Introducción

Los dispositivos de E/S se pueden dividir en 2 categorías:

- **Dispositivos de Bloque**: almacena información en bloques de tamaño fijo, cada uno con su propia dirección. Se pueden leer o escribir cada bloque de manera independiente de los demás. Además las transferencias se realizan en unidades de uno o más bloques completos (pero siempre consecutivos). **[Dispositivos de Disco, USB...]**
- **Dispositivos de Carácter**: envía o acepta un flujo de caracteres sin importar la estructura del bloque. No es direccionable (ergo, no tiene operación de búsqueda). **[Impresoras, ratones, auriculares...]**

Cabe destacar que esta **clasificación no es perfecta** ya que algunos dispositivos no se adaptan (como el reloj, que simplemente va generando interrupciones cada cierto tiempo).

Velocidades

Cada dispositivo tiene una velocidad de transferencia de datos, lo que impone una presión al software para planificarlos.

El SO y la E/S

El Sistema Operativo controla los dispositivos de E/S, para ello debe:

1. **Emitir comandos**.
2. **Captar interrupciones**.
3. **Manejar errores**.

Controladores de Dispositivos

- El componente **electrónico** se llama **controlador de dispositivo** o **adaptador**.
- El componente **mecánico** es el **dispositivo en sí**

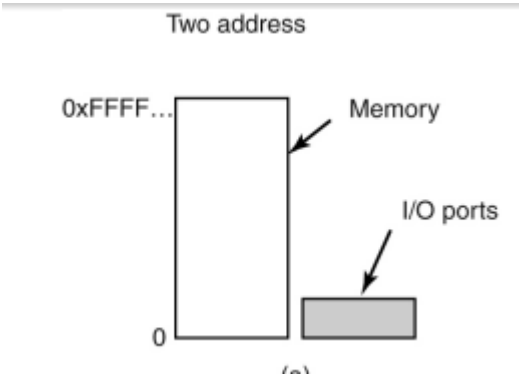
E/S por Asignación de Memoria

Cada controlador tiene unos cuantos registros que se utilizan para comunicarse con la CPU. Al escribir en ellos, el SO puede hacer que un dispositivo envíe o acepte datos (también puede leerlos y obtener información del dispositivo).

```
Además, existen dispositivos con un buffer de datos (como las pantallas) para
almacenar los datos antes de utilizarlos "oficialmente" en pantalla
```

En este punto, nos preguntamos cómo se comunica la CPU con estos regístros y búferes. Existen dos alternativas:

1. A cada registro de control se le asigna un número **Puerto E/S** (el conjunto de puertos forman el **Espacio de Puertos E/S**). Este método permite que **únicamente el SO puede utilizarlo** mediante una instrucción especial de E/S:




```
IN  REG, PUERTO     // Lee el registro de control PUERTO y lo almacena en el
registro de cpu REG

OUT PUERTO, REG     // Escribe el contenido de REG en un registro de control
PUERTO


// En estas instrucciones, las direcciones de memoria se van a referir siempre a
un registro de CPU y un registro de CONTROL
// Por lo tanto estas dos instrucciones se refieren a cosas completamente
distintas:

IN  R0, 4   // Aquí el 4 es un registro de CONTROL (de E/S)
MOV R0, 4   // Aquí el 4 es un registro de la memoria de la CPU
```

2. **E/S con Asignación de Memoria** Se asignan todos los Registros de CONTROL al espacio de memoria. A cada Registro de CONTROL se le asigna una dirección de memoria única a la cual no hay memoria asignada.

3. **Esquema Híbrido** donde los búferes de datos de E/S se asignan a memoria mientras que los Puertos de E/S están separados para los Registros de CONTROL.

¿Cómo funcionan estos esquemas?

- Cuando la CPU quiere **leer** una palabra:

    - Coloca la **dirección** en el Bus de Direcciones.
    - Impone la **señal Read** en el Bus de Control.
    - Un **bit adicional** especifica si es espacio de E/S o de Memoria.

    Responde a esta solicitud la Memoria o la E/S (dependiendo de donde esté).

    Si se da el Esquema 2 (E/S con Asignación de Memoria) los dispositivos de E/S y la Memoria comparan la dirección con el rango de direcciones que le corresponde a cada uno: Responde el que lo tenga en su Rango.

