

Generación de código para funciones

Ejemplo introductorio: escritura de funciones en NASM

Marina de la Cruz
Alfonso Ortega

- En estas transparencias pondremos una subrutina ASPLE y la generación de código equivalente

```
function int cero()  
begin  
    return 0  
end
```

```
_cero:  
    push ebp  
    mov ebp, esp  
    sub esp, 0  
  
    push dword 0  
    pop eax  
  
    mov esp, ebp  
    pop ebp  
    ret
```

Generación de código para funciones

Ejemplo introductorio: escritura de funciones en NASM

Marina de la Cruz
Alfonso Ortega

- En estas transparencias pondremos una subrutina ASPLE y la generación de código equivalente

```
begin  
    .  
    .  
    .  
    . . . cero();  
    .  
    .  
end
```

```
.  
.  
.  
main:  
    .  
    .  
    .  
    . . . call _cero  
    . . . add esp, 0  
    .  
    .  
ret
```

Generación de código para funciones

Condiciones para subrutinas recursivas y reentrantes

Marina de la Cruz
Alfonso Ortega

- Una subrutina **reentrante** [deMiguel01] es aquella que puede ser llamada desde varios puntos simultáneamente sin modificar su comportamiento
- Una subrutina **recursiva** es aquella que puede llamarse a sí misma, por lo tanto, una subrutina recursiva necesariamente tiene que ser reentrante. Las condiciones para que una subrutina sea recursiva y reentrante son las mismas.
- Las condiciones necesarias para que una subrutina sea recursiva y reentrante tiene que ver con que los datos utilizados por ellas no estén contenidos en posiciones fijas de memoria. Por lo tanto **es suficiente con utilizar la pila** para almacenar **datos globales, locales y parámetros**.

Generación de código para funciones

Llamada a subrutinas: conceptos preliminares

Marina de la Cruz
Alfonso Ortega

- Para llamar a una subrutina se cumple lo que se conoce como “convenio de llamadas” que debe ser especificado explícitamente al diseñar el compilador.
- El convenio de llamadas tiene que dar respuesta a las siguientes preguntas:
 - ¿Cómo se comunican los argumentos desde el programa llamante a la función llamada?
 - ¿Cómo se comunican los resultados desde la función llamada hasta el programa llamante?
- Se recomienda utilizar el siguiente convenio de llamadas:
 - Para comunicar los argumentos desde el programa llamante a la función llamada:
 - Se utilizará la pila
 - El programa llamante dejará en la pila los argumentos de la llamada **en el mismo orden** en el que serán utilizados
 - Si la función está codificada correctamente, tras terminar su ejecución, dejará la pila en el mismo estado en el que la encontró
 - Tras la llamada, el programa llamante tiene la responsabilidad de eliminar de la pila
 - Para comunicar los resultados desde la función llamada:
 - Se utilizará el registro extendido `eax`.

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Explicando más detalladamente el proceso consideremos el siguiente programa ASPLE

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
segment .data
  _z dd 0
segment .text
global main
extern lee_entero,
imprime_entero
main:
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
  push dword _z
```

El programa llamante introduce en la pila los argumentos

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
```

Realiza la llamada a la función

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4
```

Y "limpia" la pila. Esta instrucción es equivalente a `pop <registro>` pero no necesita modificar ningún registro

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4
```

De la misma manera hay que hacer con una llamada a una función definida por el programador

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
```

El retorno de la función se encuentra en `eax`

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Imaginemos el código NASM equivalente. Comencemos por el programa principal

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Para comprender el código que hay que generar para la función imaginemos cómo está la pila (donde se tiene que almacenar toda la información) en una llamada si z contiene el valor 3

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Recuérdese que la pila de 32 bits tiene su puntero almacenado en el registro esp recuerde, también, que la pila “crece” hacia posiciones de memoria de valor menor, se representará gráficamente con crecimiento “hacia abajo”

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

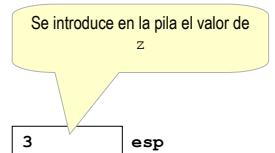
  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```



Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Recuérdese que la pila de 32 bits tiene su puntero almacenado en el registro `esp` recuerde, también, que la pila “crece” hacia posiciones de memoria de valor menor, se representará gráficamente con crecimiento “hacia abajo”

```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg;
end

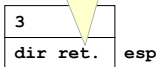
input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```

