Generación de código: funciones avanzadas de librería de generación de código

Índice

- Funciones "avanzadas" de la librería de generación de código
  - La cuestión de la identificación correcta de las etiquetas utilizadas
  - Gestión de los errores en tiempo de ejecución
    - `void escribir_fin(FILE* fpasm)`
  - Generación de código para las sentencias condicionales
    - Previas al inicio de la estructura
      - `void ifthenelse_inicio(FILE * fpasm, int exp_es_variable, int etiqueta)`
      - `void if_then_inicio(FILE * fpasm, int exp_es_variable, int etiqueta)`
    - En un punto intermedio de la estructura
      - `void ifthenelse_fin_then( FILE * fpasm, int etiqueta)`
    - Al final de la estructura
      - `void ifthen_fin(FILE * fpasm, int etiqueta)`
      - `void ifthenelse_fin( FILE * fpasm, int etiqueta)`
  - Generación de código para las sentencias iterativas
    - Previas al inicio de la estructura
      - `void while_inicio(FILE * fpasm, int etiqueta)`
    - En un punto intermedio de la estructura
      - `void while_exp_pila (FILE * fpasm, int exp_es_variable, int etiqueta)`
    - Al final de la estructura
      - `void while_fin( FILE * fpasm, int etiqueta)`
  - Generación de código para indexación de vectores
    - `void escribir_elemento_vector(FILE * fpasm,char * nombre_vector, int tam_max, int exp_es_direccion)`

Índice

- Funciones "avanzadas" de la librería de generación de código
  - Generación de código para declaración de funciones
    - Para declarar
      - `void declararFuncion(FILE * fd_asm, char * nombre_funcion, int num_var_loc)`
    - Terminar una función
      - `void retornarFuncion(FILE * fd_asm, int es_variable)`
    - Funciones adicionales
      - Gestionar variables locales y parámetros
        - `void escribirParametro(FILE* fpasm, int pos_parametro, int num_total_parametros)`
        - `void escribirVariableLocal(FILE* fpasm, int posicion_variable_local)`
  - Generación para llamada a funciones
    - Ajuste de las expresiones que serán utilizadas como argumento
      - `void operandoEnPilaAArgumento(FILE * fd_asm, int es_variable)`
    - Invocación de la función, que realiza la llamada en sí
      - `void llamarFuncion(FILE * fd_asm, char * nombre_funcion, int num_argumentos)`

Generación de código: gestión de las etiquetas

- A lo largo de un programa NASM hay múltiples etiquetas necesarias para implementar las estructuras de control de flujo del programa.

- Cuando el compilador genere código deberá articular algún mecanismo para poder distinguir unas etiquetas de otras.
  - A lo largo del curso se sugiere el uso de un contador que se incremente cada vez que se utilice una etiqueta y que aparezca explícitamente en ella.
  - Así, por ejemplo, una etiqueta de fin de if podría ser `fin_if_99:`, siendo en este caso 99 el correspondiente contador.
  - Para la siguiente etiqueta que se necesite (por ejemplo una etiqueta para una rama then) se incrementará el contador de etiquetas y se generará, por ejemplo, la etiqueta `then_100:`.

- Sólo cuando se genera código de manera independiente al compilador hay que tener en cuenta estas reflexiones. Las herramientas utilizadas para generar el compilador proporcionan otro mecanismo para hacer lo que se describe en estas páginas.

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- La generación de código de un ifthenelse o de un while implica el uso de un conjunto de etiquetas que está relacionado (salto al final del bucle cuando se cumple la condición de salida, salto al inicio en otro caso)
- Como los programas pueden tener múltiples whiles o ifthenelses, es necesario llevar un contador de cuántas etiquetas se han usado para usarlo como parte de la propia etiqueta para distinguir unas de otras.
- En los casos en los que las estructuras de control de flujo (bloques estructurales) se anidan se da el hecho de que a medida que se profundiza en el anidamiento hay que ir conservando las etiquetas de los bloques de los que aún no se ha salido porque cuando se vaya saliendo de ellos habrá que recuperarlas para realizar esa gestión.
- Si se utiliza esta librería de manera independiente al compilador, hay que programar este control de etiquetas con estructuras de datos propias del programa principal. Una aproximación sencilla es una pila, ya que el orden de acceso a sus elementos coincide con el orden de acceso a las sucesivas etiquetas.

