.2	Ejemplos	3
1.3	Advertencias	3
2	Especificaciones Tecnicas	3
2.1	Almacenar	3
2.2	Relocalizable	5
2.3	Enlazar y cargar	6
2.3.1	Enlazar	6
2.3.2	Cargar	7
2.4	Registros	8
2.5	Comandos del operador del Z80	8
2.6	InstruccionesCompilador y ContenedorInstrucciones	10
2.7	Memoria	10

1 Especificaciones del Usuario

1.1 Sintaxis y ejecucion

Todos los comandos en codigo ensamblador deben estar escritos de la siguiente forma para su correcta lectura por parte del emulador:

'Etiqueta'(Si la posee)'#'(Simbolo numeral)'Comando'(El nombre del comando, separado por espacio y sus variables).

Para ejecutar el codigo primero debera cargarlo (boton cargar) luego seleccionar la posicion de memoria inicial y finalmente proceder con la ejecucion paso a paso del codigo.

1.2 Ejemplos

Suma simple:

```
#LD      A,5  
#LD      B,7  
#ADD     A,B  
#HALT
```

Resta simple:

```
#LD      A,7  
#LD      B,5  
#SUB     A  
#HALT
```

Bucle infinito:

```
bucle#INC      A  
#LD      (4000),A  
#JP      bucle
```

1.3 Advertencias

- El programa no reconocera nada que no cumpla la sintaxis establecida.
- La mayoría de los comandos existentes en el microprocesador z80 no han sido implementados funcionalmente.

2 Especificaciones Tecnicas

2.1 Almacenar

```
public void almacenar() throws IOException{  
    instruccionesCompilador();  
    String input;
```

```

FileReader f = new FileReader("archivo.txt");
BufferedReader b1 = new BufferedReader(f);
String []prueba;
String auxIn [];
String auxEtiquetas [];
input = b1.readLine();
do {

    auxEtiquetas = input.split("#");
    if (!auxEtiquetas[0].isEmpty()) {
        etiquetas.put(auxEtiquetas[0], ""+auxDireccion);
    }
} while ((input = b1.readLine())!=null);
b1.close();

FileReader f2 = new FileReader("archivo.txt");
BufferedReader b2 = new BufferedReader(f2);
input = b2.readLine();
int lineCout =0;
do {
    auxEtiquetas = input.split("#");
    etiquetas.put(Integer.toHexString(auxDireccion)+"&", ""+lineCout);
    lineCout++;
    if (!auxEtiquetas[0].isEmpty()) {
        etiquetas.replace(auxEtiquetas[0], ""+auxDireccion);
    }
    input = auxEtiquetas[1];
    auxIn = isInstruccionFuente(input);
    prueba = auxIn[0].split(",");
    switch (prueba.length) {
        case 1:
            cargar1byte(auxIn[0].split(","));
            break;
        case 2:
            cargar2bytes(auxIn[0].split(","), auxIn[1]);
            break;
        case 3:
            cargar3bytes(auxIn[0].split(","), auxIn[1]);
            break;
        case 4:
            cargar4bytes(auxIn[0].split(","), auxIn[1]);
            break;
    }
} while ((input = b2.readLine())!=null);
b2.close();
File ft = new File("achivoReLoc.txt");

```

```

ft.createNewFile();
FileWriter flwriter = new FileWriter(ft);
int i=direccionInicial;
input = Memoria[i];
do {
    flwriter.write(input+" ");
    i++;
} while ((input = Memoria[i])!=null);
flwriter.close();
File fEtit = new File("archivoEtiquetas.txt");
fEtit.createNewFile();
FileWriter flEtitWriter = new FileWriter(fEtit);
Enumeration e = etiquetas.keys();
Object clave;
Object valor;
while( e.hasMoreElements() ){
    clave = e.nextElement();
    valor = etiquetas.get( clave );
    flEtitWriter.write(clave+" "+valor+"\n");
}
flEtitWriter.close();
conjuntoInstrucciones();
PC = direccionInicial;
}

```

Empieza leyendo "archivo.txt" el cual va a contener nuestro código en lenguaje ensamblador del Z80, para posteriormente cargar el archivo en el "archivoReLoc.txt" el cual es la transformación del código en ensamblador en código decimal, para poder ser cargado en memoria en el enlazador cargador.

