

# Conceptos de arquitectura de computadores

Adaptación de múltiples referencias bibliográficas

Juan Felipe Muñoz Fernández

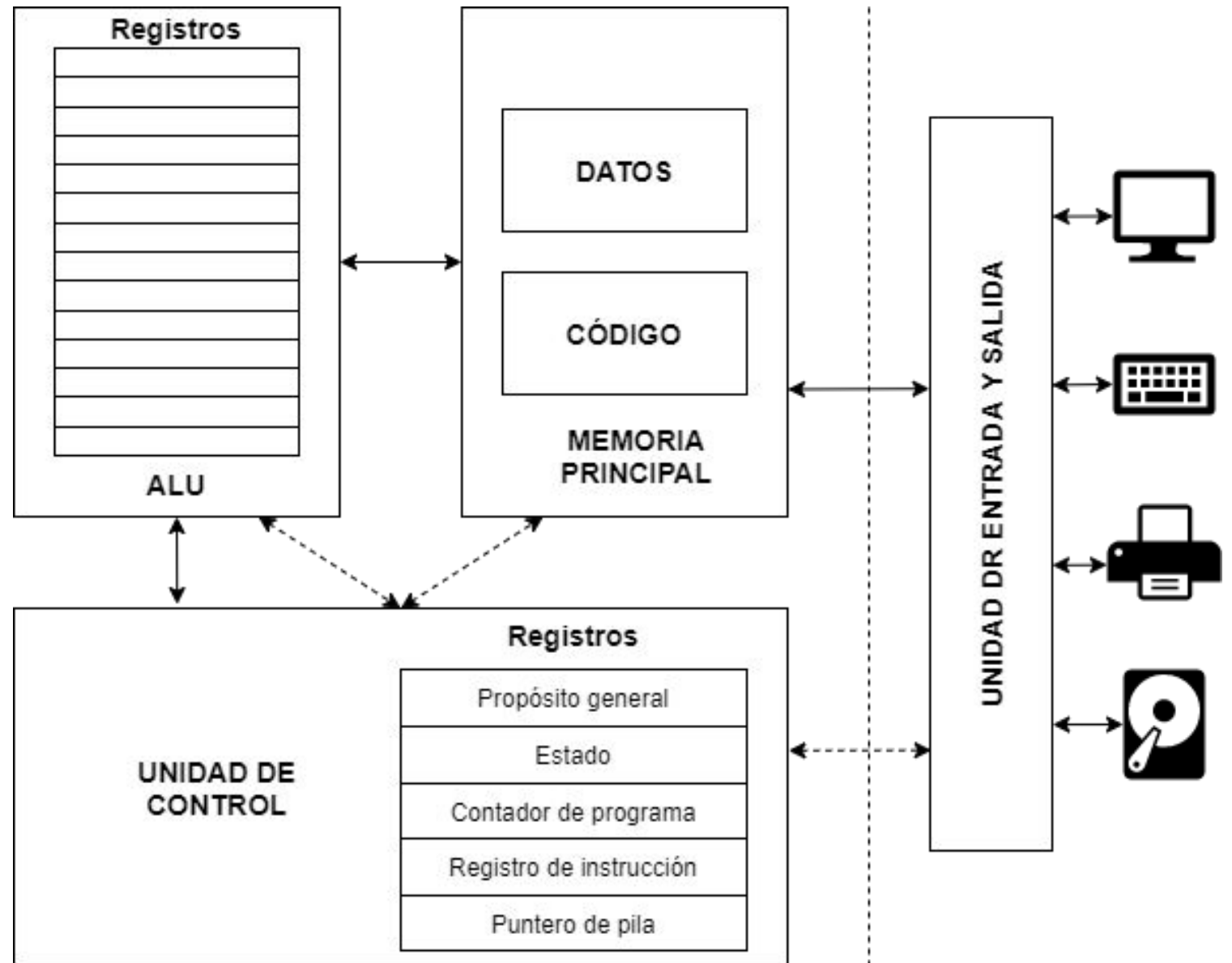
# Arquitectura John von Neumman



John von Neumman  
Fuente: Wikipedia

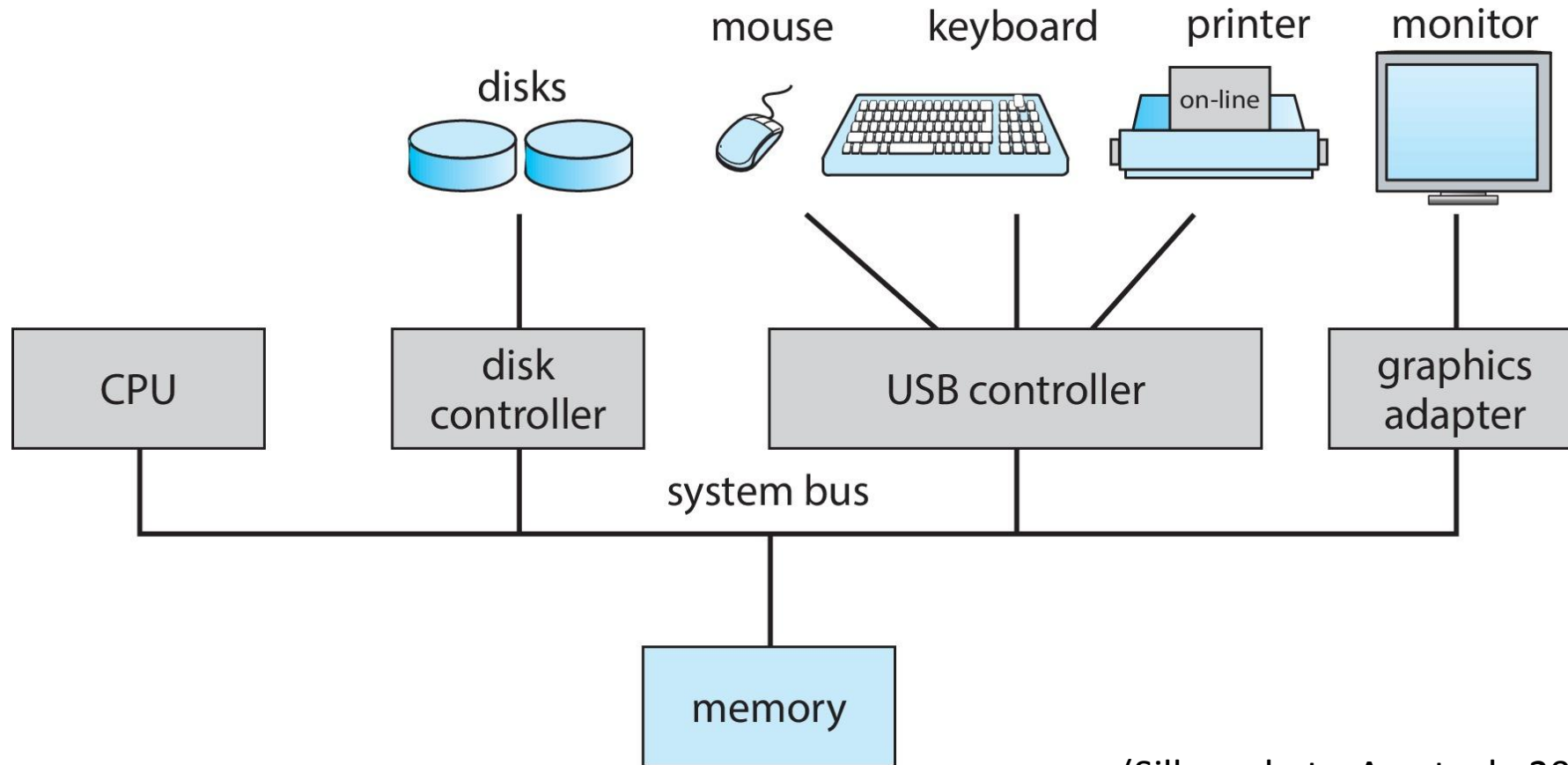
- Un computador se compone de:
  - Unidad de control
  - Memoria principal
  - Los dispositivos de E/S o periféricos
  - Unidad Aritmético – Lógica
- Todos los elementos conectados a través de un bus común
  - Bus del sistema o *System bus*
    - Bus de direcciones
    - Bus de datos
    - Bus de control

