Introducción a los Sistemas Operativos

Anexo – Arquitectura de Computadoras

- Versión: Agosto 2024
- Palabras Claves: Sistemas Operativos, Harware, Interrupciones, Registros

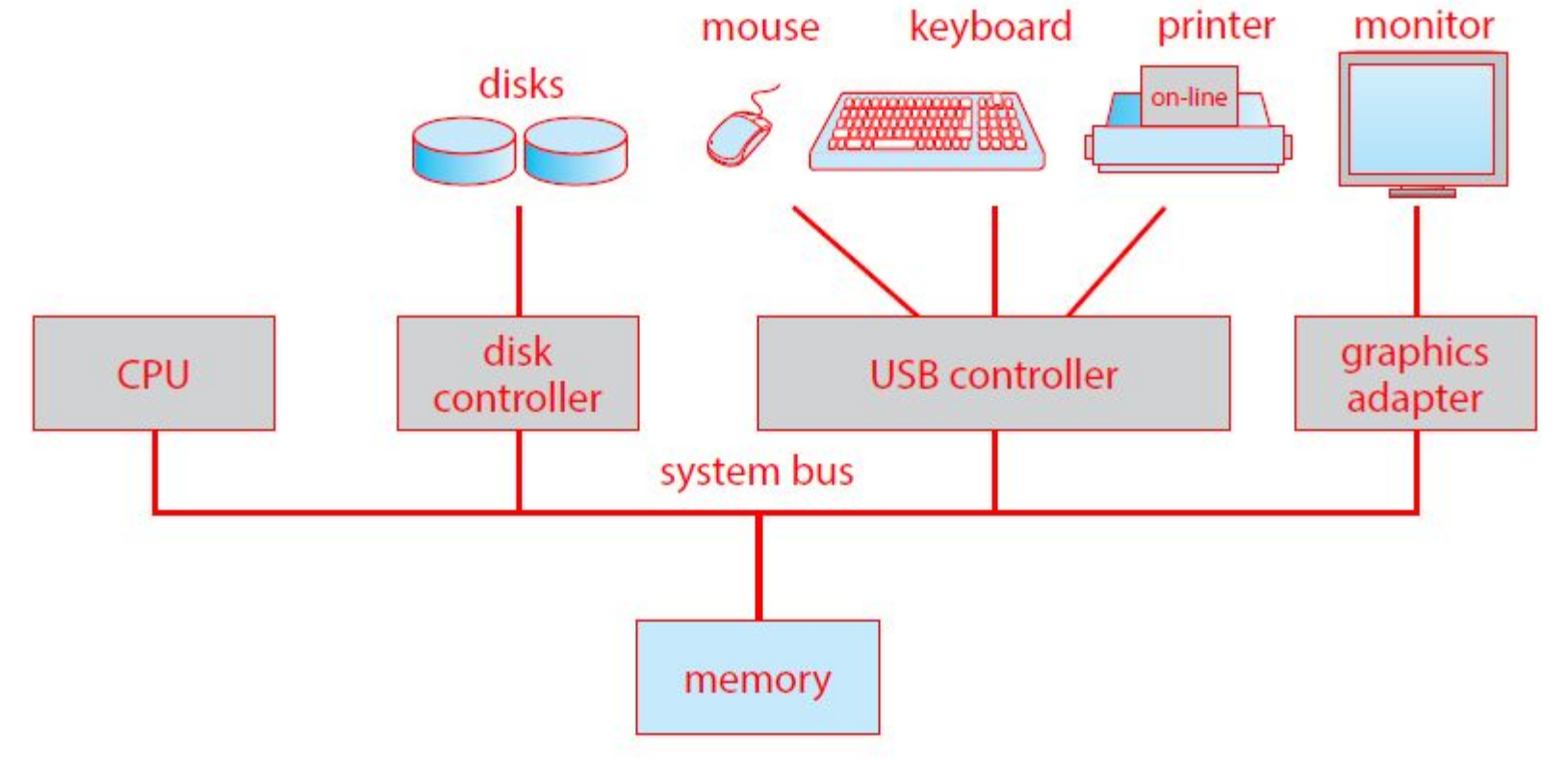
Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño) y y Conceptos de sistemas operativos (Silberschatz, Galvin, Gagne)

Objetivo de la presentación

- Incluye los conceptos vistos en asignatura/s previa/s y que son necesarios tener presentes para conceptos que se verán durante la cursada.

Elementos Básicos de una computadora

- Procesador
- Memoria Principal
  - ✔ Volátil
  - ✔ Se refiere como memoria real o primaria
- Componentes de E/S
  - ✔ Dispositivos de memoria secundaria
  - ✔ Equipamiento de comunicación
  - ✔ Monitor / teclado / mouse
- Bus Sistema
  - ✔ comunicación entre procesadores, memoria, dispositivos de E/S



Fuente: Operating System Concepts. Silberschatz, Galvin, Gagne

Componentes de alto nivel

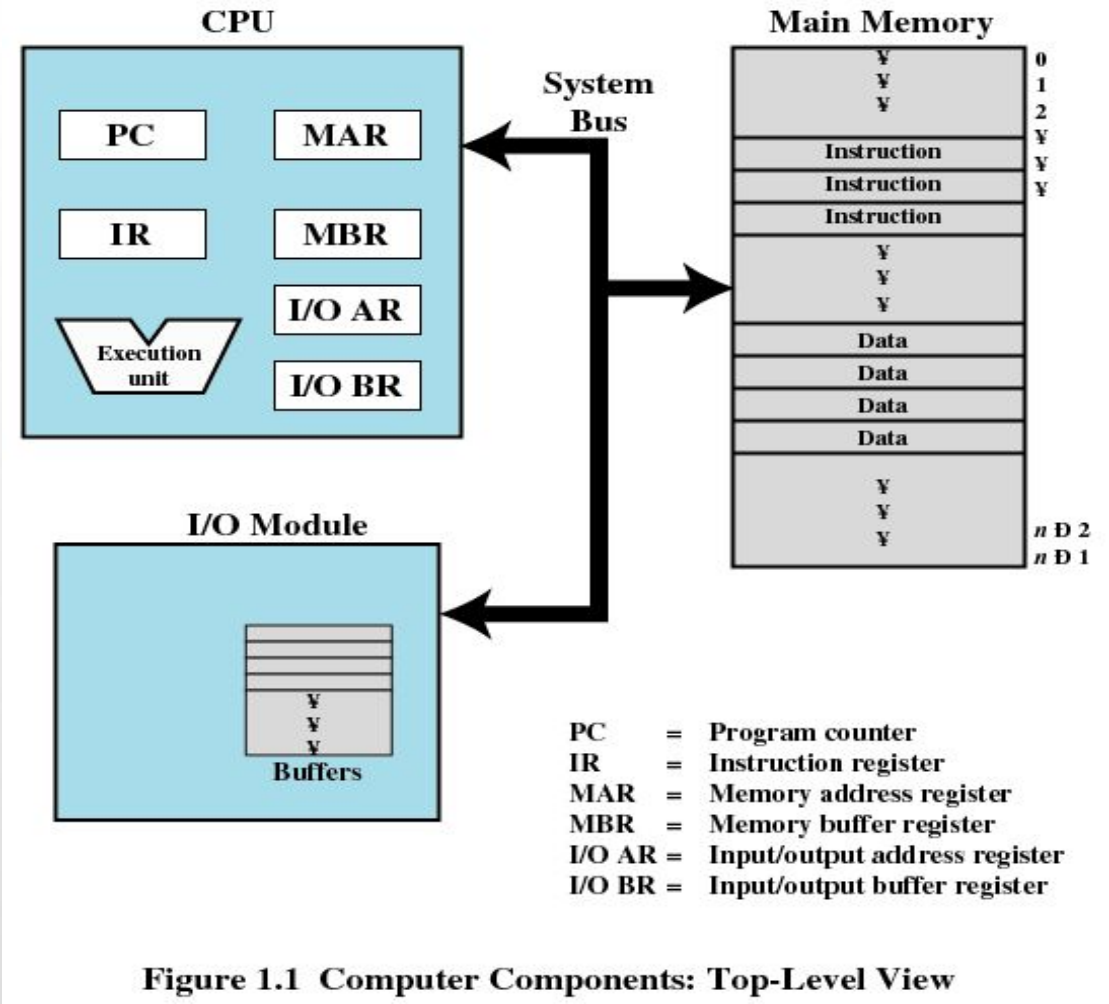


Figure 1.1 Computer Components: Top-Level View

Registros del Procesador

- Visibles por el usuario
  - ✔ Registros que pueden ser usados por las aplicaciones
- De Control y estado
  - ✔ Para control operativo del procesador
  - ✔ Usados por rutinas privilegiadas del SO para controlar la ejecución de procesos

Registros Visibles por el usuario

- Pueden ser referenciados por lenguaje de máquina
- Disponible para programas/aplicaciones
- Tipos de registros
  - ✔ Datos
  - ✔ Direcciones
    - Index
    - Segment pointer
    - Stack pointer

Registros de Control y Estado

- Program Counter (PC)
  - ✔ Contiene la dirección de la proxima instrucción a ser ejecutada
- Instruction Register (IR)
  - ✔ Contiene la instrucción a ser ejecutada
- Program Status Word (PSW)
  - ✔ Contiene códigos de resultado de operaciones
  - ✔ habilita/deshabilita Interrupciones
  - ✔ Indica el modo de ejecución (Supervisor/usuario)

Ciclo Ejecución de Instrucción

- Dos pasos
  - ✔ Procesador lee la instrucción desde la memoria
  - ✔ Procesador ejecuta la instrucción

