

Bloque 3

Entrada salida sobre el computador elemental



Prácticas de
Introducción a los computadores
Curso 2006-2007

SESIÓN 1

Interfaces de vídeo y teclado del computador teórico

Objetivos

En esta sesión se pretende que el alumno se familiarice con los conceptos de periférico, interfaz, mapa de direcciones y salida mapeada en memoria. Para esto se desarrollarán ejemplos sobre el interfaz de vídeo y teclado del computador teórico. La entrada desde el teclado se realizará mediante la técnica de muestreo periódico.

Conocimientos y materiales necesarios

Para poder realizar esta sesión el alumno debe:

- Comprender el funcionamiento básico del computador teórico así como de su ensamblador.
- Repasar la teoría relativa a sistemas de memoria y Entrada/Salida mapeada en memoria.
- Repasar la teoría del interfaz de vídeo y de teclado del computador teórico.

Desarrollo de la práctica

1. El mapa de memoria y los periféricos

En esta práctica se van a conectar dos periféricos al computador teórico: una pantalla de 8 filas por 15 columnas, y un teclado con dos registros internos. Para imprimir caracteres en esta pantalla, se deben escribir sobre la memoria de video de su interfaz. Para leer datos del teclado se deberán leer de sus registros de control. Por tanto, antes de poder realizar programa alguno, es necesario *mapear* estos dispositivos en el espacio de direcciones y conectarlos al computador elemental.

Realiza los pasos siguientes. Si tienes dudas en alguno de ellos, puedes seguir los pasos más detallados del Anexo 5.1:

- ❑ Configura la memoria del simulador para que tenga dos módulos: uno de 32K que ocupa las primeras direcciones de memoria, y otro de 16K que vaya a continuación. Los 16K finales del espacio de direcciones deben quedar libres para mapear en ellos los interfaces.
- ❑ Conecta un interfaz de vídeo, poniéndole como nombre Pantalla1 y como dirección base F000h (es la dirección en que se *mapeará*).
- ❑ Conecta un interfaz de teclado, poniéndole como nombre Teclado1, como dirección base F700h, como vector de interrupción el 1, prioridad 0. Deja sin marcar la casilla que indica si se generan interrupciones.
- ❑ Guarda el estado del simulador en el fichero 3-1interfaces.sim

2. Mostrando texto en la pantalla

Para ver cómo se escribe en la pantalla, vamos a hacer un sencillo programa que escriba un asterisco en la esquina superior izquierda de la pantalla del computador teórico. La explicación teórica del funcionamiento del interfaz la tienes en el anexo 5.2, por si necesitas repasarla.

Vamos a hacer el programa que imprime un asterisco en la esquina superior izquierda:

- ❑ Edita un archivo llamado 3-1ast1.ens.
- ❑ El programa debe cargarse en la dirección 500h.
- ❑ El código debe empezar cargando en R0 la dirección donde vamos a escribir el asterisco. Teniendo en cuenta que cuando *instalaste* la pantalla elegiste como dirección base F000h, ¿cuánto tiene que valer R0?^[1] Escribe las instrucciones necesarias para cargar ese valor en R0.
- ❑ A continuación vamos a cargar en R1 el valor que queremos que aparezca en la memoria del interfaz de vídeo. Escribe las instrucciones necesarias para que la parte baja de R1 contenga el código ASCII del asterisco y la parte alta los atributos para que se escriba en blanco sobre fondo negro (quizás necesites consultar el anexo 5.2).
- ❑ Por último, escribe la instrucción necesaria para que el asterisco aparezca en pantalla. Para ello debes usar los dos registros anteriores.
- ❑ Finaliza el programa, guárdalo, sal del editor, ensambla el programa y cuando hayas obtenido el .exe, cárgalo en simulador.
- ❑ Sin perder la pantalla de vista, ejecuta el programa instrucción a instrucción comprobando que el asterisco aparece cuando debe hacerlo. Nota: Para que **F8** funcione, la ventana activa debe ser la del simulador y no la pantalla.

Vamos a ponerle un poco más de color al programa:

- ☐ Edita de nuevo el archivo 3-1ast1.ens. Guárdalo con el nombre 3-1ast2.ens.
- ☐ Modifica el programa para que el asterisco se escriba en rojo sobre verde. ¿Cuánto debe valer ahora el atributo?^[2]
- ☐ Haz que el asterisco se escriba en la segunda celda de la segunda fila. ¿Cuál es la dirección de memoria asociada con esa celda?^[3] (Recuerda que la pantalla es de 8 filas por 15 columnas.)
- ☐ Guarda, ensambla y ejecuta el programa para comprobar que funciona correctamente.

2

3

3. Leyendo datos del teclado, por muestreo periódico

En este apartado vamos a realizar un ejemplo de programación del teclado utilizando la técnica de sincronización del *muestreo periódico*. Para ello debes conocer el funcionamiento del registro de control del teclado, y el concepto de *máscara de bits* (puedes repasarlo en el anexo 5.3)

Queremos hacer un programa con un bucle infinito en el que se estará mirando si hay una tecla pulsada. Cuando se detecte la pulsación de una tecla, se imprimirá un asterisco en la pantalla del computador teórico, y se volverá al bucle de espera. El listado, con algunas instrucciones incompletas, sería el siguiente:

```

1  ORIGIN 300h
2  .CODIGO
3  MOVL R0, ??h      ; Cargar la dirección del registro de control
4  MOVH R0, ??h      ; del teclado en R0
5
6  MOVL R1, ??h      ; Cargar la máscara en R1
7  MOVH R1, ??h
8
9  MOVL R3, ??h      ; Cargar en R3 la dirección de la primera celda
10 MOVH R3, ??h      ; de pantalla
11
12 MOVL R4, '*'      ; Cargar lo necesario para imprimir un asterisco
13 MOVH R4, 7h       ; en color blanco sobre fondo negro
14
15 bucle:
16 MOV R2, [R?]      ; Leer el registro de control a R2
17 ??? R2, R2, R1    ; Realizar operación con la máscara
18 BR?? bucle        ; Si no se pulsó tecla, volver al principio del bucle
19
20 ; Si estamos aquí, se pulsó tecla. Imprimir un asterisco
21 MOV [R3], R4       ; Escribir asterisco
22 INC R3             ; Avanzar a la sgte. posición de pantalla
23 JMP bucle          ; Volver al bucle a ver si se pulsan más teclas
24 FIN