# Arquitectura John von Neumman

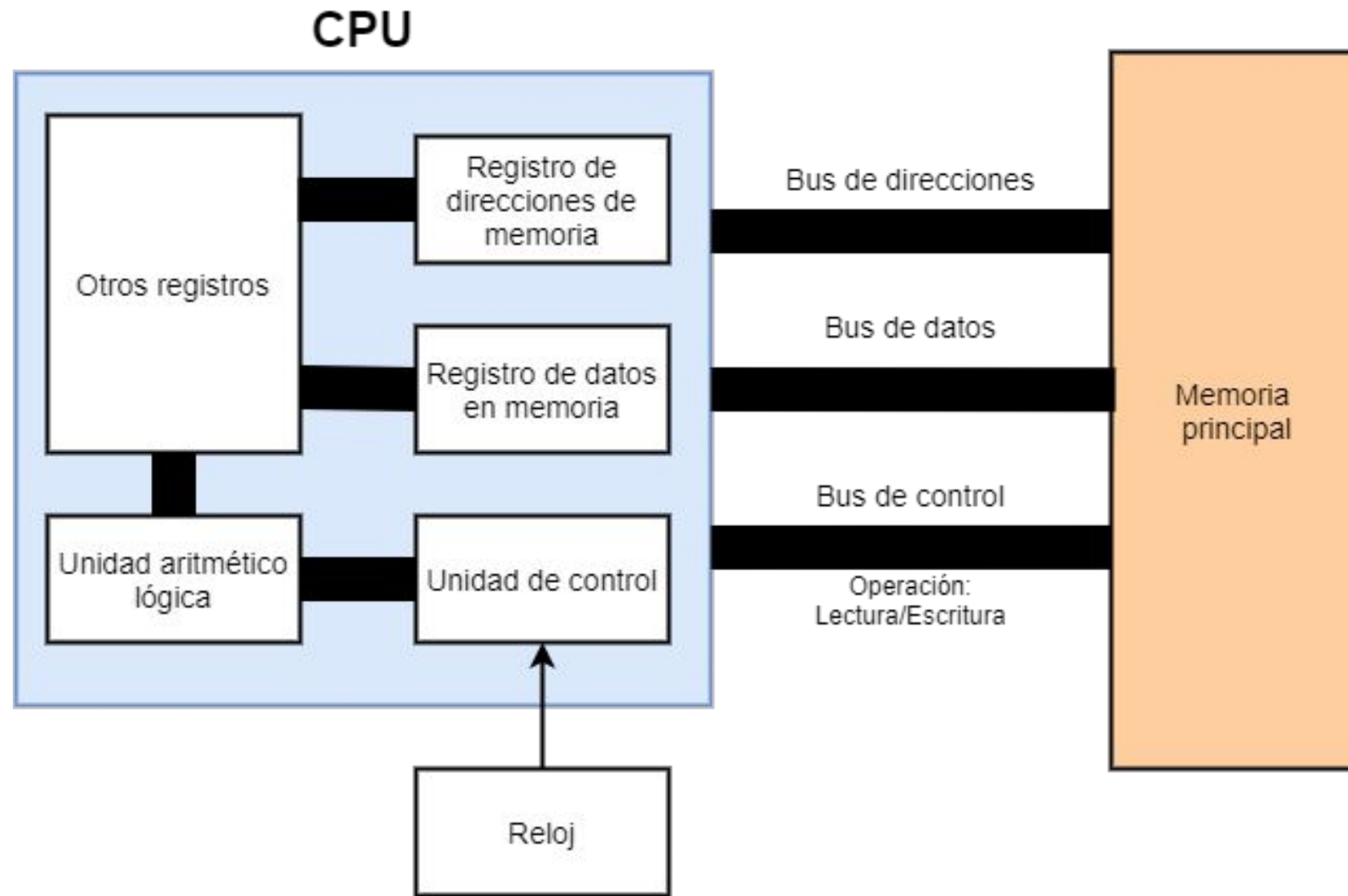


Con modificaciones tomada de:  
(Carretero Pérez, J. et. al., 2001)

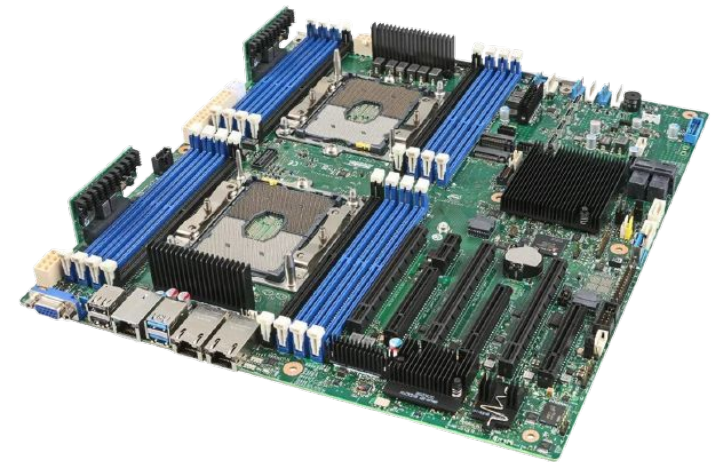
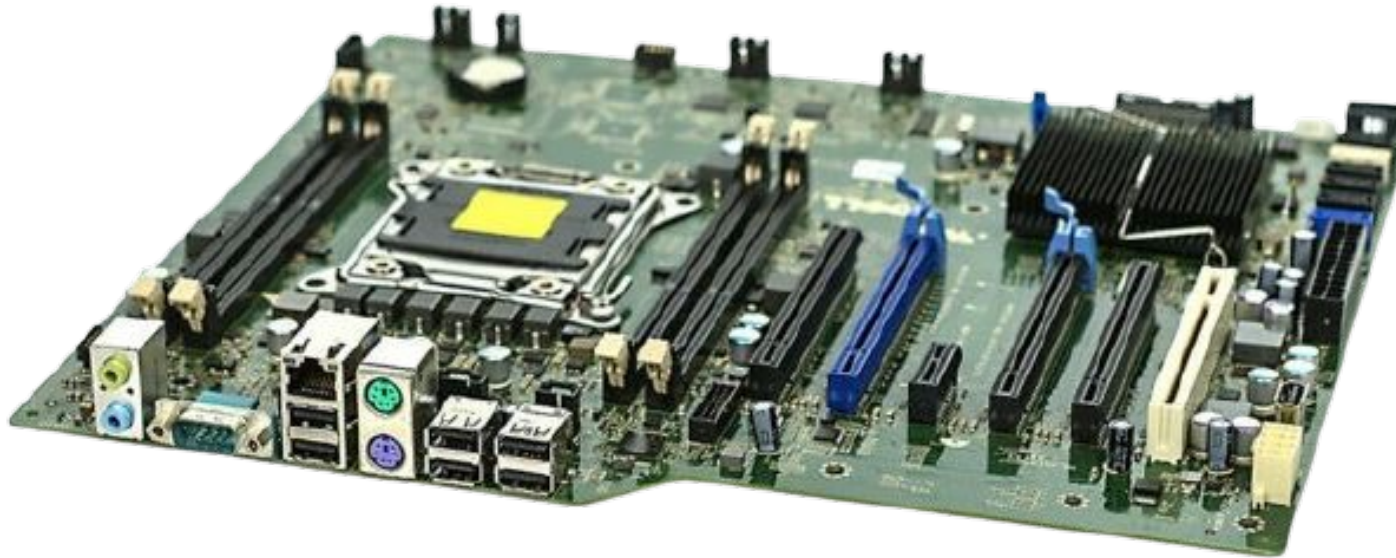
# Arquitectura John von Neumman



(Silberschatz, A., et. al., 2018)



