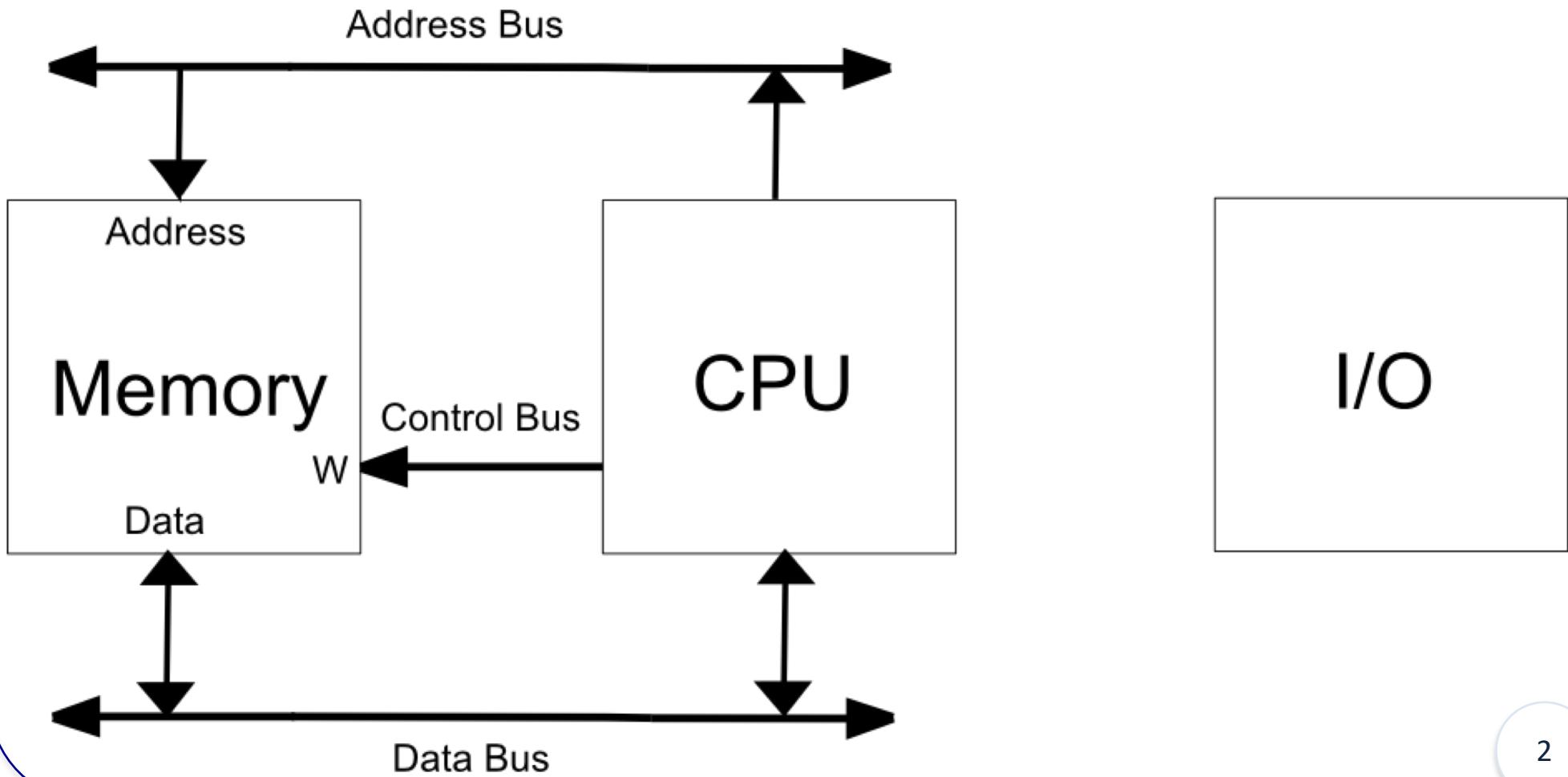


Input / output

Arquitectura de Computadores – IIC2343

Un sistema computacional tiene tres componentes principales:

