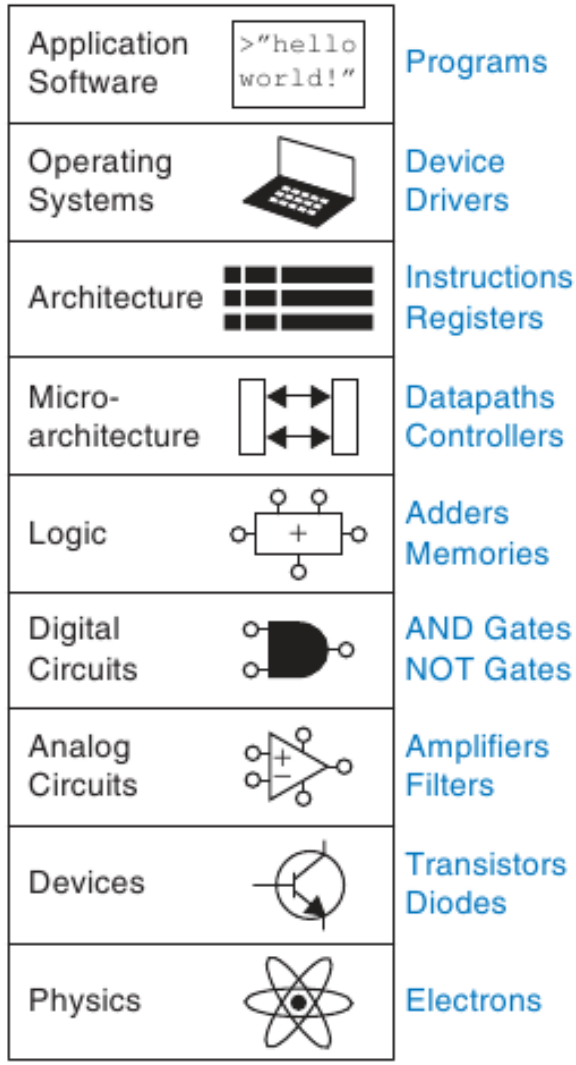
Introducción a Lenguaje Ensamblador y Arquitectura x86-64

Arquitectura del Computador - LCC - FCEIA-UNR

11 de septiembre de 2023

Niveles de abstracción para una computadora[1]



[1] Harris, David and Harris, Sarah L. Digital design and computer architecture. Morgan Kaufmann, 2010.

Niveles de abstracción para una computadora

- El nivel de abstracción **Arquitectura** describe una computadora desde la perspectiva del programador.
- Por ejemplo, la arquitectura Intel x86 está definida por un conjunto de instrucciones y registros.
- Sin embrago, una arquitectura particular puede implementarse mediante diferentes microarquitecturas con diferentes relaciones de precio/rendimiento/potencia.
- Por ejemplo, Intel y Advanced Micro Devices (AMD) venden varios microprocesadores (con diferentes microarquitecturas) que pertenecen a la misma arquitectura.

