Ayuda de ensamblador MIPS a ARM

`lw $1, dato($0)`	`ldr r2, =dato` `ldr r1, [r2]`	// cargar dirección de dato en r2 // cargar dir de mem apuntada por r2 en r1
`sw $1, dato($0)`	`str r2, =dato` `str r1, [r2]`	// cargar dirección de dato en r2 (la $2, dato) // guardar en dir de mem apuntada por r2, r1
`add $1, $2, $3` `(sub …)`	`add r1, r2, r3` `(sub …)`	
`addi $1, $2, 1`	`add r1, r2, #1`	
`sll $1, $2, 4`	`mov r1, r2, LSL #4`	// desplazamiento lógico a izq. 4 bits de r2 a r1

`sra $1, $2, 2`
`srl $1, $2, 2`
	`mov r1, r2, ASR #2`
	`mov r1, r2, LSR #2`
	`add r1, r1, r1, LSL #1`
		// desplazamiento aritmético a drch. 2 bits de r2 a r1
		// desplazamiento lógico a drch. 2 bits de r2 a r1
		// r1 ← r1 + (r1 << 1) = **3*r1** (multiplicar por 3)

`j dir`	`b dir`	// salta a la dirección dir
`jal fun`	`bl fun`	// salta a fun y guarda dir. sig. instrc. en reg. lr
`jr $ra`	`bx lr`	// salta a la dir. almacenada en lr (dir. retorno)

`beq $1, $2, dir`
`(bne)`
	`cmp r1, r2`
	`beq dir`
	`(bne)`
		// compara r1 y r2 (r1-r2) cargando flag Z.
		// mira flag Z, si Z=1 salta
		// (mira flag Z, si Z=0 salta)

Manejo de la pila		
`addi $sp, $sp, -4` `sw $ra, 0($sp)`	`push {lr}`	// salvar contenido de un registro (lr) en pila. Se pueden almacenar varios a la vez {r1, r2, r3, r4}
`lw $ra, 0($sp)` `addi $sp, $sp, 4`	`pop {lr}`	// sacar contenido de pila y meterlo en un registro (lr). Se puede hacer con varios rg. {r1, r2, r3, r4}
Finalización de un bucle cuando contador llega a 0 usando flags en op. aritméticas		
`addi $8, $8, -1` `beq $8, $0, exit`	`adds r8, r8, #-1` `beq exit`	// decrementa r8 y guarda estado en flags (Z) // mira flag Z, si Z=1 salta (Z cargado en adds)
Direccionamiento con autoincremento para recorrer estructuras en memoria		
`lw $8, 0($9)` `add $9, $9, 4`	`ldr r8, [r9], #4`	// direccionamiento post-auto-incrementado

AAPCS (Procedure Call Standard for the ARM Architecture)

Registro	Sinónimo	Especial	Papel en el estándar de llamada a procedimiento
r15		PC	Program Counter
r14		LR	Link Register
r13		SP	Stack Pointer
r12		IP	Intra-Procedure-call scratch register
r11	v8		Variable register 8
r10	v7		Variable register 7
r9		v6	Platform register (meaning defined by platform)
r8	v5		Variable register 5
r7	v4		Variable register 4
r6	v3		Variable register 3
r5	v2		Variable register 2
r4	v1		Variable register 1
r3	a4		Argument / scratch register 4
r2	a3		Argument / scratch register 3
r1	a2		Argument / result / scratch register 2
r0	a1		Argument / result / scratch register 1

Paso de parámetros a procedimiento:

- 4 primeros parámetros por a1, a2, a3 y a4
- el resto por la pila

Resultado del procedimiento

- hasta 2 resultados por a1 y a2

No se preservan a1, a2, a3 y a4 en la llamada a procedimiento
Se preservan en la llamada a procedimiento de v1 a v8.

