Z80 Documentación

Francisco Javier Bernal Caro, Kevin Andrey Herrera Silva, Marcel Julian Martinez Vanegas, Jose David Salazar Moreno, and Elmar Santofimio Suarez

Universidad Nacional de Colombia

Facultad de Ingenieria

Departamento de Ingenieria de sistemas e industrial

Estudiantes de pregrado de Ingenieria de sistemas y computacion

18 de Julio del 2019

Contents

1 Tokens **4**

2 Gramatica EBNF **5**

3 Diagramas de sintaxis **9**
3.1 Algoritmo 9
3.2 Lista$_i ds$ 10
3.3 Op$_l og$ 10
3.4 Expresion log 10
3.5 Expresion mat 10
3.6 Ecuaciones 11
3.7 Parametros 11
3.8 Parametro 11
3.9 Definición de conjunto 11
3.10 Definición de var 11
3.11 Lista de parámetros 12
3.12 Identificador de tipo de retorno 12
3.13 Bloque 12
3.14 Secuenciación 13
3.15 Conjunto de posibles sentencias 13
3.16 Modo de asignación 14
3.17 Expresión a evaluar y bloque de sentencias a ejecutar 14
3.18 Seleccion si no si 14
3.19 Selección IF 14
3.20 Iteración WHILE 14
3.21 Iteración DO WHILE 15
3.22 Selección SWITCH 15
3.23 Coma y asignación 15
3.24 Iteración FOR 15
3.25 Función 15
3.26 Procedimiento 15
3.27 Sentencias de función 16
3.28 Operadores aritmeticos 16
3.29 Operadores de comparación 17

4 Assembler **17**
4.1 Especificaciones del Usuario 17
4.1.1 Sintaxis y ejecucion 17
4.1.2 Ejemplos 17
4.1.3 Advertencias 18
4.2 Especificaciones Tecnicas 18
4.2.1 Almacenar 18
4.2.2 Relocalizable 20
4.2.3 Enlazar 21

4.2.4 Cargar 22
4.2.5 Registros 23
4.2.6 Comandos del operador del Z80 23
4.2.7 Instrucciones Compilador y Contenedor Instrucciones . . 25
4.2.8 Memoria 25

1 Tokens

```
Letra = [A-Za-z]
Digito5 = [0-5]
Digito6 = [6-9]
SaltoDeLinea = \n|\r|\r\n
Punto = "."
Espacio = " "
tabulador = \t|\r|\r\t
Alfanumerico2 = {Letra} | {Digito5} | {Digito6}
Alfanumerico = {Letra} {Alfanumerico2}*
Alfanumerico_esp1 = {Espacio}* {Alfanumerico}* {Espacio}*
Alfanumerico_esp = {Alfanumerico_esp1}*
Dominio2 = "com"|"net"|"edu"|"co"
Dominio3 = {Punto} {Dominio2}
Dominio = {Dominio3}+
Correo = {Alfanumerico} {Arroba} {Alfanumerico} {Dominio}
Digito = {Digito5} | {Digito6}
Numero = {Digito}+
Arroba = "@"
Nombre_Apellido = {Letra} {Letra}*
Nota = {Digito5} {Punto} {Digito5} | {Digito5} {Punto} {Digito6}
NombreCompuesto = {Nombre_Apellido} {Espacio} {Nombre_Apellido} | {Nombre_Apellido}
COMENTARIO = {SaltoDeLinea}* "#" .*? | {SaltoDeLinea}* "/*" (.*? {SaltoDeLinea}*)* "*/"
ALGORITMO = "ALGORITMO"
TERMINA = "."
ENTONCES = "entonces"
O = "||"
Y = "&&"
IGUAL = "=="
DIFERENTE = "!="
MAYOR = ">"
MENOR = "<"
MAY_IGUAL = ">="
MEN_IGUAL = "<="
SUMA = "+"
MENOS = "-"
MULT = "*"
DIV = "/"
MOD = "%"
POTENCIA = "^"
NO = "!"
DEF = "def"
RETORNAR = "retornar"
INT = "entero"
FLOAT = "decimal"
```

```
STRING = "cadena"
BOOL = "booleano"
MATRIZ = "matriz"
VECTOR = "vector"
PCOMA = ";"
ASIGNAR = "="
PAREN_AP = "("
PAREN_CI = ")"
LLAVEIZ = "{"
LLAVEDE = "}"
ANGIZ = "["
ANGDE = "]"
COMA = ","
DOSPUNTOS = ":"
VERDADERO = "verdadero"
FALSO = "falso"
NULO = "nulo"
SI = "si"
SI_NO = "si_no"
MIENTRAS = "mientras"
SELECCIONAR = "seleccionar"
CASO = "caso"
ROMPER = "romper"
HACER = "hacer"
PARA = "para"
DEFECTO = "defecto"
IDENTIFICADOR = {Alfanumerico}
ENTERO = {Numero}+
exp = "eE"
masme = "+-"
REAL = {Numero}* '.' {Digito}* | {Numero}* '.' {Digito}* {exp} {masme}? {Digito}+
i = "i"
COMPLEJO = {ENTERO} {SUMA} {ENTERO} {i} | {ENTERO} {SUMA} {REAL} {i} | {REAL} {SUMA} {REAL}
ESPACIO = " "
COMILLAS_D = \"
CADENA = \"([^\\\"]|\\.)*\"
```