Ciclo Instrucción

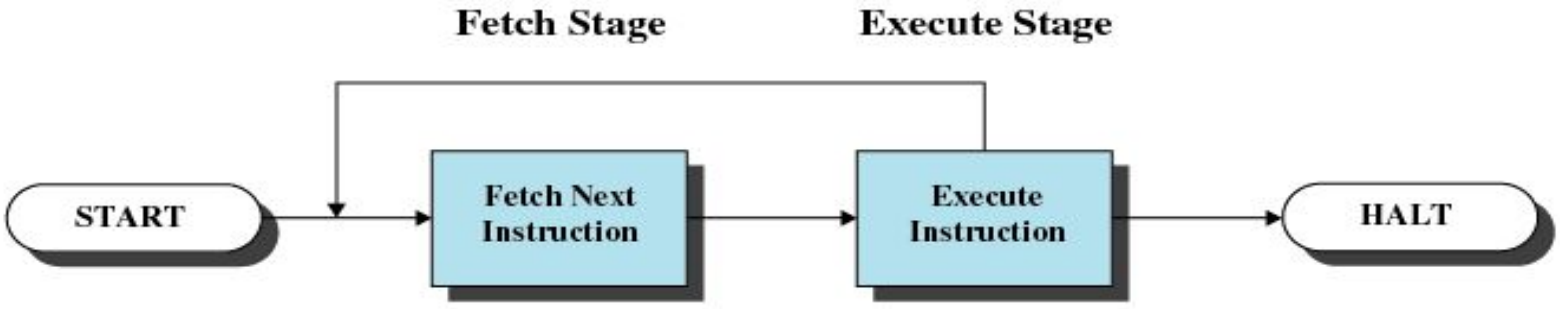


Figure 1.2 Basic Instruction Cycle

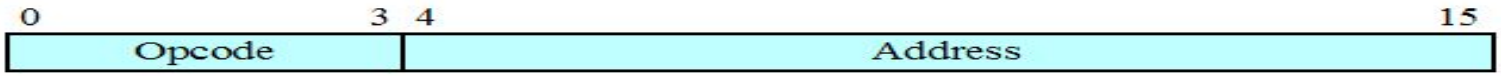
Instrucción: Fetch y Execute

- El procesador busca (fetch) la instrucción en la memoria
  - (PC) → IR
- El PC se incrementa después de cada fetch para apuntar a la próxima instrucción
  - PC = PC + 4

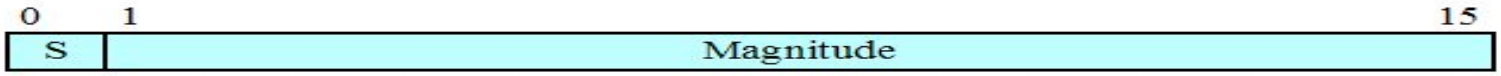
IR - Instruction Register

- La instrucción referenciada por el PC se almacena en el IR y se ejecuta
- Categorías de instrucciones
  - ✔ Procesador - Memoria
    - Transfiere datos entre procesador y memoria
  - ✔ Procesador - E/S
    - Transfiere datos a/o desde periféricos
  - ✔ Procesamiento de Datos
    - Operaciones aritméticas o lógicas sobre datos
  - ✔ Control
    - Alterar secuencia de ejecución

Características de una máquina hipotética



(a) Instruction format



(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

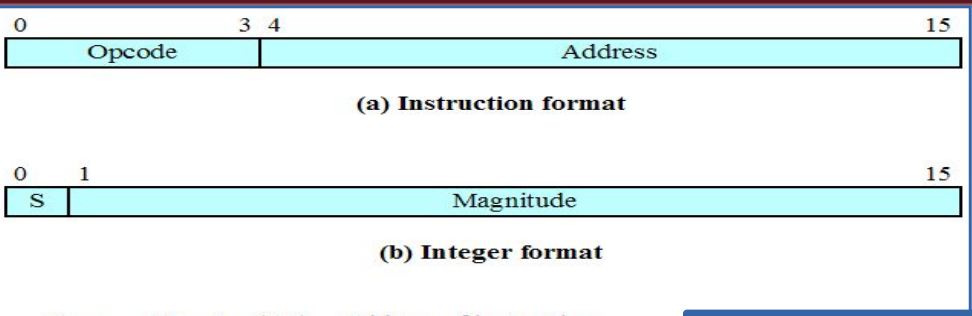
(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Ej. de una ejecución de programa



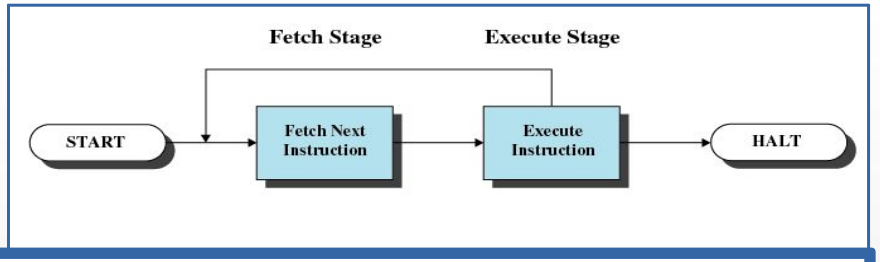
Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypoth



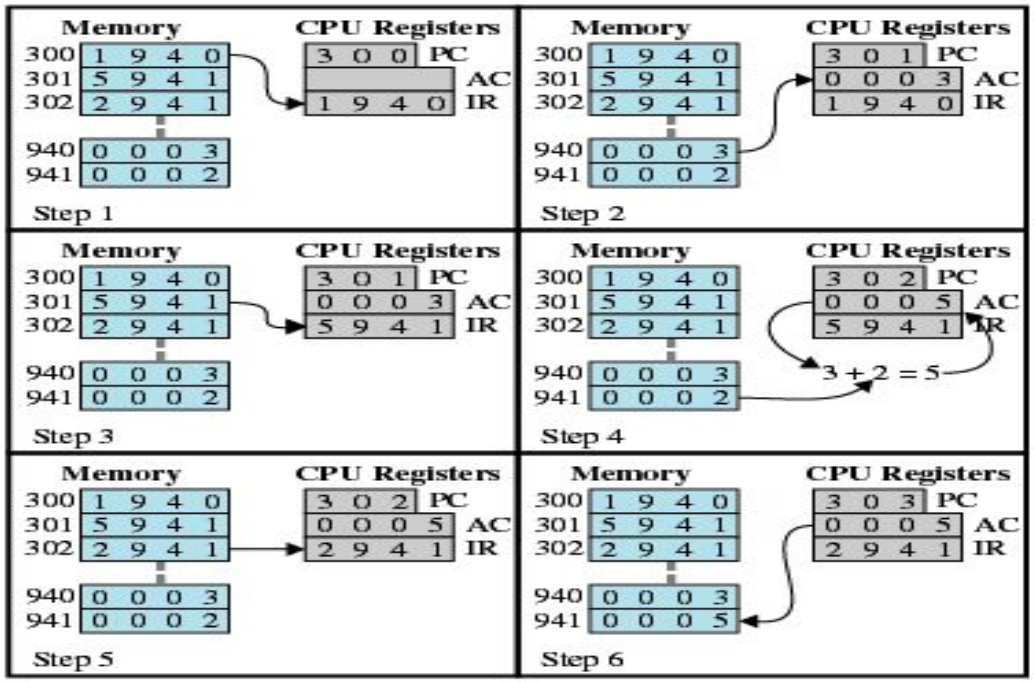


Figure 1.4 Example of Program Execution (contents of memory and registers in hexadecimal)

Interrupciones

- Interrumpen el secuenciamiento del procesador durante la ejecución de un proceso
- Dispositivos de E/S más lentos que el procesador
  - ✔ Procesador debe esperar al dispositivo

Necesidades del SO

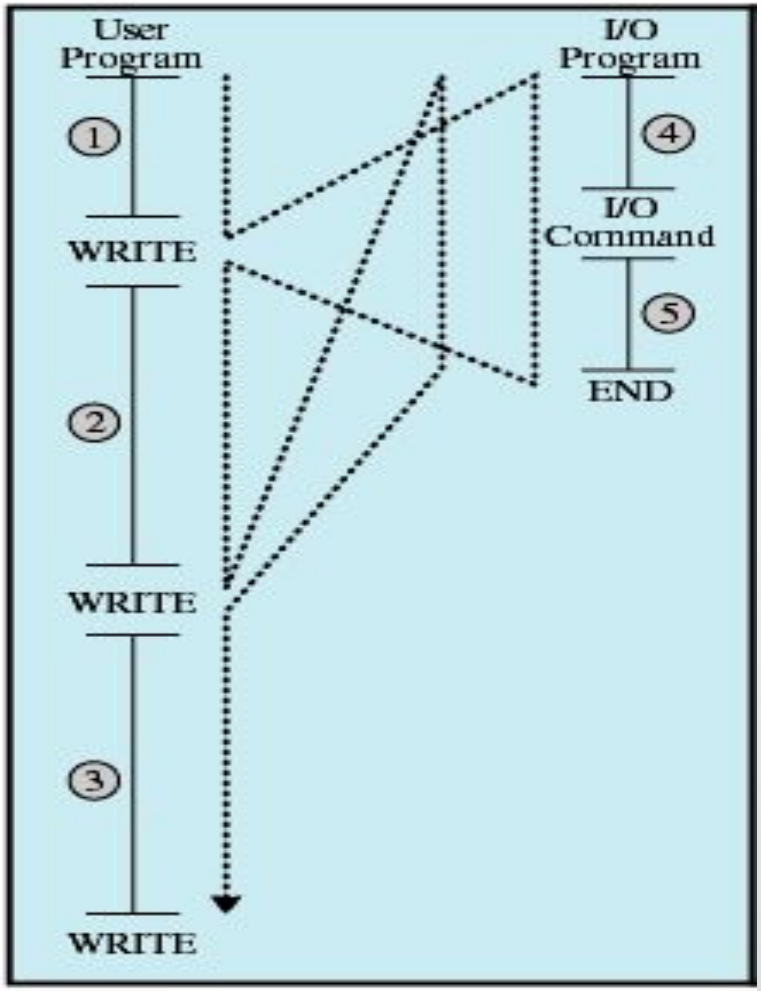
- Postergar el manejo de una interrupción en momentos críticos
- Atender en forma eficiente: la rutina de atención adecuada según el dispositivo
- Tener varios niveles de interrupción para que el SO pueda distinguir entre interrupciones de alta prioridad y de baja prioridad y responder adecuadamente

