Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 Arquitectura de Computadores

Comunicación de CPU y Memoria con I/O

1. Motivación

Una vez entendidas las bases de la programabilidad de un computador, el siguiente paso corresponde a estudiar la comunicación entre las distintas partes del computador, la comunicación con los usuarios y la comunicación con otros computadores. Un primer aspecto a analizar es como la CPU y la Memoria se comunican con el resto de los componentes que pueda tener un computador.

2. Comunicación entre las partes del computador

El modelo de computador visto hasta el momento incluye principalmente dos componentes: Memoria y CPU (Central Processing Unit). La memoria corresponde al componente donde se almacenan datos e instrucciones (juntos si es la arquitectura Von Neumann, separados si es Harvard). La CPU corresponde a todos los elementos adicionales que permiten implementar programas en la máquina: registros de uso general, registros de uso específico, unidades de ejecución, unidad de control, program counter y todos los componentes digitales que permiten la conexión entre los elementos.

En las figuras 1 y 2 se observa el diagrama de un computador básico (con arquitecturas Harvard y Von Neumann, respectivamente), agrupando todos los elementos de la CPU en un solo componente. Se puede observar que existen distintos tipos de comunicación, los cuales pueden ser categorizados en tres grupos:

- **Comunicación de direcciones:** Tanto en el computador Harvard como en el Von Neumann existe una conexión entre la CPU y la dirección de la RAM, que permite que el computador indique que dato (o instrucción en el caso Von Neumann) se va a leer o escribir. En el computador Harvard, adicionalmente se tiene una conexión hacia la dirección de la ROM, indicando la instrucción a ejecutar.
- **Comunicación de datos:** Nuevamente, en ambos modelos existen canales de comunicación de datos con la RAM: uno para enviar datos que entren a la memoria, y otro para recibir datos que salgan de esta. En Harvard adicionalmente existe una conexión de datos recibidos de la ROM, que corresponde a las instrucciones del programa en ejecución.
- **Comunicación de control:** Por último, para controlar la RAM ambas arquitecturas tienen una señal de control que indica si la RAM debe leer o escribir en la dirección indicada.

Dado que los tipos de comunicación para ambas arquitecturas son similares, de ahora en adelante se trabajará con la arquitectura Von Neumann, pero todos los conceptos serán aplicables también a la arquitectura Harvard. Para completar un modelo de computador general Von Neumann con una memoria y una CPU, es necesario considerar la implementación en detalle de los canales de comunicación entre estos dos elementos, y además especificar como se comunican los otros dispositivos externos que permiten ingresar y extraer datos desde el computador.

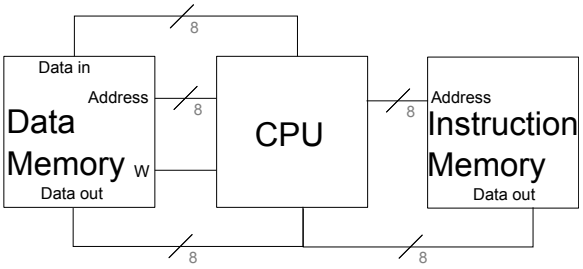

Figura 1: Abstracción de computador básico con microarquitectura Harvard.

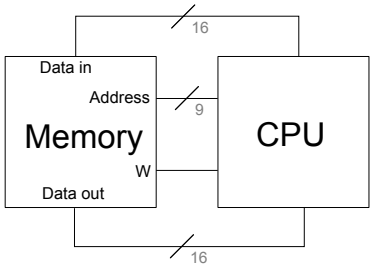

Figura 2: Abstracción de computador básico con microarquitectura Von Neumann.

2.1. Canales de comunicación: Buses compartidos

Los canales de comunicación que habitualmente se ocupan para los distintos tipos de comunicación en un computador corresponden a **buses compartidos**. Un bus compartido consiste en un canal de cables eléctricos que puede ser accedido por más de un dispositivo a la vez. La ventaja de este modelo con respecto a buses no compartidos, es que permite reducir la cantidad de conexiones entre los distintos componentes del computador. La desventaja es que si dos dispositivos transmiten información al mismo tiempo se puede producir una falla, ya que se estará tratando de definir voltajes distintos para las mismas líneas eléctricas.

En la figura 3 se observa un computador Von Neumann con buses compartidos. Se puede ver que los canales de datos de entrada y salida fueron reducidos a un **bus compartido de datos**, en el cual habrá datos de salida o entrada dependiendo de lo que se requiera. El **bus de dirección** y el **bus de control** completan el modelo de computador con buses compartidos.

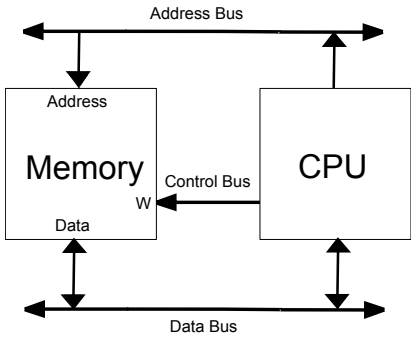

Figura 3: Abstracción de computador Von Neumann con buses compartidos.

2.2. Dispositivos de Entrada y Salida: I/O

El modelo general de un computador incluye una gran variedad de dispositivos que pueden ser conectados a éste, para ingresar y recibir datos. A este conjunto de dispositivos se les conoce como dispositivos de entrada

y salida (E/S) o input/output (I/O). Considerando los dispositivos de I/O, el modelo del computador Von Neumann se extiende de la siguiente forma:

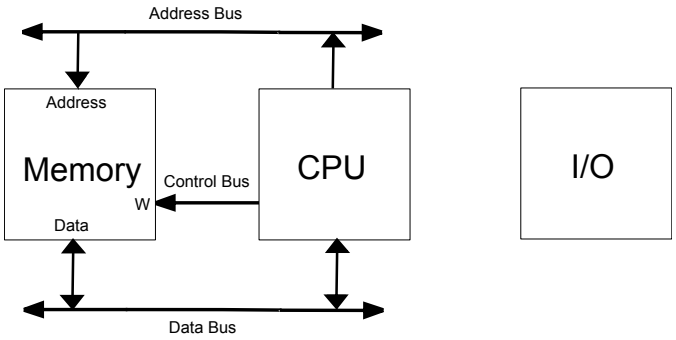

Figura 4: Abstracción de computador Von Neumann con I/O.

Antes de detallar la comunicación entre los dispositivos de I/O y el resto del computador, es necesario caracterizar a estos dispositivos.

2.2.1. Características

Los dispositivos de I/O son muy diversos en sus funcionalidades y estructuras internas. Sin embargo, existen tres características relevantes que sirven para organizarlos:

- **Comportamiento**: El comportamiento de un dispositivo I/O indica que puede hacer el computador con el dispositivo. Las opciones son:
  - Entrada (Input): entregan datos al computador, pero no permiten que el computador envíe datos.
  - Salida (Output): reciben datos del computador, pero no entregan datos.
  - Entrada o salida (Input or Output): permiten entregar datos al computador o recibir datos, pero no de manera simultánea.
  - Almacenamiento (Input and Output): entregan y reciben datos almacenados en el dispositivo.
- **Comunicante**: La segunda característica indica con quien se está comunicando el dispositivo:
  - Humano: dispositivos como el mouse, teclado y la pantalla, son dispositivos que se comunican con un ser humano.
  - Máquina: dispositivos como la tarjeta de red, o los discos duros se comunican con máquinas, sin intervención de un ser humano.
- **Velocidad de transferencia**: La velocidad de transferencia de los dispositivos de I/O es muy variable entre dispositivos, y a veces en un mismo dispositivo. En general se utiliza la velocidad **peak** como referencia para cada dispositivo.

2.2.2. Componentes de los dispositivos I/O

A pesar de las diferentes características que tienen los dispositivos de I/O, la mayoría de estos contienen dos tipos de componentes definidos: **elementos electromecánicos** que realizan las operaciones de interacción, y **controladores** electrónicos que regulan el funcionamiento de los componentes mecánicos y se comunican con el computador.