# Arquitectura Jhon von Neumman

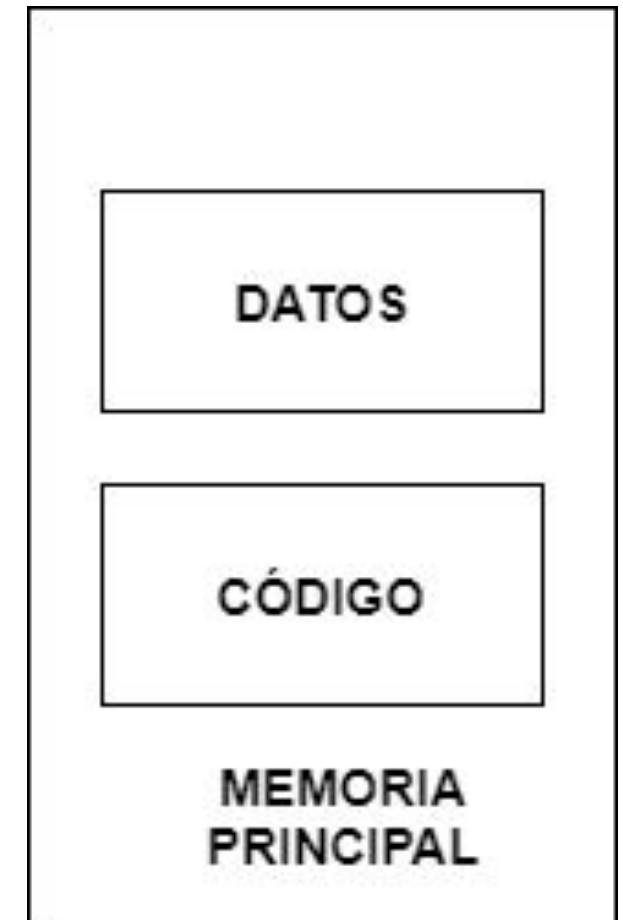


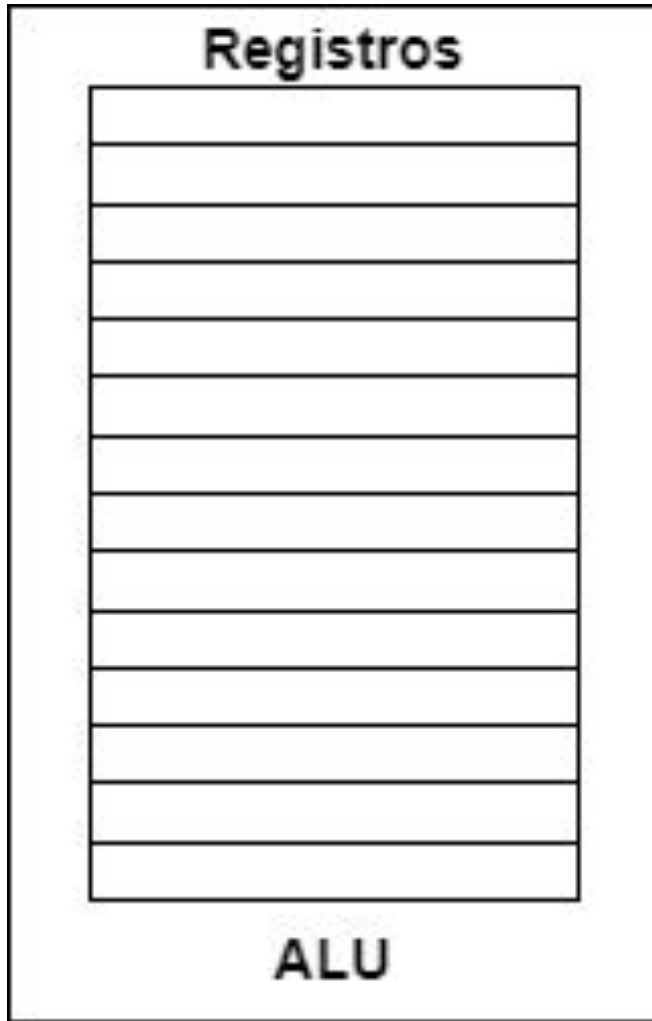
# Memoria principal

Demo x64dbg  
Demo Visual  
Studio

```
#include <stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
}
```

- Compuesta por
  - RAM (*Random Access Memory*)
  - ROM (*Read Only Memory*)
- Contiene
  - Datos de los programas
  - Código de los programas
  - Resultados de las operaciones
- Se accede a través de las direcciones
  - Estructura de “celdas”
  - Cada celda (byte) tiene una dirección
  - Usualmente palabras de 4 u 8 bytes
  - Un arreglo de muchos bytes





# Unidad Aritmético – Lógica

- Realiza operaciones aritméticas y lógicas
  - Uno o dos operandos
  - Operandos almacenados en registros o en memoria principal
  - Resultados se almacenan en registros o en memoria principal

Demo x64dbg  
Demo Visual  
Studio

```
#include <stdio.h>
int main()
{
    short a = 1;
    short r = 0;
    r = a + 3;
    printf("%d", r);
    return 0;
}
```



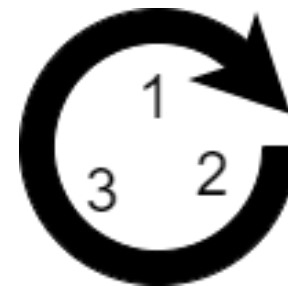
# Unidad de control

- Funciones

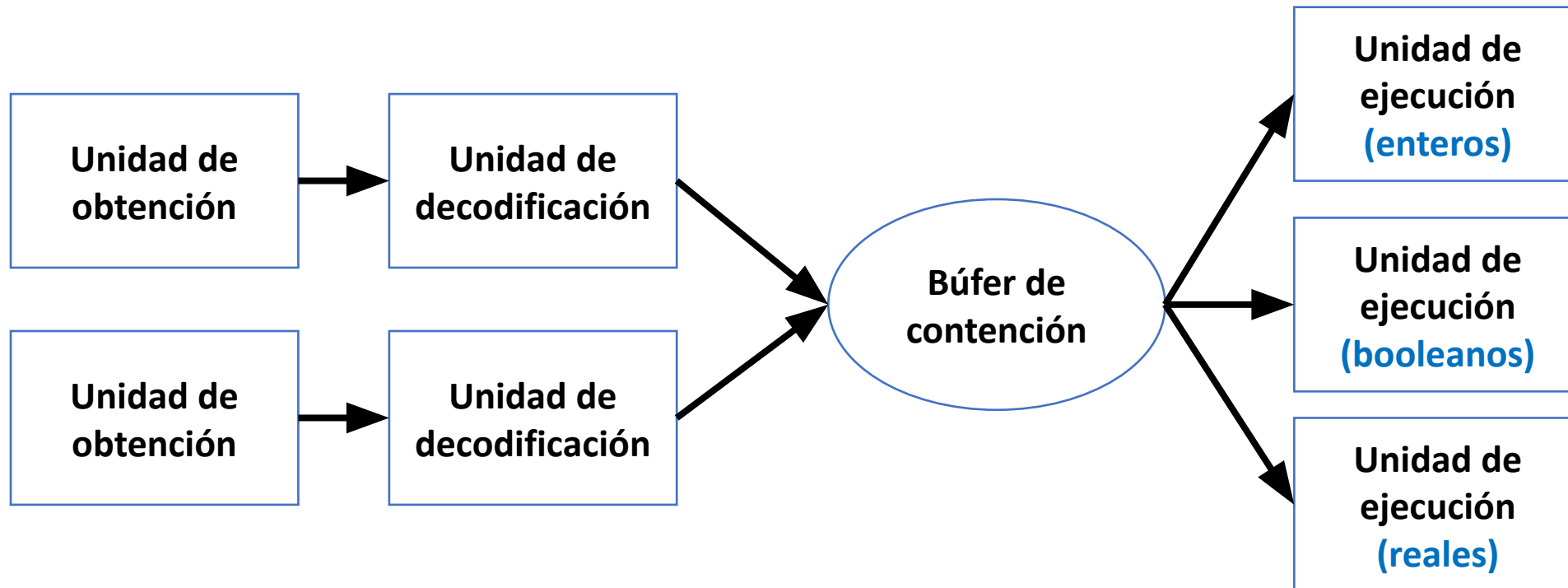
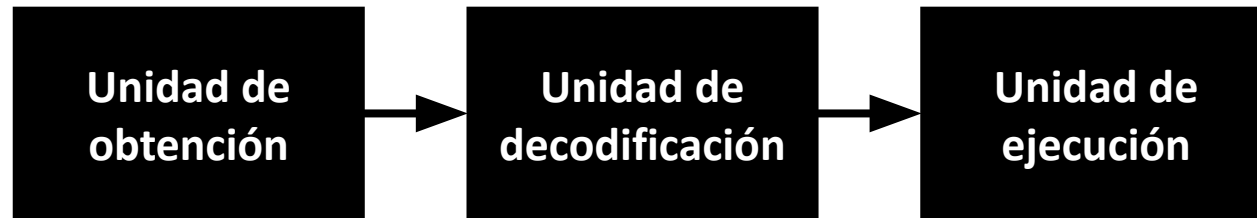
- Lee de memoria instrucciones que forman el programa
- Interpreta instrucción leída
- Lee datos referenciados por cada instrucción
- Ejecuta cada instrucción
- Almacena el resultado de operaciones
- Sincroniza todo el sistema

- Se mantiene en una secuencia lineal (instrucciones consecutivas) de alta velocidad

1. Lectura memoria principal de la instrucción apuntada por *Instruction Pointer*
2. Incremento del *Instruction Pointer*
3. Ejecución de la instrucción



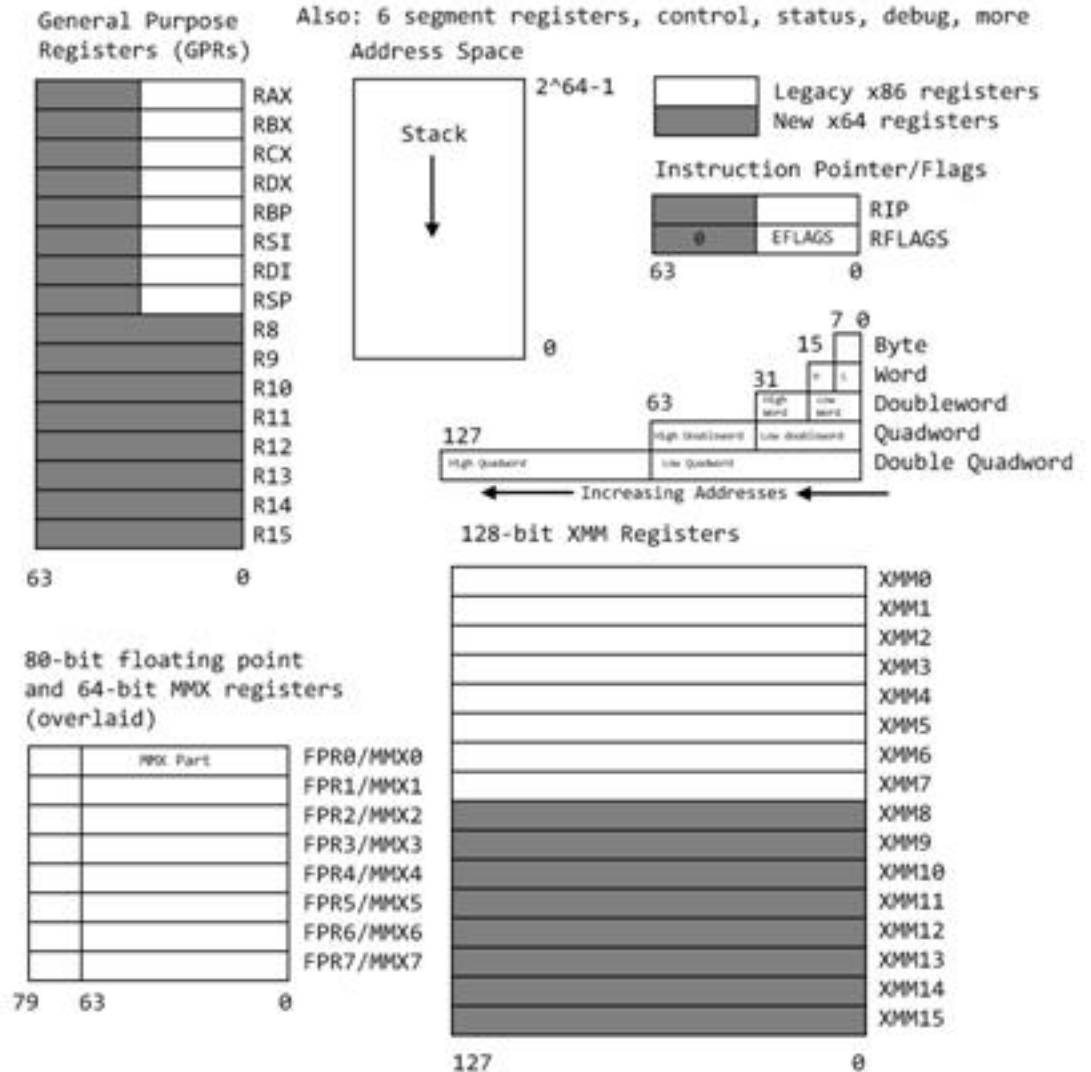
# Ciclo obtención, decodificación, ejecución