La instrucción `call` introduce en la pila la dirección de retorno



Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Esta es la pila que se encuentra la función cuando comienza su ejecución
- Está claro dónde se encuentra el argumento de la función (`arg`)
- Se tiene que utilizar también la pila para localizar las variables locales (`auxArg`)
- Debemos conocer qué expresión, en función de los registros de gestión de la pila, sirve para localizar en ella argumentos y variables locales.
- Obsérvese que el compilador tendrá que generar el código necesario para el cuerpo de la función.
- Por las decisiones de diseño tomadas, se utilizará la pila, por lo tanto, el valor del registro `esp` cambiará.
- NASM proporciona otro registro `ebp` específicamente para poder solucionar esta circunstancia.
- Para ello hay que hacer lo siguiente:
 - Guardar el valor de `ebp` (también en la pila)
 - Utilizar `ebp` para mantener una copia del valor de `esp` en el momento de la entrada
 - Utilizar `ebp` para localizar los parámetros y las variables locales
 - Recuperar los valores de los dos registros antes de salir de la función

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

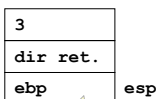
```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg;
end

input z;
output doble(z)
end
```

```
_doble:
push ebp
```



Se guarda el valor de `ebp` en la pila

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

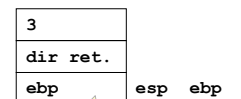
```
begin
int z;

function int doble(int arg)
begin
int auxArg;

auxArg := arg;
return 2*arg;
end

input z;
output doble(z)
end
```

```
_doble:
push ebp
mov ebp, esp
```



Se guarda en `ebp` el valor de `esp`

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Recuérdese que usamos un tamaño de dato de 32 bits, que son 4 bytes. Así es fácil saber que `arg` se encuentra separado de `ebp` por 2 dword, es decir + 8 bytes

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg end input z; output doble(z) end</pre>	<pre>_doble: push ebp mov ebp, esp</pre>	<table border="1"><tr><td>3</td><td>ebp+8</td></tr><tr><td>dir ret.</td><td></td></tr><tr><td>ebp</td><td>esp ebp</td></tr></table>	3	ebp+8	dir ret.		ebp	esp ebp
3	ebp+8							
dir ret.								
ebp	esp ebp							

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Ahora sólo queda localizar (reservar espacio en la pila y saber como acceder posteriormente) las variables locales (`auxArg`). Para ello se utiliza las posiciones libres bajo la cima de la pila (como con `push`, restamos a `esp` lo necesario)

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg end input z; output doble(z) end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4</pre>	<table border="1"><tr><td>3</td><td>ebp+8</td></tr><tr><td>dir ret.</td><td></td></tr><tr><td>ebp</td><td>ebp</td></tr><tr><td></td><td>esp</td></tr></table>	3	ebp+8	dir ret.		ebp	ebp		esp
3	ebp+8									
dir ret.										
ebp	ebp									
	esp									

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Por lo tanto, `auxArg` se encontrará siempre 4 bytes por debajo de `ebp` es decir, en `ebp-4`

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg end input z; output doble(z) end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4</pre>	<table border="1"><tr><td>3</td><td>ebp+8</td></tr><tr><td>dir ret.</td><td></td></tr><tr><td>ebp</td><td>ebp</td></tr><tr><td></td><td>esp ebp-4</td></tr></table>	3	ebp+8	dir ret.		ebp	ebp		esp ebp-4
3	ebp+8									
dir ret.										
ebp	ebp									
	esp ebp-4									

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Ahora se puede generar el código para el cuerpo de la función (aquí se genera "a mano" y seguro que diferirá de lo generado por el compilador)

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg end input z; output doble(z) end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4 mov dword eax, [ebp+8] mov dword [ebp-4], eax</pre>	<table border="1"><tr><td>3</td><td>ebp+8</td></tr><tr><td>dir ret.</td><td></td></tr><tr><td>ebp</td><td>ebp</td></tr><tr><td>3</td><td>esp ebp-4</td></tr></table>	3	ebp+8	dir ret.		ebp	ebp	3	esp ebp-4
3	ebp+8									
dir ret.										
ebp	ebp									
3	esp ebp-4									

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Ahora se puede generar el código para el cuerpo de la función (aquí se genera "a mano" y seguro que diferirá de lo generado por el compilador)

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg; end input z; output doble(z); end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4 mov dword eax, [ebp+8] mov dword [ebp-4], eax mov dword edx, 2 mul edx</pre>	<table border="1"> <tr><td>3</td><td>ebp+8</td></tr> <tr><td>dir ret.</td><td></td></tr> <tr><td>ebp</td><td>ebp</td></tr> <tr><td>3</td><td>esp ebp-4</td></tr> </table>	3	ebp+8	dir ret.		ebp	ebp	3	esp ebp-4
3	ebp+8									
dir ret.										
ebp	ebp									
3	esp ebp-4									

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Antes de salir de la función se debe "recuperar" los valores iniciales de `ebp` y `esp`.