Registro de estado y ejecución condicional

Flags en el registro de estado	Significado
N	Negativo: 1 si el resultado de la operación ha sido negativo
Z	Cero: 1 si el resultado de la operación ha sido 0
C	Carry: 1 si la operación ha generado acarreo
V	Overflow: 1 si el resultado de la operación no se puede representar en complemento a 2 con 32 bits

Campo condición {cond}	Significado
EQ	(equal) When Z is enabled (Z is 1)
NE	(not equal). When Z is disabled. (Z is 0)
GE	(greater or equal than, in two's complement). When both V and N are enabled or disabled (V is N)
LT	(lower than, in two's complement). This is the opposite of GE, so when V and N are not both enabled or disabled (V is not N)
GT	(greather than, in two's complement). When Z is disabled and N and V are both enabled or disabled (Z is 0, N is V)
LE	(lower or equal than, in two's complement). When Z is enabled or if not that, N and V are both enabled or disabled (Z is 1. If Z is not 1 then N is V)
MI	(minus/negative) When N is enabled (N is 1)
PL	(plus/positive or zero) When N is disabled (N is 0)
VS	(overflow set) When V is enabled (V is 1)
VC	(overflow clear) When V is disabled (V is 0)
HI	(higher) When C is enabled and Z is disabled (C is 1 and Z is 0)
LS	(lower or same) When C is disabled or Z is enabled (C is 0 or Z is 1) CS/HS (carry set/higher or same) When C is enabled (C is 1)
CS/HS	(carry set/higher or same) When C is enabled (C is 1)
CC/LO	(carry clear/lower) When C is disabled (C is 0)

Escribir registro estado	cmp inst{s}: add**s**, sub**s**, and**s**, …
Salto (b) condicional	b{cond}: b**eq**, b**ne**, b**gt**, b**le** …
Ejecución condicional	inst{cond}: add**eq**, sub**ne**, ldr**gt**, …

Comandos del Shell de Linux

Descripción	Comando
Conectar por red a la Raspberry Pi (usuario: tc, password: tc)	`ssh –X tc@nombreRaspPi`
Copiar fichero por red a la Raspberry Pi (password: tc)	`scp fichero tc@nombreRaspPi:`
Copiar fichero desde la Raspberry Pi (password: tc)	`scp tc@nombreRaspPi:fichero .`
Mostrar contenido directorio	`ls`
Cambiar de directorio	`cd nombreDir`
Crear nuevo directorio	`mkdir nombreDir`
Borrar fichero	`rm nombreFichero`
Editar fichero ensamblador	`geany programa.s &`

Compilar fichero ensamblador
Enlazar
Enlazar con librería wiringPi
	`as –o programa.o programa.s`
	`gcc –o programa programa.o`
	
	`gcc –o programa programa.o -lwiringPi`

Compilar con Makefile	`make programa`

Ejecutar programa compilado
Ejecutar programa que usa wiringPi (pide password: tc)
	`./programa`
	
	`sudo ./programa`

Apagar Raspberry Pi y borrar ficheros	`./borraYapaga.sh`

Copiar ficheros desde/hacia la Raspberry Pi de forma gráfica

1. Abrir explorador de ficheros de Linux
2. En la barra de dirección poner:
   sftp://tc@nombreRaspPi
3. Cambiar de directorio a:
   /home/tc/practicas
4. Arrastrar y soltar ficheros desde/hacia esta ventana.

Nota: Se puede utilizar la tarea del campus virtual para la entrega de las prácticas como almacenamiento temporal de vuestros fichero aunque aún no estén terminados.

Usando GDB (supuesto se ha compilado con "-g")