2.2 Relocalizable

```

public void mostrarRelok() {
    FileReader f = null;
    try {
        String input;
        f = new FileReader("archivoReLoc.txt");
        BufferedReader b = new BufferedReader(f);
        input = b.readLine();
        String [] auxIn = input.split(" ");
        input = "";
        for (int i = 0; i < auxIn.length; i++) {
            input = input+(i+1)+"\t"+auxIn[i)+"\n";
        }
        txtSalidaRelok.setText(input);
        b.close();
    }
}

```

```

    } catch (FileNotFoundException ex) {
        Logger.getLogger(enlazador.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(enlazador.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            f.close();
        } catch (IOException ex) {
            Logger.getLogger(enlazador.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

En esta parte del código lo que hace es mostrar nuestro archivo relocizable antes de mandarlo a enlazar y cargar por medio de la siguiente función `liker` del procesador.

2.3 Enlazar y cargar

2.3.1 Enlazar

```

public void liker(int partida) throws IOException{
    etiquetas.clear();
    String et;
    FileReader fEt = new FileReader("archivoEtiquetas.txt");
    BufferedReader bEt = new BufferedReader(fEt);
    String auxEt [];
    while ((et = bEt.readLine())!=null) {
        auxEt = et.split(" ");
        etiquetas.put(auxEt[0],auxEt[1]);
    }
    bEt.close();
    for (int i = 0; i < Memoria.length; i++) {
        Memoria[i]="00000000";
    }

    String input;
    FileReader f = new FileReader("archivoReLoc.txt");
    BufferedReader b = new BufferedReader(f);
    input = b.readLine();
    int posicion;
    String [] auxIn = input.split(" ");
    for (int i = 0; i < auxIn.length; i++) {
        if (etiquetas.containsKey(auxIn[i].replace("&low", "")) || etiquetas.containsKey(
            auxIn[i] = etiquetas.get(auxIn[i].replace("&low", ""));

        posicion = Integer.parseInt(auxIn[i]) ;
    }
}

```

```

        auxIn[i] = conversorINTtoHEXA(partida+posicion) ;
        Memoria[partida+i]= toCadenaBinaria(Arrays.toString(conversorHEXAtobin(auxIn[i]))) ;
        Memoria[partida+i+1]= toCadenaBinaria(Arrays.toString(conversorHEXAtobin(auxIn[i+1]))) ;
        i++;
    }else{
        Memoria[partida+i]= auxIn[i];
    }
}
b.close();

File ft = new File("archivoBin.txt");
ft.createNewFile();
FileWriter flwriter = new FileWriter(ft);
for (int i = 0; i < Memoria.length; i++) {
    input = Memoria[i];
    flwriter.write(input+" ");
}
flwriter.close();

conjuntoInstrucciones();
cargar(partida);

}

```

Usando el "archivoEtiquetas.txt" verifica cuales son las etiquetas del documento para posteriormente leer el archivo "archivoReLoc.txt" el cual contiene lo que se va a cargar a la memoria la cual esta almacena en "archivoBin.txt" por medio de "cargar(int start)".

2.3.2 Cargar

```

public void cargar(int start) throws IOException{
    String input;
    FileReader f = new FileReader("archivoBin.txt");
    BufferedReader b = new BufferedReader(f);
    input = b.readLine();
    String [] auxIn = input.split(" ");
    for (int i = start; i < auxIn.length; i++) {
        Memoria[i]= auxIn[i];
    }
    b.close();

    conjuntoInstrucciones();
    PC = start;
}

```

```
}
```

Este código se implementa directamente en el código del enlazador(like) y es el que permite el almacenamiento en memoria del código a partir del de los anteriores archivos de textos definidos.

2.4 Registros

```
public void inicializarRegistros(){

    for (int i = 0; i < IO.length; i++) {
        IO[i]="00000000";
    }
    //// inicializa en cero el registro de las banderas
    regF = "00000000".toCharArray();
    regFo = "00000000".toCharArray();
    //// inicializa en cero el registro
    regA = "00000000".toCharArray();
    regB = "00000000".toCharArray();
    regC = "00000000".toCharArray();
    regD = "00000000".toCharArray();
    regE = "00000000".toCharArray();
    regH = "00000000".toCharArray();
    regL = "00000000".toCharArray();
    regAo = "00000000".toCharArray();
    regBo = "00000000".toCharArray();
    regCo = "00000000".toCharArray();
    regDo = "00000000".toCharArray();
    regEo = "00000000".toCharArray();
    regHo = "00000000".toCharArray();
    regLo = "00000000".toCharArray();
    IX = "0000000000000000".toCharArray();
    IY = "0000000000000000".toCharArray();
    direcciones = "0000000000000000".toCharArray();
}
```

Inicializamos todos los registros pedidos según la documentación del Z80 en los cuales se tienen A, B, C, D, E, H, L, como registros principales, además de sus registros auxiliares, todos ellos con 8 bits además de los registros indexados IX, IY y las direcciones, las cuales corresponden a 16 bits de longitud

2.5 Comandos del operador del Z80

Debido a lo extenso que resulta ser, se puede resumir en que cada comando del código ensamblador del Z80 se traduce en la memoria del mismo de manera única.

Un ejemplo viene siendo las instrucciones ADD


