Bloque 3

Entrada salida sobre el computador elemental

Prácticas de
Fundamentos de Computadores

SESIÓN 1

Interfaz de vídeo 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 la interfaz de vídeo del computador teórico.

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 de la interfaz de vídeo del computador teórico.

Desarrollo de la práctica

1. El mapa de memoria y los periféricos

En esta práctica se va a conectar un periférico al computador teórico: una pantalla (de 8 filas por 15 columnas). Para imprimir caracteres en esta pantalla se deben escribir sobre la memoría de video de su interfaz[1], que estará mapeada secuencialmente a partir de una dirección dada que podremos escoger.

¿Pero 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?[1] Si la posición más baja es la `0000h`, ¿cuál será la más alta?[2] La pantalla necesitará 120 (8×15) de estas direcciones para mapear su

1

2

[1] Se utilizan indistintamente las denominaciones de interfaz de video e interfaz de pantalla.

memoria de vídeo. Además, tiene un registro de control que también deberá ser mapeado en el espacio de direcciones. Por tanto, la 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 tanto, la interfaz de la pantalla necesitará 128 direcciones del espacio direccinable de la CPU.

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 dispositivos de memoria. Pero si tenemos que reservar algunas direcciones para la pantalla, tendremos que eliminar alguno de estos dispositivos. Si no, cuando la CPU generase un dato para una dirección que tuviesen asignada a la vez la interfaz de pantalla pantalla y un dispositivo de memoria, ¿quién cogería el dato?

El simulador de la CPU teórica permite *conectar* y *desconectar* dispositivos 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 conectados dos módulos de memoria de 32K palabras, que recubren el espacio de direcciones de 64Ks sin dejar ningún hueco.

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

- ❏ Pulsa sobre el dispositivo situado más abajo en la figura (el que cubre el rango de direcciones `8000h` hasta `FFFFh`). Ese módulo desaparecerá (será desconectado).

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 interfaz de pantalla, pero sólo necesitamos 128 direcciones, así que para tener más memoria vamos a conectar un módulo de 16Ks a continuación del de 32Ks:

- ❏ 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 interfaz de pantalla. Vamos a conectarla:

- ❏ Cierra la ventana de `Configuración de memoria` pulsando el botón Aceptar.
- ❏ 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 Aceptar. Deberás ver una ventana que representa la pantalla que acabas de conectar.

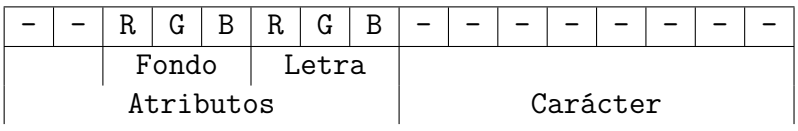
Con esto ya habrás equipado al computador elemental con una pantalla cuya interfaz estará mapeada a partir de la dirección `F000h`. Para no tener que repetir todo este proceso en sucesivas ocasiones, guarda la configuración actual del simulador en un archivo llamado `3-1pantalla.sim`.

2. Análisis de la salida mapeada en memoria

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 interfaz de la pantalla está mapeada 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:



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	Magenta
110	Amarillo
111	Blanco

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

En la parte alta de la dirección de la memoria de vídeo hay que fijar los atributos del carácter. Vamos a empezar siendo discretos: el asterisco deberá ser blanco sobre fondo negro. ¿Cuánto deberá valer el byte de atributo?[3]

[3]

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 a partir de la dirección `0500h`.
- ❏ El código debe empezar cargando en el registro `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 el registro `R0`?[4] 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 de la 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 color blanco sobre fondo negro.
- ❏ 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 `.eje`, 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.

[4]

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?[5]
- ❏ Haz que el asterisco se escriba en la segunda celda de la segunda fila. ¿Cuál es la dirección de memoria asociada con esa celda?[6] (Recuerda que las dimensiones de la pantalla son 8 filas por 15 columnas.)
- ❏ Guarda, ensambla y ejecuta el programa para comprobar que funciona correctamente.