2 Gramatica EBNF

```
//Símbolo inicial
 algoritmo ::= ALGORITMO IDENTIFICADOR PAREN_AP lista_ids PAREN_CI
               DOSPUNTOS bloque TERMINA
               | ALGORITMO IDENTIFICADOR DOSPUNTOS bloque TERMINA
```

```
//Lista de identificadores
lista_ids ::= IDENTIFICADOR COMA lista_ids
            | IDENTIFICADOR

//Operadores logicos
op_log ::= O
        | Y

//Expresiones logicas
expresion_log ::= PAREN_AP expresion_log PAREN_CI
                | expresion_log op_log expresion_log
                | PAREN_AP NO PAREN_AP expresion_log PAREN_CI PAREN_CI
                | variable

//Expresiones matematicas
expresion_mat ::= PAREN_AP expresion_mat PAREN_CI
                | expresion_mat op_arit expresion_mat
                | PAREN_AP MENOS PAREN_AP expresion_mat PAREN_CI PAREN_CI
                | ENTERO
                | REAL
                | variable


//Ecuaciones
ecuaciones ::= variable op_comp expresion_mat
            | expresion_log


//Parametros
params ::= param COMA params
        | param

//Parametro
param ::= variable
        | expresion_mat
        | expresion_log

//Definición de conjunto
conjunto ::= LLAVEIZ params LLAVEDE

//Definición de variable
variable ::= IDENTIFICADOR conjunto
            | IDENTIFICADOR
```

