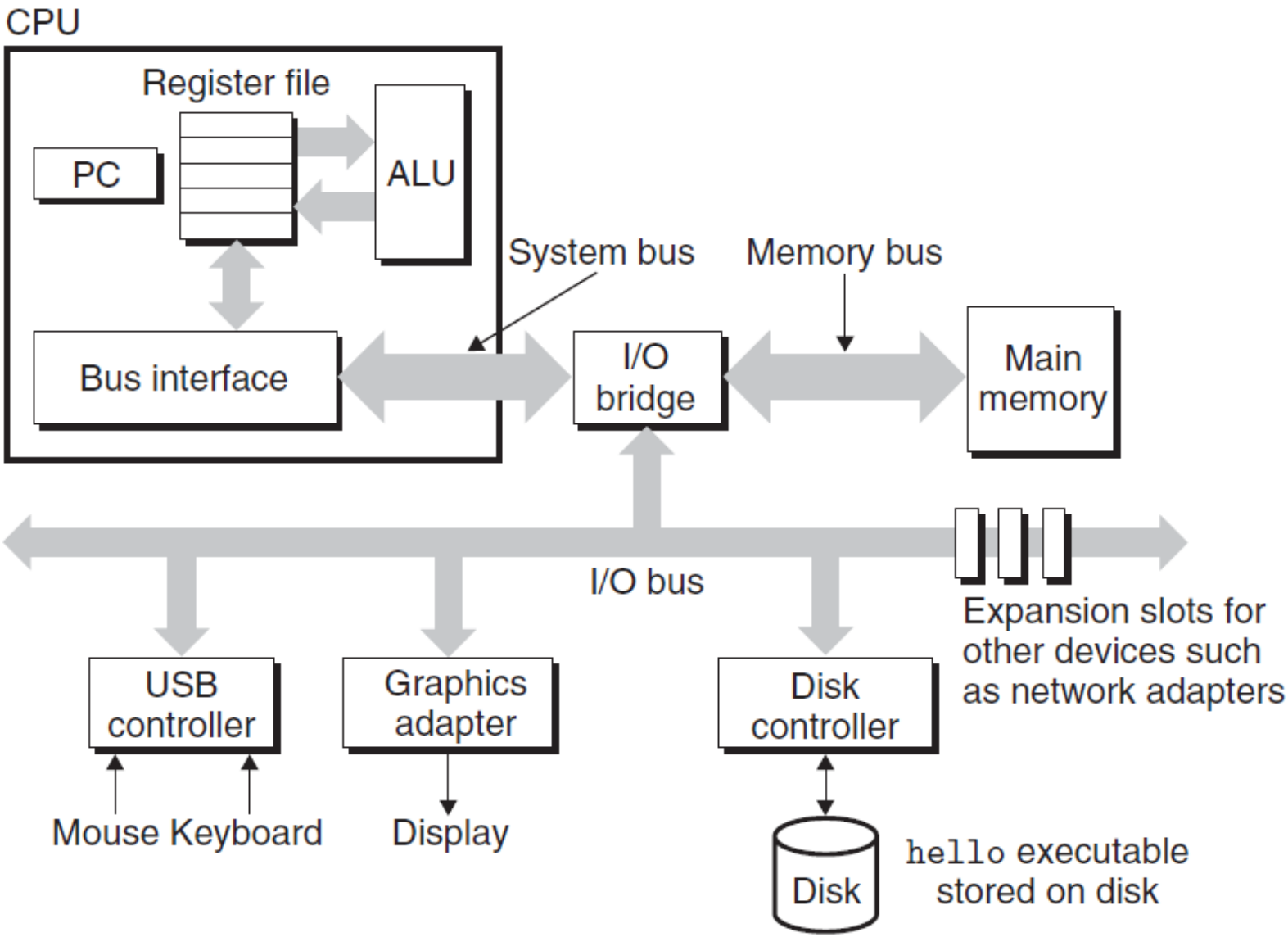
La arquitectura x86-64

- 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.