[5]

[6]

Vamos a ver ahora cómo funciona el programa a nivel de señales de control:

- ❏ Carga el archivo `3-1pantalla.sim`.
- ❏ Carga el archivo `3-1ast2.eje`.
- ❏ Ejecuta las cuatro primeras instrucciones con F8. La quinta instrucción del programa debería ser la que escribe el asterisco en pantalla, así que la vamos a ejecutar paso a paso.
- ❏ Ejecuta los tres primeros pasos con F7 y comprueba que estás en la instrucción deseada.
- ❏ Antes de ejecutar el siguiente paso, contesta a estas preguntas: ¿qué señales se activarán?[7] Pulsa F7 y comprueba que has respondido adecuadamente.
- ❏ ¿Qué señales se activarán en el siguiente paso (T5)?[8] ¿Aparecerá en este paso el asterisco en pantalla?[9] ¿Por qué? Pulsa F7 y comprueba si has respondido correctamente. Fíjate en la señal *WRITE* abajo a la derecha: como la dirección que hay en MAR pertenece al espacio de direcciones de E/S, la señal va dirigida al módulo de E/S y no al de memoria.
- ❏ ¿Qué señales se activarán en el siguiente paso (T6)?[10] ¿Aparecerá ahora el asterisco?[11] Pulsa F7 y compruébalo.

7

8

9

10

11

3. Un programa más complejo: imprimir cadenas

Vamos a hacer ahora un programa que imprima una cadena de caracteres 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 (terminador de la cadena). 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 atributos en 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 a partir de la dirección `0500h`.
- ❏ 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 ha comentado anteriormente indicará el final de la cadena.
- ❏ Comienza el código cargando en el registro `R0` la dirección base de la interfaz de la pantalla.
- ❏ A continuación, carga en el registro `R1` la dirección de la primera posición de la cadena.
- ❏ Después inicializa el registro `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 acabas de traer con cero (que, como recordarás, está en el registro `R3`).
- ❏ La siguiente instrucción tiene que ser un salto condicional. Si la comparación anterior determinó que hay un cero en el registro `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 del registro `R2` un atributo que haga que se vea la cadena. Haz que la cadena aparezca en color verde sobre fondo negro.
- ❏ Introduce ahora la instrucción para escribir el carácter en la pantalla. Recuerda que tienes el carácter en el registro `R2` y la posición en la que quieres escribir en el registro `R0`.
- ❏ Escribe la instrucción para que el registro `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 el registro `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 correctamente. Para no tener que pulsar muchas veces F8, prueba a pulsar F9 (como siempre, tienes que tener la ventana principal del simulador activa para que el simulador reciba estas órdenes). 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 pulsar continuamente F8.

Cuando hayas visto que la cadena se escribe en pantalla, vuelve a pulsar F9 para que la CPU pare de ejecutar instrucciones (de nuevo, la ventana activa debe ser la principal del simulador).

4. Ejercicios adicionales

4.1. Archivos en la carpeta de trabajo

En tu carpeta de trabajo de prácticas deberás tener los archivos de programa `3-1pantalla.sim`, `3-1ast1.ens`, `3-2ast2.ens` y `3-1cad1.ens`.

4.2. Ejercicios

- ✏ Carga el archivo `3-1pantalla.sim` y vete a la opción de configurar memoria. Rellena con dispositivos de memoria todo el espacio posible hasta llegar a la dirección de la pantalla pero sin sobreescribirla. ¿Cuántos dispositivos has usado?[12] ¿De qué tamaño?[13] Carga el archivo `3-1cad1.ens` y comprueba que sigue funcionando con la nueva configuración.

12

13

- ✏ Escribe un programa que borre toda la pantalla. Para ello deberás escribir en el bit de peso 0 del registro de control un 1. Recuerda que el registro de control está mapeado después de la memoria de vídeo. Para escribir un 1 en un bit del registro de control sin modificar el resto de bits debes leer primero el contenido del registro y luego, utilizando una máscara y una operación lógica, cambiarlo. Cuando hayas escrito y ensamblado el programa, carga el archivo `3-1pantalla.sim`, y carga y ejecuta el archivo `3-1cad1.ens` para escribir algo en pantalla. Carga después el programa para borrar la pantalla y ejecútalo para comprobar que funciona correctamente.