Dispositivo	Comportamiento	Comunicante	Velocidad de transferencia (Mbit/s)
Teclado	Input	Humano	0.0001
Mouse	Input	Humano	0.0038
Voice in	Input	Humano	0.2640
Scanner	Input	Humano	3.2
Voice out	Output	Humano	0.2640
Impresora láser	Output	Humano	3.2
Display	Output	Humano	800
Red por cable	Input o Output	Máquina	100-1000
Red inalámbrica	Input o Output	Máquina	11-54
Disco óptico	Almacenamiento	Máquina	80-220
Disco magnético	Almacenamiento	Máquina	800-3000

Tabla 1: Ejemplos de dispositivos I/O y sus características.

Controladores

Los controladores de dispositivos de I/O son circuitos digitales encargados de controlar las partes electromecánicas del dispositivo según la comunicación realizada con el computador. En la mayoría de los casos, los controladores son microprocesadores completos, que ejecutarán programas especialmente diseñados para controlar al dispositivo. Las funciones principales del controlador son las siguientes:

- Comunicación con el computador.
- Comunicación con el dispositivo.
- Almacenamiento temporal.
- Detección de errores.
- Control de elementos mecánicos/físicos.
- Conversión de señales continuas en digitales o viceversa.

Para cumplir las funciones antes descritas, los controladores cuentan con los siguientes componentes:

- **Circuitos de control:** Dependiendo de la complejidad del dispositivo será la complejidad de estos circuitos, que en muchos casos corresponden a microprocesadores completos, y estarán encargados de regular el funcionamiento del dispositivo, coordinar la comunicación con el computador y ejecutar la detección y/o corrección de errores.
- **Conversores ADC o DAC:** Dependiendo de la funcionalidad y complejidad, los controladores pueden incluir conversores analógico-digital o digital-analógico para traducir señales eléctricas continuas en información para el computador o viceversa.
- **Memoria:** Los controladores contendrán una memoria la cual será el mecanismo fundamental para comunicarse con el computador. Algunas direcciones de esta memoria se reservarán para funciones específicas, y se denominan registros:
  - **Buffer:** parte de la memoria, que almacena los datos que el dispositivo esté entregando o recibiendo del computador.

- **Registros de control:** direcciones específicas de la memoria que son utilizadas por el computador para indicar comandos que debe ejecutar el dispositivo.
- **Registros de status:** direcciones específicas de la memoria que son utilizadas por el dispositivo para indicar información al computador.
- **Registros de datos:** direcciones específicas de la memoria para leer o escribir datos individuales o asociados a la memoria local (buffer) del dispositivo.
- **Registros de dirección:** direcciones específicas de la memoria para direccionar la memoria local (buffer) del dispositivo.

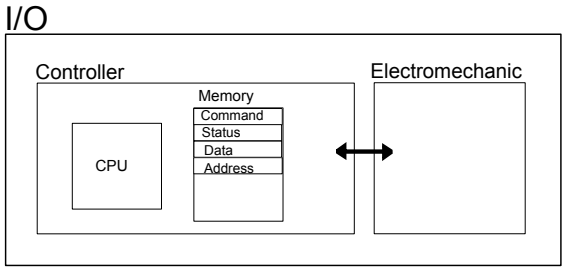

Figura 5: Diagrama genérico de I/O.

2.3. Comunicación entre los dispositivos de I/O, la CPU y la Memoria

Para poder interactuar con los dispositivos de I/O es necesario que la CPU pueda comunicarse con estos. Como se señaló anteriormente, el controlador de cada dispositivo será el encargado de realizar esta comunicación con la CPU, para lo cual el procedimiento que se utiliza es que la CPU escriba o lea directamente información de los registros o el buffer del controlador. En base a este procedimiento, existen tres tipos de comunicaciones que pueden ocurrir entre la CPU y los dispositivos de I/O:

- **Comunicación de comandos:** Cuando la CPU quiere indicarle a un dispositivo que realice una determinada acción, debe hacerlo enviándole comandos a los registros de control. Por ejemplo si la CPU quiere avisarle a un cierto dispositivo (e.g. cámara web) que se encienda, deberá escribir en un determinado registro de control del controlador del dispositivo un cierto número, que este interpretará como el comando de activación.
- **Comunicación de estado:** Cuando la CPU quiere obtener información sobre el estado del dispositivo, debe hacerlo leyendo uno de los registros de status de éste. Por ejemplo si la CPU quiere saber si un dispositivo tiene nueva información que enviar (e.g. el mouse se movió, y tiene nuevas posiciones que enviar), la CPU deberá leer el valor almacenado en el registro específico e interpretarlo según corresponda.
- **Transferencia de datos:** Cuando la CPU quiere enviarle datos a un dispositivo o leer datos de él, si es poca información esto se podrá hacer ocupando registros de datos que tenga el controlador del dispositivo. Si es más información, esto se realizará escribiendo o leyendo desde el buffer (memoria) del dispositivo. Por ejemplo, si el disco duro quiere enviarle información a la CPU, se accederá a ésta mediante el buffer de datos del disco.

2.3.1. Acceso a los dispositivos de I/O y transferencia de datos

Para poder realizar los tipos de comunicación antes descritos es necesario que la CPU pueda acceder a los registros y memorias de los distintos dispositivos de I/O. Para esto es necesario definir un mecanismo general que permita acceder de manera equivalente a todos los dispositivos y enviarle datos, es decir hay que poder **direccionar** los distintos registros o buffers y **transferir datos** a ellos.

El modelo de direccionamiento y transferencia de datos ya existe en el computador, en la comunicación de la CPU con la memoria de datos. En ese caso, si se tiene una instrucción de transferencia, por ejemplo `MOV A, (120)`, lo que ocurrirá en el sistema es:

- La CPU coloca el valor 120 en el bus de direcciones.
- La CPU envía la señal de control $W = 0$ a la RAM indicando que va a leer y no escribir de la RAM
- La RAM recibe desde el bus de direcciones en su entrada de dirección el valor 120.
- La RAM coloca en el bus de datos el valor $Mem[120]$, es decir, el valor de memoria apuntado por la dirección 120.
- La CPU recibe desde el bus de datos el valor $Mem[120]$ y lo almacena en el registro A.

Para acceder a los dispositivos de I/O se necesitan ejecutar pasos similares a los de comunicación con la memoria. Por ejemplo si la CPU quiere leer el estado de un cierto dispositivo, los pasos a seguir serían:

- La CPU coloca en el bus de direcciones algún valor que «direcciona» al registro de status de ese dispositivo.

- El dispositivo recibe desde el bus de direcciones ese valor y lo interpreta apuntando el valor del registro de status.
- El dispositivo coloca el valor del registro de status en el bus de datos.
- La CPU recibe desde el bus de datos el valor del registro de status del dispositivo.

Existen dos mecanismos principales para realizar el proceso de direccionamiento y transferencia de datos: mapeo a memoria (**memory mapped I/O**) y usando el mecanismo de puertos (**port I/O**).

Memory mapped I/O

La idea central de memory mapped I/O es la siguiente: si el computador ya tiene soporte a nivel de instrucciones para direccionar y transferir datos desde y hacia la memoria de datos, por que no aprovechar esos mismos mecanismos para acceder a los registros y buffers de los dispositivos de I/O. La implementación de esta idea consiste en reservar un espacio de direcciones de memoria para ser ocupado para «mapear» los registros y buffer de los dispositivos de I/O. De esta forma es posible acceder directamente ocupando una instrucción como `MOV A, (dir)` a un dispositivo.

El mapeo explícito a los distintos registros y buffers de los dispositivos los coordina el **address decoder**, una pieza de hardware especializada que estará «vigilando» el bus de dirección para determinar si la dirección colocada corresponde a la RAM o a alguno de los dispositivos.