La arquitectura x86-64

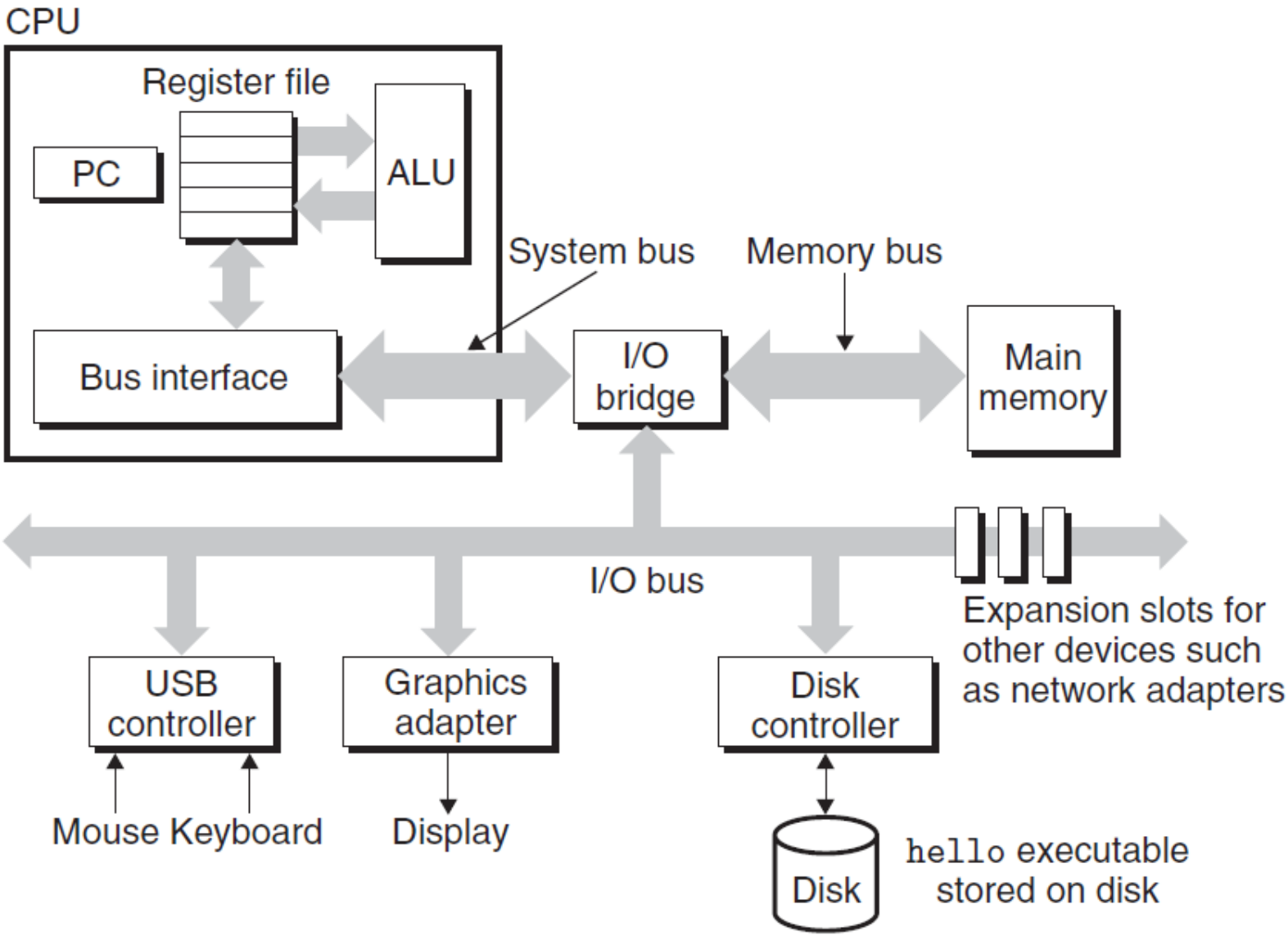
- La arquitectura x86-64 es una ampliación de la arquitectura x86 lanzada por Intel con el procesador Intel 8086 en el año 1978 como una arquitectura de 16 bits.
- Luego evolucionó a una arquitectura de 32 bits cuando apareció el procesador Intel 80386 en el año 1985, denominada inicialmente i386 o x86-32 y finalmente IA-32.
- Desde 1999 hasta el 2003, AMD amplió esta arquitectura de 32 bits de Intel a una de 64 bits y la llamó x86-64 en los primeros documentos y posteriormente AMD64.
- Intel pronto adoptó las extensiones de la arquitectura de AMD bajo el nombre de IA-32e o EM64T, y finalmente la denominó Intel 64.

La arquitectura x86-64

- La arquitectura x86-64 (AMD64 o Intel 64) es de tipo CISC (Complex Instruction Set Computer).
- Registros y buses de datos y direcciones de 64 bits.
- También permite operaciones con valores de 256, 128, 32, 16 y 8 bits.
- SSE (Streaming SIMD[2] Extensions).
- Soporte mucho mayor al espacio de direcciones virtuales y físicas.

[2] SIMD (del inglés Single Instruction, Multiple Data, en español: "una instrucción, múltiples datos".)

