Práctica 1: Microprocesador segmentado

NOTA: Se recomienda encarecidamente leer detenidamente el enunciado entero de cada ejercicio antes de comenzar con la codificación.

Ejercicio 1 (3 puntos)

El objetivo de esta práctica es implementar la versión segmentada del microprocesador MIPS, cuyos detalles se pueden encontrar en el libro de referencia de la asignatura: "*Estructura y Diseño de Computadores: La interfaz Hardware/ Software*", por David A. Patterson y John L. Hennessy, disponible en la biblioteca de la EPS. Se recomienda seguir el libro para realizar esta práctica. En concreto, se sigue el modelo segmentado del MIPS, detallado en el capítulo 6 (secciones 6.2 y 6.3).

Se pide implementar el microprocesador MIPS en su versión uniciclo el cual servirá de base para su posterior segmentación. Dicho procesador no necesita soportar todo el juego de instrucciones completo, sino las siguientes instrucciones: ADD, ADDI, AND, OR, XOR, SUB, LW, SW, SLTI, LUI, J, BEQ y NOP, cuyos códigos de operación y descripción se incluyen más abajo.

Para la realización de este ejercicio, en el material de la práctica se entrega el código VHDL de las memorias de datos e instrucciones (fichero "`memoria.vhd`") así como un esqueleto para el procesador (fichero "`procesador.vhd`") y un banco de pruebas para el mismo (fichero "`procesador_TB.vhd`").
En la siguiente figura se describe la jerarquía de algunos de los ficheros VHDL que conformarían el diseño a realizar:

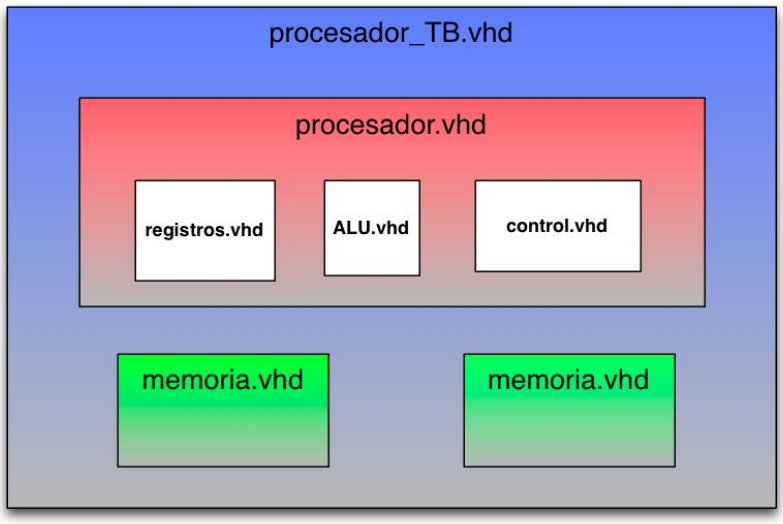


Figura 1. Jerarquía de ficheros VHDL en el diseño.

El esquema del procesador uniciclo a desarrollar es el indicado en la siguiente figura:

NOTA: Para la realización de este ejercicio se pueden revisar y reutilizar las prácticas realizadas a lo largo de la asignatura "Estructura de Computadores" (del segundo cuatrimestre del primer curso).