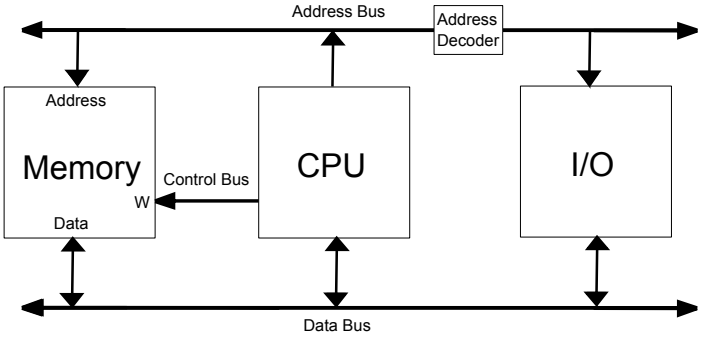

Figura 6: Computador ocupando memory mapped I/O.

Los buffers de los dispositivos pueden estar mapeados completamente en memoria o no. En caso de estar mapeados completamente, el acceso a cada dirección de memoria del buffer es directo. En caso contrario, el mecanismo que se ocupa es tener mapeados los registros de dirección y datos del dispositivo, y ocupar esos registros para direccionar y escribir/leer datos, respectivamente.

El mecanismo de mapeo a memoria tiene varias ventajas: su implementación es simple a nivel de hardware, es posible ocupar las mismas instrucciones de transferencia y modos de direccionamiento que al trabajar con memoria y también es posible realizar operaciones en la ALU, de la misma forma que se realizaban con datos de memoria.

La desventaja de este sistema es que limita el espacio direccionable de la memoria, lo que se conoce como **memory barrier**. Debido a que se debe reservar un grupo de direcciones para apuntar a los dispositivos, estas direcciones no pueden ser ocupadas para almacenar en memoria.

En la arquitectura x86 el acceso a muchos de los dispositivos se realiza mediante mapeo a memoria. En el IBM PC original (primer computador con arquitectura x86), se tenía un espacio direccionable de memoria de 1MB, del cual el espacio entre las direcciones 640KB y 1MB se reservó para el mapeo de dispositivos de I/O. Uno de los dispositivos que ocupaban mayor parte de ese espacio de direcciones era la tarjeta de video, la cual tenía mapeada su memoria de video de manera de permitir pintar pixeles en el display ocupando instrucciones de copia a memoria.

Port I/O

El segundo mecanismo para acceder a los dispositivos es mediante port I/O. En este mecanismo se definen instrucciones específicas y un espacio de direccionamiento propio para acceder a los dispositivos. En el caso de la arquitectura Intel, por ejemplo, se definen dos instrucciones: `IN` que se ocupa para leer datos desde un dispositivo y `OUT` que se ocupa para escribir datos en un dispositivo.

Para direccionar, se ocupa un espacio de direcciones separado del de la RAM, por ejemplo, la instrucción `IN A, (120)` copiaría el valor de I/O asociado a la dirección 120 en el registro A. Esa dirección 120 no tendría relación alguna con la dirección 120 de RAM, y por tanto no se limita el espacio direccionable con este método.

Para efectivamente implementar este direccionamiento especial, existen dos mecanismos. El primero es utilizar un bus de direcciones especial sólo para I/O (figura 7) con lo que se asegura de que no habrá choque entre las direcciones. El segundo método consiste en ocupar el mismo bus que el de la RAM, pero indicarle mediante una señal de control al bus, que se está refiriendo a una dirección de I/O, y por tanto la RAM no debe considerar ese valor (figura 8).

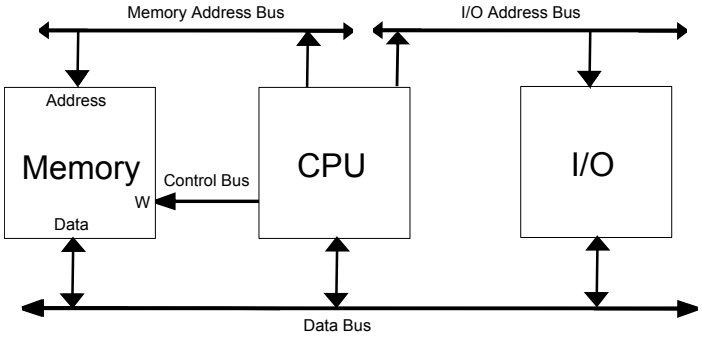

Figura 7: Computador ocupando port I/O con bus de direcciones separado.

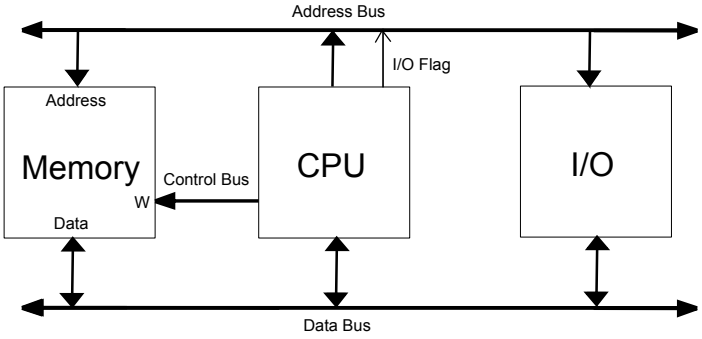

Figura 8: Computador ocupando port I/O con bus de direcciones compartido.

Para utilizar Port I/O en la arquitectura x86, existe un espacio direccionable de 16 bits, permitiendo acceder a 65536 puertos, que van desde la dirección `0000h` hasta la `FFFFh`. Existen dos instrucciones especializadas para acceder a los puertos: `IN` y `OUT`. Ambas instrucciones trabajan con dos parámetros: el número (dirección) del puerto al cual se quiere acceder y el registro `AX` o `AL`, que se ocupará para recibir o enviar datos al dispositivo. El formato de ambas instrucciones es el siguiente:

- `IN Reg, Port` donde `Reg` es `AX` o `AL` y `Port` es un número que va desde 0 hasta 65535.
- `OUT Port, Reg` donde `Reg` es `AX` o `AL` y `Port` es un número que va desde 0 hasta 65535.

A continuación se muestra un ejemplo ocupando uno de los dispositivos I/O de prueba asociados al software emu8086, el cual simula el control de un robot. El controlador del robot tiene los siguientes registros: registro de comandos (asociado al puerto 9), registro de datos (asociado al puerto 10) y registro de estado (asociado al puerto 11). Los detalles de los valores asociados a cada registro se encuentran en: **http://www.emu8086.com/assembler_tutorial/io.html**

```
#start=robot.exe#

org 100h
start:
        CALL waitcommand
        MOV AL, 1  ; move forward.
        OUT 9, AL  ;

        CALL waitcommand
        MOV AL, 4  ; examine.
        OUT 9, AL  ;

        CALL waitdata
        IN AL, 10
        CMP AL, 0
        JE start

        CALL TURN
        JMP start
        RET
waitcommand:
loop1:  IN AL, 11
        AND AL, 10b
        JNE loop1
        RET
waitdata:
loop2:  IN AL, 11
        AND AL, 01b
        JE loop2
        RET
turn:
        CALL waitcommand
        MOV AL, 3  ; turn right.
        OUT 9, AL  ;
        RET
```

2.3.2. Comunicación I/O → CPU

Polling

Ocupando los mecanismos de direccionamiento y transferencia previamente descritos, es posible implementar toda la comunicación necesaria entre la CPU, memoria e I/O. En el caso particular de la comunicación que va desde el dispositivo a la CPU, por ejemplo cuando un dispositivo quiere avisar que hizo algo (e.g. el mouse se movió) el algoritmo que maneja esa comunicación sería de la siguiente forma:

1. Ejecutar instrucciones del programa.
2. Revisar el estado del dispositivo leyendo su registro de status
3. Si el dispositivo tiene algo nuevo que reportar, se pasa a 4. Si no, se vuelve a 1.
4. Leer un dato del dispositivo
5. Si quedan datos por leer, volver a 4. Si se terminó de leer, se vuelve a 1.

Este mecanismo se conoce como **polling** porque la CPU debe estar preguntándole o encuestando al dispositivo continuamente para saber si este tiene algo nuevo que reportar. El problema de esto es que es

muy ineficiente, ya que la CPU debe gastar varias instrucciones para determinar esto, y si fueran varios dispositivos los que hay que revisar, el problema sería aún peor.