- ✏ Transforma el código del programa `3-1cad1.ens` 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 protector de pantalla 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 de la 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 `3-1pantalla.sim`. Añade otra pantalla al computador teórico con dirección base `D700h`. Modifica el protector de pantalla para que vaya escribiendo en las dos pantallas a la vez.

SESIÓN 2

Entrada por muestreo. El teclado

Objetivos

En esta sesión se pretende que el alumno se familiarice con uno de los mecanismos clásicos de sincronización de las operaciones de E/S, el denominado *muestreo periódico*. Para ello se desarrollarán diversos ejemplos utilizando la interfaz del teclado del computador teórico.

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 sobre el funcionamiento del teclado y su interfaz, prestando una especial atención a la funcionalidad de los dos registros básicos de la interfaz de teclado: el de registro de datos y el registro de estado y control.
- Repasar la teoría relativa a entrada mediante muestreo periódico.
- Tener el archivo `3-1pantalla.sim` de la sesión anterior.

Desarrollo de la práctica

1. Conexión del teclado

En esta práctica se van a desarrollar ejemplos de entrada mediante muestreo periódico a través del teclado, así que lo primero que tendremos que hacer es conectarle un teclado a nuestro computador teórico. Para ello:

❑ Abre el simulador.

- ❏ Carga el archivo `3-1pantalla.sim` que has creado en la sesión anterior. Recuerda que en este archivo se guardaba un estado del simulador en el que habías dejado sin chips de memoria 16K a partir de la dirección `C000h` y habías situado la interfaz de una pantalla a partir de la dirección `F000h`, ocupando 121 posiciones del espacio direccionable.
- ❏ 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 `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 1, como prioridad 0 y deja sin marcar la casilla que indica si se generan interrupciones.
- ❏ Guarda el estado actual del simulador del computador en un archivo llamado `3-2Teclado.sim`. Te servirá para acceder rápidamente a un computador con una pantalla y un teclado conectados.

2. Muestreo periódico

En este apartado vamos a realizar un ejemlo de programación del teclado utilizando la técnica de sincronización denoinada *muestreo periódico*. Para ello debemos conocer el funcionamiento del registro de estado y control del teclado. Este registro se mapea en la segunda dirección de la interfaz de teclado. Para el teclado que hemos conectado anteriormente, ¿en qué dirección estará mapeado?[1]

[1]

El registro de estado y control del teclado funciona de la siguiente forma: cuando se produce una pulsación de teclado, el bit de peso 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 estado y 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 de peso 8? Para ello primero tendremos que traer el contenido del registro de estado y control del teclado a un registro de propósito general 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?[2] (Piensa una operación de la ALU que dé 0 si el registro de estado y control es, por ejemplo, `0000 0000 0000 0001`, y que dé distinto de 0 si es `0000 0001 0000 0001`.)

[2]

Vamos a hacer un programa para ver cómo funciona la entrada por muestreo. El programa consistirá en un bucle infinito en el que se estará comprobando si hay una tecla pulsada. Cuando se detecte la pulsación de una tecla, se imprimirá un asterisco en la pantalla del computador teórico.

El listado, con algunas instrucciones incompletas, sería el siguiente:

```
1  ORIGEN 300h
2  .CODIGO
3    MOVL R0, ??h          ; Cargar la dirección del registro de estado
4    MOVH R0, ??h          ; y control 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 estado y 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
25 FIN
```

- ❏ Crea un archivo llamado `3-2pulsa.ens`.
- ❏ Copia el listado anterior completando los huecos indicados por las interrogaciones para que las instrucciones hagan lo que indican los comentarios. No hace falta que copies dichos comentarios en tu fichero fuente.
- ❏ Guarda, ensambla y carga el programa en el simulador.
- ❏ Vete ejecutando 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 pulsando 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?[3] ¿Por qué? Comprueba si has acertado pulsando F8.

