Arquitectura de Computadoras

Nicolás Margenat

1Q 2022

Contents

1 Compilación

1.1 GCC Flags

Flags	Descripción
-c	compila y no linkedita
-m32	le indica a gcc que genere archivo con formato elf para arquitectura de 32bits
-o $1	renombra el archivo de output a $1
-fno-exceptions	no te tira excepciones
-S	te genera el file .asm
-masm=intel	es para que el -S te devuelva el .asm con flavour de Intel

1.2 NASM Flags

Flags	Descripcion
-f elf32	le indica a nasm que genere el file en formato elf para arquitectura de 32 bits
-f elf64	le indica a nasm que genere el file en formato elf para arquitectura de 64 bits

Compilación de archivo C y ASM

```
> nasm -f elfXX $1.asm -o $1.o
> gcc -c -mXX $2.c -o $2.o
> gcc -mXX $1.o $2.o -o $3
```

donde XX es la arquitectura para la que generamos el archivo (32 o 64 bits).

Compilación archivos ASM 32bits

```
> nasm -f elf32 $1.asm -o $1.o
> ld -melf_i386 $1.o -o $1
```

2 ASM

2.1 Sections

1. .rodata : datos constantes, inalterables
2. .data : reserva espacio
3. .bss : reserva espacio que no se aloca si no se usa
4. .text : zona de código

2.2 Stack

Como se ve el stack al llamar una función de ASM:

ESP	Cantidad de argumentos
ESP + 4	Path al programa
ESP + 8	Dirección del 1er argumento
ESP + 12	Dirección del 2do argumento
ESP + 16	Dirección del 3er argumento
ESP + (n+1)*4	Dirección del n-argumento
	NULL (4 bytes)
	Dirección de la 1er variable de entorno
	Dirección de la 2da variable de entorno
	Dirección de la 3er variable de entorno
	Dirección de la n-variable de entorno
	NULL (4 bytes)

Una manera rápida de *armar y desarmar el stack* es usando las macros: `enter` y `leave`.

2.3 Registros

Registro	AL/AH	AX	EAX	RAX
Size(bytes)	1	2	4	8
Designation	Byte	Word	DWord	QWord

3 ASM y C

3.1 Registros a preservar entre llamadas a funciones

- EBX
- ESI
- EDI
- EBP
- ESP

3.2 Parámetros de una función

Según la arquitectura, el compilador pasa de manera diferente los argumentos de las funciones:

32 bits $\Rightarrow$ Stack
64 bits $\Rightarrow$ 1° Registros, 2° Stack

Los **registros** que se utilizan para pasar argumentos en 64 bits son (por convención de C):

- RDI
- RSI
- RDX
- R10
- R9
- R8

Ejemplo (en C)

Si en C tenemos:

```
foo(paramA, paramB, paramC);
```

se pushea en el stack de derecha a izquierda, de manera que queda:

^	Dirección de retorno
|	paramA
|	paramB
|	paramC

3.3 Valores a retornar

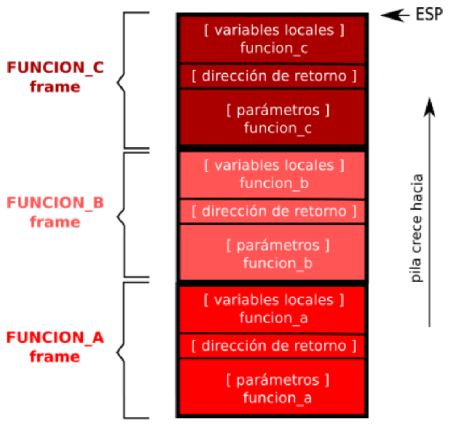
Si el valor es $\leq$ 32bits se retorna en EAX.
Si el valor es $>$ 32bits se retorna la parte alta en EDX y la parte baja en EAX.
Si es un dato más complejo retorna un puntero formado por $\underbrace{\text{EDX}}_{\text{segmento}} : \underbrace{\text{EAX}}_{\text{offset}}$.

3.4 StackFrame

Para armar (y desarmar) el StackFrame hacemos:

```
push ebp        ; Armado de SF
mov ebp, esp    ;
...
; codigo
...
mov esp, ebp    ; Desarmado de SF
pop ebp         ;
```

Finalmente, la pila terminaría pareciéndose a esto:



4 Links útiles

4.1 Syscall IDs

Arquitectura de 32bits: `https://tinyurl.com/mpj5kxbw`
Arquitectura de 64bits: `https://tinyurl.com/53khdbsv`