Práctica 1: Microprocesador segmentado

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

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).

Generalidades del diseño

Se pide implementar el microprocesador MIPS en su versión uniciclo el cual servirá de base para su posterior segmentación No es necesario que el procesador soporte el juego de instrucciones completo de MIPS, sino las siguientes instrucciones: **ADD, ADDI, SUB, SLTI, AND, OR, XOR, LUI, LW, SW, J, BEQ y NOP**, cuyos códigos de operación y descripción se incluyen más abajo. En cualquier caso, tras la segmentación, la instrucción *beq*, que implica riesgos de control por ser un salto, funcionará "anómalamente" en esta versión básica del microprocesador.

Para facilitar la realización de este ejercicio, se facilita una versión simplificada del procesador en versión uniciclo con los diferentes sub-bloques del procesador ya definidos: el banco de registros, la unidad aritmética lógica (ALU), la unidad de control (óvalo "Control" en las figuras 2 y 3), el bloque que genera los códigos de control para la ALU (óvalo "ALU control") y el propio procesador. Los alumnos deberán completar estos ficheros escribiendo el código de las correspondientes arquitecturas VHDL.

Por otra parte, para verificar el funcionamiento del procesador se entrega un fichero VHDL completo con un testbench muy simple que instancia el micro y las memorias de instrucciones y datos, para las cuales se entrega también completo un fichero VHDL.

La relación jerárquica en el diseño es pues la siguiente (ver figura 1):

- El testbench (*processor_tb*), instancia al procesador (*processor*) y a las 2 memorias (*memory*).
- El procesador (*processor*) instancia el banco de registros (*reg_bank*), la ALU (*alu*), la unidad de control (*control_unit*) y el bloque para el control de la ALU (*alu_control*).
- El fichero runsim_arq.do que permite lanzar la simulación invocando este script Modelsim/Questasim desde la consola de modelsim. Este fichero compila, simula y abre las formas de onda descriptas en wave_arq.do

Todos los registros utilizados deben usar el flanco de subida de reloj y *reset* asíncrono activo a nivel alto. Por otra parte, para la codificación del procesador se recomienda:

- No crear componentes adicionales: hacer el contador de programa, los registros del pipeline, multiplexores, etc. como código dentro de la arquitectura de *processor*.
- Utilizar asignaciones concurrentes (simples y condicionales).

El material se entrega con la siguiente estructura, que deberá mantenerse:

- Directorio *rtl/* : contiene el código del procesador:

processor.vhd	procesador, a completar
alu.vhd	ALU, completa
reg_bank.vhd	Banco de registros, completo

control_unit.vhd	Unidad de control, a completar
alu_control.vhd	Control de la ALU, a completar

- Directorio *sim/* : contiene lo necesario para simular el procesador (todos los ficheros están completos salvo *wave.do*, donde los alumnos podrán grabar la configuración de ondas que deseen):

processor_tb.vhd	Testbench
memory.vhd	Modelo simple de memoria síncrona
programa.s	Código fuente de un programa ensamblador de prueba
programa.lst	Listado con la codificación del programa
instrucciones	Fichero de datos para la memoria de instrucciones
datos	Fichero de datos para la memoria de datos
runsim.do	Script de simulación para ModelSim
wave.do	Script de configuración de ondas paras ModelSim

El testbench instancia las memorias, las cuales leen, en una fase de inicialización en tiempo = 0, sus datos desde los ficheros "*instrucciones*" y "*datos*" (nótese que estas memorias son modelos que simplifican lo que sería un escenario con memorias reales). Estos dos ficheros resultan del ensamblado de *programa.s*. Por otra parte, el fichero *programa.lst* nos permite saber en qué dirección está cada instrucción y cómo está codificada. Este programa ejecuta todas las instrucciones del micro y, como explican sus comentarios, incluye instrucciones que, por riesgos de datos y control, no pueden ser ejecutadas correctamente por esta versión del procesador. Como referencia adicional, en Moodle se puede encontrar un enlace con la equivalencia entre nombres de registro MIPS y su número 0-31 (*registers.html*).