Clases de Interrupciones

Table 1.1 Classes of Interrupts

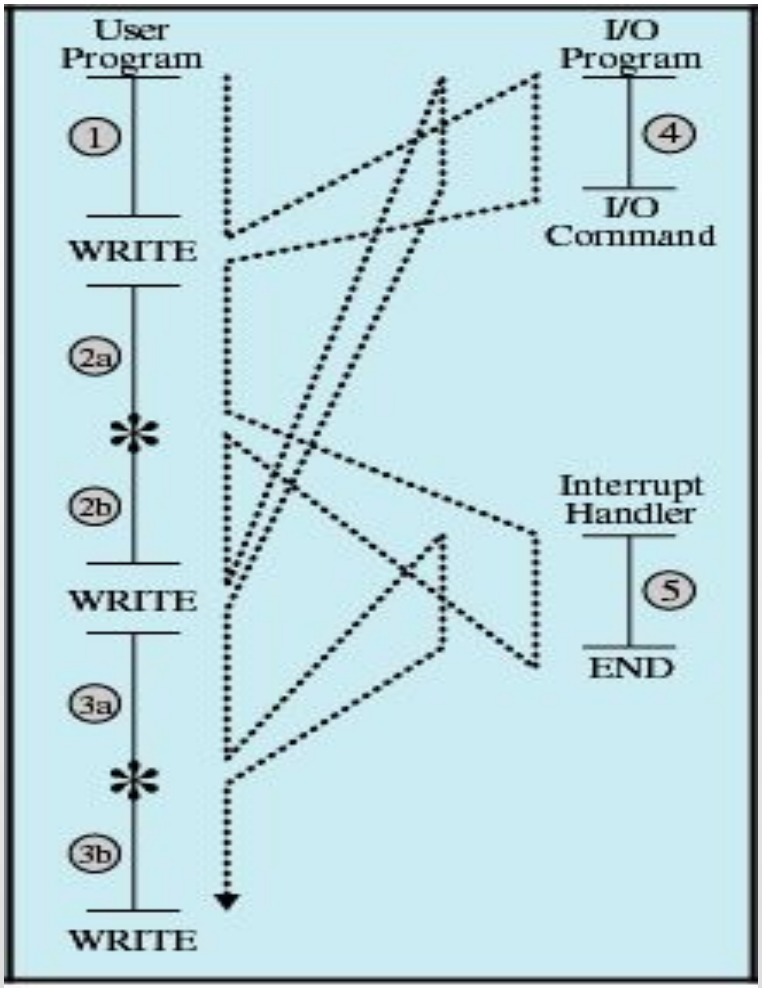
Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Flujo de control SIN interrupciones



(a) No interrupts

Flujo de control CON interrupciones



(b) Interrupts; short I/O wait

Interrupt Handler

- Programa (o rutina) que determina la naturaleza de una interrupción y realiza lo necesario para atenderla
  - ✔ Por ejemplo, para un dispositivo particular de E/S
- Generalemente es parte del SO

Interrupciones

- Suspende la secuencia normal de ejecución

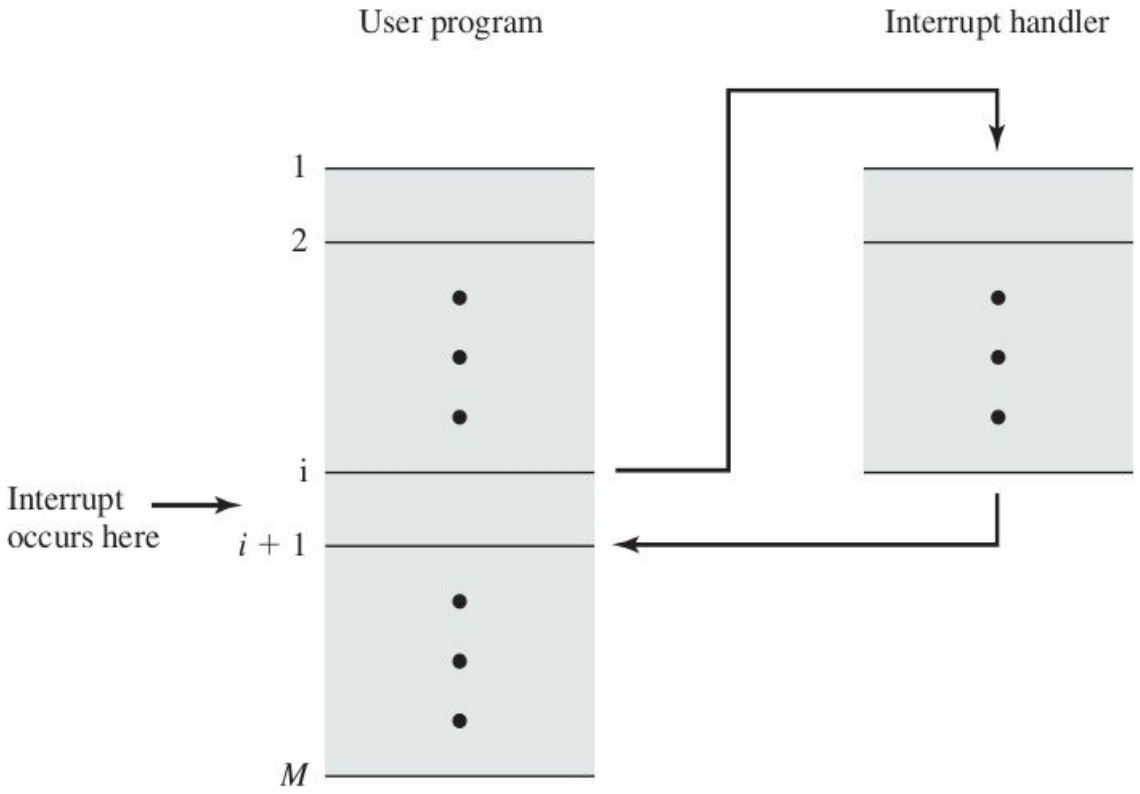


Figure 1.6 **Transfer of Control via Interrupts**

Ciclo de interrupción

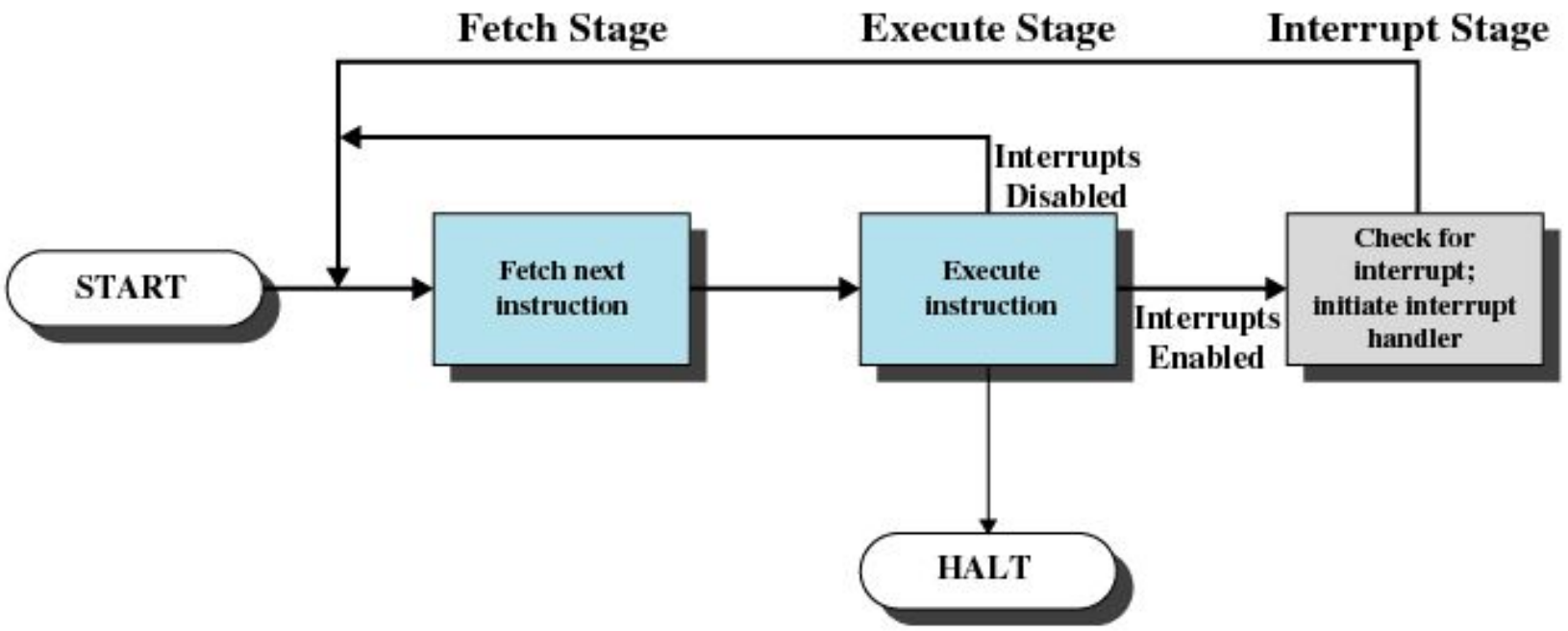


Figure 1.7 Instruction Cycle with Interrupts

Ciclo de interrupción

- El procesador chequea la existencia de interrupciones.
- Si no existen interrupciones, la próxima instrucción del programa es ejecutada
- Si hay pendiente alguna interrupción, se suspende la ejecución del progama actual y se ejecuta la rutina de manejo de interrupciones.

Procesamiento simple de una interrupción

IMPORTANTE!!!

