Universidad Zaragoza
1542

PRÁCTICA 5: ENTRADA/SALIDA

INTRODUCCIÓN

Esta práctica consiste en programar un juego de un gusano/snake en ensamblador ARM. Se utilizará para jugar una botonera (teclado-virtual) y una zona de memoria (pantalla). El gusano se tiene que comer las manzanas que van apareciendo de forma aleatoria.

OBJETIVOS Y COMPETENCIAS A ADQUIRIR:

- Poner en práctica la gestión de periféricos en bajo nivel
- Consolidar conceptos relativos entrada y salida e interrupciones.

TRABAJO PREVIO:

Para la realización de esta práctica conviene estudiar y comprender los siguientes aspectos:

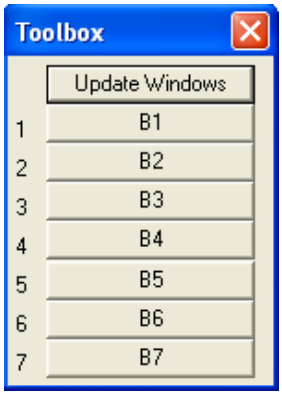
- Gestión de excepciones tipo IRQ en la arquitectura ARM v4.
- Funcionamiento del sistema de E/S del microcontrolador NXP LPC 2105 configurado para AOC1. Esta configuración básica es:
  - Procesador: Flags I=0, F=1, modo supervisor, pilas inicializadas (modos svc, irq).
  - Controlador de interrupciones (VIC): Todas las peticiones de interrupción quedan configuradas como IRQ vectorizadas manteniendo el orden de prioridades.
  - Funcionamiento del Timer 0 (100 interrupciones por segundo a través de la IRQ4 del VIC).
- Cualquier duda sobre el funcionamiento de este sistema de E/S se puede mirar en el manual de usuario del microcontrolador NXP LPC 2105 (disponible en moodle/hendrix).

CARACTERISTICAS DEL PROYECTO KEIL:

En moodle/hendrix podéis descargar los ficheros necesarios para esta práctica. Consisten en un proyecto Keil configurado para la realización de prácticas de E/S. Al abrir el proyecto veréis que incluye dos ficheros fuente en ensamblador:

- **Startup.s** Contiene el código de inicialización del microcontrolador, VIC y periféricos. No debéis modificar nada de este fichero ni hace falta entenderlo.
- **prac5.s** Contiene las directivas necesarias para que la compilación y enlace con Startup.s funcione correctamente. Aquí debéis poner vuestro código de la práctica.
- **rand.s** Contiene el código para la generación de números pseudo-aleatorios.

Durante el desarrollo, depuración y ejecución de la práctica, se recomienda poner un punto de parada (breakpoint) en la primera instrucción de vuestro código en el fichero fuente prac5.s. De esta forma, con el comando (Run = F5) se ejecuta todo el código de inicialización de Startup.s seguido y el simulador se para en la primera instrucción de vuestro programa. A partir de ahí podéis depurar vuestro programa como queráis.

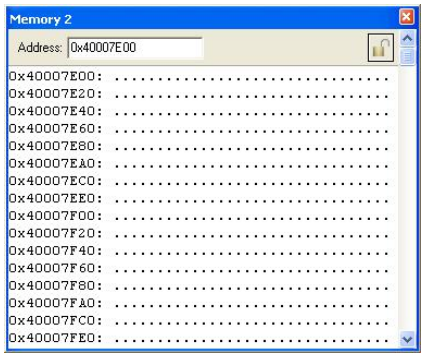


En el proyecto que se adjunta se han definido unos botones (ver figura adjunta) visibles cuando el simulador está en modo depuración (debug session). Esta ventana aparece y desaparece seleccionando el comando View->Toolbox Window ( ). Cada vez que se pulsa un botón numerado del 1 al 7 ocurre lo siguiente:

- Se almacena en la dirección 0xE001C008 (equivalente a un registro de datos) el código del botón pulsado (número de 1 a 7).
- Se genera una petición de interrupción por la IRQ1 del VIC. Esta petición se mantiene hasta que el programador la baje activando el bit 1 del registro VICSoftIntClear (0xFFFFF01C).