La solución a esta situación es desarrollar un mecanismo especial con soporte de hardware para que los dispositivos le avisen al computador cuando tienen algo nuevo que reportar. Este mecanismo se conoce como **interrupciones**.

Interrupciones

En un sistema de I/O basado en interrupciones, el dispositivo de I/O será el encargado de avisar cuándo ocurre un suceso relevante a la CPU, liberando a ésta de tener que estar preguntándole a cada dispositivo si ha habido algún evento relevante. Para lograr esto, el dispositivo de I/O se debe conectar a la CPU con una señal de control dedicada, denominada **interrupt request** o IRQ. Mediante esta señal de 1 bit el dispositivo le notificará a la CPU si ocurrió algo relevante.

Para atender la solicitud del dispositivo, se definen subrutinas especiales, denominadas **Interrupt Service Routine (ISR)**. En estas rutinas se debe definir el código encargado de atender la solicitud del dispositivo.

Cuando la CPU detecta que ocurrió la interrupción los pasos que se deben realizar son los siguientes:

1. Dispositivo interrumpe (IRQ).
2. CPU termina de ejecutar la instrucción actual, y guarda en el stack los condition codes.
3. CPU deshabilita la atención de más interrupciones.
4. CPU llama a la ISR asociada al dispositivo.
5. ISR respalda el estado actual de la CPU.
6. ISR ejecuta su código.
7. ISR devuelve el estado previo a la CPU.
8. ISR retorna.
9. CPU rehabilita la atención de interrupciones.
10. CPU recupera condition codes desde el stack.

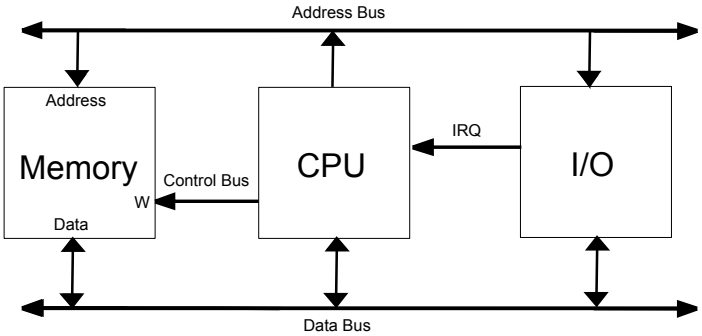

Figura 9: Computador con un dispositivo I/O ocupando interrupciones

En la práctica, un computador cuenta con múltiples dispositivos de I/O que pueden interrumpir. Debido a esto el manejo de interrupciones se complica, ya que todos los dispositivos deben poder notificarle a la CPU

algún evento. Para lograr esto una opción sería que a la CPU llegara una señal IRQ por cada dispositivo. El problema de esto es que no es escalable. Para solucionar este problema se agrega un **controlador de interrupciones** al cual se conectan todos los dispositivos y el cual se comunica directamente con la CPU. Cuando ocurre una interrupción, el controlador notificará a la CPU. Esta, para saber quien fue el que interrumpió, le envía una señal **Interrupt Acknowledge (INTA)** al controlador, para que le envíe el id del dispositivo por el bus de datos.

Como existen distintos dispositivos, existirá una ISR asociada al id de cada dispositivo. Las direcciones donde se encuentran almacenadas las ISR se encuentran en lo que se conoce como **vector de interrupciones**. El vector de interrupciones es un conjunto de palabras de memoria (habitualmente almacenados al comienzo de ésta), en el cual cada palabra tiene almacenada la dirección de una ISR de un dispositivo. De esta manera, el id que entrega el controlador de interrupciones luego de que la CPU le envíe la señal INTA, va a corresponder a una dirección dentro del vector de interrupciones, que indicará a su vez la dirección de memoria donde comienza la ISR asociada, y por tanto esa dirección se cargará en el program counter, para que la CPU ejecute la subrutina.

Una vez que se termina de ejecutar la ISR, esta se encarga de avisarle al controlador de interrupciones que ya se completó la atención, enviando un comando **End of Interrupt (EOI)**.

Para deshabilitar una interrupción específica que no se quiere recibir, se utiliza el concepto de **enmascaramiento o masking**, que consiste en poder especificar que algunas interrupciones se desestimen. Para eso, el controlador de interrupciones tiene un registro especial, el Interrupt Mask Register o **IMR** que permite definir que interrupciones se atenderán y cuáles no.

Otro potencial problema de trabajar con múltiples dispositivos es que pueden ocurrir múltiples interrupciones mientras ya se esté atendiendo una. Para manejar esto, lo que se hace es que el controlador de interrupciones, mientras no recibe una señal INTA, encola las solicitudes. Cuando se recibe un EOI, se le envía a la CPU la señal de la primera interrupción en la cola a ser atendida. Si hay más de una interrupción en cola, el controlador las ordenará de acuerdo a **prioridades** que estarán preestablecidas para los distintos dispositivos.

En algunos casos particulares, algunas interrupciones son demasiado relevantes como para encolarlas. Estas interrupciones se denominan **no enmascarables** y son capaces de interrumpir la ejecución de una ISR.

Interrupciones en x86

El manejo de las interrupciones en la arquitectura x86 está controlado por el **Programmable Interrupt Controller 8259** (PIC8259), una pieza de hardware especialmente diseñada para manejar interrupciones de dispositivos de I/O. El PIC está compuesto por los siguientes componentes:

- IRQ: Cada PIC8259 maneja hasta 8 IRQ (IRQ0-IRQ7), cada uno de los cuales estará asociado a un dispositivo I/O.
- Interrupt Request Register: Registro de 8 bits que mantiene la información de las interrupciones que están actualmente esperando un acknowledge de la CPU (INTA). Cada bit se asocia a un IRQ, donde un 1 en ese bit representa que el dispositivo asociado a ese IRQ interrumpió y espera INTA.
- In-Service Register: Registro de 8 bits que mantiene la información de las interrupciones que están siendo atendidas. Cada bit se asocia a un IRQ, donde un 1 en ese bit representa que el dispositivo asociado a ese IRQ está siendo atendido y espera EOI.
- Interrupt Mask Register: Registro de 8 bits que mantiene la información de las interrupciones que deben ser consideradas. Cada bit se asocia a un IRQ, donde un 1 en ese bit representa que si el dispositivo interrumpe debe ser atendido; un 0 indicará que una interrupción de ese dispositivo no

será considerada. Este registro puede ser modificado en la ISR usando el port I/O mediante el puerto 0x21 para el primer PIC, y 0xA1 para el segundo PIC.

- Priority Solver: Circuito que define que interrupción notificar primero a la CPU, en caso de existir varias interrupciones pendientes.

El PIC8259 tiene la capacidad de trabajar en modo «maestro» o «esclavo», lo que permite que a un PIC «maestro», se le puedan conectar las salidas IRQ de otros PIC «esclavos» a alguna de sus entradas IRQ. En la arquitectura x86 tradicional se utilizan dos PIC8259, uno maestro y uno esclavo. El PIC esclavo se conecta al IRQ2 del PIC maestro, totalizando 15 posibles IRQ para conectar dispositivos. El diagrama de conexión entre los PIC y la CPU se observa en la figura 10.

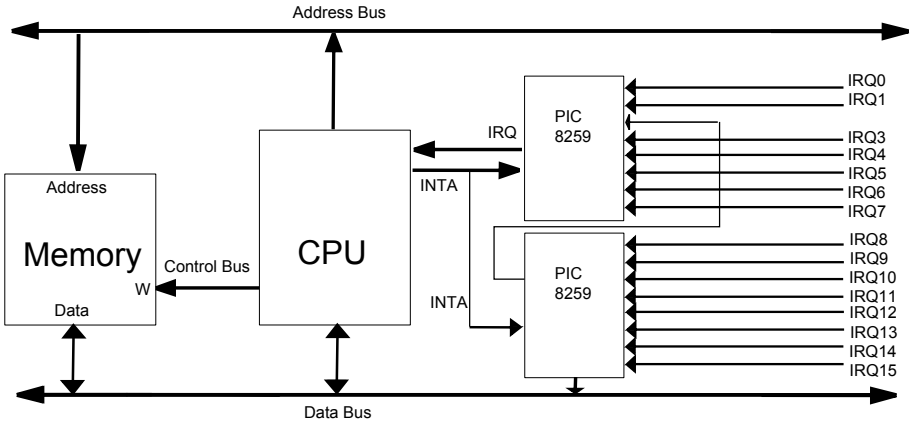

Figura 10: Conexión entre controladores de interrupciones y la CPU en la arquitectura x86 tradicional.

