DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

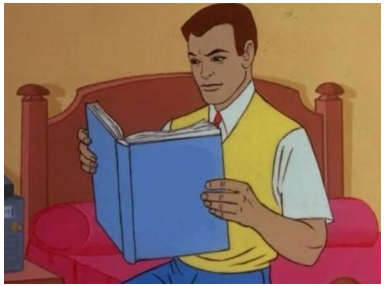
Clase 9 - Comunicación de CPU y Memoria con I/O

Profesor: Germán Leandro Contreras Sagredo

Bibliografía

- **Apuntes históricos. Hans Löbel, Alejandro Echeverría**
  - 08 - Comunicación de CPU y Memoria con IO

- **D. Patterson, Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann, 2020.**
  - Capítulo 6.9. Página 529.e1, 690 en PDF.



Objetivos de la clase

- Conocer, a nivel general, la composición de un dispositivo I/O.
- Conocer los mecanismos de comunicación entre el computador y los dispositivos I/O.
- Realizar ejercicios que consoliden los conocimientos anteriores.

Hasta ahora…

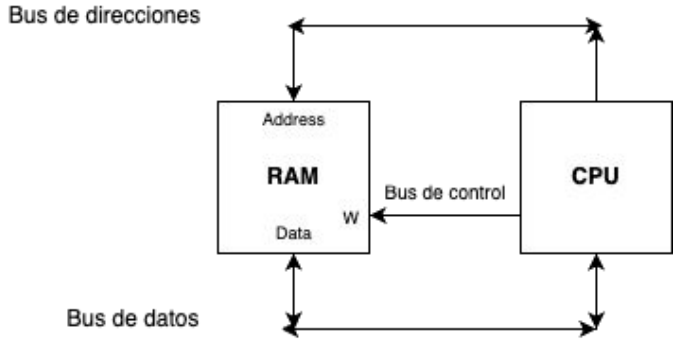
- Ya sabemos cómo construir nuestro computador básico y cómo programar con él a través de Assembly.

- Adicionalmente, conocimos la arquitectura RISC-V, de distinta complejidad.

No obstante lo anterior, hemos obviado un aspecto fundamental: **la interacción entre el computador y los usuarios, entre sus partes, con otros dispositivos y otros computadores**.

Microarquitectura del computador básico

- Al finalizar la construcción del computador básico, el diagrama abstracto se componía de la comunicación entre la CPU y la memoria a través de un bus de datos, un bus de direcciones y un bus de control.

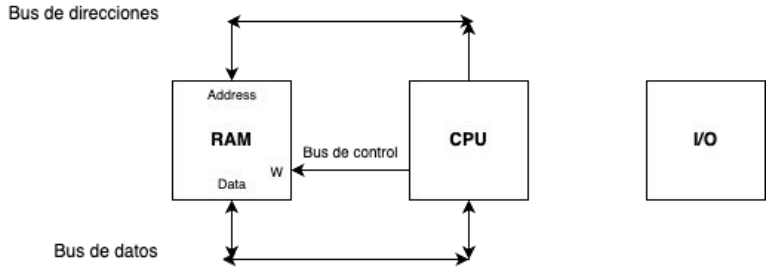


Por comodidad, usaremos la abstracción de la arquitectura Von Neumann.

- En la práctica, no obstante, **esto no es real**.

Microarquitectura del computador básico - Dispositivos I/O

- En la práctica, requerimos de ciertos elementos **externos** al procesador para poder hacer uso de este: teclado, pantalla, *mouse*, etc.



- Por lo tanto, el computador se compone de tres elementos principales: CPU, memoria y **dispositivos I/O**.

Dispositivos I/O - Estructura general

Si bien varían bastante entre ellos, los dispositivos I/O poseen dos elementos principales:

- **Elementos electromecánicos:** Encargados de las operaciones de interacción.
- **Controladores:** Pieza de *hardware* que regula el comportamiento de los elementos electromecánicos y, a su vez, la comunicación con el computador. El computador puede solicitar comandos al controlador de un dispositivo a través del ***driver***, *software* instalado en el sistema operativo.

Dispositivos I/O - Estructura general

Los dispositivos I/O poseen un espacio de memoria (*buffer*) con segmentos (registros) de memoria con funcionalidades distintas:

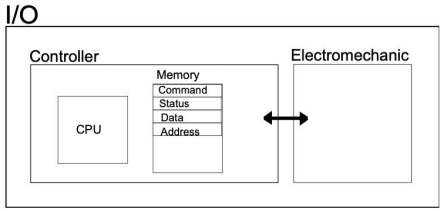


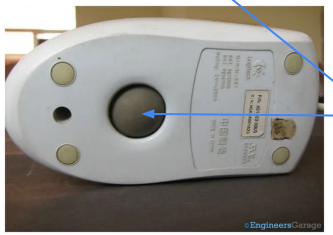
Diagrama genérico de un dispositivo I/O.

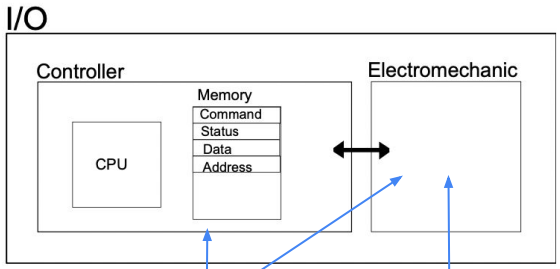
- **Registros de comando**, sobreescritos por el computador para que el dispositivo ejecute una acción.
- **Registros de estado**, para otorgar información al computador.
- **Registros de datos**, para escribir datos sobre el dispositivo o transferirlos hacia el computador.
- **Registros de dirección**, para direccionar los datos recibidos dentro del dispositivo (por ejemplo, un disco duro).

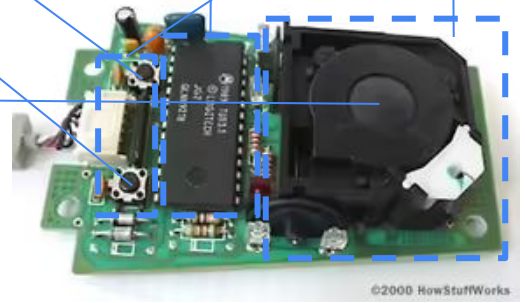
Dispositivos I/O - Algunos ejemplos

***Mouse*:** Ahora usan sensores, pero antes usaban “bolitas” y, mediante discos y luces infrarrojas, aproximaban la velocidad y sentido del movimiento para posicionar el cursor en pantalla.

Más información general aquí y sobre el posicionamiento aquí.





