GRADO EN INGENIERÍA INFORMÁTICA EN SISTEMA DE INFORMACIÓN

Fundamentos de Computadores

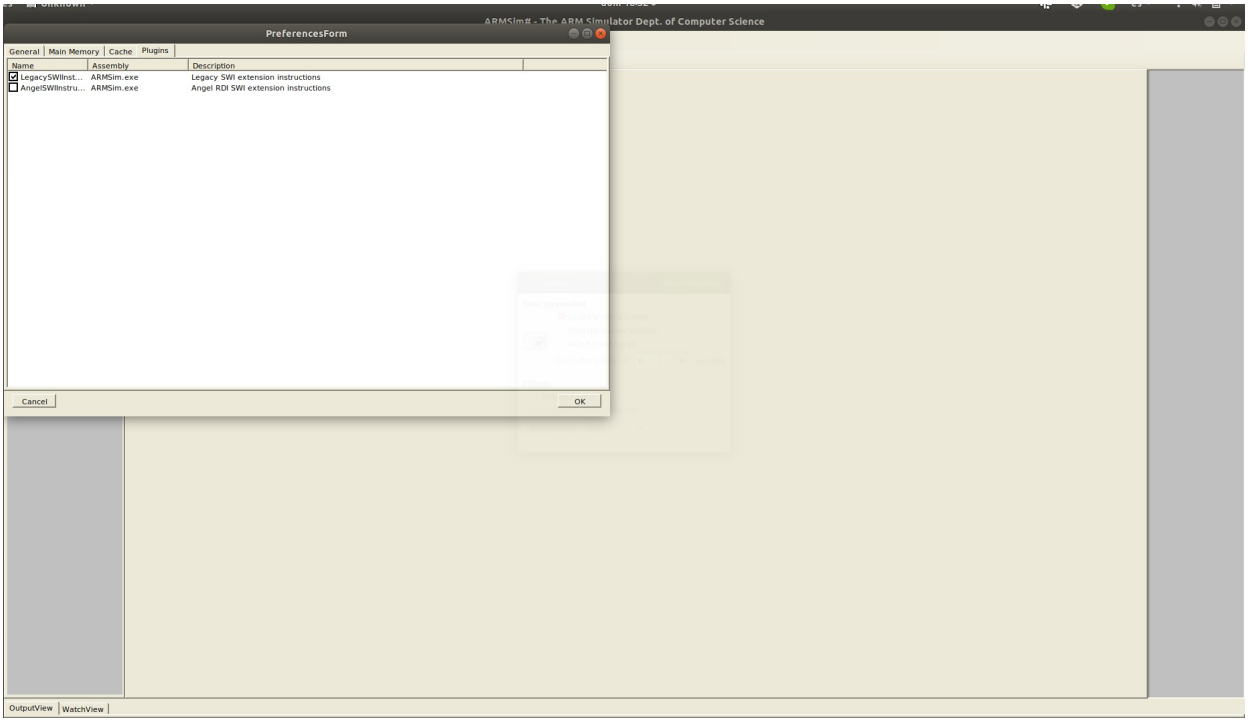
COMPONENTES DEL GRUPO DE TRABAJO
- Rodrigo Bautista Hernández
- Ricardo Álamo Fernández
-

EPD 5

Dentro de las excepciones, existe un tipo que podemos invocar a nuestra conveniencia; estas son las interrupciones software. El motivo de usar interrupciones software en lugar de subrutinas es que al ejecutarlas, pasamos a un modo más privilegiado de funcionamiento dentro de la arquitectura ARM, llamado modo supervisor (svc). En este modo, se pueden hacer algunas cosas que no estarían permitidos en modo usuario, como el acceso a dispositivos. Es por ello que este tipo de interrupciones software son implementadas en el sistema operativo y pueden ser usadas por el código usuario para este tipo de tareas (acceso a consola, archivos, etc.).

El código que ejecutamos en ARMSim corre por lo general en modo usuario. No obstante, el simulador ARMSim pone a nuestra disposición un pequeño sistema operativo al que podemos dotar de funcionalidades por medio de Plugins. En particular, vamos a usar el conjunto de excepciones software Legacy SWI, que tiene implementadas las siguientes interrupciones software que podemos emplear para el uso de operaciones básicas de E/S con diversos dispositivos e implementar un temporizador sencillo. Para su activación, habilite el “Checkbox”:

File → Prefences → Plugins → LegaciSWIInstructions



La siguiente tabla muestra las rutinas disponibles en el conjunto de instrucciones LegacySWIInstructions para el uso de la E/S estándar (consola y ficheros) y temporizadores. Tenga en cuenta que a las interrupciones software se les pasa parámetros de forma similar a como se hace en las subrutinas. Por tanto, previamente a llamar a la interrupción software deberemos preparar los registros adecuados.