Ventajas de E/S con Asignación de Memoria

1. Las instrucciones como **IN y OUT no se pueden ejecutar en lenguajes** como C o C++. Es necesario recurrir a ensamblador. Sin embargo, al usar **este método no es necesario que los usemos y se puede escribir todo el controlador del dispositivo en C** (los registros de control que se encuentran en la memoria son variables de memoria que se pueden acceder en C).

2. Además, **no se necesitan protecciones especiales** para evitar que los procesos hagan operaciones E/S. Ya que el **SO no lo coloca en las direcciones virtuales de ningún proceso** por lo que son inaccesibles.

3. Por último, cada instrucción que puede hacer **referencia a memoria**, también puede hacer **referencia a los registros de Control**. Por lo tanto, para evaluar un registro de Control es necesario **una sola instrucción**:

   ciclo: TEST PUERTO_4 BEQ listo BNEQ ciclo

Desventajas de E/S con Asignación de Memoria

1. En la última ventaja también se encuentra una **desventaja con respecto a la Caché**. El caso es que sería **desastroso colocar en Caché un registro de Control de E/S**: cuando una variable se toma en caché, las demás veces que se hagan referencia a ella no se va a volver a evaluar de nuevo así que pueden tomar el valor sin preguntar al dispositivo.

Esto se soluciona con una **Caché Selectiva** (por páginas) que añade **complejidad** al Hardware y SO.

2. Todos los controladores deben examinar las direcciones del bus para ver cuales deben responder, lo que es fácil de implementar en un sólo bus (se analizan todas y se responden a las que se deban responder). Si son **Buses Separados** los dispositivos E/S no ven todas las direcciones de memoria (la solución es **enviar todas las referencias a memoria, si esta no responde se envían a otros buses**).

DMA (Acceso Directo a Memoria)

La CPU necesita direccionar los Controladores de Dispositivos para intercambiar datos (independientemente del esquema de memoria que se utilice).

La CPU puede solicitar datos de un Controlador bit por bit pero es muy lento: **DMA libera a la CPU del intercambio de Datos**.

La solución es usar un Controlador DMA:

- Puede estar integrado en un Controlador de Dispositivo (pero entonces se necesita un DMA por cada Dispositivo).
- Normalmente se utiliza un solo Controlador DMA que regule todos los dispositivos.

El DMA tiene acceso al bus de manera independiente a la CPU.

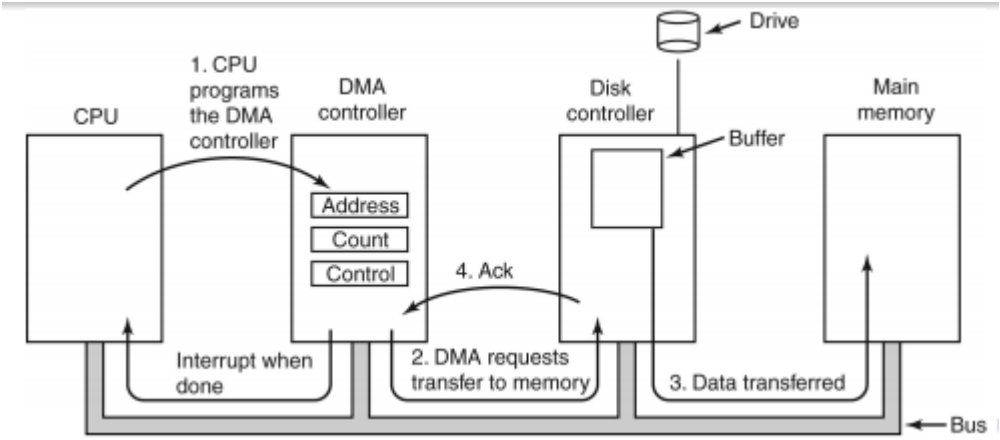
Contiene registros en los que la CPU puede leer y escribir: 1. **Registro de Dirección de Memoria**: indica el Dirección de Memoria. 2. **Registros de Control**: indican el puerto E/S, tipo de transferencia (R/W) y los bytes por ráfaga de transmisión. 3. **Contador de Bytes**.

Lectura de Disco sin DMA

1. El **Controlador de Disco** lee un bloque de la **Unidad** (Disco), lo coloca en el **Buffer Interno** y **verifica que no haya errores**.
2. El **Controlador de Disco** produce una INTERRUPCIÓN.
3. El **SO lee el bloque del Buffer** palabra a palabra y lo **almacena en Memoria Principal** (lo que lleva **varios ciclos**).