Generación de código: gestión de las etiquetas

Se propone el siguiente esquema

- Utilizar un contador global de cuántas etiquetas se han usado (getiquetas) inicialmente a -1
- Se utilizará una variable global con la etiqueta actual que todos podrán usar o consultar para saber en qué parte de las estructuras anidadas se encuentra el código (etiqueta).
- Será necesaria una pila para poder guardar en el correcto orden las etiquetas adecuadas al nivel de anidación y para simular el proceso de recuperación de etiquetas a medida que se van cerrando y saliendo de estructuras más internas (etiquetas)

Generación de código: gestión de las etiquetas

A continuación se muestra qué tratamiento hay que realizar en las siguientes circunstancias

- Al inicio del programa para dar valor inicial adecuado a los diferentes elementos del esquema

```
getiqueta = -1;
cima_etiquetas = -1
```

- Al inicio de una estructura de control (if-then-else, if-then o while)

```
getiqueta++;
cima_etiquetas++;
etiquetas[cima_etiquetas]=getiqueta;
etiqueta = getiqueta;
```

- En un punto intermedio cuando se debe saber qué etiqueta "toca"

```
etiqueta = etiquetas[cima_etiquetas];
```

- Y cuando se termina un bucle (la primera instrucción recupera la etiqueta para su uso y la segunda debe realizarse una vez terminadas todas las tareas de la estructura)

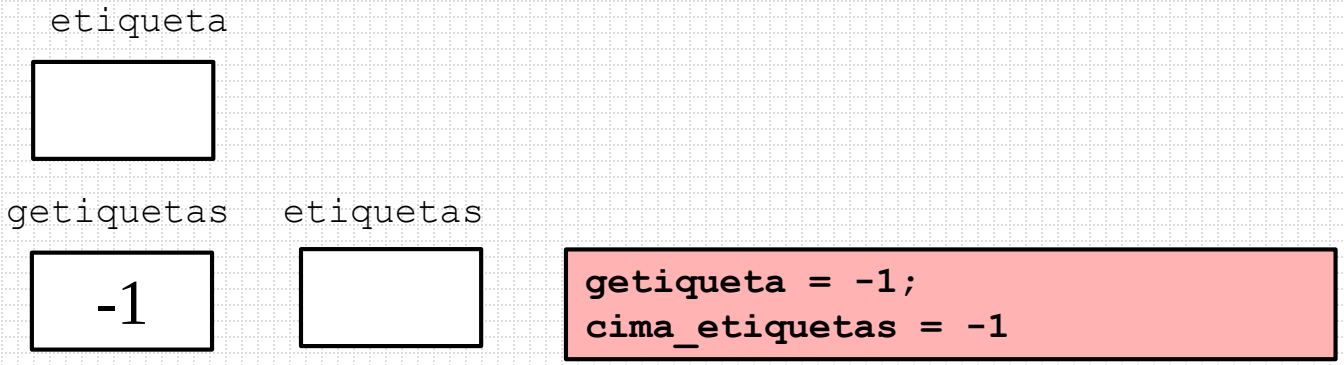
```
etiqueta = etiquetas[cima_etiquetas];

cima_etiquetas--;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Inicialmente

```
main{
 /*  ··· */
 if ()
 {
         if
         {
                  /*··· */
         };}
 else
 {
         if
         {
                  /*··· */
         }
         else
         {
                  /*··· */
         }
  }
/*···*/}
```

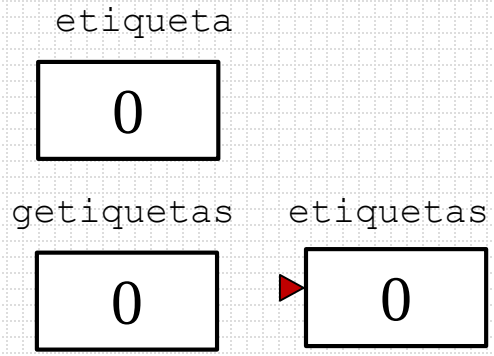


Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Antes de la estructura

```
main{
 /*  ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
  }
/*···*/}
```



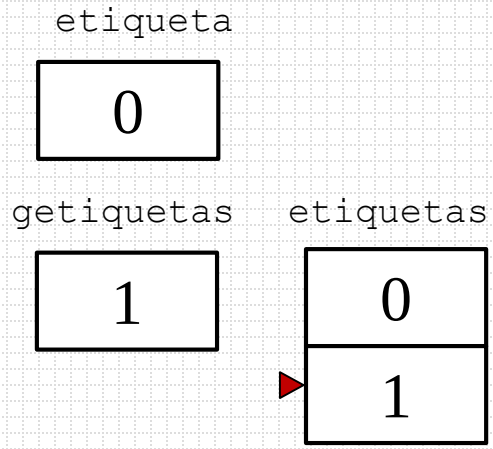

```
getiqueta++;
cima_etiquetas++;
etiquetas[cima_etiquetas]
        =getiqueta;
etiqueta = getiqueta;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Antes de la estructura

```
main{
 /*  ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
  }
/*···*/}
```