- El programador, después de leer el código del botón pulsado, debe escribir un 1 en el bit 2 de la dirección 0xE001C018 (equivalente a un registro de control) para indicar al controlador que se ha leído el código de botón pulsado

Estos 7 botones constituyen una versión simplificada de un teclado. Se emplearán en la práctica para introducir comandos y controlar la ejecución del programa. Este periférico virtual habrá que gestionarlo con sincronización por interrupción.



Para observar las actividades del programa, se gestionará una zona de la memoria RAM (direcciones 0x40007E00 a 0x40007FFF) como si fuese una pantalla de texto. Para ver el contenido de esa pantalla, conviene añadir (en debug session) una nueva vista de memoria (View->Memory Windows->Memory 2) con la geometría de la figura adjunta (formato ASCII, 16 filas, 32 columnas a partir de la dirección 0x40007E00).

Para ver el estado del VIC debéis seleccionar el comando Peripherals -> Vectored Interrupt Controller (en debug session). Se recomienda no cambiar la configuración del VIC. Si por error se cambia la configuración del VIC, volver a ejecutar el código de Startup.s pulsando el botón de reset ( ).

Esta práctica incluye dos opciones una básica, cuya nota máxima alcanzable es un 7 y segunda opción más avanzada, que incorpora pequeñas variantes, algo más complejas y que permite alcanzar la máxima calificación.

OPCION 1 BÁSICA: **(NOTA MÁXIMA 7)**

Abrir el proyecto “pract5” adjunto a la práctica. Diseñar, codificar e implementar un programa en ensamblador de ARMv4 que borre la pantalla de texto (el código ASCII del espacio en blanco es 32) y dibuje un carácter ‘#’ (código ASCII 35) en el centro de la pantalla de texto (fila 8, columna 16). Inicialmente la marca está parada. El carácter ‘#’ se moverá cuando se pulsen los botones de dirección a ritmo inicial de una posición cada 0,08 segundos (según el reloj de simulación). Si una marca llega a al extremo derecho (izquierdo) de la pantalla de texto, debe aparecer por el extremo izquierdo

(derecho) continuando el movimiento a la misma velocidad, con la misma dirección y en la misma fila. Si una marca llega a al extremo superior (inferior) de la pantalla de texto, debe aparecer por el extremo inferior (superior) continuando el movimiento a la misma velocidad, con la misma dirección y en la misma columna. El programa debe terminar cuando se pulse el botón B1.

Además, cada 5,12 segundos aparecerá un carácter '$' (código ASCII 36) en una posición aleatoria de la pantalla y la velocidad del juego se incrementará (dividiendo max por 2). Si el carácter '#' pasa por encima del '$' la velocidad del juego se decrementará (duplicando max).

La función de los botones debe ser (se pueden cambiar las etiquetas en el fichero prac5.ini para hacerlos más legible):

B1: Fin de programa
B2: Izquierda
B3: Derecha
B4: Arriba
B5: Abajo
B6: + Velocidad. Duplicar la velocidad de movimiento (máximo 1 movimiento cada 0,01 segundos).
B7: - Velocidad. Dividir por dos la velocidad movimiento (mínimo 1 movimiento cada 2,56 segundos).

La gestión de los botones y del timer debe realizarse con sincronización por interrupción y transferencia programada.