- CPU, memoria, *input/output (I/O)*

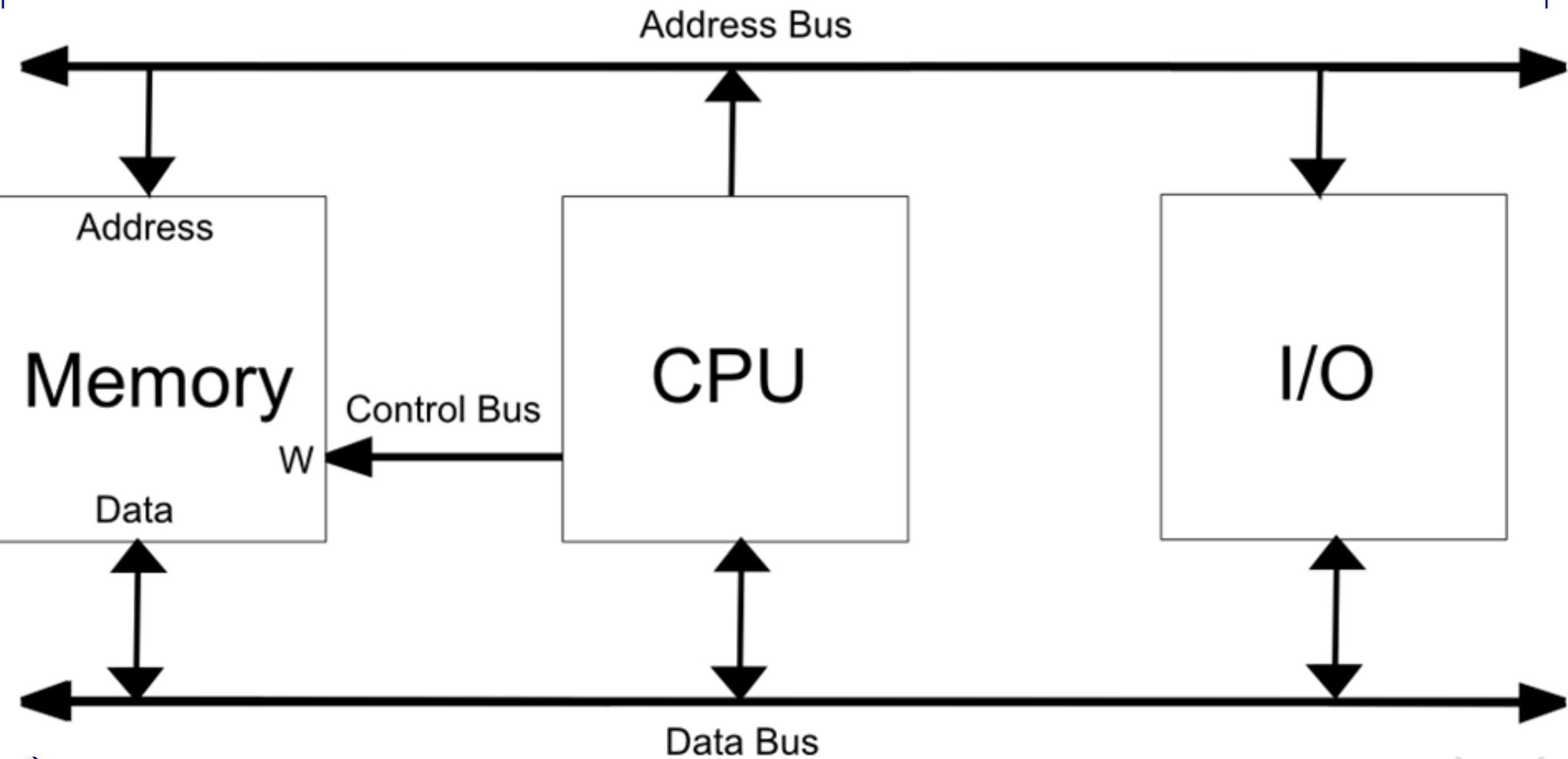


Estas componentes están conectadas mediante **buses**:

- cada **bus** es una colección de alambres paralelos
- transmiten direcciones, datos y señales de control
- pueden ser externos a la CPU —conectándola a la memoria y a los dispositivos de I/O
 - ... o internos —como vimos— conectando el *register file* con la ALU
- un computador moderno tiene múltiples buses

Todos los dispositivos que no sean la CPU o la memoria y que se comuniquen con ellos a través de los buses son llamados **dispositivos de I/O**; p.ej.,

- monitor, teclado, disco duro, impresora, tarjeta de sonido, *modem*



Todo dispositivo de I/O tiene dos partes:

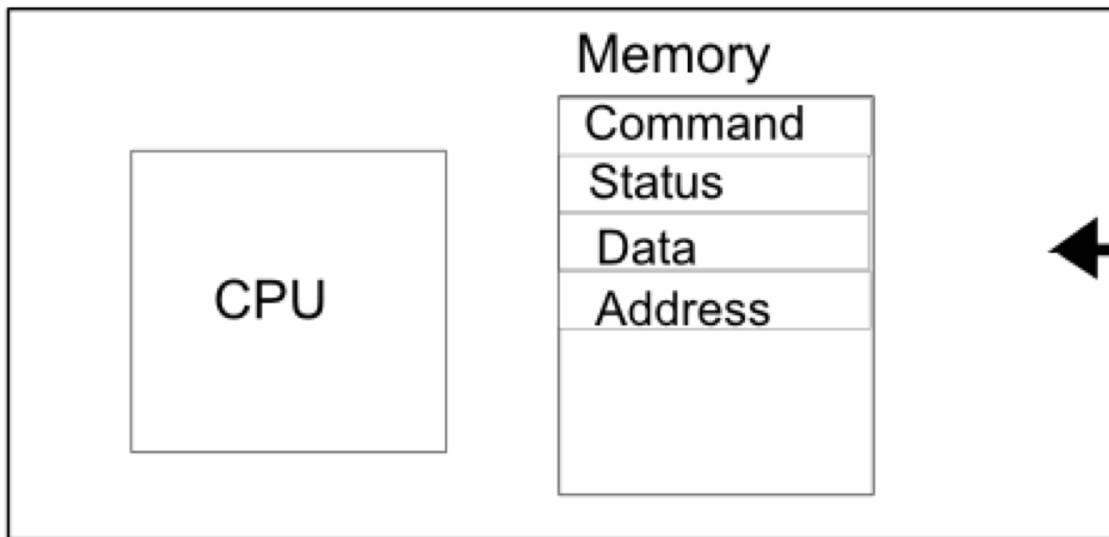
- el **controlador**
 - ... contiene la mayor parte de la electrónica
 - ... normalmente está integrado directamente en la tarjeta madre
 - ... o viene en una tarjeta que se conecta en un slot libre del bus
- el dispositivo propiamente dicho
 - ... formado por elementos electromecánicos
 - ... que realizan las operaciones de interacción

Todo dispositivo de I/O contiene un **controlador**:

- tiene algunos registros para comunicarse con la CPU
- maneja el acceso al bus para el dispositivo (muchos controladores pueden manejar varios dispositivos idénticos)
- controla la parte electromecánica (el dispositivo propiamente tal)

I/O

Controller



Electromechanic

P.ej., si un programa necesita datos desde el disco:

- le da una instrucción al controlador del disco
 - ... y el controlador le da instrucciones a su vez al *drive* del disco
- cuando el *drive* ha encontrado la pista y sector apropiados, le envía los, digamos, 512 bytes del sector —un *stream* de bits— al controlador
- el controlador (re)agrupa los bits en bytes, usando un *buffer* interno
 - ... verifica el *checksum* del bloque de bytes
 - ... y finalmente escribe el bloque en la memoria

¿Cómo tiene acceso la CPU a los registros del controlador?

I/O port:

- a cada registro se le asigna un número de **puerto de I/O**
- el conjunto de todos los puertos de I/O forman el **espacio de puertos de I/O**, al que solo tiene acceso el sistema operativo (está protegido)
- los espacios de direcciones para la memoria y para I/O son diferentes
- la CPU usa instrucciones especiales, p.ej.,
 - ... **in reg, port** (lee el registro **port** del controlador y guarda el valor en el registro **reg** de la CPU) y
 - ... **out port, reg** (escribe el valor del registro **reg** en el registro del controlador)

memory-mapped I/O:

- todos los registros son parte del espacio de direcciones de la memoria del computador
- las direcciones asignadas a los registros están en la parte alta de la numeración (el extremo 0xFFFF)
- la CPU usa instrucciones normales de acceso a memoria

Tres esquemas de I/O:

- I/O programado
- I/O dirigido por interrupciones
- DMA I/O

I/O programado —usado en microprocesadores en sistemas embedidos o de tiempo real— en que la CPU hace todo el trabajo es el más simple

La CPU tiene una sola instrucción de *input* y una sola instrucción de *output*:

- cada una selecciona uno de los dispositivos
- se transfiere un solo carácter entre un determinado registro en la CPU y el dispositivo
- la CPU ejecuta una secuencia fija de instrucciones por cada carácter escrito o leido