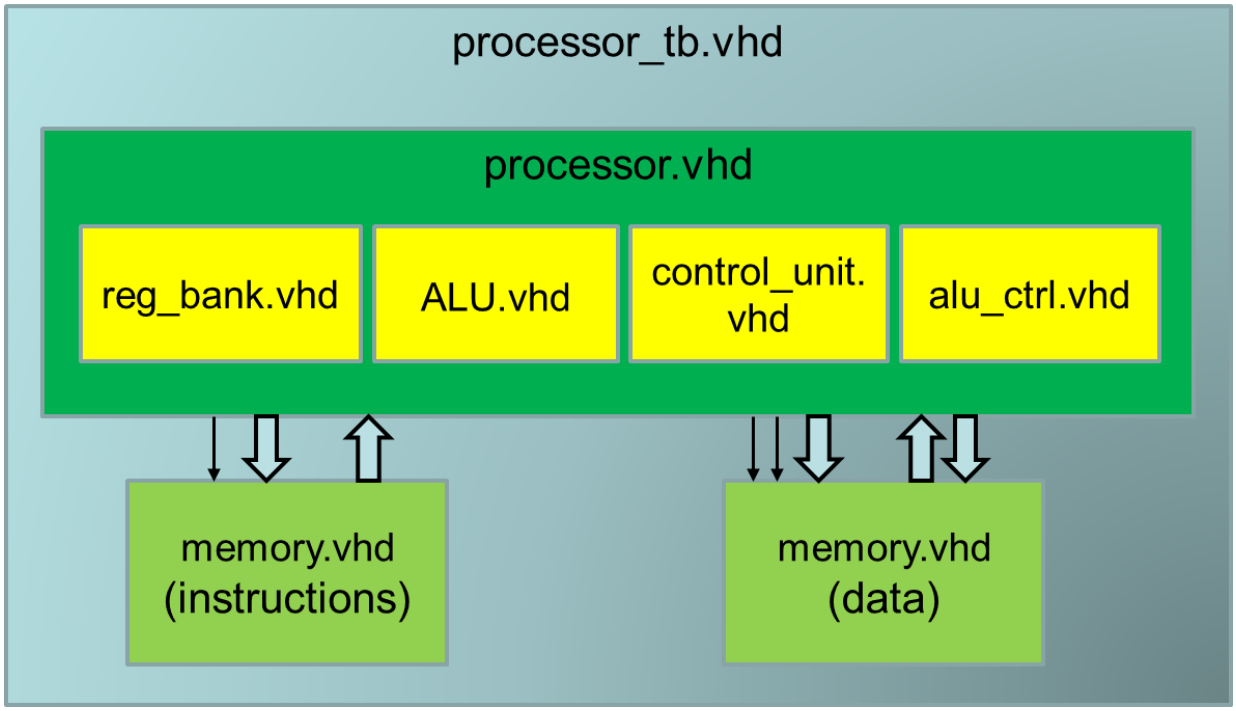


Figura 1. Jerarquía de ficheros VHDL en el diseño.

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). Pero se deben utilizar los ficheros provistos.

