

Práctica de Organización del Computador II

x86 Assembly y ABI

Primer Cuatrimestre 2023

Organización del Computador II
DC - UBA

Introducción

Hoy vamos a ver:

Hoy vamos a ver:

- Programación en Assembly x86

Hoy vamos a ver:

- Programación en Assembly x86
- Convención C

Hoy vamos a ver:

- Programación en Assembly x86
- Convención C
- Uso de la pila y registros en llamadas a funciones

Hoy vamos a ver:

- Programación en Assembly x86
- Convención C
- Uso de la pila y registros en llamadas a funciones
- Ejercitación

Programación en Assembly x86

Estructura de programa en asm

```
1 ; Programa Hola Mundo que usa syscall de x86_64
2 ;
3 ;* Obtenido de man 2 syscall
4 ;* En x86-64
5 ;*
6 ;* System Ret Ret Error
7 ;* call # val val2 -
8 ;* rax rax rdx
9 ;*
10 ;* Los parámetros de syscall deben pasarse así:
11 ;* arg1 arg2 arg3 arg4 arg5 arg6
12 ;* rdi rsi rdx r10 r8 r9
13
14 %define SYS_WRITE 1
15 %define SYS_EXIT 60
16 %define STDOUT 1
17
18 section .data
19 msg db '¡Hola Mundo!', 10
20 len EQU $ - msg
21
22 global _start
23 section .text
24 _start:
25     mov     rax, SYS_WRITE ; ssize_t write(int fd, const void *buf, size_t count);
26     mov     rdi, STDOUT
27     mov     rsi, msg
28     mov     rdx, len
29     syscall ; fast system call ; RCX --> dirección de retorno
30             ; RFLAGS --> R11
31     ; en este punto RAX tiene los bytes escritos por sys_write()
32     mov     rax, SYS_EXIT ; void exit(int status);
33     mov     rdi, 0
34     syscall
```

```
1 ; Programa Hola Mundo que usa syscall de x86_64
2 ;
3 ;* Obtenido de man 2 syscall
4 ;* En x86-64
5 ;*
6 ;* System Ret Ret Error
7 ;* call # val val2 -
8 ;* rax rax rdx
9 ;*
10 ;* Los parámetros de syscall deben pasarse así:
11 ;* arg1 arg2 arg3 arg4 arg5 arg6
12 ;* rdi rsi rdx r10 r8 r9
13
```

```
14 %define SYS_WRITE 1
15 %define SYS_EXIT 60
16 %define STDOUT 1
```

directivas de
preprocesador

```
17
18 section .data
19 msg db '¡Hola Mundo!', 10
20 len EQU $ - msg
```

directivas de
ensamblador

```
21
22 global _start
23 section .text
24 _start:
25     mov     rax, SYS_WRITE ; ssize_t write(int fd, const void *buf, size_t count);
26     mov     rdi, STDOUT
27     mov     rsi, msg
28     mov     rdx, len
29     syscall ; fast system call ; RCX --> dirección de retorno
30             ; RFLAGS --> R11
31     ; en este punto RAX tiene los bytes escritos por sys_write()
32     mov     rax, SYS_EXIT ; void exit(int status);
33     mov     rdi, 0
34     syscall
```

```
1 ; Programa Hola Mundo que usa syscall de x86_64
2 ;
3 ;* Obtenido de man 2 syscall
4 ;* En x86-64
5 ;*
6 ;* System Ret Ret Error
7 ;* call # val val2 -
8 ;* rax rax rdx
9 ;*
10 ;* Los parámetros de syscall deben pasarse así:
11 ;* arg1 arg2 arg3 arg4 arg5 arg6
12 ;* rdi rsi rdx r10 r8 r9
13
14 %define SYS_WRITE 1
15 %define SYS_EXIT 60
16 %define STDOUT 1
17
18 section .data
19 msg db '¡Hola Mundo!', 10
20 len EQU $ - msg
21
22 global _start
23 section .text
24 _start:
25     mov     rax, SYS_WRITE ; ssize_t write(int fd, const void *buf, size_t count);
26     mov     rdi, STDOUT
27     mov     rsi, msg
28     mov     rdx, len
29     syscall ; fast system call ; RCX --> dirección de retorno
30             ; RFLAGS --> R11
31     ; en este punto RAX tiene los bytes escritos por sys_write()
32     mov     rax, SYS_EXIT ; void exit(int status);
33     mov     rdi, 0
34     syscall
```

pseudo-instrucciones

```
1 ; Programa Hola Mundo que usa syscall de x86_64
2 ;
3 ;* Obtenido de man 2 syscall
4 ;* En x86-64
5 ;*
6 ;* System Ret Ret Error
7 ;* call # val val2 -
8 ;* rax rax rdx
9 ;*
10 ;* Los parámetros de syscall deben pasarse así:
11 ;* arg1 arg2 arg3 arg4 arg5 arg6
12 ;* rdi rsi rdx r10 r8 r9
13
14 %define SYS_WRITE 1
15 %define SYS_EXIT 60
16 %define STDOUT 1
17
```

```
18 section .data
19 msg db '¡Hola Mundo!', 10
20 len EQU $ - msg
```

sección .data

```
21
22 global _start
23 section .text
24 _start:
25     mov     rax, SYS_WRITE ; ssize_t write(int fd, const void *buf, size_t count);
26     mov     rdi, STDOUT
27     mov     rsi, msg
28     mov     rdx, len
29     syscall ; fast system call ; RCX --> dirección de retorno
30             ; RFLAGS --> R11
31     ; en este punto RAX tiene los bytes escritos por sys_write()
32     mov     rax, SYS_EXIT ; void exit(int status);
33     mov     rdi, 0
34     syscall
```

sección .text

Los operandos pueden ser:

Los operandos pueden ser:

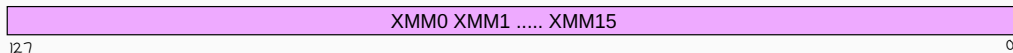
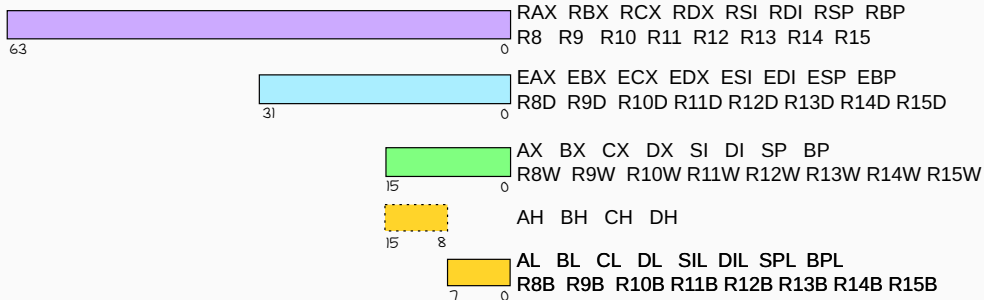
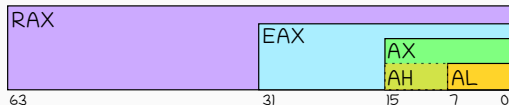
- Registros

Los operandos pueden ser:

- Registros
- Memoria

Los operandos pueden ser:

- Registros
- Memoria
- Inmediatos



- [displacement]

- [displacement]
- [reg]

- [displacement]
- [reg]
- [reg + reg*scale] scale es 1, 2, 4, u 8

- [displacement]
- [reg]
- [reg + reg*scale] scale es 1, 2, 4, u 8
- [reg + displacement]

- [displacement]
- [reg]
- [reg + reg*scale] scale es 1, 2, 4, u 8
- [reg + displacement]
- [reg + reg*scale + displacement]

Forma general

$$\left[\frac{\text{Base}}{\text{RAX}} + \left(\frac{\text{Index}}{\text{RAX}} * \frac{\text{Scale}}{1} \right) + \frac{\text{Displacement}}{\text{Cte. 32 bits}} \right]$$

...	...	2
R15	R15	4
	(NO RSP)	8

[1003]	; sólo displacement
[rbx]	; registro base solamente
[rbx + rsi*4]	; base + index * scale
[rbx + rdx]	; scale es 1
[rax - 8]	; displacement es -8
[rax + rdi*8 - 200]	; todos los componentes
[rbx + counter]	; dirección de variable 'counter' como desplazamiento

200	; decimal
0200	; sigue siendo decimal - el 0 no lo hace octal
0xc8	; hexa - el querido 0x
0hc8	; hexa - 0h también es aceptado
0q310	; octal - prefijo 0q
11001000b	; binario - sufijo b
0b1100_1000	; binario - prefijo 0b, guiones permitidos

La mayoría de las instrucciones con 2 operandos toman la siguiente forma:

- `inst reg, reg`
- `inst reg, mem`
- `inst reg, imm`
- `inst mem, reg`
- `inst mem, imm`

db	0x55	; sólo el byte 0x55
db	0x55,0x56,0x57	; 3 bytes sucesivos
db	'a',0x55	; 0x97, 0x55
db	'hello',13,10,'\$'	; strings como cadenas de bytes
db	`hola\nmundo\n\0`	; strings con "C-style \-escapes"
dw	0x1234	; 0x34 0x12
dd	0x12345678	; 0x78 0x56 0x34 0x12
dq	0x123456789abcdef0	; constante de 8 bytes

Para reservar espacio (sin inicializar):

buffer:	resb	64	; reserva 64 bytes
wordvar:	resw	1	; reserva un word
realarray:	resq	10	; array de 10 qwords

Estas pseudo-instrucciones deben ir en **section .bss**

Para ensamblar:

```
$ nasm -f elf64 -g -F DWARF holamundo.asm
```

Linking:

```
$ ld -o holamundo holamundo.o
```

Ejecutamos:

```
$ ./holamundo
```

Makefile básico para nasm:

```
AS      := nasm
ASFLAGS := -f elf64 -F DWARF -g -Wall
LD      := ld
LDFLAGS := -g
TARGET  := holaorga

.PHONY: all clean
all: $(TARGET)
# assembly
holaorga.o: holaorga.asm
    $(AS) $(ASFLAGS) $<
# linking
$(TARGET): holaorga.o
    $(LD) $(LDFLAGS) $< -o $@
clean:
    rm -rf *.o $(TARGET)
```