[3]

- ¿Cuántas veces más deberás pulsar F8 para que se imprima el asterisco en pantalla?[4] Compruébalo pulsando F8 hasta que aparezca el asterisco.

[4]

- Parece que el programa funciona... Pero vamos a probarlo un poco más. Sigue pulsando F8 hasta que aparezca de nuevo en el registro `IR` el código máquina de la instrucción `MOV R2, [R0]`. Esta instrucción acaba de traer el contenido del registro de control de la interfaz del teclado al registro `R2` de la CPU. Según el valor de ese registro, ¿se ha pulsado alguna tecla?[5] 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.

[5]

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 de peso 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, pulsa F9 y observa la pantalla. Cuando te canses de ver aparecer asteriscos que no deberían aparecer, pulsa de nuevo a F9 para detener la simulación.

Vamos a arreglar el problema:

- Edita otra vez el archivo `3-2pulsa.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 rango de direcciones asignado para la interfaz de teclado. ¿Cuál es esa dirección?[6] Añade las instrucciones necesarias para inicializar antes del bucle el registro `R6` a ese valor.

[6]

- 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 `3-2Teclado.sim` para volver a la situación inicial y luego carga el archivo con el programa modificado, es decir, `3-2pulsa.eje`.
- Ejecuta instrucciones hasta que hayas ejecutado el bucle una vez y llegues a la instrucción `BRZ -3`. Fíjate que el programa sigue funcionando mientras no se pulse ninguna tecla del teclado conectado al computador teórico.
- Pulsa una tecla en el teclado del computador teórico.
- Pulsa F8 hasta que llegues otra vez a la instrucción `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?[7]
- ❏ Pulsa F8 hasta que aparezca la instrucción de lectura del registro de estado y control, es decir, `MOV R2, [R0]`. Mira `R2`. Según el registro de estado y control, ¿hay alguna tecla pulsada?[8]
- ❏ Pulsa F9 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.

7

8

3. Códigos ASCII

En este apartado vamos a complicar un poco el programa realizado en el apartado anterior. No sólo detectaremos que se produce una pulsación de teclado, sino que también examinaremos qué tecla se ha pulsado. Para ello utilizaremos el registro de datos del teclado. Cuando se pulsa una tecla, en este registro queda registrado de dos formas qué tecla se ha pulsado:

- En el byte bajo se almacena el código ASCII de la tecla pulsada.
- En el byte alto se almacena el código SCAN asociado a la tecla.

Vamos a modificar el programa anterior para que cada vez que se pulse una tecla, en lugar del asterisco, se imprima el carácter correspondiente a esa tecla. Sigue estos pasos:

- ❏ Edita el archivo `3-2pulsa.ens` y guárdalo con el nombre `3-2letra.ens`.
- ❏ Busca la instrucción que lee el registro de datos (es la última que añadiste). Después de esa instrucción tendrás en `R5` el contenido del registro de datos. Para imprimir por pantalla la letra correspondiente a esa tecla necesitamos tener en la parte baja el código ASCII de la letra. Este código lo genera la interfaz de teclado, así que ya lo tenemos en el registro `R5`. En la parte alta necesitamos poner el byte de atributo. Añade una instrucción después de ésta que ponga como atributo en `R5` un `7` (carácter blanco sobre fondo negro).
- ❏ Ya tenemos en `R5` el atributo y el código ASCII, así que sólo queda escribirlo en pantalla. Sustituye la instrucción `MOV [R3], R4`, que escribía siempre un asterisco, por otra que escriba el carácter leído.
- ❏ Guarda y ensambla el programa. En el simulador, carga el archivo `3-2Teclado.sim` para que vuelva a la situación inicial. A continuación carga el archivo que acabas de ensamblar.
- ❏ Ejecuta el programa con F9. Comprueba que funciona pulsando teclas en el teclado del computador teórico. Si no funciona correctamente, ejecuta el programa paso a paso para intentar detectar el origen del problema.