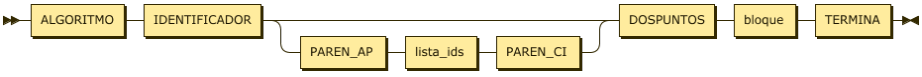
```
//Lista de parámetros
lista_parsv ::= PAREN_AP params PAREN_CI



//Identificador de tipo de retorno
tipo ::= INT
        | FLOAT
        | STRING
        | BOOL
        | MATRIZ
        | VECTOR

//Bloque
bloque ::= LLAVEIZ LLAVEDE
        | LLAVEIZ sec_proposiciones LLAVEDE


//Secuenciación
sec_proposiciones ::= proposicion sec_proposiciones
                    | proposicion


//Conjunto de posibles sentencias
proposicion ::= RETORNAR expresion_mat PCOMA
            | RETORNAR expresion_log PCOMA
            | fun_senten
            | proc_senten
            | si_senten
            | seleccionar_senten
            | mientras_senten
            | para_senten
            | hacer_mientras_senten
            | asignacion PCOMA
            | IDENTIFICADOR lista_parsv PCOMA
            | LLAVEIZ sec_proposiciones LLAVEDE
            | OTRO


//Modo de asignación
asignacion ::= IDENTIFICADOR ASIGNAR expresion_mat
            | IDENTIFICADOR ASIGNAR expresion_log
            | IDENTIFICADOR ASIGNAR conjunto

//Expresión a evaluar y bloque de sentencias a ejecutar
bloque_condicional ::= expresion_log ENTONCES bloque
```

```
                    | ecuaciones ENTONCES bloque

//Seleccion si no si
si_no_si_senten ::= SI_NO SI bloque_condicional si_no_senten
                | SI_NO SI bloque_condicional


//Selección IF
si_senten ::= SI bloque_condicional si_no_si_senten
            | SI bloque_condicional
            | SI bloque_condicional si_no_si_senten



//Iteración WHILE
mientras_senten ::= MIENTRAS bloque_condicional


//Iteración DO WHILE
hacer_mientras_senten ::= HACER bloque MIENTRAS ecuaciones


//Selección SWITCH
seleccionar_senten ::= SELECCIONAR IDENTIFICADOR LLAVEIZ casos LLAVEDE


//Bloque de casos del SWITCH
casos ::= CASO ecuaciones DOSPUNTOS sec_proposiciones ROMPER PCOMA casos
        | CASO ecuaciones DOSPUNTOS sec_proposiciones  casos
        | DEFECTO DOSPUNTOS sec_proposiciones

//Coma y asignación
com_asig ::= COMA asignacion com_asig
           | COMA asignacion

//Iteración FOR
para_senten ::= PARA asignacion com_asig PCOMA ecuaciones PCOMA
                asignacion com_asig bloque
            | PARA asignacion PCOMA ecuaciones PCOMA asignacion com_asig bloque
            | PARA asignacion com_asig PCOMA ecuaciones PCOMA asignacion bloque
            | PARA asignacion PCOMA ecuaciones PCOMA asignacion bloque
            | PARA PAREN_AP asignacion com_asig PCOMA ecuaciones PCOMA asignacion
                com_asig PAREN_CI bloque
            | PARA PAREN_AP asignacion PCOMA ecuaciones PCOMA asignacion com_asig
                PAREN_CI bloque
            | PARA PAREN_AP asignacion com_asig PCOMA ecuaciones
```

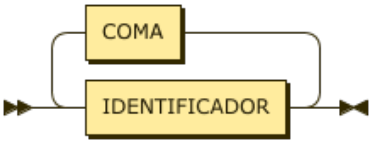
```
                PCOMA asignacion PAREN_CI bloque
            | PARA PAREN_AP asignacion PCOMA ecuaciones PCOMA asignacion
                PAREN_CI bloque

//Función
fun_senten ::= DEF tipo IDENTIFICADOR PAREN_AP lista_ids PAREN_CI bloque


//Procedimiento
proc_senten ::= DEF IDENTIFICADOR PAREN_AP lista_ids PAREN_CI bloque


//Sentencias de función
funcion ::= LLAVEIZ sec_proposiciones PCOMA LLAVEDE

//Operadores aritmeticos
op_arit ::= SUMA
        | MENOS
        | MULT
        | DIV
        | MOD
        | POTENCIA

//Operadores de comparación
op_comp ::= IGUAL
        | DIFERENTE
        | MAYOR
        | MENOR
        | MAY_IGUAL
        | MEN_IGUAL
```