El esquema del programa es el siguiente:

```
            AREA datos, DATA
reloj       DCD 0             ;contador de centesimas de segundo
max         DCD 16            ;velocidad de mov. caracter '#' (en centesimas s.)
cont        DCD 0             ;instante siguiente movimiento caracter '#'
dirx        DCB 0             ;direccion mov. caracter '#' (-1 izda.,0 stop,1 der.)
diry        DCB 0             ;direccion mov. caracter '#' (-1 arriba,0 stop,1 abajo)
fin         DCB 0             ;indicador fin de programa (si vale 1)

            AREA codigo, CODE
            EXPORT inicio           ;etiqueta enlace con Startup.s
inicio
            ;programar @IRQ1 -> RSI_boton
            ;programar @IRQ4 -> RSI_reloj
            ;activar IRQ1,IRQ4

bucle       ;si toca añadir '$'
            ;  calcular instante siguiente aparición '$'
            ;  generar posicion aleatoria '$'
            ;  dibujar nuevo '$'
            ;  max=max/2;
            ;si toca mover '#'
            ;  calcular instante siguiente movimiento
            ;  borrar '#' anterior
            ;  calcular nueva posicion '#'
            ;  dibujar nuevo '#'
            ;si fin=0 salto a bucle

            ;desactivar IRQ1,IRQ4
            ;desactivar RSI_reloj
            ;desactivar RSI_boton
bfin        b bfin

RSI_reloj   ;Rutina de servicio a la interrupcion IRQ4 (timer 0)
            ;Cada 0,01 s. llega una peticion de interrupcion

RSI_boton   ;Rutina de servicio a la interrupcion IRQ1 (SW interrupt)
            ;al pulsar cada boton llega peticion de interrupcion IRQ1
            END
```

Una vez depurado vuestro programa, **para observar correctamente el movimiento del carácter '#', activad la opción View -> Periodic Window Update (debug session)** y luego ejecutad todo el programa seguido (Run = F5). De esta forma se observa perfectamente el movimiento de las marcas de bits activos.

AYUDA: NÚMEROS ALEATORIOS

Para la generación de números pseudo-aleatorios (imprescindibles cuando nos ponemos a jugar) podéis utilizar las subrutinas `rand` y `srand` del fichero fuente rand.s.

- **srand** sirve para inicializar la secuencia de números pseudo-aleatorios mediante una semilla (número natural de 32 bits). Esta subrutina **no devuelve ningún resultado** y tiene **un único parámetro** (la semilla) que se pasa **por valor en la pila**. Debéis invocar a esta SBR una única vez al comienzo del programa. Cambiando la semilla se obtiene una secuencia distinta de números pseudo-aleatorios.
- **rand** sirve para generar el siguiente número pseudo-aleatorio a partir del anterior. Esta subrutina **no tiene parámetros** y como **resultado** devuelve **en la pila** un **número pseudo-aleatorio de 31 bits**. Debéis invocar a esta SBR cada vez que necesitéis un número pseudo-aleatorio para vuestro programa.

OPCION 2 AVANZADA: **(NOTA MÁXIMA 10)**

Igual que la opción 1 pero cambiando el carácter '#' por un gusano formado por k caracteres y '$' por un dígito entre '1' y '9'. Inicialmente k=1 y cada vez que el gusano pase por encima de un dígito se aumenta la longitud del gusano en el valor del dígito (k=k+dígito). El gusano se mueve como en la opción 1, pero hay que controlar que no se atropelle a si mismo. Si el primer carácter del gusano (cabeza) llega a una posición ya ocupada por el propio gusano, el movimiento se detiene y se hará parpadear la cabeza durante 0,64 segundos (4 parpadeos). Después se vuelve a empezar la ejecución con otro gusano. Para evitar auto-atropellos, los dígitos no podrán ponerse en una posición ya ocupada por el gusano.

EVALUACIÓN:

La evaluación de esta práctica se realizará durante la sesión de entrega de prácticas que se convocará en fecha próxima al examen de cada convocatoria (posible el mismo día del examen). Debe mostrarse el funcionamiento correcto de la práctica y responder a las preguntas que os hagan los profesores al respecto.