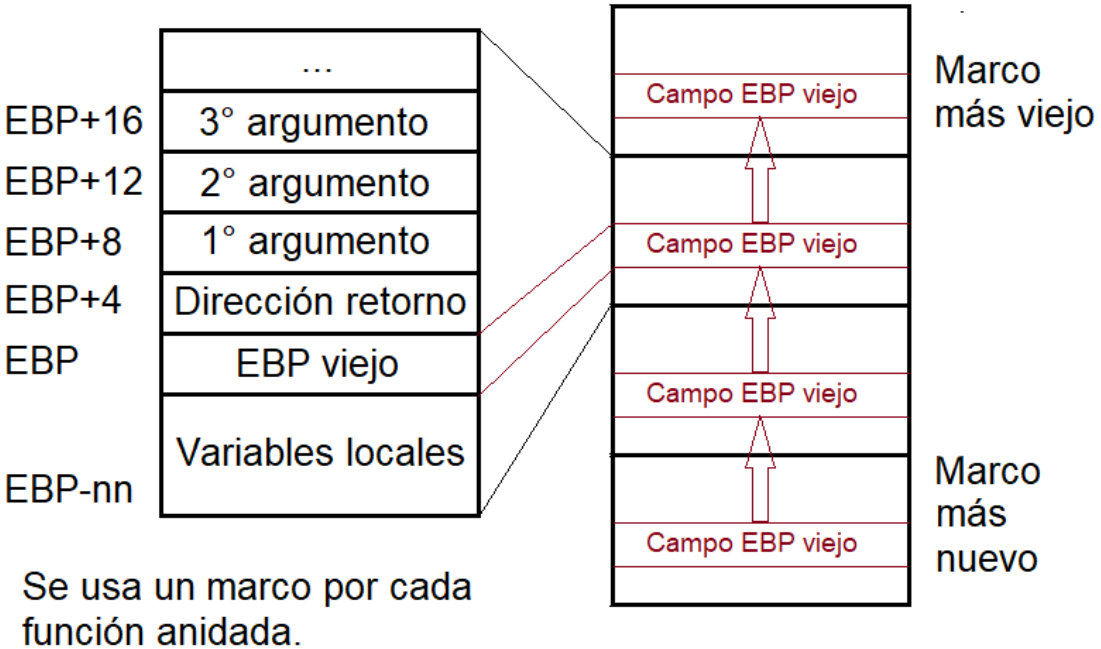
Application Binary Interface (ABI)

Es la interfaz entre dos módulos en binario.

En nuestro caso vamos a ver el caso especial de la interfaz entre el compilador gcc y programas en Assembler en procesadores Intel de 32 bits.

Distintos compiladores usan diferentes métodos de comunicación con módulos y, necesariamente, la interfaz debe variar cuando se usan distintos procesadores.

Uso de la pila



Se usa un marco por cada función anidada.

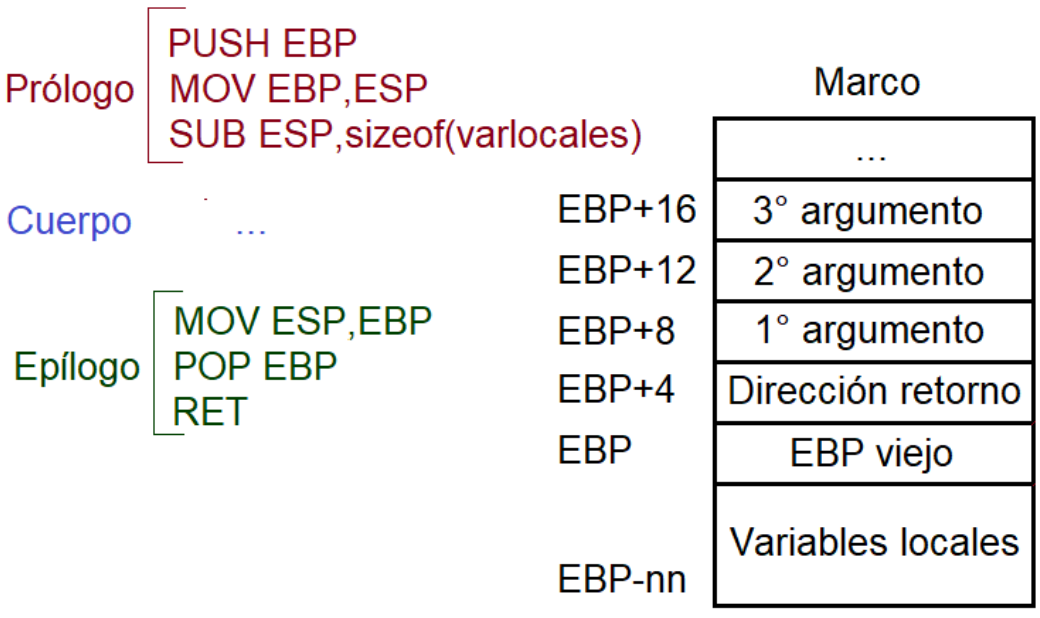
Llamada a función de C en Assembler

Los argumentos se ponen en la pila de derecha a izquierda. Aunque sean de 1 o 2 bytes, se ponen 4 bytes en la pila. Si el argumento es de 8 bytes, se ponen los 8 bytes en la pila. Al volver de la función, la rutina llamadora debe balancear la pila.