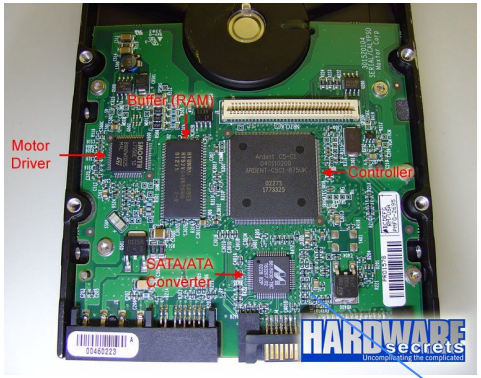


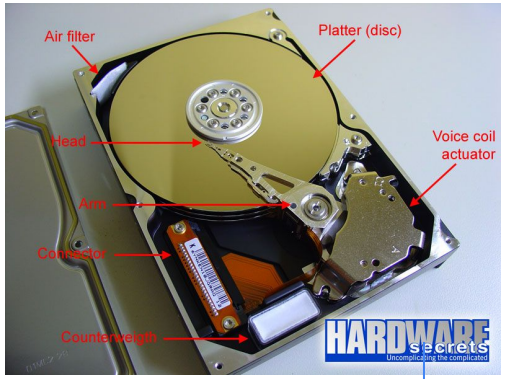
El chip corresponde al controlador (el computador del *mouse*)

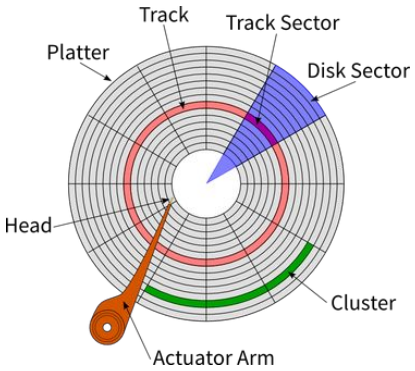
Dispositivos I/O - Algunos ejemplos

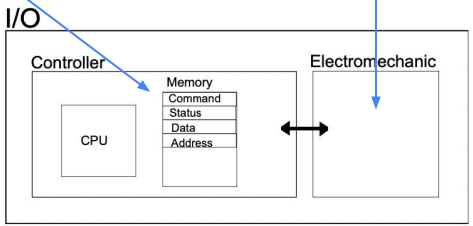
Disco duro: Leen/escriben datos según la posición de un cabezal sobre un plato de metal magnético. Posee un esquema más complejo al ser varias las componentes electromecánicas usadas.

Más información general aquí.





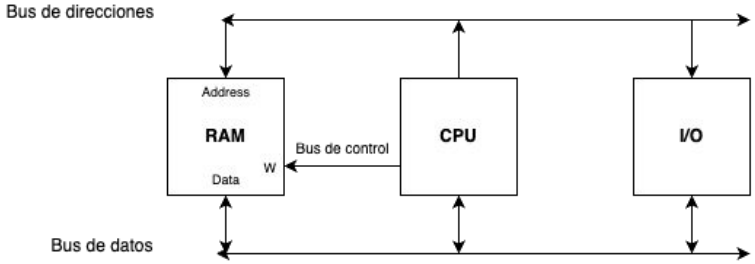




Dispositivos I/O - Comunicación con CPU y Memoria

- Volviendo al diagrama general, un dispositivo I/O se comunica con la CPU y la memoria a través de los mismos buses de datos y direcciones que estos comparten.

- Ahora, existen distintos tipos de comunicación y mecanismos que debemos considerar **a falta de un bus de control**.



El bus de datos del dispositivo I/O es bi-direccional (ya que podemos querer obtener datos de este o transferirlos). El bus de direcciones es unidireccional dado que la CPU se encarga de configurarlas. Al no contar con un bus de control para el dispositivo I/O, debemos elaborar mecanismos que permitan que este se comunique con la CPU y vice-versa.

Dispositivos I/O - Comunicación con CPU y Memoria

Tipos de comunicación

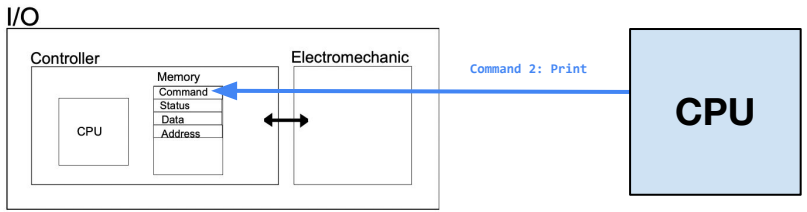
- **Comunicación de comandos:** CPU → I/O
- **Comunicación de estado:** I/O → CPU
- **Transferencia de datos:** Memoria ↔ I/O

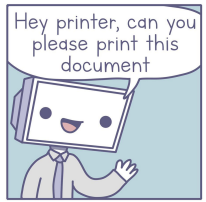
De momento, vamos a asumir que la transferencia de datos también involucra a la CPU, pero más adelante **mejoraremos este esquema**.

Dispositivos I/O - Comunicación con CPU y Memoria

Tipos de comunicación - Comunicación de comandos

- CPU “ordena” a un dispositivo I/O que ejecute un comando.
- La CPU escribe sobre el **registro de comandos** del dispositivo I/O el valor asociado al comando deseado para su ejecución.
- **Ejemplo:** Solicitar a la impresora la impresión de un documento.

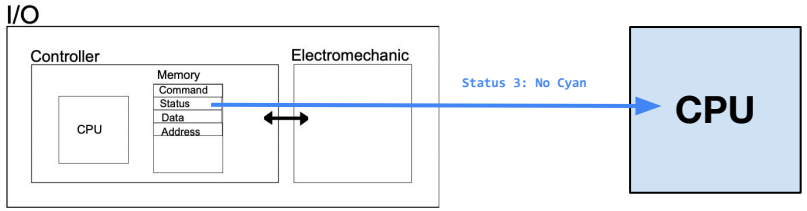


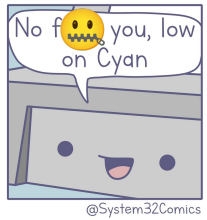


Dispositivos I/O - Comunicación con CPU y Memoria

Tipos de comunicación - Comunicación de estado

- Dispositivo I/O “avisa” su estado a la CPU.
- Dispositivo I/O actualiza su estado en su **registro de estado** y luego la CPU lee el valor de dicha dirección.
- **Ejemplo:** Impresora avisa que le falta tinta de color “cyan”.

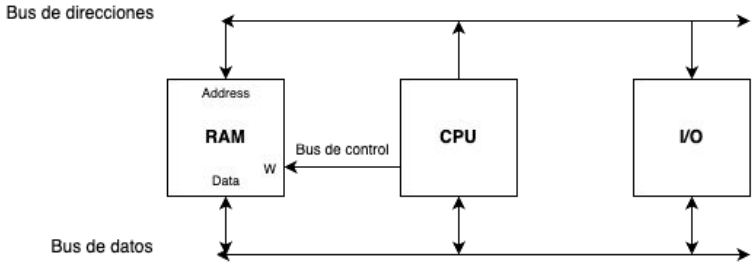




Dispositivos I/O - Comunicación con CPU y Memoria

- Si bien un dispositivo I/O usa el mismo bus de direcciones de la memoria principal y la CPU, **sus direcciones aluden a dispositivos distintos de almacenamiento**.

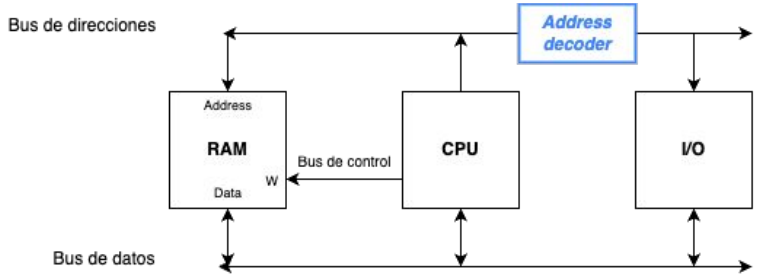
- ¿Cómo se diferencian las direcciones? Existen dos esquemas para resolver este problema.

