Arquitectura de computadores I

Entrada/Salida

Problemas

- Gran variedad de periféricos
  - Enviar diferentes tamaños de datos
  - Diferentes velocidades
  - Diferentes formatos
- Más lento que CPU y RAM
- Necesita módulos de entrada/salida (E/S)

Módulos de entrada/salida

- Interface de CPU a memoria
- Interface a uno o más periféricos

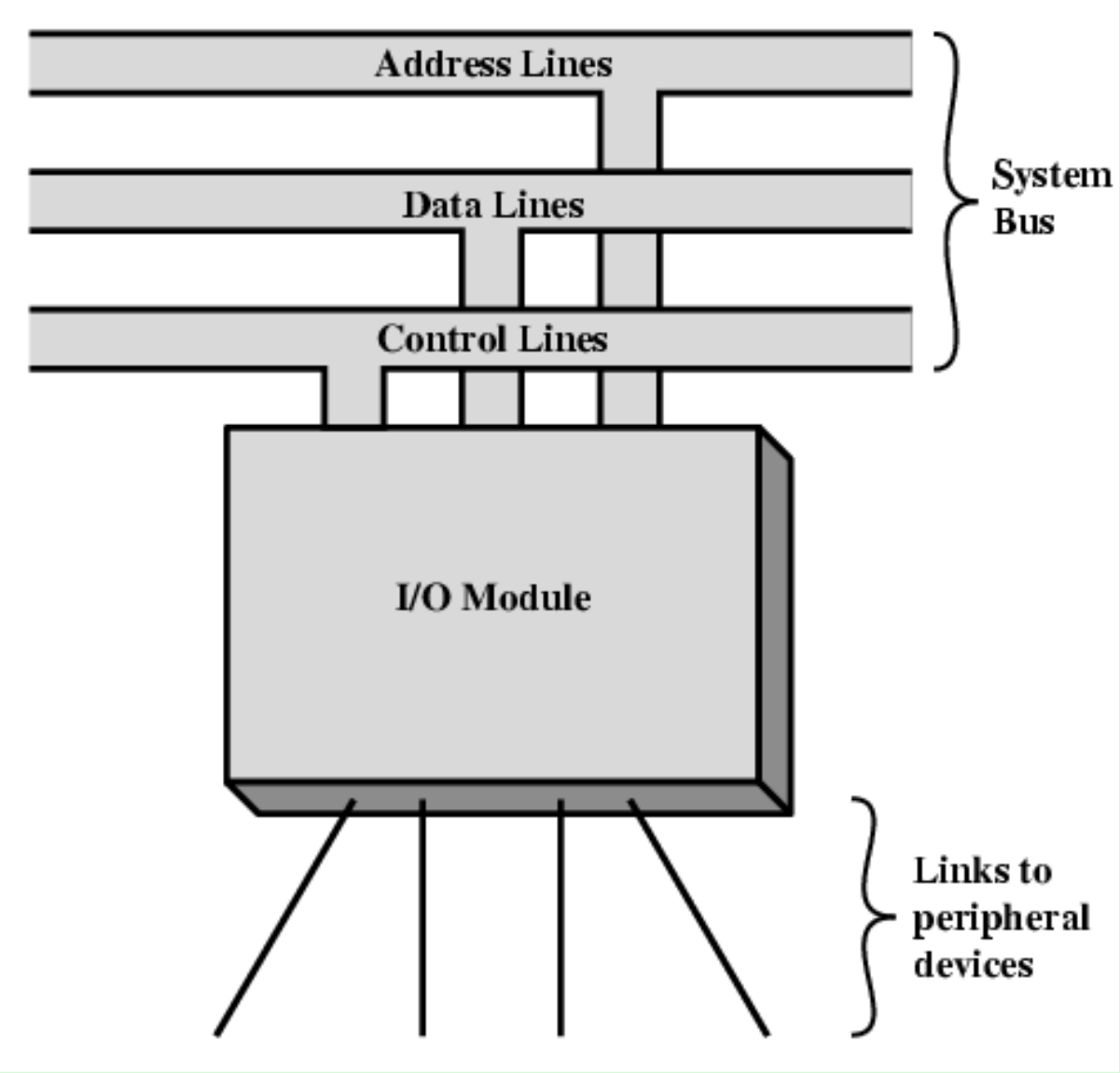
Ejemplo módulo E/S Linux

```
 */

#include <linux/init.h>           // Macros used to mark up functions e.g. __init __exit
#include <linux/module.h>         // Core header for loading LKMs into the kernel
#include <linux/device.h>         // Header to support the kernel Driver Model
#include <linux/kernel.h>         // Contains types, macros, functions for the kernel
#include <linux/fs.h>             // Header for the Linux file system support
#include <asm/uaccess.h>          // Required for the copy to user function
#define  DEVICE_NAME "ebbchar"    ///< The device will appear at /dev/ebbchar using this value
#define  CLASS_NAME  "ebb"        ///< The device class -- this is a character device driver

MODULE_LICENSE("GPL");            ///< The license type -- this affects available functionality
MODULE_AUTHOR("Derek Molloy");    ///< The author -- visible when you use modinfo
MODULE_DESCRIPTION("A simple Linux char driver for the BBB");  ///< The description -- see modinfo
MODULE_VERSION("0.1");            ///< A version number to inform users

static int    majorNumber;                  ///< Stores the device number -- determined automatica
static char   message[256] = {0};           ///< Memory for the string that is passed from userspa
static short  size_of_message;              ///< Used to remember the size of the string stored
static int    numberOpens = 0;              ///< Counts the number of times the device is opened
static struct class*  ebbcharClass  = NULL; ///< The device-driver class struct pointer
static struct device* ebbcharDevice = NULL; ///< The device-driver device struct pointer

// The prototype functions for the character driver -- must come before the struct definition
static int     dev_open(struct inode *, struct file *);
static int     dev_release(struct inode *, struct file *);
static ssize_t dev_read(struct file *, char *, size_t, loff_t *);
static ssize_t dev_write(struct file *, const char *, size_t, loff_t *);

/** @brief Devices are represented as file structure in the kernel. The file_operations structure
 *  /linux/fs.h lists the callback functions that you wish to associated with your file operations
 *  using a C99 syntax structure. char devices usually implement open, read, write and release cal
 */
static struct file_operations fops =
{
   .open = dev_open,
   .read = dev_read,
   .write = dev_write,
   .release = dev_release,
```

