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

**IIC2343 – Arquitectura de Computadores**

# Guía 2 – I/O – Solución propuesta

**Profesor: Yadran Francisco Eterovic Solano**

**Ayudantes: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)**

**Jurgen Dieter Heysen Palacios (jdheysen@uc.cl)**

## Nota al lector

El título dice 'solución propuesta' por una razón bien sencilla: Estos ejercicios pueden tener más de un desarrollo correcto. Lo que se pretende hacer aquí es mostrar un camino a la solución, sin excluir la posibilidad de rutas alternativas igual de correctas.

## Preguntas

1. a. **(I2 - II/2015)** Explique cómo funciona la transferencia de direcciones y datos desde/hacia los dispositivos mapeados a memoria.

   La pieza clave para el mecanismo de mapeo a memoria corresponde al ***address decoder***. La idea es que tanto la memoria como los dispositivos I/O ocupan el mismo bus de direcciones y el address decoder determina si la dirección corresponde a la memoria o a alguno de los dispositivos I/O. Si el caso es el último, entonces se accede al dispositivo correspondiente, desde el que se recibe un dato (o es enviado) desde el bus de direcciones.

   b. Describa las instrucciones de la ISA de un computador x86 que permiten acceder a dispositivos mediante port I/O.

   Las instrucciones principales son dos:

   - `IN Reg,Port`: Se copia en el registro `Reg` el dato enviado por el dispositivo I/O asociado a la dirección `Port`.
   - `OUT Port,Reg`: Se escribe en el dispositivo I/O asociado a la dirección `Port` el contenido del registro `Reg`.

   En este caso, las direcciones `Port` son independientes de las direcciones de la memoria y su valor se encuentra entre 0 y 65535 (0xFFFF) para direcciones de 16 bits.

c. **(I2 - I/2017)** ¿Con qué tipo de dispositivo de I/O es preferible utilizar mapeo de memoria por sobre puertos?

Es preferible utilizarlo con dispositivos que utilicen pocas direcciones de memoria. Si utilizan menos, el número de direcciones a ocupar será menor dentro del espacio disponible en el bus de direccionamiento (evitando llegar a una eventual *memory barrier*). Por ejemplo, no convendría en absoluto mapear a memoria una tarjeta de video (¡imagina la cantidad de direcciones que habrían tomado los pixeles!).

d. ¿Por qué es mejor hacer uso de interrupciones en vez de *polling*? Mencione un ejemplo.

Las interrupciones permiten que la interacción entre la CPU y los dispositivos I/O se realice solo cuando uno de estos dos lo requiera. En *polling*, se revisa constantemente si algún dispositivo I/O requiere enviar datos (o que le envíen), hasta que alguno lo solicita y se realiza la solicitud. La revisión constante limita la capacidad de la CPU de realizar otras tareas, haciendo que el funcionamiento general sea ineficiente.

Un ejemplo sería el uso del teclado y el *mouse*. Si el computador estuviera revisando constantemente si el usuario escribió algo o no, o si movió el cursor, la eficiencia del computador sería deplorable (más aún, si revisara más dispositivos, como el lector de discos, ¡sería prácticamente imposible que corriera!). En cambio, si espera la interrupción por teclado o por cursor, no deja de realizar su conjunto de tareas mientras no se escriba ni se mueva el *mouse*.

e. **(I2 - I/2016)** ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido?

El vector de interrupciones alberga las direcciones de las ISR de los dispositivos I/O que se encuentran registrados en el sistema. Entonces, al procesar una IRQ, el vector otorga la dirección de la ISR del dispositivo que hizo la interrupción, lo que permite finalmente ejecutarla.

f. **(I2 - II/2016)** Luego de recibir la señal INTA, ¿qué tarea(s) debe realizar un controlador de interrupciones?

El controlador de interrupciones, al recibir la INTA, busca en sus registros internos el dispositivo que produjo la IRQ. Luego, el ID de esta es enviada a la CPU a través del bus de datos. Finalmente, la CPU utiliza este ID para buscar la dirección de la ISR a ejecutar en el vector de interrupciones.

g. **(I2 - I/2017)** ¿Cómo modificaría el mecanismo de comunicación del controlador de DMA a la CPU, para poder medir el progreso de una copia de memoria?