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg; end input z; output doble(z); end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4 mov dword eax, [ebp+8] mov dword [ebp-4], eax mov dword edx, 2 mul edx mov esp, ebp</pre>	<table border="1"> <tr><td>3</td><td>ebp+8</td></tr> <tr><td>dir ret.</td><td></td></tr> <tr><td>ebp</td><td>ebp esp</td></tr> <tr><td>3</td><td>ebp-4</td></tr> </table>	3	ebp+8	dir ret.		ebp	ebp esp	3	ebp-4
3	ebp+8									
dir ret.										
ebp	ebp esp									
3	ebp-4									

Se copia en `esp` su valor que está guardado en `ebp`. Obsérvese que es como si se hicieran los `pop` necesarios para eliminar las variables locales

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Antes de salir de la función se debe "recuperar" los valores iniciales de `ebp` y `esp`.

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg; end input z; output doble(z); end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4 mov dword eax, [ebp+8] mov dword [ebp-4], eax mov dword edx, 2 mul edx mov esp, ebp pop ebp</pre>	<table border="1"> <tr><td>3</td><td>ebp+8</td></tr> <tr><td>dir ret.</td><td>esp</td></tr> <tr><td>ebp</td><td>ebp</td></tr> <tr><td>3</td><td>ebp-4</td></tr> </table>	3	ebp+8	dir ret.	esp	ebp	ebp	3	ebp-4
3	ebp+8									
dir ret.	esp									
ebp	ebp									
3	ebp-4									

Se recupera (desde la pila) el antiguo valor de `ebp`. Obsérvese que, tras modificar en la instrucción anterior `esp` la cima de la pila apunta ahora a la posición que guarda la copia de `ebp`

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Antes de salir de la función se debe "recuperar" los valores iniciales de `ebp` y `esp`.

<pre>begin int z; function int doble(int arg) begin int auxArg; auxArg := arg; return 2*arg; end input z; output doble(z); end</pre>	<pre>_doble: push ebp mov ebp, esp sub esp, 4 mov dword eax, [ebp+8] mov dword [ebp-4], eax mov dword edx, 2 mul edx mov esp, ebp pop ebp ret</pre>	<table border="1"> <tr><td>3</td><td>esp</td></tr> <tr><td>dir ret.</td><td></td></tr> </table>	3	esp	dir ret.	
3	esp					
dir ret.						

Y ya se puede devolver el control al programa llamante mediante la instrucción `ret`. Esta instrucción elimina de la pila la dirección de retorno

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Retornamos, por lo tanto, al código del programa principal que se encuentra la pila como la tenía antes de la ejecución de la instrucción `call`. Obsérvese que la pila todavía contiene los argumentos de la función, el programa debe "limpiar" la pila.

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```

3 esp

Generación de código para funciones

Llamada a subrutinas: ejemplo preliminar

Marina de la Cruz
Alfonso Ortega

- Retornamos, por lo tanto, al código del programa principal que se encuentra la pila como la tenía antes de la ejecución de la instrucción `call`. Obsérvese que la pila todavía contiene los argumentos de la función, el programa debe "limpiar" la pila.

```
begin
int z;

function int doble(int arg)
begin
  int auxArg;

  auxArg := arg;
  return 2*arg;
end

input z;
output doble(z)
end
```

```
...
main:
push dword _z
call lee_entero
add esp, 4

push dword [_z]
call _doble
add esp, 4

push dword eax
call imprime_entero
add esp, 4
ret
```

esp
3

Se vio previamente la posibilidad de simular los `pop` necesarios para limpiar la pila sumando directamente en el puntero de la cima (`esp`)

Generación de código para funciones

Código para la llamada a funciones

Marina de la Cruz
Alfonso Ortega

- Lo observado en el ejemplo anterior puede generalizarse para cualquier función con cualquier número de variables locales y cualquier número de argumentos:
- El programa llamante debe primero **copiar los argumentos de la función en la pila en el mismo orden en el que aparecen en la declaración de la función**

```
push dword <1er argumento>
...
push dword <último argumento>
```

- Debe, luego **llamar a la función**:

```
call _<nombre función>
```

- Y **limpiar la pila**:

```
add esp, <4 * n° argumentos de la función>
```

- Antes de continuar con su código sabiendo que `eax` contiene el retorno de la función (el código escrito a continuación imprime el entero devuelto por la función llamada)

```
push eax
call imprime_entero
add esp, 4
```

