Arquitectura del Computador 2015

Guía de laboratorio: RASPBERRY PI

OBJETIVO:

Crear el prototipo de una aplicación real, utilizando los conceptos aprendidos de programación de MIPS, aplicados a la programación de procesadores ARM. Se utilizará como plataforma una Raspberry Pi y se podrán elegir como entradas del sistema pulsadores y sensores táctiles y como salidas leds, buzzers y relays.

ANTES DE COMENZAR:

1) Cada grupo deberá traer una computadora, si es posible con lector de SD, donde deberá instalar:

```
$ wget http://www.cl.cam.ac.uk/freshers/raspberrypi/tutorials/os/downloads/arm-none-eabi.tar.bz2
--2012-08-16 18:26:29--
http://www.cl.cam.ac.uk/freshers/raspberrypi/tutorials/os/downloads/arm-none-eabi.tar.bz2
Resolving www.cl.cam.ac.uk (www.cl.cam.ac.uk)... 128.232.0.20,
2001:630:212:267::80:14
Connecting to www.cl.cam.ac.uk (www.cl.cam.ac.uk)|128.232.0.20|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 32108070 (31M) [application/x-bzip2]
Saving to: `arm-none-eabi.tar.bz2'

100%[======================================>] 32,108,070   668K/s   in 67s

2012-08-16 18:27:39 (467 KB/s) - `arm-none-eabi.tar.bz2' saved
[32108070/32108070]

$ tar xjvf arm-none-eabi.tar.bz2
arm-2008q3/arm-none-eabi/
arm-2008q3/arm-none-eabi/lib/
arm-2008q3/arm-none-eabi/lib/libsupc++.a
arm-2008q3/arm-none-eabi/lib/libcs3arm.a
...
arm-2008q3/share/doc/arm-arm-none-eabi/info/gprof.info
arm-2008q3/share/doc/arm-arm-none-eabi/info/cppinternals.info
arm-2008q3/share/doc/arm-arm-none-eabi/LICENSE.txt

$ export PATH=$PATH:$HOME/arm-2008q3/bin
```

* En Windows:

- Instalar YAGARTO Tools y YAGARTO GNU ARM toolchain for Windows. No usar espacios: 'C:\YAGARTO\' no 'C:\Program Files\YAGARTO\'.

  http://sourceforge.net/projects/yagarto/

- Instalar MinGW

  http://sourceforge.net/projects/mingw/?source=typ_redirect

2) Descargar de Moodle la carpeta “template”.

METODOLOGÍA DE TRABAJO:

1) Cada grupo recibirá una Raspberry Pi con acceso a 7 GPIOs (Entrada/Salida de Propósito General), una SD con el sistema operativo Raspbian funcionando y los periféricos que considere necesarios para la aplicación que desea implementar.
2) Deberán realizar un programa que permita leer las entradas y activar las salidas seleccionadas según la lógica de su aplicación.

PARA ESCRIBIR EL CÓDIGO ASSEMBLY:

- Abrir el archivo “main.s”. Respetar las líneas de encabezado (instrucciones para el ensamblador).
- En el pdf “BCM2835-ARM-Peripherals” buscar las direcciones de los registros necesarios para la configuración y control de los GPIO. **Importante**: hay un error en la dirección base de los registros del GPIO (GPFSEL0), se debe reemplazar 0x7E200000, por **0x20200000**.
- Habilitar el puerto y configurar como entrada o salida: Configurar los GPFSELn correspondientes a los GPIO que se dese utilizar siendo: 000 = entrada y 001 = salida. Notar que cada registro GPFSEL permite configurar 10 puertos (GPFSEL0 del 0 al 9, GPFSEL1 del 10 al 19…) y que se utilizan 3 bits para la configuración de cada GPIO.
- Leer una entrada: se puede utilizar cualquier registro de lectura, como GPLEVn, que devuelve en el bit correspondiente al GPIO que se desea leer: 0 si la entrada está en nivel bajo y 1 si está en nivel alto. (Recordar el uso de máscaras para leer el estado de un bit).
- Encender una salida: utilizar GPSETn, colocando 1 en los bits correspondientes a los GPIO que se desea encender.
- Apagar una salida: utilizar GPCLRn, colocando 1 en los bits correspondientes a los GPIO que se desea apagar.
- Crear un bucle infinito para que el programa se ejecute mientras la Raspberry Pi permanezca encendida.

- Dejar una línea vacía al final del código. El toolchain espera esta línea vacía para asegurarse de que el archivo realmente terminó. Si no se coloca, aparece una advertencia cuando se corre el ensamblador.

PARA CREAR LA IMAGEN:

- Abrir el terminal de la computadora y dirigirse al directorio de la carpeta "template", escribir "make" ("mingw32-make" en Windows) y presionar enter. Si no ocurre ningún error, dentro de la carpeta deben generarse tres archivos: kernel.list, kernel.map y kernel.img, que es la imagen de su sistema operativo.

PARA CORRER EL PROGRAMA EN LA RASPBERRY PI:

- Renombrar el archivo "kernel.img" como "recovery.img" y copiar en la SD, reemplazando la imagen actual.
- Montar la SD en la Raspberry.
- Conectar los módulos periféricos en los puertos de GPIO correspondientes.
- Alimentar la placa desde el USB de la computadora.

ANEXO: INSTRUCCIONES DE ARM

Registros disponibles: r0 a r12.

ldr reg,=val puts the number **val** into the register named **reg**.
mov reg,#val puts the number **val** into the register named **reg**.
lsl reg,#val shifts the binary representation of the number in **reg** by **val** places to the left.
str reg,[dest,#val] stores the number in **reg** at the address given by **dest** + **val**.
ldr reg,[dest,#val] puts the number in the address given by **dest** + **val** into the register named **reg**.
sub reg,#val subtracts the number **val** from the value in **reg**.
mov reg1,reg2 copies the value in **reg2** into **reg1**.
add reg,#val adds the number **val** to the contents of the register **reg**.
and reg,reg,#val computes the Boolean and function of the number in **reg** with **val**.
name: labels the next line **name**.
b label causes the next line to be executed to be **label**.
teq reg,#val checks if the number in **reg** is equal to **val**.
bne name checks that the top bit of the status field is 0, and loops back to **name**.