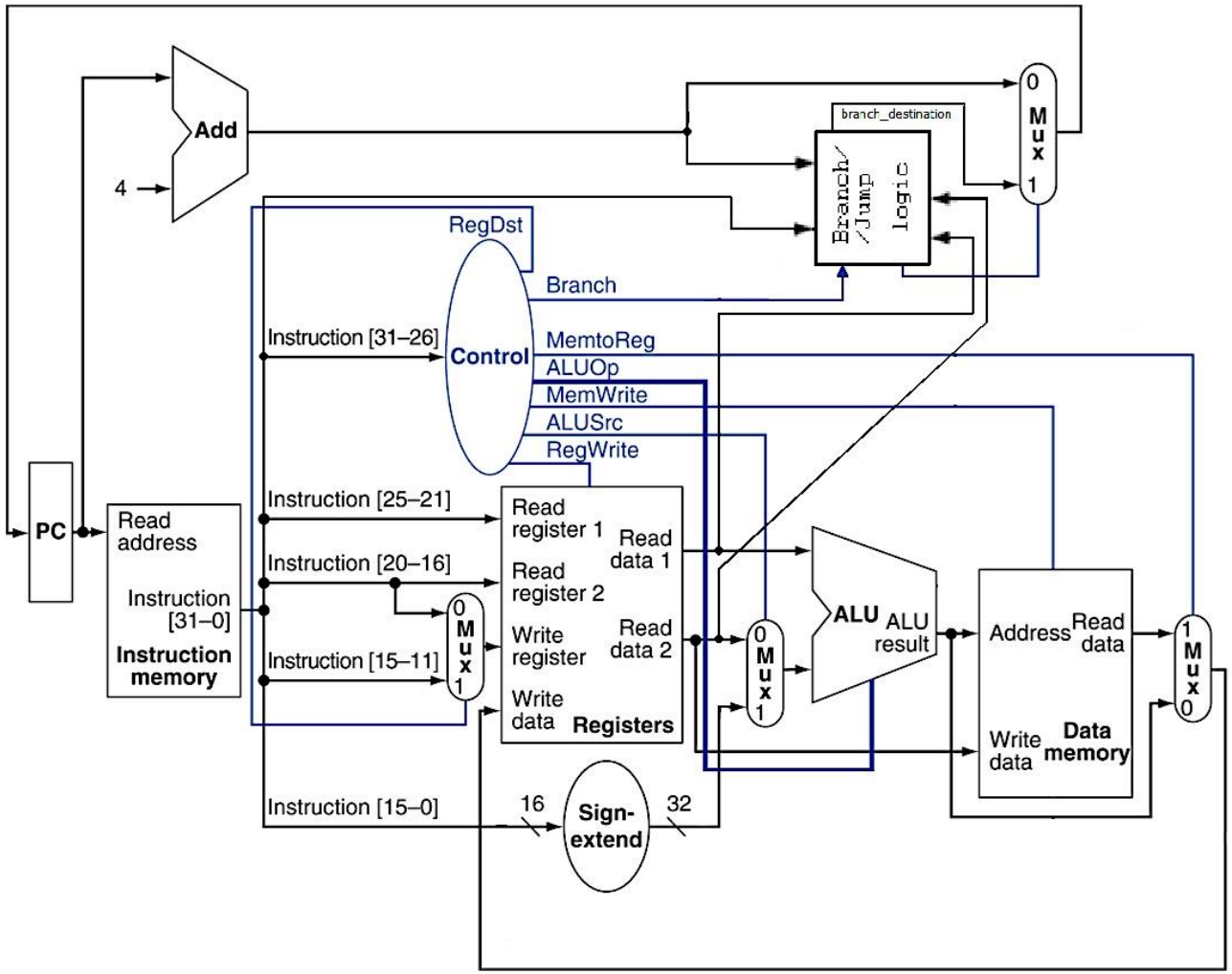
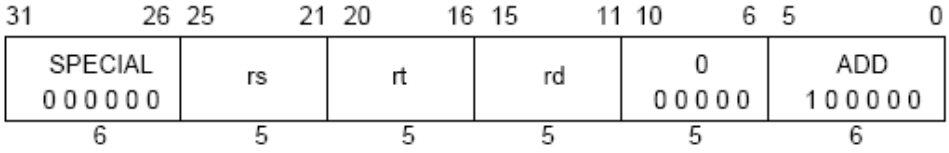


Figura 2. Modelo de microprocesador uniciclo.