# Unidad de control

```
#include <stdio.h>
int main()
{
    short a;
    short b = 2;
    a = 1;
    if (a > b)
        printf("True");
    else
        printf("False");
}
```

Demo x64dbg  
Demo Visual  
Studio



# Registros en CPU

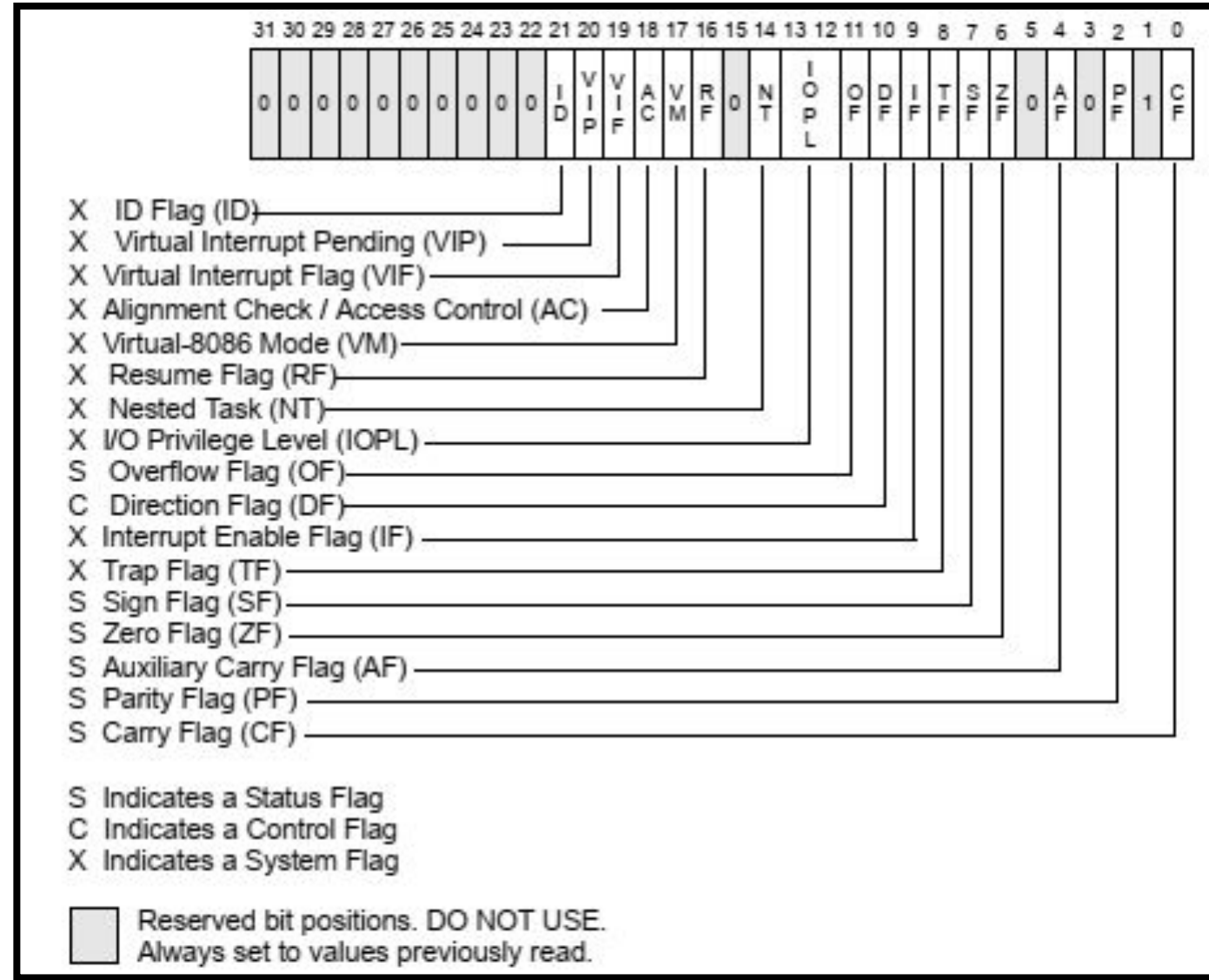
- Registros de direcciones de memoria
  - *MAR (Memory Address Registers)*
  - Mantienen la dirección de una ubicación en memoria
- Registros de datos en memoria
  - *MDR (Memory Data Registers)*
  - Mantienen los datos que se acaban de leer o escribir a la memoria
- Contador del programa
  - Mantiene la dirección de la siguiente instrucción a ejecutar
- Registro de instrucción
  - Mantiene la dirección de la instrucción que está siendo ejecutada actualmente
- Registros de uso general
  - Usados por los programadores

# Registros en CPU x64

- Registros de propósito general
  - RAX, RBX, RCX, RDX
  - +8 adicionales: R8 a R15
- Registros apuntadores
  - RBP, RSP
- Registros índice
  - RSI, RDI
- Registros de segmentos
  - CS, DS, ES, SS, FS, GS
- Apuntador de instrucciones
  - RIP
  - CS:RIP □ Próxima instrucción a ejecutar
- Registro de estado (R/EFLAGS)
  - Mantiene el registro del estado de la ejecución de instrucciones

# Registro de estado

- No todos los bits son modificables por el programador.
- Algunos requieren **ejecución privilegiada de instrucciones**.
- Mantiene información del estado del sistema



# Registros CPU x64

---

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
RAX	EAX	AX	AL
RBX	EBX	BX	BL
RCX	ECX	CX	CL
RDY	EDX	DX	DL
RSI	ESI	SI	SIL
RDI	EDI	DI	DIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL

---

Fuente: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

# Registros CPU x64

---

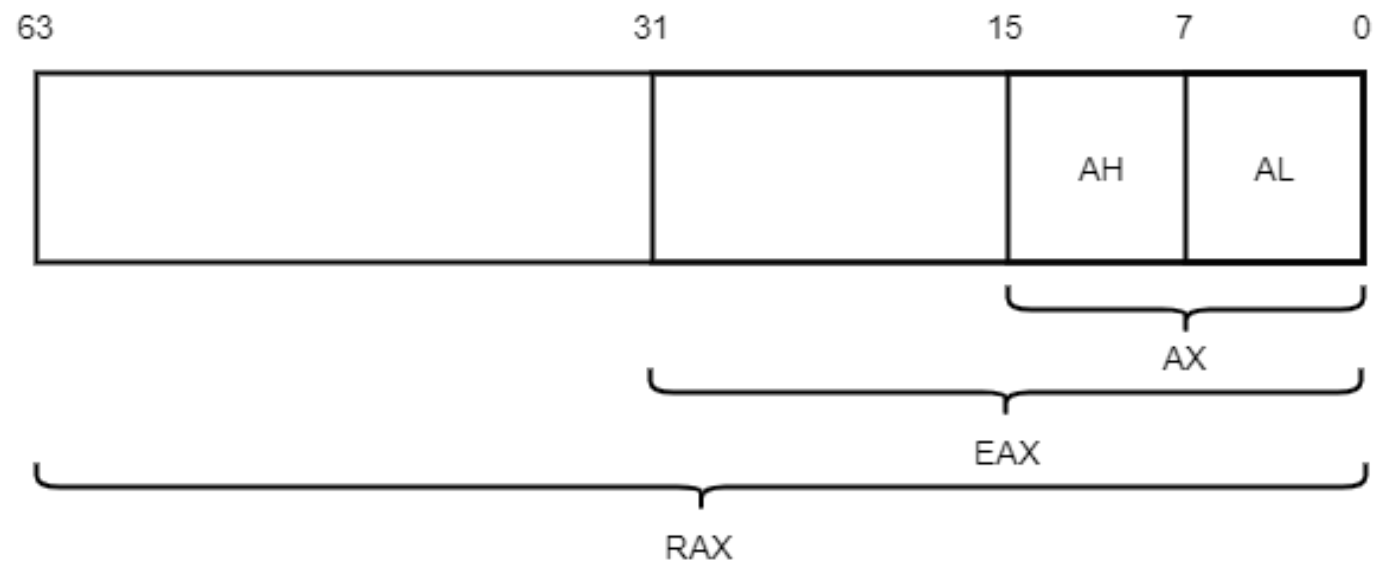
64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
R8	R8D	R8W	R8B
R9	R9D	R9W	R9B
R10	R10D	R10W	R10B
R11	R11D	R11W	R11B
R12	R12D	R12W	R12B
R13	R13D	R13W	R13B
R14	R14D	R14W	R14B
R15	R15D	R15W	R15B