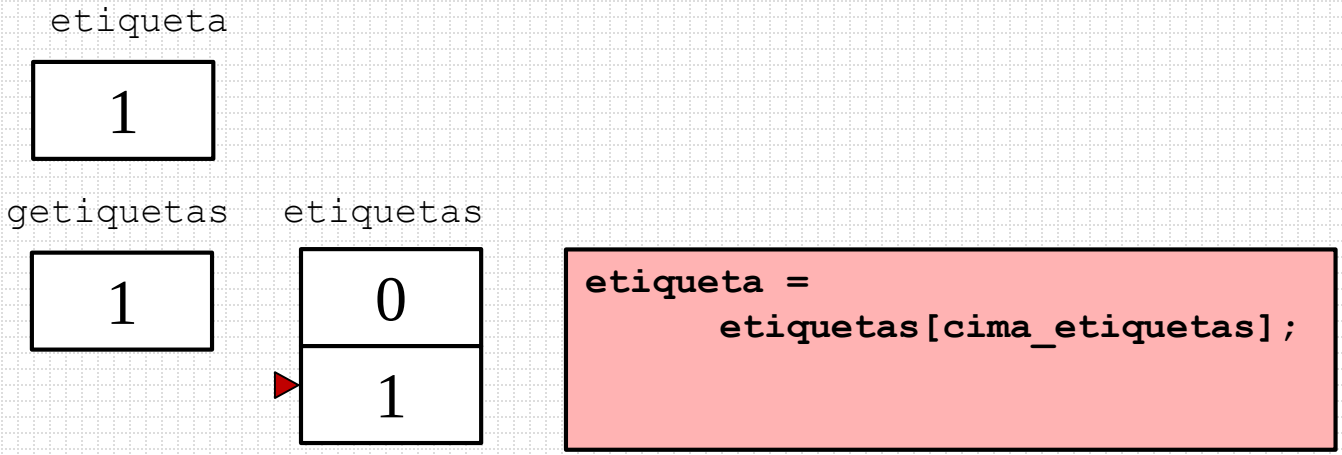

```
getiqueta++;
cima_etiquetas++;
etiquetas[cima_etiquetas]
        =getiqueta;
etiqueta = getiqueta;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta

```
main{
 /* ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
  }
/*···*/}
```

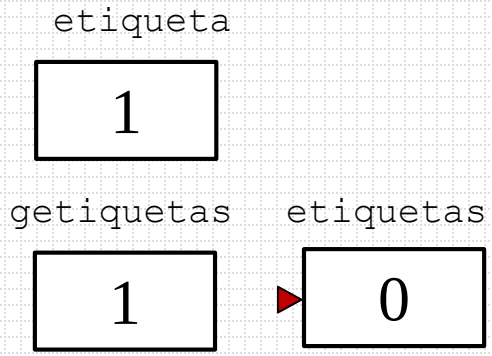


Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta
  - Tras haberla usado

```
main{
 /*  ··· */
 if ()
 {
          if
          {
                        /*··· */
          };}
 else
 {
          if
          {
                        /*··· */
          }
          else
          {
                        /*··· */
          }
  }
/*···*/}
```



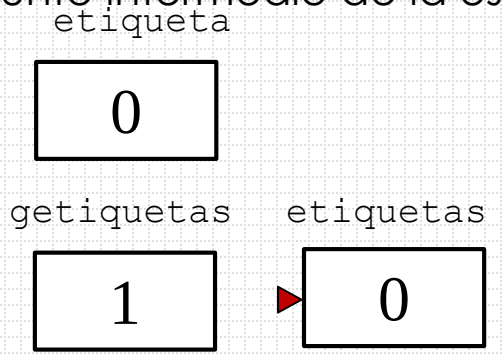

```
etiqueta =
      etiquetas[cima_etiquetas];

cima_etiquetas--;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Uso de etiqueta actual en punto intermedio de la estructura

```
main{
 /*  ··· */
 if ()
 {
         if
         {
                  /*··· */
         };}
 else
 {
         if
         {
                  /*··· */
         }
         else
         {
                  /*··· */
         }
  }
/*···*/}
```



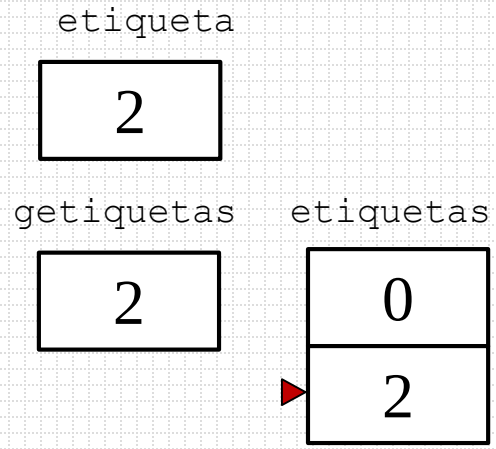

```
etiqueta =
etiquetas[cima_etiquetas];
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al inicio de la estructura