Las instrucciones soportadas tienen la siguiente descripción:

ADD — Add Word

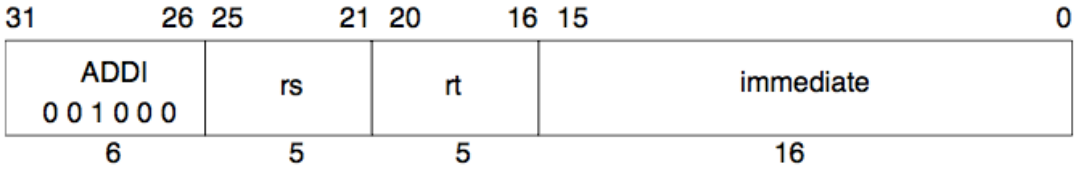


Format: ADD rd, rs, rt — **MIPS I**

Purpose: To add 32-bit integers. If overflow occurs, then trap.

Description: rd ← rs + rt

ADDI — Add Immediate Word

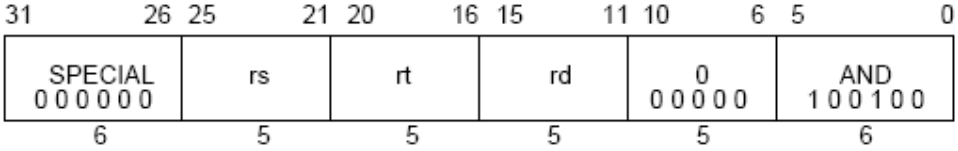


Format: ADDI rt, rs, immediate — **MIPS I**

Purpose: To add a constant to a 32-bit integer. If overflow occurs, then trap.

Description: rt ← rs + immediate

AND — And



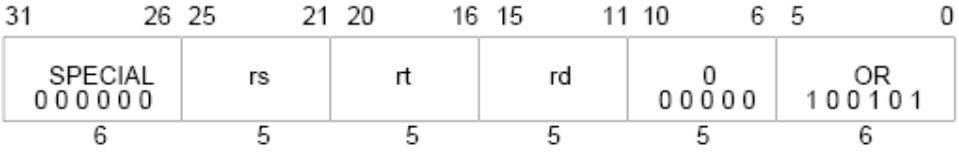
Format: AND rd, rs, rt — **MIPS I**

Purpose: To do a bitwise logical AND.

Description: rd ← rs AND rt

OR

Or



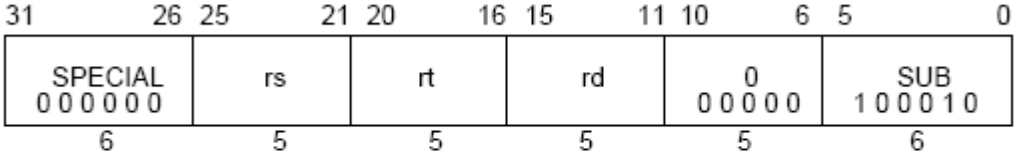
Format: OR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical OR.

Description: rd ← rs OR rt

SUB

Subtract Word



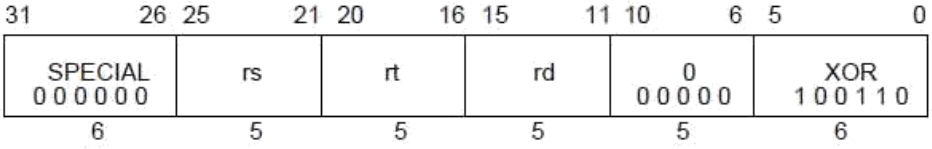
Format: SUB rd, rs, rt **MIPS I**

Purpose: To subtract 32-bit integers. If overflow occurs, then trap.

Description: rd ← rs - rt

XOR

Exclusive OR



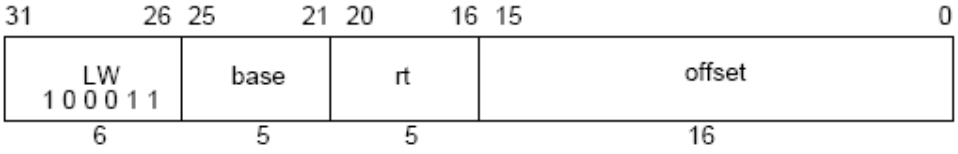
Format: XOR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical EXCLUSIVE OR.

Description: rd ← rs XOR rt

Combine the contents of GPR *rs* and GPR *rt* in a bitwise logical exclusive OR operation and place the result into GPR *rd*.