Ahora vamos a complicar un poco más el programa. Vamos a hacer que cuando se pulse la barra espaciadora se vaya a un bucle infinito en el que ya no se compruebe si se pulsan más teclas. Sigue estos pasos:

- ❏ Edita el fichero `3-2letra.ens`.
- ❏ Busca la inicialización del registro `R4` (está antes del bucle). Esa inicialización es un resto de cuando escribíamos asteriscos. Ahora ya no la necesitamos, pero vamos a utilizar `R4` para contener un valor que nos permita detectar si se pulsa la barra espaciadora. Como valor vamos a usar en la parte baja el código ASCII de la barra espaciadora y en la alta el código SCAN. En hexadecimal sería el valor `3220h`. Modifica las instrucciones de inicialización de `R4` para que carguen este valor.
- ❏ Busca en el código la instrucción donde se lee el byte de datos del teclado.
- ❏ Justo después de esa instrucción vas a introducir dos nuevas instrucciones para ver si se ha pulsado la barra espaciadora y, en caso de que sea así, saltar fuera del bucle de muestreo.
- ❏ La primera instrucción a añadir tiene que comparar (instrucción `COMP`) la tecla que se acaba de leer (está en `R5`) y el valor que pusimos antes en `R4`.
- ❏ La siguiente instrucción a introducir tiene que saltar si se produjo un cero, porque en ese caso `R4` y `R5` serían iguales, lo que querría decir que se pulsó la barra espaciadora. Pon como etiqueta del salto `fuera`.
- ❏ Sólo nos falta por añadir, justo antes de la directiva `FIN`, un bucle infinito que esté etiquetado con `fuera` para que sea el destino del salto anterior. Añade esta línea antes del `FIN`:

  ```
  fuera: JMP -1
  ```

- ❏ Guarda y ensambla el programa. En el simulador, carga el archivo `3-2Teclado.sim` para que vuelva a la situación inicial. A continuación carga el archivo que acabas de ensamblar.
- ❏ Ejecuta el programa pulsando F9. Comprueba que funciona pulsando las teclas `A` y `B` del teclado del computador teórico. Si todo va bien, se deberían haber escrito ambos caracteres en pantalla.
- ❏ Pulsa ahora la barra espaciadora. Si todo va bien, no se debería escribir en pantalla. Además la CPU debería estar en un bucle infinito ejecutando la instrucción `JMP -1`. Comprueba que ya no se detectan más teclas pulsando la `C` y la `D` y observa cómo quedan acumuladas en el buffer del teclado.

4. Ejercicios adicionales

4.1. Archivos en la carpeta de trabajo

En tu carpeta de trabajo de prácticas deberás tener los archivos de programa `3-2Teclado.sim`, `3-2pulsa.ens` y `3-2letra.ens`.

4.2. Ejercicios

✏ Modifica el programa `3-2letra.ens` para que la letra que se imprime en pantalla sea siempre mayúscula, aunque en el teclado no esté pulsado Bloq M..

✏ Modifica el programa `3-2letra.ens` para que cuando se pulse la tecla 'Q' se salga del programa, sin importar si estaba pulsado Bloq M. o no. Tendrás que comparar con el código SCAN de la tecla, que, al contrario que el código ASCII, no cambia de mayúsculas a minúsculas.

SESIÓN 3

Interrupciones: Mecanismo de funcionamiento

Objetivos

En esta sesión se pretende 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 `.eje`.
- 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". Realiza los siguientes pasos:

- Inicia el simulador de la CPU. En el menú de la Memoria, elige la opción "Configurar".
- Elimina el chip de 32K que cubre las direcciones altas de la memoria (de `8000h` a `FFFFh`), y pon uno nuevo de 16K que ocupe las direcciones bajas de este hueco que has creado. Por tanto, los últimos 16K del mapa de direcciones quedarán vacíos (sin memoria), y este rango estará disponible para interfaces de E/S. Cierra la ventana de configuración de la memoria.
- 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 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 crearlo de nuevo con los datos correctos.

- Presiona el botón [Minimizar] en esa ventana.

Ahora escribe el siguiente programa en el fichero `3-3int1.ens`, haciendo uso de `EDIT` y 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, en el registro `PC` se ha cargado 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.