Ejemplo módulo E/S Windows

```
/**********************************************************************
 *
 *  DriverEntry
 *
 *    This is the entry point for this video miniport driver
 *
 **********************************************************************/
ULONG DriverEntry(PVOID pContext1, PVOID pContext2)
{
    VIDEO_HW_INITIALIZATION_DATA hwInitData;
    VP_STATUS vpStatus;
    /*
     * The Video Miniport is "technically" restricted to calling
     * "Video*" APIs.
     * There is a driver that encapsulates this driver by setting your
     * driver's entry points to locations in itself. It will then
     * handle your IRP's for you and determine which of the entry
     * points (provided below) into your driver that should be called.
     * This driver however does run in the context of system memory
     * unlike the GDI component.
     */
    VideoPortZeroMemory(&hwInitData,
                        sizeof(VIDEO_HW_INITIALIZATION_DATA));
    hwInitData.HwInitDataSize  = sizeof(VIDEO_HW_INITIALIZATION_DATA);
    hwInitData.HwFindAdapter            = FakeGfxCard_FindAdapter;
    hwInitData.HwInitialize             = FakeGfxCard_Initialize;
    hwInitData.HwStartIO                = FakeGfxCard_StartIO;
    hwInitData.HwResetHw                = FakeGfxCard_ResetHW;
    hwInitData.HwInterrupt              = FakeGfxCard_VidInterrupt;
    hwInitData.HwGetPowerState          = FakeGfxCard_GetPowerState;
    hwInitData.HwSetPowerState          = FakeGfxCard_SetPowerState;
    hwInitData.HwGetVideoChildDescriptor =
                                    FakeGfxCard_GetChildDescriptor;
    vpStatus = VideoPortInitialize(pContext1,
                                    pContext2, &hwInitData, NULL);
    return vpStatus;
}
```

