Índice general

1 Requisitos **3**

2 Conceptos generales **5**
2.1 Ejecución del programa 5
2.2 Comando `analizar` 5
2.2.1 Sintaxis 5
2.2.2 Parámetros 5
2.2.3 Opciones 6
2.2.4 Ejemplos 6
2.3 Comando `convertir` 6
2.3.1 Sintaxis 6
2.3.2 Parámetros 6
2.3.3 Opciones 6
2.3.4 Ejemplos 7
2.4 Manejo de errores 7

3 Gramática del archivo **9**
3.1 Programación de repertorio y lógica en microprogramado 9
3.1.1 Definición de lógica en microprogramado 9
3.1.2 Definición de repertorio en microprogramado 13
3.2 Programación de repertorio y lógica en cableado 18
3.3 Programación del programa 20
3.3.1 Variables 21
3.3.2 Llamadas a instrucciones 22
3.4 Comentarios 24
3.5 Listado de microoperaciones 26
3.6 Listado de banderas 29

Capítulo 1

Requisitos

Capítulo 2

Conceptos generales

AsiTranspiler es un programa que permite convertir archivos en formatos *.asi* a los ficheros utilizados para la ejecución de SICOME (*.lcb*, *.rep*, *.prog*).

2.1 Ejecución del programa

La aplicación se ejecuta desde la línea de comandos usando Java:

```
java -jar AsiTranspiler.jar [COMANDO] [OPCIONES]
```

Comandos disponibles:

- **analizar** — Analiza un archivo `.asi`
- **convertir** — Convierte un archivo `.asi` a formatos SICOME

También puedes usar:

```
java -jar AsiTranspiler.jar --help
```

para ver la ayuda general.

2.2 Comando `analizar`

Este comando revisa un archivo `.asi` y muestra los errores encontrados sin generar archivos de salida.

2.2.1 Sintaxis

```
java -jar AsiTranspiler.jar analizar OBJETIVO ARCHIVO [opciones]
```

2.2.2 Parámetros

OBJETIVO Indica qué parte del archivo se analizará:

- `logica`
- `repertorio`

- `todo`

ARCHIVO Ruta al archivo `.asi` que se desea analizar.

2.2.3 Opciones

`-i`, `--incluir` Archivo adicional que se utilizará como soporte para lógica o repertorio.

2.2.4 Ejemplos

Analizar solo la lógica:

```
java -jar AsiTranspiler.jar analizar logica ejemplo.asi
```

Analizar todo usando un archivo adicional:

```
java -jar AsiTranspiler.jar analizar todo ejemplo.asi -i base.asi
```

2.3 Comando `convertir`

Este comando convierte un archivo `.asi` a uno o varios archivos compatibles con SICOME.

2.3.1 Sintaxis

```
java -jar AsiTranspiler.jar convertir OBJETIVO ARCHIVO DIRECTORIO [opciones
   ]
```

2.3.2 Parámetros

OBJETIVO Define qué se va a generar:

- `logica` — Genera archivo `.lcb`
- `repertorio` — Genera `.lcb` y `.rep`
- `todo` — Genera `.lcb`, `.rep` y `.prog`

ARCHIVO Archivo `.asi` de entrada.

DIRECTORIO Carpeta donde se crearán los archivos generados.

2.3.3 Opciones

`-n`, `--nombre` Nombre base de los archivos generados. Si no se especifica, se usará el nombre del archivo original.

`-i`, `--incluir` Archivo adicional de soporte para repertorio y lógica.

2.3.4 Ejemplos

Convertir solo la lógica:

```
java -jar AsiTranspiler.jar convertir logica ejemplo.asi ./salida
```

Convertir todo con nombre personalizado:

```
java -jar AsiTranspiler.jar convertir todo ejemplo.asi ./salida -n
    sistema_final
```

Convertir usando un archivo adicional:

```
java -jar AsiTranspiler.jar convertir repertorio ejemplo.asi ./salida -i
    base.asi
```

2.4 Manejo de errores

Durante el análisis o la conversión, el sistema mostrará los errores encontrados en consola. Si existen errores críticos, la conversión no se realizará. Se recomienda ejecutar primero el comando `analizar` antes de convertir.