Lectura de Disco con DMA

1. El **CPU programa el DMA** (establece *Dirección*, *Cuenta* y *Control*) **+** ejecuta un **comando al Controlador de Disco** que realice el *Paso 1* del anterior apartado.
2. El **Controlador DMA** envía **petición de lectura al Controlador de Disco** mediante el BUS (incluye dirección de memoria en la que debe escribir).
3. En otro Ciclo de BUS se pasan los **datos del Buffer a la Memoria Principal**.
4. Se envía desde el Controlador Disco al Controlador DMA un **ACK**.
5. Mediante el ACK se **comprueba la *Cuenta***, si es 0 está completo -> Se produce una INTERRUPCIÓN a la CPU.



Operación del DMA

Los DMA más complejos pueden manejar varias transferencias a la vez (por lo que necesitan más de un registro interno, etc.). Para cada transferencia se deve utilizar un Controlador de Dispositivo distinto.

MODO RÁFAGA: La CPU para, deja de ejecutar instrucciones. El Controlador del DMA ocupa el BUS colocando palabras y realizando operaciones E/S (puede hacer varias palabras). Es decir, el Controlador pone una **ráfaga de palabras** durante un tiempo.

MODO CICLO: Se permite que cada cierto tiempo el Controlador del DMA "robe" un ciclo o unos pocos ciclos para colocar una palabra de DMA que permitirá realizar alguna operacion de E/S.

Aquí se pueden usar distintos algoritmos pero normalmente van alternándose Instrucciones y ciclos de tal forma que los ciclos ocupan menos ciclos que las instrucciones para dar la sensación de continuidad por parte de la CPU realizando tareas.

MODO BUS COMPARTIDO: Hay una serie de etapas de la instrucción en la que se necesita el **uso del BUS** para leer o escribir en memoria. En las etapas que no se ejecuta el BUS en las instrucciones, se puede usar el BUS para el uso del DMA (etapas de decodificación... Etapas que 100% seguro no se necesitará el BUS).

Repaso de Interrupciones

A nivel de Hardware, las instrucciones funcionan de la siguiente manera: Cuando un dispositivo de E/S ha terminado el trabajo que se le asignó, produce una interrupción.

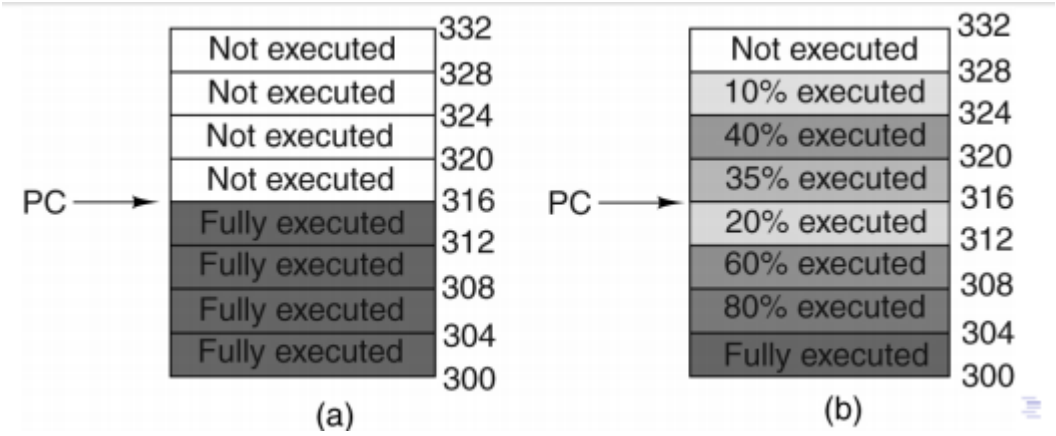
Para ello **impone una señal en una línea de bus que se le haya asignado**. Esta señal es detectada por el chip del controlador de interrupciones en la tarjera principal, que después decide lo que hacer:

```
Interrumpe la CPU.
Vector de Interrupción para obtener la rutina de atención al dispositivo

Guardamos la información de retorno de la rutina en la pila
```

Interrupciones Precisas e Imprecisas

Cuando ocurre una interrupción después de una instrucción, es posible que las instrucciones anteriores no hayan terminado



En las **Interrupciones Precisas** el PC apunta a la instrucción que generó la interrupción y puede reiniciarse más tarde. Además, las instrucciones anteriores se han ejecutado por completo y ninguna de las posteriores se ha comenzado a ejecutar.