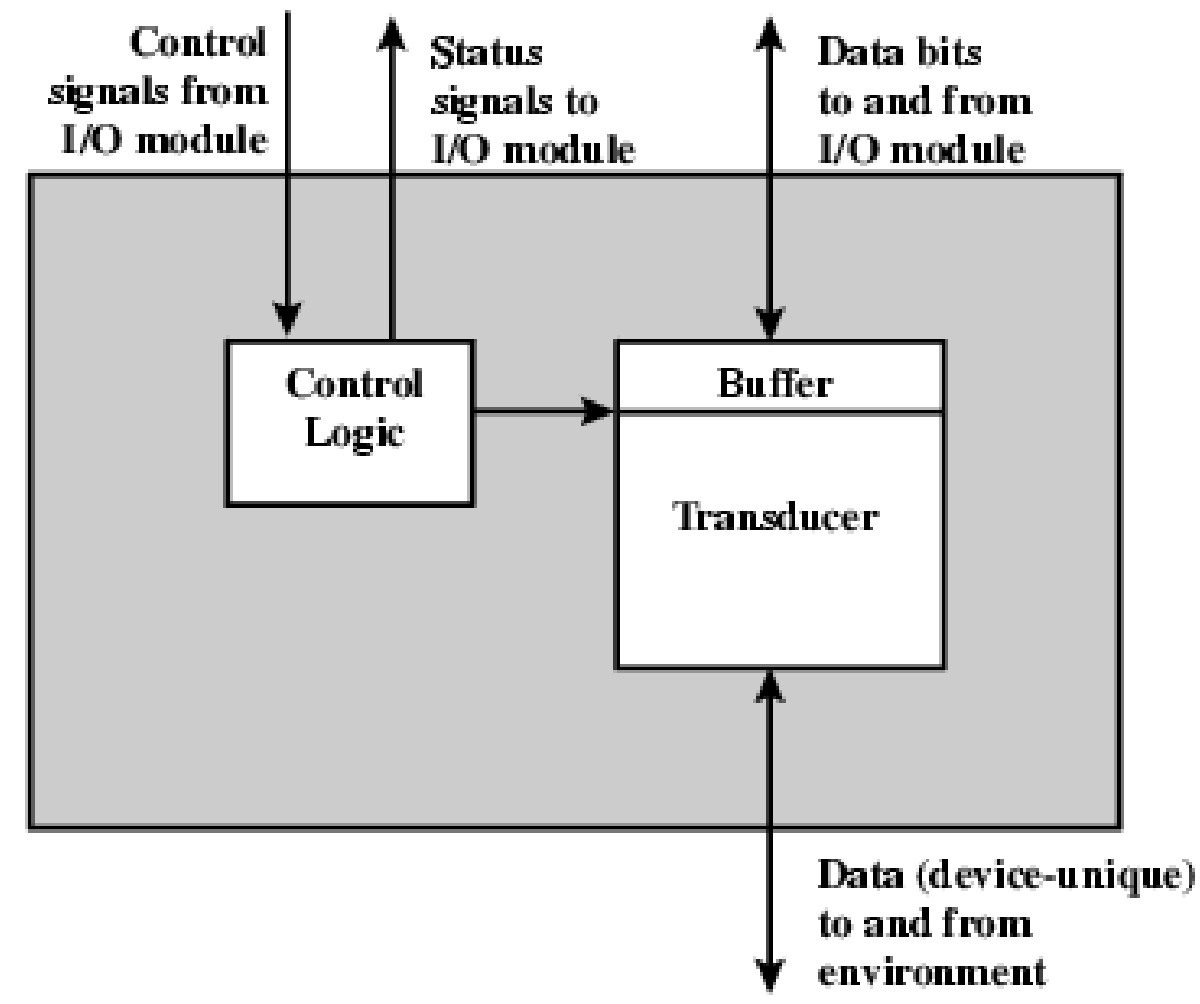
Modelo genérico de interface E/S



Dispositivos externos

- Legible para humanos
  - Pantalla, impresora, teclado
- Legile para la máquina
  - Monitoreo y control
- Comunicación
  - Modém
  - Interface de red (NIC)

Diagrama de bloques para dispositivo externo



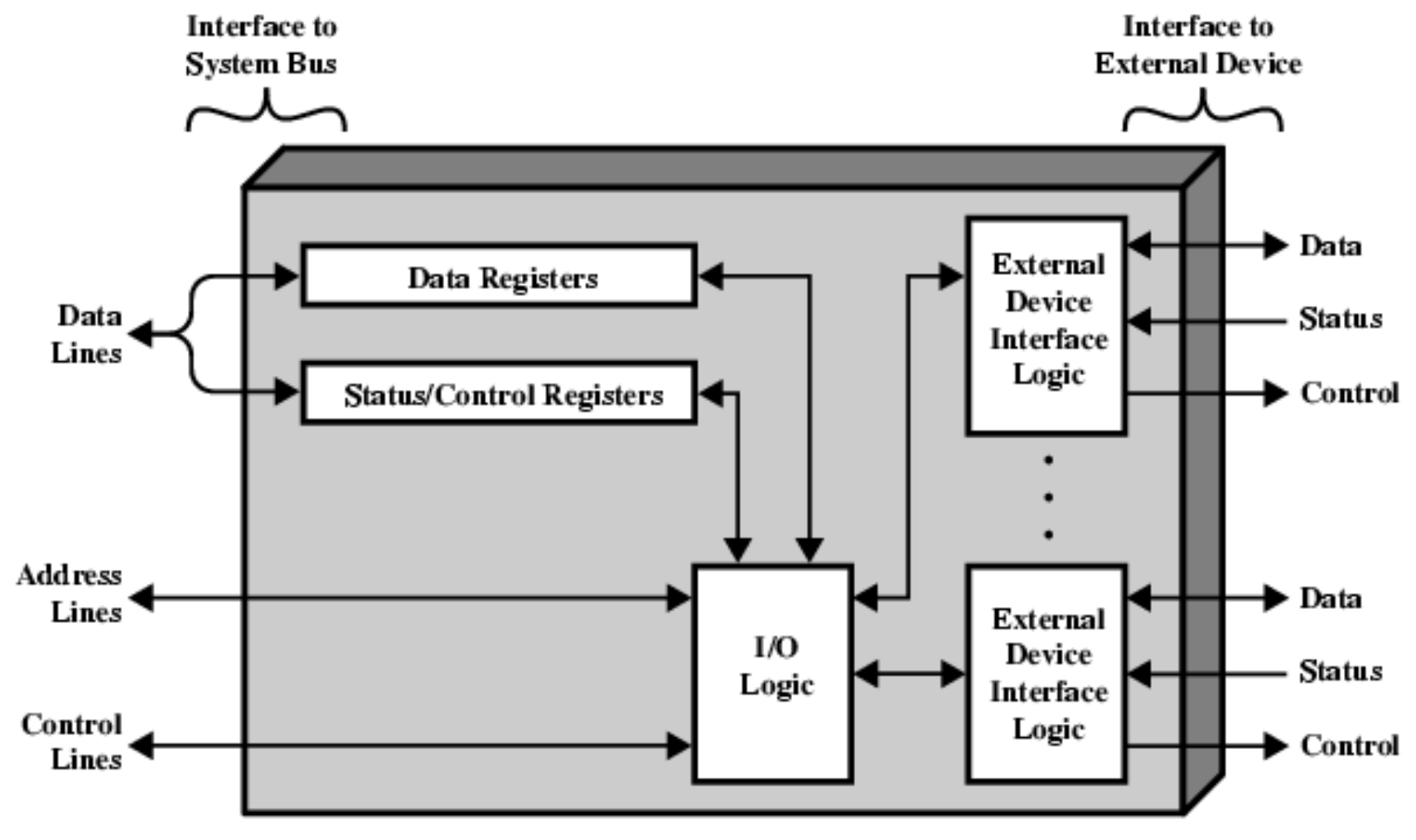
Función de módulo E/S

- Control y temporización
- Comunicación con la CPU
- Dispositivo de comunicación
- Almacenamiento de datos (Buffer)
- Detección de errores

Pasos de E/S

- CPU verifica estado del módulo E/S
- Módulo E/S retorna estado
- Si está listo, la CPU envia los datos
- Módulo E/S obtiene la información desde el dispositivo
- Módulo E/S transfiere la información a la CPU