---

Fuente: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

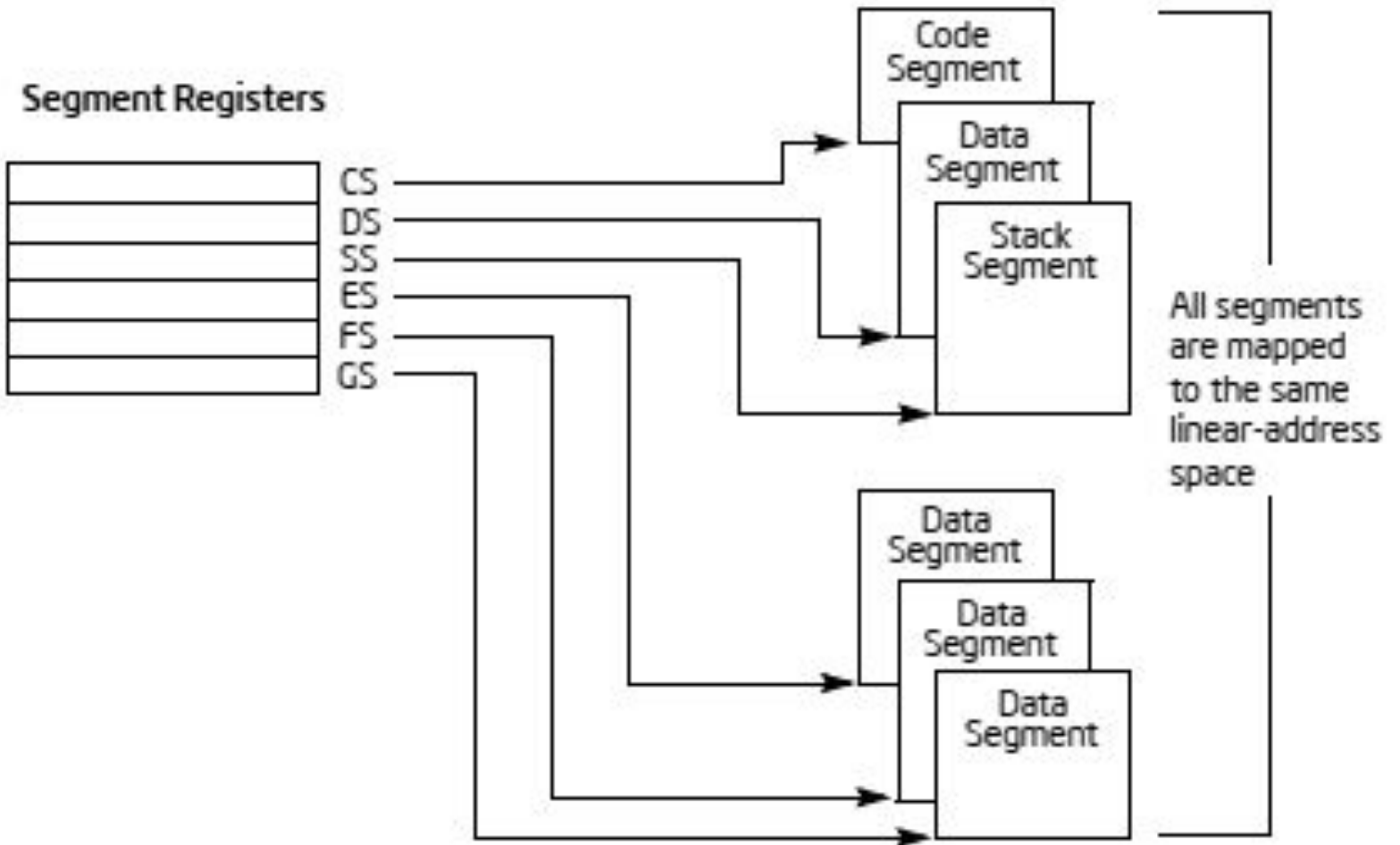


# Registros CPU x64



# Registros de segmentos

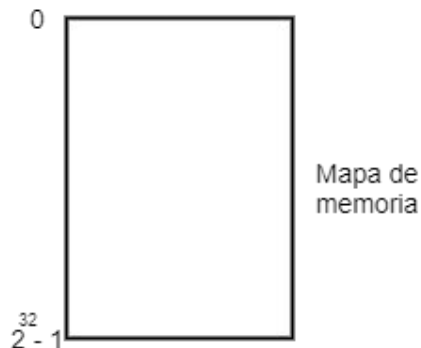
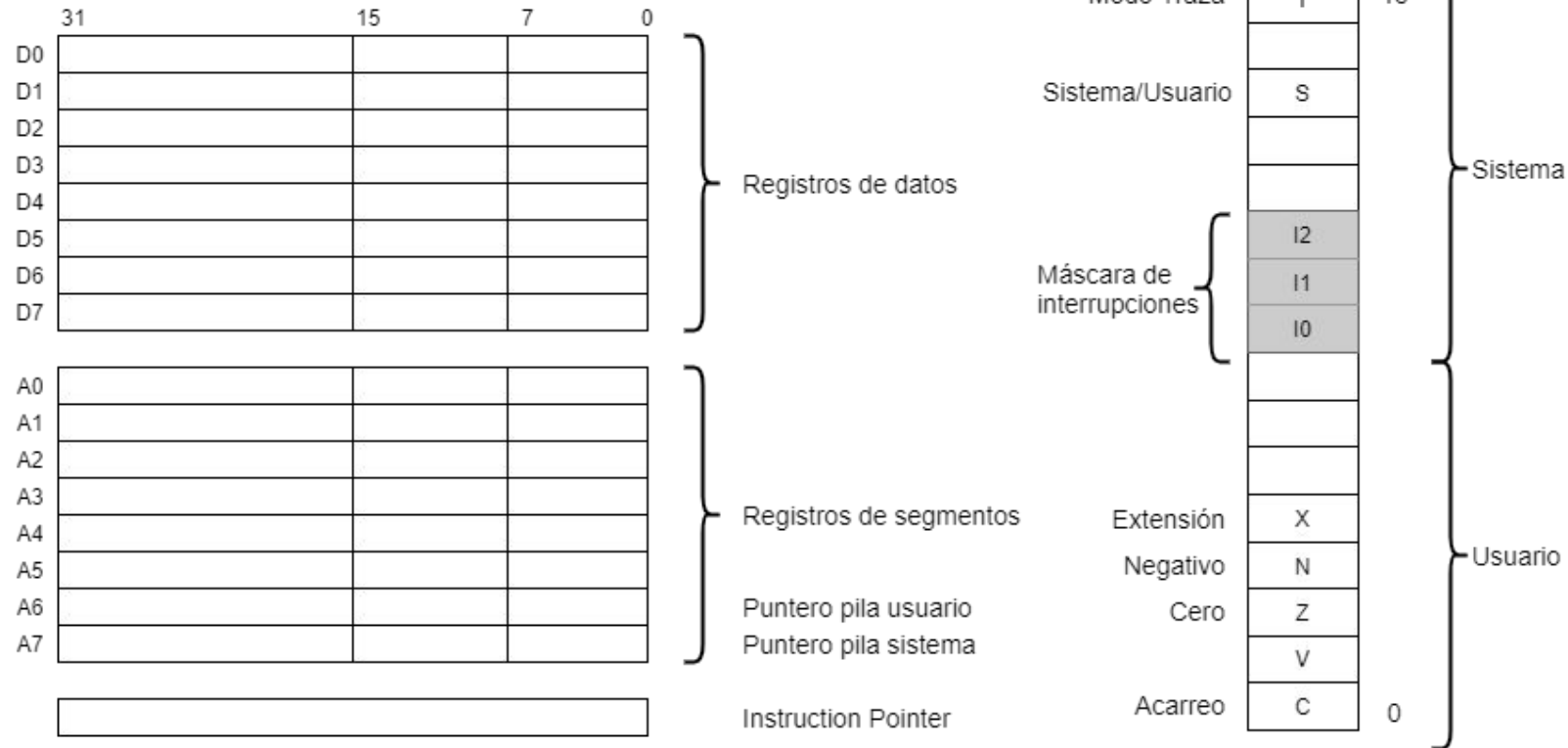
Fuente: Intel® 64 and IA-32  
Architectures. Software Developer's  
Manual. Volume 1: Basic Architecture



# Unidad de entrada/salida

- Realiza la transferencia de información entre:
  - Memoria principal y periféricos
  - Registros y periféricos
  - Acceso por interrupciones de E/S
- También se puede acceder a través de la CPU a los dispositivos
  - E/S programada
  - Procesador ejecuta un programa de E/S
  - No hay concurrencia entre procesador y E/S
- También se puede hacer de forma independiente (DMA)
  - *Direct Memory Access*
  - Puede existir concurrencia de procesos
  - Responsable el controlador del dispositivo

# Modelo de programación



Con modificaciones tomada de:  
(Carretero Pérez, J. et. al., 2001)