Capítulo 3

Gramática del archivo

Los ficheros se dividirán en 4 bloques principales que deberán ser definidos en el siguiente orden:

1. **Bloque de lógica de control de bifurcación**: Será necesario únicamente cuando las instrucciones sean definidas en microprogramado.
2. **Bloque de repertorio de instrucciones**: Las instrucciones podrán ser escritas en microprogramado o en cableado.
3. **Bloque de variables**: Se utiliza a la par con el siguiente bloque.
4. **Bloque de programa**: Necesitará de la definición del bloque de variables, bloque de repertorio de instrucciones y bloque de lógica de bifurcación en caso de que sea el programa de tipo microprogramado.

3.1 Programación de repertorio y lógica en microprogramado

Para programar en microprogramado se ha de definir primero un apartado para la lógica de bifurcación y después un apartado para la definición del repertorio de instrucciones.

3.1.1 Definición de lógica en microprogramado

La sintaxis para la definición de la lógica de control de bifurcación se vería como en la figura 3.1.

```
Lógica {
  ...
}
```

Figura 3.1: Sintaxis para la definición del bloque de lógica de control de bifurcación en microprogramado

Para definir una lógica de control simple se debe escribir el nombre de la lógica de control a definir, seguido de una flecha, y a continuación las primitivas de control: *BIF* (Bifuración), *INCR* (Incremento), *RTN* (Carga de rutina) y opcionalmente *DISABLE*, para que no se ejecute las microinstrucciones de ese paso. (Ver figura 3.2)

```
 Lógica {
 nop -> INCR DISABLE // ID 0
 inc ->  INCR //ID 1
 bif ->  BIF  //ID 2
 ret -> RET //ID 3
}
```

Figura 3.2: Ejemplo de definición de lógica de control de bifurcación simple para microprogramado.

Si en cambio se quiere definir una lógica de control compleja, después de la flecha se ha de abrir llaves y escribir los estados de las banderas a la izquierda del doble punto,y a la derecha escribiendo las primitivas de control. (Ver figura 3.3)

```
 ...
bif_disable_if_F ->{ // ID 5
  !F: BIF
   F: BIF DISABLE
}
...
```

Figura 3.3: Ejemplo de definición de lógica de control de bifurcación compleja para microprogramado.

Los identificador numéricos de las lógicas de bifurcación empezaran por 0 y seguirán según el orden de aparición.

Error: LOGICA_BIFURCACION_MISMO_NOMBRE

No puede haber dos lógicas de bifurcación con el mismo nombre.

Error: NUMERO_LOGICA_BIFURCACION_SUPERADO

No puede haber definidas más de 16 lógicas de bifurcación por estar guardado en un registro de 4 bits.

Error: BANDERA_NO_RECONOCIDA

La bandera escrita no se corresponde con ninguna de las existentes, un listado de todas las banderas se encuentran en el apartado 3.6.

Error: LOGICA_CONTROL_NO_EXHAUSTIVA

La combinación de banderas definidas está mal formada

- Comprueba que no haya dos banderas definidas en la misma condición.
- Comprueba que no hayas dos condiciones iguales definidas.
- Comprueba que no haya condiciones sin definir.

Error: BANDERA_INVALIDA

No esta permitido el uso de la bandera *Ovf* exclusiva de cableado en microprogramado.

```
Lógica{
     nop -> INCR DISABLE //0
     inc ->  INCR //1
     bif ->  BIF  //2
     ret -> RTN //3
     jmp_if_Z -> { //4
        !F:INCR
         F:BIF
     }
     bif_disable_if_F ->{ //5
        !F:BIF
        F: BIF DISABLE
     }
     jmp_if_not_Zb -> { //6
        !Zb:BIF
         Zb:INCR
     }
     bif_and_disable_if_not_Zb -> { //7
        !Zb:BIF DISABLE
        Zb:BIF
     }
     jmp_and_disable_if_Zac -> { //8
        !Zac: INCR
        Zac: BIF DISABLE
     }
     bifurcate_and_disable_if_not_As -> { //9
        !As: BIF DISABLE
         As: BIF
     }
     continue_and_disable_if_As -> { //10
        !As: INCR
         As: INCR DISABLE
     }

     jmp_and_disable_if_Zsc -> { //11
        !Zsc:BIF DISABLE
        Zsc:INCR
     }

     continue_and_disable_if_Zb -> {  //12
        !Zb:INCR
        Zb: INCR DISABLE
     }
 }
```