Diagrama módulo E/S



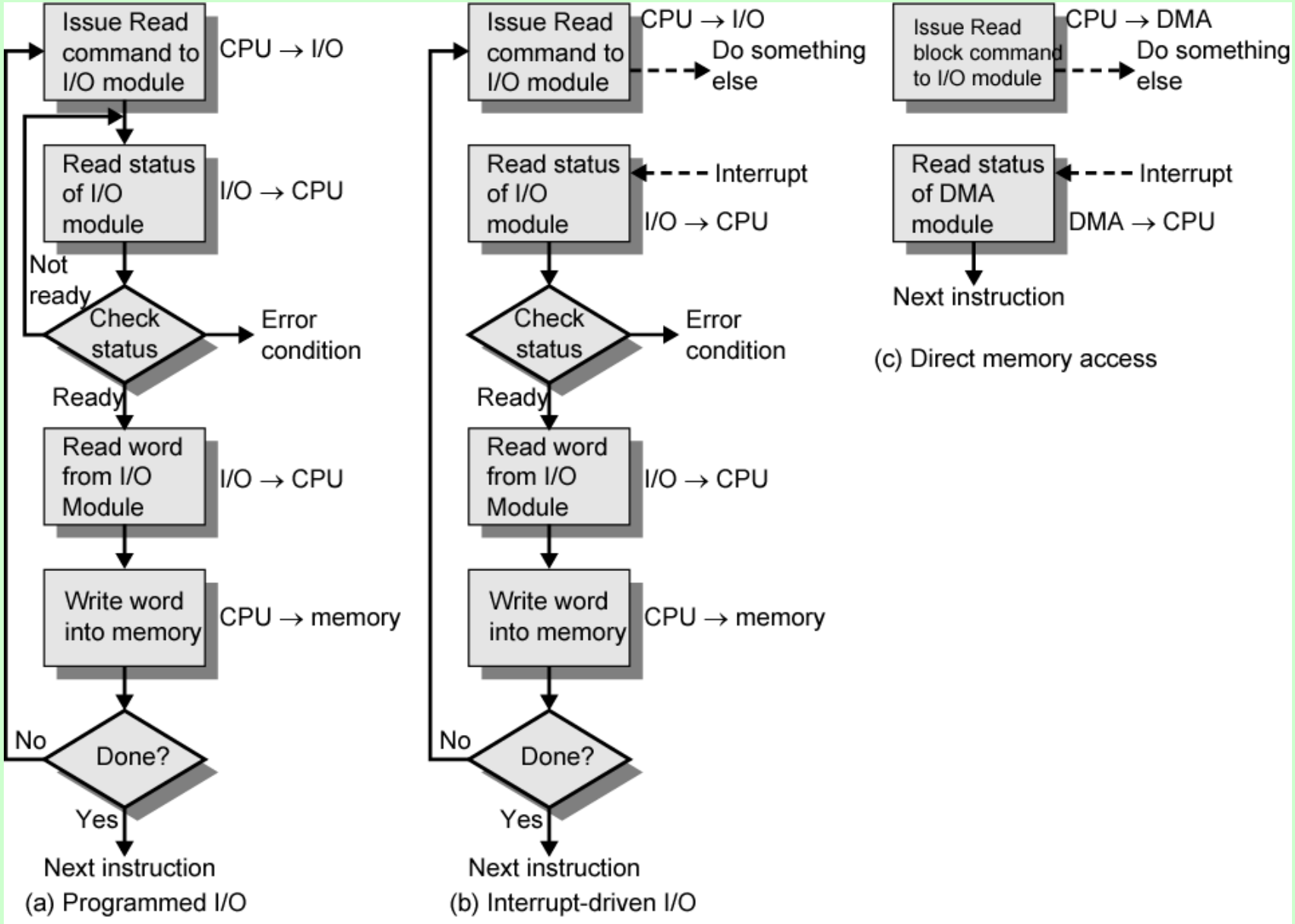
Decisiones módulo E/S

- Ocultar o mostrar propiedaes del dispositivo a la CPU
- Soporte uno o más dispositivos
- Controla las funciones del dispositivo
- Toma las decisiones del sistema
  - Ejemplo. Unix trata todo como si fuera un archivo
    - /dev/sdax (Disco duro)
    - /dev/cdrom (CD)
    - /dev/usb/xxxx (USB)

Técnicas de E/S

- Programada
- Mediante interrupciones
- Acceso directo a memoria (DMA)

Técnicas de E/S



E/S Programado

- CPU tiene control directo sobre E/S
  - Revisar estado
  - Comandos de entrada/salida
  - Transferir datos
- CPU espera por módulo E/S para realiza la operación
- Gasta tiempo de CPU

E/S Programado: Detalles

- CPU requiere operación E/S
- Módulo E/S realiza operación
- Módulo E/S establece estado
- CPU revisa periodicamente el estado
- Módulo E/S no informa a la CPU directamente
- Módulo E/S no lanza interrupción en CPU
- CPU realiza otras tareas si el dispositivo está ocupado

Comandos E/S

- CPU envía dirección
  - —Identifica módulo
- CPU envia control
  - —Control: Decir que hacer el módulo
    - Ejemplo: rotar disco
  - —Prueba: Revisa estado
    - Ejemplo: ¿Encendido? ¿Error?
  - —Lectura/Escritura
    - Módulo transfiere datos vía buffer desde o hacia el dispositivo