El mecanismo será descrito asumiendo que se tienen a disposición el vector de interrupciones de una arquitectura x86 tradicional.

Una posibilidad es la siguiente:

- Reservar tres direcciones en memoria para usos específicos. Estos serán:
  1) Almacenar el número de palabras total que se van a transferir.
  2) Almacenar el número de palabras total que queda por enviar.
  3) Almacenar el porcentaje de envío efectuado.
- Añadir en la configuración del DMA la escritura del valor registrado en el contador de palabras sobre la memoria del computador. De forma preliminar se escribirá en la dirección que almacena el total a transferir (primer dirección reservada). No obstante, en los siguientes envíos escribirá en la dirección de palabras restantes (segunda dirección reservada).
- Mantener otro DMA auxiliar que tendrá dos funcionalidades principales:
  1) Transferir la cantidad de palabras total y las que quedan por enviar al I/O correspondiente al coprocesador matemático. Este deberá ejecutar una operación que retorne el valor $1 - \frac{P_r}{P_t}$, siendo $P_r$ la cantidad de palabras restante por enviar y $P_t$ el total.
  2) Transferir dicho resultado a la dirección de memoria reservada para dicho propósito al comienzo (tercera dirección reservada).
  3) Ahora, generar una transferencia de ese valor de memoria al I/O encargado de las funcionalidades de video. Este, a partir del valor recibido, ejecutará una ISR que cambiará lo desplegado en la ventana con una barra que represente el porcentaje recibido.

Notar que esta implementación es bastante abierta, pueden existir más formas para lograr el objetivo.

h. Detalle, paso a paso, cómo se manejaría una interrupción realizada por un dispositivo (llamémoslo $IO_i$) que se encuentra conectado a otro dispositivo (llamémoslo $IO_j$), donde la ISR de este último se encuentra almacenada en el vector de interrupciones del computador.

Enumeramos el paso a paso lo más detallado posible.

1) Asumimos que $IO_j$ genera una IRQ a partir de la que realiza $IO_i$.
2) La PIC revisa su registro IMR para ver si la interrupción no está enmascarada. En este caso, marca un 1 en el bit correspondiente del *Interrupt Request Register*.
3) PIC escoge la interrupción de mayor prioridad (menor IRQ). En este caso, asumimos que es la enviada por $IO_i$. Se 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* (*flags*).
6) CPU revisa si el *flag* de las interrupciones está activo (`IF = 1`), en cuyo caso la atiende (asumimos que lo hace).

7) CPU deshabilita la atención de más interrupciones (`IF = 0`).
8) CPU envía INTA para saber quién interrumpió.
9) PIC revisa el *In-Service Register* para saber el ID del IRQ que está siendo atendido (en este caso, el de $IO_j$) y lo envía mediante el bus de datos.
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 (`CALL Mem[id]`).
12) ISR respalda el estado actual de la CPU.
13) ISR ejecuta su código. Esta es la parte clave: Definimos la ISR como la búsqueda en el registro de estado del controlador de $IO_j$ de la dirección de la ISR asociada a $IO_i$ (que puede ser una subrutina, por ejemplo) y la llama.
14) Terminada la subrutina, el ISR original envía un comando EOI a la PIC, con la que se cambia a 0 el bit asociado en el *In-Service Register*, lo que indica que se terminó de atender 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 los *conditions codes* (*flags*) desde el *stack*.

i. Explique la diferencia entre las interrupciones realizadas por *hardware* y *software*, dando un ejemplo de cada una.

Las interrupciones por *hardware* pueden ser de dos tipos:

- **Gatilladas por dispositivos I/O:** Son las que realizan estos dispositivos para actualizar su estado o el de la CPU, generando una interrupción que ejecuta una ISR específica. Un ejemplo sería la actualización del cursor en el monitor de un computador a partir de la posición del *mouse*.
- **Excepciones:** Son las que se gatillan al encontrarse errores en la programación, generando una interrupción que ejecuta una ISR especializada (*exception handlers*). Un ejemplo sería dividir por 0 en una instrucción.