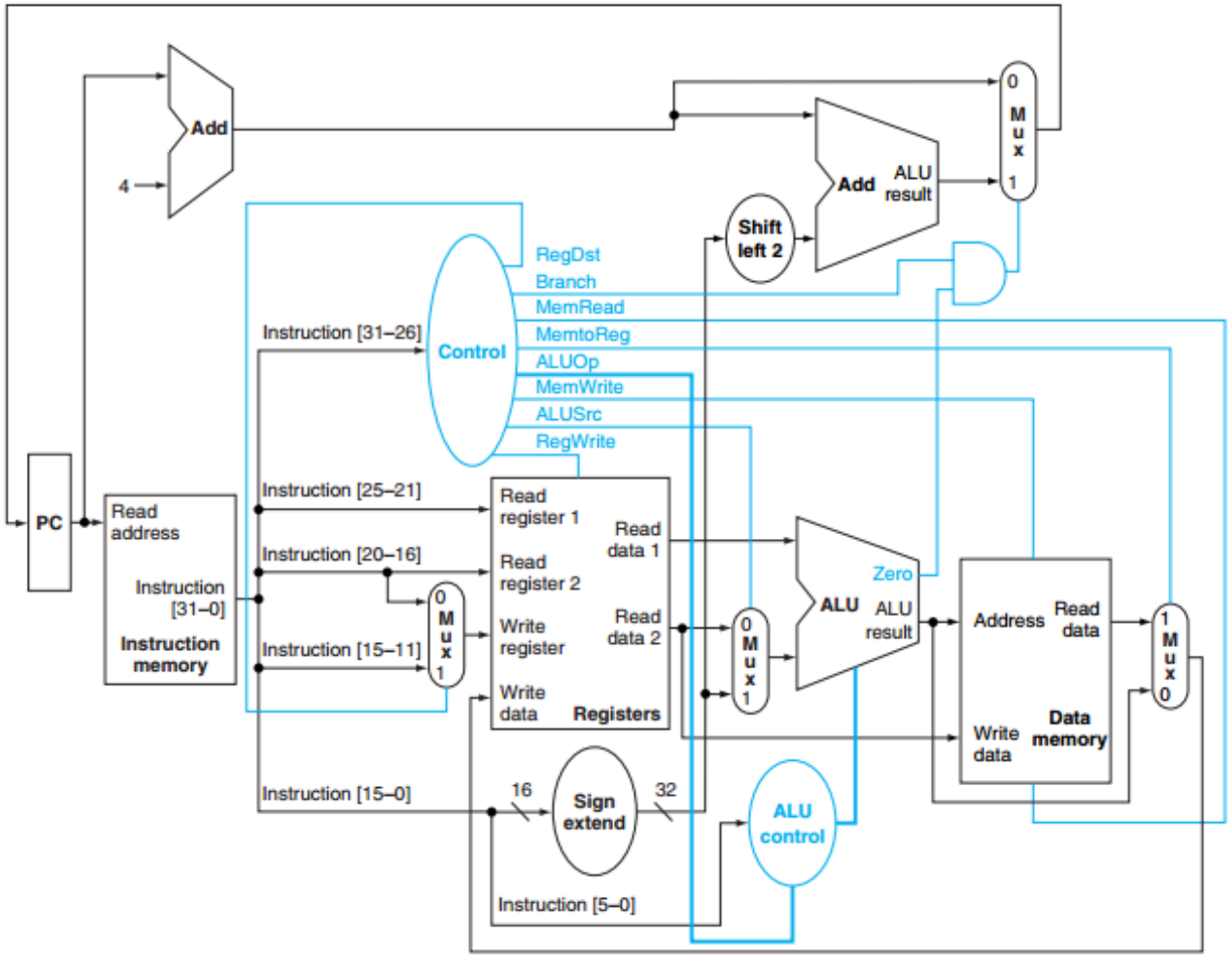
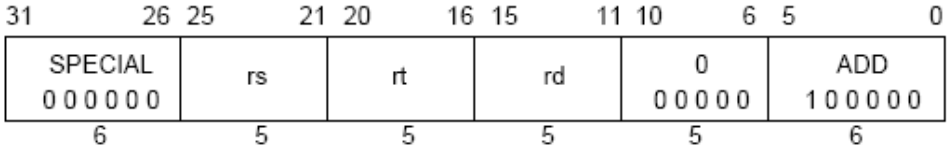


Figura 2. Modelo de microprocesador uniciclo.

Las instrucciones soportadas tienen la siguiente descripción:

ADD — Add Word



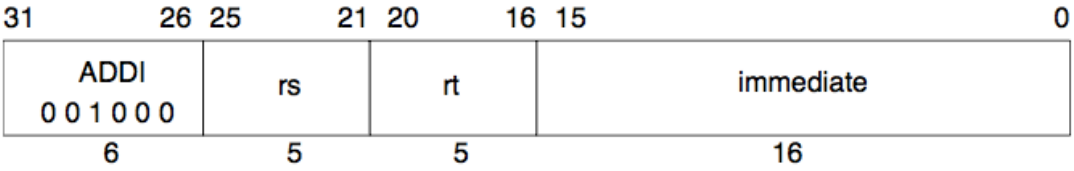
MIPS I

Format: ADD rd, rs, rt

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

Description: rd ← rs + rt

ADDI — Add Immediate Word



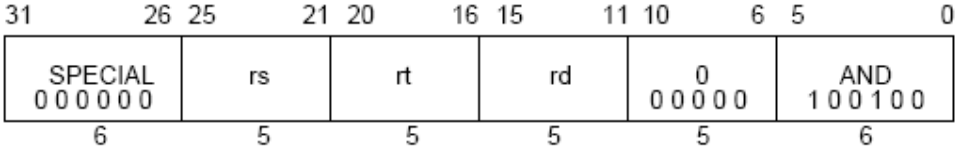
MIPS I

Format: ADDI rt, rs, immediate

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

Description: rt ← rs + immediate

AND — And



MIPS I

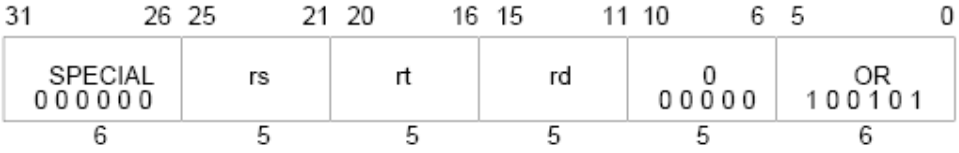
Format: AND rd, rs, rt

Purpose: To do a bitwise logical AND.

Description: rd ← rs AND rt

OR

Or



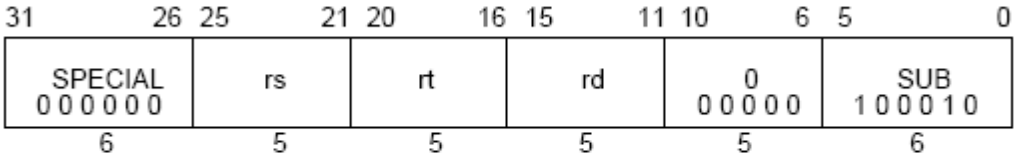
Format: OR rd, rs, rt **MIPS I**

Purpose: To do a bitwise logical OR.

Description: rd ← rs OR rt

SUB

Subtract Word



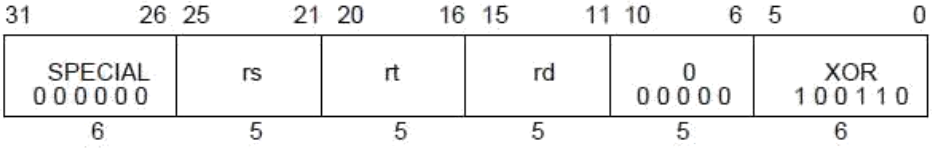
Format: SUB rd, rs, rt **MIPS I**

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

Description: rd ← rs - rt

XOR

Exclusive OR



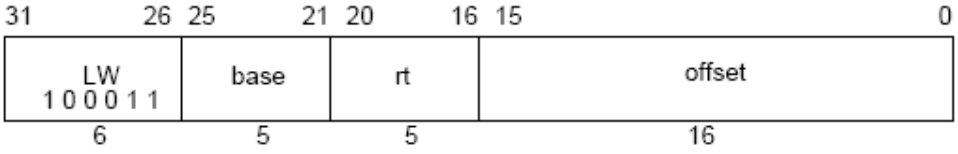
Format: XOR rd, rs, rt **MIPS I**

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

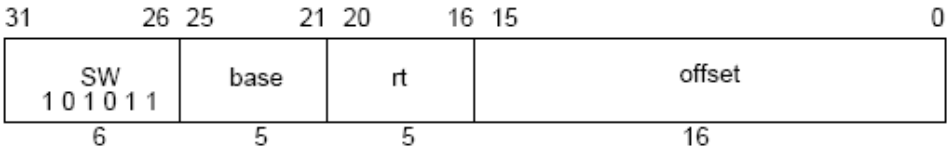


Format: LW rt, offset(base) **MIPS I**

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

Description: rt ← memory[base+offset]

SW — Store Word

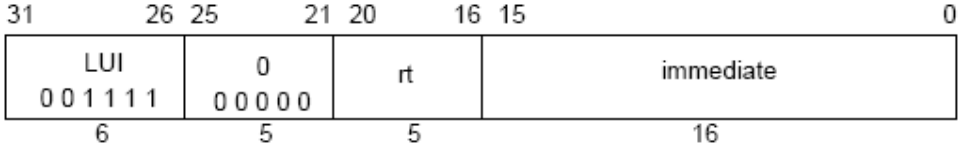


Format: SW rt, offset(base) **MIPS I**

Purpose: To store a word to memory.