```
main{
 /*  ··· */
 if ()
 {
        if
        {
                    /*··· */
        };}
 else
 {
        if
        {
                  /*··· */
        }
        else
        {
                  /*··· */
        }
  }
/*···*/}
```



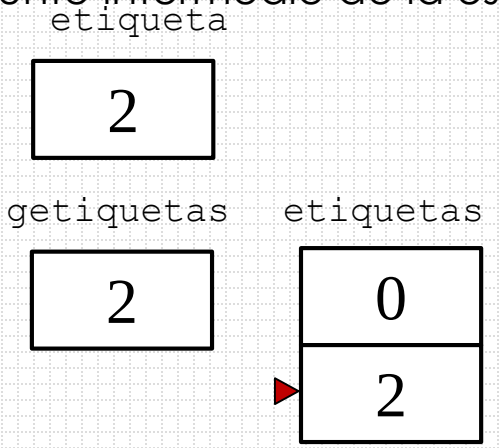

```
getiqueta++;
cima_etiquetas++;
etiquetas[cima_etiquetas]
        =getiqueta;
etiqueta = getiqueta;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Uso de etiqueta actual en punto intermedio de la estructura

```
main{
 /* ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
  }
/*···*/}
```



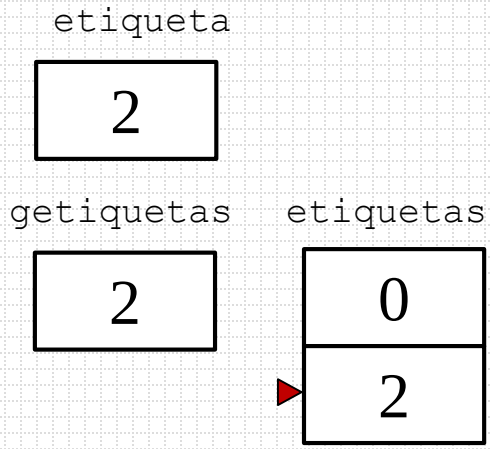

```
etiqueta =
etiquetas[cima_etiquetas];
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta

```
main{
 /* ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
  }
/*···*/}
```



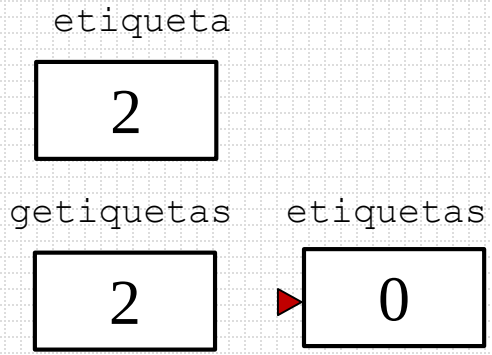

```
etiqueta =
        etiquetas[cima_etiquetas];
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta
  - Tras haberla usado

```
main{
 /*  ··· */
 if ()
 {
          if
          {
                    /*··· */
          };}
 else
 {
          if
          {
                    /*··· */
          }
          else
          {
                    /*··· */
          }
   }
/*···*/}
```



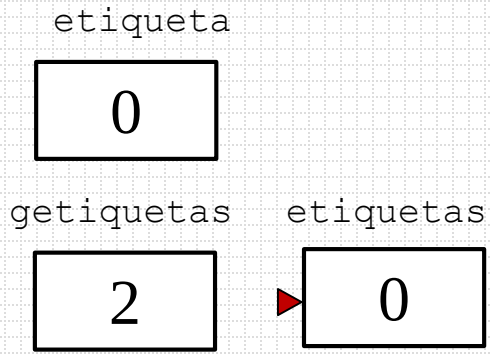

```
etiqueta =
      etiquetas[cima_etiquetas];

cima_etiquetas--;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta
  - Tras haberla usado

```
main{
 /*  ··· */
 if ()
 {
         if
         {
                  /*··· */
         };}
 else
 {
         if
         {
                  /*··· */
         }
         else
         {
                  /*··· */
         }
/**/}
/*···*/}
```



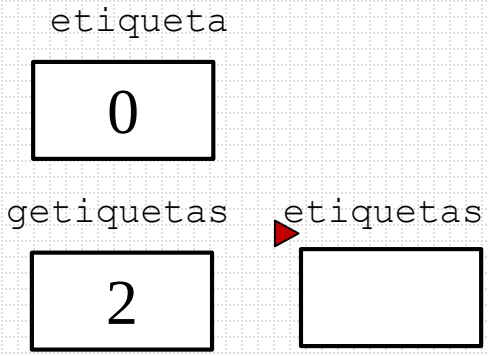

```
etiqueta =
      etiquetas[cima_etiquetas];
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- Al final de la estructura,
  - Cuando se usa etiqueta

```
main{
 /*  ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
/**/}
/*···*/}
```