1

2

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]

3

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

- ❏ Pulsa F8 cuatro veces comprobando tras cada pulsación que efectivamente el registro PC va tomando valores consecutivos comenzando en `0300h` y que el registro `R7` en todo momento tiene el mismo valor.
- ❏ Cuando el registro `IR` contenga el código máquina de la instrucción `MOV R0,R1` el programa habrá finalizado, y el registro `R0` deberá contener el valor `8005h`.

Hasta aqui, 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 los regisros `PC` y `R7`.
- ❏ Pulsa F8. Se ejecuta la 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` del registro de estado ahora está a 1. Esto significa que el procesador admite ser interrumpido.
- ❏ La siguiente instrucción a ejecutar es `MOVL R1, 05h`. Si pulsáramos de nuevo F8, ¿qué valor aparecería en el registro `R1`?[6] ¿Y en el registro`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 la lñinea `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 el registro `R1`?[8] (Como consecuencia de la ejecución de la instrucción `MOVH`.)
- ❏ ¿Qué valor esperarías en el registro `PC` si la ejecución fuese "normal", es decir, sin interrupciones?[9]
- ❏ ¿Qué valor hay en el registro `R7`?[10]

[8]

[9]

[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 el registro `R1`?[11] ¿Era el esperado según tu respuesta anterior?

[11]

- ❏ ¿Qué valor aparece en el registro PC?[12] ¿Era el que habías respondido anteriormente? Después veremos de dónde ha salido este valor.
- ❏ Fíjate en el registro de estado, ¿ha cambiado de valor?[13]
- ❏ ¿Qué valor hay ahora en R7?[14] ¿En cuánto se diferencia del que habías respondido?[15]
- ❏ 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 el registro 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.
- ❏ Y ¿cuál es el número que hay en la pila justo a continuación?[17] ¡Este es el antiguo valor del registro de estado!
- ❏ Observa que la línea INT ya está de nuevo de color negro. 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 la línea INT. Más adelante lo veremos en detalle.
- ❏ Observando el nuevo valor del registro PC y utilizando el desensamblador de la memoria puedes saber cuál será la próxima instrucción que va a ejecutar la CPU. ¿Cuál sería?[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]
- ❏ 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 0000h, donde no hay nada para ejecutar salvo instrucciones 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 extraida.

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

"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, salvo 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 realidad, deberíamos repetir los tres pasos anteriores para cada posible periférico, puesto que cada uno de ellos generará un número de vector diferente y probablemente requerirá también un programa diferente que se ocupe de él.

2.1. La rutina de servicio más simple posible

Se trataría de una rutina "vacía", 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 `B800h`. 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 `0000h`. 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]

20

21

- ❏ Fíjate que esta respuesta explica por qué el registro PC tomaba el valor 0000h tras aceptarse la interrupción. El valor que toma el registro PC al aceptar una interrupción es el que se halla almacenado en el vector de interrupción asociado al periférico que generó dicha interrupción.
- ❏ Modifica el contenido de esa dirección y escribe 50C0h en su lugar. 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.

¡Nuestra rutina ya está "instalada"! A partir de este momento, cuando "Luces" genere una interrupción, el registro PC tomará el valor 50C0h y no 0000h, por lo que ahora en lugar de ejecutar una instrucción NOP se ejecutará la instrucción IRET que hemos colocado allí. ¡Y por tanto la ejecución volverá a nuestro programa que podrá seguir funcionando!