Figura 3.4: Ejemplo de conversión bloque control de bifurcación a archivo .lcb: entrada

```
B3 B2 B1 B0 F Zb Za Zac Zsc X Qn Qn1 As Qs Bs N I B R E
0  0  0  0  X X  X  X   X   X X  X   X  X  X  X 1 0 0 0
0  0  0  1  X X  X  X   X   X X  X   X  X  X  X 1 0 0 1
0  0  1  0  X X  X  X   X   X X  X   X  X  X  X 0 1 0 1
0  0  1  1  X X  X  X   X   X X  X   X  X  X  X 0 0 1 1
0  1  0  0  0 X  X  X   X   X X  X   X  X  X  X 1 0 0 1
0  1  0  0  1 X  X  X   X   X X  X   X  X  X  X 0 1 0 1
0  1  0  1  0 X  X  X   X   X X  X   X  X  X  X 0 1 0 1
0  1  0  1  1 X  X  X   X   X X  X   X  X  X  X 0 1 0 0
0  1  1  0  X 0  X  X   X   X X  X   X  X  X  X 0 1 0 1
0  1  1  0  X 1  X  X   X   X X  X   X  X  X  X 1 0 0 1
0  1  1  1  X 0  X  X   X   X X  X   X  X  X  X 0 1 0 0
0  1  1  1  X 1  X  X   X   X X  X   X  X  X  X 0 1 0 1
1  0  0  0  X X  X  0   X   X X  X   X  X  X  X 1 0 0 1
1  0  0  0  X X  X  1   X   X X  X   X  X  X  X 0 1 0 0
1  0  0  1  X X  X  X   X   X X  X   0  X  X  X 0 1 0 0
1  0  0  1  X X  X  X   X   X X  X   1  X  X  X 0 1 0 1
1  0  1  0  X X  X  X   X   X X  X   0  X  X  X 1 0 0 1
1  0  1  0  X X  X  X   X   X X  X   1  X  X  X 1 0 0 0
1  0  1  1  X X  X  X   0   X X  X   X  X  X  X 0 1 0 0
1  0  1  1  X X  X  X   1   X X  X   X  X  X  X 1 0 0 1
1  1  0  0  X 0  X  X   X   X X  X   X  X  X  X 1 0 0 1
1  1  0  0  X 1  X  X   X   X X  X   X  X  X  X 1 0 0 0
```

Figura 3.5: Ejemplo de conversión bloque control de bifurcación a archivo .lcb: salida

3.1.2 Definición de repertorio en microprogramado

La sintaxis para la definición de la lógica de control de bifurcación se vería como en la figura 3.6.

```
@Micro
Instrucciones {
  ...
}
```

Figura 3.6: Sintaxis para la definición de repertorio para microinstrucciones

Para definir una instrucción se deberá escribir el nombre de esta, comenzando en minúscula, y a continuación, entre parentesis, el tipo de argumentos que recibirá, que se encuentra explicado en el apartado 3.3.

Para la definición de los pasos en instrucciones se escribirá entre | | el control de bifurcación que se quiere asociar a ese paso, y después las microinstrucciones separadas por espacios. Los pasos han de terminar en ";".(Ver figura 3.7)

```
pop(Value){
    |inc| SP->MAR;
    |inc| M->QR SP+1->SP;
    |inc| GPR[AD]->MAR;
    |bif(0)| QR->M;
}
```

Figura 3.7: Ejemplo de definición de instrucción en microprogramado.