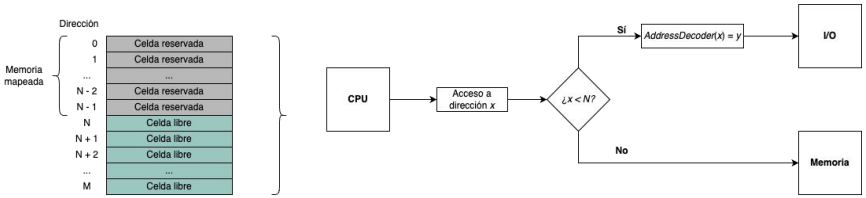


Comunicación I/O - *Memory mapped* I/O

- Este esquema evita tener que modificar la ISA del computador. Se reservan *N* espacios de memoria para direcciones de registros de dispositivos I/O.

- Si la CPU usa una dirección del espacio mapeado, el *address decoder* lo transforma a la dirección “real” del dispositivo para que se acceda a este.





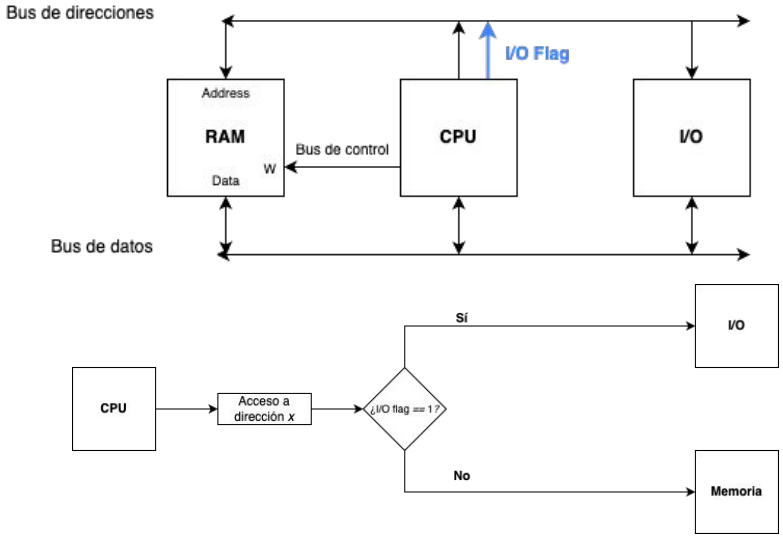
La gran desventaja de este esquema es la limitación del espacio direccionable, lo que se conoce como *memory barrier*. Aunque las celdas de la memoria mapeada estuvieran disponibles, en este esquema no se pueden ocupar. Mientras más direcciones de dispositivos I/O se ocupen, el espacio de memoria se verá más reducido.

Comunicación I/O - *Port mapped* I/O

- Este esquema modifica la ISA del computador añadiendo instrucciones nuevas para interactuar con dispositivos I/O.

- Se utiliza un nuevo espacio de direccionamiento (puertos) que apunta a direcciones de dispositivos I/O y se agrega una *flag* para indicar si una dirección alude a un puerto o no.

* Otra alternativa a la I/O *Flag*, es el uso de un bus de direcciones separados del utilizado para acceder a la memoria.

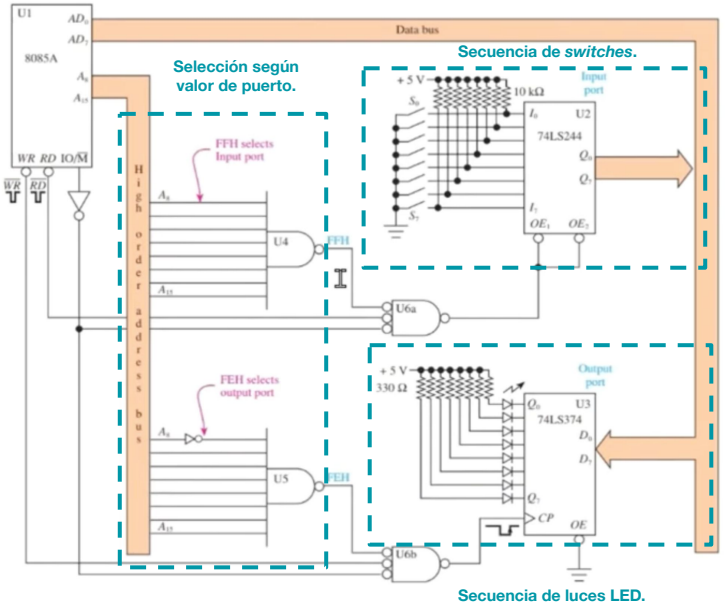


La cantidad de puertos varía según la implementación. En la arquitectura x86, se posee un direccionamiento de puertos de 16 bits (*i.e.* 65.536 puertos).

Comunicación I/O - *Port mapped* I/O

Ejemplo de implementación

- Se accede solo a dos dispositivos: uno de entrada, correspondiente a una secuencia de 8 *switches*; y uno de salida, correspondiente a una secuencia de 8 luces LED.
- A partir del puerto, se puede seleccionar como ingreso al bus de datos el valor de la secuencia de *switches*; o bien la escritura sobre la secuencia de luces LED.



Comunicación I/O - *Port mapped* I/O

Las instrucciones de I/O varían según la implementación. Si tomamos como ejemplo la arquitectura x86, tenemos dos principales:

- `IN Reg, Port:` Escribe en el registro `Reg` el valor almacenado en el puerto `Port`, una dirección de un dispositivo I/O. Se utiliza para **leer datos desde un dispositivo** (por ejemplo, su estado).

- `Out Port, Reg:` Escribe en el puerto `Port` (dirección de un dispositivo I/O) el valor almacenado en el registro `Reg`. Se utiliza para **escribir datos en un dispositivo** (por ejemplo, un comando).

Comunicación I/O - Ejercicio en clases

Asuma que la ISA del computador básico posee las instrucciones de *port mapped* I/O implementadas para los registros *A* y *B*. A su vez, asuma que el computador básico utiliza los dos mecanismos de comunicación con dispositivos I/O vistos para conectarse a una impresora usando las siguientes direcciones y puertos:

Dirección de memoria	Dirección I/O
`0x70`	Registro de estado impresora.
`0x71`	Registro de comando impresora.

Esquema *memory mapped*

Puerto	Dirección I/O
`0x00`	Registro de modo de color impresora.
`0x01`	Registro de tamaño de papel impresora.
`0x02`	Registro de orientación de impresión impresora.

Esquema *port mapped*

Comunicación I/O - Ejercicio en clases

Por otra parte, la impresora posee las siguientes especificaciones:

Valor registro	Estado
0x00	Impresora apagada.
0x01	Impresora prendida.
0x02	Impresora imprimiendo.

Valor registro	Comando
0x00	Apagar impresora.
0x01	Prender impresora.
0x02	Imprimir.

Valor registro	Modo de color
0x00	Blanco y negro.
0x01	Color.