A nivel de la CPU, la habilitación y deshabilitación de las interrupciones están controladas mediante un flag del registro de status denominado **Interrupt Flag (IF)** el cual cuando está en 0 indica que no se atenderán interrupciones, y cuando está en 1 indica que sí se atenderán.

Para enviar una señal EOI, los dos PIC tienen un registro de comando, asociados a los puertos 0x20 y 0xA0 respectivamente. La señal EOI se envía mediante un comando 0x20 al puerto que corresponda.

A continuación se presenta la asociación habitualmente ocupada en la arquitectura x86 entre IRQ y dispositivos:

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

Tabla 2: IRQ en arquitectura x86 tradicional.

Considerando todos estos elementos, el flujo completo de manejo de interrupciones es el siguiente:

1. Dispositivo envía señal IRQ al controlador.
2. PIC revisa su registro IMR, si la interrupción no está enmascarada la atiende, marcando un 1 en el bit correspondiente del Interrupt Request Register.
3. PIC decide cuál de las interrupciones actuales es prioritaria (menor IRQ) y marca un 1 en el bit correspondiente del In-Service Register.
4. PIC envía interrupción (INT) a la CPU.
5. CPU termina de ejecutar la instrucción actual, y guarda en el stack los condition codes.
6. CPU revisa si el flag de interrupciones está activo (IF = 1), en cuyo caso va a atender a la interrupción.
7. CPU deshabilita la atención de más interrupciones (IF=0).
8. CPU envía INTA para saber quien interrumpió.
9. PIC revisa In-Service Register para saber el id del IRQ que está siendo atendido, y lo envía mediante bus de datos.
10. CPU usa el id para buscar dirección de ISR en el vector de interrupciones.
11. CPU llama a la ISR asociada al dispositivo (CALL Mem[id]).
12. ISR respalda el estado actual de la CPU.
13. ISR ejecuta su código.
14. ISR envía un comando EOI al PIC, con lo cual éste setea en 0 el In-Service Register, indicando que terminó la atención de la interrupción.
15. ISR devuelve el estado previo a la CPU.
16. ISR retorna.
17. CPU rehabilita la atención de interrupciones (IF=1).
18. CPU recupera condition codes desde el stack.

A continuación se muestra un código que implementa interrupciones de hardware. En este código, primero se inscriben las ISR asociándolas a direcciones del vector de interrupción (no es relevante entender el detalle de esta parte en código). Luego de esto el programa principal entra en un loop, esperando el llamado de una ISR. En cada ISR se observa la estructura básica de una subrutina de este tipo:

1. Backup de todos los registros mediante la instrucción `PUSHA`
2. Ejecución del código de la subrutina
3. Envío de señal EOI (0x20) al PIC (en este caso el primer PIC, asociado al puerto 0x20).
4. Devolución de los valores de los registros mediante la instrucción `POPA`
5. Retorno mediante la instrucción `IRET`.

```
    #start=robot.exe#
    #start=InterruptGenerator.exe#

org 100h

start:
        MOV AX, 0
        MOV ES, AX
        MOV AL, 90H ; EN LA DIRECCION 0X90 DEL VECTOR DE INTERRUPCIONES
        MOV BL, 4H
        MUL BL
        MOV BX, AX
        MOV SI, OFFSET [turnleft] ;REFERENCIAMOS LA ISR TURNLEFT
        MOV ES:[BX], SI
        ADD BX, 2
        MOV AX, CS
        MOV ES:[BX], AX

        MOV AX, 0
        MOV ES, AX
        MOV AL, 91H  ; EN LA DIRECCION 0X91 DEL VECTOR DE INTERRUPCIONES
        MOV BL, 4H
        MUL BL
        MOV BX, AX
        MOV SI, OFFSET [forward]  ;REFERENCIAMOS LA ISR FORWARD
        MOV ES:[BX], SI
        ADD BX, 2
        MOV AX, CS
        MOV ES:[BX], AX

        MOV AX, 0
        MOV ES, AX
        MOV AL, 92H   ; EN LA DIRECCION 0X92 DEL VECTOR DE INTERRUPCIONES
        MOV BL, 4H
        MUL BL
        MOV BX, AX
        MOV SI, OFFSET [turnright]  ;REFERENCIAMOS LA ISR TURNRIGHT
        MOV ES:[BX], SI
        ADD BX, 2
        MOV AX, CS
        MOV ES:[BX], AX

        MOV AX, 0
        MOV ES, AX
        MOV AL, 93H   ; EN LA DIRECCION 0X93 DEL VECTOR DE INTERRUPCIONES
        MOV BL, 4H
        MUL BL
        MOV BX, AX
        MOV SI, OFFSET [examine]    ;REFERENCIAMOS LA ISR EXAMINE
        MOV ES:[BX], SI
        ADD BX, 2
        MOV AX, CS
        MOV ES:[BX], AX

        MOV AX, 0
        MOV ES, AX
        MOV AL, 94H   ; EN LA DIRECCION 0X94 DEL VECTOR DE INTERRUPCIONES
```

```
  MOV BL, 4H
  MUL BL
  MOV BX, AX
  MOV SI, OFFSET [turnon]      ;REFERENCIAMOS LA ISR TURNON
  MOV ES:[BX], SI
  ADD BX, 2
  MOV AX, CS
  MOV ES:[BX], AX

  MOV AX, 0      ; EN LA DIRECCION 0X95 DEL VECTOR DE INTERRUPCIONES
  MOV ES, AX
  MOV AL, 95H
  MOV BL, 4H
  MUL BL
  MOV BX, AX
  MOV SI, OFFSET [turnoff]      ;REFERENCIAMOS LA ISR TURNOFF
  MOV ES:[BX], SI
  ADD BX, 2
  MOV AX, CS
  MOV ES:[BX], AX

;Programa principal
waiting:
        MOV AL, 7
        JMP waiting

        RET

;Subrutinas
waitcommand:
loop1:  IN AL, 11
        AND AL, 10b
        JNE loop1
        RET

waitdata:
loop2:  IN AL, 11
        AND AL, 01b
        JE loop2
        RET

;ISRs

forward:
        PUSHA  ;Backup de todos los registros

        CALL waitcommand
        MOV AL, 1  ; forward.
        OUT 9, AL  ;

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL

        POPA   ;Devolver todos los registros
        IRET

turnright:
        PUSHA

        CALL waitcommand
        MOV AL, 3  ; turn right.
        OUT 9, AL  ;

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL

        POPA
        IRET
```

```
turnleft:
        PUSHA

        CALL waitcommand
        MOV AL, 2  ; turn left.
        OUT 9, AL  ;

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL

        POPA
        IRET

examine:
        PUSHA

        CALL waitcommand
        MOV AL, 4  ; examine.
        OUT 9, AL  ;
        CALL waitdata

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL


        POPA
        IRET

turnon:
        PUSHA

        CALL waitcommand
        MOV AL, 5  ; on.
        OUT 9, AL  ;

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL


        POPA
        IRET

turnoff:
        PUSHA

        CALL waitcommand
        MOV AL, 6  ; off.
        OUT 9, AL  ;

        MOV AL, 20h   ;EOI al PIC1
        OUT 20h, AL


        POPA
        IRET
```

2.3.3. Excepciones e interrupciones de software

Además de existir interrupciones gatilladas por dispositivos de I/O, existen otros eventos que son capaces de gatillar interrupciones. Un posible tipo de interrupción son las **excepciones**. Las excepciones ocurren cuando la CPU detecta alguna condición de error al ejecutar alguna instrucción. Una de las excepciones habituales es la división por 0, que ocurre cuando el dividendo de una instrucción de división tiene el valor 0, y por tanto no es posible realizar el cálculo de la operación. Un código que gatilla esta excepción se observa a continuación:

```
    org 100h
    MOV BX, 0
    DIV BX
    RET
```

Las excepciones, al igual que las interrupciones de hardware, serán atendidas por una ISR especializada, que habitualmente se denominan **exception handlers**. A diferencia de las interrupciones de hardware, las excepciones las controla directamente la CPU, y no un elemento externo (como el PIC).