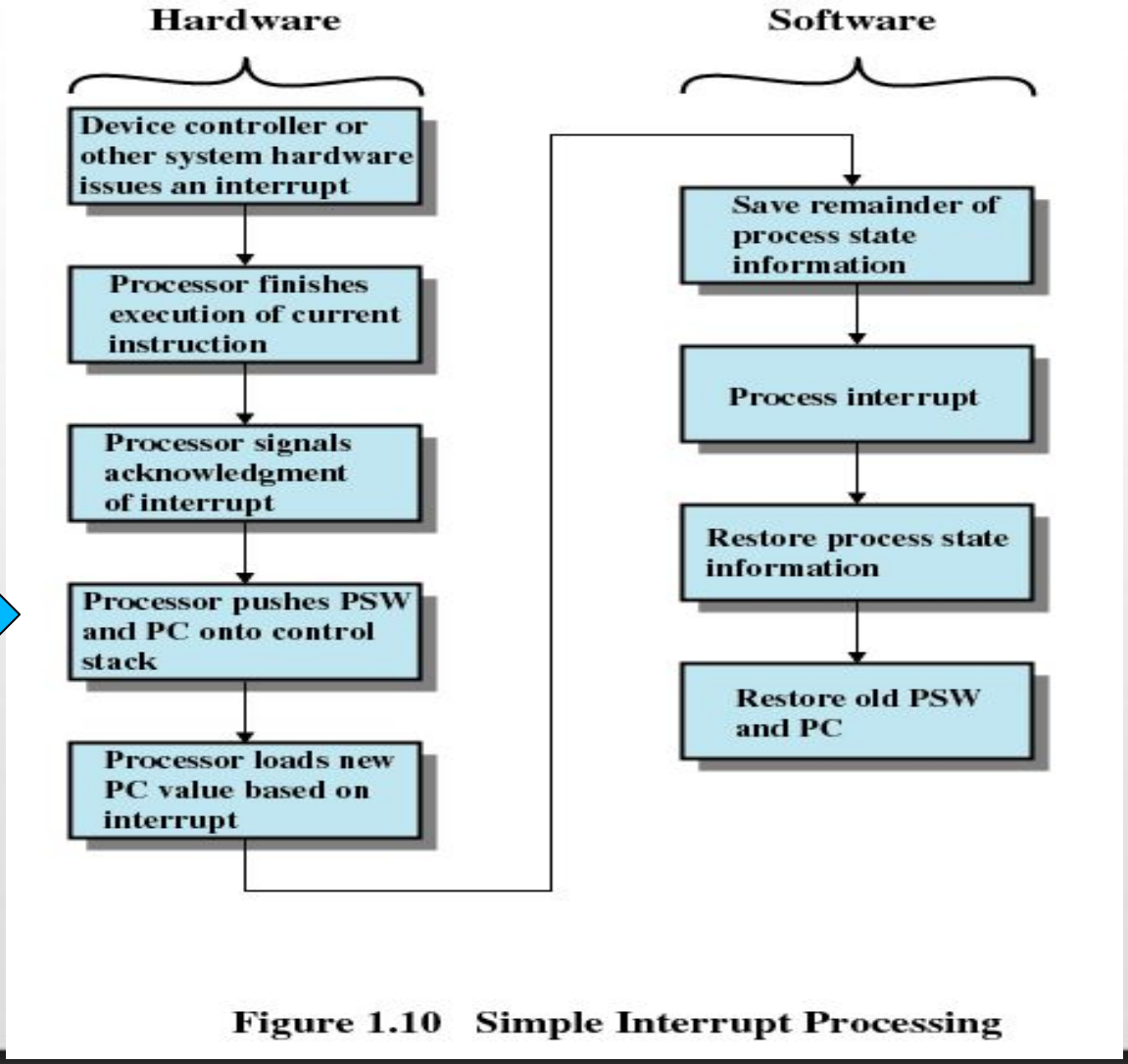


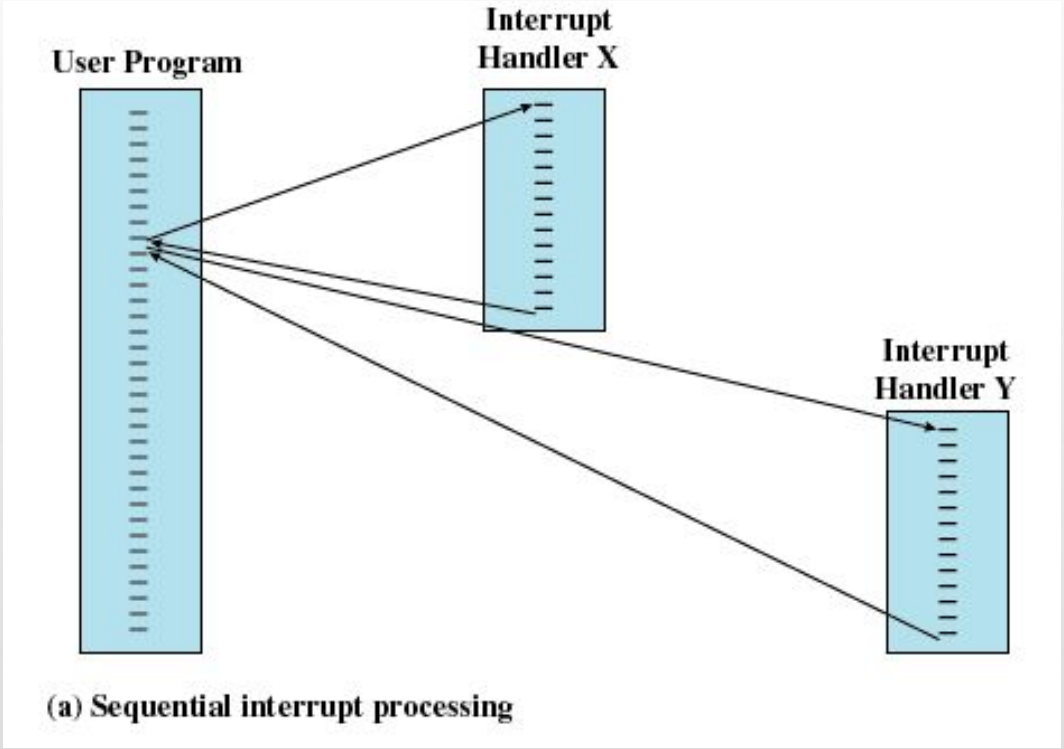
Figure 1.10 Simple Interrupt Processing

Interrupciones no enmascarables y enmascarables

- La mayoría de las CPUs tienen dos líneas de requerimiento de interrupciones: la de no enmascarables y las enmascarables.
- La de no enmascarables se reservan para eventos tales como errores de memoria no recuperables.
- La de enmascarables puede ser "apagada" por la CPU si en ese momento se está ejecutando una secuencia crítica de instrucciones. Estas son las que usan los controladores de dispositivo cuando necesitan servicio.

Multiples Interrupciones

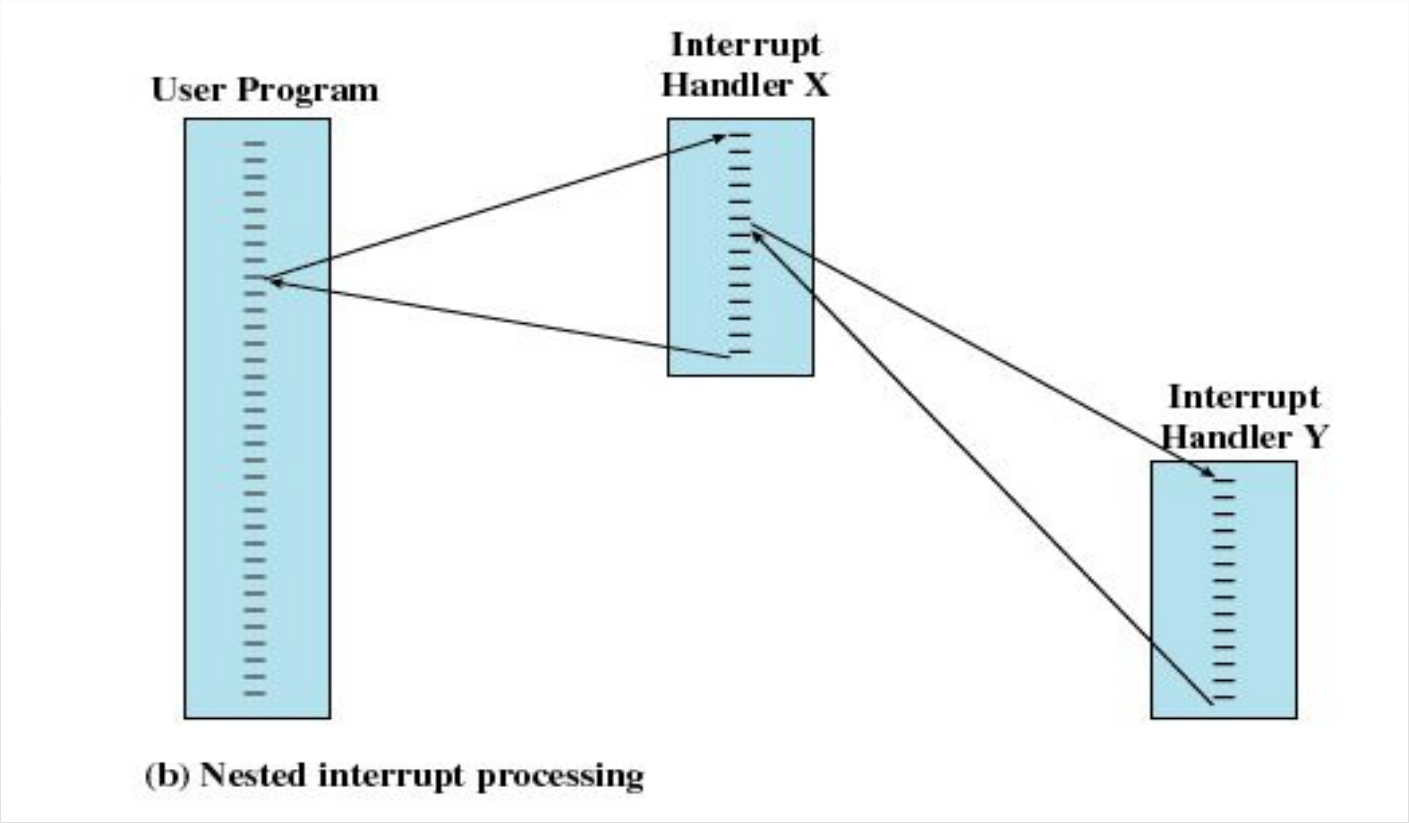
- Deshabilitar las interrupciones mientras una interrupción está siendo procesada.