Antes de definir ninguná de las instrucciones se deberá definir el ciclo de busqueda (fetch), como se ve en la figura 3.8.

```
Fetch {
    |inc| PC->MAR;
    |inc| M->GPR PC+1->PC;
    |rtn| GPR[OP]->OPR GPR[AD]->MAR;
}
```

Figura 3.8: Ejemplo de definición de ciclo de búsqueda.

En caso de tener el control de ejecución una primitiva *BIF* en algunas de sus combinaciones, esta deberá recibir un argumento en su uso. Debido a la naturaleza de microprogramado, se podrá bifurcar a cualquier otra instrucción, esto se hará escribiendo: "nombre_instruccion" "∼" "desplazamiento", siendo desplazamiento el paso a bifuración, contando la primera instrucción como la 0. En caso de querer terminar la instrucción y volver al ciclo de busqueda se hará de escribir "0" como argumento.

```
estados{
                //Los estados están definidos en el ejemplo anterior
}
@Micro
Instrucciones {
    Fetch {
        |inc| PC->MAR;
        |inc| M->GPR PC+1->PC;
        |ret| GPR[OP]->OPR GPR[AD]->MAR;
    }
    halt(){
    }
    isz(Value){
          |inc| M->GPR;
          |inc| GPR+1->GPR;
          |inc| GPR->M;
          |jmp_if_not_Zb(0)| ;
          |bif(0)| PC+1->PC;
    }
    isz_r(Value){
          |inc| M->GPR;
          |inc| GPR+1->GPR;
          |inc| GPR->M;
          |bif_and_disable_if_not_Zb(0)| PC+1->PC;
    }
    push(Value){
        |inc| M->GPR SP-1->SP;
        |inc| SP->MAR;
        |bif(0)| GPR->M;
    }
    pop(Value){
        |inc| SP->MAR;
        |inc| M->QR SP+1->SP;
        |inc| GPR[AD]->MAR;
        |bif(0)| QR->M;
    }
    push_i(Value){
        |inc| M->GPR SP-1->SP;
        |inc| GPR[AD]->MAR;
        |inc| M->GPR;
        |inc| SP->MAR;
        |bif(0)| GPR->M;
    }
    pop_i(Value){
        |inc| SP->MAR;
        |inc| M->QR SP+1->SP;
        |inc| GPR[AD]->MAR;
        |inc| M->GPR;
        |inc| GPR[AD]->MAR;
        |bif(0)| QR->M;
    }
    jazpn(){
        |jmp_and_disable_if_Zac(0)| PC+1->PC;
        |bifurcate_and_disable_if_not_As(0)| PC+1->PC;
    }
}
```

Figura 3.9: Ejemplo de conversión bloque de repertorio de instrucciones a archivo .rep: entrada

```
$
CB 4000100
CB 0201100
CB B000300
$
halt false 0
isz true 1100 4100 1000100 600 200200
isz_r true 1100 4100 1000100 200700
push true 801100 C000100 1000200
pop true C000100 670100 8000100 2000200
push_i true 801100 8000100 1100 C000100 1000200
jazpn false 200800 200900
```

Figura 3.10: Ejemplo de conversión bloque de repertorio de instrucciones a archivo .rep: salida

Error: INSTRUCCION_MISMO_NOMBRE

Ya existe otra instrucción con el mismo nombre.

Error: MICROINSTRUCCION_NO_RECONOCIDA

La microinstrucción escrita no se corresponde con ninguna de las microinstrucciones definidas (Ver apartado 3.5).

Error: MICROINSTRUCCION_INVALIDA

La microinstrucción utilizada es inválida en el contexto en el que se utiliza.

- Comprueba que no haya dos microinstrucciones del mismo tipo en el mismo paso.
- Comprueba que no haya microinstrucciones de tipo cable (*SR+1->SR* y *LOAD_SR*).
- Comprueba que no haya instrucciones con argumentos a la vez que una lógica de bifurcación que requiera argumento.

Error: ARGUMENTO_USO_LOGICA_BIFURCACION_INVALIDO

El argumento de la lógica de bifurcación es invalido.