3 Diagramas de sintaxis

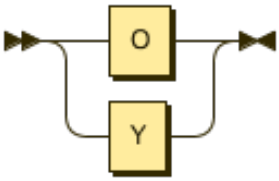
3.1 Algoritmo



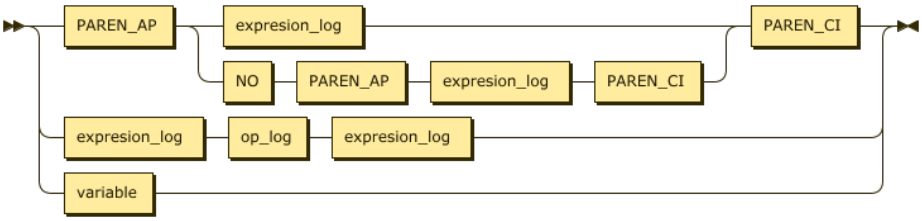
3.2 Lista$_ids$



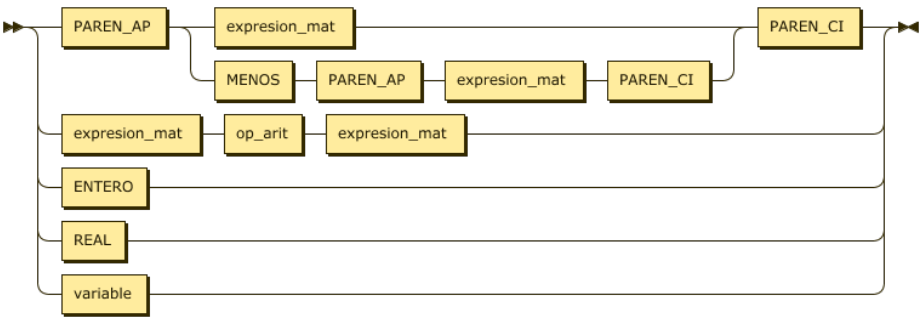
3.3 Op$_log$



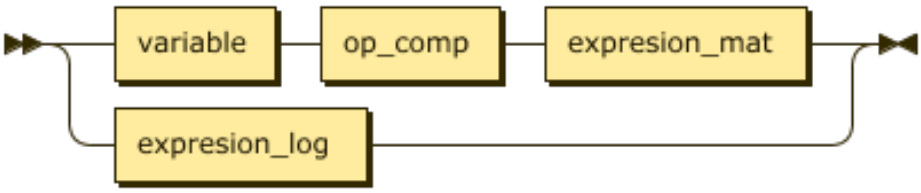
3.4 Expresion log



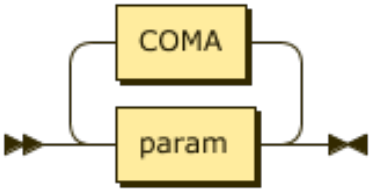
3.5 Expresion mat



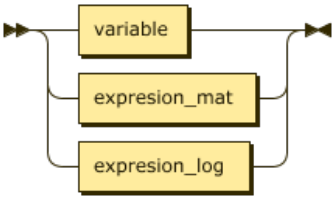