Direccionamiento dispositivos E/S

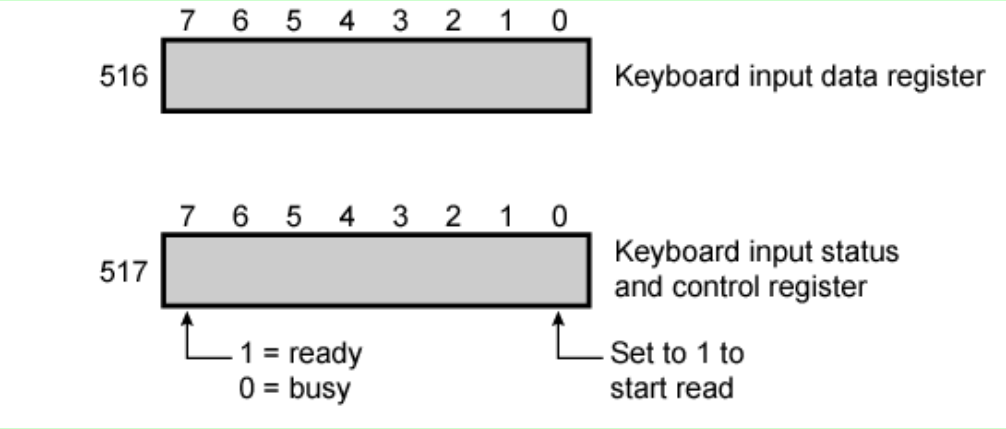
- Bajo E/S programado la transferencia de datos es similar a un acceso de memoria
- Cada dispositivo tiene un identificador único
- CPU envía información al identificador de Hardware

```
$ lspci
00:00.0 Host bridge: Intel Corporation 82G35 Express DRAM Controller (rev 03)
00:02.0 VGA compatible controller: Intel Corporation 82G35 Express Integrated Graphics Controller (r
00:02.1 Display controller: Intel Corporation 82G35 Express Integrated Graphics Controller (rev 03)
00:19.0 Ethernet controller: Intel Corporation 82566DC Gigabit Network Connection (rev 02)
00:1a.0 USB controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #4 (rev 02)
```

Mepeo de E/S

- Memoria mapeada
  - Dispositivos y memoria comparten un espacio de direcciones
  - Lectura/Escritura en E/S es muy similar que en memoria
  - No se requiere comandos especiales
- Isolado
  - Separa espacios de direcciones
  - Necesita líneas de E/S o Memoria
  - Comandos especiales para E/S

E/S Memoria mapeada e Isolado



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

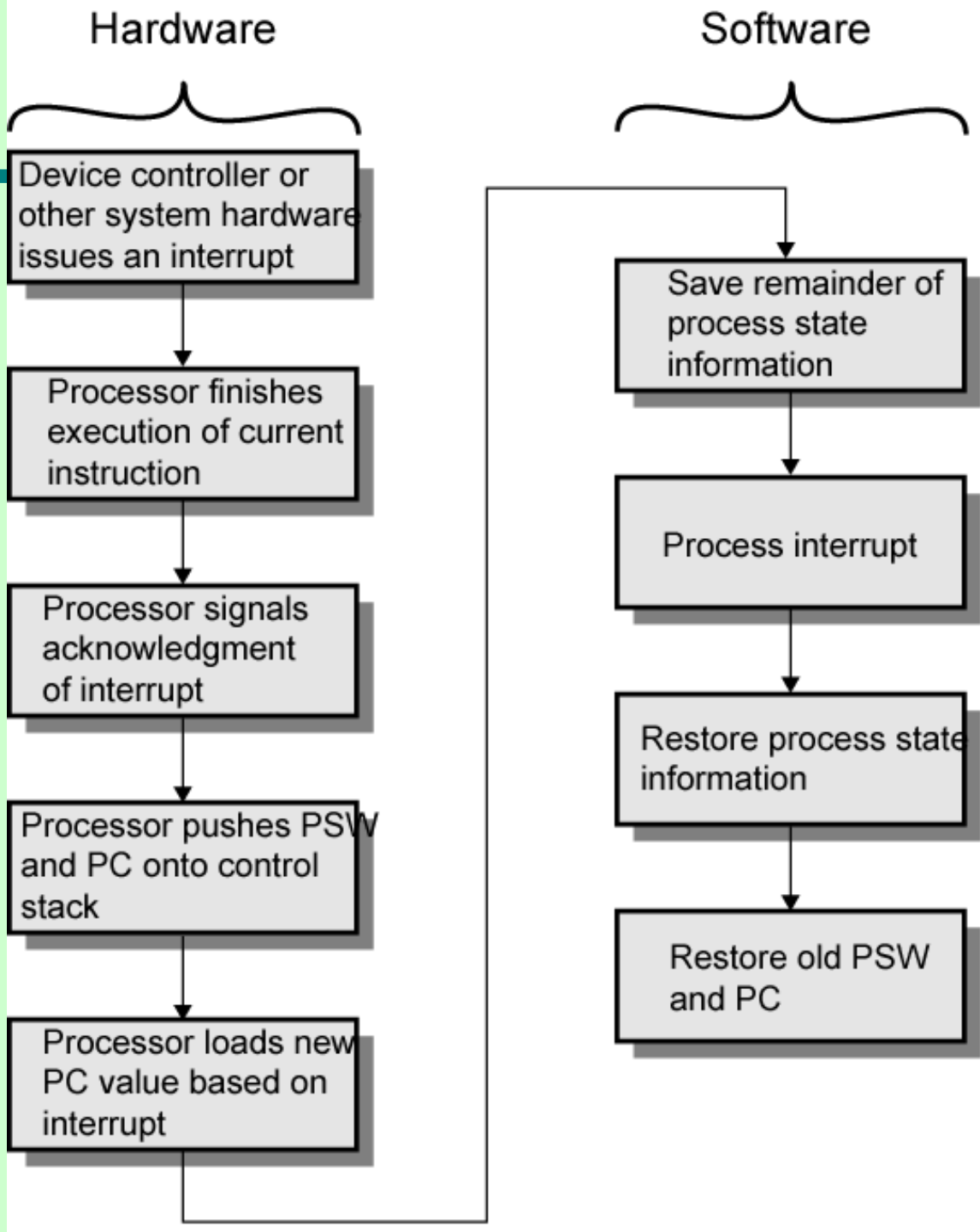
E/S mediante interrupciones

- Ordena a la CPU esperar
- No hay chequeos al dispositivo repetidos desde la CPU
- Módulo E/S realiza la interrupción cuando está listo