Ejemplo: compilar mifunc(23, 47)

```
PUSH DWORD 47
PUSH DWORD 23
CALL mifunc
ADD ESP, 8
```

Partes de una función en C compilada



Valor de retorno

La ubicación del valor de retorno depende de su tamaño:

1 byte: AL
2 bytes: AX
4 bytes: EAX (entero o puntero) o ST(0) (float)
8 bytes: EDX:EAX (entero) o ST(0) (double)

ST(0) es el primer elemento de la pila de la unidad de punto flotante.

Ejemplo 1

Convertir a Assembler:

```
int func1(int pri, int seg)
{
  seg++;
  return pri + seg * 7;
}
```

```
func1:
    PUSH EBP
    MOV EBP,ESP
    INC DWORD [EBP+12]
    MOV EAX,[EBP+12]
    MOV EDX,7
    MUL EDX
    ADD EAX,[EBP+8]
    POP EBP
    RET
```

Conversión de tipos de datos enteros

Aquí se ve la conversión de un tipo de datos pequeño a otro más grande. Por ejemplo short a int.

Si el tipo más pequeño es signado: se completan los bits del tipo más grande con el bit de signo (más significativo) del más pequeño.

Ejemplos: short 0xA0A0 a int: 0xFFFFA0A0.
short 0x2020 a int: 0x00002020.

Si el tipo más pequeño no es signado: se completa con ceros.

Ejemplo unsigned short 0xA0A0 a int: 0x0000A0A0.

Instrucciones MOVZX y MOVSX

Sirven para convertir (cast en inglés) de tipos de datos pequeños y tipos de datos más grandes.

MOVZX se usa cuando el tipo pequeño es no signado
MOVSX se usa cuando el tipo pequeño es signado

Ejemplo 1: Para convertir el char (signado) almacenado en BL en int y ponerlo en EAX se usa MOVSX EAX,BL

Ejemplo 2: Para convertir el unsigned short almacenado en el primer argumento a int en ECX se usa MOVZX ECX,WORD [EBP+8]

Instrucción LEA

La instrucción LEA tiene dos operandos: un registro y un direccionamiento indirecto a memoria. La instrucción carga en el registro el puntero a esa dirección de memoria (lo que está dentro de los corchetes). Esta instrucción no valida el puntero, así que no genera excepción 13 o 14 si el puntero es inválido.

Ejemplos:

```
LEA ECX,[EBP-4]                 ECX⇦EBP-4
LEA EDX,[ESI+4*EDI+0x224]       EDX⇦ESI+4*EDI+0x224
```

Ejemplo 2

```
int func2(int k, char m)
{
  int varlocal;   // [EBP-4]
  varlocal = k - m;
  return varlocal * k;
}
```

```
func2:
   PUSH EBP
   MOV EBP,ESP
   SUB ESP,4
   MOVSX EBX,BYTE [EBP+12]
   MOV EAX,[EBP+8]
   SUB EAX,EBX
   MOV [EBP-4],EAX
   MUL DWORD [EBP+8]
   MOV ESP,EBP
   POP EBP
   RET
```

Compilación de comparaciones

En lenguaje C, es habitual usar instrucciones if, while, for que comparan dos operandos y saltan si es igual, mayor, menor, etc. Las instrucciones de salto dependen de si los datos son signados o no:

Salto si:	Signado	No signado
Igual	JE etiqueta	JE etiqueta
Mayor	JG etiqueta	JA etiqueta
Menor	JL etiqueta	JB etiqueta
Distinto	JNE etiqueta	JNE etiqueta
Mayor o igual	JGE etiqueta	JAE etiqueta
Menor o igual	JLE etiqueta	JBE etiqueta

Ejemplo 3

```
void func3(void)
{
  int arr[6];    // [EBP-24]
  short m;       // [EBP-28]
  int *ptrArr;   // [EBP-32]
  ptrArr = arr;
  for (m=0; m<6; m++)
  {
    arr[m] = m;
  }
}

// arr[m] =
// [EBP - 24 + m*4]
```

```
func3:
   PUSH EBP
   MOV EBP,ESP
   SUB ESP,6*4+4+4
   LEA EAX,[EBP-24]
   MOV [EBP-32],EAX
   MOV WORD [EBP-28],0
ciclo:
   MOVSX ECX,WORD [EBP-28]
   MOV [EBP-24+ECX*4],ECX
   INC WORD [EBP-28]
   CMP WORD [EBP-28],6
   JL ciclo
   MOV ESP,EBP
   POP EBP
   RET
```

Secciones

Los módulos en Assembler deben usar las mismas secciones que usa el compilador para que pueda coexistir código en C y código en Assembler sin problemas.

Las secciones son:

.text = código
.rodata = datos constantes
.data = datos inicializados que se pueden modificar
.bss = datos no inicializados (el runtime de C los inicializa a cero).

Ejemplo 4

Variables globales y estáticas:

```
int arr[100];
char str[] = "Hola mundo";
static int ctr = 3;
extern char value;
```

```
    global arr
    global str
    extern value
    section .bss
arr resd 100
    section .data
str db "Hola mundo", 0
ctr dd 3
```

Pasaje de parámetros en 64 bits

Al contrario que en el caso de 32 bits en que todos los parámetros se pasan por la pila, en el caso de 64 bits los primeros parámetros se pasan por registro:

Los primeros seis parámetros enteros se ubican en los registros RDI, RSI, RDX, RCX, R8, R9 en ese orden.

Los primeros ocho parámetros de punto flotante se ubican en los registros XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 en ese orden.

Si hay más parámetros, se pasan por la pila.