Un segundo tipo de interrupción que no es gatillado por un dispositivo de I/O son las **interrupciones de software**. Las interrupciones de software corresponden a interrupciones que son gatilladas explícitamente en un programa ejecutando una instrucción especial. En el caso de la arquitectura x86, la instrucción usada para gatillar una interrupción de software es `INT dir` donde dir corresponde a una dirección de memoria dentro del vector de interrupciones.

Una interrupción de software puede ser pensada como una llamada directa a una ISR particular. A diferencia de las interrupciones de hardware al llamar a la ISR de una interrupción de hardware, las interrupciones de hardware no se deshabilitan. Para realizar esto es necesario ejecutar instrucciones explícitas que modifiquen el valor del Interrupt Flag: para deshabilitar las interrupciones se cuenta con la instrucción `CLI` (clear interrupt flag); para setear en 1 el flag y habilitar las interrupciones se cuenta con la instrucción `STI`.

Las interrupciones de software son habitualmente usadas para acceder a un dispositivo I/O. Para lograr esto, las ISR asociadas a estas interrupciones tienen implementada la comunicación con un determinado dispositivo (ya sea mediante memory mapped I/O o port I/O), lo que permite ahorrarse tener que implementar esa comunicación cada vez que se quiera acceder a un dispositivo de I/O. Adicionalmente, como se verá más adelante, en los computadores con múltiples programas y un sistema operativo, el acceso a los dispositivos de I/O es posible de realizar sólo cuando la CPU está en un modo especial, por lo que directamente los programas no podrían acceder, y el mecanismo de interrupciones de software permite este acceso, protegiendo la comunicación.

El siguiente ejemplo muestra el uso de interrupciones de software para acceder a la tarjeta de video del computador y pintar franjas de distintos colores. En este caso, la llamada `INT 10h` ejecutará una ISR asociada al manejo de la tarjeta de video. Dependiendo del parámetro almacenado en AH antes de hacer la llamada, se ejecutará una ISR distinta. En este caso con $AH = 0$ se ejecuta la ISR de selección de modo de video (texto o gráfico). El valor almacenado en el AL corresponderá al parámetro que indique efectivamente que modo elegir (en este caso el modo 13h). En el segundo llamado `INT 10h` se utiliza el parámetro $AH = 0Ch$ el cual indica que se pintará un pixel. La posición del pixel está dada por los valores almacenados en DX (fila) y CX (columna). El color estará almacenado en los 4 bits menos significativos de AL en formato IRGB: el bit 3 representa la intensidad (1 alta, 0 baja); el bit 2 representa si hay componente rojo (1) o no (0), el bit 1 si hay componente verde o no, y el bit 0 si hay componente azul o no, totalizando 16 posibles colores.

```
    ORG 100H
    MOV AL, 13H
    MOV AH, 0
    INT 10H     ; MODO VIDEO
    MOV BX, COLORES
    MOV CX, 10  ; COLUMNA
FOR1:
    MOV AL, [BX]
    MOV DX, 20  ; FILA
FOR2:
    MOV AH, 0CH
    INT 10H     ; SET PIXEL(CX,DX)=AL
    INC DX
    CMP DX, 100
    JNE FOR2
    INC BX
    ADD CX,5
    CMP CX, 40
```

```
    JNE FOR1
    RET
COLORES:    DB 1100B
            DB 1010B
            DB 1001B
            DB 1110B
            DB 1111B
            DB 0111B
```

2.3.4. Tabla de vectores de interrupciones en la arquitectura x86 tradicional

Dirección del vector	Tipo	Función asociada
00-01	Excepción	Exception handlers
02	Excepción	Usada para errores críticos del sistema, no enmascarable
03-07	Excepción	Exception handlers
08	IRQ0	Timer del sistema
09	IRQ1	Puerto PS/2: Teclado
0A	IRQ2	Conectada al PIC esclavo
0B	IRQ3	Puerto serial
0C	IRQ4	Puerto serial
0D	IRQ5	Puerto paralelo
0E	IRQ6	Floppy disk
0F	IRQ7	Puerto paralelo
10	Int. de Software	Funciones de video
11-6F	Int. de Software	Funciones varias
70	IRQ8	Real time clock (RTC)
71 - 73	IRQ9-11	No tienen asociación estándar, libre uso
74	IRQ12	Puerto PS/2: Mouse
75	IRQ13	Coprocesador matemático
76	IRQ14	Controlador de disco 1
77	IRQ15	Controlador de disco 2
78-FF	Int. de Software	Funciones varias

2.3.5. Transferencia de datos entre I/O y memoria

Programmed I/O (PIO)

Aun considerando el uso de interrupciones, todavía existe una fuente de ineficiencia en la comunicación con los dispositivos de I/O. El problema ocurre cuando un dispositivo tiene que copiar datos a memoria (por ejemplo el disco duro). Supongamos que la CPU inicia la comunicación y le solicita al dispositivo copiar ciertos datos a memoria, el algoritmo sería de la siguiente forma:

1. Configurar el dispositivo para ser leído, enviando señales de control correspondientes.
2. Leer un dato del dispositivo, ocupando memory map o port I/O. El dato queda guardado en un registro de la CPU.
3. Copiar el registro desde la CPU a la memoria.
4. Revisar status del dispositivo para saber si quedan datos por leer. Si quedan, volver a 1. Si se terminó de leer, se pasa a los siguientes.

Esta forma de transferencia de datos se conoce como **Programmed I/O (PIO)**, ya que se realiza a través del programa. Se puede observar en este caso que en los pasos 2 y 3 la CPU actúa simplemente como un depósito intermedio de datos. El problema de esto es que la CPU debe estar ocupada realizando la transacción (que pueden ser muchos datos) perdiendo la posibilidad de hacer otra funcionalidad útil. Para solucionar esto, lo ideal es que el dispositivo pudiera acceder directamente a la memoria para copiar los datos y solamente contactarse con la CPU al comienzo y al final, mecanismo que se conoce como **Direct Memory Access**.

Direct Memory Access (DMA)

Para permitir que los dispositivos de I/O se comuniquen directamente con la memoria, se agrega un componente de hardware conocido como **controlador DMA**. El controlador DMA tendrá acceso al bus de datos y se encargará de traspasar la información que el dispositivo le envíe directamente a memoria sin pasar por la CPU (figura 11).

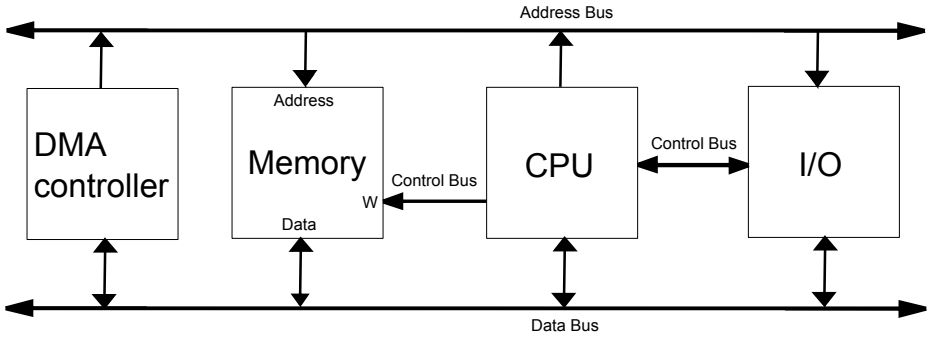

Figura 11: Computador con DMA compartido

El controlador se compone de los siguientes elementos:

- Registro de dirección destino: registro que tiene la dirección de memoria donde se copiará un dato. Esta dirección puede corresponder a la propia memoria, o estar mapeada a un I/O.
- Registro de dirección origen: registro que tiene la dirección de memoria desde donde se obtendrá un dato. Esta dirección puede corresponder a la propia memoria, o estar mapeada a un I/O.
- Registro contador de palabras: Almacena el número de palabras que quedan por enviar. Se decrementa automáticamente después de cada transferencia.
- Registro de control: Permite que la CPU configure el controlador
- Registro de estado: Contiene información del estado actual del proceso de transferencia.
- Buffer: para almacenar temporalmente los datos transferidos y otra información.
- Unidad de control: Contiene el programa de transferencia que se encarga de ir copiando los datos desde el dispositivo hacia la memoria. Cuando se termina la transferencia, envía una señal de interrupción a la CPU para indicarle que la transferencia ha finalizado.