E/S mediante interrupciones

- La CPU envía comando de lectura
- Módulo E/S envía los datos desde el periférico al buffer
- Módulo E/S envía interrupción a la CPU
- La CPU requiere los datos
- Módulo E/S envía los datos

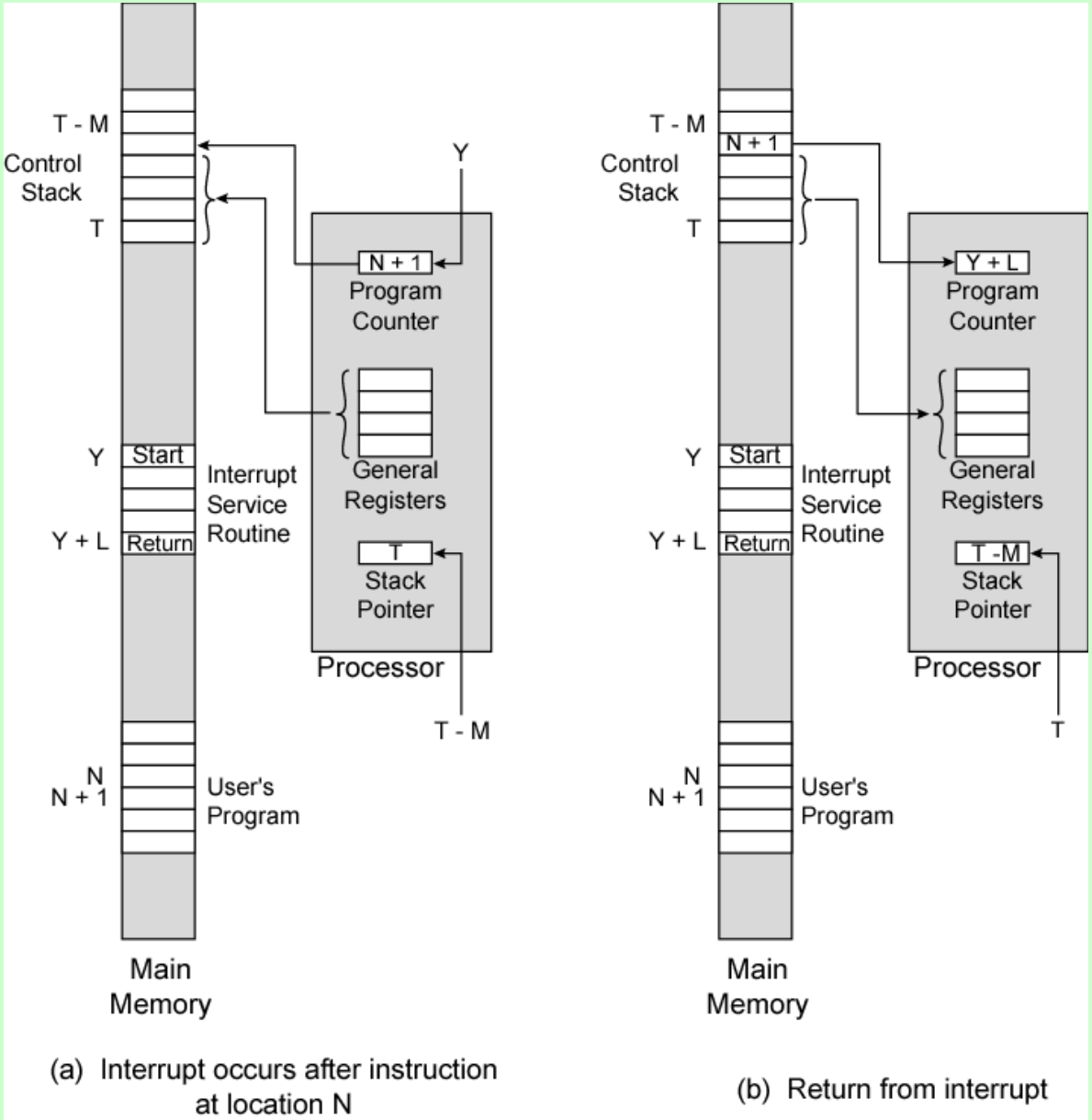
Proceso con Interrupción



Operaciones desde la CPU

- Enviar comando de lectura
- … Hacer otra cosa ….
- Revisar si hay interrupción en cada ciclo de instrucción
- Si hay interrupción:
  - Almacene contexto (Registros)
  - Proceso de interrupción
    - Capte datos y almacene

Cambios en memoria y registros para una interrupción



Problemas de diseño

- ¿Como se puede identificar el módulo que envia la interrupción?
- ¿Que hace cuando hay multiples interrupciones?
  - Ejemplo: Una interrupción puede ser interrumpida.