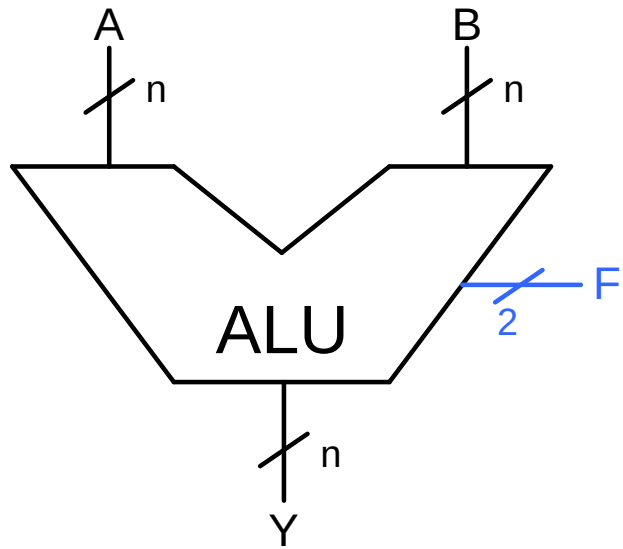
Organización del hardware de una computadora típica[3]



[3] Bryant, Randal E and David Richard, O'Hallaron and David Richard, O'Hallaron. Computer systems: a programmer's perspective. 2015.

La ALU

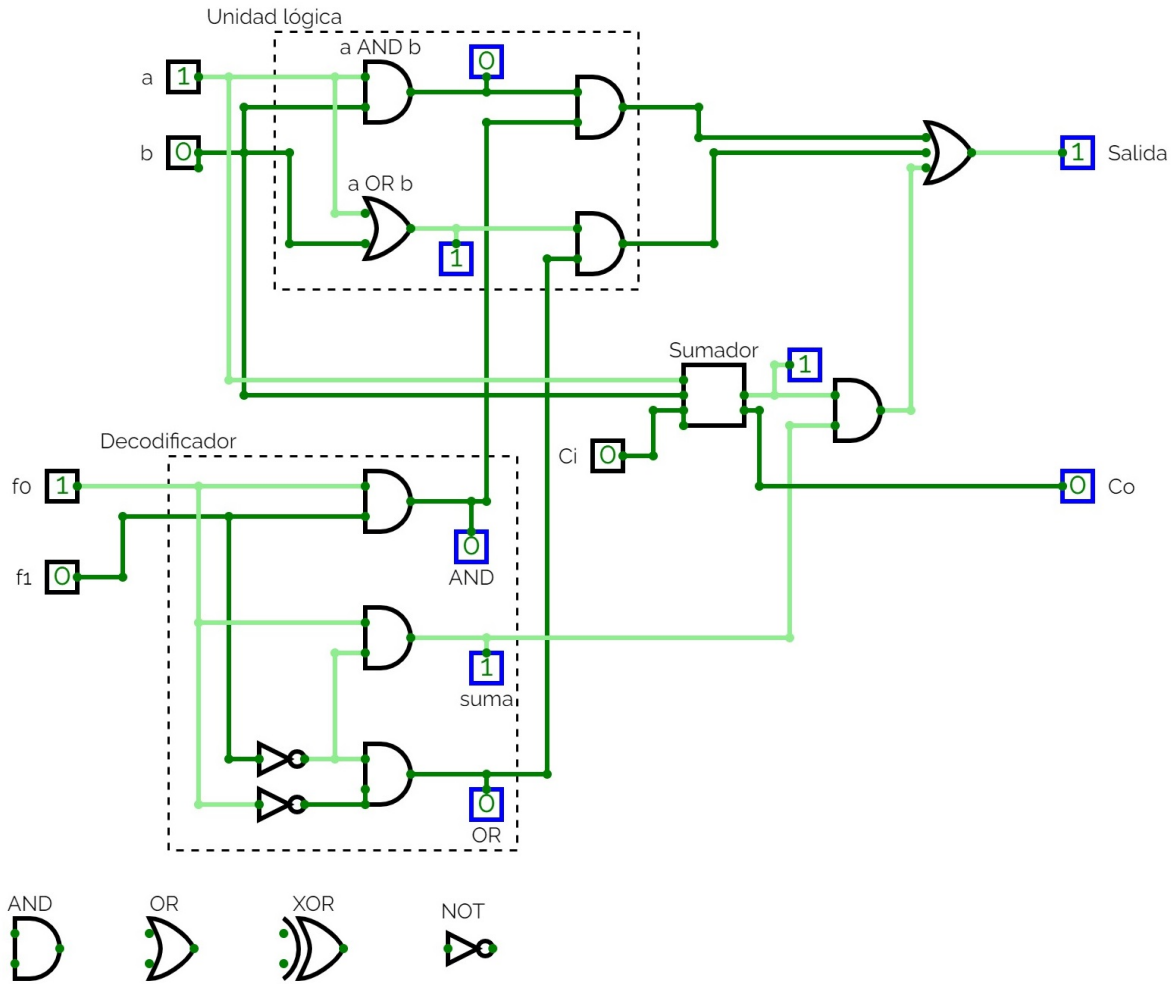
- La unidad aritmética/lógica (ALU) realiza una variedad de operaciones matemáticas y lógicas en una sola unidad.
- Por ejemplo, una ALU típica puede realizar operaciones de suma, resta, comparación de magnitudes y operaciones AND y OR, etc.
- La ALU forma el corazón de una computadora.
- Esquema de una ALU muy simple con entradas de n bits y solo 4 [4] operaciones:



[4] Con dos entradas de control se pueden codificar 2^2 operaciones.

La ALU

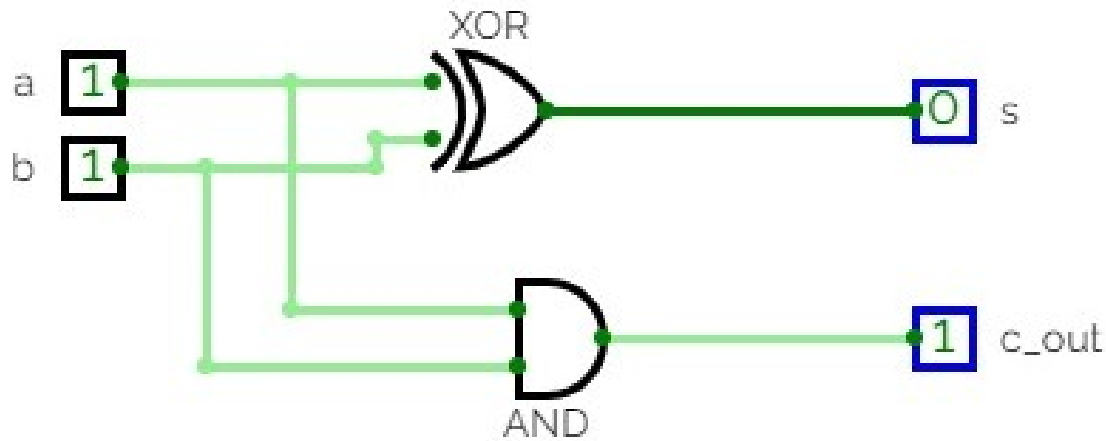
Esquema muy simplificado de una ALU muy simple que solo realiza dos operaciones lógicas (AND y OR) y la suma entre dos bits:



f0	f1	Operación
0	0	OR
0	1	------
1	0	Suma
1	1	AND

El semi-sumador (half-adder)

El semi-sumador permite sumar dos bits y además realiza el cálculo del acarreo hacia la etapa siguiente.

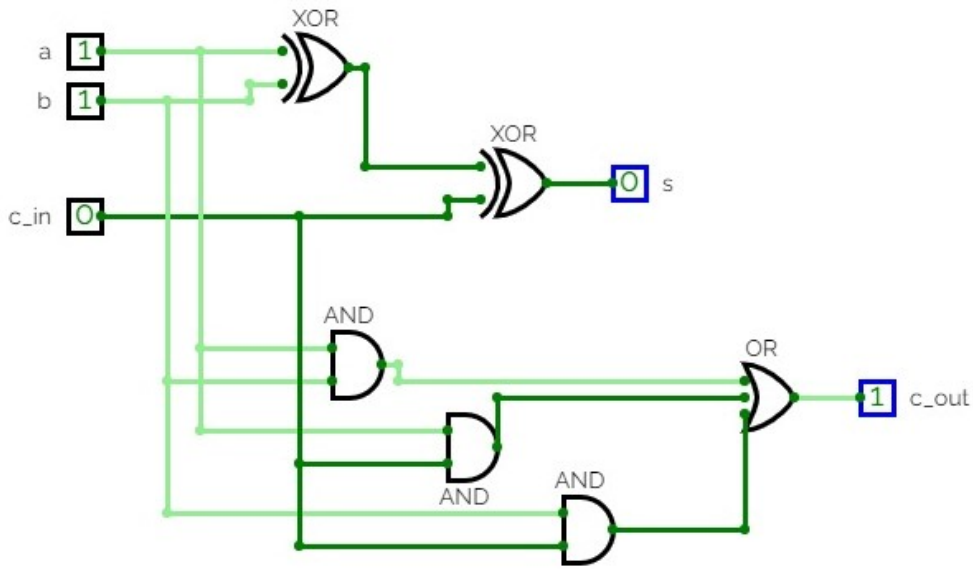


Expresiones lógicas:

s = a XOR b
c-out = a AND b

El sumador (full-adder)