```

- ☐ Crea un archivo llamado 3-1pulsa.ens.
- ☐ Copia el listado anterior completanto las instrucciones como indican los comentarios. No hace falta que copies éstos.
- ☐ Guarda, ensambla y carga el programa en el simulador.
- ☐ Ejecuta el programa paso a paso. Comprueba que todos los registros se cargan con los valores adecuados y ejecuta el bucle dos veces, sin haber pulsado ninguna tecla del teclado del computador teórico, para ver que mientras no se pulse ninguna tecla el programa funciona bien (debes ir dándole a **F8** hasta que se ejecute dos veces la instrucción MOV R2, [R0]).
- ☐ Cuando estés viendo en IR la instrucción MOV R2, [R0] (por lo tanto esta instrucción ya estará ejecutada), pulsa una tecla en el teclado del computador teórico. Fíjate que a la derecha de la ventana que representa el teclado se muestra el contenido del buffer de teclado y debe aparecer la tecla que acabas de pulsar.
- ☐ Cuando pulses **F8** vas a ejecutar la instrucción AND R2, R2, R1. ¿Será el resultado de la operación distinto de cero?^[4] ¿Por qué? Comprueba si has acertado pulsando **F8**.
- ☐ ¿Cuántas veces más deberás pulsar **F8** para que se pinte el asterisco en pantalla?^[5] Compruébalo pulsando **F8** hasta que aparezca el asterisco.
- ☐ Parece que el programa funciona... Pero vamos a probarlo un poco más. Sigue pulsando **F8** hasta que aparezca de nuevo en IR la instrucción MOV R2, [R0]. Esta instrucción acaba de traer el contenido del registro de control del interfaz del teclado al registro R2 de la CPU. Según el valor de ese registro, ¿se ha pulsado alguna tecla?^[6] Entonces, ¿qué va a ocurrir? Compruébalo pulsado 3 veces más **F8** (y comprendiendo lo que pasa). ¡Sólo has pulsado una vez el teclado del computador teórico y tienes dos asteriscos en pantalla! Algo va mal.

4

5

6

El problema es que mientras no se lea el carácter que hay en el registro de datos del teclado, no se pondrá a 0 el bit 8 del registro de control. Esto lo puedes comprobar además porque en la ventana que representa el teclado sigue apareciendo en la primera posición del buffer la tecla que pulsaste. Vamos a arreglar nuestro programa para que sólo ponga un asterisco por cada letra, pero antes, para que compruebes hasta que punto funciona mal el programa, dale a **F9** y observa la pantalla. Cuando te canses de ver aparecer asteriscos que no deberían aparecer, dale de nuevo a **F9** para detener la simulación.

Vamos a arreglar el problema:

- ☐ Edita otra vez el archivo 3-1pulsa.ens.
- ☐ Vamos a añadir las instrucciones necesarias para leer el contenido del registro de datos del teclado. Recuerda que este registro está mapeado en la primera dirección del interfaz de teclado. ¿Cuál es esa dirección?^[7] Añade las instrucciones necesarias para inicializar antes del bucle el registro R6 a ese valor.

7

- ❑ Busca la instrucción donde escribes el asterisco en pantalla. Justo antes de esa instrucción vamos a introducir otra que lea el contenido del registro de datos del teclado. Para ello, debe mover lo que hay en la dirección donde apunta R6 al registro R5.
- ❑ Guarda el programa y ensámblalo. Carga en el simulador el archivo 3-1interfaces.sim para volver a la situación inicial y luego carga el archivo con el programa modificado, es decir, 3-1pulsa.eje.
- ❑ Vete ejecutando instrucciones hasta que hayas ejecutado el bucle una vez y llegues a la instrucción BRZ -3, fijándote en que el programa sigue funcionando mientras no se pulse ninguna tecla del computador teórico.
- ❑ Pulsa una tecla en el teclado del computador teórico.
- ❑ Ejecuta con [F8] hasta que llegues otra vez al BRZ -3. La siguiente instrucción que ejecutes debe ser la nueva que has añadido. Antes de pulsar [F8], fíjate en la ventana del teclado en que de momento sigue en el buffer la tecla que has pulsado.
- ❑ Ejecuta la instrucción añadida con [F8]. ¿Ha desaparecido la tecla del buffer?^[8]
- ❑ Vete ejecutando con [F8] hasta que aparezca la instrucción de lectura del registro de control, es decir, MOV R2, [R0]. Mira R2. Según el registro de control, ¿hay alguna tecla pulsada?^[9]
- ❑ Pulsa [F9] para pasar al modo de ejecución continua (equivale a pulsar repetidamente [F8]) y comprueba pulsando teclas del teclado del computador teórico que el programa funciona ahora correctamente, es decir, sólo escribe un asterisco por cada tecla pulsada.

8

9

4. Autoevaluación

4.1. Archivos en el disco

En tu disco de prácticas deberás tener los archivos de programa 3-1interfaces.sim, 3-1ast1.ens, 3-1ast2.ens y 3-1pulsa.ens.

4.2. Ejercicios

- ⇒ Carga el archivo 3-1interfaces.sim y vete a la opción de configurar memoria. Rellena con chips de memoria todo el espacio posible hasta llegar a la dirección de la pantalla pero sin sobreescribirla. ¿Cuántos chips has usado?^[10] ¿De qué tamaño?^[11] Carga el archivo 3-2ast2.ens y comprueba que sigue funcionando con la nueva configuración.

10

11

- ⇒ Modifica el programa `3-1pulsa.ens` para que, en lugar de mostrar por pantalla un asterisco, muestre la letra que se ha pulsado en el teclado. Fíjate que al leer del registro de datos del teclado ya obtendrás en la parte baja el código ASCII de la letra en cuestión, por lo que bastará con que cambies su parte alta (donde está el código *scan*) para poner ahí los colores con que quieres que aparezca en pantalla, y enviarla después a la memoria de vídeo.
- ⇒ Haz un programa que borre toda la pantalla. Para ello deberás escribir en el bit 0 del registro de control un 1. Recuerda que el registro de control está mapeado después de la memoria de vídeo. Para poner un 1 en el registro de control sin modificar el resto de bits, debes leerlo primero y luego utilizar una máscara para cambiarlo. Cuando hayas hecho y ensamblado el programa, carga el archivo `3-1interfaces.sim`, y carga y ejecuta el archivo `3-1ast2.ens` para escribir algo en pantalla. Carga después el programa para borrar la pantalla y ejecútalo para comprobar que funciona.

4.3. Ampliaciones

Estos son ejercicios opcionales, cuya realización te llevará más tiempo, pero de los cuales puedes aprender mucho, ya que ponen en juego conceptos aprendidos a lo largo de varias sesiones prácticas.

- ⇒ Basándote en el programa `3-1ast.ens`, escribe un otro que, en lugar de un asterisco, escriba un pequeño mensaje de texto en la pantalla. Piensa primero cómo lo harías, después puedes mirar las instrucciones detalladas del anexo [5.4](#)
- ⇒ Transforma el código del programa anterior en un procedimiento que permita escribir una cadena acabada en cero en cualquier posición de la pantalla. Debe recibir como parámetros la dirección de la cadena y la dirección donde debe comenzar a escribirla. Define dos cadenas en la sección de datos y haz que el programa principal imprima las dos cadenas en dos lugares distintos de la pantalla mediante dos llamadas al procedimiento.
- ⇒ Haz un salvapantallas para nuestro computador teórico. Debe consistir en un programa que esté continuamente escribiendo caracteres cambiantes en la pantalla. Para ello, haz un bucle que vaya escribiendo un carácter e incrementando tanto el carácter a escribir como la posición. Comprueba después de cada incremento que la posición no se salga de la pantalla (no sea superior a la última dirección del interfaz); cuando ocurra eso, haz que el puntero que apunta a la posición a escribir vuelva a apuntar a la primera celda de la pantalla. Ejecuta el programa con `[F9]` para comprobar que funciona.
- ⇒ Carga el archivo `3-1interfaces.sim`. Añade otra pantalla al computador teórico con dirección base `D700h`. Modifica el salvapantallas para que vaya escribiendo en las dos pantallas a la vez.

