

Índice
Conocimientos y materiales necesarios
1. Programación en ensamblador de rutinas de servicio
2. Breve explicación del dispositivo Luces
3. Una rutina de servicio que hace algo
4. Ejercicios adicionales

# Interrupciones: Programación de rutinas de servicio

Área de Arquitectura y Tecnología de Computadores – Versión 1.0.422, 23/03/2020

## Objetivos

Una vez se ha asentado claramente en la sesión anterior el mecanismo por el cual la CPU puede saltar a otra zona de la memoria a petición de un periférico, y retornar más tarde al programa abandonado a través de la instrucción `iret`, en esta sesión se tratará ya de la programación e instalación de rutinas de servicio escritas en lenguaje ensamblador.

Para ello, de nuevo se utilizará el periférico `Luces` para generar interrupciones, pero en esta ocasión la rutina de servicio será algo más que un simple `iret`. Para que la rutina haga algo más “visible” se utilizará también el periférico `Pantalla`.

## Conocimientos y materiales necesarios

Para poder realizar esta sesión el alumno debe:

- Saber escribir programas en el lenguaje ensamblador de la CPU elemental y utilizar el programa ensamblador para obtener archivos `.eje`.
- Comprender claramente el mecanismo de interrupción y la forma de instalar `a mano` una rutina, tal como se hizo en la sesión anterior.
- Conocer la programación del dispositivo de Salida `Pantalla`.
- Tener el fichero `5-2teclado.sim` construido durante la sesión 2 de este bloque de prácticas.

Durante la sesión se plantearán una serie de preguntas que deben responderse en el correspondiente [cuestionario](#) del Campus Virtual. El cuestionario se puede abrir en otra pestaña del navegador pinchando en el enlace mientras se mantiene pulsada la tecla `Ctrl`.

## 1. Programación en ensamblador de rutinas de servicio

En la sesión anterior hemos codificado e instalado “a mano” una rutina de servicio mínima para el periférico `Luces` que tenía asignado el número de vector 3. Esta rutina contenía tan sólo una instrucción `iret`. La instalación consistió en copiar en la memoria el código de la instrucción, y escribir en la tabla de vectores, en la posición 3, la dirección en la cual habíamos colocado la rutina.

Ahora veremos cómo hacer esto mismo, pero desde un programa escrito en ensamblador.

La rutina de servicio se escribe como un procedimiento más, con la diferencia de que ha de retornar usando la instrucción `iret` en lugar de `ret`. Por tanto, nuestra rutina mínima podría ser así:

```
PROCEDIMIENTO rutina_minima
IRET
FINP
```

Este procedimiento formará parte de un fichero `.ens`, en el cual estará también el `programa principal`, y también el código que se ocupa de `instalar la rutina`. La `instalación` de la rutina consistirá simplemente en escribir en la dirección de memoria 3 (puesto que se trata del vector 3) el número que indica dónde está colocada la instrucción `iret`. Es decir, si la instrucción `iret` estuviera colocada en `50C0h` como era el caso de la sesión anterior, el código necesario para `instalarla` sería algo así:

```
MOVL R0, 3 ; Número del vector a modificar
MOVH R0, 0
MOVL R1, 0C0h ; Parte baja de la dirección donde está la rutina
MOVH R1, 50h ; Parte alta de la dirección donde está la rutina
XXXXXX ; Instrucción que pone 50C0h en la dirección 0003
```

¿Qué instrucción iría en lugar de `XXXXXX`? Responde en el [cuestionario](#): pregunta 1. Esta sería la instrucción que modifica el vector 3, de la misma forma que nosotros habíamos hecho `a mano` en la sesión anterior usando el editor hexadecimal de la memoria.