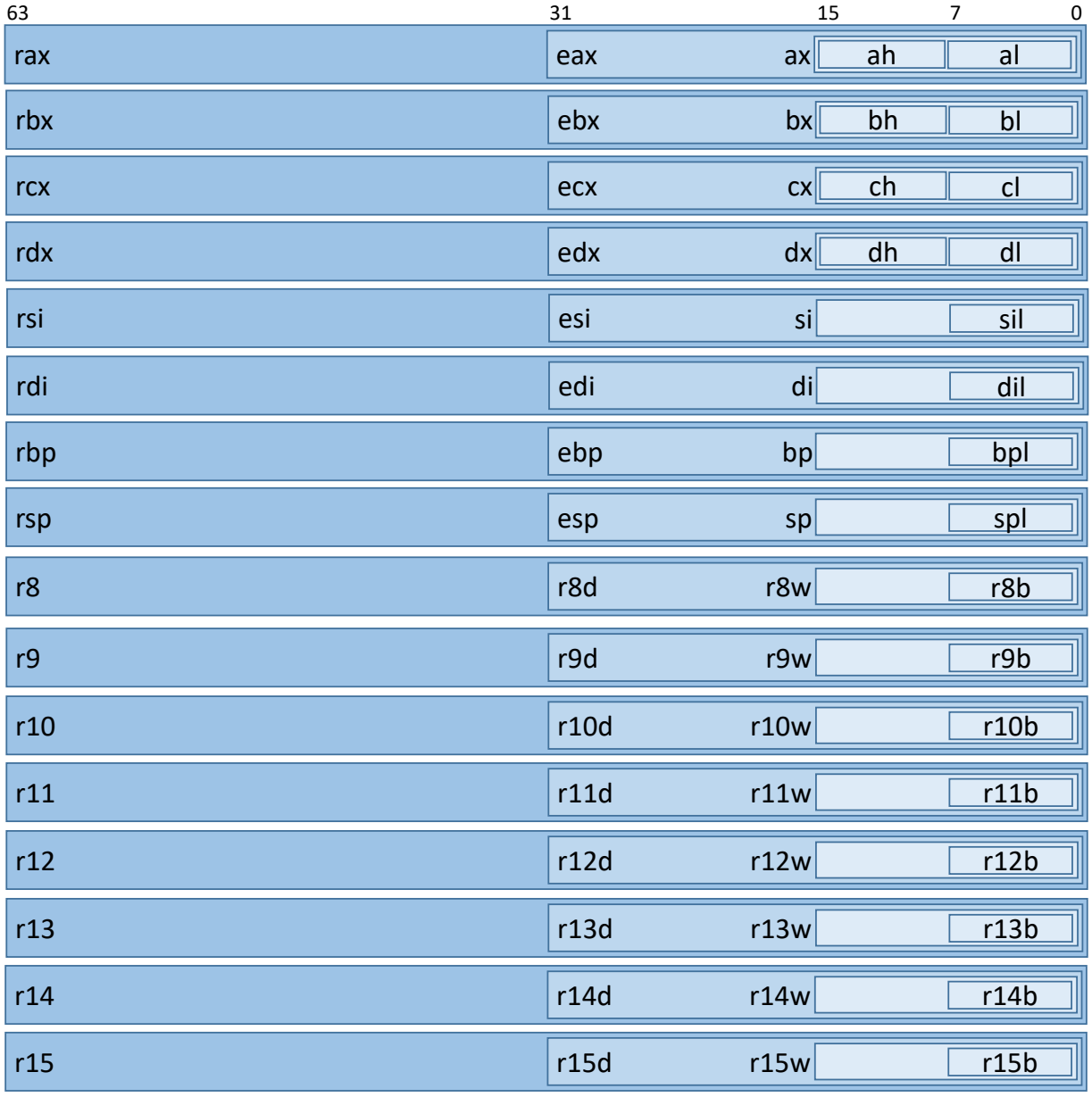
- El sumador permite sumar dos bits más el acarreo de la etapa anterior y calcular el acarreo hacia la etapa siguiente.
- Con un conjunto de N sumadores podemos sumar números de N bits.