5. Anexos

5.1. Mapa de memoria e interfaces

Queremos *mapear* una pantalla y un teclado. ¿Qué dirección escoger? Evidentemente, tiene que ser una dirección que pueda generar la CPU teórica, es decir, una dirección que esté en su *espacio de direcciones*. Como el bus de direcciones de la CPU teórica tiene 16 líneas, ¿cuántas direcciones distintas podrá generar?^[12] Si la posición más baja es la 0000h, ¿cuál será la más alta?^[13] La pantalla necesitará 120 (8x15) de estas direcciones para mapear su memoria de vídeo. Además tiene un registro de control que también deberá ser mapeado en el espacio de direcciones. Por lo tanto, el interfaz de vídeo necesitará en total 121 direcciones. Recordarás de teoría que se debe utilizar un número de direcciones que sea potencia de 2. Por lo tanto, el interfaz de la pantalla necesitará 128 posiciones de memoria.

12

13

En todas las prácticas anteriores hemos supuesto que podíamos escribir en cualquiera de las direcciones de memoria un dato o una instrucción. Esto era así porque todo el espacio de direcciones del computador teórico estaba lleno de chips de memoria. Pero si tenemos que reservar algunas direcciones para la pantalla, tendremos que eliminar algún chip de memoria. Si no, cuando la CPU generase un dato para una dirección que tuviesen asignada a la vez la pantalla y un chip de memoria, ¿quién cogería el dato?

El simulador de la CPU teórica permite *pinchar* y *despinchar* chips de memoria. Vamos a ver cómo está en estos momentos el sistema de memoria:

- ☐ Abre el simulador del computador teórico.
- ☐ Pulsa con el botón izquierdo del ratón sobre el dibujo etiquetado con MEMORIA en la esquina inferior izquierda.
- ☐ En el menú que te saldrá, escoge la opción Configurar.

Aparecerá una pantalla que te muestra, en la parte izquierda, el espacio de direcciones del computador teórico. En ella puedes comprobar si tu respuesta a las preguntas anteriores relativas al tamaño del espacio de direcciones fue acertada. Fíjate en que el computador tiene *pinchados* dos módulos de memoria de 32Ks, que recubren el espacio de direcciones de 64Ks sin dejar ningún hueco.

Vamos a *despinchar* un módulo de memoria para hacer sitio a la pantalla:

- ☐ Pulsa sobre el chip situado más abajo en la figura (el que cubre las direcciones desde 8000h hasta FFFFh). Ese módulo desaparecerá (será *despinchado*).

Ahora nuestro computador sólo tiene 32Ks de memoria instalados. Si intentases escribir en una posición por encima de los primeros 32Ks (por ejemplo, la B000h), el simulador te daría un error porque ¡en esa posición no hay memoria!

Hemos dejado libres 32Ks para mapear la pantalla, pero sólo necesitamos 128 direcciones, así que para tener más memoria, y ya que nos sale gratis, vamos a pinchar un módulo de 16Ks a continuación del de 32 Ks:

- ❑ Haz clic en el módulo de 16Ks dibujado en la parte derecha de la pantalla. De esta forma pasará a estar seleccionado.
- ❑ Haz clic ahora en el hueco que hay entre las direcciones 8000h y 9000h. Se situará un chip de 16Ks a partir de la dirección 8000h.

Con esto dejamos un hueco de 16Ks, suficiente para la pantalla. Vamos a conectarla:

- ❑ Cierra la ventana de Configuración de memoria pulsando el botón .
- ❑ Haz clic en el dibujo etiquetado con ENTRADA/SALIDA en la esquina inferior derecha de la ventana principal del simulador.
- ❑ Escoge la opción Conectar Pantalla.
- ❑ Te aparecerá una ventana en la que tendrás que introducir las características de la pantalla que quieres conectar. En el lugar reservado para el nombre escribe Pantalla1. Como dirección base (es decir, dirección a partir de la cual se mapearán las 128 direcciones del interfaz de la pantalla) introduce la F000h.
- ❑ Pulsa el botón . Deberás ver una ventana que representa la pantalla que acabas de conectar.

Para conectar el teclado sigue estos pasos:

- ❑ Haz clic en el dibujo etiquetado con ENTRADA/SALIDA en la esquina inferior derecha de la ventana principal del simulador.
- ❑ Escoge la opción Conectar Teclado.
- ❑ Te aparecerá una ventana en la que tendrás que introducir las características del teclado que quieres conectar. En el lugar reservado para el nombre escribe Teclado1. Como dirección base (es decir, la dirección a partir de la cual se mapearán los dos registros del teclado) introduce la F700h. Tendrás que rellenar también tres campos relativos a interrupciones aunque ahora no los necesitamos porque vamos a trabajar con entrada mediante muestreo periódico y no mediante interrupciones. Como vector de interrupción introduce el 1, como prioridad la 0 y deja sin marcar la casilla que indica si se generan interrupciones.

Con esto ya habrás equipado al computador elemental con una pantalla que estará mapeada en la dirección F000h y un teclado cuyos registros de datos y control están a partir de F700h. Para no tener que repetir todo este proceso en sucesivas ocasiones, guarda la configuración actual del simulador en un archivo llamado 3-1interfaces.sim.

5.2. Funcionamiento del interfaz de vídeo

El interfaz de la pantalla está mapeado de tal forma que a la dirección base (F000h en nuestro caso) se le asigna la posición de memoria de vídeo que representa la primera celda de la primera fila, a la siguiente dirección se le asigna la posición de memoria de vídeo que representa la segunda celda de la primera fila, etc. Después de todas las posiciones de memoria de vídeo está mapeado el registro de control.

Cada posición de la memoria de vídeo tiene la siguiente estructura:

-	-	R	G	B	R	G	B	-	-	-	-	-	-	-
		Fondo			Letra									
Atributos								Carácter						

Para representar los colores se usa el siguiente código RGB (Red, Green, Blue):

Código	Color
000	Negro
001	Azul
010	Verde
011	Cian
100	Rojo
101	Morado
110	Amarillo
111	Blanco

Para escribir un asterisco en una posición de pantalla habrá que poner el código ASCII del carácter asterisco en la parte baja de la posición de memoria de vídeo asociada a esa posición. Para obtener el código ASCII de un carácter en el ensamblador se pone el carácter entre comillas simples. Por ejemplo: '*'.