Generación de código para funciones

Código inicial y final para funciones recursivas y reentrantes

Marina de la Cruz
Alfonso Ortega

- Lo observado en el ejemplo anterior puede generalizarse para cualquier función con cualquier número de variables locales y cualquier número de argumentos:
- Las **primeras sentencias NASM del cuerpo de cualquier función** deben ser las siguientes:

```
_<nombre_función> :
push ebp
mov ebp, esp
sub esp, <4*n° var locales>
```

- Y **las últimas** deben ser las siguientes:

```
mov esp, ebp
pop ebp
ret
```

Generación de código para funciones

Localización de contenido de parámetros

Marina de la Cruz
Alfonso Ortega

- Y, en el **código del cuerpo de la función**, cuando se quiera acceder al valor de los parámetros se utilizará la expresión

```
[ebp + <4 + 4*posición del parámetro en declaración>]
```

- Recuerde que, el primer 4 es para “saltar” la dirección de retorno.
- Por ejemplo:
 - El valor del último argumento será `[ebp+8]`.
 - El valor del penúltimo argumento será `[ebp+12]`.
 - ...
 - El valor del i-esimo argumento será `[ebp+<4+4*(num-argumentos-i)>]`, contando i con origen 0.

Generación de código para funciones

Localización de contenido de variables locales

Marina de la Cruz
Alfonso Ortega

- Y, en el **código del cuerpo de la función**, cuando se quiera acceder al valor de las variables locales se utilizará la expresión

```
[ebp - <4*posición de la variable en declaración>]
```

- Por ejemplo:
 - El valor de la primera variable local será `[ebp-4]`.
 - El valor de la segunda variable local será `[ebp-8]`.
 - ...
 - El valor de la i-esima variable local será `[ebp-<4*i>]`.

Generación de código para funciones

Localización de dirección de parámetros y variables locales

Marina de la Cruz
Alfonso Ortega

- Sin embargo, como se ha indicado en la generación de código de otras construcciones de ASPL (por ejemplo en el de las expresiones que contienen identificadores), a veces es necesario utilizar “la dirección” y no “el contenido”

- Recuerdese, por ejemplo, que para la variable global ASPL `z`,
 - La expresión para referirse al contenido es

```
[_z]
```

- Pero, la expresión para la dirección es

```
__z
```

- Lamentablemente, las funciones reentrantes y recursivas no utilizan nombres explícitos para los parámetros y las variables locales, por lo que la expresión equivalente para su dirección (`ebp+<desplazamiento>` y `ebp-<desplazamiento>`, respectivamente) son incorrectas en NASM.
- NASM proporciona la instrucción `lea` para resolver este problema.

Generación de código para funciones

Localización de dirección de parámetros y variables locales

Marina de la Cruz
Alfonso Ortega

- En este curso se va a utilizar la siguiente sintaxis para la instrucción `lea` de NASM

```
lea <registro> [<expresión>]
```

- Donde
 - `<registro>`, es el nombre de un registro (por ejemplo `eax`)
 - `<expresión>`, puede ser, por ejemplo `ebp+12`.

Generación de código para funciones

Resumen: tratamiento de identificadores

Marina de la Cruz
Alfonso Ortega

- Como resumen de estos últimos puntos, considérese el identificador ASPLE *z*. Se utilizará las siguientes expresiones NASM para

- Acceder a su contenido

- Si es una variable global

```
[_z]
```

- Si es el argumento *i*-ésimo de una función

```
[ebp+<4+4*i>]
```

- Si es la variable local *i*-ésima de una función

```
[ebp-<4*i>]
```

Generación de código para funciones

Resumen: tratamiento de identificadores

Marina de la Cruz
Alfonso Ortega

- Acceder a su dirección

- Si es una variable global

```
_z
```

- Si es el argumento *i*-ésimo de una función, lo siguiente deja en el registro *eax* la dirección

```
lea eax, [ebp+<4+4*i>]
```

- Si es la variable local *i*-ésima de una función

```
lea eax, [ebp-<4*i>]
```

Generación de código para funciones

Otras modificaciones necesarias

Marina de la Cruz
Alfonso Ortega

- Es **muy importante que el alumno propague este tratamiento a todos los lugares donde pueda aparecer un identificador.**
- Es decir, siempre que hay un identificador, se tiene que determinar
 - Si es
 - una variable global
 - una variable local de una función
 - un argumento de una función
 - Si el código que se necesita generar accederá
 - A su contenido
 - A su dirección
- Ya que cada opción necesita un tratamiento diferente.

Generación de código para funciones

Bibliografía

Marina de la Cruz
Alfonso Ortega

- [deMiguel01] Pedro de Miguel Anasagasti, *Fundamentos de los computadores*. Ed. Paraninfo, Thomson Learning. 2001