A diferencia de estas, las interrupciones por *software* son las que generan los mismos programas ejecutados, generalmente para ejecutar una ISR específica. La principal diferencia es que este no deshabilita la atención a otras interrupciones en *hardware*, por lo que hay que modificar la IF directamente mediante instrucciones: `CLI` (deshabilitar las interrupciones) y `STI` (habilitar las interrupciones). Un ejemplo concreto de esto es, por ejemplo, el manejo gráfico en una consola (tomaremos la Super Nintendo). El programa del juego *Super Mario World* genera una interrupción por *software* para modificar la tarjeta de video y desplegar una nueva imagen (por ejemplo, el movimiento de Mario y de un par de Goombas). Entonces, se ejecuta `INT dir` (siendo `dir` la dirección de memoria correspondiente a la tarjeta en el vector de interrupciones) y la ISR ejecutada se encarga de actualizar el contenido gráfico. Notar que aquí se trabaja directamente con la memoria de la CPU, por lo que convendría, en este caso, utilizar un controlador DMA para actualizar el estado (y así la CPU se encarga de ejecutar otras tareas).

2. **(I2 - II/2016)** Para los siguientes ejercicios, considere la siguiente tabla, que presenta el vector de interrupciones completo de un computador con ISA x86 de 16 bits. El vector de interrupciones se encuentra almacenado a partir de la dirección de memoria `0x0000`:

| IRQ | Dispositivo | Pos. en vector |
|---|---|---|
| IRQ0 | Timer del sistema | 00 |
| IRQ1 | Disco Duro | 01 |
| IRQ2 | Interfaz USB | 02 |
| IRQ3 | Interrupción software | 03 |

   a. ¿Cuantos dispositivos que generen solicitudes de interrupción pueden conectarse?

      Es importante notar que uno de los dispositivos disponibles es una interfaz USB. Como se puede conectar un número arbitrario de dispositivos en ella, la respuesta es un número infinito.

   b. Dos dispositivos, teclado y *mouse*, están conectados a la interfaz USB. Describa un mecanismo para ejecutar la ISR correspondiente al *mouse*, cuando este genera una interrupción.

      Un mecanismo posible es el siguiente:

      - Se llama a la ISR de la controladora USB.
      - Esta ISR en particular puede tener como función leer un registro de estado específico de la interfaz para determinar la ISR real que se busca ejecutar (en este caso, la del *mouse*).
      - Se invoca dicha ISR. Un método puede ser, por ejemplo, implementarla como subrutina. También podría generar otra interrupción por *software* (trap).

   c. ¿Que ocurriría en este computador si se ejecuta la instrucción `MOV [0],AX`?

      Esto generaría un cambio en el puntero que apunta a la ISR del *timer* del sistema. Esto es **fatal**, ya que es utilizado para generar interrupciones que controlan la ejecución de los procesos dentro del computador y permiten realizar cambios de contexto.

   d. Proponga un esquema para permitir el acceso (lectura y escritura) controlado y centralizado al vector de interrupciones por parte de los programas, *i.e.*, el acceso sólo puede realizarse a través de una interfaz entregada por el sistema operativo (o la BIOS).

      **Hint**: El esquema puede incluir cambios a la arquitectura del computador.

      Una opción, sería traspasar completamente el vector de interrupciones a un registro especializado (llamémoslo I), que solo puede ser escrito en modo supervisor/*kernel* (*i.e.* por el sistema operativo) y que es accedido/modificado por una instrucción especializada: `SINT id,ISR`, siendo `id` el ID del vector e `ISR` la dirección nueva de un ISR a almacenar. Se seguiría usando INT en la ISA, pero su ejecución debe ser modificada para ir acorde a lo especificado recientemente.

      **Nota:** Recuerde que en la ISA x86 existe la instrucción `INT dir`, la que corresponde a una interrupción por *software*, siendo `dir` una dirección dentro del vector de interrupciones.