5.3. Uso del registro de control del teclado

Este registro se mapea en la segunda dirección del interfaz de teclado. Para el teclado que hemos conectado anteriormente, ¿en qué dirección estará mapeado?^[14]

14

El registro de control del teclado funciona de la siguiente forma: cuando se produce una pulsación de teclado, el bit 8 de este registro se pone a 1. Este es el hecho que vamos a utilizar para hacer muestreo periódico sobre este registro. Así, aplicando esta técnica de sincronización podemos detectar cuándo se produce una pulsación en el teclado: simplemente tendremos que estar leyendo el registro de control y, si hay un 1 en el bit 8, sabremos que se ha pulsado una tecla.

Pero ¿cómo sabemos cuando hay un 1 en el bit 8? Para ello primero tendremos que traer el registro de control del teclado a un registro de la CPU y luego realizar una operación entre ese registro y una *máscara* (una combinación de unos y ceros) que dé cero cuando el bit 8 esté a 0 y distinto de 0 cuando el bit 8 esté a 1. Como máscara vamos a utilizar el valor 0000 0001 0000 0000. ¿Qué operación necesitaremos?^[15] (Piensa una operación de la ALU que dé cero si el registro de control es, por ejemplo, 0000 0000 0000 0001, y que dé distinto de 0 si es 0000 0001 0000 0001.)

15

5.4. Mostrando un mensaje por pantalla

Vamos a hacer ahora un programa que imprima una cadena en la pantalla. El programa simplemente irá recorriendo la cadena e imprimiendo cada uno de sus caracteres en posiciones consecutivas de la pantalla hasta que se encuentre un cero. Utilizaremos los siguientes registros:

Registro	Uso
R0	Apuntará a la dirección de pantalla donde escribir
R1	Apuntará al carácter de la cadena a escribir
R2	Contendrá temporalmente el carácter a escribir
R3	Contendrá un cero para hacer comparaciones

El pseudocódigo del programa será el siguiente:

```
1  Inicializaciones
2  bucle: Traer carácter a R2
3      Si es cero
4          Salir
5      Si no
6          Preparar el atributo de R2
7          Imprimir el carácter
8          Avanzar la posición donde imprimir
9          Avanzar a la siguiente posición de la cadena
10         Ir a bucle
11     Fin Si
```

Sigue los siguientes pasos:

- ☐ Edita un fichero que se llame 3-1cad1.ens.
- ☐ El programa se debe cargar en la dirección 500h.
- ☐ Define en la sección de datos una cadena que sea "Hola, mundo". Utiliza la etiqueta cad para apuntar a ella.
- ☐ Define a continuación un dato que valga cero. Como se explicó más arriba, indicará el final de la cadena.
- ☐ Comienza el código cargando en R0 la dirección base del interfaz de la pantalla.
- ☐ A continuación carga en R1 la dirección de la primera posición de la cadena.
- ☐ Después inicializa R3 a cero.
- ☐ Vas a comenzar ahora un bucle, así que márcalo con la etiqueta bucle.
- ☐ La primera instrucción del bucle debe ser traerse el carácter desde memoria al registro R2.
- ☐ A continuación, para comprobar si ya se ha acabado la cadena o todavía quedan más caracteres, compara el carácter que te acabas de traer con cero (que, como recordarás, está en R3).

- ❑ La siguiente instrucción tiene que ser un salto condicional. Si la comparación anterior determinó que hay un cero en R2, debes saltar al final de programa, que vas a señalar convenientemente con una etiqueta llamada `final`.
- ❑ Sigamos con las instrucciones que se ejecutan cuando todavía no hemos encontrado el cero. En primer lugar, debes colocar en la parte alta de R2 un atributo que haga que se vea la cadena. Haz que la cadena aparezca en verde sobre negro.
- ❑ Introduce ahora la instrucción para escribir el carácter en la pantalla. Recuerda que tienes el carácter en R2 y la posición en la que quieres escribir en R0.
- ❑ Escribe la instrucción para que R0 apunte a la siguiente posición y así cuando escribas la próxima letra se escriba a continuación de la anterior.
- ❑ Escribe la instrucción para que R1 apunte al siguiente carácter de la cadena.
- ❑ Escribe la instrucción para hacer un salto incondicional al principio del bucle.
- ❑ Completa el programa poniendo la etiqueta `final` (si no lo has hecho ya) y la directiva de FIN.
- ❑ Guarda el archivo, sal del editor, ensambla el programa, cárgalo en el simulador y comprueba que funciona. Para no tener que darle muchas veces a `[F8]`, prueba a darle a `[F9]` (como siempre, tienes que tener la ventana principal del simulador activa para que funcionen las teclas). La CPU se pone en estado de ejecución, es decir, cada vez que acaba de ejecutar una instrucción, va a por la siguiente. Es similar a estar dándole continuamente a `[F8]`. Cuando hayas visto que la cadena se escribe en pantalla, vuelve a darle a `[F9]` para que la CPU pare de ejecutar instrucciones (de nuevo, la ventana activa debe ser la principal del simulador).

SESIÓN 2

Interrupciones: Programación de rutinas de servicio

Objetivos

En esta sesión se practicará con un periférico capaz de generar interrupciones, y se programará e instalará una rutina de servicio que atienda las interrupciones de este periférico.

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 .exe.
- Comprender claramente el mecanismo de interrupción y la forma de instalar una rutina.
- Conocer la programación del dispositivo de Salida “Pantalla”.
- Tener el fichero 3-1perifericos.sim construido durante la sesión 2 de este bloque de prácticas.

Desarrollo de la práctica

1. Configuración inicial

Carga el fichero 3-1interfaces.sim que has generado en la sesión 1, de modo que aparezcan los periféricos pantalla y teclado conectados al simulador.

Para poder estudiar qué ocurre cuando la CPU recibe una interrupción, necesitaremos conectar a ésta un dispositivo capaz de generar interrupciones. Para esta práctica usaremos el dispositivo llamado “Luces”. Realiza los siguientes pasos:

- Pulsando sobre la “Entrada/Salida” selecciona la opción “Conectar Luces”.
- Elige como nombre para el dispositivo “Luces”, como dirección base E000h, como vector el 3 y como prioridad 1. Marca la casilla “Generar Int.” para indicar que este periférico tiene capacidad de generar interrupciones.
- Cuando hayas finalizado la configuración, se te mostrará un dispositivo como el de la siguiente figura:



Verifica que en los “Datos del periférico” aparecen correctamente la dirección base, el vector, la prioridad y sobre todo que el círculo INT está marcado. Si no fuera así, debes eliminar este periférico pulsando con el ratón en “Entrada/Salida” y eligiendo “Desconectar periférico”, para seguidamente crearlo de nuevo con los datos correctos.

- Presiona el botón en esa ventana.
- Guarda la configuración que tiene en este momento el simulador, con el nombre 3-2interfaces.sim, para poder recuperarla si cierras el simulador.