- Comprueba que las lógicas de bifurcacion que necesiten de argumento, lo tengan.
- Comprueba que las lógicas de bifurcaciónes que tengan como argumento, una instrucción y un offset, no tenga un offset mayor que el nº de pasos de la instrucción.

Error: INSTRUCCION_NO_DEFINIDA

Comprueba que las lógicas de bifurcaciónes que tengan como argumento, una instrucción y un offset, se refieran a instrucciones definidas.

Error: MICROINSTRUCCION_CON_ARGUMENTO_INVALIDO

Comprueba que las microinstrucciones que necesiten de argumento, lo reciban.

- Comprueba que el argumento pasado a LOAD SC no supere el límite de 8 bits.

Error: MICROINSTRUCCION_CON_ARGUMENTO_INNECESARIO

La microinstrucción tiene un argumento no necesario.

Error: TAMANYO_ROM_SUPERADO

El uso de la ROM por parte de los pasos de las instrucciones se ha superado (256 lineas).

Error: NUMERO_INSTRUCCIONES_SUPERADO

El número de instrucciones máximo se ha superado de 32, por el tamaño dedicado a instrucciones en la celdas de memoria (5 bits).

3.2 Programación de repertorio y lógica en cableado

Para la programación del repertorio solo hará falta un bloque donde se defina las instrucciones, ya que la lógica de control de bifurcación se definirá a la misma vez que las microinstrucciones de los pasos.

El identificador de las instrucciones (q0, q1, etc..) corresponderá con el orden en el que están definidas.

La sintaxis para la definición de la lógica de control de bifurcación se vería como en la figura 3.11.

```
@Cable
Instrucciones{
  ...
}
```

Figura 3.11: Sintaxis para la definición del bloque de repertorio de instrucciones para cableado.

Un paso simple se definirá escribiendo entre "| |" *LOAD_SR* o *SR+1->SR*. Y posteriormente se escribirán las microinstrucciones a ejecutar separadas por espacio. Debe ser finalizado con ";". (Ver figura 3.12)

```
//Carga del registro ACC con el contenido de la dirección de memoria
   indicada.
  lda(Var){
    |SR+1->SR| M->GPR 0->ACC;
    |LOAD_SR (0)|  GPR+ACC->ACC;
  }
```

Figura 3.12: Ejemplo de instrucción con pasos simples en cableado.

Para la definición de un paso complejo se abrirá una llave, y en su interior se definirán las opciones que se ejecutaran dependiendo del estado de las flags definidas. (Ver figura 3.13)

```
paraq(Var){
  ...
  /*ONES*/
  |SR+1->SR| LOAD SC(16) QR->GPR; // Paso 5
  |SR+1->SR| SC-1->SC ROL_FAQ ; //Paso 6
  { // Paso 7
    F Zsc :|SR+1->SR| GPR+1->GPR;
    !F Zsc : |SR+1->SR|;
    F !Zsc : |LOAD_SR (6)| GPR+1->GPR;
    !F !Zsc :|LOAD_SR (6)| ;
  }
  ...
}
```

Figura 3.13: Ejemplo de instrucción con pasos complejos en cableado.

```
@Cable
Instrucciones {
    instruccion1(){
     |SR+1->SR| PC+1->PC;
     |LOAD_SR(0)| GPR->PC;
    }
}
```

Figura 3.14: Ejemplo de conversión bloque de repertorio de instrucciones en cableado a archivo .rep: entrada

```
$
M->GPR:t1
GPR(OP)->OPR:t2
PC->MAR:t0
GPR(AD)->MAR:t2
PC+1->PC:t1 + t3·q1
GPR->PC:t4·q1
$
SR+1->SR:t0 + t1 + t2 + t3·q1
LOAD SR:t4·q1-0
LOAD SC:t3·q1-8
$
```

Figura 3.15: Ejemplo de conversión bloque de repertorio de instrucciones en cableado a archivo .rep: salida

Error: INSTRUCCION_MISMO_NOMBRE

Ya existe otra instrucción con el mismo nombre.