Ahora bien, hay un problema. Y es que en la sesión anterior sabíamos que la rutina estaba en la dirección `50C0h` porque nosotros mismos habíamos colocado allí el código de `iret`, pero en esta ocasión la `rutina_minima` forma parte de nuestro programa y no sabemos a priori a qué dirección de memoria irá a parar ese `iret`. Por tanto no podemos cargar directamente el valor `50C0h` en `r1` como hemos hecho en el listado anterior. En vez de ello, usaremos la directiva `DIRECCION` del compilador, para que él mismo averigüe dónde está colocada la rutina.


El código sería este. Complétalo:

```
ORIGEN 300h
.PILA 20h
.CODIGO
MOVL R0, 3 ; Vector a modificar
MOVH R0, 0
MOVL R1, BYTEBAJO DIRECCION rutina_minima
MOVH R1, XXXXXX
XXXXXX
STI ; Permitir interrupciones

; Una vez instalada la rutina, el programa
; principal se encierra en un bucle infinito
PorSiempre:
JMP PorSiempre

; Y esta sería la rutina mínima:
PROCEDIMIENTO rutina_minima
XXXXXX
FINP

FIN
```

- 
- Escribe el programa anterior (una vez completado) en el fichero `5-4int1.ens` y ensámbalo para obtener `5-4int1.eje`.
  - Abre el simulador de la CPU y carga el archivo `5-2teclado.sim`. De este modo recuperarás la configuración en la cual nuestro ordenador tenía conectados los periféricos de pantalla y teclado.
  - Conecta a este ordenador el dispositivo `Luces`, tal y como hiciste en la sesión anterior, es decir: como nombre para el dispositivo `Luces`, como dirección base `E000h`, como vector el 3 y como prioridad 1. Marca la casilla `Interrupciones` para indicar que este periférico tiene capacidad de generar interrupciones.
  - Carga en el simulador el programa que has obtenido antes `5-4int1.eje`.
  - Vamos a ejecutarlo instrucción por instrucción. Pulsa `F8` cuatro veces y los registros `r0` y `r1` serán inicializados mediante las instrucciones `movh` y `movl`. ¿Qué valor aparece en `r1`? Responde en el [cuestionario](#): pregunta 2.
  - Observa que la respuesta anterior ha de ser la dirección de memoria donde se halla almacenada nuestra `rutina_minima`. Vamos a comprobarlo mediante el desensamblador de la memoria. Ve a esa dirección y comprueba qué instrucción se encuentra allí. Se trata de la primera instrucción de nuestra rutina. (Si no es así, es que no has inicializado correctamente `r1` mediante la instrucción `movh` que tenías que completar en el listado.)
  - En realidad, podíamos haber calculado “a mano” la dirección donde se halla esa instrucción, pues sabemos en qué dirección está la primera instrucción de nuestro programa (pues lo hemos especificado con la directiva `ORIGEN`) y podemos también contar cuántas instrucciones hay hasta llegar a `iret`. Haz la cuenta y comprueba que te sale lo mismo (recuerda que las directivas y comentarios no cuentan como instrucciones).
  - Pulsa `F8` de nuevo. Esto debe modificar el vector 3 si has colocado la instrucción correcta. Compruébalo con el editor hexadecimal de la memoria (la dirección 3 debe almacenar la dirección de nuestra `rutina_minima`).
  - Pulsando `F8` otra vez, se ejecuta `sti`. Las interrupciones están permitidas.
  - Pulsa `F9`. Esto causa que a partir de ahora el simulador ejecute instrucciones sin parar, como si pulsaras repetidamente `F8`. Observa el valor de `IR`. Puesto que el programa principal no es más que un bucle infinito, `IR` contiene la misma instrucción `jmp -1` una y otra vez.
  - Si generásemos una interrupción con el dispositivo `Luces`, por un breve instante, `IR` dejaría de mostrar la instrucción `jmp -1` y mostraría otra; ¿cuál sería? Responde en el [cuestionario](#): pregunta 3. Comprueba tu respuesta generando una interrupción con el dispositivo `Luces`.
  - Cambia alguno de los interruptores del dispositivo `Luces`. Como verás, las bombillas no se iluminan. No hay una relación directa entre el estado de las bombillas y el estado de los interruptores. Para encender y apagar las luces, la CPU debería enviar datos al dispositivo.

