

## PRÁCTICA 5: ENTRADA/SALIDA

### 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:

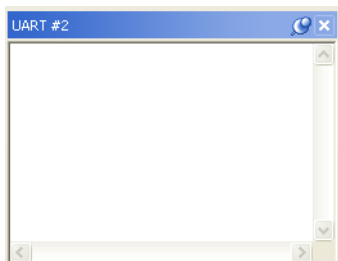
- Gestión de interrupciones en la arquitectura ARM v4.
- Funcionamiento del sistema de E/S del microcontrolador NXP LPC 2105 configurado para AOC1. Esta configuración básica es:
  - Procesador: Flags I=0, F=1, modo supervisor, pilas inicializadas (modos svc, irq).
  - Controlador de interrupciones (VIC): 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).
  - Conexión del teclado a través de la UART 1 (IRQ7 del VIC).
- Cualquier duda sobre el funcionamiento de este sistema de E/S se puede consultar en el manual de usuario del microcontrolador NXP LPC 2105 (disponible en moodle/hendrix).

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 tres ficheros fuente en ensamblador:

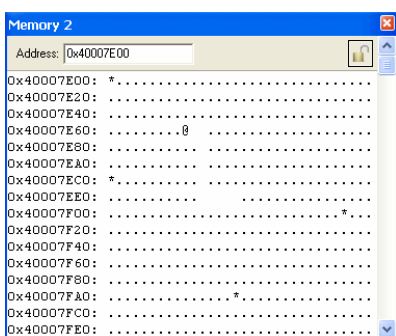
- **Startup.s** Contiene el código de inicialización del microcontrolador, VIC y periféricos. No debéis modificar nada de este fichero ni hace falta entenderlo.
- **prac5.s** Contiene las directivas necesarias para que la compilación y enlace con Startup.s funcione correctamente. Aquí debéis poner vuestro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


Para el correcto funcionamiento del teclado, es necesario añadir (en debug session) una nueva ventana (View->Serial Windows->UART#2). Esta ventana se corresponde con la UART1 (Universal Asynchronous Receiver/Transmitter) del LPC 2105 y hay que dejarla como ventana activa. Cualquier tecla pulsada en el teclado será registrada por la UART a través de esa ventana y ocurre lo siguiente:



- Se almacena en el puerto 0xE0010000 (equivalente a un registro de datos) el código ASCII de la letra correspondiente a la tecla pulsada (mayúsculas y minúsculas son distintas).
- Se genera una petición de interrupción por la IRQ7 del VIC. Esta petición se mantiene hasta que el programador lea el dato del puerto 0xE0010000.



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 (numeradas de 0 a 15) y 32 columnas (numeradas de 0 a 31) 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 primera básica, cuya nota máxima alcanzable es un 7 y una segunda opción más avanzada, que incorpora variantes más complejas y permite alcanzar la máxima calificación.

### OPCION 1: (NOTA MÁXIMA 7)

Abrir el proyecto "pract5" adjunto a la práctica. Diseñar, codificar e implementar en ensamblador ARMv4 un juego. En este juego, el jugador toma el control de una hambrienta arropa '@' (ASCII 64) que deberá saciar su gula con todos los puntos '.' (ASCII 46) que caben en la pantalla.

La pantalla de juego se puede observar en la figura ('Memory 2'). Inicialmente, nuestra arropa aparecerá en uno de los 4 puntos centrales de la pantalla y el resto de las posiciones de la pantalla estarán inicializadas con tokens '.' (ASCII 46). La dinámica del juego consistirá en guiar a la arropa para dar buena cuenta de todos y cada uno de los tokens. Los tokens comidos se sustituyen por espacios en blanco (ASCII 32). Para controlar el movimiento de la arropa utilizaremos las teclas: 'A' (ASCII 65 ó 97) izquierda, 'D' (ASCII 68 ó 100) derecha, 'S' (ASCII 83 ó 115) abajo y 'W' (ASCII 87 ó 119) arriba. Nuestra arropa será perseguida a lo largo y ancho de la pantalla por 4 asteriscos '\*' (ASCII 42) que intentarán comerla. El momento y lugar de aparición de los asteriscos puede realizarse haciendo uso del generador de números pseudo-aleatorios. La velocidad de movimiento de la arropa y los asteriscos se controla con la tecla '+' (ASCII 43) para duplicar la velocidad de movimiento (máximo 1 movimiento cada 0,01 segundos) y tecla '-' (ASCII 45) para dividir por dos la velocidad de movimiento (mínimo 1 movimiento cada 1,28 segundos).

La partida termina cuando la arroba consume todos los tokens o es alcanzada por un asterisco por tercera vez. El jugador puede finalizar el juego en cualquier momento pulsando la tecla 'Q'.

La gestión del teclado y del timer debe realizarse con sincronización por interrupción y transferencia programada. Cualquier otro aspecto no especificado en la memoria queda a elección del alumno.

El esquema del programa es el siguiente:

```

AREA datos, DATA
reloj      DCD      0      ;contador de centesimas de segundo
max        DCD      8      ;velocidad de movimiento (en centesimas s.)
cont       DCD      0      ;instante siguiente movimiento
dirx       DCB      0      ;mov. horizontal arroba (-1 izda.,0 stop,1 der.)
diry       DCB      0      ;mov. vertical arroba (-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 @IRQ4 -> RSI_reloj
;programar @IRQ7 -> RSI_teclado
;activar IRQ4,IRQ7

;dibujar pantalla inicial
;para cada elemento movil
; si toca mover elemento
;   calcular instante siguiente movimiento
;   borrar elemento anterior
;   calcular nueva posicion (dirx,diry) elemento
;   dibujar nuevo elemento
;   fin
;si fin=0 salto a bucle

;desactivar IRQ4,IRQ7
;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_teclado ;Rutina de servicio a la interrupcion IRQ7 (teclado)
            ;al pulsar cada tecla llega peticion de interrupcion IRQ7
            END

```

Una vez depurado vuestro programa, **para observar correctamente el movimiento del juego, 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:

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 y una partida diferente.
- **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: (NOTA MÁXIMA 10)

Añadir a la opción 1 al menos dos de las siguientes características:

- Reservar la primera fila de la pantalla para marcadores de puntos y vidas.
- Incluir un token con significado especial, destruir todos los asteriscos durante un intervalo aleatorio de tiempo.
- Incluir un token con significado especial, aumentar el área de absorción de @ a un cuadrado de 5x5 caracteres.
- Dar capacidad de disparo a la arroba
- Implementar velocidades diferentes para arroba y asteriscos.
- Cualquier otra característica que creáis conveniente para mejorar la experiencia de juego y de complejidad similar a las anteriores.

#### 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.