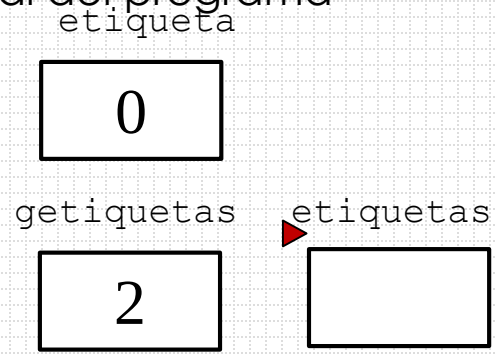

```
etiqueta =
      etiquetas[cima_etiquetas];

cima_etiquetas--;
```

Generación de código: gestión de las etiquetas

Análisis de las situaciones que debe afrontar el control de etiquetas

- No hay cambios hasta el final del programa

```
main{
 /* ··· */
 if ()
 {
        if
        {
                /*··· */
        };}
 else
 {
        if
        {
                /*··· */
        }
        else
        {
                /*··· */
        }
/**/}
/*···*/}
```




```
etiqueta =
      etiquetas[cima_etiquetas];

cima_etiquetas--;
```

Generación de código para gestión de errores en tiempo de ejecución

- Revisa el material de generación de código t13-14
  - Se trata de gestionar correctamente los siguientes errores en tiempo de ejecución
    - División por cero
    - Indexar con índice cuyo valor está fuera de rango (negativo o superior al máximo según la declaración)

```
void escribir_fin(FILE* fpasm)
{
        /* ESCRITURA DEL FINAL DEL PROGRAMA GESTIÓN DE ERROR EN TIEMPO DE EJECUCIÓN (DIVISION POR 0)
        Y ÍNDICE FUERA DE RANGO Y
        RESTAURACION DEL PUNTERO DE PILA A PARTIR DE LA VARIABLE __esp
        SENTENCIA DE RETORNO DEL PROGRAMA*/
        jmp near fin                        // FIN CORRECTO, SALTO AL FIN DE PROGRAMA
        fin_error_division:                 // PROCESO DE ERROR DIVISION
        push dword msg_error_division       // ESCRITUR ADEL MENSAJE
        call print_string
        add esp, 4
        call print_endofline
        jmp near fin                        // SALTO AL FIN DE PROGRAMA
        fin_indice_fuera_rango:             // PROCESO DE ERROR POR ÍNDICE FUERA DE RANGO
        push dword msg_error_indice_vector  // ESCRITURA DEL MENSAJE
        call print_string
        add esp, 4
        call print_endofline
        jmp near fin                        // SALTO AL FIN DE PROGRAMA
        fin:                                // FIN DE PROGRAMA
        mov esp, [__esp]                    // RESTAURACIÓN DE PUNTERO DE PILA
        ret                                 // SALIDA DEL MAIN

}
```

Generación de código para sentencias condicionales

- Revisa el material de generación de código 56-59

```
// PREVIA A LA INVOCACIÓN DE ESTA: acciones al inicio del bloque
/*          getiqueta++;
            cima_etiquetas++;
            etiquetas[cima_etiquetas]=getiqueta;
            etiqueta = getiqueta;
*/
// PARA EL INICIO TANTO DE UN IFTHEN COMO THE UN IFTHENELSE
void ifthenelse_inicio /  if_then_inicio(FILE * fpasm, int exp_es_variable, int etiqueta)
{
            ; SE SACA DE LA PILA EL VALOR DE LA EXPRESIÓN
            pop eax
            if (exp_es_variable == 1)
                        mov eax, [eax]
            cmp eax,0
            ; SI ES CERO SE SALTA AL FINAL DE LA RAMA THEN
            je near fin_then etiqueta
}

// ESTA FUNCION SÓLO USA ETIQUETA
/* Acciones cuando hay que usar la etiqueta : e<-c
**********************************************
            etiqueta = etiquetas[cima_etiquetas];
*/
// PARA EL FINAL DE LA RAMA THEN DE UN BLOQUE IF THEN Y, POR LO TANTO, FINAL DE SU
void ifthen_fin(FILE * fpasm, int etiqueta)
{
            ...
            ; SE IMPRIME LA ETIQUETA DE FINAL DE BLOQUE THEN
            fin_then etiqueta:
}
```