2. Programación en ensamblador de rutinas de servicio

2.1. La rutina más simple posible

Vamos a escribir una rutina de interrupción que se limite a retornar cuando sea invocada, y vamos a ver cómo se instala esta rutina para que resulte ejecutada “automáticamente” cuando se produzca la interrupción.

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:

```
1 PROCEDIMIENTO rutina_minima
2   IRET
3 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 rutina. Es decir, si la rutina comenzara en 50C0h, el código necesario para “instalarla” sería algo así:

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

¿Qué instrucción iría en lugar de XXXXXX? ¹ Esta sería la instrucción que modifica el vector 3.

1

Ahora bien, el dato 50C0h ¿de dónde ha salido? Lo cierto es que desconocemos en qué dirección de memoria está la rutina (aunque podríamos calcularlo a partir de la directiva ORIGEN), es un dato inventado. Lo que hay que hacer en realidad es usar la directiva DIRECCION del compilador, para que él mismo averigüe dónde está colocada la rutina.

El código correcto sería este. Complétalo:

```
1  ORIGEN 300h
2  .PILA 20h
3  .CODIGO
4  MOVL  R0, 3    ; Vector a modificar
5  MOVH  R0, 0
6  MOVL  R1, BYTEBAJO DIRECCION rutina_minima
7  MOVH  R1, 
8  
9  STI    ; Permitir interrupciones
10
```

```

11 ; Una vez instalada la rutina, el programa
12 ; principal se encierra en un bucle infinito
13 PorSiempre:
14     JMP PorSiempre
15
16 ; Y esta sería la rutina minima:
17 PROCEDIMIENTO rutina_minima
18     
19 FINP
20
21 FIN

```

- ☐ Escribe el programa anterior (una vez lo has completado) en el fichero 3-2int1.ens y ensámblalo para obtener 3-2int1.eje
- ☐ Carga en el simulador el ejecutable que has obtenido.
- ☐ 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?^[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 ¿qué instrucción encuentras allí?^[3] 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 (en la dirección 3 debe estar almacenada 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 mostrará otra ¿cuál sería?^[4] Comprueba tu respuesta generando una interrupción con el dispositivo “Luces”.

2

3

4

- ❑ 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 enfrascado 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.

Antes de continuar, si no tienes claro cómo funciona el dispositivo “Luces”, consulta el anexo [4.1](#).

2.2. 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 `3-2int1.ens` con el nombre `3-2int2.ens` y abre éste último con EDIT.
- ❑ Ve al punto donde estaba la rutina `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 programa que has escrito, para obtener `3-2int2.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.

2.3. Juntandolo con la pantalla

Modificaremos ahora la rutina de servicio para que cada vez que se genere la interrupción, salga algo por la pantalla.

- ❑ Copia `3-2int2.ens` con el nombre `3-2int3.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, el 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?

3. Autoevaluación

3.1. Archivos en el disco

En tu disco de prácticas deberás tener los archivos de programa `3-2interfaces.sim`, `3-2int1.ens`, `3-2int1.eje`, `3-2int2.ens`, `3-2int2.eje`, `3-3int3.ens` y `3-3int3.eje`.

3.2. Ejercicios

- ⇒ Copia el fichero `3-2int3.ens` con el nombre `3-2int4.ens`.
- ⇒ Abre con el editor el fichero `3-2int4.ens` y modifica 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 video, 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 se 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.

Compila 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 ¹ e intenta encontrar una explicación a lo que ocurre. ¡Es una pregunta muy difícil! Pregúntale al profesor si tienes dudas.

4. Anexos

4.1. 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 0003, 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 dado².

¹Deberás configurar el teclado para que genere interrupciones

²Sin 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

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 valor del registro de interruptores.

SESIÓN 3

Interrupciones: Mecanismo de funcionamiento

Objetivos

Se pretende en esta sesión que el alumno conozca y comprenda claramente el mecanismo de generación y aceptación de interrupciones.

En una primera fase se estudiarán los cambios que una interrupción produce en el estado del procesador, para seguidamente observar paso a paso la secuencia de señales que motivan estos cambios. Finalmente se manipulará la memoria y la tabla de vectores de interrupción para lograr la ejecución y retorno con éxito de una rutina de servicio mínima.

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 .exe.
- Repasar en los apuntes de teoría el concepto de interrupción y el mecanismo que la unidad de control pone en marcha cuando detecta una interrupción.

Desarrollo de la práctica

1. Primeros experimentos

1.1. Configuración inicial del computador

Para poder estudiar qué ocurre cuando la CPU recibe una interrupción, necesitaremos conectar a ésta un dispositivo capaz de generar interrupciones. Para esta práctica usaremos el dispositivo llamado “Luces”, ya usado en la sesión anterior.

Carga en el simulador el estado 3-2interfaces.sim y comprueba que aparecen conectados los periféricos pantalla, teclado y luces, y que la configuración de éste último es: dirección base E000h, vector 3 y prioridad 1, y que tiene activada la capacidad de generar interrupciones.

Ahora escribe el siguiente programa en el fichero 3-3int1.ens, haciendo uso de EDIT, y seguidamente ensámblalo para obtener 3-3int1.eje haciendo uso de ensambla. El programa en sí no hace nada útil, son sólo unas pocas instrucciones para tener algo que ejecutar; excepto la primera de ellas, STI, que es fundamental para esta práctica.