Valor registro	Tamaño
0x00	Carta.
0x01	Oficio.

Valor registro	Orientación
0x00	Vertical.
0x01	Horizontal.

Comunicación I/O - Ejercicio en clases

A partir del esquema anterior, elabore un programa con el Assembly del computador básico que realice lo siguiente:

- Que prenda la impresora si está apagada.
- Si está imprimiendo, que termine el programa.
- Si no está imprimiendo, que imprima una hoja a color en tamaño oficio y en orientación vertical antes de terminar el programa.

Comunicación I/O - Ejercicio en clases

Manejo de impresora - Assembly computador básico

```
DATA:                   ;Se podrían guardar las direcciones como variables,
CODE:                   ;pero usaremos directamente los literales.
  main:
    MOV B,112           ;B = 0x70 = Registro de estado
    MOV A,(B)
    CMP A,2
    JEQ end             ;Está imprimiendo, no hacemos nada.
    CMP A,1
    JEQ print           ;Está prendida, imprimimos.
  turnOn:               ;Está apagada, la prendemos e imprimimos.
    MOV B,113           ;B = 0x71 = Registro de comando.
    MOV A,1
    MOV (B),A           ;Prender impresora, luego imprimir.
  print:
    MOV A,0
    OUT 2,A             ;Orientación vertical
    MOV A,1
    OUT 0,A             ;Impresión a color
    OUT 1,A             ;Tamaño oficio
    MOV B,113           ;B = 0x71 = Registro de comando
    MOV A,2
    MOV (B),A           ;Imprimir
  end:
```

Posible respuesta en Assembly del computador básico. Cabe destacar que se usa direccionamiento indirecto para los registros de la impresora mapeados en memoria (estado y comando); mientras que se utilizan las instrucciones de *port mapped* I/O para los registros que tienen puertos asignados (color, tamaño y orientación).

Comunicación I/O - Ejercicio en clases

¿Podemos escribir el código anterior en RISC-V?

No directamente. En esta ISA, no contamos con instrucciones para interactuar con dispositivos *port-mapped*. Por otra parte, si usamos registros de 32 bits, cada dispositivo usará cuatro direcciones de memoria y no una. Ajustaremos las tablas para que todos los dispositivos sean *memory-mapped* de 32 bits.

Veamos cómo queda el programa.

Dirección de memoria	Dirección I/O
`0x70`	Registro de estado impresora.
`0x74`	Registro de comando impresora.
`0x78`	Registro de modo de color impresora.
`0x7C`	Registro de tamaño de papel impresora.
`0x80`	Registro de orientación de impresión impresora.

Comunicación I/O - Ejercicio en clases

Manejo de impresora - Assembly RISC-V

```
.text                   # Se podrían guardar las direcciones como variables,
  main:                 # pero usaremos directamente los literales.
    li s0, 112          # s0 = 0x70 = Registro de estado
    li s1, 2            # s1 = Estado “imprimiendo”
    li s2, 1            # s2 = Estado “prendida”
    li s3, 1            # s3 = Comando “prender”
    li s4, 1            # s4 = Impresión a color
    li s5, 1            # s5 = Tamaño oficio
    li s6, 0            # s6 = Orientación vertical
    li s7, 2            # s7 = Comando “imprimir”
    lw t0, 0(s0)        # t0 = Valor de registro de estado
    beq t0, s1, end     # Está imprimiendo, no hacemos nada.
    beq t0, s2, print   # Está prendida, imprimimos.
  # Está apagada, la prendemos e imprimimos.
  turnOn:               # Prender impresora, luego imprimir.
    sw s3, 4(s0)        # s0 + 4 = 0x74 = Registro de comando.
  print:
    sw s4, 8(s0)        # s0 + 8  = 0x78 = Registro de modo de color.
    sw s5, 12(s0)       # s0 + 12 = 0x7C = Registro de tamaño de papel.
    sw s6, 16(s0)       # s0 + 16 = 0x80 = Registro de orientación impresión.
    sw s7, 4(s0)        # Imprimir.
  end:
```

Posible respuesta en Assembly RISC-V. Cabe destacar que se usa direccionamiento indirecto para acceder a todos los registros. Basta con guardar la dirección base y acceder a todos ellos a través del *offset*.

Comunicación I/O - Modos de comunicación

Hasta ahora, hemos visto cómo leer datos de dispositivos I/O y cómo escribir datos sobre ellos a través de dos mecanismos distintos.

¿En qué momento ocurre esta interacción?

- ¿Cómo sabe el computador si cambió la posición del cursor?
- ¿Cuándo se revisa si se presionó una tecla?
- ¿Cómo sabemos cuándo termina una transferencia de datos?

Las preguntas anteriores aluden a la naturaleza **dinámica** de la interacción entre el computador, la memoria y los dispositivos I/O.

Comunicación I/O - Comunicación vía *polling*

Un primer acercamiento a la forma de llevar a cabo la comunicación sería que la CPU, cada cierto tiempo, consulte el estado de los dispositivos I/O para ver si requiere realizar alguna acción. Esto se conoce como ***polling***.

Su ventaja radica en que **no es necesario realizar cambios sobre la arquitectura para su implementación**.

Comunicación I/O - Ejercicio en clases

Suponga que a la impresora del ejercicio anterior se le añade un nuevo estado “Impresora prendiendo” asociado al valor 0x03.

Modifique el código desarrollado en el Assembly del computador básico y RISC-V para que, mediante *polling*, la impresión se lleve a cabo solo si la impresora ya está en estado “Impresora prendida”.

Comunicación I/O - Ejercicio en clases

Manejo de impresora con *polling* - Assembly computador básico

```
DATA:                   ;Se podrían guardar las direcciones como variables,
CODE:                   ;pero usaremos directamente los literales.
  main:
    MOV B,112           ;B = 0x70 = Registro de estado
    MOV A,(B)
    CMP A,2
    JEQ end             ;Está imprimiendo, no hacemos nada.
    CMP A,1
    JEQ print           ;Está prendida, imprimimos.
    CMP A,3
    JEQ waitTurningOn   ;Está prendiendo, esperamos a que prenda.
  turnOn:               ;Está apagada, la prendemos e imprimimos.
    MOV B,113           ;B = 0x71 = Registro de comando.
    MOV A,1
    MOV (B),A           ;Prender impresora, luego imprimir.
  waitTurningOn:
    MOV B,112           ;B = 0x70 = Registro de estado
    MOV A,(B)
    CMP A,1
    JNE waitTurningOn   ;Si todavía no está prendida, repetimos la pregunta.
  print:
    MOV A,0
    OUT 2,A             ;Orientación vertical
    MOV A,1
    OUT 0,A             ;Impresión a color
    OUT 1,A             ;Orientación horizontal
    MOV B,113           ;B = 0x71 = Registro de comando
    MOV A,2
    MOV (B),A           ;Imprimir
  end:
```

Posible respuesta en Assembly del computador básico. Ahora, revisamos el nuevo estado “3” para ver si la impresora está prendiendo. En este caso, ejecutamos un *loop* que solo revisa el estado hasta que cambie a “Impresora prendida” (cuyo valor es “1”).