(a) Sequential interrupt processing

Multiples Interrupciones

- Definir prioridades a las interrupciones



Multiples Interrupciones

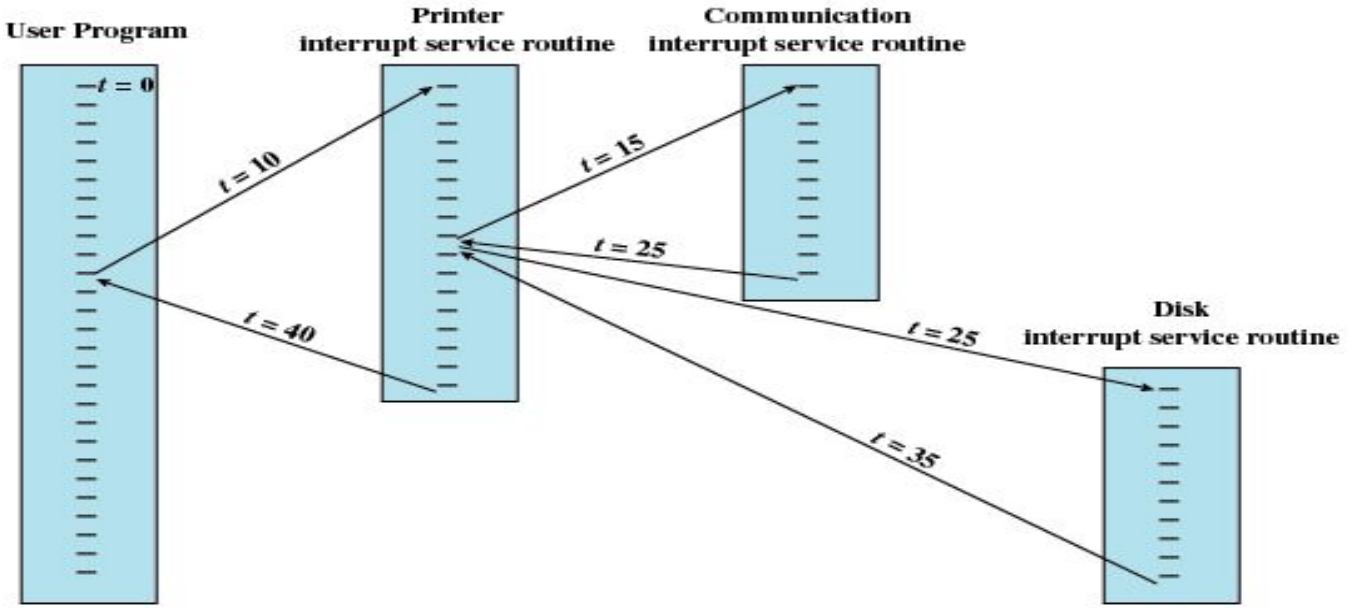


Figure 1.13 Example Time Sequence of Multiple Interrupts

Descripción de número en el vector de interrupciones de procesador Intel:

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

- De 1 a 31: **no enmascarables**, por ejemplo, errores de condición
- 32 a 255, **enmascarables**, usadas para interrupciones generadas por dispostivos

Introducción a los Sistemas Operativos

Anexo – Evolución

I.S.O.

- Versión: Agosto 2024
- Palabras Claves: Sistema Operativo, Servicios, Evolución, Batch, Multiprogramación, Timesharing

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño)

Evolución de un S.O.

Los SO evolucionan con el objeto de:

- Soportar nuevos tipos de HW
- Brindar nuevos Servicios
- Ofrecer mejoras y alternativas a problemas existentes
  - en la planificación
  - en el manejo de la memoria
  - etc

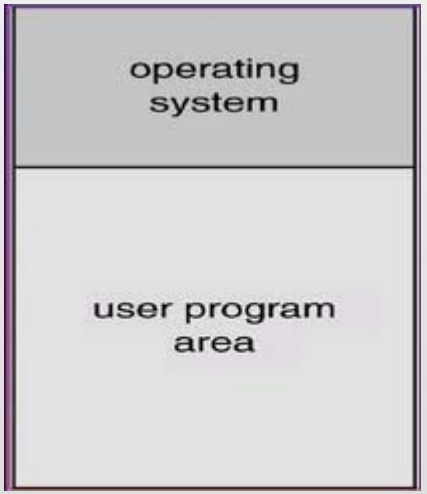
S.O. - Evolución Histórica

- Procesamiento en Serie
  - ✔ No existía un SO
  - ✔ Máquinas eran utilizadas desde una consola que contenía luces, interruptores, dispositivos de entrada e impresoras.
  - ✔ Problemas:
    - Planificación. Alto nivel de especialización. Costos
    - Configuración: Carga del compilador, fuente, salvar el programa compilado, carga y linkeo.

S.O. - Evolución Histórica (cont.)

- Sistemas por Lotes Sencillos (batch)
  - ✔ Monitor Residente
    - Software que controla la secuencia de eventos
    - Los trabajos se colocan juntos
    - Los programas vuelven al monitor cuando finaliza la ejecución
    - No hay interacción con el usuario

    mientras se ejecutan los trabajos

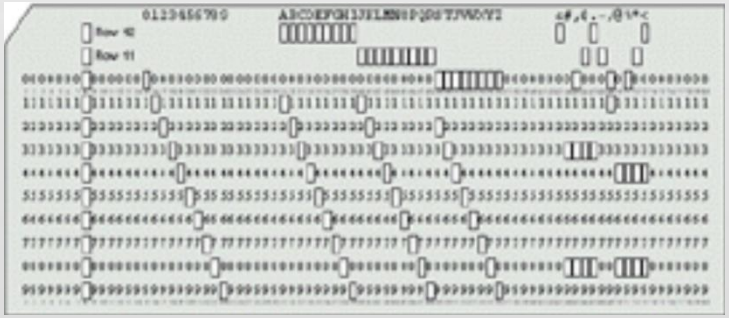


S.O. - Evolución Histórica (cont.)

Batch processing

The elements of the basic IBM 1401 system are the 1401 Processing Unit, 1402 Card Read-Punch, and 1403 Printer.

Punching cards

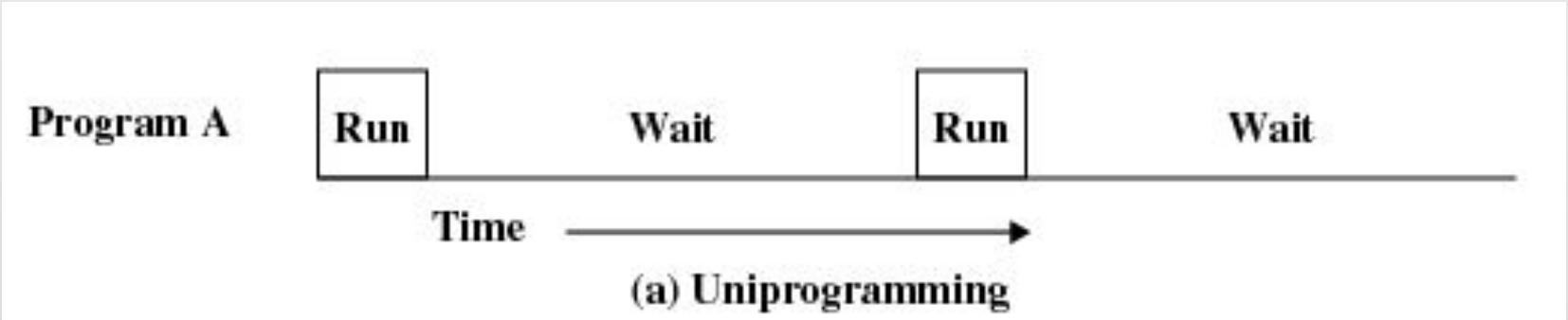


Sistema Batch

Baja utilización de la CPU

Dispositivos de E/S mucho mas lentos con respecto a la CPU

Ante instrucción de E/S, el procesador permanece ocioso. Cuando se completa la E/S, se continua con la ejecución del programa que se estaba ejecutando



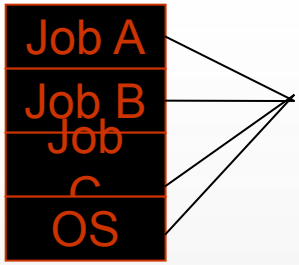
Multiprogramación

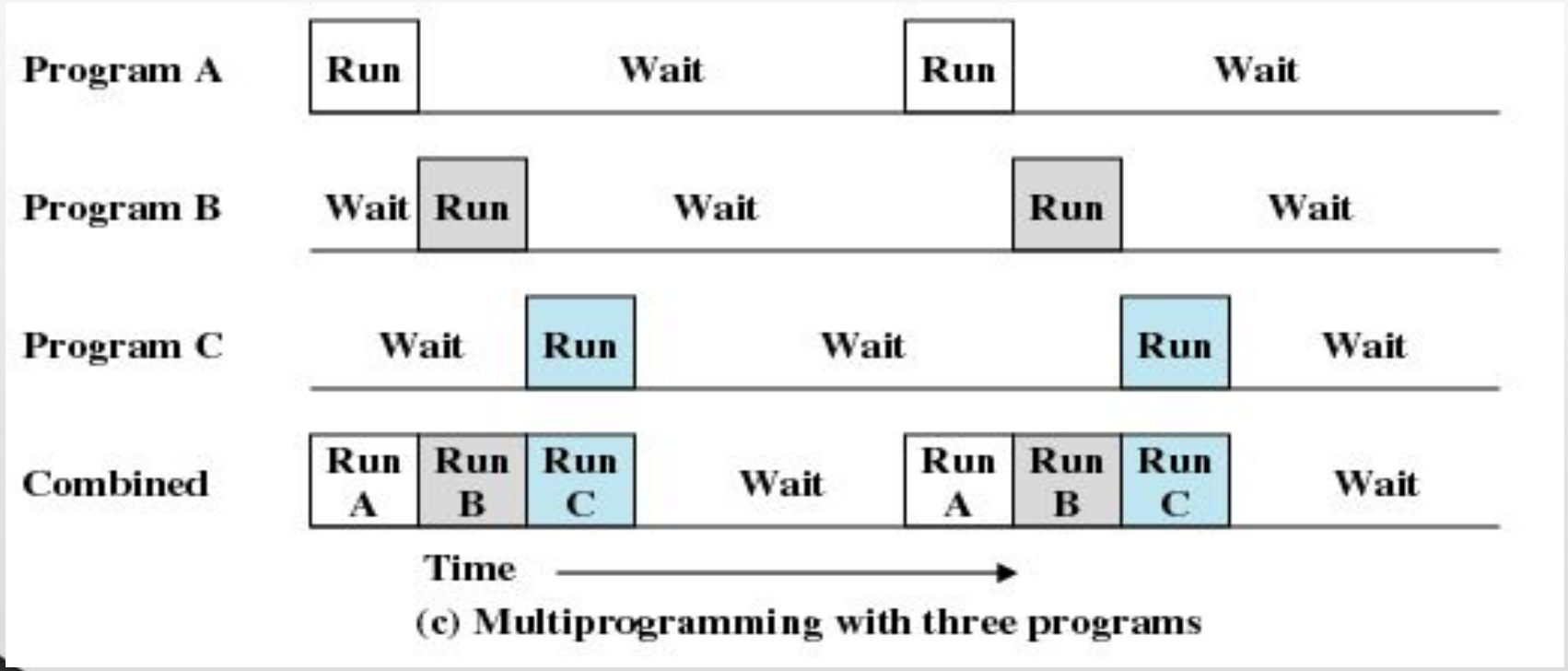
- La operación de los sistemas batch se vio beneficiada del spooling de las tareas, al solapar la E/S de una tarea de la ejecución de otra
- Al estar las tareas cargadas en disco, ya no era necesario ejecutarlas en el orden en el que fueron cargadas (job scheduling)
- El SO mantiene varias tareas en memoria al mismo tiempo.