Opcode	Description and Action	Inputs	Outputs
swi 0x00	Display Character on Stdout	r0: the character	
swi 0x02	Display String on Stdout	r0: address of the string	
swi 0x11	Halt execution		
swi 0x12	Allocate Block of Memory on Heap	r0: block size in bytes	
swi 0x13	Deallocate All Heap Blocks		
swi 0x66	Open File	r0: address of filename r1: mode (0-in, 1-write)	r0: file handle (-1=error)
swi 0x68	Close File	r0: File Handle (FH)	
swi 0x69	Write String to a File (or to Stdout)	r0: FH or 1(stdout) r1: address of string*	
swi 0x6a	Read a String from a File	r0: FH r1: destination address r2: max bytes	r0: number of bytes stored
swi 0x6b	Write Integer to a File	r0: FH r1: integer	
swi 0x6c	Read Integer from a File	r0: file handle	r0: the integer
swi 0x6d	Get the current time (ticks)		r0: ticks (millisecond)

Además, existen tres descriptores de ficheros (file handles) especiales que se usan para la salida y entrada estándar:

Stdin = 0 Stdout = 1 Stderr = 2

C1 Primera prueba de uso de dispositivos en ARMSim

Descargue el programa EPD5_C1.s del aula virtual e intente ejecutarlo con ayuda de la utilidad ARMSim. Compruebe que se ejecute correctamente y muestre una captura de la salida estándar después del mismo.

ARCHIVOS RESULTADO: EPD5_C1.S, CAPTURAS_PANTALLA

C2 Carga de un archivo de texto y muestra del contenido en pantalla

En este ejercicio, se quiere realizar un programa que vaya cargando un archivo de texto línea a línea. Dichas líneas se deben representar por la salida estándar. El nombre del archivo a cargar tiene estará almacenado la variable “File”. Puede usar el archivo EPD5_C2.s del aula virtual para empezar.

ARCHIVOS RESULTADO: EPD5_C2.S, CAPTURAS_PANTALLA

C2:

```
/* EPD5, C2: carga un fichero de texto linea a linea y lo
            muestra por pantalla */
.equ SWI_PrChr, 0x00
.equ SWI_PrStr, 0x02
 @ Write an ASCII char to Stdout
.equ SWI_OpenFile, 0x66 @ Open a text file
.equ SWI_CloseFile, 0x68 @ Close a text file
.equ SWI_RdStr, 0x6a @ Read a String from File
.equ SWI_WrStr, 0x69 @ Write a Sting
 .equ Stdout, 1  @ Set output mode to be Output View
 .equ Stdin, 0
 .equ SWI_Exit, 0x11 @ Stop execution

.data
Line:     .space 400 @ Buffer para leer la linea
```

```
 .asciz "\n"
NewL:   .ascii "\n"
EOL:    .asciz "\n"
Blank:  .ascii " "
Error_msg: .asciz "Error while reading the file\n"
File:  .asciz "textoc1.txt"

.align

.text
.global main
main:   ldr r0,=File
        mov r1,#0
        swi SWI_OpenFile
        cmp r0,#-1
        beq show_error
        stmdb sp!,{r4,r7}
        mov r7,r0
leer:   mov r0,r7
        ldr r1,=Line
        mov r2,#100
        swi SWI_RdStr
        ldrb r4,[r1]
        cmp r4,#'*'        /* Marcamos el fin del documento con un * */
        beq end
        mov r0,#1
        swi SWI_WrStr
        ldr r1,=NewL
        mov r0,#1
        swi SWI_WrStr
        b leer

end:    ldmia sp!,{r4,r7}
  swi SWI_Exit @ stop executing: end of program

show_error:
  /* Print message 2*/
  ldr r0,=Error_msg
  swi #SWI_PrStr
  b end


.end
```

C3 Carga y escritura de matrices con ARMSim

Se desarrolla un programa que cargue dos matrices cuadradas desde fichero y almacene su suma en un tercer fichero. Para ello se os proporciona el programa EPD5_C3.s que tenéis que completar. El formato de los ficheros es:

<tamaño(n)>
<fila 1>
….
<fila n>
Ejemplo:

```
---------------------------------------
3
1 2 3
4 5 6
7 8 9
---------------------------------------
```

NOTA: Se considera que las matrices se almacenarán en memoria en formato word.
ARCHIVOS RESULTADO: EPD5_C3.S, CAPTURAS_PANTALLA