3.6 Ecuaciones



3.7 Parametros

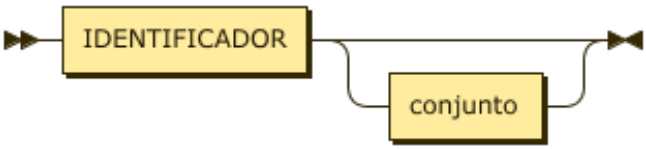


3.8 Parametro



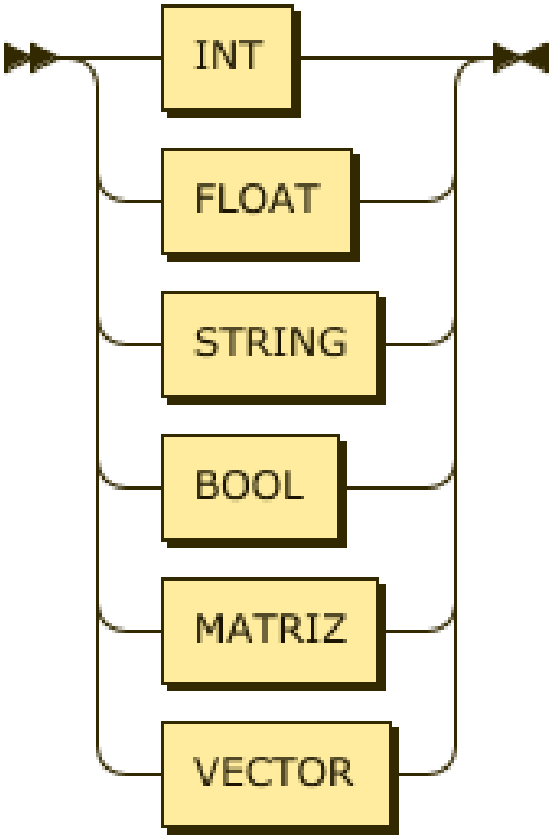
3.9 Definición de conjunto

3.10 Definición de var

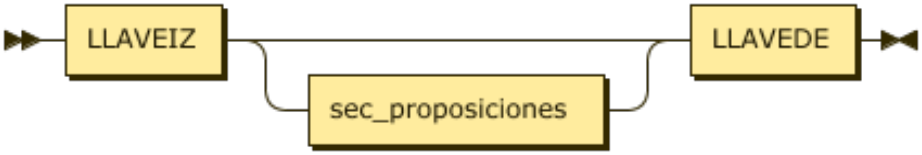


3.11 Lista de parámetros

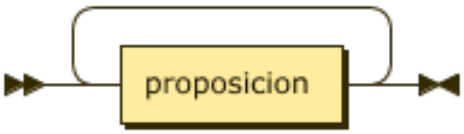
3.12 Identificador de tipo de retorno