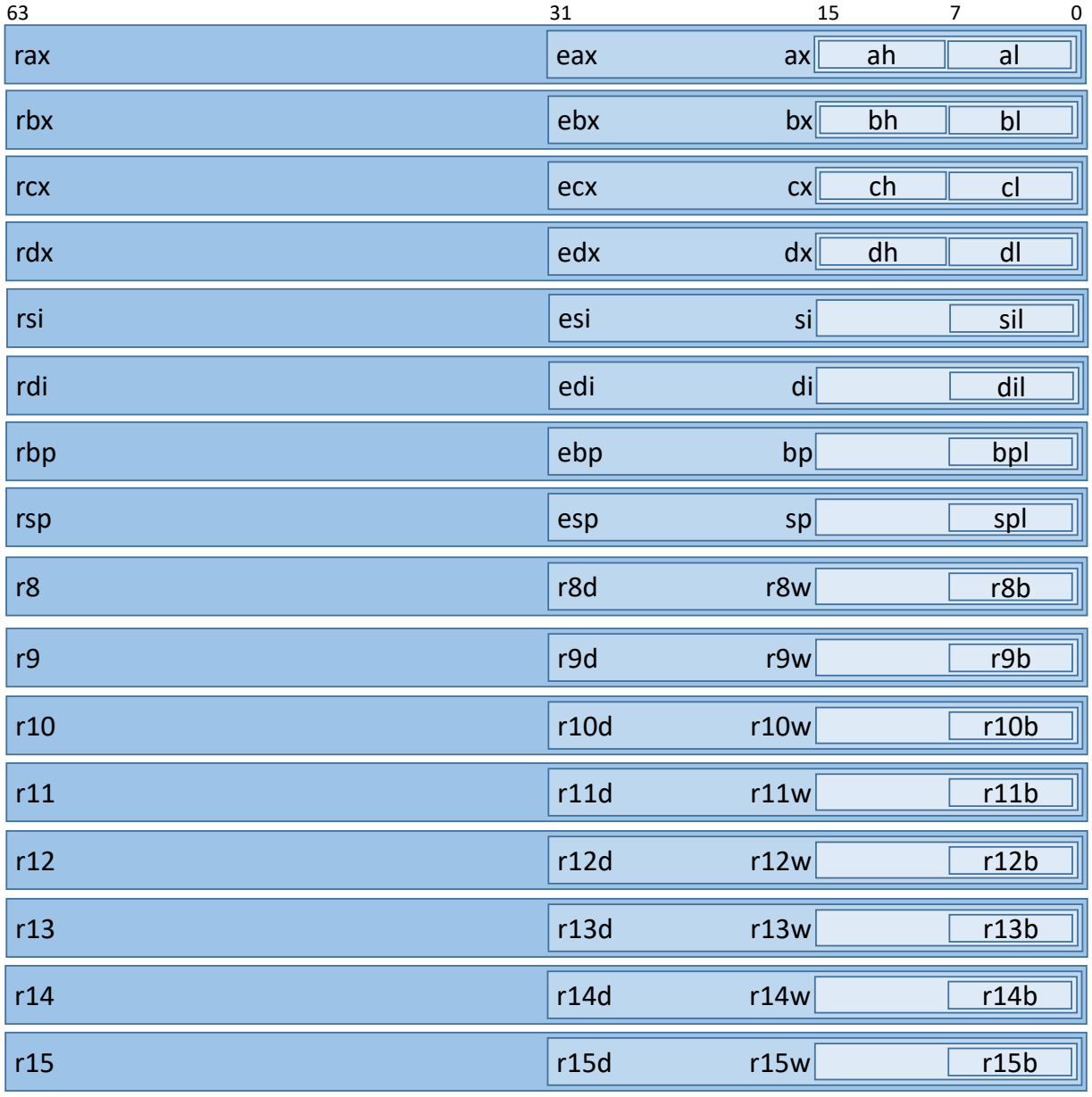
La arquitectura x86-64

- La arquitectura x86-64 (AMD64 o Intel 64) de 64 bits da un soporte mucho mayor al espacio de direcciones virtuales y físicas.
- Proporciona registros y buses de datos y direcciones de 64 bits.
- Aunque posee registros de 64 bits también permite operaciones con valores de 256, 128, 32, 16 y 8 bits.

Organización del hardware de una computadora típica



Registros de propósito general



Lenguaje ensamblador de x86-64

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, %rbx, etc.
  - Direcciones de memoria: 0x4000, etc.

Sufijos

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>
```

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
```