Identificando el módulo de interrupción

- Diferente línea para cada módulo
  - —Número limitado de dispositivos
- Sondeo por Software
  - —CPU pregunta por cada módulo
  - —Lento

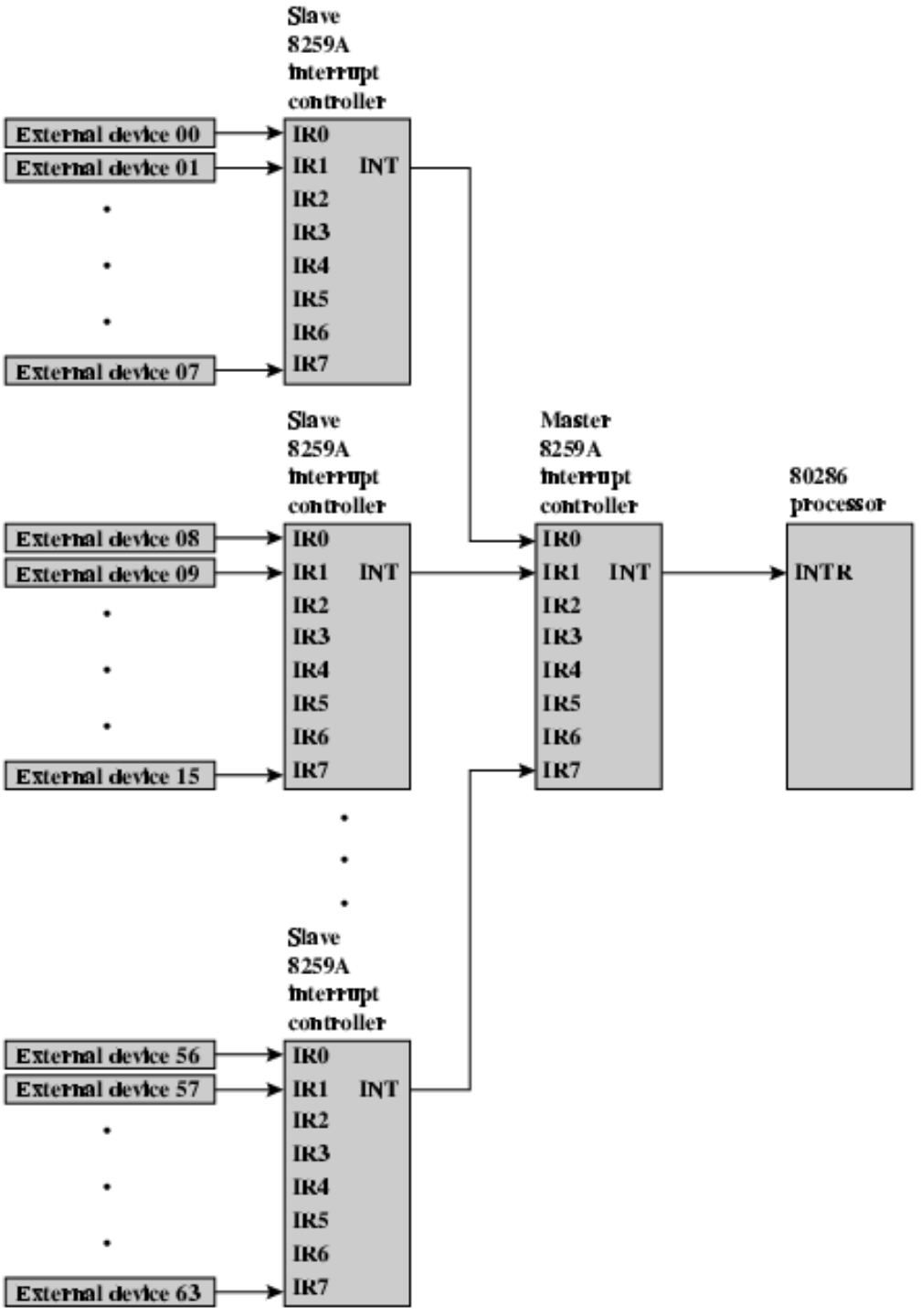
Identificando el módulo de interrupción

- Sondeo por Hardware
  - El módulo envía un vector de identificación por el Bus
  - CPU usa el vector para identificar el módulo
- Maestro de bus
  - Módulo reclama el uso del bus para enviar la interrupción
  - Ejemplo. PCI & SCSI