Generación de código para sentencias condicionales

- Revisa el material de generación de código 56-59

```
// ESTA FUNCION SÓLO USA ETIQUETA ES EL FIN DE LA RAMA THEN PERO NO DEL BLOQUE ESTRUCTURAL COMPLETO IFTHENELSE
/* Acciones cuando hay que usar la etiqueta : e<-c
*********************************************
            etiqueta = etiquetas[cima_etiquetas];
*/
// PARA EL FINAL DE LA RAMA THEN DE UNA ESTRUCTURA IFTHENELSE
void ifthenelse_fin_then( FILE * fpasm, int etiqueta)
{
            // SE SALTA AL FIN DEL IFTHENELSE, ES DECIR, LA RAMA ELSE
            jmp near fin_ifelse etiqueta
            // SE ESCRIBE LA ETIQUETA DE FIN DE LA RAMA THEN
            fin_then etiqueta:
}


// ESTA FUNCIÓN REQUIERE ADEMÁS DE USO DE ETIQUETA FIN DE BLOQUE TRAS ELLA
/*En aquellos casos que una función use etiqueta y acabe el bloque e<-c; pop
*************************************************************************
            etiqueta = etiquetas[cima_etiquetas];
            <llamada a la función>
            cima_etiquetas--;
*/
// PARA EL FINAL DE UNA RAMA THEN EN UNA ESTRUCTURA IFTHENELSE COMPLETA (AL FINAL DE LA RAMA ELSE)
void ifthenelse_fin( FILE * fpasm, int etiqueta)
{
            // SE ESCRIBE LA ETIQUETA DEL FINAL DE LA ESTRUCTURA IFTHENELSE
            fin_ifelse etiqueta:
}
```

Generación de código para sentencias iterativas

- Revisa el material de generación de código 60-62

```
// PREVIA A LA INVOCACIÓN DE ESTA: acciones al inicio del bloque
/*          getiqueta++;
            cima_etiquetas++;
            etiquetas[cima_etiquetas]=getiqueta;
            etiqueta = getiqueta;
*/
void while_inicio(FILE * fpasm, int etiqueta)
{
            // SE ESCRIBE LA ETIQUETA DE INICIO DE WHILE
            inicio_while etiqueta:
}

// ESTA FUNCION SÓLO USA ETIQUETA
/* Acciones cuando hay que usar la etiqueta : e<-c
**********************************************
            etiqueta = etiquetas[cima_etiquetas];
*/
void while_exp_pila (FILE * fpasm, int exp_es_variable, int etiqueta)
{
            // SE SACA DE LA CIMA DE LA PILA EL VALOR DE LA EXPRESIÓN QUE GOBIERNA EL BUCLE
            tpop eax
            if (exp_es_variable >0)
                        mov eax, [eax]
            cmp eax, 0
            // SI ES 0 SE SALTA AL FINAL DEL WHILE, HABRÍAMOS TERMINADO
            je near fin_while etiqueta
}
```

Generación de código para sentencias iterativas

- Revisa el material de generación de código 60-62

```
// ESTA FUNCIÓN REQUIERE ADEMÁS DE USO DE ETIQUETA FIN DE BLOQUE TRAS ELLA
/*En aquellos casos que una función use etiqueta y acabe el bloque e<-c; pop
**************************************************************************
            etiqueta = etiquetas[cima_etiquetas];
            <llamada a la función>
            cima_etiquetas--;
*/
void while_fin( FILE * fpasm, int etiqueta)
{
            // SE SALTA DE NUEVO AL PRINCIIO DEL BUCLE PARA VOLVER A EVALUAR LA CONDICION DE SALIDA
            jmp near inicio_while etiqueta
            // SE ESCRIBE LA ETIQUETA DE FIN DEL BUCLE
            fin_while etiqueta:
}
```

Generación de código para indexar vectores

- Revisa el material de generación de código 41-46

```
// EN LA CIMA DE LA PILA ESTA EL VALOR DE LA EXPRESIÓN QUE CONTIENE EL VALOR DEL ÍNDICE
// SE PROPORCIONA LA INDICACIÓN DE SI ES UNA VARIABLE O NO COMO ARGUMENTO
// DEJA EN LA CIMA DE LA PILA LA DIRECCIÓN DEL ELEMENTO VECTOR
// ES IMPORTNTE DEJAR CLARO QUE ES UNA DIRECCIÓN
void escribir_elemento_vector(FILE * fpasm,char * nombre_vector, int tam_max, int exp_es_direccion)
{
        // SE SACA DE LA PILA A UN REGISTRO EL VALOR DEL ÍNDICE
        pop dword eax
        // HACIENDO LO QUE PROCEDA EN EL CASO DE QUE SEA UNA DIRECCIÓN (VARIABLE O EQUIVALENTE)
        if (exp_es_direccion == 1)
                tmov dword eax, [eax]
        // SE PROGRAMA EL CONTROL DE ERRORES EN TIEMPO DE EJECUCIÓN
        /* SI EL INDICE ES <0 SE TERMINA EL PROGRAMA, SI NO, CONTINUA          */
        cmp eax, 0
        // SE SUPONE QUE EN LA DIRECCIÓN fin_indice_fuera_rango SE PROCESA ESTE ERROR EN TIEMPO DE EJECUCIÓN
        jl near fin_indice_fuera_rango
        /* SI EL INDICE ES > MAXIMO PERMITIDO SE TERMINA EL PROGRAMA, SI NO, CONTINUA      */
        // EL TAMANO MÁXIMO SE PROPORCIONA COMO ARGUMENTO
        cmp eax, tam_max-1
        jg near fin_indice_fuera_rango
        // UNA OPCIÓN ES CALCULAR CON lea LA DIRECCIÓN EFECTIVA DEL ELEMENTO INDEXADO TRAS CALCULARLA
        // DESPLAZANDO DESDE EL INICIO DEL VECTOR EL VALOR DEL INDICE
        mov dword edx, _nombre_vector
        lea eax, [edx + eax*4]  /* DIRECCION ELEMENTO INDEXADO EN eax          */
        push dword eax  /* DIRECCION ELEMENTO INDEXADO EN CIMA PILA */
}
```