Multiprogramación (cont)

- La secuencia de programas es de acuerdo a prioridad u orden de llegada
- Cuando el proceso necesita realizar una operación de E/S, la CPU en lugar de permanecer ociosa, es utilizada para otro proceso.
- Después que se completa la atención de la interrupción, el control puede o no retornar al programa que se estaba ejecutando al momento de la interrupción

Multiprogramación (cont)

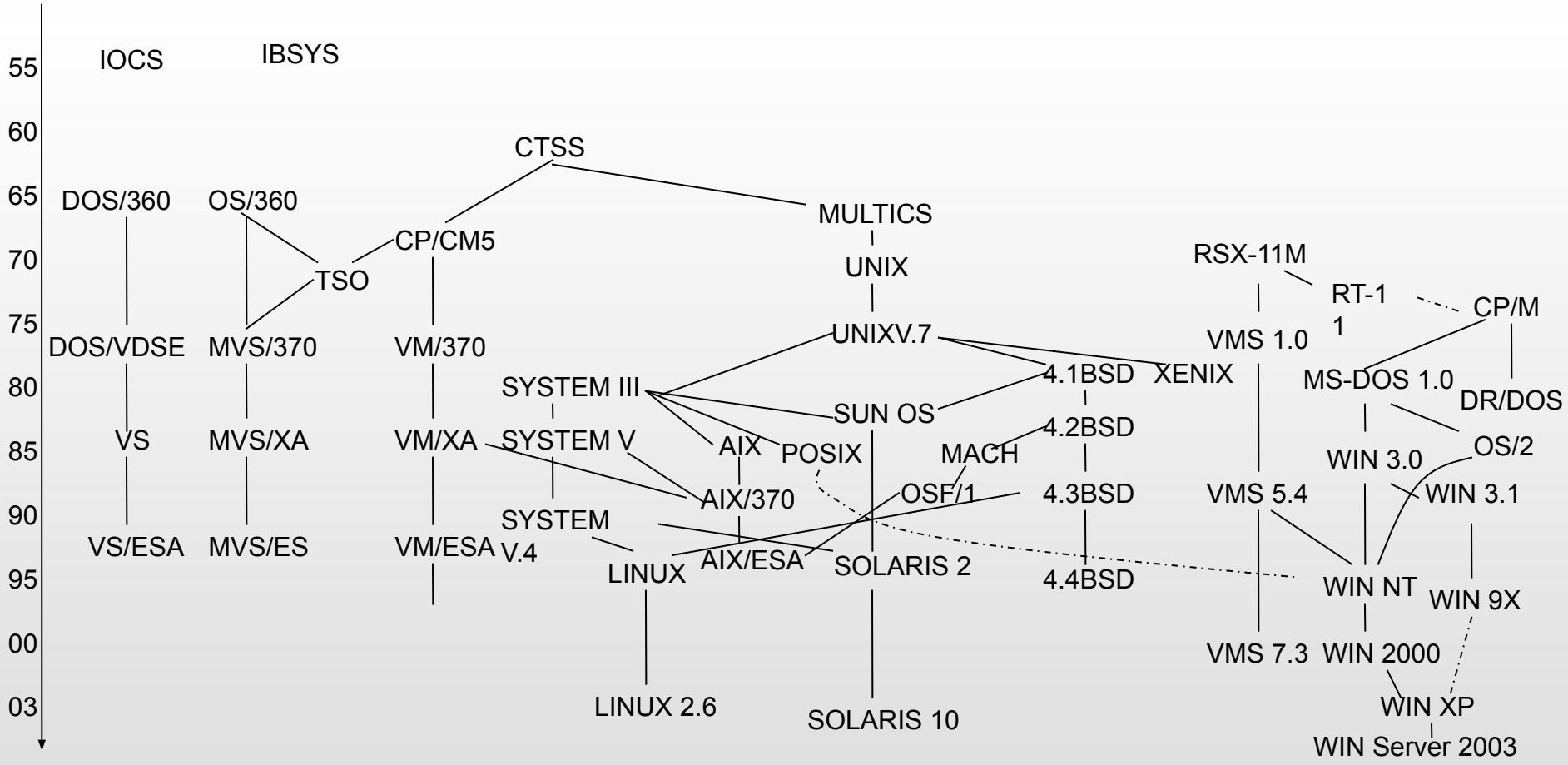




Tiempo Compartido

- Utiliza la multiprogramación para manejar múltiples trabajos interactivos
- El tiempo del procesador es compartido entre múltiples trabajos.
- Múltiples usuarios podrían acceder simultáneamente al sistema utilizando terminales
- Los procesos usan la CPU por un periodo máximo de tiempo, luego del cual se le da la CPU a otro proceso

Operating Systems Evolution



Referencias

- Historia de los S.O.

http://es.wikipedia.org/wiki/Historia_y_evoluci%C3%B3n_de_los_sistemas_operativos

- Línea del tiempo

http://en.wikipedia.org/wiki/Operating_systems_timeline

Referencias

- Historia de la primer Computadora Argentina

Buscá estos capítulos en youtube

CLEMENTINA / EPISODIOS

Capítulo 1
PRESENTACIÓN EN SOCIEDAD

VER

Capítulo 2
TÉ CON AMIGOS

VER

Capítulo 3
UN AMOR LÓGICO

VER

Capítulo 4
EL LEGADO

VER

Introducción a los Sistemas Operativos

Introducción – IV

Anexo llamadas al Sistema

Objetivo

- Programar un llamado a una “System Call” de manera directa. Sin utilizar ninguna librería.
- Considerar distintos aspectos al intentar realizar lo mismo en las siguientes arquitecturas:
  - 32 bits
  - 64 bits

Hello World!!

- Para programar el clasico “hello world” se necesitan mínimo realizar hacer 2 llamadas al sistema:
  - Una para escribir en pantalla un mensaje **SYSCALL WRITE**
  - Otra para terminar la ejecución de un proceso
  - **SYSCALL EXIT**

Hello World!!

- Para obtener información sobre estas SYSCALLs podemos utilizar los manuales del sistema.
- El comando man permite acceder a distintos tipos de documentación, en particular a información referida a systemcalls
  - write (man 2 write)
  - exit (man exit)

Hello World!!!

- Los manuales de las system calls permiten saber cuales son los parámetros

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

```
NAME
       exit - cause normal process termination

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Número de syscalls a utilizar

- Para indicarle al sistema operativo lo que queremos hacer (write o exit), es necesario saber cuál es el número asociado que tiene cada una de las syscalls
- Puede ser distinto en distintas arquitucturas

Del github de Linus Torvald

- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Hello World en x86 32bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl

```
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        sys_fork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_sys_open
6       i386    close                   sys_close
```

- En x86 32bit las sistem calls tienen los siguientes números:
  - write → syscall número 4
  - exit → syscall número 1

Hello World en x86 64bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

```
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
```

```
57      common  fork                    sys_fork/ptregs
58      common  vfork                   sys_vfork/ptregs
59      64      execve                  sys_execve/ptregs
60      common  exit                    sys_exit
61      common  wait4                   sys_wait4
62      common  kill                    sys_kill
```

- En x86 64bit las sistem calls tienen los siguientes números:
  - write → syscall número 1
  - exit → syscall número 60

Pasaje de parámetros en x86 32bit

- https://syscalls.kernelgrok.com/
  - EAX lleva el numero de syscall que se desea ejecutar
  - EBX lleva el primer parámetro
  - ECX lleva el segundo parámetro
  - EDX …
  - ESI
  - EDI

Instrucción que inicia la system call: **int 80h**

Pasaje de parámetros en x86 64bit

- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
  - EAX lleva el numero de syscall que se desea ejecutar
  - RDI lleva el primer parámetro
  - RSI lleva el segundo parámetro
  - RDX …
  - R10
  - R8
  - R9

Instrucción que inicia la system call: **syscall**

Hello world en x86 32 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count b
       file referred to by the file des
```

```
# 32-bit system call numbers and entr
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is always "i386" for this f
#
0       i386    restart_syscall
1       i386    exit
2       i386    fork
3       i386    read
4       i386    write
5       i386    open
6       i386    close
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

```
_start:

    ; sys_write(stdout, message, length)

    mov eax, 4          ; sys_write syscall
    mov ebx, 1          ; stdout
    mov ecx, message    ; message address
    mov edx, 14         ; message string length
    int 80h

    ; sys_exit(return_code)

    mov eax, 1          ; sys_exit syscall
    mov ebx, 0          ; return 0 (success)
    int 80h

section .data
    message: db 'Hello, world!',0x0A    ; message and newline
```

Hello world en x86 64 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

```
; sys_write(stdout, message, length)

mov rax, 1  ; sys_write
mov rdi, 1  ; stdout
mov rsi, message ; message address
mov rdx, length ; message string length
syscall

; sys_exit(return_code)

mov rax, 60  ; sys_exit
mov rdi, 0  ; return 0 (success)
syscall

section .data
 message: db 'Hello, world!',0x0A ; message and newline
 length: equ 14 ;
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Resumen

- Los manuales del sistema indican los parámetros necesarios para activar una system call

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

- Dependiendo la arquitectura, cambiará:
  - el número de system call utilizado para realizar una función determinada
  - La forma de pasar los parámetros al kenel

Resumen

- Los procesadores 32 bit y 64 bits usan un esquema de registros diferentes.