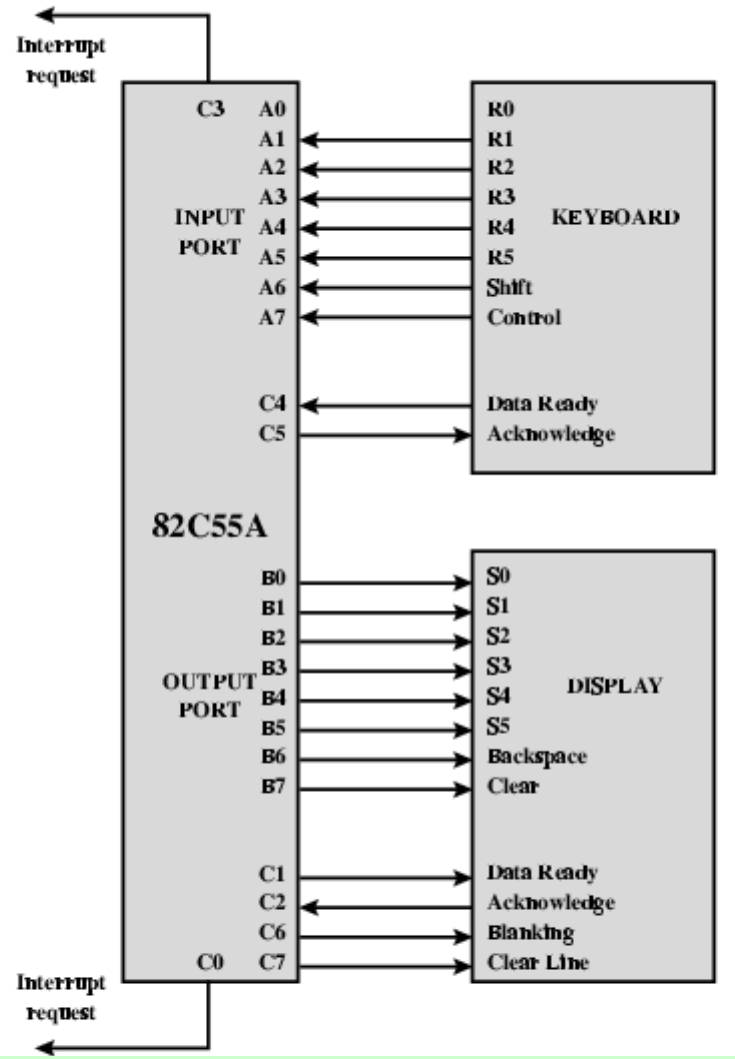
Interrupción de módulo

- Cada línea de interrupción tiene una prioridad
- Líneas de mayor prioridad pueden interrumpir a líneas con menor prioridad

Controlador de interrupciones 82C59A



Interfaces de teclado y tarjeta de videos al 82C55A



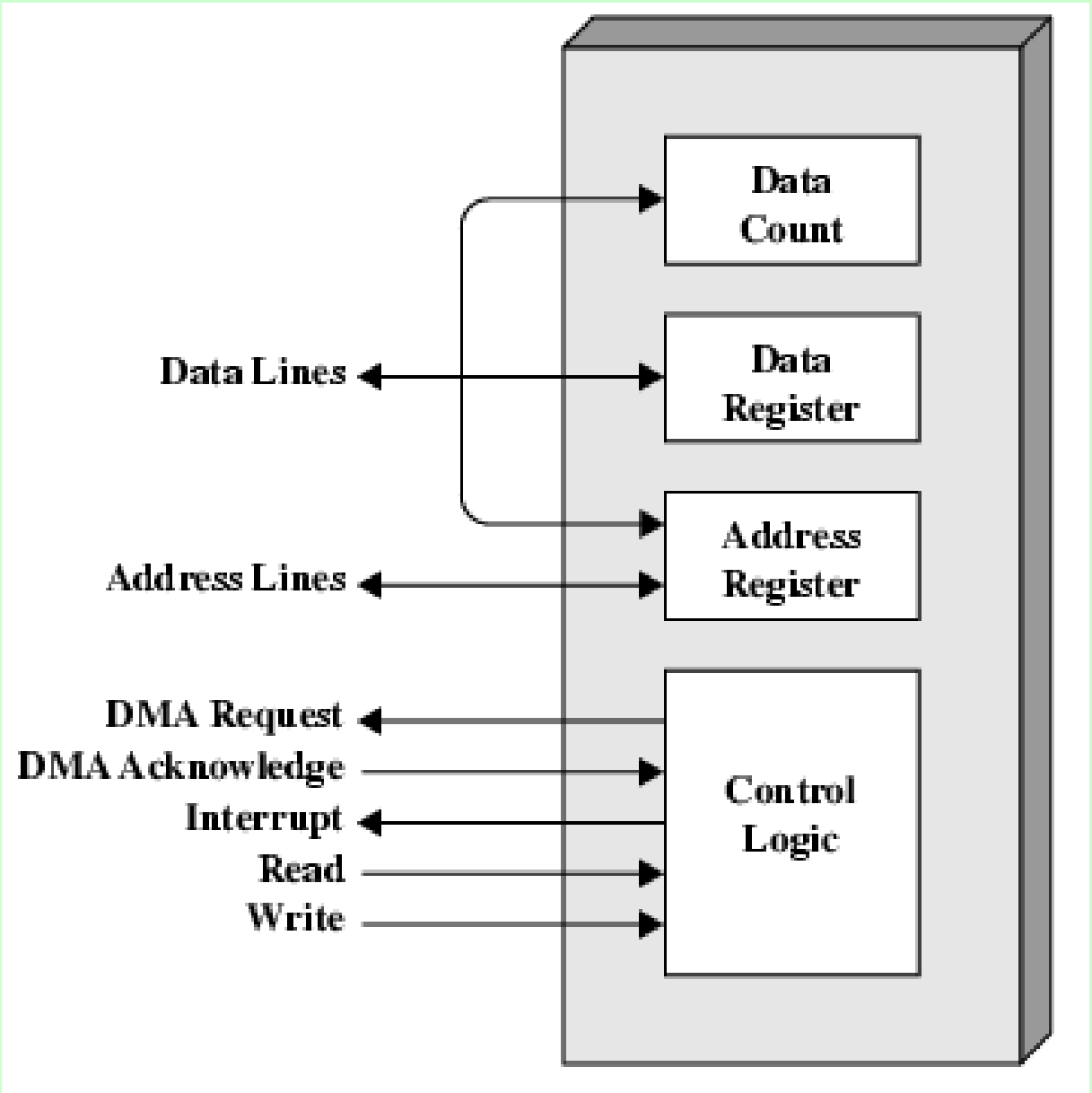
Acceso directo a memoria (DMA)

- Las interrupciones dirigidas y el E/S programado requiere intervención activa de la CPU
  - Transferencia es limitado
  - El CPU gasta recursos y tiempo
- DMA es la respuesta

Funcionamiento de DMA

- Módulo adicional en el Bus
- DMA toma el control desde la CPU hasta el E/S

Diagrama de DMA típico



Operación de DMA

- CPU habla con el controlador de DMA
  - Lectura/Escritura
  - Dirección de dispositivo
  - Direcciones en memoria
  - Tamaño de datos transferidos
- CPU puede hacer otras cosas
- Controlador DMA oferta una transferencia
- Controlador DMA envía una interrupción cuando termina la transferencia

Gracias

- Próxima clase
  - Sistemas operativos