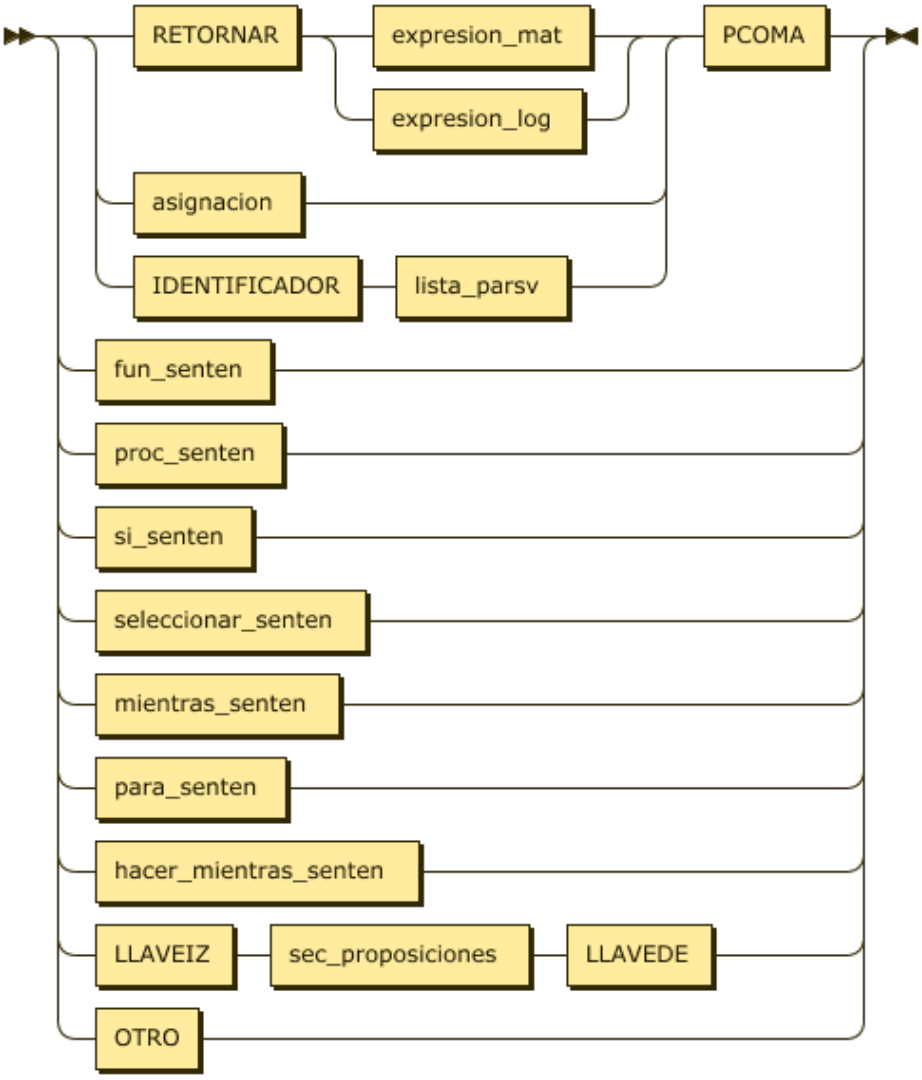
3.13 Bloque



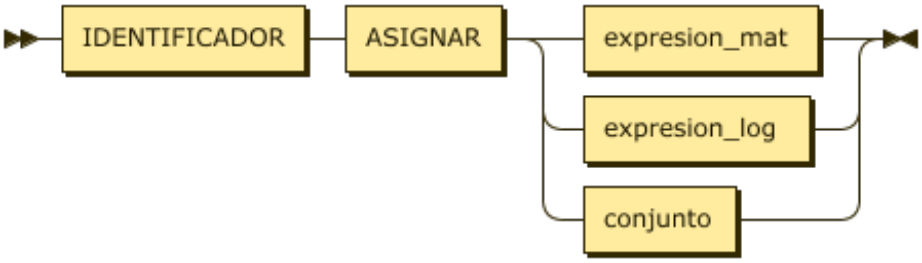
3.14 Secuenciación



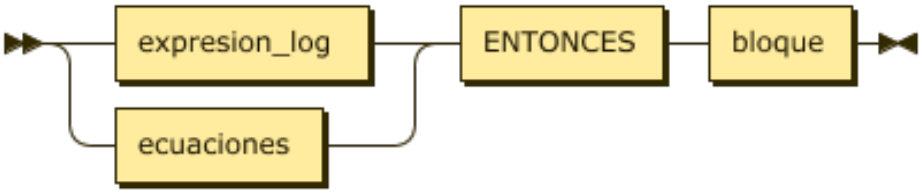
3.15 Conjunto de posibles sentencias



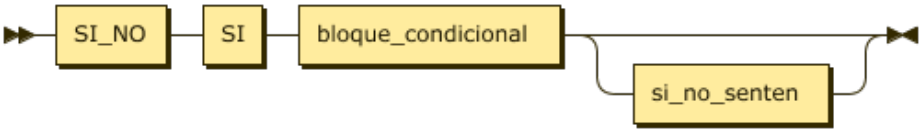
3.16 Modo de asignación



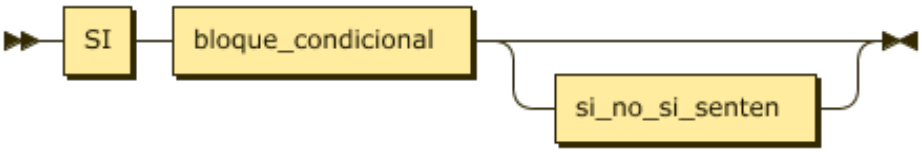
3.17 Expresión a evaluar y bloque de sentencias a ejecutar