- Los procesadores 32 bit y 64 bits usan una instrucción distinta para activar las systemcalls:
  - 32 bits: **int 80h**
  - 64 bits: **syscall**

Referencias

- **Como programar un “hello world” en x86 32bit y 64bit**
- http://shmaxgoods.blogspot.com.ar/2013/09/assembly-hello-world-in-linux.html
- https://stackoverflow.com/questions/19743373/linux-x86-64-hello-world-and-register-usage-for-parameters

- **Mas información sobre formas de pasar parametros a una syscall**
- https://github.com/torvalds/linux
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl
- https://syscalls.kernelgrok.com/
- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- http://www.int80h.org/bsdasm/#system-calls

Introducción a los Sistemas Operativos / Conceptos de Sistemas Operativos

Procesos – Anexo I

Ejecución de procesos y dispatcher

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

- Versión: Septiembre 2023

Palabras Claves: Procesos, Planificación, Dispatcher, cambio de contexto, context switch

Los temas vistos en estas diapositivas han sido extraídos del libro de Stallings (Sistemas Operativos)

Ubicación de los procesos en RAM

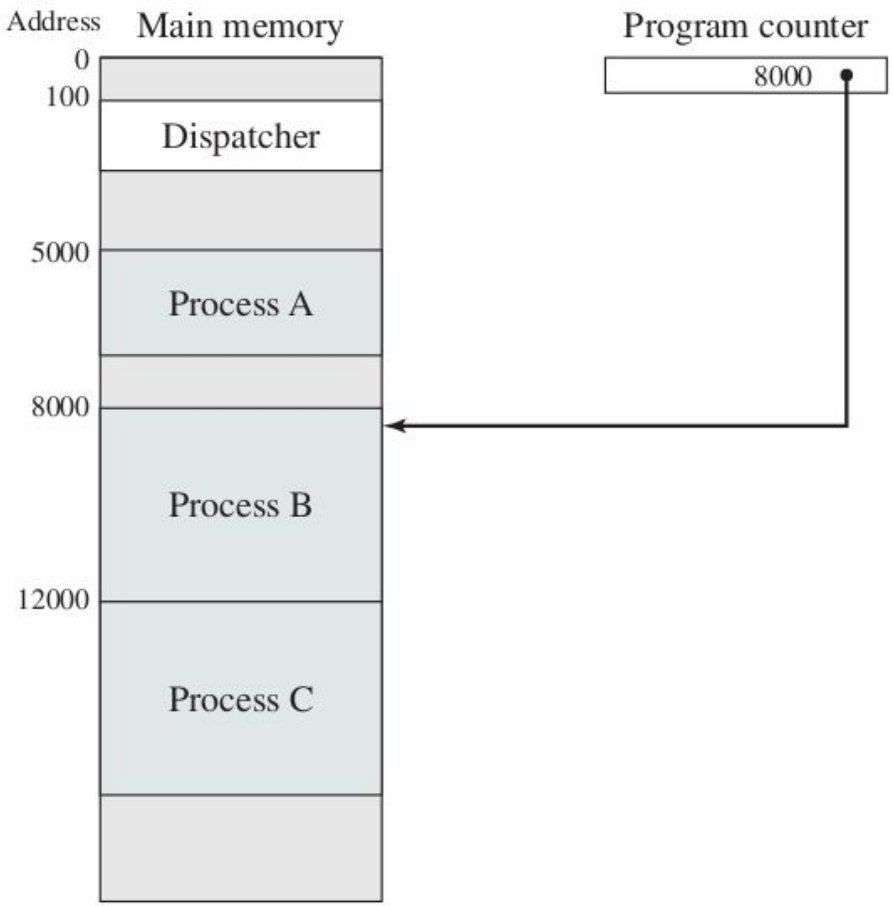


Figure 3.2 **Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13**

Secuencia de dir. de instrucciones

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of process A (b) Trace of process B (c) Trace of process C

5000 = Starting address of program of process A
8000 = Starting address of program of process B
12000 = Starting address of program of process C

Figure 3.3 Traces of Processes of Figure 3.2

Traza ejecución – Procesos y Dispatcher

1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
----------	Time-out
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
----------	I/O request
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003
27	12004
28	12005
----------	Time-out
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
----------	Time-out
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
----------	Time-out

100 = Starting address of dispatcher program

Introducción a los Sistemas Operativos

Anexo I
Arquitectura de Entrada/Salida

I.S.O.

- Versión: Octubre 2017
- Palabras Claves: Dispositivos de IO, Hardware de IO, IO programada, Polling, Interrupciones, DMA

Algunas diapositivas han sido extraídas de las ofrecidas para docentes desde el libro de Stallings (Sistemas Operativos) y el de Silberschatz (Operating Systems Concepts). También se incluyen diapositivas cedidas por Microsoft S.A.

Variedad en los dispositivos de I/O

- Legible para el usuario
  - ✔ Usados para comunicarse con el usuario
    - Impresoras, Terminales: Pantalla, Teclado, Mouse
- Legible para la máquina
  - ✔ Utilizados para comunicarse con los componentes electrónicos
    - Discos, Cintas, Sensores, etc.
- Comunicación
  - ✔ Usados para comunicarse con dispositivos remotos
    - Líneas Digitales, Modems, Interfaces de red, etc.

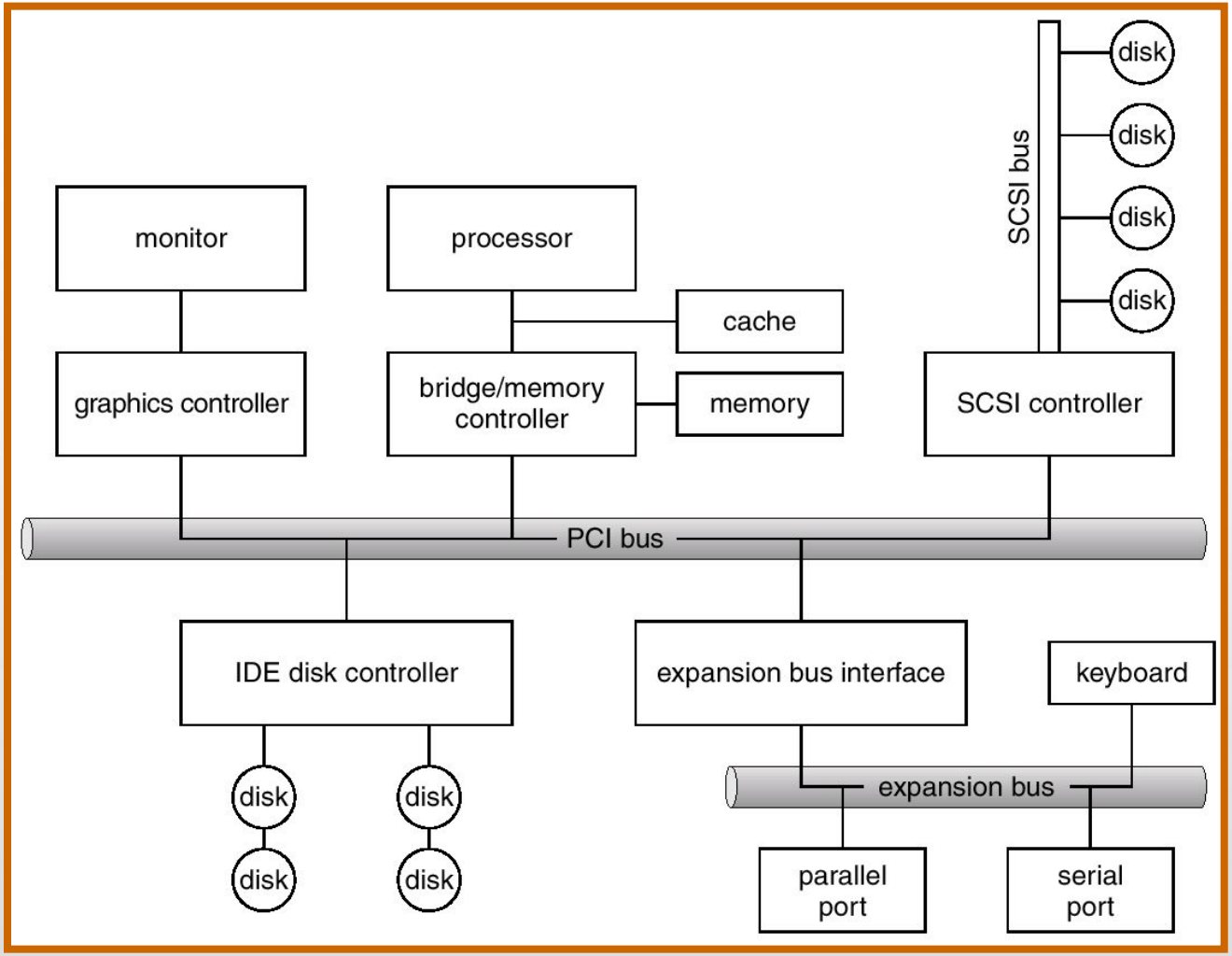
Problemas que surgen

- Amplia Variedad
  - ✔ Manejan diferentes cantidad de datos
  - ✔ En Velocidades Diferentes
  - ✔ En Formatos Diferentes
- La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

Hardware y software involucrado

- Buses
- Controladores
- Dispositivos
- Puertos de E/S – Registros
- Drivers
- Comunicación con controlador del dispositivo: I/O Programada, Interrupciones, DMA

Estructura de Bus de una PC



Comunicación: CPU - Controladora

- ¿Cómo puede la CPU ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo?
  - ✔ La controladora tiene uno o mas registros:
    - Registros para señales de control
    - Registros para datos
- La CPU se comunica con la controladora escribiendo y leyendo en dichos registros

Comandos de I/O

- CPU emite direcciones
  - ✔ Para identificar el dispositivo
- CPU emite comandos
  - ✔ Control – Que hacer?
    - Ej. Girar el disco
  - ✔ Test – Controlar el estado
    - Ej. power? Error?
  - ✔ Read/Write
    - Transferir información desde/hacia el dispositivo

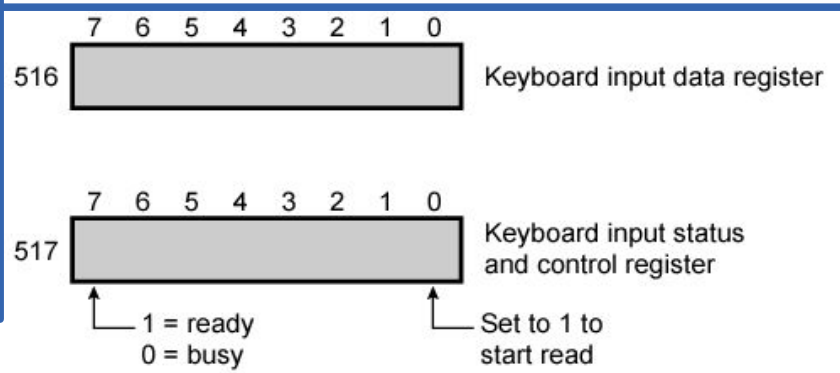
Mapeo de E/S y E/S aislada

- Correspondencia en memoria (Memory mapped I/O)
  - ✔ Dispositivos y memoria comparten el espacio de direcciones.
  - ✔ I/O es como escribir/leer en la memoria.
  - ✔ No hay instrucciones especiales para I/O
    - Ya se dispone de muchas instrucciones para la memoria
- Isolated I/O (Aislada, uso de Puertos de E/S)
  - ✔ Espacio separado de direcciones
  - ✔ Se necesitan líneas de I/O. Puertos de E/S
  - ✔ Instrucciones especiales
    - Conjunto Limitado