```

1  ORIGEN 300h
2  .PILA 20h
3  .CODIGO
4  STI
5  MOVL R1, 05h
6  MOVH R1, 80h
7  MOV R0, R1
8  FIN

```

Finalmente, carga 3-3int1.eje en el simulador. Todo está listo ahora para iniciar la práctica.

- ☐ Guarda el estado de la CPU en este momento en el archivo 3-3int1.sim, pues necesitaremos volver a este mismo estado más adelante.

1.2. Ejecución “normal” del programa (sin interferencias)

Observa que, como es habitual, PC se ha cargado con la dirección en la que comienza nuestro programa ¿Cuál es?^[1] Por otra parte, nuestro programa tiene 4 instrucciones, por tanto, ¿en qué dirección de memoria estaría el primer NOP que pone fin a nuestro programa?^[2] Compruébalo con el desensamblador de la memoria.

A partir de esa dirección, comienza la pila. Ya que hemos indicado un tamaño de 20h mediante el uso de .PILA, ¿en qué dirección debería terminar la zona de pila?^[3] ¿Coincide este valor con el registro R7?^[4]

Si ahora pulsáramos varias veces **F8**, las diferentes instrucciones se irían ejecutando en secuencia. Ya que ninguna de ellas es una instrucción de salto, el valor de PC

1

2

3

4

simplemente se irá incrementando. Por otra parte, tampoco hay ninguna instrucción PUSH ni POP ni CALL ni RET, de modo que la pila no se toca nunca y por tanto R7 no debería cambiar de valor.

- ☐ Pulsa **[F8]** cuatro veces comprobando tras cada pulsación que efectivamente PC va tomando valores consecutivos comenzando en 300h y que R7 en todo momento tiene el mismo valor.
- ☐ Cuando IR muestre la instrucción MOV R0,R1 el programa habrá finalizado, y R0 deberá contener 8005h

Hasta aquí, todo era conocido. Ahora vamos a repetir la ejecución del programa, pero causando desde el periférico una interrupción hacia la mitad del mismo, para ver qué ocurre.

1.3. Ejecución interrumpida por el periférico

- ☐ Carga de nuevo el estado 3-3int1.sim para volver a la situación inicial. Comprueba los valores en PC y R7.
- ☐ Pulsa **[F8]**. Se ejecuta la primera instrucción STI. ¿Ha cambiado algún registro en el procesador, además de PC? (El simulador muestra en color rojo los registros que han cambiado como consecuencia del último ciclo de reloj.) ¿Cuál?^[5]
- ☐ Comprueba que el bit IF de SR ahora está a 1. Esto significa que el procesador admite ser interrumpido.
- ☐ La siguiente instrucción a ejecutar es MOVL R1, 5. Si pulsáramos de nuevo **[F8]**, ¿qué valor aparecería en R1?^[6] ¿Y en PC?^[7] Pulsa **[F8]** y comprueba que es así.

5

6

7

Podríamos seguir pulsando **[F8]** y la ejecución proseguiría como antes. Pero lo que vamos a hacer es solicitar una interrupción desde el periférico “Luces”:

- ☐ Haz reaparecer la ventana del periférico “Luces” (en la barra de tareas de Windows, en la parte inferior de la pantalla, debe haber un botón con el título “Luces”. Basta pulsar sobre él).
- ☐ Pulsa el botón “Generar Interrupción **[INT]**”.
- ☐ Vuelve a minimizar esta ventana y observa el dibujo de la CPU. ¿Ves una línea llamada INT que ahora aparece en color rojo? Esto indica que un dispositivo de E/S ha solicitado una interrupción.

Por el momento, la CPU aún no se ha dado cuenta de que INT está activada, ya que sólo examina esa línea cuando la Unidad de Control genera la señal FIN, pero ya había generado esa señal antes de que nosotros activáramos INT, por tanto “no se dará cuenta” hasta la próxima activación de FIN.

Si ahora pulsáramos **F8** (no lo hagas), se ejecutaría la próxima instrucción MOVH R1, 80h y cuando esta ejecución llegara a su FIN, entonces sería cuando la unidad de control miraría el estado de INT y la encontraría activada. Antes de pulsar **F8**, responde a las siguientes preguntas:

- ☐ ¿Qué valor aparecerá en R1?^[8] (Como consecuencia de la ejecución del MOVH.) 8
- ☐ ¿Qué valor esperarías en PC si la ejecución fuese “normal”, es decir, sin interrupciones?^[9] 9
- ☐ ¿Qué valor hay en R7?^[10] 10

Ahora pulsa **F8**. En un breve instante ocurren muchas cosas, puesto que se ejecuta la instrucción MOVH, y a continuación se detecta INT, lo que causa unos importantes cambios en varios registros del procesador. Tratemos de localizar esos cambios:

- ☐ ¿Qué valor aparece en R1?^[11] ¿Era el esperado (cuestión 8)? 11
- ☐ ¿Qué valor aparece en PC?^[12] ¿Era el que habías respondido en la cuestión 9? Después veremos de dónde ha salido este valor. 12
- ☐ Fíjate en el registro de estado. ¿Ha cambiado de valor?^[13] 13
- ☐ ¿Qué valor hay ahora en R7?^[14] ¿En cuánto se diferencia del que habías respondido en 10?^[15] 14
- ☐ La respuesta anterior implica que algo ha sido introducido en la pila. Si utilizas el editor hexadecimal para mirar lo que hay en la dirección apuntada por R7 encontrarás qué es lo que se ha guardado en la pila. ¿Cuál es el primer número que encuentras allí (en la cima de la pila)?^[16] ¿Te suena este número? Fíjate que coincide con una de tus respuestas anteriores. 15
- ☐ Y ¿cuál es el número que hay en la pila justo a continuación?^[17] ¿Este es el antiguo valor del registro de estado! 16
- ☐ Observa que la línea INT ya está de nuevo negra. Esto implica que nuestro periférico “Luces” la ha desactivado. ¿Por qué habrá hecho esto? En algún momento (entre todas estas cosas que acaban de ocurrir) la CPU le ha dicho a “Luces” que desactive INT. Más adelante lo veremos en detalle. 17
- ☐ Observando el nuevo valor de PC y utilizando el desensamblador de la memoria puedes saber cuál será la próxima instrucción que va a ejecutar la CPU. ¿Cuál sería?^[18] 18
- ☐ Teniendo en cuenta que INT ya no está activa (y además IF es cero), si pulsamos **F8** la siguiente instrucción se ejecutará normalmente y sin interrupciones. ¿Cuál sería el siguiente valor de PC al pulsar **F8**?^[19] 19
- ☐ Pulsa **F8** y comprueba tus dos respuestas anteriores.

Observa que a partir de este instante, todo irá mal en nuestra CPU. El control ha saltado a la dirección de memoria 0000, donde no hay nada para ejecutar salvo NOP. Nuestro programa ha sido abandonado a la mitad y nunca se regresará a él. La información que se guardó en la pila nunca es extraída.

Todas estas calamidades se deben a que se produjo una interrupción, pero el sistema no tenía correctamente preparada la tabla de vectores de interrupción ni las rutinas de servicio.

2. Instalando una rutina de servicio “a mano”

Como ya has aprendido en la sesión anterior, “instalar” una rutina de servicio consta en realidad de tres sencillos pasos:

1. Escribir el programa que queremos que se ejecute cuando se reciba una interrupción de un periférico concreto. Este programa se escribe como si fuera un procedimiento, sólo que finaliza con la instrucción IRET en lugar de RET.
2. Convertirlo en código máquina y cargarlo en algún lugar de la memoria, digamos en la dirección de memoria X .
3. Modificar la tabla de vectores de interrupción, introduciendo el valor X en una de sus posiciones (la posición concreta será la del número de vector asociado con el periférico).