# Modelo de programación

- Almacenamiento
  - **Visibles a las instrucciones**
    - Registros generales (RAX, EAX, RBX, EBX,...,etc.)
    - *Instruction Pointer* (CS: IP)
    - Puntero de pila (SS)
    - Estado (R/EFLAGS)
    - Memoria principal
    - Mapa de E/S
- Secuencia de funcionamiento
  - Define cómo se ejecutan las instrucciones

# Modelo de programación

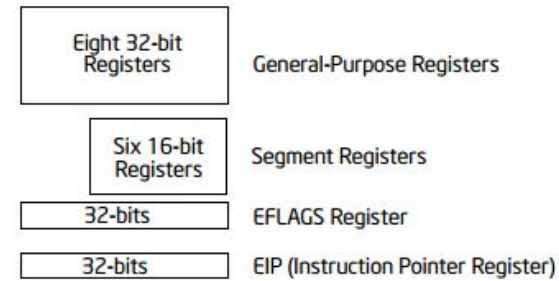
- Juego de instrucciones
  - Operaciones que es capaz de hacer la CPU
  - Modos de direccionamiento
    - Destino, fuente: `MOV AX, BX`
    - Modalidad inmediata: `MOV AX, 5`
    - Modalidad directa: `MOV AX, [200H]`
    - Modalidad indirecta: `MOV AX, SI:[200H]. MOV AX, DI:[200H]`
    - Modalidad base relativa: `MOV AX, [BP]. MOV AX, [BP+SI+5]`

# Modelo de programación Intel x86

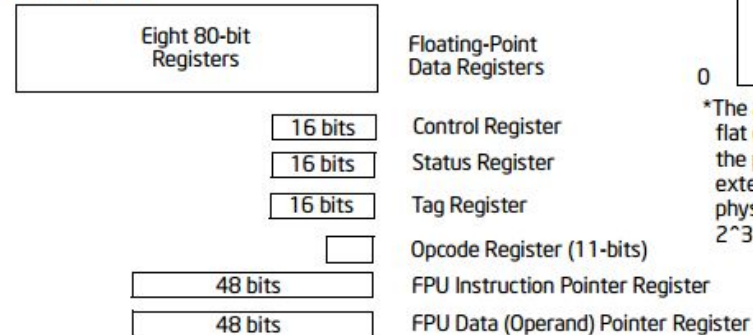
Fuente: Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture

Operaciones SIMD  
Una instrucción múltiples datos  
Paralelismo a nivel de datos

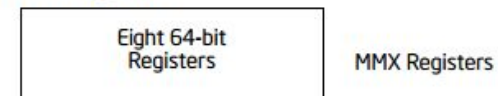
## Basic Program Execution Registers



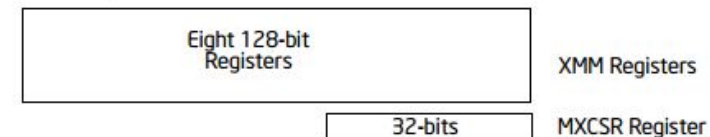
## FPU Registers



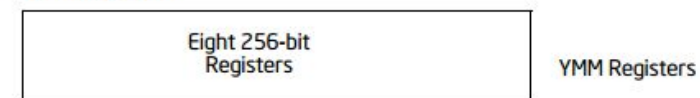
## MMX Registers



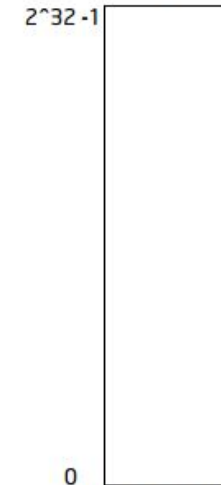
## XMM Registers



## YMM Registers

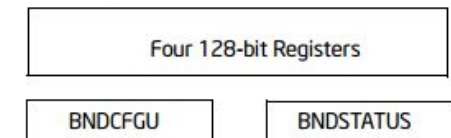


## Address Space\*



\*The address space can be flat or segmented. Using the physical address extension mechanism, a physical address space of  $2^{36} - 1$  can be addressed.

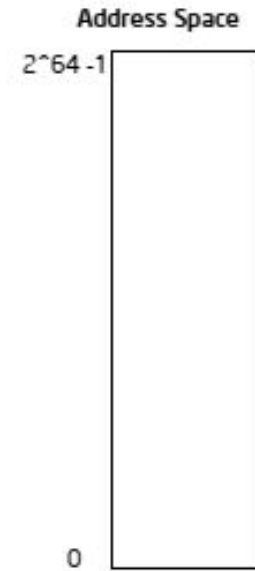
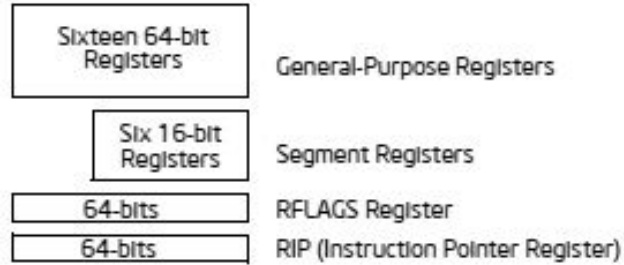
## Bounds Registers



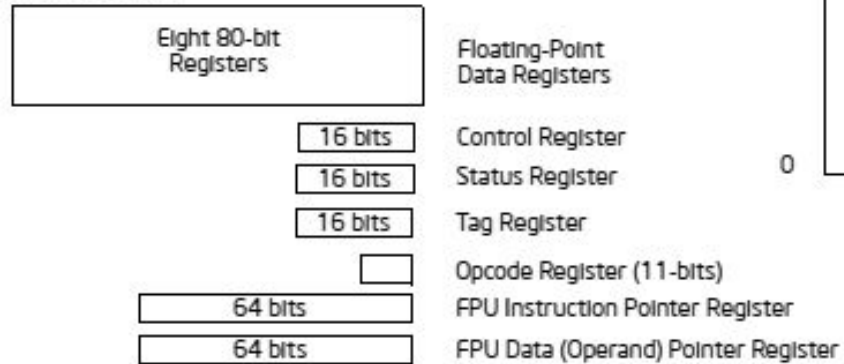
# Modelo de programación Intel x64

Fuente: Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture

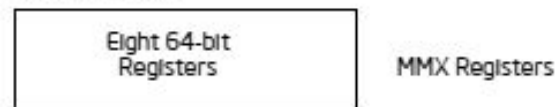
## Basic Program Execution Registers



## FPU Registers



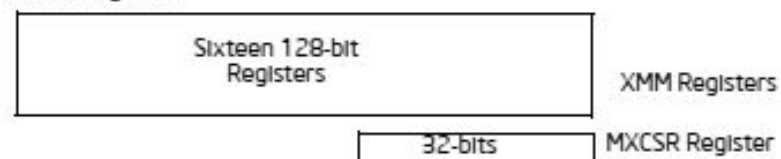
## MMX Registers



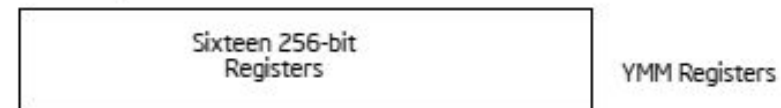
## Bounds Registers



## XMM Registers



## YMM Registers





# Modelo de programación

- Lecturas sugeridas
  - Intel® 64 and IA-32 Architectures. Software Developer's Manual. Volume 1: Basic Architecture
- Sitio interesante con documentos sobre arquitectura de procesadores Intel
  - <https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>

# Niveles de ejecución

- **Nivel de usuario (ring 3)**

- Disponible un subconjunto de todo el juego de instrucciones
- Demás instrucciones prohibidas
- Prohibido el acceso a ciertos registros
- Prohibido al acceso a determinadas zonas del mapa de memoria y del mapa de E/S.

- **Nivel de núcleo (ring 0)**

- No hay restricciones
- Kernel del sistema operativo
- Controladores de algunos dispositivos
- Depuradores tipo **SoftICE** y **WinDbg**

# SoftICE y WinDbg

```
0010:00000000 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000010 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000020 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000030 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000040 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000050 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000060 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??
0010:00000070 ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??

0008:B1F0A293 STI
==> B1F0A294 JMP ↑B1F0A294 (JUMP
0008:B1F0A296 INT
0008:B1F0A298 IRE
0008:B1F0A299 PUS
0008:B1F0A29A PUS
0008:B1F0A29B MOV B60
0008:B1F0A2A0 MOV [EBX],203D5344
0008:B1F0A2A6 ADD
0008:B1F0A2A9 MOV
0008:B1F0A2AC CAL
0008:B1F0A2B1 MOV
0008:B1F0A2B7 ADD
(PASSIVE)-KTEB(81B
: WD 8
:SOFTICE1.FREE.FR
Invalid command
```