Expresiones lógicas:

s = (a XOR b) XOR c-in
c-out = a AND b OR c-in(a XOR b)

Registros de propósito general



Lenguaje ensamblador de x86-64

- El hardware de la computadora sólo entiende 1 y 0, por lo que las instrucciones se codifican como números binarios en un formato llamado **lenguaje de máquina**.
- Los microprocesadores son sistemas digitales que leen y ejecutan instrucciones en lenguaje de máquina.
- Sin embargo, el lenguaje de máquina es muy tedioso y por lo tanto se prefiere representar las instrucciones en un formato simbólico llamado **lenguaje ensamblador**.

Lenguaje ensamblador de x86-64

Lenguaje de máquina	Lenguaje ensamblador
55	push %rbp
48 89 e5	mov %rsp,%rbp
8b 05 d9 2e 00 00	mov 0x2ed9(%rip),%eax
83 c0 02	add $0x2,%eax
89 05 d8 2e 00 00	mov %eax,0x2ed8(%rip)
8b 05 ca 2e 00 00	mov 0x2eca(%rip),%eax
01 c0	add %eax,%eax
89 05 ca 2e 00 00	mov %eax,0x2eca(%rip)
b8 00 00 00 00	mov $0x0,%eax
5d	pop %rbp
c3	ret

Lenguaje ensamblador de x86-64

Sintaxis AT&T

En general, las instrucciones se escriben como:

operadorS <operando origen>, <operando destino>

donde:

- **S** es el sufijo de tamaño.
- Los operandos pueden ser:
  - Valores inmediatos: $5, $0x4000, etc.
  - Registros: %rax, %ebx, etc.
  - Direcciones de memoria: 0x4000, a, etc.

Sufijos

- Las instrucciones llevan sufijos que indican el tamaño de los operandos.
- Es recomendable poner siempre el sufijo correspondiente.

Sufijo	**Denominación**	**Tamaño (bytes)**	**Equivalente en C**
b	*Byte*	1	char
w	*Word*	2	short
l	*Double word (o long)*	4	int
q	*Quad word*	8	long int
t	*Ten*	10	————
s	*Single precision float*	4	float
d	*Double precision float*	8	double

Instrucción MOV

Forma general:

```
movS <operando origen>, <operando destino>
```

Diferentes formas que puede tomar la instrucción:

```
movS <registro>, <registro>
movS <memoria>, <registro>
movS <registro>, <memoria>
movS <valor immediato>, <memoria>
movS <valor immediato>, <registro>
```

Ejemplos:

```
movq %rax, %rbx
movl a, %ebx
movb %bl, a
movw $0xff, a
movl $45, %eax
```

Primer programa muy básico

```
.global main
main:
    movq $0, %rax    # Comentarios aquí!
    ret
```

Lo podemos compilar como:

```
gcc -o ejemplo ejemplo.c
```

y ejecutar como

```
./ejemplo
```

Este programa es equivalente al siguiente programa en C:

```
int main(){
    return 0;
}
```

Algunas instrucciones básicas

- Instrucción ADD

  `addS <operando fuente>, <operando destino>`

  Realiza la suma:

  `<operando destino=operando fuente + operando destino>`

- Instrucción SUB

  `subS <operando fuente>, <operando destino>`

  Realiza la resta:
  `operando destino = operando destino - operando fuente.`

- Instrucción INC

  `incq %rax`

  es equivalente a `addq $1, %rax`

- Instrucción XOR

  `xorS <operando fuente>, <operando destino>`

  Realiza la operación lógica `xor` bit a bit.

- Instrucción RET

  `ret`

  Esta instrucción se utiliza para hacer un retorno de subrutina.

Trabajando con subregistros

Ejemplo 1: (`subregistros.s`)

```
.global main
main:
    movq $-1, %rax
    movb $0, %al
    movw $0, %ax
    movl $0, %eax
    ret
```

Ejemplo 2: (`suma_subregistros.s`)

```
.global main
main:
    movb $-1, %al
    addb $1, %al
    movl $-1, %eax
    addl $1, %eax
    ret
```