Con el ejemplo anterior hemos logrado un programa principal que se halla enfascado en una tarea (en este caso la inútil tarea de repetir la misma instrucción una y otra vez), pero a la vez permitimos que la CPU ejecute otras tareas diferentes cuando recibe interrupciones. Ha llegado el momento de hacer algo más útil.

## 2. Breve explicación del dispositivo Luces

Como hemos visto, este dispositivo tiene 16 bombillas y 16 interruptores, si bien el manipular los interruptores parece que no afecta a las bombillas. ¿Cómo funciona y para qué sirve este dispositivo?

Internamente el dispositivo tiene dos registros, que vamos a llamar `registro de luces` y `registro de interruptores`, ambos de 16 bits.


- Registro de Luces.** Este registro puede ser modificado por la CPU, que puede escribir en él. Cuando la CPU pone un dato de 16 bits en este registro, inmediatamente las 16 bombillas cambiarán de estado, encendiéndose aquellas para las cuales el bit sea 1. Por ejemplo, si el dato es `0003h`, se encenderían las dos bombillas del extremo derecho. Este registro no puede ser leído por la CPU. Es decir, la CPU puede cambiar el estado de las luces, pero no puede saber qué luces hay encendidas en un momento dado <sup>[1]</sup>.
- Registro de Interruptores.** Este registro refleja el estado de los interruptores. Cada vez que manipulas un interruptor, estás modificando un bit de este registro. Si por ejemplo levantas los dos interruptores de la izquierda, dejando todos los demás bajados, el registro tomaría el valor `C000h` (los dos bits más altos a 1, los demás a 0). Este registro puede ser leído por la CPU, pero no puede ser escrito. La única forma de cambiar el valor del registro es manipulando los interruptores.

Así pues, desde el punto de vista del dispositivo luces, tenemos un registro que sólo puede ser escrito (el de luces) y otro que sólo puede ser leído (el de interruptores). Para economizar direcciones ambos registros se mapean en la misma dirección de memoria (en nuestro caso se tratará de la dirección `E000h`, que es la que hemos elegido como dirección base al conectar este dispositivo en la sesión anterior). Cuando se intenta escribir en esa dirección, el dispositivo modificará el registro de luces. Cuando se intente leer de esa misma dirección, el dispositivo responderá con el valor del registro de interruptores.

Por tanto, desde el punto de vista de la CPU, tenemos un solo registro de datos que actúa de forma diferente según se escriba o se lea. Al escribir en esa dirección, encendemos y apagamos las luces. Al leer **de esa misma dirección** lo que obtendremos será el estado de los interruptores.


## 3. Una rutina de servicio que hace algo

Nuestra rutina de servicio contenía un simple `iret`. Vamos a escribir una rutina más útil, que actualice el estado de las luces según el estado de los interruptores.