C3:

```
/* EPD5, C3: programa que suma dos matrices obtenidas desde fichero y almacena su suma en un tercer fichero */
.equ SWI_PrChr, 0x00  @ Write an ASCII char to Stdout
.equ SWI_PrStr, 0x02   @ Write a null-ending string
.equ SWI_OpenFile, 0x66         @ Open a text file
.equ SWI_CloseFile, 0x68        @ Close a text file
.equ SWI_PrInt, 0x6b    @ Write an Integer
.equ SWI_RdInt, 0x6c   @ Read an Integer
.equ SWI_WrStr, 0x69   @ Write a string to file

@ Standard file handles
.equ Stdout, 1
.equ Stdin, 0
.equ Stderr, 2

.equ SWI_Exit, 0x11 @ Stop execution

.global _main
.text
_main:
  /* prints loading message */
  ldr r0, =Message1 /* load address of Message1 */
  swi SWI_PrStr /* display message to Stdout */
  ldr r0, =First /* Load the first filename */
  swi SWI_PrStr
  ldr r0, =EOL /* end of line */
  swi SWI_PrStr
  /* Calls to load matrix */
  ldr r0, =First
  ldr r1, =matrix1
  bl load_matrix
  blt show_first_error
  mov r4, r0 /* r4 is the size of matrix 1 */

  /* Carga de la segunda matriz */
  ldr r0, =Message1 /* load address of Message1 */
  swi SWI_PrStr /* display message to Stdout */
  ldr r0, =Second /* Load the first filename */
  swi SWI_PrStr
  ldr r0, =EOL /* end of line */
  swi SWI_PrStr
  /* Calls to load matrix */
  ldr r0, =Second
  ldr r1, =matrix2
  bl load_matrix
  blt show_second_error
  /* End of removal*/

  /* Comparamos para ver si tienen la misma dimension */
  cmp r0, r4
  bne show_diff_size_error

  /* Call to subroutine for adding two matrices */
  mov r0,r4
```

```
  ldr r1, =matrix1
  ldr r2, =matrix2
  ldr r3, =matrixsum
  bl add_matrix

  /* Call to subroutine for saving a matrix */
  ldr r0, =SumFile
  ldr r1, =matrixsum
  mov r2, r4
  bl save_matrix
  blt show_third_error
  b end

show_first_error:
  ldr r0, =Error_msg1
  swi SWI_PrStr
  b end
show_second_error:
  ldr r0, =Error_msg2
  swi SWI_PrStr
  b end
show_third_error:
  ldr r0, =Error_msg3
  swi SWI_PrStr
  b end
show_diff_size_error:
  ldr r0, =Error_msg_diff_size
  swi SWI_PrStr
  b end

end:
  swi SWI_Exit @ stop executing: end of program

/* Opens a file and reads a matrix from file */
/* Inputs: r0 = filename; r1 = pointer to matrix */
/* Output: r0 = Size of the matrix. A negative value if an error occurred */
load_matrix:
  stmdb sp!, {r4-r6}
  mov r6, r1 /*Matrix pointer to r6 */
  /* Open file */
  mov r1, #0 /* Open for read */
  swi SWI_OpenFile
  cmp r0,#0
  blt lm_error
  mov r4, r0 /* Store the file handle */
  swi SWI_RdInt
  bcs lm_error_and_close
  mov r5,r0   /*save the size in r5*/
  mov r1,#0   /* r1 will be the row counter */

lm_row_loop:
  cmp r5,r1
  moveq r0,#0
  beq lm_close_end

  mov r2,#0 /* Column counter */
lm_column_loop:
  cmp r5,r2
  beq lm_end_column
```

```
  /* Get the number */
  mov r0, r4 /* File handle to r0*/
  swi SWI_RdInt
  bcs lm_error_and_close
  str r0, [r6], #4 /* Store the number and actualize pointer */

  add r2,r2, #1   /* Actualize col counter */
  b lm_column_loop
lm_end_column:
  add r1,r1,#1
  b lm_row_loop

lm_error:
  mov r0,#-1
  b lm_end

lm_error_and_close:
  mov r0,r4
  swi SWI_CloseFile
  mov r0,#-1
  b lm_end

lm_close_end:
  mov r0,r4
  swi SWI_CloseFile
  bcs lm_error
  mov r0,r5

lm_end:
  ldmia sp!, {r4-r6}
  bx lr

/* Completad las funciones save matrix y add matrix */


/* Opens a file and writes the matrix to file */
/* Inputs: r0 = filename; r1 = pointer to matrix r2 = tam */
/* Output: r0 = Size of the matrix. A negative value if an error occurred */
save_matrix: stmdb sp!,{r4-r7}
                    mov r4,r1               /* Se guarda el pointer de matrixsum en r4 */
                    mov r1,#1
                    swi SWI_OpenFile        /* Abre la file en modo write */
                    bcs show_third_error
                    mov r5,r0               /* Guardar FH */
                    mov r1,r2               /* tam -> r1 */
                    swi SWI_PrInt           /* Escribir tam en file */
                    ldr r1,=EOL
                    swi SWI_WrStr           /* EOL */
                    mov r6,#0               /* r6 va a ser contador del loop de fila */

sm_loop_fila:       cmp r6,r2
                    beq sm_close
                    mov r7,#0               /* r7 va a ser contador del loop de columna */

sm_loop_col:        cmp r7,r2
                    beq sm_fin_col
                    mov r0,r5               /* FH */
                    ldr r1,[r4],#4          /* Número de la matriz -> r1, actualiza el pointer */
                    swi SWI_PrInt
                    ldr r1,=Blank
```

```
                swi SWI_WrStr       /* Espacio entre números */
                add r7,r7,#1
                b sm_loop_col

sm_fin_col:     add r6,r6,#1
                ldr r1,=EOL
                swi SWI_WrStr       /* EOL entre filas de la matriz */
                b sm_loop_fila

sm_error:       mov r0,#-1
                b show_third_error

sm_close:       mov r0,r5
                swi SWI_CloseFile
                bcs sm_error

sm_end:         ldmia sp!,{r4-r7}
                bx lr

/* Sums two matrices and stores the result in another */
/* Inputs r0: tam; r1 = Matrix 1; r2 Matrix 2; r3 Matrix 3 */
/* Outputs: none */
add_matrix:     stmdb sp!,{r4-r8}
                mul r5,r0,r0        /* r5 = (r0)^2, r5 = número de elementos en la matriz */
                mov r4,#0           /* r4 va a ser contador del loop */
am_loop:        cmp r4,r5
                beq am_end
                ldr r7,[r1],#4
                ldr r8,[r2],#4
                add r6,r7,r8
                str r6,[r3],#4
                add r4,r4,#1
                b am_loop

am_end:         ldmia sp!,{r4,r6}
        bx lr

/* Parte de datos. Aqui se incluyen los nombres de los archivos, mensajes y espacio para matrices */
.data
First: .asciz "Matrix1.txt"
Second:    .asciz "Matrix2.txt"
SumFile:   .asciz "MatrixSum.txt"
Message1: .asciz "Opening file: "
EOL:
 .asciz "\n"
Blank:
 .asciz " "
Error_msg1: .asciz "Error while reading Matrix 1\n"
Error_msg2: .asciz "Error while reading Matrix 2\n"
Error_msg3: .asciz "Error while writing Matrix\n"
Error_msg_diff_size: .asciz "Error: matrices have different size\n"

.align
matrix1: .space 400
matrix2: .space 400
matrixsum: .space 400

.end
```