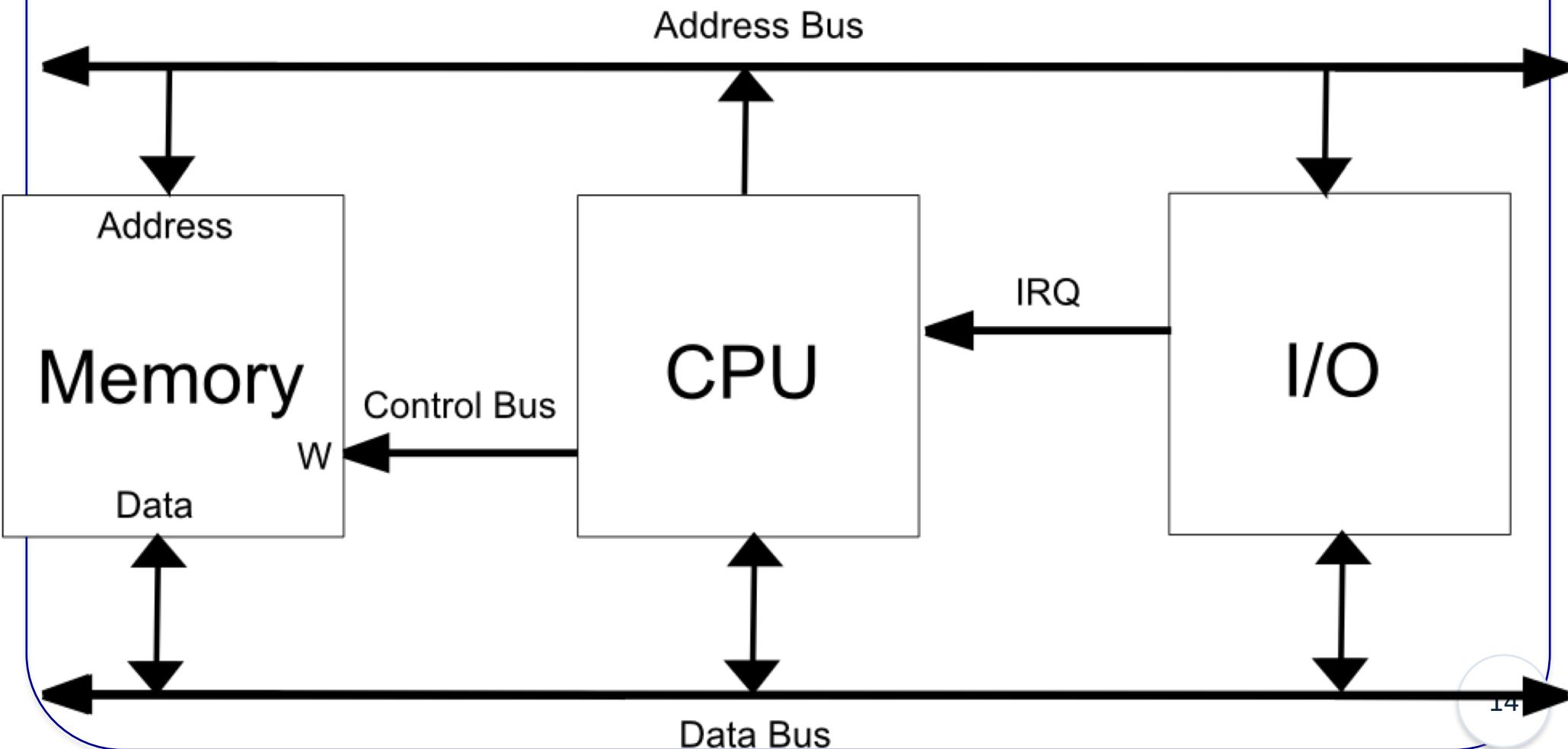
P.ej., supongamos un terminal con cuatro registros, dos de *input* (p.ej., *status* y dato del teclado) y dos de *output* (p.ej., *status* y dato de la pantalla), cada uno con una dirección única:

- el bit 7 del registro de *status* del teclado se pone en 1 cada vez que llega un nuevo carácter
 - ... la CPU lee repetidamente este registro hasta que el bit 7 esté en 1
 - ... entonces lee el registro de dato del teclado
- similarmente, el bit 7 del registro de *status* de la pantalla se pone en 1 cada vez que ésta está lista para aceptar un carácter
 - ... la CPU lee repetidamente este registro hasta que el bit esté en 1
 - ... entonces escribe un carácter en el registro de dato de la pantalla

El problema con I/O programado es que la CPU pasa la mayor parte del tiempo en un *loop* esperando a que se ponga en 1 el bit 7 de los registros de *status* —***busy waiting***

I/O dirigido por interrupciones —La CPU hace partir al dispositivo y le dice que la interrumpa cuando esté listo (cuando se haya completado el I/O):

- la CPU “setea” el bit *interrupt enable* del controlador del dispositivo (p.ej., el bit 6 del registro de *status*)



Una **interrupción** cambia el flujo de control:

- normalmente por algo relacionado con I/O, y no debido (al menos, directamente) al programa que se está ejecutando

La interrupción detiene el programa y transfiere el control a un **manejador de interrupciones**:

- primero, ejecuta las acciones apropiadas a la interrupción
- luego, devuelve el control al programa, que debe ser reanudado en exactamente el mismo estado que tenía cuando ocurrió la interrupción

P.ej., supongamos que el computador quiere mostrar una línea de caracteres en la pantalla:

- el software primero recolecta todos los caracteres en un buffer
 - ... inicializa las variables ptr (dirección del buffer) y count (número de caracteres)
 - ... y verifica si el terminal está listo, en cuyo caso envía el primer carácter
- la CPU ha iniciado la operación de I/O y puede empezar a hacer otra cosa
- a su debido tiempo el carácter va a ser desplegado en la pantalla
 - ... y empieza la interrupción → los 6 pasos de la próxima diapositiva

1. El controlador del dispositivo pone en 1 una línea de interrupción en el bus
2. Cuando la CPU está lista para manejar la interrupción, pone en 1 la señal de *acknowledge* de la interrupción en el bus
3. Cuando el controlador ve esta señal, coloca un número entero pequeño en las líneas de datos para identificarse así mismo
4. La CPU guarda este número temporalmente
5. La CPU pone en el stack el *program counter* y el registro *PSW*
6. La CPU encuentra un nuevo *program counter* usando el número como índice en una tabla → este *program counter* apunta al comienzo de la función que maneja la interrupción del dispositivo correspondiente

Problema:

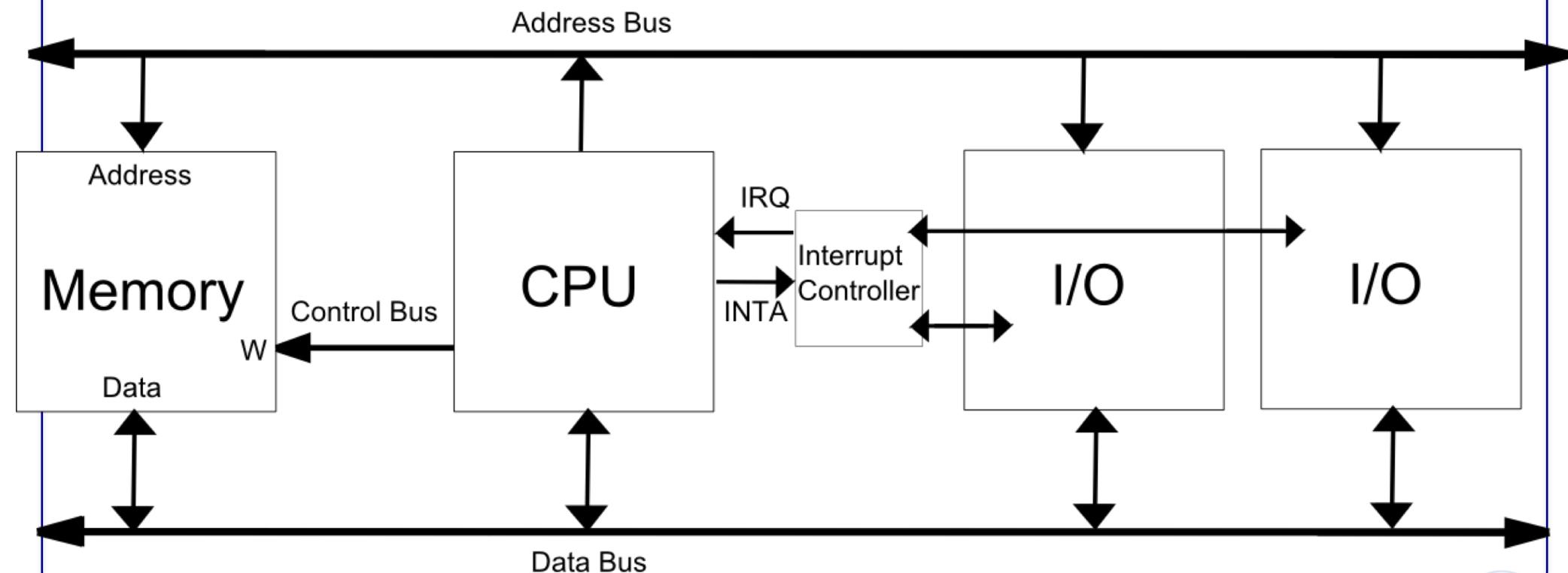
- muchos dispositivos distintos pueden querer producir interrupciones simultáneamente

Solución:

- usar un **controlador de interrupciones**

Un controlador de interrupciones:

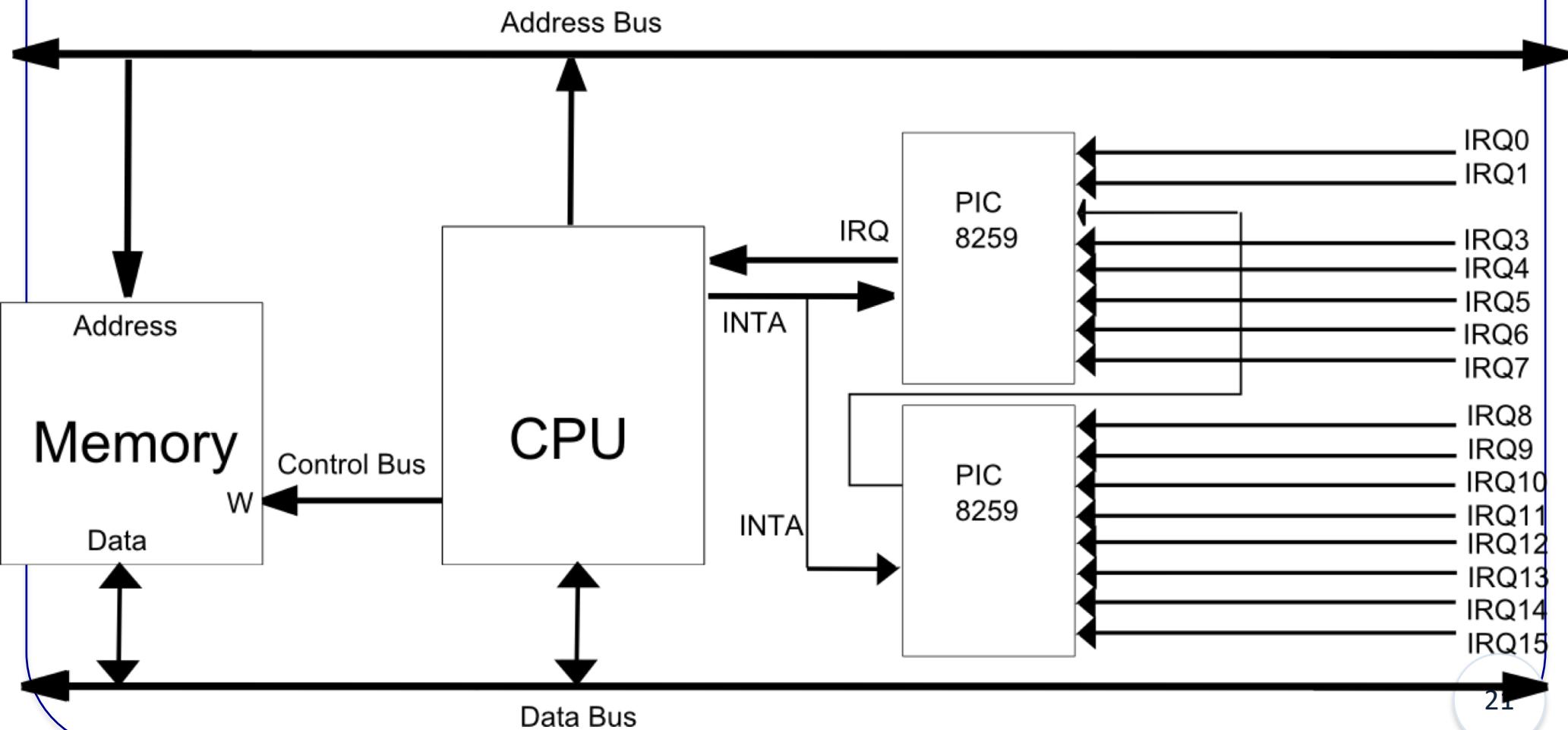
- árbitro que da prioridad a los dispositivos más críticos
- se conecta con múltiples dispositivos de I/O por un lado, y con la CPU por otro



P.ej., el controlador de interrupciones 8259A:

- permite conectar hasta 8 controladores de dispositivos —p.ej., reloj, teclado, disco, impresora, etc.
- cuando cualquiera de éstos quiere producir una interrupción, pone en 1 su línea de input IRQx
- cuando una o más líneas de input están en 1, el controlador coloca en 1 su salida INT, que a su vez opera directamente el pin de interrupción de la CPU
- cuando la CPU está en condiciones de manejar la interrupción, envía una señal de vuelta al controlador por su entrada INTA
- el controlador especifica en el bus de datos cuál input causó la interrupción
- el hardware de la CPU usa este número como índice en su tabla de punteros —o **vectores de interrupción**— para encontrar la dirección de la función que hay que ejecutar para manejar la interrupción