Descripción	Comando
Lanzar el gdb	`gdb [-tui] executable_name`
Listar el código fuente	**l**`ist`
Poner breakpoint	**b**`reak line-number`
Listar nros. breakpoints	**i**`nfo` **break**
Borrar un breakpoint	**d**`el num_breakpoint`
Borrar todos los breakpoints	**cl**`ear`
Ejecutar programa	**r**`un`
Ver contenidos registros CPU	**i**`nfo registers`
Desensamblar código máquina	**disas**`semble [starting_address ending_address]`
Ejecución por pasos de instrucciones máquina	**s**`tep i [number_steps]`
Ejecución por pasos (entra en funciones que estén en el fuente)	**n**`ext i`
Continuar ejecución	**c**`ontinue`
Ver contenidos memoria	**x**`/nfs starting_address` **Options** / **Possible values** n: Number of ítems — any number f: Format — **o**ctal, he**x**, **d**ecimal,**u**nsigned decimal, bi**t**, **f**loat, **a**ddress, **i**nstruction, **c**har, and **s**tring s: Size — **b**yte, **h**alfword, **w**ord, **g**iant(8B) Ej.: ver en hexadecimal las 12 palabras en tope pila: `x/12xw $sp`
Imprimir pos_mem/registro	**p**`rint/f label/reg`
Salir	**q**`uit`

Comandos TUI (Text Uslayouter Interface for gdb)

Descripción	Comando
Activar/desactivar TUI	`C-x A`
Cambiar ventana activa	`focus next/prev src/asm/split/regs, C-x o`
Cambiar ventanas mostradas	`layout next/prev/src/asm/split/regs, C-x 1, C-x 2`
Refrescar pantalla	`refresh, C-L`

Librería *wiringPi*

Llamadas a función desde ensamblador:

1. Cargar los parámetros de la función en los registros (hasta cuatro parámetros de entrada se pasan por registro: 1er parmámetro en r0, 2º en r1, …)
2. Llamar a la función:
   bl *nombre_función*
3. Recoger el valor de retorno de la función (si lo hay) del registro r0

Función	Argumentos	Descripción
int **wiringPiSetup** (void)	Ninguno	Función que inicializa la librería. Debe ser llamada antes de utilizer cualquiera de las funciones de la librería.
void **pinMode** (int pin, int mode)	**pin:** id del pin a configurar **mode:** modo de operación del pin (INPUT, OUTPUT)	Configura el modo de funcionamiento (mode) del pin espeficifcado por el parametro "pin". El modo puede ser de entrada (INPUT, 0) o salida (OUTPUT, 1).
void **digitalWrite** (int pin, int value)	**pin:** id del pin en el que escribir **value:** valor a escribir en el pin	Genera la salida especificada por el parámetro "value" (0, 1) en el pin especificado por el parámetro "pin". Dicho pin ha debido ser incializado como de salida (OUTPUT) con la función pinMode.
int **digitalRead** (int pin)	**pin:** id del pin cuyo estado (0, 1) se quiere leer	Lee el estado del pin (0, 1) especificado por el parámetro "pin". El valor leído lo devuelve por el registro r0.
void **delay** (unsigned int howLong)	**howLong:** número de milisegundos	Suspende la ejecución del programa durante los milisegundos especificados en el parámetro "howLong".
void **delayMicroseconds** (unsigned int howLong)	**howLong:** número de microsegundos	Suspende la ejecución del programa durante los microsegundos especificados en el parámetro "howLong".

Ejemplo de lectura del estado de un pin, correspondiente a un pulsador de la placa conectada a la Raspberry Pi.

```
.include "wiringPiPins.s"      // fichero con definiciones de los pines
                               // de la placa #BUTTON1, #RLED1, …
                               // y modos para los pines #INPUT, #OUTPUT

      bl wiringPiSetup         // inicializamos la librería wiringPi

                               // Vamos a configurar el pin asociado al
                               // pulsador 1 como de entrada (pinMode)
      mov r0, #BUTTON1         // indicamos el pin del pulsador en r0
                               // (1er parámetro de la función)
      mov r1, #INPUT           // indicamos el modo en el registro r1
                               // (2º parámetro de la función)
      bl pinMode               // llamamos a la función pinMode

                               // Vamos a leer el estado del pulsador 1
                               // con la función digitalRead
      mov r0, #BUTTON1         // indicamos el pin del pulsador en r0
                               // (1er parámetro de la función)
      bl digitalRead           // llamamos a la función digitalRead
      cmp r0, #0               // En el registro r0 está el valor leído
      …                        // del pin asociado al pulsador (0, 1)
```