Fundamentos del Software E/S

- **Independencia de Dispositivos**: debe ser posible escribir programas que puedan acceder a cualquier dispositivo de E/S sin tener que especificar el dispositivo por adelantado.

  sort salida // No se especifica qué dispositivo, eso es en la ejecución

- **Denominación Uniforme**: el nombre del archivo no debe depender del dispositivo (gracias a los árboles de directorios)

- **Manejo de Errores**: resolverlos lo más cerca del Hardware que sea posible.

  Si el Controlador descubre un error debe arreglarlo él, sino puede, entonces se lo pasa al SOFTWARE del Controlador del Dispositivo

- **Transferencias Síncronas o Asíncronas**: La mayoría de las operaciones de E/S son asíncronas (hay interrupciones que paran la CPU). Sin embargo, los programas de usuario suelen ser síncronos (después de un *read* el proceso se para hasta que los datos estén en el buffer).

  El SO debe hacer que las operaciones asíncronas parezcan de bloqueo para los usuarios

- **Utilización del Buffer**: Es un almacenamiento temporal de los E/S hasta que se lleva al destino final

  En este se realiza el chequeo de errores y a veces operaciones de copia

FORMAS DE LLEVAR A CABO LA E/S

1. **E/S Programada**.
2. **E/S por Interrupciones**.
3. **E/S mediante DMA**.

E/S Programada

La CPU sondea el dispositivo para ver si está listo (**ocupa la CPU hasta completar E/S**)

```
copy_from_user (buffer, p, count);  // Copia del buffer de usuario a buffer de
kernel (p) todo el String

for (i = 0; i < count; i++)         // Bucle en cada caracter
{
    while (*printer_status_reg != READY);   // Comprueba si la impresora está
lista
    *printer_data_register = p[i];          // Pone 1 caracter en el output
}

return_to_user();


>Se comprueba el estado del Reg. Controlador: está listo cuando no está procesando
ningún caracter.
```

E/S por Interrupciones

La CPU realiza la transferencia y **se planifica algún otro proceso**

```
copy_from_user (buffer, p, count);  // Pasa la cadena al buffer kernel
enable_interrupts();                // Permite interrupciones para no pasar
demasiado tiempo en el siguiente bucle

while (*printer_status_reg != READY);   // Comprueba si la impresora está lista
*printer_data_register = p[0];          // Pone 1 caracter en el output
scheduler();                            // Llama al planificador y se ejecuta otro
proceso
```

```
>El proceso que mandó imprimir la cadena se bloquea hasta que se imprima por
completo (va a tener que esperar a que el planificador le vaya dando ciclos)

>Después de que mande imprimir 1 caracter en el output la impresora se queda en
estado != READY y por eso se llama al scheluder. Cuando el estado de la impresora
cambia a READY produce una interrupción


if (count == 0) {       // Si no hay más caracteres en el contador
    unblock_user();     // Desbloqueamos el proceso que mandó imprimir
} else {                // Si tenemos algún caracter en el kernel buffer
    *printer_data_register = p[i];  // Lo imprimimos
    count--;                        // Y bajamos los contadores
    i++;                            // Y subimos el auxiliar...
}

acknowledge_interrupt();    // Reconoce la interrupción
return_from_interrupt();    // Regresa al proceso que estaba ejecutándose antes de
la interrupcion
```

E/S por DMA

En esencia, el DMA es E/S programada, solo que el Controlador DMA realiza todo el trabajo sin molestar la CPU:

```
copy_from_user (buffer, p, count);  // Pasa la cadena al buffer kernel
set_up_DMA_controller();            // Pone la "Direccion", "Cuenta" y "Reg de
Control" y programa la DMA
scheduler();                        // Hace otras tareas mientras tanto la CPU

acknowledge_interrupt();            // Cuando la DMA acaba interrumpe la CPU y
manda el ack
unblock_user();                     // Se desbloquea el llamador
return_from_interrupt();            // Y regresa al proceso que se estaba
ejecutando cuando ocurrió la interrupción

>Con DMA se reduce de una interrupción por cada carácter a una por cada buffer
impreso
```

Obviamente, si la CPU no tiene NADA QUE HACER mientras está trabajando la DMA es una tontería y puede no ser conveniente usar este método pero cae de puto cajón macho, no se ni para qué lo escribo aquí.