LW — Load Word

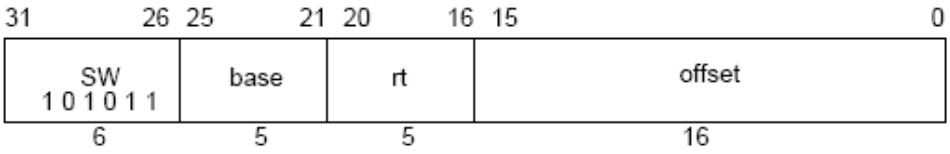


Format: LW rt, offset(base) **MIPS I**

Purpose: To load a word from memory as a signed value.

Description: rt ← memory[base+offset]

SW — Store Word

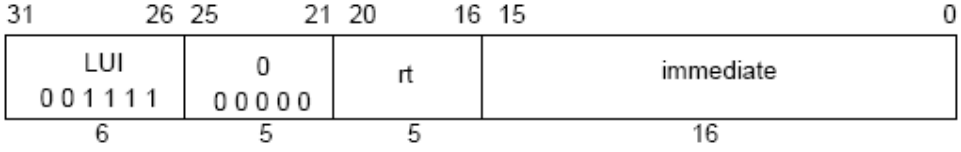


Format: SW rt, offset(base) **MIPS I**

Purpose: To store a word to memory.

Description: memory[base+offset] ← rt

LUI — Load Upper Immediate

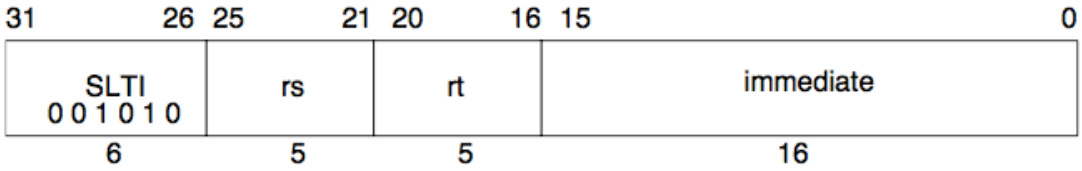


Format: LUI rt, immediate **MIPS I**

Purpose: To load a constant into the upper half of a word.

Description: rt ← immediate || 0^{16}

Set on Less Than Immediate **SLTI**



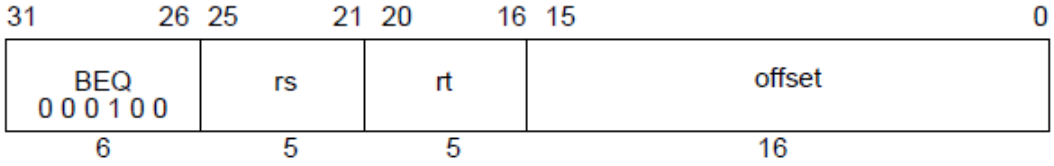
Format: SLTI rt, rs, immediate **MIPS I**

Purpose: To record the result of a less-than comparison with a constant.

Description: rt ← (rs < immediate)

Compare the contents of GPR *rs* and the 16-bit signed *immediate* as signed integers and record the Boolean result of the comparison in GPR *rt*. If GPR *rs* is less than *immediate* the result is 1 (true), otherwise 0 (false).

BEQ **Branch on Equal**



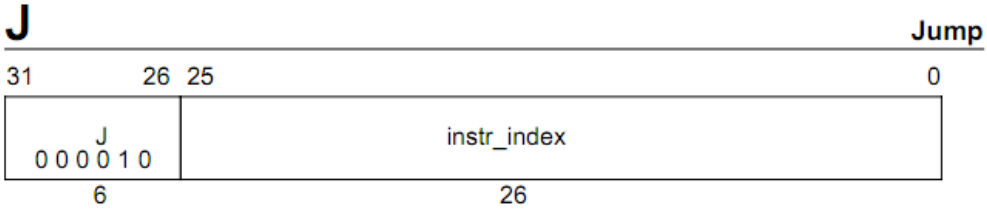
Format: BEQ rs, rt, offset **MIPS I**

Purpose: To compare GPRs then do a PC-relative conditional branch.