Generación de código para indexar vectores

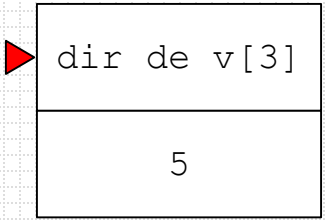
- Revisa el material de generación de código 41-46
  - A la hora de asignar valor a un elemento de un vector hay que tener en cuenta que

```
v[3] = 5;
```

    - El lugar al que se va asignar ya no es (como en el caso de `v = 5;`) solucionable generando código para algo como

```
mov dword [_x], <LO QUE HAYA EN LA CIMA DE LA PILA>
```

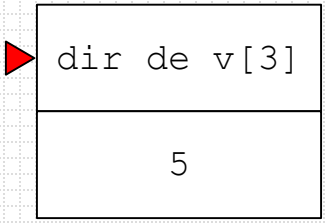
    - Y se optará por un enfoque que utilice la pila también para guardar la dirección a la que se va a asignar que antes de utilizar la siguiente función tendrá el siguiente aspecto



Generación de código para indexar vectores

- Revisa el material de generación de código 41-46
  - A la hora de asignar valor a un elemento de un vector hay que tener en cuenta que

v[3] = 5;




```
void asignarDestinoEnPila(FILE* fpasm, int es_variable)
{/* ESCRIBE EL CÓDIGO PARA REALIZAR UNA ASIGNACIÓN AL DESTINO QUE ESTÁ EN LA CIMA DE LA PILA DE LO QUE ESTÉ DEBAJO DE ÉL
ES LA QUE SE TIENE QUE UTILIZAR PARA HACER ASIGNACIONES A ELEMENTOS DE VECTOR (NO INCLUIDO ESTE AÑO) Y A ATRIBUTOS
DE INSTANCIA
  SE RECUPERA DE LA PILA LO QUE HAYA POR EJEMPLO EN EL REGISTRO eax
  SI es_variable == 0 (ES UN VALOR) DIRECTAMENTE SE ASIGNA A LA VARIABLE _nombre
  EN OTRO CASO es_variable == 1 (ES UNA DIRECCIÓN, UN NOMBRE DE VARIABLE) HAY QUE OBTENER SU VALOR DESREFERENCIANDOEL
VALOR ES [eax]
  SE RECUPERA DE LA PILA LA DIRECCIÓN DONDE SE VA A ASIGNAR, POR EJEMPLO EN EL REGISTRO ebx
  SE HACE EFECTIVA LA ASIGNACION CUIDADO, SÓLO SE TRATA COMO NO VARIABLE CUANDO SE TIENE LA SEGURIDAD DE QUE EN LA CIMA DE
LA PILA ESTÁ EL VALOR, SI A LA DERECHA DE LA ASIGNACIÓN HAY UN ACCESO A VECTOR O A INSTANCIA O A VARIABLE NO ES CTE Y HAY
QUE LLAMARLO CON 1*/

  // TOMAMOS LA DIRECCIÓN DONDE TENEMOS QUE ASIGNAR
            pop dword ebx
  // TOMAMOS EL VALOR QUE SE DEBE ASIGNAR INCLUSO DESREFERENCIANDO EN EL CASO DE QUE SEA UNA VARIABLE
            pop dword eax
            if (es_variable)  mov dword eax, [eax]
  // ASIGNAMOS
            mov dword [ebx], eax
}
```