Error: MICROINSTRUCCION_NO_RECONOCIDA

La microinstrucción escrita no se corresponde con ninguna de las microinstrucciones definidas (Ver apartado 3.5).

Error: MICROINSTRUCCION_INVALIDA

La microinstrucción utilizada es inválida en el contexto en el que se utiliza.

- Comprueba que no haya dos microinstrucciones del mismo tipo en el mismo paso.
- Comprueba que no haya instrucciones de tipo cable (*SR+1->SR* y *LOAD_SR*) en la parte derecha de un paso.
- Comprueba que no haya instrucciones que no sean de tipo no cable en la parte izquierda del paso.

Error: BANDERA_NO_RECONOCIDA

La bandera escrita no se corresponde con ninguna de las existentes, un listado de todas las banderas se encuentran en el apartado 3.6.

Error: MICROINSTRUCCION_CON_ARGUMENTO_INVALIDO

Comprueba que las microinstrucciones que necesiten de argumento, lo reciban.

- Comprueba que el argumento pasado a LOAD SC no supere el límite de 8 bits.
- Comprueba que el argumento pasado a LOAD SR no supere el límite de pasos de la instrucción en la que está.

Error: MICROINSTRUCCION_CON_ARGUMENTO_INNECESARIO

La microinstrucción tiene un argumento no necesario.

Error: NUMERO_INSTRUCCIONES_SUPERADO

El número de instrucciones máximo se ha superado de 32, por el tamaño dedicado a instrucciones en la celdas de memoria (5 bits).

Error: PASO_COMPLEJO_NO_EXHAUSTIVO

La combinación de banderas definidas está mal formada:

- Comprueba que no haya dos banderas definidas en la misma condición.
- Comprueba que no hayas dos condiciones iguales definidas.
- Comprueba que no haya condiciones sin definir.

3.3 Programación del programa

Para la definición de los espacios de memoria para un programa se reservan los primeros para las variables a definir dejando los siguientes espacios para la llamadas a las

instrucciones del programa.

3.3.1 Variables

La sintaxis para el bloque de definición de variables se vería como en la figura 3.16.

```
variables{
  ...
}
```

Figura 3.16: Sintaxis para la definición del bloque de variables.

El lugar en la memoria de las variables se corresponderán con su orden de definición. El valor de las variables se podrán especificar de forma:

- **Representación binaria**: Comienza por "0b" y puede contener "." en medio para mejorar la legibilidad. Ejemplo: " 0b100.1101.0010 "
- **Representación hexadecimal**: Comienza por "0x" . Ejemplo "0x4D2"
- **Representación decimal signo-magnitud**: Comienza por "0s" Ejemplo: "0s-1234"
- **Representación decimal complemento a dos**: Comienza por "0c". Ejemplo: "0c-43"

```
valor1 = 0c14;
valor2 = 0x2E3;
```

Figura 3.17: Ejemplo de definición de variables de tipo básico.

Además de poder definir variables simples como en la figura 3.17, se puede definir espacios de memorias continuos con el uso de vectores, asignándoles valores predeterminados o valores específicos en cada posición, como se ve en la figura.

```
data (3) = {0s1,0s2,0s3}; //En memoria se asignara 1, 2 ,3
cumulative (3) = {0s0} //En memoria se asignará 0 , 0, 0
```

Figura 3.18: ejemplo de definición de variables de tipo vector.

Error: VARIABLE_MISMO_NOMBRE

Ya existe otra variable con el mismo nombre.

Error: TAMANYO_VECTOR_INVALIDO

El tamaño del vector definido es inválido, debe ser mayor igual o superior a 2.

Error: INICIALIZACION_VECTOR_INVALIDA

El numero de valores inicializados no coinciden con el tamaño del vector, o no es 1 para indicar valor predeterminado.

Error: VALOR_VARIABLE_NO_VALIDO

El valor de la variable supera el tamaño máximo de la celda de memoria (16bits).