En la sesión anterior, la rutina la escribíamos como parte del programa principal, usando el lenguaje ensamblador y un editor de texto. La convertíamos en código máquina con ayuda del programa `ensambla`, y finalmente, para modificar la tabla de vectores de interrupción, usábamos la instrucción MOV con los parámetros apropiados para que escribiera en la dirección 3 (vector), la dirección de la rutina. En esta sesión haremos manualmente todo lo anterior, para comprender más claramente el mecanismo.

Para simplificar, la rutina de servicio que instalaremos será la más simple posible, es decir, que no haga absolutamente nada útil, salvo retornar. Por tanto constará de una sola instrucción: IRET.

Usando las tablas de codificación, vemos que el código máquina de IRET es B800. Así pues, ya tenemos resuelto el primer paso (escribir la rutina y convertirla en código máquina). Vamos ahora a “instalar” esta rutina en nuestro computador.

- ❑ Carga de nuevo en el simulador el estado `3-3int1.sim`, para devolver el sistema a su estado inicial, con nuestro programa de prueba cargado en memoria y los registros PC y R7 correctamente inicializados.

- ❑ Vamos a colocar nuestra “mini-rutina de servicio” en la dirección de memoria $X = 50C0h$ (se trata de un valor cualquiera elegido al azar, entre las diferentes direcciones de memoria que tenemos disponibles, con la precaución de que no sea una zona utilizada por nuestro programa de prueba ni su pila).
- ❑ Abre el editor hexadecimal de la memoria y ve a la posición $50C0h$
- ❑ Introduce allí el código máquina de IRET. Cierra el editor hexadecimal y comprueba con el desensamblador que la codificación ha sido correcta.

La rutina ya está en memoria. Ahora tenemos que relacionar esta rutina con el periférico “Luces”, de modo que cada vez que “Luces” genere una interrupción, se ejecute nuestra rutina. Para ello, hay que modificar la tabla de vectores de interrupción. La tabla de vectores de la CPU elemental comienza en la dirección de memoria 0. Cada vector ocupa una posición, por tanto para modificar un vector de la tabla basta modificar la dirección de memoria igual al número del vector.

- ❑ ¿Cuál es el número de vector que hemos asignado al periférico “Luces”?^[20]
(Puedes verlo de nuevo abriendo la ventana de este periférico, pulsando sobre su nombre en la barra de tareas de Windows.)
- ❑ Esa será la dirección de memoria que tenemos que modificar ahora. Abre el editor hexadecimal y ve a esa posición de memoria. ¿Qué había allí?^[21]
- ❑ Fíjate que esta respuesta explica por qué PC tomaba el valor 0000 tras aceptarse la interrupción. El valor que toma PC al aceptar una interrupción es el que se halla almacenado en la tabla de vectores.
- ❑ Modifica el contenido de esa dirección, y pon $50C0$ en su lugar. Date cuenta que este número es la dirección donde hemos puesto nuestra rutina de servicio.
- ❑ Cierra el editor hexadecimal de la memoria.
- ❑ Guarda el estado del simulador en el archivo 3-3int2.sim, pues lo necesitaremos más adelante.

20

21

¡Nuestra rutina ya está “instalada”! A partir de este instante, cuando “Luces” genere una interrupción, PC tomará el valor $50C0$ en vez de 0000 , por lo que ahora en lugar de ejecutar NOP, se ejecutará el IRET que hemos colocado allí. ¡Y por tanto la ejecución volverá a nuestro programa que podrá seguir funcionando!

Comprobemos todo esto:

- ❑ Pulsa **[F8]**. Esto ejecuta STI y el bit IF se pone a 1, con lo que estamos listos para aceptar interrupciones.
- ❑ Pulsa **[F8]**. Esto ejecuta MOVL R1, 5
- ❑ Abre la ventana del periférico “Luces” y pulsa sobre “Generar Interrupción **[INT]**”. Observa cómo la línea INT se pone roja en la CPU.

- ❑ Si ahora pulsáramos **[F8]**, tendría lugar la ejecución de la instrucción actual (MOVH), e inmediatamente, al estar INT activada, tendrían lugar los cambios que ya conocemos (apilación de PC y del registro de estado, modificación de R7 y de PC). Veamos si has comprendido los cambios que ocurrirán:

- ¿Qué valor aparecerá en R7?^[22]
- ¿Qué valor aparecerá en PC?^[23] (Recuerda que se trata del valor que hemos guardado en la tabla de vectores, que ya no es 0000.)
- ¿Qué dos datos quedarán guardados en la pila?^[24]

22

23

24

- ❑ Pulsa **[F8]** y verifica tus respuestas anteriores. En la situación actual ¿qué instrucción será la próxima en ejecutarse?^[25]

25

- ❑ La instrucción IRET simplemente saca dos datos de la pila (por lo que R7 se incrementa en 2), y guarda el primero de ellos en PC y el segundo en el registro de estado. Por tanto, al ejecutar esta instrucción

- ¿Qué valor aparecerá en R7?^[26]
- ¿Qué valor aparecerá en PC?^[27]
- ¿Qué valor tomará el bit IF del registro de estado?^[28]

26

27

28

- ❑ Pulsa **[F8]** y verifica tus respuestas anteriores. En la situación actual ¿cuál sería la próxima instrucción en ejecutarse?^[29]

29

Observa que en esta ocasión la interrupción ha causado la ejecución de una rutina “ajena” a nuestro programa (una rutina que en este caso tan sólo contenía un IRET), pero que la ejecución prosigue en el punto en que nuestro programa fue interrumpido. En esta situación podríamos generar una nueva interrupción, que sería atendida tan pronto como la CPU terminara de ejecutar la instrucción que le toca.

3. Ejecución paso a paso de la aceptación de interrupción

Ya sabemos qué ocurre cuando la CPU acepta una interrupción. Veremos ahora qué señales va activando la unidad de control para lograr llevar a cabo todas las acciones necesarias (apilar SR y PC, consultar la tabla de vectores, asignar a PC el nuevo valor obtenido de la tabla y desactivar el bit IF).

- Carga de nuevo el estado 3-3int2.sim en el simulador. Tenemos de nuevo nuestro programa a punto de empezar, y la rutina “instalada” en la dirección 50C0h.
- Pulsa **[F8]** para ejecutar STI. La próxima instrucción sería MOVL R1,5, pero esta vamos a ejecutarla paso a paso (**[F7]**) para ver qué ocurre exactamente en el instante en que la interrupción se acepta.
- Pulsa **[F7]** dos veces para ejecutar los dos primeros ciclos de traída e incremento del PC.
- Abre la ventana del periférico “Luces” y pulsa sobre “Generar Interrupción **[INT]**”. Observa cómo la línea INT en la CPU se pone de color rojo.

- Hemos activado INT mientras la CPU estaba “en medio” de la ejecución de una instrucción. Esta situación es muy normal ya que cuando un periférico activa INT no sabe qué está haciendo la CPU en ese instante. Lo más probable es que la pille “en medio” de una ejecución.
- Pulsa **[F7]** para pasar al siguiente ciclo de ejecución. Observa que la unidad de control prosigue con su secuencia de pasos “normal”, ignorando la línea INT activada. Sólo examinará esta línea cuando llegue a la señal FIN. Por tanto ahora se finaliza el ciclo 3 que completa la traída de la instrucción. Observa cómo en IR aparece la instrucción `MOVL R1, 5`.