3.18 Seleccion si no si

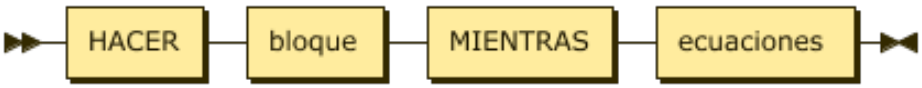


3.19 Selección IF

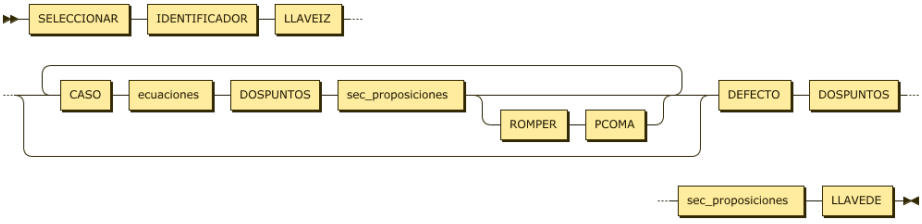


3.20 Iteración WHILE

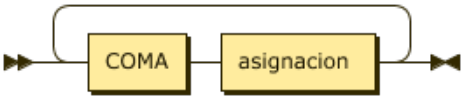
3.21 Iteración DO WHILE



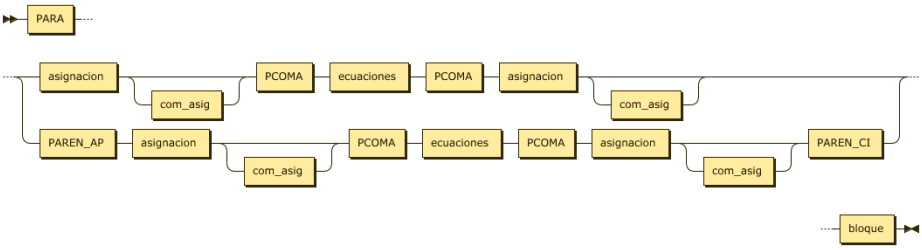
3.22 Selección SWITCH



3.23 Coma y asignación

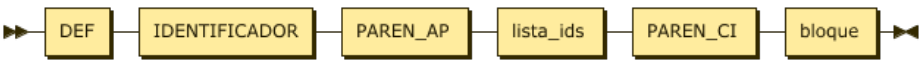


3.24 Iteración FOR



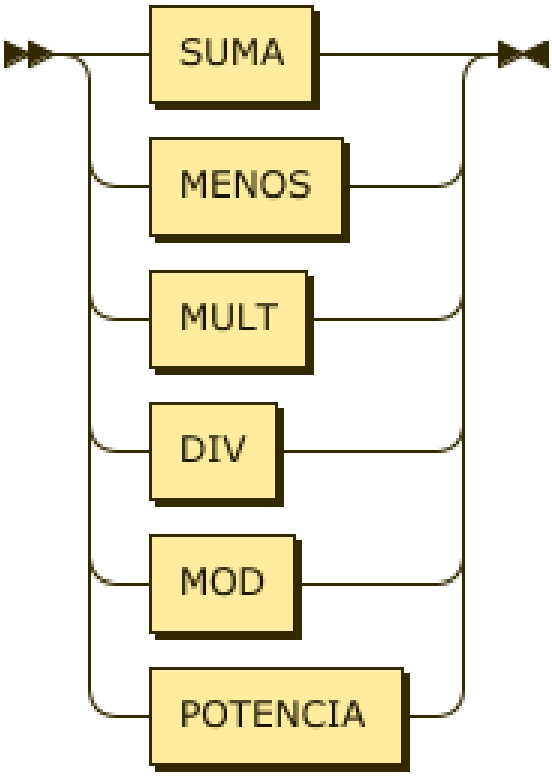
3.25 Función

3.26 Procedimiento

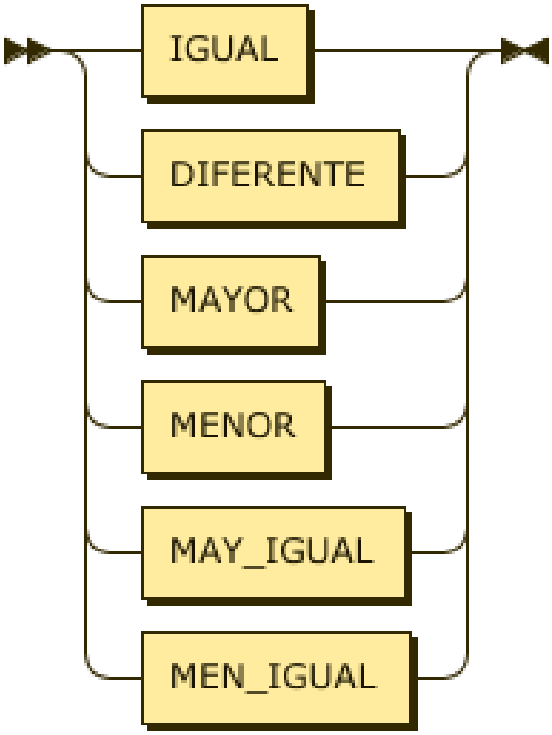


3.27 Sentencias de función

3.28 Operadores aritmeticos



3.29 Operadores de comparación



4 Assembler

4.1 Especificaciones del Usuario

4.1.1 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.

4.1.2 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

4.1.3 Advertencias

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

4.2 Especificaciones Tecnicas

4.2.1 Almacenar

```
public void almacenar() throws IOException{
    instruccionesCompilador();
    String input;
    FileReader f = new FileReader("archivo.txt");
    BufferedReader b1 = new BufferedReader(f);
    String []prueva;
    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 = isntruccionFuente(input);
    prueva = auxIn[0].split(",");
    switch (prueva.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("achivoEtiquetas.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 codigo en lenguaje ensamblador del Z80, para posteriormente cargar el archivo en el "achivoReLoc.txt" el cual es la transformacion del codigo en ensamblador en codigo decimal, para poder ser cargado en memoria en el enlazador cargador.

4.2.2 Relocalizable

```
    public void mostrarRelok() {
        FileReader f = null;
        try {
            String input;
            f = new FileReader("achivoReLoc.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 codigo lo que hace es mostrar nuestro archivo relocalizable antes de mandarlo a enlazar y cargar por medio de la siguiente funcion liker del procesador.

4.2.3 Enlazar

```
public void liker(int partida) throws IOException{
    etiquetas.clear();
    String et;
    FileReader fEt = new FileReader("achivoEtiquetas.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("achivoReLoc.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].replace("&high", ""))) {