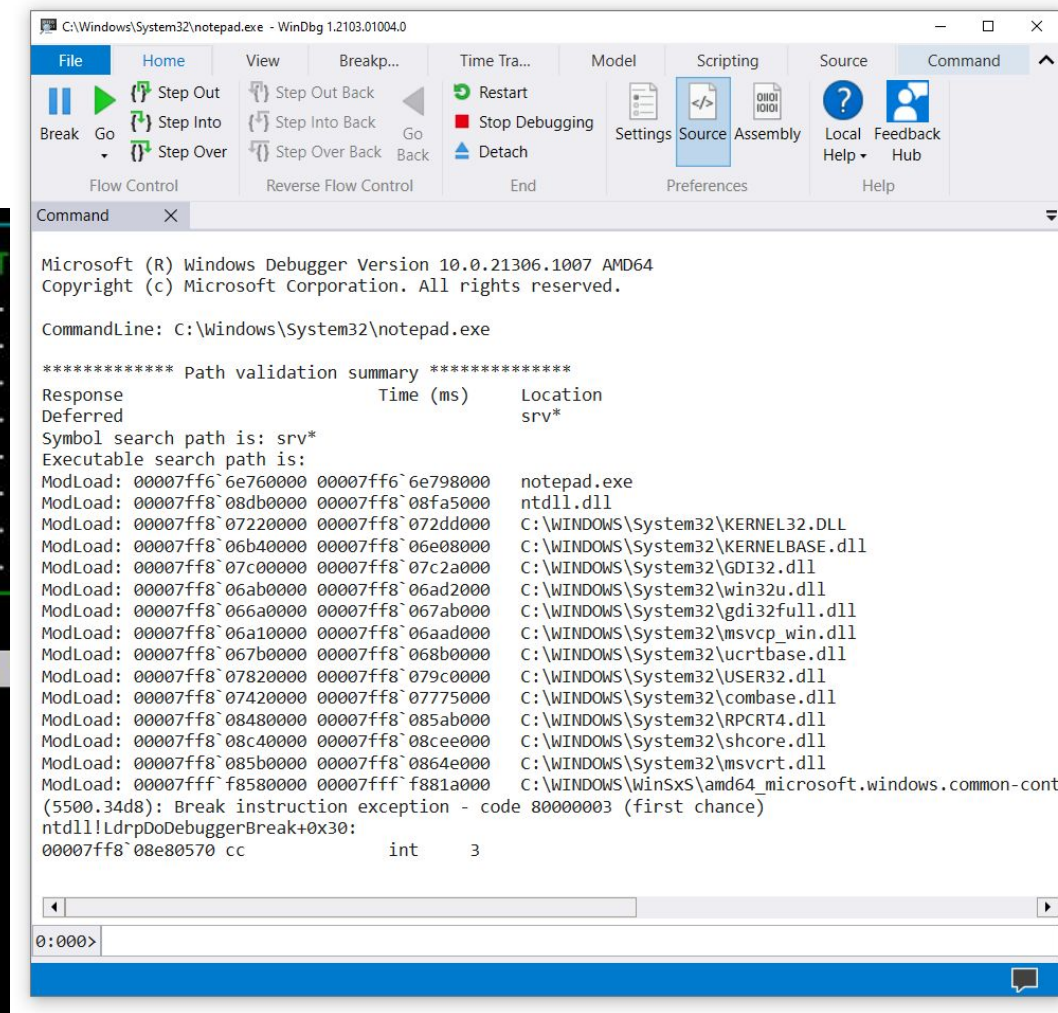
- Copy
- Paste
- Copy&Paste
- Display
- Un-Assemble
- What
- Prev
- Reip
- Add Watch
- Break On Text
- Name

B60 [EBX],203D5344

28]

[EBX+04],0020

NTice!.text+2F13

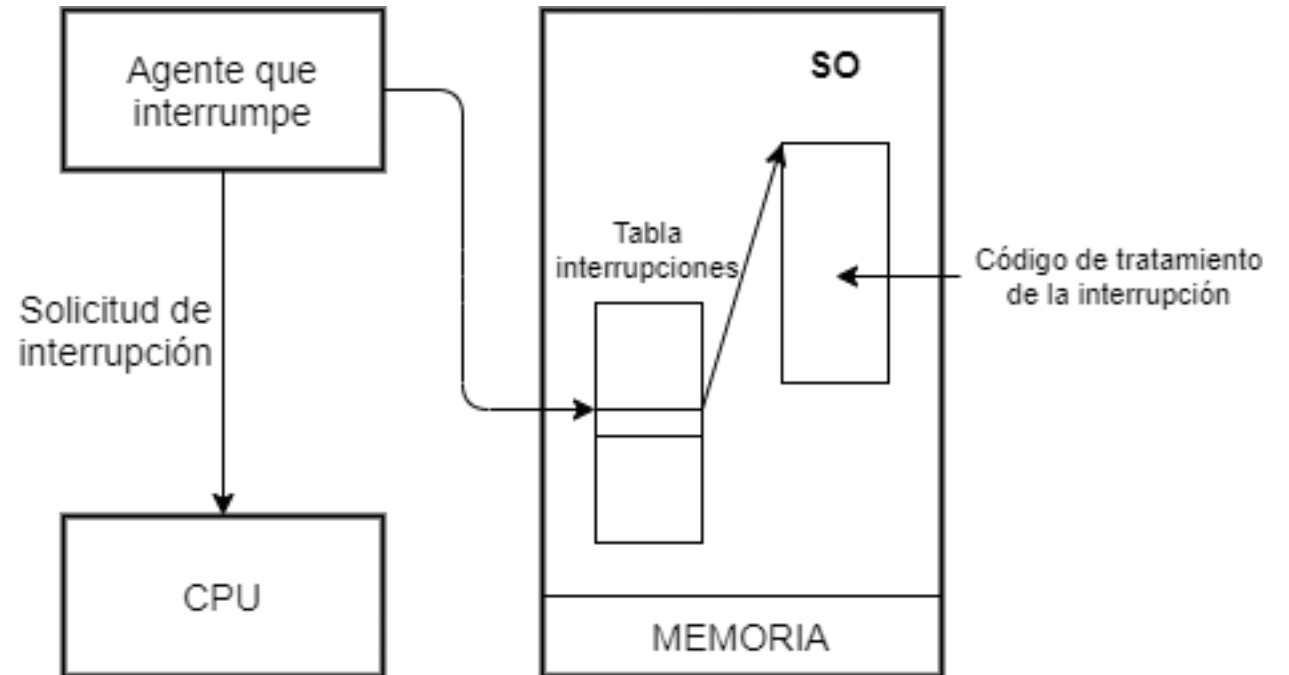


# Interrupciones

- Se envía una señal a la CPU solicitando atención, cuando se atiende
  - Guardar registros: estado, generales, *Instruction Pointer* (IP)
  - Elevar nivel de ejecución a nivel de núcleo.
  - Carga en IP instrucción de tratamiento de interrupción
- El agente que interrumpe
  - Suministra vector de interrupción
  - Dirección de comienzo del programa que atiende (tratamiento) de la interrupción
- Mecanismo que permite al S.O manejar eventos asíncronos

# Interrupciones

- Por seguridad en SO
  - Tabla de interrupciones
  - Código de tratamiento de la interrupción
  - Evitar que programas de otros usuarios hagan daños.



Con modificaciones tomada de:  
(Carretero Pérez, J. et. al., 2001)

# Interrupciones

- Causas

- Excepciones del programa
  - Desbordamiento
  - División por cero
  - Operaciones inválidas
- Interrupciones de reloj
- Interrupciones de E/S
  - Controladores de dispositivos
- Interrupciones de hardware
  - Solicitud de atención a un dispositivo

- Causas

- Excepciones de hardware
  - Errores de paridad
  - Corte de suministro eléctrico
- **Instrucciones TRAP para solicitar servicios al SO**
  - Pasa ejecución a modo Kernel.

# INT 21 en DOS (interrupción de software)

```
org 0x100
```

```
mov dx, msg
```

```
mov ah, 9
```

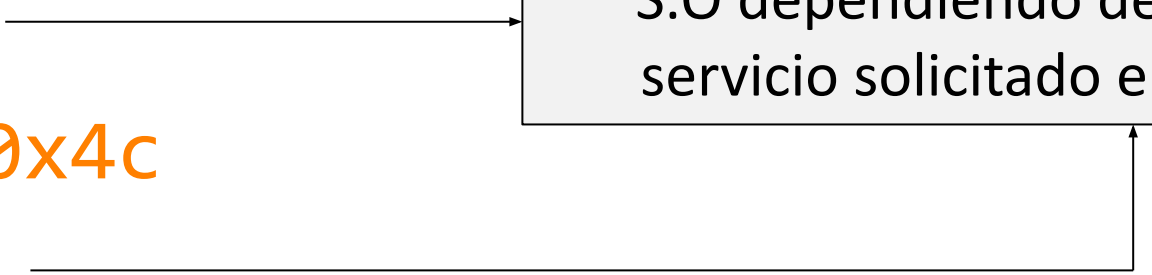
```
int 0x21
```

```
mov ah, 0x4c
```

```
int 0x21
```

```
msg db 'Hello, World!', 0x0d, 0x0a, '$'
```

Mismo número de interrupción.  
S.O dependiendo de parámetros ubica  
servicio solicitado en el código del S.O



# Reloj

- Circuito oscilador con tres funciones básicas
  - Señal que gobierna el ritmo de ejecución de las instrucciones
  - Generador de interrupciones periódicas
  - Contador de fecha y hora
- Por ejemplo 1 Hz
  - El procesador se activa cada segundo
  - Activación completa
  - Realiza todas las tareas de las que es capaz cada segundo
- Por ejemplo 1 GHz
  - El procesador se activa 1.000.000.000 (mil millones) veces por segundo
- Por ejemplo Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
  - El procesador se activa 2.300 millones de veces por segundo