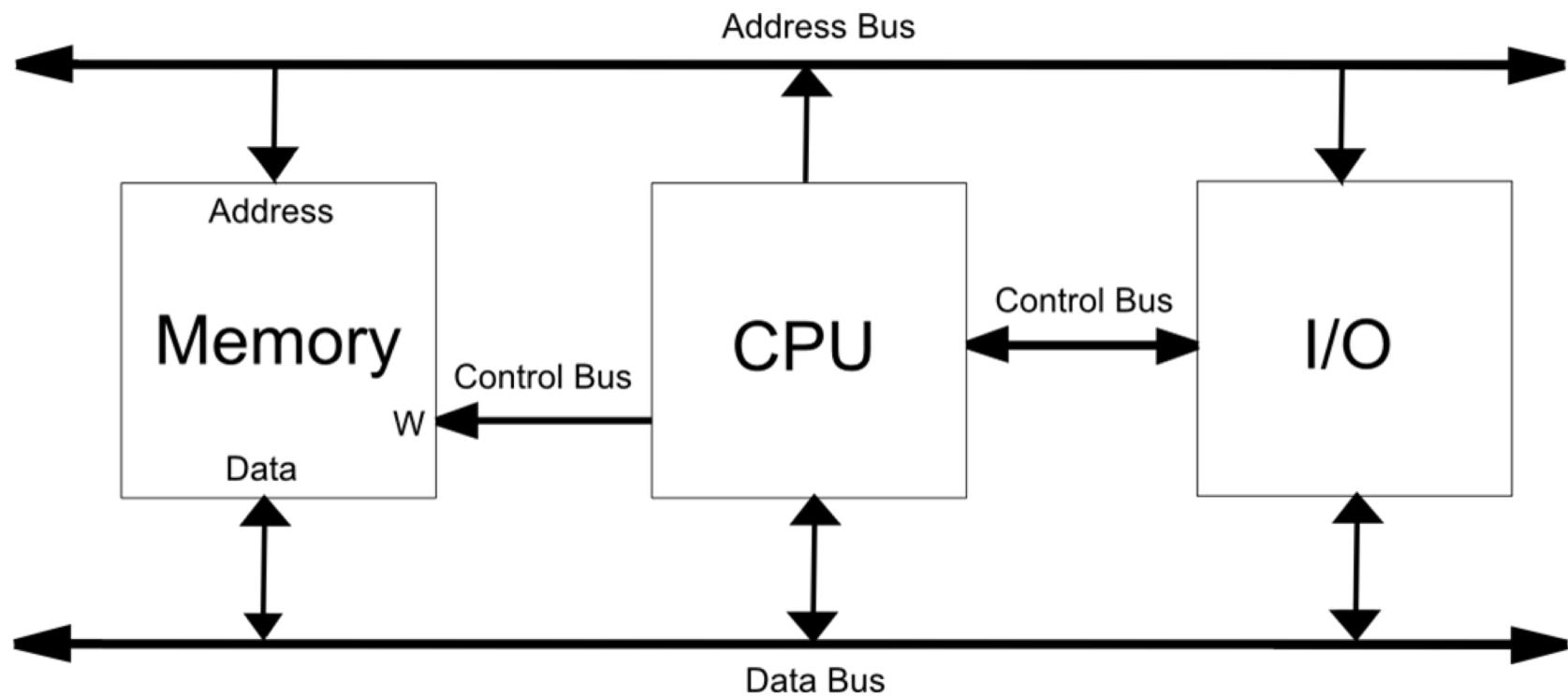
El Core i7 incluye el controlador Intel ICH10, un chip compuesto por dos controladores 8259A conectados en cascada, configuración que permite conectar 15 controladores de dispositivos



Cada input IRQx está asociado a un dispositivo
y un vector de interrupción específicos

IRQ	Dispositivo	Vector de interrupción
IRQ0	Timer del sistema	08
IRQ1	Puerto PS/2: Teclado	09
IRQ2	Conectada al PIC esclavo	0A
IRQ3	Puerto serial	0B
IRQ4	Puerto serial	0C
IRQ5	Puerto paralelo	0D
IRQ6	Floppy disk	0E
IRQ7	Puerto paralelo	0F
IRQ8	Real time clock (RTC)	70
IRQ9-11	No tienen asociación estándar, libre uso.	71-73
IRQ12	Puerto PS/2: Mouse	74
IRQ13	Coprocesador matemático	75
IRQ14	Controlador de disco 1	76
IRQ15	Controlador de disco 2	77

Hasta el momento, todos los datos deben pasar por la CPU para llegar a la memoria o a un dispositivo de I/O