3.3.2 Llamadas a instrucciones

La sintaxis para el bloque de llamadas a instrucciones se vería como en la figura 3.19.

```
Programa{
  ...
}
```

Figura 3.19: Sintaxis de bloque de llamada a instrucciones.

El comportamiento relacionado con la llamada a instrucciones dependerá de la definición de los argumentos que toman. Podrán tener como posibles argumentos:

- **Valor (Val)**: Podrá recibir un números literal. El valor máximo de los números serán menores que los posibles como guardados en memoria.
- **Variable (Var)**: Podrán recibir una dirección de memoria asociada a una variable.
- **Dirección (Dir)**: Podrán recibir una dirección de memoria asociado a un punto de ejecución del programa.
- **Ninguno**: No recibirá ningún tipo de argumento.

Error: INSTRUCCION_NO_DEFINIDA

La instrucción no se encuentra definida en el repertorio.

Error: ARGUMENTO_DE_TIPO_VALOR_NO_ENCONTRADO

La llamada a la instrucción requiere de tipo valor pero ha recibido un argumento invalido.

Error: ARGUMENTO_DE_TIPO_VARIABLE_NO_ENCONTRADO

La llamada a la instrucción requiere de tipo variable pero ha recibido un argumento invalido.

Error: ARGUMENTO_DE_TIPO_DIRECCION_NO_ENCONTRADO

La llamada a la instrucción requiere de tipo dirección pero ha recibido un argumento invalido.

Error: ARGUMENTO_INSTRUCCION_INNECESARIO

La llamada a la instrucción no requiere de argumentos pero ha recibido un argumento.

Error: VALOR_ARGUMENTO_LITERAL_NO_VALIDO

El valor del argumento literal pasado supera el tamaño máximo de la celda de memoria (11bits).

Para pasar como argumento una espacio de una variable de tipo vector se utilizará la notación "nombrevariable(índice)", el valor del índice podrá ser desde 0 hasta tamaño del vector menos uno.(Ver figura 3.20).

Para hacer referencia a un punto en la ejecución del programa se hará uso de la palabra "MARK" junto con la nombre por la cual no referiremos a la etiqueta (Ver figura 3.20)

```
MARK inicio
sumar data(0);
guardar cumulative(0);
sumar data(1);
guardar cumulative(1);
sumar data(2);
guardar cumulative(2);
saltarSiNoZ inicio;
```

Figura 3.20: Ejemplo de uso de etiquetas.

Error: ETIQUETA_MISMO_NOMBRE

Ya existe otra etiqueta con el mismo nombre.

```
@Cable
Instrucciones {
    jmp( Dir ){

    }
}
variables{
    a(9) = {0};
}
programa{
    MARK inicio;
        jmp inicio;
    MARK medio;
        jmp medio;
    MARK final;
        jmp final;
        halt;
}
```

Figura 3.21: Ejemplo de conversión bloque de programa a archivo .prog: entrada

```
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
@
9
@
jmp 9
jmp A
jmp B
halt
@
```

Figura 3.22: Ejemplo de conversión bloque de programa a archivo .prog: salida

Error: VARIABLE_NO_DEFINIDA

La variable referenciada no se encuentra definida.

Error: ETIQUETA_NO_DEFINIDA

La etiqueta referenciada no se encuentra definida.

Error: INDICE_ARGUMENTO_VECTOR_INVALIDO

El índice del acceso al vector es inválido (Debe ser entre 0 y tamaño_vector-1).

Error: ESPACIO_MEMORIA_SUPERADO

La definición del programa (variables + llamadas a instrucciones) supera el máximo de espacio de memoria de la computadora (2048 celdas).

Error: FALTA_BLOQUE_NECESARIO

Falta bloque/s necesarios para el objetivo definido.

3.4 Comentarios

En todo el fichero .asi se puede hacer uso de comentarios en linea(Ver figura 3.23 como multilinea (Ver figura 3.24).

```
//Comentarios en linea
```

Figura 3.23: Ejemplo de comentario en linea.

```
/*Hola
Soy
Un
Comentario
multilinea
*/
```

Figura 3.24: Ejemplo de comentario en linea.