```

case "ADD":{
    switch(operandos[1]){
        case "A":{
            regA = sumarBinyBin(regA, regA,'0');
            PC++;
            break;
        }
        case "B":{
            regA = sumarBinyBin(regA, regB,'0');
            PC++;
            break;
        }
        case "C":{
            regA = sumarBinyBin(regA, regC,'0');
            PC++;
            break;
        }
        case "D":{
            regA = sumarBinyBin(regA, regD,'0');
            PC++;
            break;
        }
        case "E":{
            regA = sumarBinyBin(regA, regE,'0');
            PC++;
            break;
        }
        case "H":{
            regA = sumarBinyBin(regA, regH,'0');
            PC++;
            break;
        }
        case "L":{
            regA = sumarBinyBin(regA, regL,'0');
            PC++;
            break;
        }
    }
}

```

para las cuales cada una va a tener su propio OpCode el cual varia segun la instruccion usada.

```

contenedorInstrucciones.put("87", "ADD A,A");
contenedorInstrucciones.put("80", "ADD A,B");
contenedorInstrucciones.put("81", "ADD A,C");
contenedorInstrucciones.put("82", "ADD A,D");
contenedorInstrucciones.put("83", "ADD A,E");

```

```

        contenedorInstrucciones.put("84", "ADD A,H");
        contenedorInstrucciones.put("85", "ADD A,L");

```

Lo mismo ocurre para todas las demás instrucciones del Z80 y estas varían según el número de bits que se usen en la ejecución de la operación.

2.6 InstruccionesCompilador y ContenedorInstrucciones

Es importante mencionar estas 2 partes del código, puesto que estas nos permiten trasladarnos del OpCode a las instrucciones usadas, así como de las instrucciones usadas al OpCode de las mismas, lo cual nos favorece la navegación en el Z80 según las claves usadas para definir, puesto que ambas se definen como HashTables.

```

        contenedorInstrucciones.put("87", "ADD A,A");
        contenedorInstrucciones.put("80", "ADD A,B");
        contenedorInstrucciones.put("81", "ADD A,C");

        InstruccionesCompilador.put("ADD A,A", "87");
        InstruccionesCompilador.put("ADD A,B", "80");
        InstruccionesCompilador.put("ADD A,C", "81");

```

2.7 Memoria

```

public void ramInit(){

    String input = "00000000";
    for (int i = 1; i < memoria.length; i++) {
        input = input+",00000000";
    }
    memoria = input.split(",");
    input = "00000.00000000";
    String linea ;
    for (int i = 1; i < memoria.length; i++) {
        linea = ""+i;
        switch(linea.length()){
            case 1:
                linea = "0000"+i;
                input = input+","+linea+".00000000";
                break;
            case 2:
                linea = "000"+i;
                input = input+","+linea+".00000000";
                break;
            case 3:
                linea = "00"+i;
                input = input+","+linea+".00000000";

```

```

        break;
    case 4:
        linea = "0"+i;
        input = input+", "+linea+".00000000";
        break;
    case 5:
        linea = ""+i;
        input = input+", "+linea+".00000000";
        break;
    }

    }
    input = input.replace(",", " ,\n");
    txtRam.setText(input);
}

```

Para mostrar la memoria, la mostramos directamente en binario, mostrando ya cuando se realiza la carga del código en ensamblador y el cual se va alterando según la ejecución paso a paso del código por medio de "ram()".

```

public void ram(){
    String input = procesador.getMemoria();
    input = input.replace("[", "");
    input = input.replace("]", "");
    input = input.replace(" ", "");
    String [] aux = input.split(",");
    for (int i = 0; i < memoria.length; i++) {
        if (!aux[i].equals(memoria[i])) {
            txtRam.replaceRange(aux[i], (i*16+6), (i*16+14));
            memoria[i]=aux[i];
        }
    }
}

```