C4 Cronómetro por salida estándar

a) Se desea hacer un cronómetro que imprima los minutos y segundos desde su ejecución. Para la visualización, el prográma irá imprimiendo una línea cada vez con el formato:

m:s

Donde m es el número de minutos y s de segundos.

Por tanto, el cronómetro debe actualizarse a una frecuencia de 1 Hz. Para realizar una espera precisa, contamos con la instrucción 0x6d, que indica el tiempo transcurrido en ticks (cada tick equivale a un milisegundo) desde el inicio de la ejecución. Podemos ver el tiempo en milisegundos entre dos instantes si restamos el resultado obtenido en el segundo instante con respecto a un primer instante. En el aula virtual tenéis el fichero EPD5_C4.s, donde tenéis un ejemplo de uso de los ticks que podéis modificar para resolver este ejercicio.

Pistas: podemos usar las funciones que representan números y cadenas por un fichero (en este caso el handler es 1, stdout). El carácter de salto de línea tiene código ASCII 10.

b) Implementación de la cuenta atrás: leed un carácter de la entrada estándar antes de empezar el contador para establecer el número de minutos de la cuenta atrás. Una vez leído dicho carácter, el contador empezará a contar descendentemente desde ese número de minutos hasta llegar a cero.
ARCHIVOS RESULTADO: EPD5_C4a.S, EPD5_C4b.S, CAPTURAS_PANTALLA.