Comunicación I/O - Ejercicio en clases

Manejo de impresora con *polling* - Assembly RISC-V

```
.text                          # Se podrían guardar las direcciones como variables,
  main:                        # pero usaremos directamente los literales.
    li s0, 112                 # s0 = 0x70 = Registro de estado
    li s1, 2                   # s1 = Estado “imprimiendo”
    li s2, 1                   # s2 = Estado “prendida”
    li s3, 3                   # s3 = Estado “prendiendo”
    li s4, 1                   # s4 = Comando “prender”
    li s5, 1                   # s5 = Impresión a color
    li s6, 1                   # s6 = Tamaño oficio
    li s7, 0                   # s7 = Orientación vertical
    li s8, 2                   # s8 = Comando “imprimir”
    lw t0, 0(s0)               # t0 = Valor de registro de estado
    beq t0, s1, end            # Está imprimiendo, no hacemos nada.
    beq t0, s2, print          # Está prendida, imprimimos.
    beq t0, s3, waitTurningOn  # Está prendida, imprimimos.
  # Está apagada, la prendemos y esperamos a que prenda para imprimir.
  turnOn:                      # Prender impresora, luego esperar a que prenda.
    sw s4, 4(s0)               # s0 + 4 = 0x74 = Registro de comando.
  waitTurningOn:
    lw t0, 0(s0)
    beq t0, s3, waitTurningOn # Si todavía no está prendida, repetimos la pregunta.
  print:
    sw s5, 8(s0)               # s0 + 8  = 0x78 = Registro de modo de color.
    sw s6, 12(s0)              # s0 + 12 = 0x7C = Registro de tamaño de papel.
    sw s7, 16(s0)              # s0 + 16 = 0x80 = Registro de orientación impresión.
    sw s8, 4(s0)               # Imprimir.
  end:
```

Posible respuesta en Assembly RISC-V.

Comunicación I/O - Comunicación vía *polling*

Este modo de comunicación, por simple que sea su implementación, presenta desventajas importantes:

- Si la periodicidad de consulta es alta, la CPU ocupa poco tiempo para ejecutar otro tipo de tareas.
- Si la periodicidad es baja, la actualización de operaciones según el estado de cada dispositivo I/O podría ser lenta.

Por este motivo, se opta por usar otro modo de comunicación que mejora estos problemas.

* Esto no significa que nunca se utilice. Por ejemplo, la comunicación a través de los puertos USB se realiza vía ***polling***.

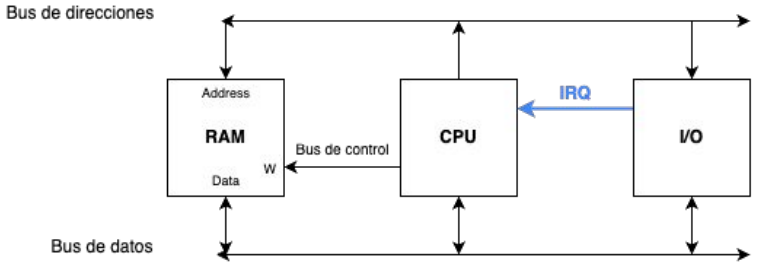
Comunicación I/O - Comunicación vía interrupciones

En este modo de comunicación, **los dispositivos I/O se encargan de avisar a la CPU que requieren su atención**. Esto se hace con una **interrupción**.

A continuación veremos cómo se realiza esto a nivel de *hardware*.

Comunicación I/O - Comunicación vía interrupciones

- Se define una nueva señal llamada **IRQ** (***I**nterrupt **R**equest*).
- Si un dispositivo I/O requiere de la CPU, emite esta señal con valor 1 para que esta, en base a su disponibilidad, la atienda.
- La “atención” se traduce en la ejecución de una subrutina: **ISR** (***I**nterrupt **S**ervice **R**outine*).



Adicionalmente, se define un nuevo *flag* para el registro Status llamado *Interruption Flag* (IF). Este indica si la CPU está disponible para atender interrupciones I/O (IF = 1) o no (IF = 0).

Comunicación I/O - Comunicación vía interrupciones

Pasos a seguir después de una interrupción

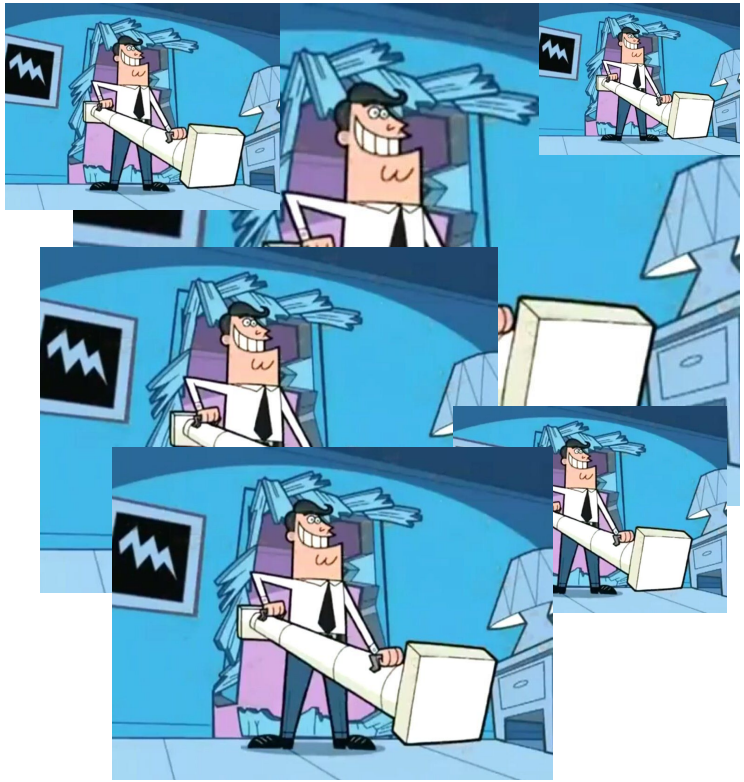
1. Dispositivo interrumpe (IRQ = 1, transmitida a la CPU).
2. CPU termina la ejecución de la instrucción actual y guarda sus *condition codes* (registro *Status*) en el *stack*.
3. CPU revisa si el *flag* de interrupciones está activo (IF = 1). Si no es el caso, saltamos al paso 11.
4. CPU deshabilita la atención de más interrupciones (IF = 0).
5. CPU llama a la ISR asociada al dispositivo (`CALL`).
6. ISR respalda el estado actual de la CPU (x86: `PUSHA` respalda en el *stack* todos los registros de propósito general).
7. ISR ejecuta su código.
8. ISR devuelve el estado previo a la CPU (x86: `POPA` restaura desde el *stack* todos los registros de propósito general).
9. ISR retorna (x86: `IRET`).
10. CPU habilita la atención de interrupciones (IF = 1).
11. CPU recupera *condition codes* desde el *stack* y retoma su ejecución.

* En el paso 3, la revisión del *flag* IF modifica los *condition codes* al hacer una comparación en la ALU. Por ese motivo es importante respaldarlos antes.

Comunicación I/O - Comunicación vía interrupciones

En la práctica, existen diversos dispositivos I/O comunicados con la CPU que pueden estar enviando solicitudes de interrupción de forma simultánea.