Memory Mapped and Isolated I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

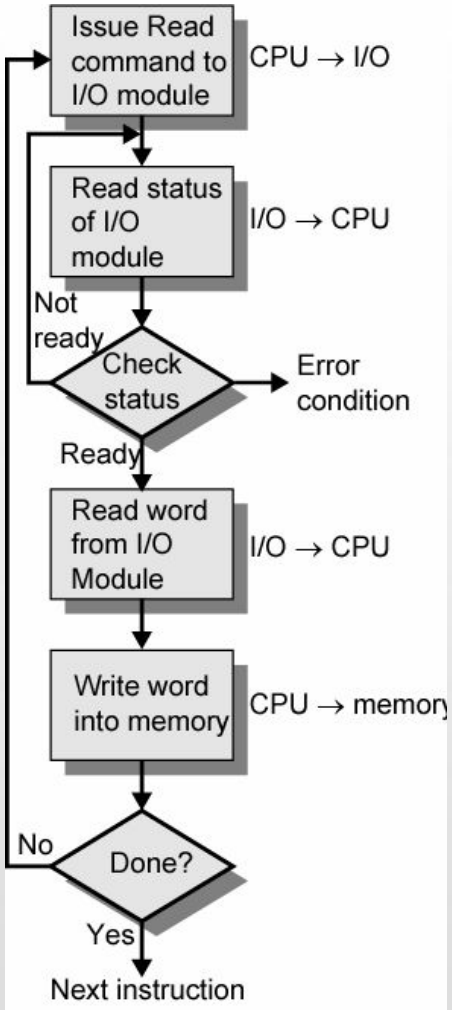
(b) Isolated I/O

```
En MS-DOS
----------------------
o 70 02
i 71
<retorna los minutos>

o 70 00
i 71
<retorna los segundos>
```

Técnicas de I/O - Programada

- CPU tiene control directo sobre la I/O
  - ✔ Controla el estado
  - ✔ Comandos para leer y escribir
  - ✔ Transfiere los datos
- CPU espera que el componente de I/O complete la operación
- Se desperdician ciclos de CPU

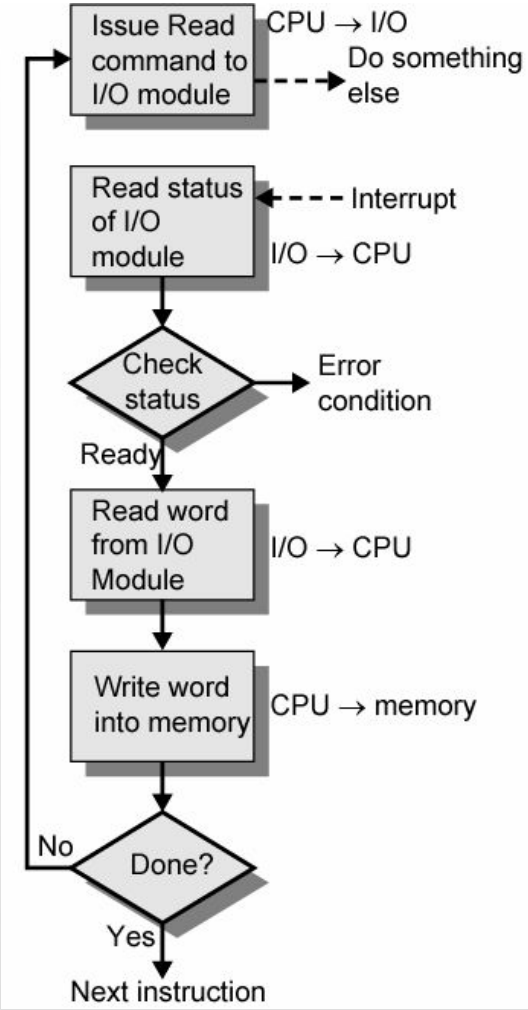


Polling

- En la I/O Programada, es necesario hacer polling del dispositivo para determinar el estado del mismo
  - ✔ Listo para recibir comandos
  - ✔ Ocupado
  - ✔ Error
- Ciclo de “Busy-wait” para realizar la I/O
- Puede ser muy costoso si la espera es muy larga

Técnicas de I/O - Manejada por Interrupciones

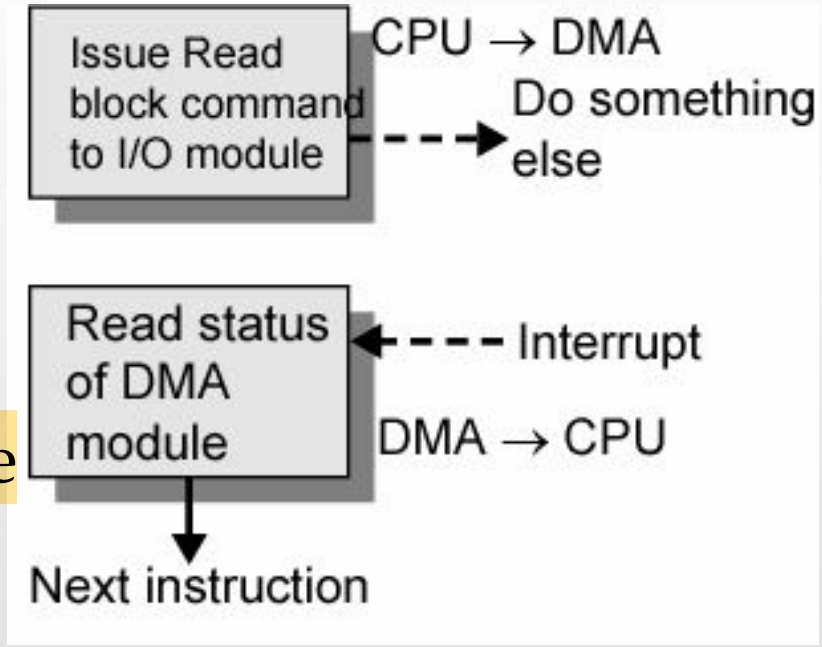
- Soluciona el problema de la espera de la CPU
- La CPU no repite el chequeo sobre el dispositivo
- El procesador continúa la ejecución de instrucciones
- El componente de I/O envía una interrupción cuando termina



Técnicas de I/O - DMA

DMA (Direct Memory Access)

- Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo
- El procesador es interrumpido luego de que el bloque entero fue transferido.



Pasos para una transferencia DMA

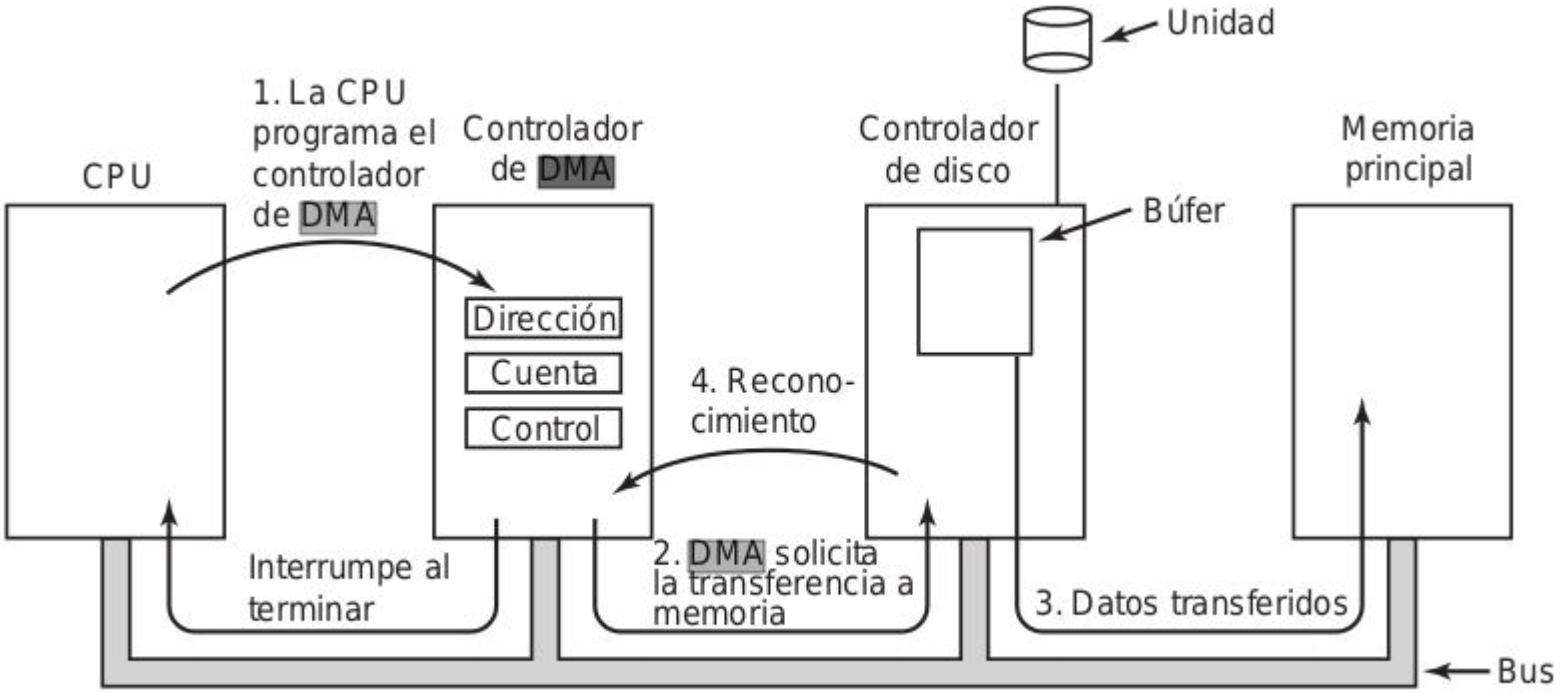


Figura 5-4. Operación de una transferencia de DMA.