Generación de código para funciones

- Revisa el material de generación de código 63-103: declaración de funciones
  - Estas funciones escriben el código necesario para
    - Declarar
    - Terminar una función

```
// ESCRIBE LA PLANTILLA DE ENTRADA PARA UNA FUNCION nombre_funcion CON num_var_loc VARIABLES LOCALES
/*          _nombre_funcion:
                    push ebp
                    mov ebp, esp
                    sub esp , 4*num_var_loc
*/
void declararFuncion(FILE * fd_asm, char * nombre_funcion, int num_var_loc)
{
          _nombre_funcion:                  // ETIQUETA DE INICIO DE LA FUNCIÓN
          push ebp                          // PRESERVACIÓN DE ebp / esp
          mov ebp, esp
          sub esp , 4*num_var_loc           // RESERVA DE ESPACIO PARA LAS VARIABLES LOCALES EN LA PILA
}

// TAREAS ASOCIADAS CON EL FINAL DEL CUERPO DEL CÓDIGO DE UNA FUCIÓN EN SU DECLARACION
void retornarFuncion(FILE * fd_asm, int es_variable)
{
          fprintf(fd_asm, "\tpop eax\n");     // RETORNO DE LA FUNCIÓN (EL VALOR DE LA EXPRESIÓN ESTÁ EN LA PILA
          if (es_variable == 1)             // Y TIENE QUE DEJARSE EN eax
                    fprintf(fd_asm, "\tmov dword eax, [eax]\n");
          fprintf(fd_asm, "\tmov esp,ebp\n"); /* restaurar el puntero de pila */
          fprintf(fd_asm, "\tpop ebp\n");     /* sacar de la pila ebp */
          fprintf(fd_asm, "\tret\n");         /* vuelve al programa llamante y saca de la pila la dir de retorno */
}
```

Generación de código para funciones

- Revisa el material de generación de código 63-103: declaración de funciones
  - Funciones adicionales
    - Gestionar variables locales y parámetros
      - Recuerda que el espacio local (variables locales y parámetros) de una función se encuentra en la pila
      - Los parámetros están ya pues los ha introducido en la pila el programa llamante
      - Las variables locales se reservan cuando se inicia la función mediante el código analizado en las trasparenias anteriores

```
// SE DEJA EN LA CIMA DE LA PILA LA DIRECCIÓN DEL PARÁMETRO NECESARIO IDENTIFICADO POR SU POSICIÓN
// IMPORTANTE, EL PRIMER ARGUMENTO LE CORRESPONDE POSICION 0
void escribirParametro(FILE* fpasm, int pos_parametro, int num_total_parametros)
{           int d_ebp;
            d_ebp = 4*( 1 + (num_total_parametros - pos_parametro));

            lea eax ,[ebp + d_ebp]      // UNA ALTERNATIVA ES CALCULAR LA DIRECCIÓN EFECTIVA CON lea DESPLAZANDO DESDE ebp
            push dword eax
}

// SE DEJA EN LA CIMA DE LA PILA LA DIRECCIÓN DEL PARÁMETRO NECESARIO IDENTIFICADO POR SU POSICIÓN
// IMPORTANTE, EL PRIMER ARGUMENTO LE CORRESPONDE POSICION 1
void escribirVariableLocal(FILE* fpasm, int posicion_variable_local)
{           int d_ebp;

            d_ebp = 4*posicion_variable_local;
            lea eax , [ebp - d_edp]
            push dword eax
}
```

Generación de código para funciones

- Revisa el material de generación de código 63-103: llamada a funciones
  - Las llamadas a funciones desde el programa llamante requieren dos tareas
    - Ajuste de las expresiones que serán utilizadas como argumento
      - Cuando se trata una expresión su valor está en la cima de la pila pero puede ocurrir que sea una variable en casos, como por ejemplo, f(x), esa x es una variable
      - El paso de argumentos a las funciones se hace por valor por lo que hay que obtener el valor antes de realizar la llamada
    - Invocación de la función, que realiza la llamada en sí

```
// FUNCIÓN QUE ASEGURA QUE LOS ARGUMENTOS SE PASAN POR VALOR
void operandoEnPilaAArgumento(FILE * fd_asm, int es_variable)
{           if (es_variable == 1)
            {
                        pop eax                 // EN EL CASO DE QUE EN LA PILA TENGAMOS UNA VARIABLE Y NO UN VALOR
                        mov eax, [eax]          // SE SACA, SE ACCEDE AL VALOR Y SE VUELVE A AINTRODUCIR EN LA PILA
                        push eax
            }
}

// FUNCIÓN QUE REALIZA LA LLAMADA EN SÍ
void llamarFuncion(FILE * fd_asm, char * nombre_funcion, int num_argumentos)
{
            call nombre_funcion             // SE LLAMA A LA FUNCIÓN
            add esp, num_argumentos*4       // SE LIMPIA LA PILA (DE LOS ARGUMENTOS USADOS EN LA LLAMADA)
            push dword eax                  // EN NUESTRO LENGUAJE LAS LLAMADAS A FUNCIONES SON EXPRESIONES
                                            // POR LO QUE EL EFECTO DE SER LLAMADAS (SU RETORNO) DEBE SER DEJADO
                                            // LA PILA
}
```