Como ejemplo, supongamos que la CPU quiere guardar un archivo recién creado en disco. El algoritmo de transferencia de datos con DMA sería el siguiente:

1. CPU ejecuta un programa que quiere guardar el archivo

2. En el programa se configura el controlador DMA (probablemente mediante una interrupción de software dedicada a esto):
    - Se escribe en el registro de dirección origen la dirección de la primera palabra de memoria que se quiere copiar, del pedazo de memoria que contiene en este momento el archivo.
    - Se escribe en el registro de dirección destino la dirección que mapea la primera palabra del buffer del disco
    - Se escribe en el registro contador de palabras la cantidad de palabras a copiar.
    - Se escribe en el registro de control el comando que indica que se inicie el proceso.
3. CPU realiza otra acción mientras se ejecuta la transferencia (en un sistema de múltiples programas, se pasaría a ejecutar otro programa).
4. El controlador DMA ejecuta su programa interno encargado de transferir desde las direcciones origen a las destino, decrementando tras cada copia el registro contador.
5. Cuando el registro contador llega a 0, el controlador DMA envía una interrupción a la CPU.
6. La CPU atiende la interrupción del controlador, devolviendo el control al programa original para que siga funcionando, ya con el archivo guardado.

La arquitectura x86 implementa controladores de acceso directo a memoria para comunicarse con los dispositivos de I/O que transfieren gran cantidad de datos, como el disco duro y los discos ópticos. Los controladores de DMA son accedidos al igual que cualquier otro I/O, escribiendo y leyendo en los registros especiales del controlador.

3. Arquitectura de buses e I/O en x86

3.1. Conexión entre buses

Los computadores x86 contienen todos los elementos de comunicación con I/O antes descritos: memory mapped I/O, port I/O, interrupciones y DMA. Un diagrama lógico de todos estos elementos juntos en un computador se observa en la figura 12.

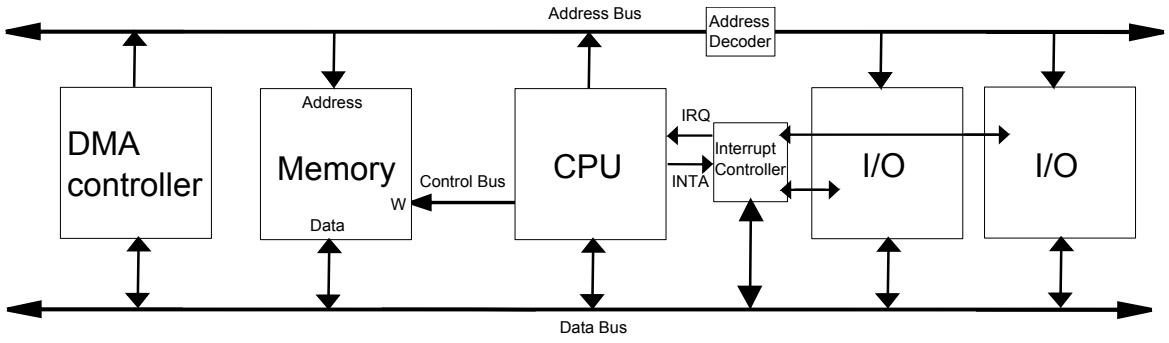

Figura 12: Diagrama de un computador con memory mapped I/O, interrupciones y DMA

En la práctica, las conexiones entre los buses que se observan en el diagrama no se pueden realizar directamente. La CPU, memoria y los distintos dispositivos de I/O tendrán diferentes velocidades, y por tanto se requieren componentes intermediarios, que sean capaces de coordinar la comunicación entre dispositivos. Estos componentes se conocen como **bridges** y la arquitectura x86 tiene dos: el **North Bridge** y el **South Bridge** (Figura 13).

El North Bridge es el controlador encargado de comunicar a la CPU, la RAM, la tarjeta gráfica y al resto de los dispositivos de I/O, conectados con el South Bridge. Habitualmente se le denomina **controlador de memoria**, ya que es el encargado de hacer la comunicación entre la RAM y la CPU. El North Bridge se caracteriza por trabajar con dispositivos y buses de alta velocidad, y por eso se encuentra separado del acceso directo a los I/O. El North Bridge funciona como Address Decoder, para manejar las instrucciones de acceso a I/O mapeadas a memoria que provengan de la CPU, y también incluye un controlador DMA.

El South Bridge es el encargado de comunicar a todo el resto de los I/O con la memoria y la CPU, a través del North Bridge, por lo que se le conoce como **controlador de I/O**. El rango de velocidades de los dispositivos que se conectan al South Bridge es amplio, pero en general los dispositivos serán más lentos que los conectados al North Bridge, lo que incluye discos, DVD, teclado, mouse, entre otros. Adicionalmente, al South Bridge se conectan dispositivos de I/O especiales, como el controlador de interrupciones y el controlador DMA.

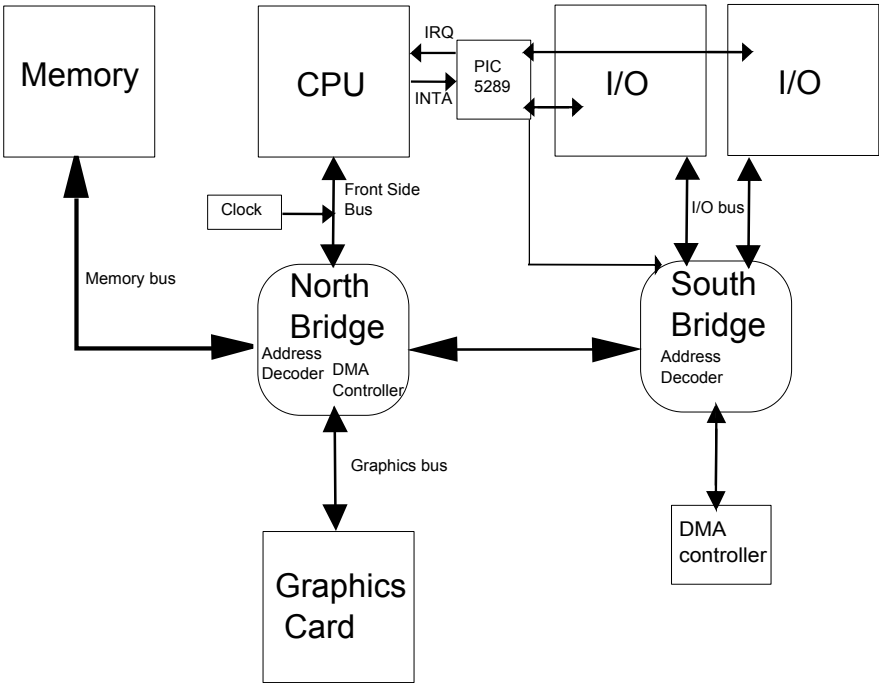

Figura 13: Diagrama de un computador con arquitectura x86

3.2. Buses de CPU y memoria

Como se observa en la figura 3, además de buses de I/O, existen buses especializados para comunicar a la CPU y a la memoria a través del North Bridge:

- **Front Side Bus (FSB)**: Es un bus especial, que se utiliza para conectar la CPU al North Bridge. Corresponde lógicamente al bus de datos y al bus de direcciones, es decir a través de este único bus se envían datos y direcciones, habitualmente por canales distintos. El clock del computador está conectado al FSB, el cual tiene frecuencias que van entre los 50MHz y los 400MHz, para computadores modernos. El clock que utiliza la CPU se basa en el clock del FSB, pero se amplifica la frecuencia ocupando un circuito multiplicador. Las velocidades habituales de transferencia de este bus oscilan entre 1GB/s y 10GB/s.