Description: if (rs = rt) then branch

An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (**not** the branch itself), in the branch delay slot, to form a PC-relative effective target address.

If the contents of GPR *rs* and GPR *rt* are equal, branch to the effective target address after the instruction in the delay slot is executed.



Format: J target **MIPS I**

Purpose: To branch within the current 256 MB aligned region.

Description:

This is a PC-region branch (not PC-relative); the effective target address is in the "current" 256 MB aligned region. The low 28 bits of the target address is the *instr_index* field shifted left 2 bits. The remaining upper bits are the corresponding bits of the address of the instruction in the delay slot (**not** the branch itself).

Jump to the effective target address. Execute the instruction following the jump, in the branch delay slot, before jumping.

La instrucción NOP

La instrucción NOP no viene definida como una instrucción en sí en el juego de instrucciones del MIPS, sino como un caso particular de otra instrucción. En lo referente esta práctica, se considera como instrucción NOP aquella que tenga los 32 bits a '0'. El comportamiento del procesador ante esta instrucción debe ser el de no modificar ninguno de los recursos del mismo (banco de registros, memoria de datos).

Ejercicio 2 (7 puntos)

Se pide implementar el microprocesador MIPS en su versión segmentada. **El resultado debe ser un micro capaz de realizar en el caso ideal (sin riesgos) una instrucción por ciclo de reloj**. Para el ejercicio básico, no es necesario que el modelo soporte riesgos (salvo el estructural de acceso a memorias separadas de instrucciones y datos).

Todos los registros utilizados para la segmentación han de cumplir las siguientes características:

1. Funcionar por flanco de subida del reloj (`rising_edge(clk)`).
2. *Resetearse* **asíncronamente** utilizando la señal "`Reset`" de la entidad "`processor_core`".

Se recomienda utilizar como guía el esquema reflejado en la siguiente figura (ligeramente diferente al del libro).

NOTA: Nótese en la figura que la lógica relativa al cálculo de las direcciones de salto y su decisión se encuentra en la etapa ID del *pipeline*.

