## Materiale

- Syscalls Table
- darkdust.net - GDB Cheatsheet
- godbolt.org | Compiler Explorer

# Playing with assembly - parte 2

## Funzioni in assembly

Per creare una funzione si utilizza un label che definisce l'indirizzo di inizio della funzione. In assembly per le funzioni si utilizza la seguente convenzione di utilizzo dei registri: 6 registri per passare argomenti alla funzione in ordine dal primo argomenti al sesto argomento `rdi, rsi, rdx, rcx, r8, r9`. Si utilizzano i registri `rax` e `rdx` rispettivamente per il primo valore e secondo valore di ritorno.

### CALL

Per chiamare una funzione in assembly si utilizza il istruzione `call`. Call salva l'indirizzo della prossima istruzione (quella immediatamente dopo `call`) nello stack: `push rip`, quindi, fa un'operazione di `JUMP` al label della funzione.

### RET

Per ritornare da una funzione si utilizza istruzione `ret`. Ret fa un'operazione di `pop` dallo stack, quindi, carica in registro `rip` l'indirizzo del prossimo commando (quello dopo `call`).

### Utilizzo di RBP - Base Pointer

#### Prologue

Quando si vuole utilizzare lo stack dentro una funziona, si può usare la tecnica dove si salva `push rbp` dentro lo stack, si assegna al `rbp` il puntatore dello stack `rsp` `mov rbp, rsp` e si crea un'area nello stack per la funzione facendo un `sub rsp, X` dove `X` è il numero di byte da riservare. Questa procedura è chiamato il **Prologue** che consiste del preparare lo stack e registri per la funzione.

```
# Prologue
push rbp
mov  rbp, rsp
sub  rsp, X

... # function's operations
```

#### Epilogue

Epilogo è l'operazione inversa al prologo, consiste del ripristinare `rbp` e `rsp`.

```
...# function's operations

#Epilogue
mov rsp, rbp
pop rbp
ret
```

Si può usato l'istruzione `leave` per fare `mov` e `pop` del epilogo in assembly `x86`.

```
leave
ret
```

## Esercizio 1 - leggi una linea da stdin, scrivi in stdout

```
.intel_syntax noprefix
.global _start

.section .text
_start:

lea rdi,buffer
mov rsi,64
call Read
lea rdi,buffer
mov rsi,64
call Write
call Exit

Read: # read from stdin

mov r8,rdi
mov r9,rsi

mov rsi,r8
mov rdi, 0
mov rdx, r9
mov rax, 0
syscall
ret

Write: # write to stdout
mov r8,rdi
mov r9,rsi

mov rsi,r8
mov rax,1
mov rdi,1
mov rdx,r9
```

```
syscall
ret

Exit: # exit app
mov rax,60
xor rdi,rdi
syscall

.section .data
buffer: .space 64
```

## Assembly Debug - GDB

`strace` stampa le syscalls dell'applicazione

`set disassembly-flavor intel` # per settare il formato intel

`info functions` # stampa le funzioni del programma

`disass _print` # disassembla la funzione _print

`layout asm` # avviare in modalità assembly live

`layout reg` # mostra i registri in tempo reale

`x/s 0x4000` # x per vedere cosa ce dentro un indirizzo, /s string, indirizzo, oppure registro, 7x per hex

`si` # step-in

`ni` # next step

```
 1 .intel_syntax noprefix
 2 .global _start
 3 .section .text
 4 _start:
 5
 6     lea rbx,buffer
 7     mov rdx,10
 8     mov rcx,20
 9     add rdx,rcx
10     mov rdi,2
11     mov [rbx],rdx
12     mov rsi,rbx
13     call Print
14
15
16 Print: #Print(size,buffer)
17     mov r8,rdi
18     mov rax,1
19     mov rdi,1
20     mov rdx,r8
21     syscall
```

```
 22     ret
 23
 24
 25 .section .data
 26 buffer: .space 100
```

## Esercizio 2

Scrivere un programma assembly ch date due variabile numeriche embedded nel programma ne stampa la somma.

```
.intel_syntax noprefix
.global _start
.section .text
_start:

mov rdx,5600
mov rsi,4000
add rsi,rdx
lea rdi,buffer
call TO_ASCII

lea rdi,buffer
mov rsi,rax
call Print

call EXIT


EXIT:
mov rax,60
mov rdi,0
syscall

Print: # (rdi = buffer, rsi = size)
mov r8,rdi
mov r9,rsi

mov rax,1
mov rdi,1
mov rsi,r8
mov rdx,r9
syscall

ret

# func to_ascii

TO_ASCII: # (rdi = buffer, rsi = number)
# prologue
push rbp
```

```
mov rbp, rsp
sub rsp, 50

xor rax,rax
mov rax,rsi
xor rcx,rcx

xor rbx,rbx
mov rbx,10

DIVID: # saving reminder in rbp
cmp rax,0
je REVERSE_INIT

xor rdx,rdx
div rbx
add dl,0x30
mov byte ptr [rbp+rcx], dl
inc rcx
jmp DIVID

REVERSE_INIT:
xor rbx,rbx
dec rcx
REVERSE: # storing in buffer ascii char in reverse order
cmp rcx,0
jl EXIT_TO_ASCII

mov dl, byte ptr [rbp+rcx]
mov byte ptr [rdi+rbx], dl
inc rbx
dec rcx
jmp REVERSE

EXIT_TO_ASCII:
inc rbx
mov byte ptr [rdi+rbx],0 # NULL char 0x0 = 0
mov byte ptr [rdi+rbx+1],10 # LINE FEED '\n' char 0xa = 10 questo toglie "%" da
shell
mov rax,rbx

# epilogue
mov rsp,rbp
pop rbp
ret

.section .data
buffer: .space 100
```