- 
- Haz una copia de `5-4int1.ens` con el nombre `5-4int2.ens` y editalo.
  - Ve al punto del programa donde estaba el código de la `rutina_minima` y cambia su nombre por `rutina_util`.
  - Escribe en el interior de esta rutina las instrucciones necesarias para leer del registro de datos del dispositivo `Luces`, dejando el resultado en `r1`. Utiliza `r0` para mantener la dirección a través de la cual accedes al registro de datos del dispositivo (esta dirección es la que has especificado al conectar el periférico, es decir `E000h`). Posteriormente, escribe el valor de `r1` de nuevo en el registro de datos del dispositivo (esto hará que las luces del dispositivo se enciendan donde `r1` tenga bits a 1).
  - La rutina que has escrito ha modificado los registros `r0` y `r1`. Esto no puede dejarse así, pues tal vez el programa principal, cuando fue interrumpido, estaba usando estos registros para otra cosa y se los encontraría de pronto con nuevos valores. La rutina de servicio debe comenzar apilando todos los registros que va a modificar, y desapilándolos de nuevo justo antes del `iret`. Modifica la rutina para que haga esto.
  - Modifica también la parte que instala la rutina, puesto que ahora se llama `rutina_util` y no `rutina_minima`.
  - Sal del editor y ensambla el nuevo programa para obtener `5-4int2.eje`.
  - Carga en el simulador este programa y pulsa `F9`. Como verás, el programa entra rápidamente en el bucle infinito en el que ejecuta `jmp -1` una y otra vez.
  - Mueve alguno de los interruptores del dispositivo `Luces`. Como ves, las bombillas permanecen apagadas.
  - Genera una interrupción en el dispositivo luces. Si has programado bien la rutina, ésta leerá el estado de los interruptores y lo escribirá sobre las bombillas. Las bombillas deben encenderse ahora reflejando el patrón que has puesto en los interruptores.

## 4. Ejercicios adicionales

Estos ejercicios adicionales permiten al alumno reforzar los conocimientos adquiridos durante la sesión práctica.

- 
- Copia `5-4int2.ens` con el nombre `5-4int3.ens`, y realiza en él las modificaciones necesarias para que la rutina de servicio, tras leer el estado de los interruptores sobre el registro `r1`, copie el valor de este registro a la primera posición de la memoria de vídeo. Recuerda almacenar y recuperar todos los registros que uses para esto.
  - Comprueba el funcionamiento del programa anterior. Cuando coloques ciertas combinaciones de interruptores y pulses **Generar interrupción**, en la esquina del periférico `Pantalla` deberá aparecer un carácter. Prueba por ejemplo con la combinación de interruptores `00000100`, `01000001`. Cuando este número sea leído de los interruptores y escrito en la memoria pantalla, la interfaz de pantalla interpretará los primeros 8 bits como un color (rojo sobre fondo negro) y los 8 bits bajos como un código ASCII (en este caso es el ASCII de la letra **A**). ¿Aparece una **A** roja en la pantalla?
  - La siguiente propuesta consistirá en modificar el programa principal, en la parte que contiene la instrucción `jmp PorSiempre`, para que en lugar de ser un tonto bucle infinito, sea un bucle infinito un poco más interesante. El nuevo **programa principal**:
    - Comienza cargando en `r0` la dirección de inicio de la memoria de vídeo, y en `r1` el dato `0740h`.
    - Carga en `r2` la dirección de donde terminaría la memoria de pantalla. (Esto será igual a la dirección en la que comienza, más 78h, que es 120 decimal)
    - Haz un bucle infinito en el que, cada vez que repite ocurre lo siguiente:
      - Se copia el contenido de `r1` a la dirección de memoria apuntada por `r0`.
      - Se incrementa `r0`.
      - Se comprueba si `r0` es igual a `r2`. Si son iguales es que `r0` ya ha alcanzado el final de la memoria de pantalla, y en este caso debe cargarse `r0` con el valor inicial de nuevo.
    - Carga y carga este programa. Al darle a `F9` deberán ir apareciendo signos @ de color blanco en el dispositivo `Pantalla`. Y al generar una interrupción con el dispositivo `Luces` el primero de los signos @ será sustituido por otra letra, según la combinación de los interruptores en ese momento.
  - Si durante la ejecución del programa anterior generásemos una interrupción desde el teclado, ¿qué crees que pasaría? ¿Y si después intentamos generar más interrupciones desde el dispositivo `Luces`? Haz el experimento e intenta encontrar una explicación a lo que ocurre. ¡Es una pregunta muy difícil! Pregúntale al profesor si tienes dudas.

1. Sin embargo, ya que la CPU es la única que puede encender o apagar las luces, el programador puede dar por cierto que las luces estarán tal y como él las dejó la última vez que escribió en el registro de luces