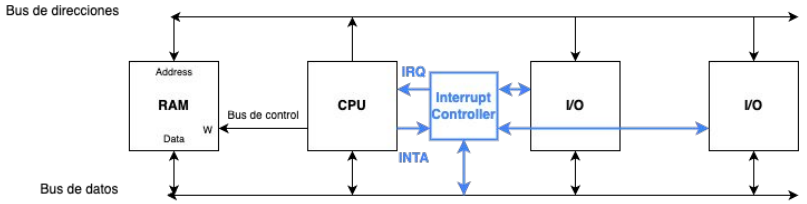
Para manejar este caso, haremos uso de piezas de *hardware* adicionales para manejar este conjunto de interrupciones.

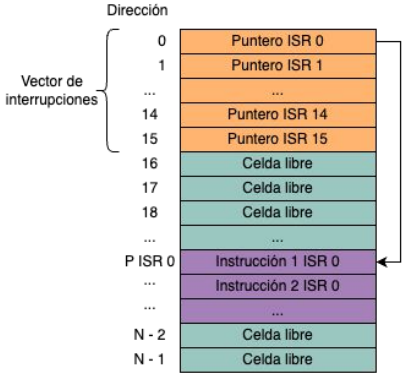


Comunicación I/O - Comunicación vía interrupciones

- Se añade el dispositivo I/O **controlador de interrupciones**. Este recibe las solicitudes del resto de dispositivos I/O y las transmite a la CPU.

- Si la CPU atiende una interrupción, envía una señal **INTA** (***Int****errupt* ***A****cknowledge*) al controlador para que le transmita el ID del dispositivo asociado.





Al existir varios dispositivos, existen también varias ISR ejecutables. Las manejamos con el vector de interrupciones, una tabla almacenada en memoria con punteros de la dirección de la ISR de cada dispositivo I/O. La CPU conoce la dirección base del vector, por lo que utiliza el ID de cada dispositivo como *offset* para obtener la dirección de la primera instrucción de su ISR.

Trivia: ¿Por qué no guardar directamente las ISR en vez de un vector?
- ISRs de largo variable, dificultad en el direccionamiento.
- ISRs pueden actualizarse, habría que actualizar las direcciones de inicio.

El vector nos otorga más versatilidad para el manejo de ISRs.

Comunicación I/O - Comunicación vía interrupciones

Un controlador de interrupciones tendrá, **al menos**, los siguientes componentes:

- Registros de **comando** y **estado**.
- ***Interrupt Request Register*:** Registro de interrupciones en **espera de atención**.
- ***In-Service Register*:** Registro de interrupciones **en atención**.
- ***Interrupt Mask Register*:** Registro de **enmascaramiento de interrupciones**. I/O enmascarado = **ignorado en la interrupción**.
- Circuito para manejar **prioridades de interrupciones**.

Comunicación I/O - Comunicación vía interrupciones

Respecto a los registros de interrupción, en estos cada bit representa el IRQ de un dispositivo. Entonces, si tuviéramos 8 dispositivos I/O, estos registros tendrían 8 bits representando 8 IRQS (IRQ0-IRQ7).

Ejemplo:

Interrupt Request Register = 00110100
In-Service Register = 01000000
Interrupt Mask Register = 00000001

- IRQ1 está siendo atendida.
- IRQ2, IRQ3, IRQ5 esperando atención.
- IRQ7 enmascarada. Aunque se le envíe la señal IRQ7 al controlador, nunca será considerada en el *Interrupt Request Register* a menos que el valor del registro de enmascaramiento cambie.

Trivia: ¿Puede haber más de un dispositivo siendo atendido? *i.e.* más de un bit activo en el *In-Service Register*.
R: Sí. Las arquitecturas antiguas generalmente admitían un único dispositivo a ser atendido a la vez, no obstante, actualmente existen técnicas más sofisticadas de cambio de contexto entre procesos que lo permiten. Esto es independiente de la cantidad de *cores* de la CPU.

Comunicación I/O - Comunicación vía interrupciones

Pasos a seguir después de una interrupción con controlador de interrupciones

1. Dispositivo I/O envía señal IRQ al controlador.
2. Controlador revisa su registro de enmascaramiento. Si la interrupción no está enmascarada, la atiende marcando un 1 en el bit correspondiente del *Interrupt Request Register*.
3. Controlador decide la interrupción actual prioritaria (por ejemplo, considerando el ID menor) y marca un 1 en el bit correspondiente del *In-Service Register*.
4. Controlador envía interrupción (x86: Instrucción `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 este caso, la atiende.
7. CPU deshabilita la atención de más interrupciones (IF = 0).
8. CPU envía señal INTA al controlador para saber qué dispositivo interrumpió.
9. Controlador revisa el ID activo según el *In-Service Register* y lo envía de vuelta a la CPU.
10. CPU usa el ID para buscar la dirección de la ISR en el vector de interrupciones.
11. CPU llama a la ISR asociada al dispositivo usando el ID (`CALL Mem[vectInt + ID]`).
12. ISR respalda el estado actual de la CPU.
13. ISR ejecuta su código.
14. ISR envía un comando EOI (***E**nd **o**f **I**nterrupt*) al controlador, configurando en 0 su bit en el *In-Service Register*.
15. ISR devuelve el estado previo a la CPU.
16. ISR retorna.
17. CPU habilita la atención de interrupciones (IF = 1).
18. CPU recupera *condition codes* desde el *stack* y retoma su ejecución.

Comunicación I/O - Comunicación vía interrupciones

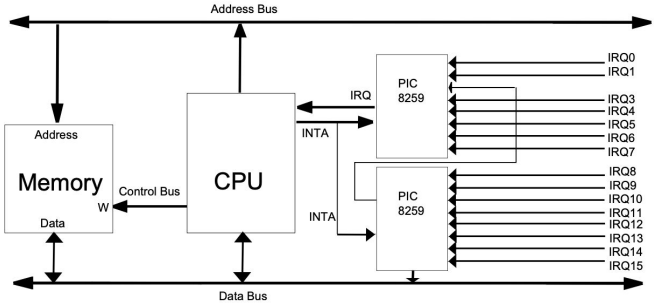
Implementación real: x86

- Se ocupan controladores PIC (*Programmable Interrupt Controller*).
- Una PIC acepta hasta 8 IRQs. Se utilizan dos, una primaria y secundaria conectadas en cascada.

Chip del PIC8259.

Conectores PS/2

Diagrama general.



IRQ	Dispositivo	Vector de interrupción
IRQ0	Timer del sistema	08
IRQ1	Puerto PS/2: Teclado	09
IRQ2	Conectada al PIC secundario	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

IRQs de la arquitectura x86 tradicional.

Comunicación I/O - Comunicación vía interrupciones

Implementación real: x86

- En el vector de interrupciones no solo están las direcciones de las ISR de los dispositivos I/O.
- Además de las interrupciones de *hardware*, contamos con **interrupciones gatilladas en la CPU**.

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 secundario
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

Vector de interrupciones de la arquitectura x86 tradicional. Notar que se señalan, además, excepciones e interrupciones de *software*.