            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].substring(2))
                    )
                );
            Memoria[partida+i+1] =
                toCadenaBinaria(
                    Arrays.toString(
                        conversorHEXAtoBIN(auxIn[i].substring(0,2))
                    )
                );
            i++;
        }else{
            Memoria[partida+i]= auxIn[i];
        }
```

```
        }
        b.close();



        File ft = new File("achivoBin.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 "achivoEtiquetas.txt" verifica cuales son las etiquetas del documento para posteriormente leer el archivo "achivoReLoc.txt" el cual contiene lo que se va a cargar a la memoria la cual esta almacena en "achivoBin.txt" por medio de "cargar(int start)".

4.2.4 Cargar

```
    public void cargar(int start) throws IOException{
        String input;
        FileReader f = new FileReader("achivoBin.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 codigo se implementa directamente en el codigo del enlazador(liker) y es el que permite el almacenamiento en memoria del codigo a partir del de los anteriores archivos de textos definidos.

4.2.5 Registros

```
public void incializarRegistros(){

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

4.2.6 Comandos del operador del Z80

Debido a lo extenso que resulta ser, se puede resumir en que cada comando del codigo ensamblador del Z80 se traduce en la memoria del mismo de manera unica.
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 demas instrucciones del Z80 y estas varian segun el numero de bits que su usen en la ejecucion de la operacion.

4.2.7 Instrucciones Compilador y Contenedor Instrucciones

Es importante mencionar estas 2 partes del codigo, puesto que estas nos permiten trasladarnos del OpCode a las instrucciones usadas, asi como de las instrucciones usadas al OpCode de las mismas, lo cual nos favorece la navegacion en el Z80 segun las claves usadas para definir, puestos 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");
```

4.2.8 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 codigo en ensamblador y el cual se va alterando segun la ejecucion paso a paso del codigo 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];
            }
        }
    }
```