Comprobemos todo esto:

- ❏ Pulsa F8. Se ejecuta la instrucciónSTI y el bit IF del registro de estado se pone a 1, con lo que el procesador está listo para aceptar interrupciones.
- ❏ Pulsa F8. Se ejecuta la instrucción MOVL R1,05h
- ❏ Abre la ventana del periférico "Luces" y pulsa sobre "Generar Interrupción INT". Observa cómo la línea INT se pone de color rojo en la CPU.
- ❏ Si ahora pulsaramos F8, tendría lugar la ejecución de la instrucción actual (MOVH), e inmediatamente, al estar la línea INT activada, tendrían lugar los cambios que ya conocemos (apilación de los registros PC y SR, y modificación de los registros R7 y PC). Veamos si has comprendido los cambios que ocurrirán:
  - ¿Qué valor aparecerá en el registro R7?[22]
  - ¿Qué valor aparecerá en el registro PC?[23] (Recuerda que se trata del valor que hemos guardado en la tabla de vectores, que ya no es 0000h.)
  - ¿Qué dos valores quedarán guardados en la pila?[24]
- ❏ Pulsa F8 y verifica tus respuestas anteriores. En la situación actual, ¿qué instrucción será la próxima en ejecutarse?[25]
- ❏ La instrucción IRET simplemente saca dos datos de la pila (por lo que el registro R7 se incrementa en 2 unidades), y guarda el primero de ellos en el registro PC y el segundo en el registro de estado. Por tanto, al ejecutar esta instrucción
  - ¿Qué valor aparecerá en el registro R7?[26]
  - ¿Qué valor aparecerá en el registro PC?[27]
  - ¿Qué valor tomará el bit IF del registro de ºestado?[28]

22

23

24

25

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 una instrucción `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 en curso.

Naturalmente, una rutina de servicio "de verdad" necesitará hacer algo más que ejecutar la instrucción `IRET`, ya que se entiende que si un periférico ha generado una interrupción es porque tiene datos que deben ser procesados. Por tanto, la rutina debería acceder a los registros de la interfaz para obtener esos datos. Este será el objetivo de la próxima sesión de prácticas.

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 los registros `SR` y `PC`, consultar la tabla de vectores de interrupción , asignar al registro `PC` el nuevo valor obtenido de la tabla y desactivar el bit `IF` del registro de estado).

- 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 la instrucción `STI`. La próxima instrucción sería `MOVL R1,05h`, 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 pasos de obtencón del código máquina de la instrucción a ejecutar e incremento del regisro `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.
- 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 paso 3 que completa la lectura del código máquina de la instrucción. Observa cómo en el registro `IR` aparece la instrucción `MOVL R1,05h`.

- Pulsa de nuevo F7 para ejecutar el cuarto (y último) paso 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.

- Observando 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 generar el paso 1 de la siguiente instrucción de nuestro programa, la unidad de control entrará en una nueva secuencia de pasos, 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 del registro R7. ¿Qué valor hay en el registro TMPS?[30] Observa que es una unidad menos que el valor de R7.

- El valor que has obtenido en el registro 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 del registro de estado en el registro MDR. ¿Qué señales serán necesarias para realizar esta operación?[31]

- Pulsa F7 y verifica tu respuesta.

- Ya tenemos el dato en el registro MDR, falta escribir la dirección en el registro MAR. Como hemos dicho, esta dirección está en TMPS. ¿Qué señales debes activar para copiarla a MAR?[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 el registro 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.

- Mientras la memoria guarda ese dato, la UC se prepara para repetir la operación, es decir, decrementar el contenido del registro R7 en una unidad para enviar a esa dirección de memoria una copia del registro PC. Pulsa F7 y comprueba cómo 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 el contenido del registro PC en el regisro MDR para enviarlo a la memoria, como se hizo antes con SR. Intenta anticipar las señales que son necesarias para realizar esta operación 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

30

31

32

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 ¿qué periférico ha activado `INT`?

El periférico concreto que haya sido deberá responder poniendo en el bus de datos su número de identificación. En nuestro caso, será "Luces" quien responda, escribiendo el valor 3 en el bus de datos. En preparar su respuesta tardará un ciclo de reloj.

- 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 al periférico "Luces" cuando lo conectamos.