Comunicación I/O - Comunicación vía interrupciones

Tipos de interrupción en la CPU

- **Excepciones:** Condiciones de error al ejecutar una instrucción. Estas son atendidas por la CPU (en particular, el sistema operativo) a través de ISRs específicas: ***exception handlers***.
  Ejemplos: división por cero, *stack overflow*.
- **Interrupciones de *software* (*traps*):** Son "llamadas al sistema operativo", interrupciones **explícitas** en los programas para ceder el control al sistema operativo para que realice una acción.
  Ejemplo en x86: Instrucciones `CLI`, `STI`, `INT dir` para *setear* en 0 la *flag* IF, *setearla* en 1 o para llamar una ISR, respectivamente.
  Ejemplo en RISC-V: `ecall`.

Comunicación I/O - Comunicación vía interrupciones

Paréntesis: Sistema operativo

- Al ejecutar la ISR de un dispositivo (o cualquier tipo de interrupción), se le cede el control al **sistema operativo** para que maneje la CPU y ejecute el programa.

- El vector de interrupciones se almacena en una **sección protegida** de la memoria, la que solo es accesible si el **modo protegido** (o "**modo kernel**") está activo. Cuando lo está, el sistema operativo tiene el control.

* Si el usuario tuviera acceso directo a los dispositivos, sería fácil caer en errores de sistema por mal funcionamiento.

Comunicación I/O - Comunicación vía interrupciones

Paréntesis: Sistema operativo

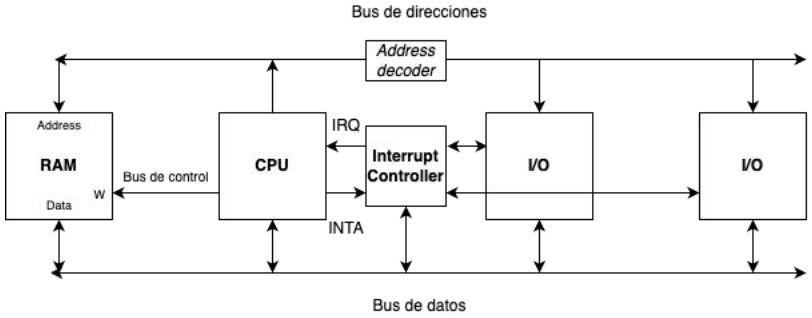
- Por otra parte, cuando prendemos un computador, lo primero que se hace es ejecutar un programa de inicio (generalmente almacenado en una ROM): el **BIOS** (***B**asic **I**nput **O**utput **S**ystem*) o la **UEFI** (***U**nified **E**xtensible **F**irmware **I**nterface*).

- Este programa (ya sea BIOS o UEFI), a grandes rasgos, se encarga de revisar y activar todos los dispositivos del computador y, posteriormente, carga al inicio de la memoria la sección de inicio del sistema operativo, para luego cederle el control.

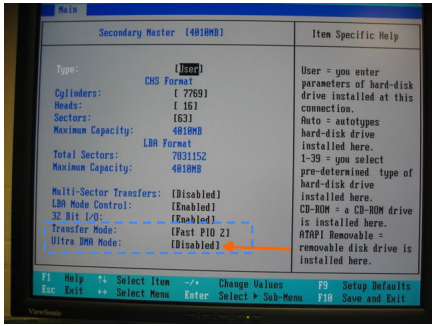
Comunicación I/O - Transferencia de datos

- Toda interacción que involucre lectura o escritura de datos entre la memoria y un dispositivo I/O se considerará una **transferencia de datos**.

- En nuestro esquema actual, la CPU puede "orquestar" la transferencia mediante comandos y atención a interrupciones.



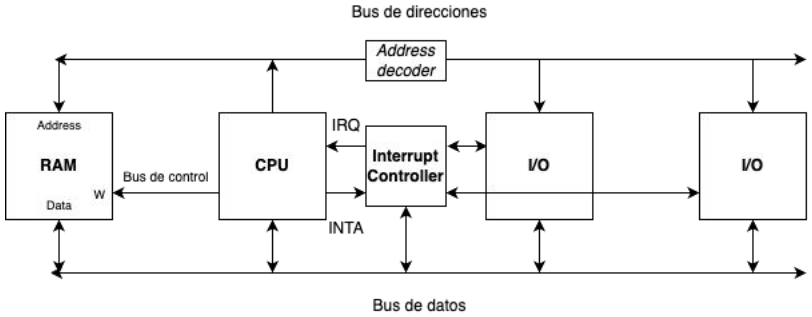
El esquema donde la transferencia de datos se realiza a nivel de código se conoce como *Programmed* I/O o PIO.



Comunicación I/O - Transferencia de datos

Si bien no requerimos *hardware* adicional, esto presenta desventajas:

- La CPU interrumpe su ejecución para atender la transferencia.
- Los datos a transferir no necesitan procesamiento de la CPU, solo deben transmitirse.
- La transferencia **toma un tiempo no menor**.

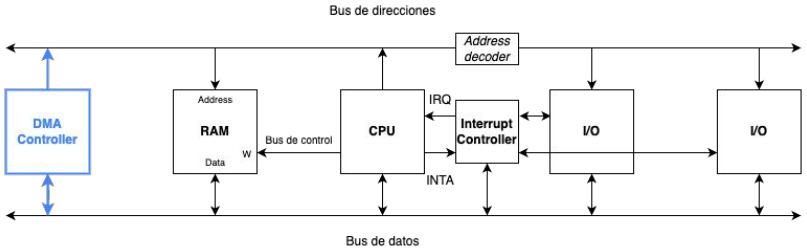


¿Cómo evitamos que la CPU se involucre?

Comunicación I/O - Transferencia de datos

DMA: *Direct Memory Access*

- Controlador con acceso al bus de datos encargado de traspasar información entre la memoria principal y los dispositivos.
- Ejecuta un programa encargado de la transferencia y notifica a la CPU de su término a través de una interrupción.



La CPU, a través del SO, solo programa la transferencia y la finaliza con la interrupción a través de *callbacks*.

Comunicación I/O - Comunicación vía interrupciones

Un controlador DMA tendrá, **al menos**, los siguientes componentes:

- Registros de **comando** y **estado**.
- Registros de **dirección de origen** y **destino***.
- Registro **contador de palabras**. Cuando llega a 0, se genera la interrupción de término de transferencia.
- *Buffer* de **almacenamiento temporal**, utilizado en transferencia de memoria entre dispositivos de distintas velocidades.
- Unidad de control, incluyendo su propia ISA.

*** Como el DMA trabaja con direcciones de memoria, solo puede utilizar dispositivos mapeados en memoria.**

Comunicación I/O - Conexión de buses

Una última característica que vale la pena señalar es la conexión entre buses.

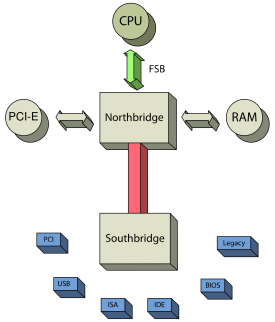
En la práctica, distintos dispositivos I/O tienen distintas tasas de transferencia de datos. Por ejemplo, podemos obtener datos más rápido de un *pendrive* que de un disco duro magnético. También influye el tipo de bus que utilizan (USB, PCIe, SATA, etc.).

Para coordinar la comunicación entre dispositivos según su tipo de bus y tasa de transferencia de datos, se introducen los **puentes** o ***bridges***.