# Reloj

- Generador de interrupciones periódicas
  - Interrupciones de reloj o ticks
  - El S.O entra ejecutar de forma sistemática cada cierto intervalo de tiempo
    - Evitar que un programa monopolice el uso de la CPU.
  - Permite que múltiples programas puedan ocupar la CPU en diferentes intervalos de tiempo.
- Contador de fecha y hora
  - El contador se incrementa en cada tick de reloj
  - Se toma de referencia una fecha para incrementar el contador
  - En computadores actuales se mantiene mediante circuito dedicado que requiere batería
  - En computadores más viejos había que indicar fecha y hora

# Periféricos

- Registros del controlador incluidos en el mapa de E/S
  - Se acceden mediante instrucciones de E/S
- Registro de datos
  - Intercambio de datos
  - Carga de datos leídos desde el dispositivo
  - Extracción de datos para escritura en el dispositivo
- Registro de estado
  - Operación de lectura: registro de datos con nuevo valor
  - Operación de escritura: necesita datos en el registro de datos
  - Indicar errores
- Registro de control: indicar operaciones ha realizar

# Periféricos



Con modificaciones tomada de:  
(Carretero Pérez, J. et. al., 2001)

# E/S y concurrencia

- E/S programada
  - Procesador ejecuta un programa de E/S
  - No hay concurrencia entre el procesador y la E/S
- E/S por interrupciones y DMA
  - El procesador no tiene que estar atendiendo la E/S
  - El procesador puede estar ejecutando otro programa mientras espera los datos
- Fases de una operación de E/S
  1. Envío de orden al periférico
  2. Lectura /escritura de datos
  3. Fin de operación

# E/S y concurrencia

- Envío de orden al periférico
  - Escribir la orden en los registros del controlador del periférico
  - Se usan instrucciones de E/S
  - Las instrucciones se pueden ejecutar a la velocidad del procesador
- Transferencia de datos
  - Interviene el periférico (más lento que la CPU)
  - Espera activa (E/S programada)
    - En E/S programada esperar a que se lean/escriban TODOS los datos
  - Espera pasiva (E/S por interrupción)
    - El controlador genera una interrupción cuando tenga datos

```
n = 0
while n < m bytes
    read registro_control
    if (registro_control ==
dato_disponible)
        read registro_datos
        store datos en memoria principal
        n = n + 1
    endif
endwhile
```

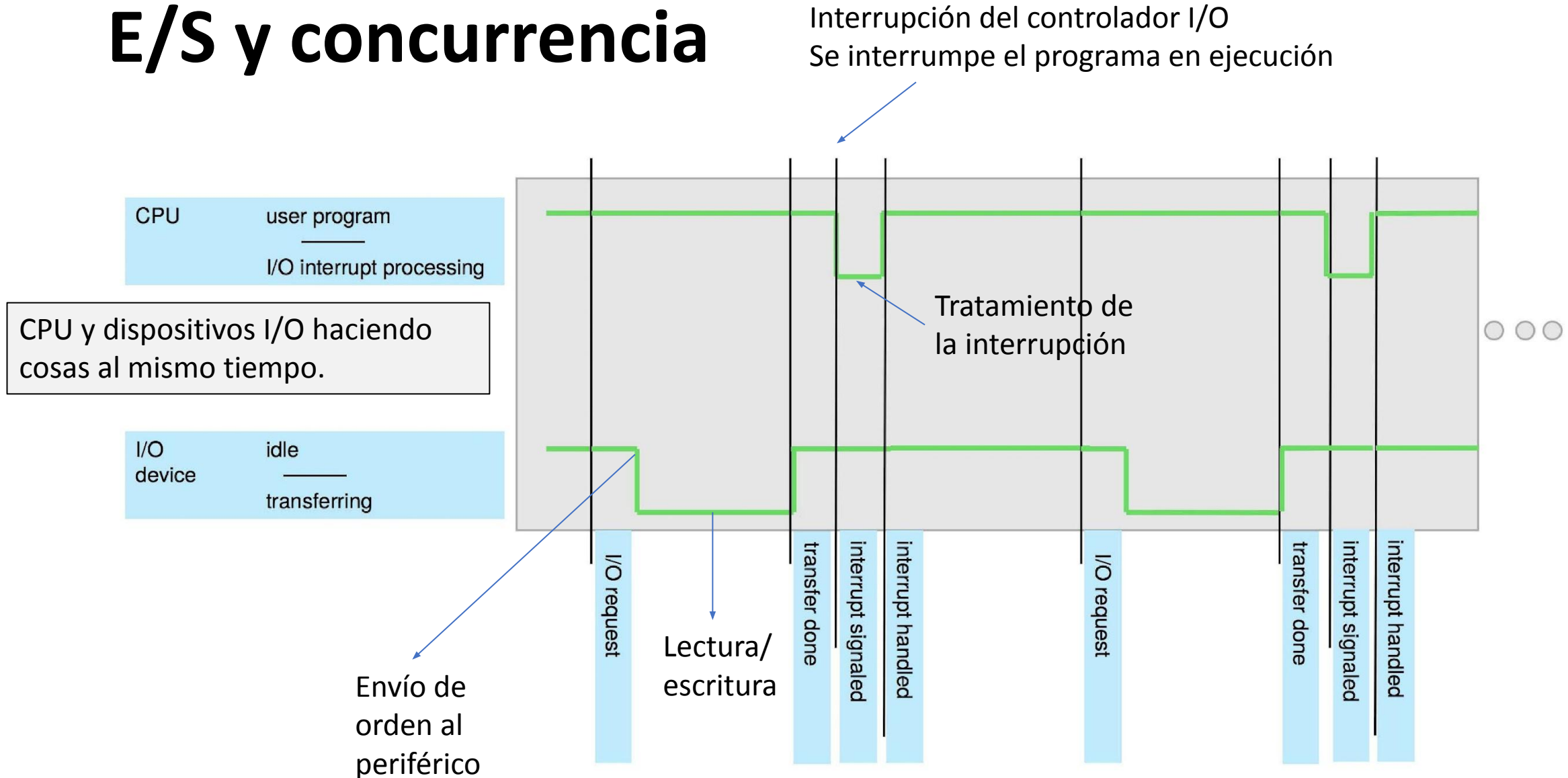
# E/S y concurrencia

- Transferencia de datos
  - E/S por DMA
    - El controlador transfiere directamente los datos a la memoria principal
    - No se usa la CPU para la transferencia de datos
    - Cuando se termina la transferencia de los datos se genera una interrupción
    - Se avisa al procesador que se tienen los datos listos en memoria
    - Se requiere mayor inteligencia por parte del controlador
- Es función del S.O explotar la concurrencia de la E/S y el procesador
  - La idea es que el procesador esté ocupado haciendo cosas útiles
  - Se requieren mecanismos (diseños) eficientes para atender a dispositivos de E/S

# E/S y concurrencia

	<b>Envío de orden</b>	<b>Espera de dato</b>	<b>Transferencia de dato</b>	<b>Fin operación</b>
<b>E/S programada</b>	Procesador	Procesador	Procesador	Procesador
<b>E/S por interrupciones</b>	Procesador	Controlador	Procesador	Procesador
<b>E/S por DMA</b>	Procesador	Controlador	Controlador	Procesador

# E/S y concurrencia



(Silberschatz, A., et. al., 2018)



# E/S y concurrencia

- **Momento 1:** Dispositivo I/O recibe una solicitud de I/O.
  - La solicitud se recibe a través del controlador del dispositivo.
  - Dispositivo I/O inicialmente está en estado *IDLE*
- **Momento 2:** Dispositivo I/O empieza a transferir datos
  - Se cambia de *IDLE* a *transferring*
- **Momento 3:** Dispositivo I/O termina de transferir datos y genera una interrupción
- **Momento 4:** CPU se interrumpe y atiende interrupción
- **Momento 5:** CPU continua su estado de ejecución del programa de usuario
- Se asume concurrencia. P. Ej.: E/S por interrupción o E/S por DMA.

# E/S y concurrencia

- Hardware puede generar una interrupción en cualquier momento.
  - Señaliza a la CPU: ¡Hey! necesito que me atiendas
  - Señal que usualmente viaja por el bus del sistema
  - Línea física solicitud-interrupción.
- CPU lee línea de solicitud-interrupción después de cada instrucción.
  - Se lee el número de interrupción (índice del vector de interrupciones)
  - Se “salta” a la dirección de memoria indicada por el vector de interrupciones.
  - Se guarda el estado de ejecución antes de la interrupción: p. ej.: PUSH registros.
  - Se atiende interrupción: ejecución del programa de interrupción
  - Se restaura estado
  - Se retorna de la interrupción

# Software usado para las demostraciones

- Visual Studio 2019 Community Edition
  - <https://visualstudio.microsoft.com/es/vs/community/>
- x64dbg
  - <https://x64dbg.com/>
- Visual Studio Code
  - <https://code.visualstudio.com/>
- Compilador C/C++ GNU para Windows
  - MinGW: <https://sourceforge.net/projects/mingw/>

# Referencias

- Carretero Pérez, J., de Miguel Anasagasti, P., García Carballeira, F., & Pérez Costoya, F. (2001). *Sistemas Operativos. Una visión aplicada*. McGraw-Hill/Interamericana de España.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts*. Wiley.