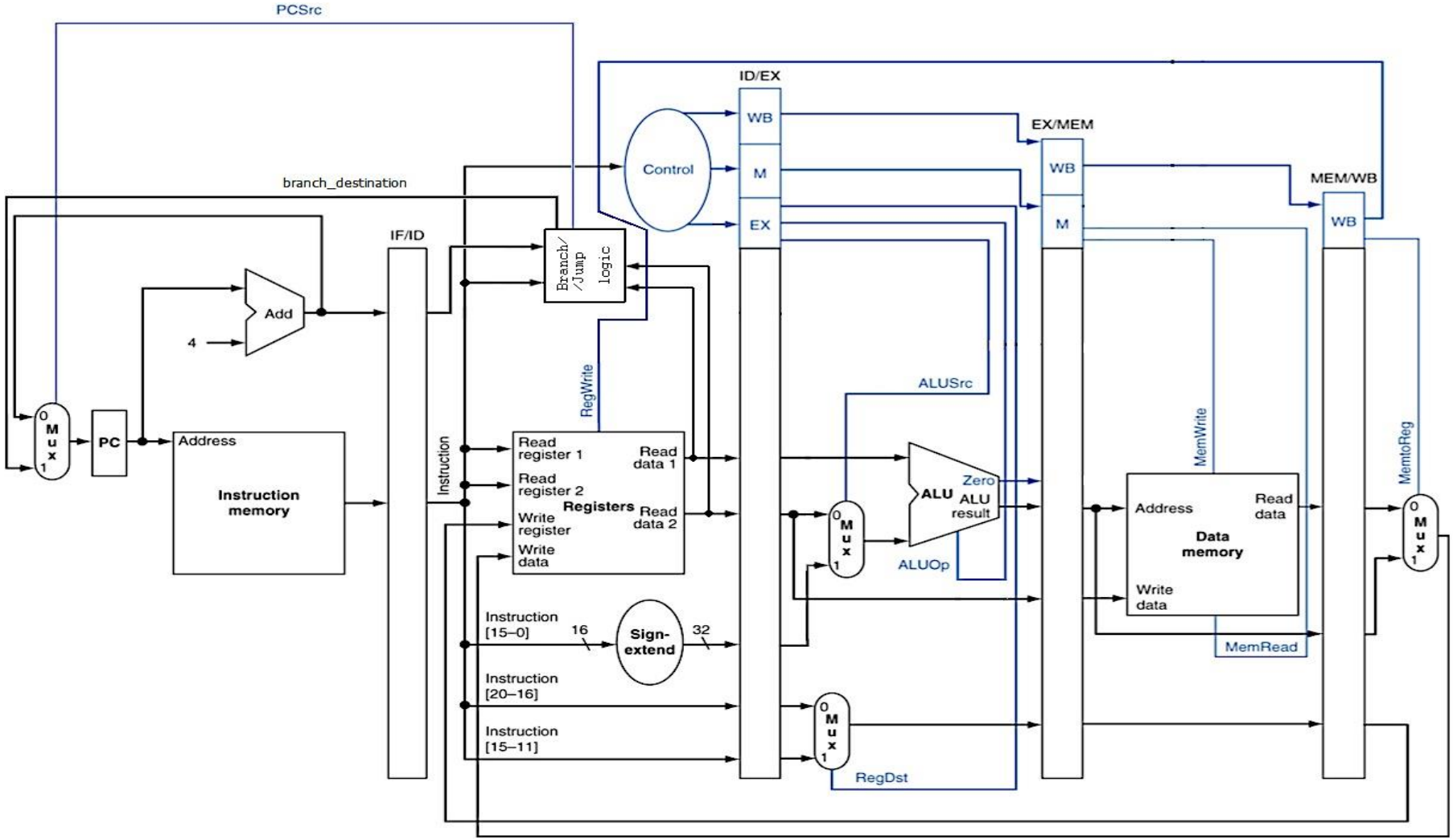


Figura 3. Modelo de microprocesador segmentado.

MATERIAL A ENTREGAR

Sólo los ficheros VHDL de los ejercicios 1 y 2.

La entrega se realizará a través de Moodle, con fecha tope el día antes del comienzo de la siguiente práctica hasta las 23:59 de la noche. Esto es:

- **Grupo 1361, 1362 y 1363 –** hasta el martes 12 de octubre de 2016 a las 23:59.
- **Grupos 1301, 1311 y 1312 –** hasta el miércoles 13 de octubre de 2016 a las 23:59.

*El profesor de cada grupo de prácticas podrá requerir una defensa

AYUDAS Y AVISOS

Los ficheros "`instrucciones`" y "`datos`" que contienen las memorias de instrucciones y datos respectivamente deben localizarse en el mismo directorio desde donde se lance la simulación del proyecto. Si no fuera así, en el fichero "`procesador_TB.vhd`" se puede dar la ruta completa de dichos ficheros cambiando las líneas de código:

```
C_ELF_FILENAME        => "instrucciones",
...
C_ELF_FILENAME        => "datos",
```

Por otras donde indique la ruta completa a ambos ficheros:

```
C_ELF_FILENAME        => "D:\nombredirectorio\instrucciones",
...
C_ELF_FILENAME        => "D:\nombredirectorio\datos",
```

En la carpeta "`sw`" que se provee en el material de la práctica se encuentra tanto un programa ensamblador de prueba como las herramientas para compilarlo al formato que utilizan nuestras memorias. El programa de prueba "`programa.asm`" proporcionado en la práctica **NO** incluye riesgos de datos y prueba todas las instrucciones del ejercicio básico. Para generar los ficheros con el contenido de las memorias a partir del programa en ensamblador es suficiente con hacer doble clic sobre el fichero "`arqo_comp.bat`".

La tabla contenida en el archivo "`registers.html`" proporcionada en la práctica muestra la traducción de los nombres de registros usados en ensamblador al número de registro en el micro, del 0 al 31.