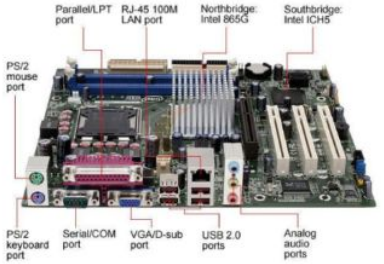
Comunicación I/O - Conexión de buses

Bridges

- *Hardware* que permite coordinar la comunicación entre dispositivos. Generalmente se utilizan dos: ***North Bridge*** y ***South Bridge***.
- El ***North Bridge*** o **controlador de memoria** se conecta con la RAM y dispositivos de alta velocidad; el ***South Bridge*** o **controlador de I/O** se conecta con el resto de dispositivos.



Layout **estándar de puentes dentro de la placa madre de un computador.**



Tarjeta madre de un computador Intel.

Comunicación I/O - Conexión de buses

Implementación real: Intel

- Tarjeta madre posee su CPU, memoria y resto de dispositivos conectados con un *North Bridge* y un *South Bridge*.

- Las tasas de transferencia de datos del *North Bridge* son más altas que las del *South Bridge* en una cantidad significativa.

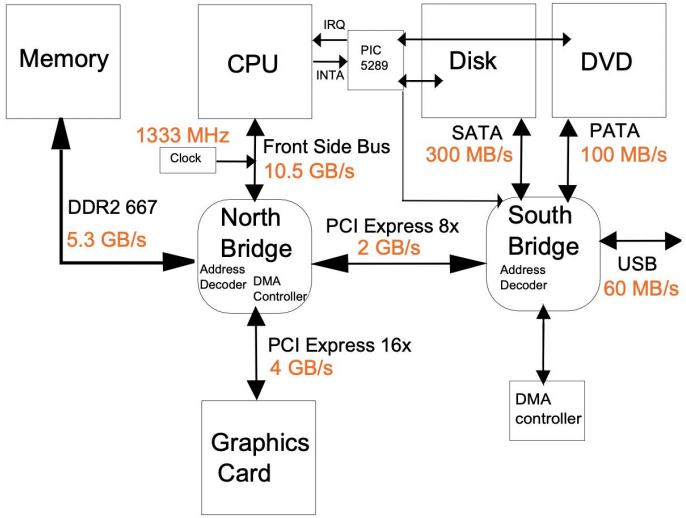


Diagrama de un computador con chipset Intel 5000P.

Ejercicios

Ahora, veremos algunos ejercicios.

Estos se basan en preguntas de tareas y pruebas de semestres anteriores, por lo que nos servirán de preparación para las evaluaciones.

Ejercicios

Indique, justificadamente, qué señales y componentes son esenciales en la comunicación entre los dispositivos I/O y la CPU para que esta pueda atender interrupciones ejecutando ISRs.

***Examen*, 2023-1**

Ejercicios

Suponga que tiene una impresora cuyos registros de estado y comando de 32 bits están mapeados en memoria desde la dirección 0x10150000. A continuación, se indican las tablas de direcciones, comandos y estado:

Offset	Nombre	Descripción
0x00	`printer_stat`	Registro de estado.
0x04	`printer_comm`	Registro de comando.
0x08	`printer_clr`	Registro de color.
0x0C	`printer_sz`	Registro de tamaño hoja.
0x10	`printer_or`	Registro de orientación hoja.

Nombre	Descripción	Valor
`printer_stat`	Apagada.	0
`printer_stat`	Prendida.	1
`printer_stat`	Prendiendo.	127
`printer_stat`	Imprimiendo.	255
`printer_comm`	Prender.	0
`printer_comm`	Apagar.	1
`printer_comm`	Imprimir.	2
`printer_clr`	Blanco y negro.	0
`printer_clr`	Color.	1
`printer_sz`	Tamaño carta.	0
`printer_sz`	Tamaño oficio.	1
`printer_or`	Horizontal.	0
`printer_or`	Vertical.	1

***Examen*, 2023-1**

Ver siguiente diapositiva.

Ejercicios

Suponga que bajo este contexto se ejecuta el siguiente programa en RISC-V:

```
.text
  main:
  li t0, 0x10150000
  lw t1, 0(t0)
  li s0, 1
  li s1, 127
  li s2, 0
  beq t1, s0, actionThree
  beq t1, s1, actionTwo
  beq t1, s2, actionOne
  j end
  actionOne:
    li t2, 0
    sw t2, 4(t0)
  actionTwo:
    lw t1, 0(t0)
    beq t1, s0, actionThree
    j actionTwo
  actionThree:
    li s4, 1
    sw s4, 8(t0)
    li s5, 0
    sw s5, 12(t0)
    li s6, 1
    sw s6, 16(t0)
    li t2, 2
    sw t2, 4(t0)
  end:
```

***Examen*, 2023-1**

Ver siguiente diapositiva.

Ejercicios

Respecto a este programa:

- Indique justificadamente qué tipo de comunicación se presenta entre la CPU y la impresora en el código anterior.

- Explique en términos prácticos, según el programa y las tablas adjuntas, lo que realizan las acciones `actionOne`, `actionTwo` y `actionThree`.

***Examen*, 2023-1**

Antes de terminar

¿Dudas?

¿Consultas?

¿Inquietudes?

¿Comentarios?

DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 9 - Comunicación de CPU y Memoria con I/O

Profesor: Germán Leandro Contreras Sagredo

Anexo - Resolución de ejercicios

¡Importante!

Estos ejercicios pueden tener más de un desarrollo correcto. Las respuestas a continuación no son más que soluciones que **no excluyen** otras alternativas igual de correctas.

Ejercicios - Respuesta

Indique, justificadamente, qué señales y componentes son esenciales en la comunicación entre los dispositivos I/O y la CPU para que esta pueda atender interrupciones ejecutando ISRs.

El componente esencial para la ejecución de interrupciones es el controlador de interrupciones que, a partir de la solicitud de los dispositivos I/O, le envía la señal IRQ (*Interrupt Request*) a la CPU. Esta, en caso de poder atender interrupciones, le envía de vuelta al controlador la señal INTA (*Interrupt Acknowledge*) para que le envíe el ID del dispositivo de mayor prioridad y acceda a la dirección de su ISR dentro del vector de interrupciones.

Ejercicios - Respuesta

Respecto a este programa:

- Indique justificadamente qué tipo de comunicación se presenta entre la CPU y la impresora en el código anterior.

La CPU se comunica con la impresora a través de *polling*, ya que esta consulta su estado consistentemente hasta que cambia, particularmente desde que está en estado “prendiendo” hasta que pasa a “prendida”.

Ejercicios - Respuesta

Respecto a este programa:

- Explique en términos prácticos, según el programa y las tablas adjuntas, lo que realizan las acciones `actionOne`, `actionTwo` y `actionThree`.

En el segmento `actionOne` la CPU le da el comando a la impresora para que se prenda; en el segmento `actionTwo` la CPU espera a que la impresora termine de prender (paso de estado “prendiendo” a “prendida”); y en el segmento `actionThree` la CPU se encarga de configurar la impresora para que imprima una hoja a color de tamaño carta con orientación vertical.