Description: memory[base+offset] ← rt

LUI — Load Upper Immediate

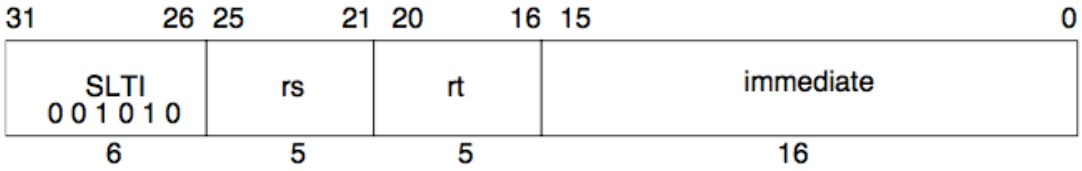


Format: LUI rt, immediate **MIPS I**

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

Description: rt ← immediate || 0^{16}

Set on Less Than Immediate **SLTI**



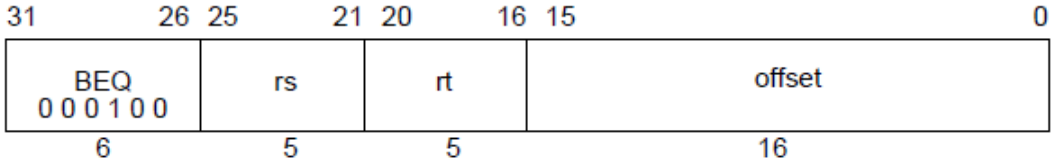
Format: SLTI rt, rs, immediate **MIPS I**

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**

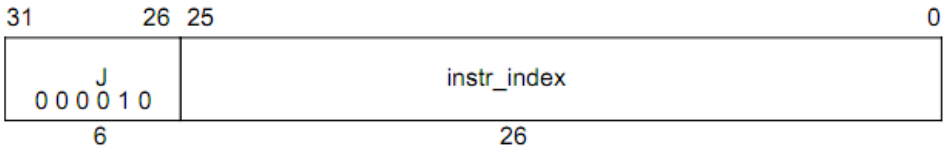
Purpose: To compare GPRs then do a PC-relative conditional branch.

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.

J **Jump**



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 1 (2 puntos)

Se pide completar el microprocesador MIPS en su versión uniciclo. El diseño entregado como punto de partida, no soporta el set de instrucciones descripto (**ADD, ADDI, SUB, SLTI, AND, OR, XOR, LUI, LW, SW, J, BEQ y NOP**).

Ejercicio 2 (1 puntos)

Se pide realizar un programa en ensamblador simple que sea capaz de probar las instrucciones que puedan haber quedado sin probar en el programa provisto como punto de partida.

Ejercicio 3 (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: Se recomienda enfáticamente disponer de una versión impresa del esquema para realizar anotaciones sobre el dibujo (o la correspondiente aplicación digital donde apuntar los nombres de señales).