  Observa que la unidad de control simplemente toma ese dato (que obtiene en el registro `MDR`) y lo transfiere al registro `MAR`, activando la señal `LEER`. ¿Qué dirección de memoria tratamos de leer?[34] ¿Qué valor hay allí?[35] (Puedes usar el editor hexadecimal de la memoria para consultarlo).
- 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 al registro `PC`. ¿Qué señales crees que activará ahora la unidad de control?[36] Pulsa F7 y verifica tu respuesta.
- Además de las señales que habías respondido, la UC genera la señal `CLI` cuya función es borrar el bit `IF`, y la señal `FIN`, con lo que finaliza la fase de aceptación de la interrupción. El próximo paso que genere la UC será el paso 1 de la siguiente instrucción a ejecutar.
- Pulsa F7 y comprueba cómo, efectivamente, la ejecución vuelve a la normalidad comenzando por el paso 1.
- ¿A qué instrucción corresponderá el código máquina que se reciba en el paso 3?[37]
- Pulsa F7 dos veces y comprueba tu respuesta.
- Puedes seguir pulsando F7 para ver cómo se lleva a cabo la instrucción `IRET` paso a paso, o puedes pulsar F8 si quieres que se ejecute completamente. La función de la instrucción `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.

33

34

35

36

37

4. Ejercicios adicionales

4.1. Archivos en la carpeta de trabajo

En tu carpeta de trabajo 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 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 la istrucción `STI` y seguidamente genera una interrupción en el periférico "Más luces". Pulsa F8 otra vez y comprueba el nuevo valor del registro `PC`.
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 `6000h` 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.

38

✏ Carga el estado `3-3int2.sim`. Pulsa F8. Ahora genera una interrupción desde el periférico "Luces". Pulsa F8 de nuevo. Si todo va bien, el registro `PC` debe tener el valor `50C0h` y la CPU está a punto de ejecutar la instrucción `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.

SESIÓN 4

Interrupciones: Programación de rutinas de servicio

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 `3-2teclado.sim` construido durante la sesión 2 de este bloque de prácticas.

Desarrollo de la práctica

1. Programación en ensamblador de rutinas de servicio

1.1. La rutina más simple posible

En la sesión anterior hemos codificado e instalado "a mano" una rutina de servicio mínima para el vector 3. Esta rutina contenía tan sólo una instrucción `IRET`. La instalación consistía 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 habiamos 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 minima podría ser así:

```
1 PROCEDIMIENTO rutina_minima
2   IRET
3 FINP
```

Este procecimiento formará parte de un fichero `.ens`, en el cual estará tambien 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 `50C0` como era el caso de la sesión anterior, 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, de la misma forma que nosotros habíamos hecho "a mano" en la sesión anterior usando el editor hexadecimal de la memoria.

[1]

Ahora bien, hay un problema. Y es que en la sesión anterior sabíamos que la rutina estaba en la dirección `50C0` 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 `50C0` en `R1` como hemos hecho en el listado anterior. En vez de ello, deberemos usar la directiva `DIRECCION` del compilador, para que él mismo averigue dónde está colocada la rutina.

El código 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 `3-4int1.ens` y ensámblalo para obtener `3-4int1.eje`
- ❏ Abre el simulador de la CPU y carga el archivo `3-2teclado.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 `3-4int1.eje`.
- ❏ Vamos a ejecutarlo instrucción por instrucción. Pulsa F8 cuatro veces y los registros `R0` y `R1` serán inicializados mediante las instruciónes `MOVH` y `MOVL`. ¿Qué valor aparece en `R1`?[2]

  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.)

  3
- ❏ 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".
- ❏ 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.

4

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.

1.2. 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 `0003`, se encenderían las dos bombillas del extremo derecho.

Este registro no puede ser leido 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[1].

[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

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 leido 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 leido (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.

1.3. 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 `3-4int1.ens` con el nombre `3-4int2.ens` y abre éste último con `EDIT`.
- ❏ Ve al punto donde estaba 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 programa que has escrito, para obtener `3-4int2.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.

2. Ejercicios adicionales

2.1. Archivos en la carpeta de trabajo

En tu carpeta de trabajo de prácticas deberás tener los archivos de programa `3-4int1.ens`, `3-4int1.eje`, `3-4int2.ens` y `3-4int2.eje`.

2.2. Ejercicios

✏ Copia `3-4int2.ens` con el nombre `3-4int3.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 ésto.

✏ 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 leido 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?

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