- Pulsa de nuevo **[F7]** para ejecutar el cuarto (y último) ciclo de esta instrucción. Observa que la unidad de control activa la señal FIN. Esto implica que en este momento, también acaba de “darse cuenta” de que la línea INT está activa. Y puesto que el bit IF también está activo, la unidad de control ya ha decidido “aceptar” la interrupción.
- Mirando el simulador no encontramos indicios de que la línea INT haya sido reconocida, pero sí lo ha sido. Esto será patente cuando pulsemos de nuevo **[F7]**, ya que entonces, en lugar de volver al ciclo 1 como era habitual, la unidad de control entrará en una nueva secuencia de ciclos, específica para “aceptar” interrupciones.
- Pulsa **[F7]** y observa la unidad de control. Fíjate en el número del paso. Ya no es 1, sino I-1 (la I nos recuerda que se halla en una secuencia de señales especial para el tratamiento de Interrupciones).
- Las señales que la Unidad de Control está generando tienen por objetivo decrementar el valor de R7. ¿Qué valor hay en TMPS?^[30] Observa que es uno menos que el valor de R7. 30
- El valor que has obtenido en TMPS es justamente el lugar de la pila donde hay que almacenar una copia del registro de estado, por tanto las señales siguientes que generará la UC irán preparando una copia de SR en MDR. ¿Qué señales serán necesarias para esto?^[31] 31
- Pulsa **[F7]** y verifica tu respuesta.
- Ya tenemos el dato en MDR, falta poner la dirección en MAR. Como hemos dicho, esta dirección está en TMPS. ¿Qué señales debes activar para moverla a MAR?^[32] 32
- Pulsa **[F7]** y observa las señales que genera la UC. Las señales que acabas de responder tienen que aparecer allí, pero aparecen algunas más. Fíjate que la UC aprovecha el dato que hay en el bus interno para cargarlo también en R7, de modo que R7 queda así decrementado. Además se activa la señal WRITE para que la memoria guarde el dato en la dirección que se le indica (tardará un par de ciclos en hacer esto).
- Mientras la memoria guarda ese dato, la UC se prepara para repetir la jugada, es decir, decrementar R7 para enviar a esa dirección de memoria una copia de PC. Pulsa **[F7]** y comprueba como el ciclo I-4 es idéntico al ciclo I-1 que ya habíamos visto (pues el objetivo es el mismo: decrementar R7).

- Ahora hay que copiar PC en MDR para enviarlo a continuación a la memoria, como se hizo antes con SR. Intenta anticipar las señales que son necesarias para esto y pulsa **F7** dos veces más para ver cómo lo hace la UC.

- Observa un detalle en este último ciclo (I-6). Además de las señales necesarias para escribir en la memoria, la UC activa otra señal llamada INTA. Si te fijas en la zona de “Entrada/Salida” ves que esa línea INTA llega hasta allí, y está de color rojo. Esta línea es recibida por todos los periféricos que tengamos conectados en nuestro computador (ahora mismo sólo hay uno, que es “Luces”). Con esta línea la CPU está preguntando ¿quién de vosotros ha activado INT?

El periférico concreto que haya sido, deberá responder poniendo en el bus de datos su número de interrupción. En nuestro caso, será “Luces” quien responda, poniendo un 3 en el bus de datos. En preparar su respuesta tardará un ciclo.

- Pulsa **F7**. Observa que la UC no genera ninguna señal. Es un ciclo de espera mientras el periférico que causó la interrupción está preparando su respuesta en el bus de datos.

- Pulsa de nuevo **F7**. ¿Qué aparece en el bus de datos del sistema (SDB)?^[33] Este es el vector que le hemos asignado a “Luces” cuando lo conectamos.

33

Observa que la unidad de control simplemente toma ese dato (que obtiene en MDR) y lo transfiere a MAR, activando LEER. ¿Qué dirección de memoria tratamos de leer?^[34] ¿Qué valor hay allí?^[35] (Puedes usar el editor hexadecimal de la memoria para consultarlo.)

34

35

- Es decir, la unidad de control está usando el número de vector para obtener la dirección de la rutina a la que debe saltar. Pulsa **F7** y observa cómo la UC está esperando por la respuesta de la memoria.

- En el siguiente ciclo, la memoria habrá respondido, y su respuesta es el nuevo valor que debe ser asignado a PC. ¿Qué señales crees que activará ahora la unidad de control?^[36] Pulsa **F7** y verifica tu respuesta.

36

- Además de las señales que habías respondido, la UC genera la señal CLI cuya función es borrar el bit IF, y FIN, con lo que todo este ciclo especial finaliza. El próximo paso que genere la UC será ya el paso 1 “habitual”.

- Pulsa **F7** y comprueba cómo efectivamente la ejecución vuelve a la normalidad comenzando por el Paso 1.

- ¿Qué instrucción será la que se reciba en el paso 3?^[37]

37

- Pulsa **F7** dos veces y comprueba tu respuesta.

- Puedes seguir pulsando **F7** para ver cómo se lleva a cabo la instrucción IRET, o puedes pulsar **F8** si quieres que se ejecute todo de un tirón hasta el final. La función de IRET como hemos visto es recuperar de la pila lo que se guardó allí durante los ciclos especiales I-1 a I-10, y de este modo causar el retorno al punto en que el programa fue interrumpido.

4. Autoevaluación

4.1. Archivos en el disco

En tu disco de prácticas deberás tener los archivos de programa 3-3int1.ens, 3-3int1.eje, 3-3int1.sim y 3-3int2.sim.

4.2. Ejercicios

- ⇒ Conectaremos otro periférico a la CPU e instalaremos una rutina para servir a este nuevo periférico. Para ello:
1. Carga en el simulador el archivo 3-3int2.sim.
 2. Conecta un periférico más, de tipo Luces, con los siguientes parámetros: nombre “Más Luces”, dirección base F010h, número de vector 7, prioridad 2, “Generar int.” permitido (marca la última casilla).
 3. Guarda el estado actual de la simulación en el archivo 3-3int3.sim.
 4. Si este nuevo periférico generara una interrupción ¿a qué dirección de memoria saltaría el procesador?^[38] Compruebalo pulsando **F8** para ejecutar STI y seguidamente genera una interrupción en el periférico “Más luces”. Pulsa **F8** otra vez y mira el nuevo valor de PC para ver si habías acertado.
 5. Carga en el simulador el fichero 3-3int3.sim para recuperar el estado que tenías en el punto 3.
 6. Instala una rutina que contenga una sola instrucción (IRET) en la dirección de memoria 6000 y modifica el vector 7 para que apunte a ella.
 7. Guarda el estado de nuevo en el archivo 3-3int4.sim.
 8. Repite el punto 4 y comprueba que ahora se salta a la dirección 6000h.
 9. Carga otra vez el estado 3-3int4.sim y repite de nuevo el paso 4, pero esta vez genera la interrupción desde el periférico “Luces”, en lugar de “Más luces”. Como verás, según cuál sea el periférico que causa la interrupción se salta a una dirección u otra de la memoria.
- ⇒ Carga el estado 3-3int2.sim. Pulsa **F8**. Ahora genera una interrupción desde el periférico “Luces”. Pulsa **F8** de nuevo. Si todo va bien, PC debe tener el valor 50C0 y la CPU está a punto de ejecutar IRET. Si ahora “Luces” generase otra interrupción, ¿qué crees que ocurriría? ¿En qué momento la UC “se dará cuenta” de esta nueva interrupción? Comprueba con el simulador lo que ocurre.