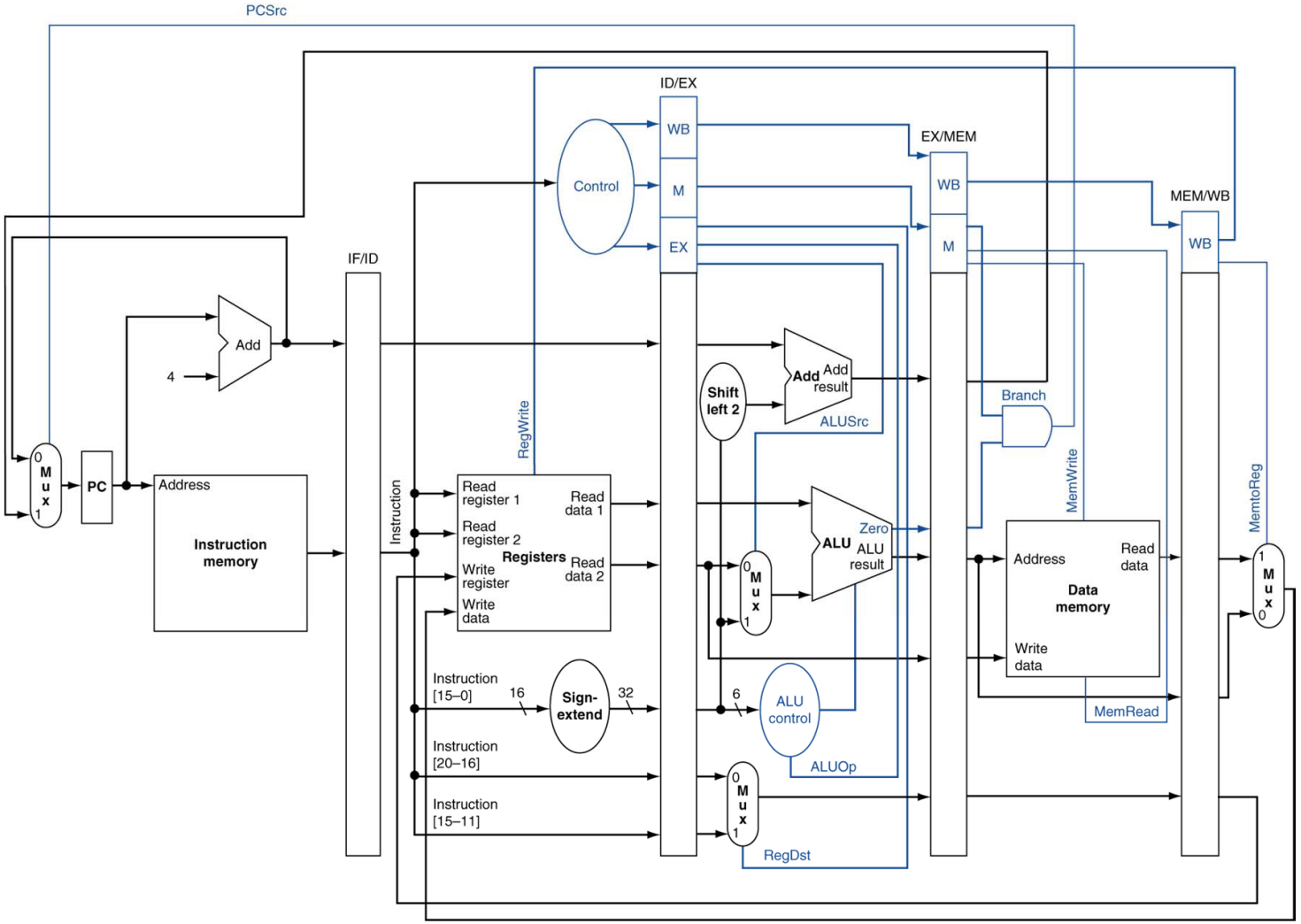


Figura 3. Modelo de microprocesador segmentado.

MATERIAL A ENTREGAR

Los ficheros VHDL de los ejercicios 1 y 3 y el ensamblador del ejercicio 2. Un fichero de texto plano con nombre P1_grupoxxxx.txt que indique número de pareja, integrantes y cualquier aclaración que considere necesario para poder corregir la práctica.

Organizarlo en 3 carpetas, uno por cada ejercicio.

La entrega se realizará a través de Moodle, con fecha tope el viernes de la última semana asignada a esta práctica hasta las 23:59 de la noche. Esto es viernes 9 de octubre de 2020.

*El profesor de cada grupo de prácticas podrá requerir una defensa, la cual es parte de la nota de la práctica

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        => "X:\nombredirectorio\instrucciones",
...
C_ELF_FILENAME        => "X:\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 (ejecutable Windows). 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.