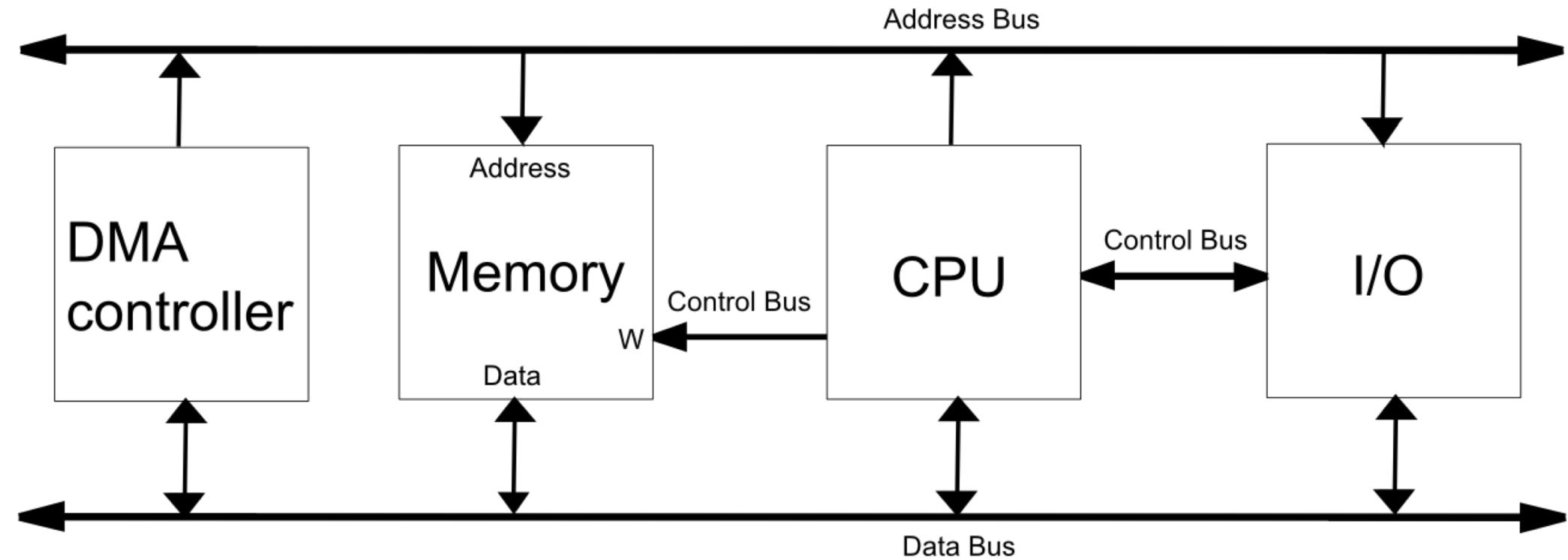
DMA I/O —similar a I/O programado, pero en que alguien más hace el trabajo

Procesar interrupciones es caro

Todos los datos deben pasar por la CPU para llegar a la memoria o a un dispositivo de I/O

Añadimos un nuevo chip al sistema, un **controlador DMA** (*direct memory access*) que permite que los dispositivos de I/O tengan acceso directo al bus

Para permitir que los I/O tengan acceso directo a memoria
se utiliza un **controlador DMA**



El controlador DMA tiene acceso directo al bus y tiene registros que pueden ser escritos por la CPU:

- la dirección de memoria a leer o escribir
- número (count) de bytes o palabras a transferir
- número (identificador) del dispositivo de I/O que se usará
- dirección (lectura o escritura) de transferencia

Una vez inicializados los registros, el controlador DMA

- hace una solicitud al bus, p.ej., para leer 32 bytes a partir de la dirección 100 de la memoria
- hace una solicitud de I/O al dispositivo correspondiente, p.ej., un terminal, para escribir allí el byte leído
- incrementa su registro de dirección y decrementa su registro *count*
- si *count* es > 0, repite las operaciones anteriores
- finalmente, cuando *count* = 0, interrumpe a la CPU

Un controlador que lee y escribe datos desde y en la memoria sin intervención de la CPU está haciendo *acceso directo a memoria* o **DMA**:

- cuando la transferencia está completa, el controlador produce una **interrupción**
 - ... obligando a la CPU a suspender de inmediato el programa vigente
 - ... y empezar a ejecutar el **manejador de interrupciones**
 - ... que, entre otros, informa al sistema operativo que el I/O ha terminado
- cuando el manejador de interrupciones termina, la CPU continúa con el programa que había suspendido

Arquitectura de un computador con memory-mapped I/O, interrupciones y DMA