3.5 Listado de microoperaciones

Nombre	Nombre en SICOME
MAR	
PC->MAR	PC->MAR
GPR[AD]->MAR	GPR(AD)->MAR
SP->MAR	SP->MAR
OPR	
GPR->M	GPR->M
QR->M	QR->M
GPR[OP]->OPR	GPR(OP)->OPR
PC, SP y SC	
PC+1->PC	PC+1->PC
GPR->PC	GPR->PC
SP+1->SP	SP+1->SP
SP-1->SP	SP-1->SP
LOAD_SC	LOAD SC
SC-1->SC	SC-1->SC
GPR	
M->GPR	M->GPR
ACC->GPR	ACC->GPR
PC->GPR	PC->GPR
GPR+1->GPR	GPR+1->GPR
QR->GPR	QR->GPR
!GPR->GPR	GPR'->GPR
!GPR+1->GPR	GPR'+1->GPR
ALU	
0->ACC	0->ACC
!ACC->ACC	!ACC->ACC
ACC+1->ACC	ACC+1->ACC
!ACC+1->ACC	ACC'+1->ACC

ACC+GPR->ACC	GPR+ACC->GPR
ROL_F_ACC	ROL_F_ACC
ROR_F_ACC	ROR_F_ACC
0->QR	0->QR
1->OVF	1->OVF
0->OVF	0->OVF
0->Qn+1	0->Qn+1
!QR+1->QR	QR'+1->QR
GPR->QR	GPR->QR
M->QR	M->QR
1->Qn	1->Qn
X->Qs	X->Qs
ASHR_ACC_QR	ASHR_ACC_QR
ROL_F_ACC_QR	ROL_F_ACC_QR
ROR_F_ACC_QR	ROR_F-ACC_QR
SHL_F_ACC_QR	SHL_F_A_Q
SHR_F_ACC_QR	SHR_F_ACC_QR
0->F	0->F
!F->F	F'->F
!GPR+1+ACC->ACC	GPR'+1+ACC->ACC
!ACCQR+1->ACCQR	ACCQR'+1->ACCQR
0->N	0->N
1->N	1->N
!A+1->A	!A+1->A
!As->As	As'->As
0->As	0->As
As->Qs	As->Qs
Qs@Bs->As	Qs@Bs->As
Qs@Bs->Qs	Qs@Bs->Qs
Q+1->Q	Q'+1->Q
0->A	0->A

A+B->EA	A+B->EA
A+!B+1->EA	A+!B+1->EA
A+!B+1->A	A+B'+1->A
E->OVF	E->OVF

3.6 Listado de banderas

- **Zb**: Es 1, si todos los bits de BR/GPR son 0 (Los 16 bits a 0, bit de signo + bits de magnitud todos a 0)
- **Za**: Es 1 si todos los bits de la magnitud del ACC son 0 (es decir, todos los bits del ACC a 0 menos el bit de signo de Acc)
- **Zac**: Es 1 si todos los bits de ACC son 0 (magnitud y signo).
- **Zsc**: Es 1 si el contador del controlador ha llegado a 0.
- **Ovf**: Indica el Overflow, es un bit que debemos activar/desactivar nosotros para indicar el desbordamiento de una operación. **No se puede utilizar en microprogramado.**
- **N**: Es un flag extra que debemos activar/desactivar nosotros según la funcionalidad que deseemos darle. Se puede utilizar dentro de la tabla de control de bifurcación.
- **Qn**: Bit menos significativo de QR (Bit 0 de QR).
- **Qn+**: Bit a la izquierda del menos significativo de QR (Bit 1 de QR).
- **As**: Bit de Signo de Acc (bit 15 de Acc).
- **Qs**: Bit de Signo de QR (bit 15 de QR).
- **Bs**: Bit de Signo de BR/GPR (bit 15 de BR/GPR).
- **X**: Es el cálculo automático de As XOR Bs. Se modifica en cada ciclo según el contenido de Acc y de BR/GPR.
- **F**:Es el valor del acarreo que resulto de la última suma realizada por el sumador.