En este ejercicio tuvimos un problema, al utilizar las órdenes swi 0x00 para escribir los ‘:’, swi 0x02 para escribir los EOL, pero la orden swi 0x6b para escribir los int, el Armsim nos escribía los ‘:’ y los EOL en la pestaña “Console”, pero los minutos y segundos en la pestaña “Stdout”. Por lo que se veían en pestañas diferentes los Outputs del programa. Por esta razón, hemos utilizado la orden swi 0x69 para imprimir ‘ : ’ y los EOL, de manera que el output del programa se ve en la pestaña “Stdout” del Armsim, con el formato:
m : s

C4.a)

```
/* EPD5 C4 Muestra una cadena cada segundo por Stdout */
.equ SWI_WrStr, 0x69 @ Write a string
.equ SWI_PrInt, 0x6b @ Write an Integer to a File Handle
.equ Stdout, 1 @ Handle of Stdout
.equ SWI_Exit, 0x11 @ Stop execution
.equ SWI_GetTicks, 0x6d @ Get current time

.data

puntos: .asciz " : "

EOL:    .asciz " \n"
.align

.global _main
.text
```

```
_main:  mov r5,#0               /* r5 contador de minutos */
        mov r6,#0               /* r6 contador de segundos */

restart_counter:
        cmp r6,#60              /* Si segundos = 60 -> Va a "min" */
        bleq min
        mov r0,#1
        mov r1,r5               /* Minutos -> r1 */
        swi SWI_PrInt           /* Imprime minutos */
        mov r0,#1
        ldr r1,=puntos
        swi SWI_WrStr           /* Imprime ':' */
        mov r0,#1
        mov r1,r6               /* Segundos -> r1 */
        swi SWI_PrInt           /* Imprime segundos */
        ldr r1,=EOL
        swi SWI_WrStr           /* EOL */

        swi #SWI_GetTicks
        mov r4, r0          @ R4 guarda los tics
buc:  swi #SWI_GetTicks
        sub r0, r0, r4
        cmp r0, #1000
        addgt r6,r6,#1
        bgt restart_counter
        b buc

min:    mov r6,#0
        add r5,r5,#1
        bx lr

end:
  swi SWI_Exit @ stop executing: end of program

.end
```

C4.b)

```
/* EPD5 C4 Muestra una cadena cada segundo por Stdout */
.equ SWI_WrStr, 0x69 @ Write a string
.equ SWI_PrInt, 0x6b @ Write an Integer to a File Handle
.equ SWI_RdInt, 0x6c @ Read an Integer
.equ Stdout, 1 @ Handle of Stdout
.equ Stdin, 0 @ Handle of Stdin
.equ SWI_Exit, 0x11 @ Stop execution
.equ SWI_GetTicks, 0x6d @ Get current time

.data
mens:   .asciz "Escriba un número de minutos: "
fin:    .asciz "Fin."
puntos: .asciz " : "
EOL:    .asciz " \n"
.align

.global _main
.text
_main:  ldr r1,=mens
        mov r0,#1
        swi SWI_WrStr
```

```
        mov r0,#0
        swi SWI_RdInt
        mov r5,r0               /* r5 contador de minutos */
        mov r6,#0               /* r6 contador de segundos */

restart_counter:
        cmp r5,#0
        bleq seg
        cmp r6,#0               /* Si segundos = 60 -> Va a "min" */
        bleq min
        mov r0,#1
        mov r1,r5               /* Minutos -> r1 */
        swi SWI_PrInt           /* Imprime minutos */
        mov r0,#1
        ldr r1,=puntos
        swi SWI_WrStr           /* Imprime ':' */
        mov r0,#1
        mov r1,r6               /* Segundos -> r1 */
        swi SWI_PrInt           /* Imprime segundos */
        ldr r1,=EOL
        swi SWI_WrStr           /* EOL */

        swi #SWI_GetTicks
        mov r4, r0          @ R4 guarda los tics
buc:  swi #SWI_GetTicks
        sub r0, r0, r4
        cmp r0, #1000
        subgt r6,r6,#1
        bgt restart_counter
        b buc

min:    mov r6,#59
        sub r5,r5,#1
        bx lr

seg:    cmp r6,#0
        beq end
        bx lr

end:
        mov r0,#1
        mov r1,r5               /* Minutos -> r1 */
        swi SWI_PrInt           /* Imprime minutos */
        mov r0,#1
        ldr r1,=puntos
        swi SWI_WrStr           /* Imprime ':' */
        mov r0,#1
        mov r1,r6               /* Segundos -> r1 */
        swi SWI_PrInt           /* Imprime segundos */
        ldr r1,=EOL
        swi SWI_WrStr           /* EOL */
        ldr r1,=fin
        mov r0,#1
        swi SWI_WrStr
  swi SWI_Exit @ stop executing: end of program

.end
```