- **Memory Bus**: Corresponde al bus que comunica a la RAM con el North Bridge. Corresponde lógicamente al bus de datos y al bus de direcciones, es decir a través de este único bus se envían datos y direcciones, habitualmente por canales distintos. La velocidad variará según la tecnología de memoria que soporte el computador. Para memorias DDR el bus tendrá velocidades de alrededor de 1.6GB/s; para memorias DDR2, las velocidades serán alrededor de los 5GB/s.

3.3. Buses de I/O

Para conectar los distintos dispositivos de I/O al North Bridge y al South Bridge, existen distintos tipos de buses, con diferentes características que permiten adecuarse a los diversos I/O que pueden estar conectados al computador. Los principales buses de un computador son los siguientes:

- **Peripheral Component Interconnect (PCI)**: El bus PCI es un bus de uso interno en el computador que funciona con protocolo paralelo, alcanzando un ancho de banda de 533 MB/s.

- **PCI Express**: El bus PCI Express es un bus de uso interno en el computador, diseñado para reemplazar al bus PCI, entregando una mayor velocidad. El bus PCI Express funciona de manera serial, codificando en paquetes los datos. La unidad mínima de conexión en PCI Express se conoce como una línea, la cual permite trasmitir bidireccionalmente con un ancho de banda de 250 MB/s. Existen buses PCI Express tanto de una línea (1x) como de múltiples líneas (2x, 4x, 8x, 16x o 32x).

- **Parallel ATA (PATA)**: El bus Parallel AT Attachment (PATA), corresponde a un bus paralelo basado en el bus IDE, y como su nombre lo indica funciona enviando información de manera paralela. El bus PATA es usado principalmente para conectar discos (magnéticos y ópticos), y tiene un ancho de banda aproximado de 100 MB/s.

- **Serial ATA (SATA)**: Corresponde a la evolución del bus PATA, ocupando protocolo serial en vez de paralelo, con lo que logra alcanzar un ancho de banda de 300 MB/s. Es usado principalmente para conectar discos duros, reemplazando a PATA en ese uso.

- **USB 2.0**: El Universal Serial Bus 2.0, es un bus de bajo costo diseñado para la conexión con dispositivos externos al computador. El USB 2.0 se caracteriza por funcionar con protocolo serial, teniendo un ancho de banda de 60 MB/s y permitiendo entregar alimentación además de datos a través de sus señales.

- **Firewire (IEEE 1294)**: El estándar IEEE1294 define la implementación de un bus serial de alta velocidad (100 MB/s), usado para conectar dispositivos extensos que requieran transferir una gran cantidad de datos. El nombre Firewire proviene de la implementación específica de Apple del estándar.

Característica	PCI	PCI Express	PATA (IDE)	SATA
Ancho de banda	533 MB/s	250 MB/s por línea	100 MB/s	300 MB/s
Protocolo	Paralelo	Serial	Paralelo	Serial
Cantidad de cables	100	4 por línea	40	8
Máximo largo	< 0.5 metros	0.5 metros	1 metro	0.5 metros
Usado para conexiones	Internas	Internas	Internas	Internas
Uso habitual	Tarjeta de sonido	Tarjeta gráfica, tarjeta de red	Discos, DVD	Discos

Tabla 3: Resumen de buses internos principales de I/O

Característica	USB 2.0	Firewire
Ancho de banda	60 MB/s	100 MB/s
Protocolo	Serial	Serial
Cantidad de cables	4	6
Máximo largo	5 metros	4.5 metros
Usado para conexiones	Externas	Externas
Uso habitual	Múltiples dispositivos	Cámaras de video

Tabla 4: Resumen de buses externos principales de I/O

3.4. Tarjeta madre y Chipset

El lugar donde se ubican todos los componentes básicos y buses del computador se conoce como **tarjeta madre**, que es un circuito integrado que incluye slots especializados para conectar la CPU, memorias y distintos dispositivos de I/O. Uno de los elementos principales que contendrá la tarjeta madre es el **chipset** del computador que define que tipo de procesador, memoria e I/O se pueden conectar. En la arquitectura x86, quienes definen estas limitaciones serán el South Bridge y el North Bridge, circuitos que vendrán integrados en la placa madre, y por tanto se suele usar chipset para referirse a estos dos componentes. Un ejemplo de chipset se observa en la figura 14.

Uno de los componentes más importantes que viene incluido en la tarjeta madre es una memoria no volátil (ROM o Flash) donde se almacena la **BIOS**. La BIOS es un programa que se ocupa para inicializar los distintos procesos que permiten el funcionamiento de un computador. Debido a que cuando el computador está apagado la información del sistema operativo (el programa principal) se encuentra en disco (I/O), y que la RAM es no volátil, al encender un computador, la CPU no tiene acceso a las instrucciones del sistema operativo. Para lograr traspasar el programa del sistema operativo desde el disco a la RAM, al comenzar a funcionar el computador, se carga el programa que se encuentra almacenado en la ROM de la BIOS. De esta forma ese programa se encargará de realizar el proceso de levantamiento o **booting** del computador, copiando desde disco a la memoria las instrucciones del sistema operativo para que este pueda comenzar.

4. Referencias e información adicional

- Hennessy, J.; Patterson, D.: Computer Organization and Design: The Hardware/Software Interface, 4 Ed., Morgan-Kaufmann, 2008. Chapter 6: Storage and Other I/O Topics.
- Emu8086 Documentation
  http://www.itipacinotti.it/pagine/sistemi2008/documentation_emulator/index.html
- Ralph Brown Interrupt List http://www.cs.cmu.edu/ralf/files.html

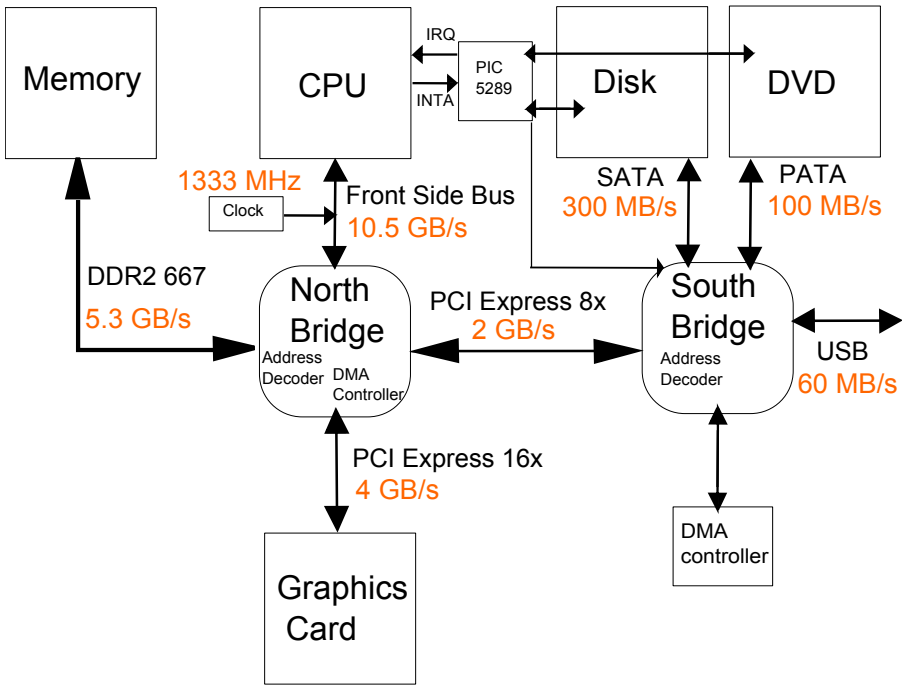

Figura 14: Diagrama de un computador con chipset Intel 5000P