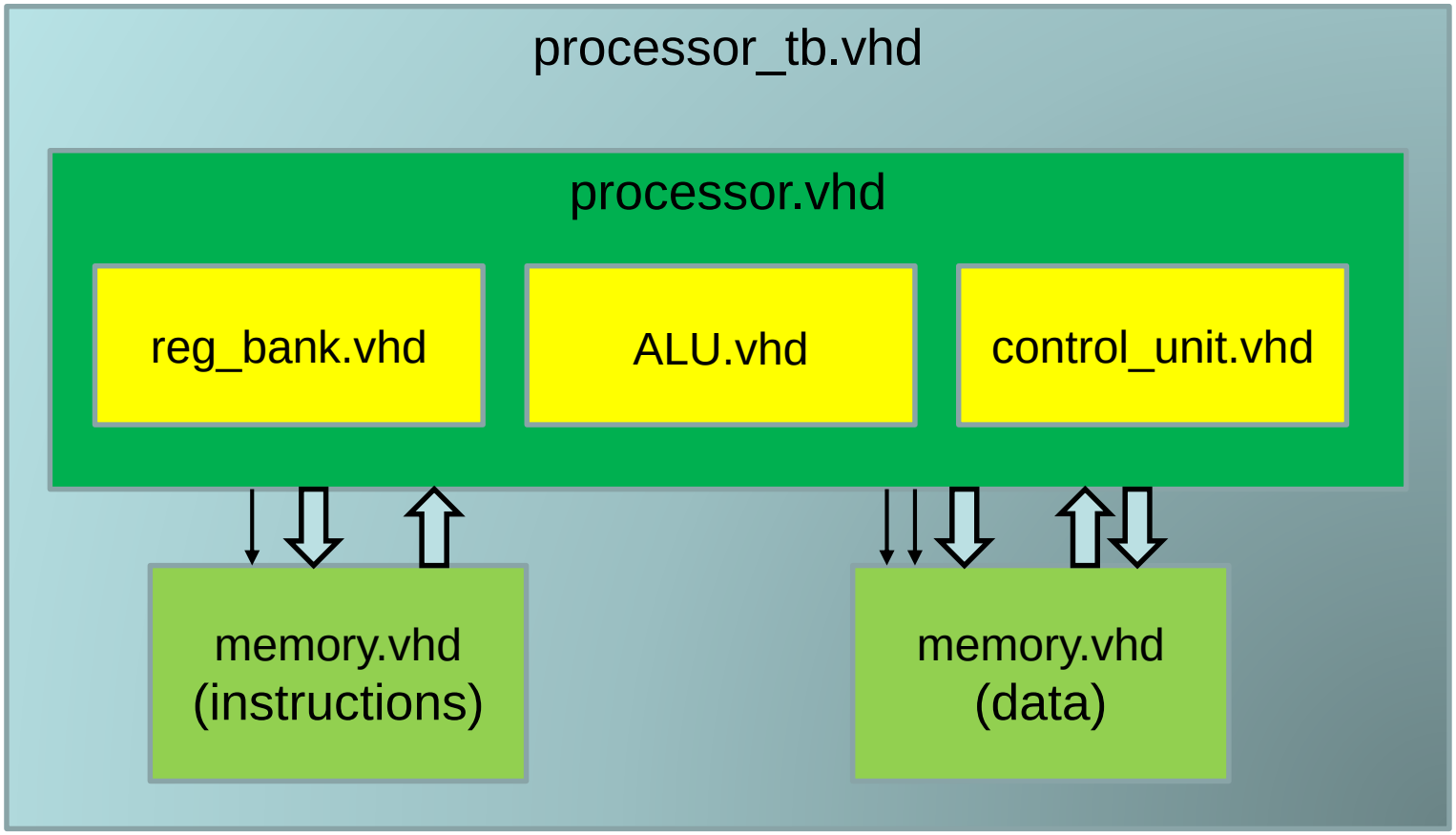
Presentación *Práctica 1*

Arquitectura de Computadores

3º de grado en Ingeniería Informática y

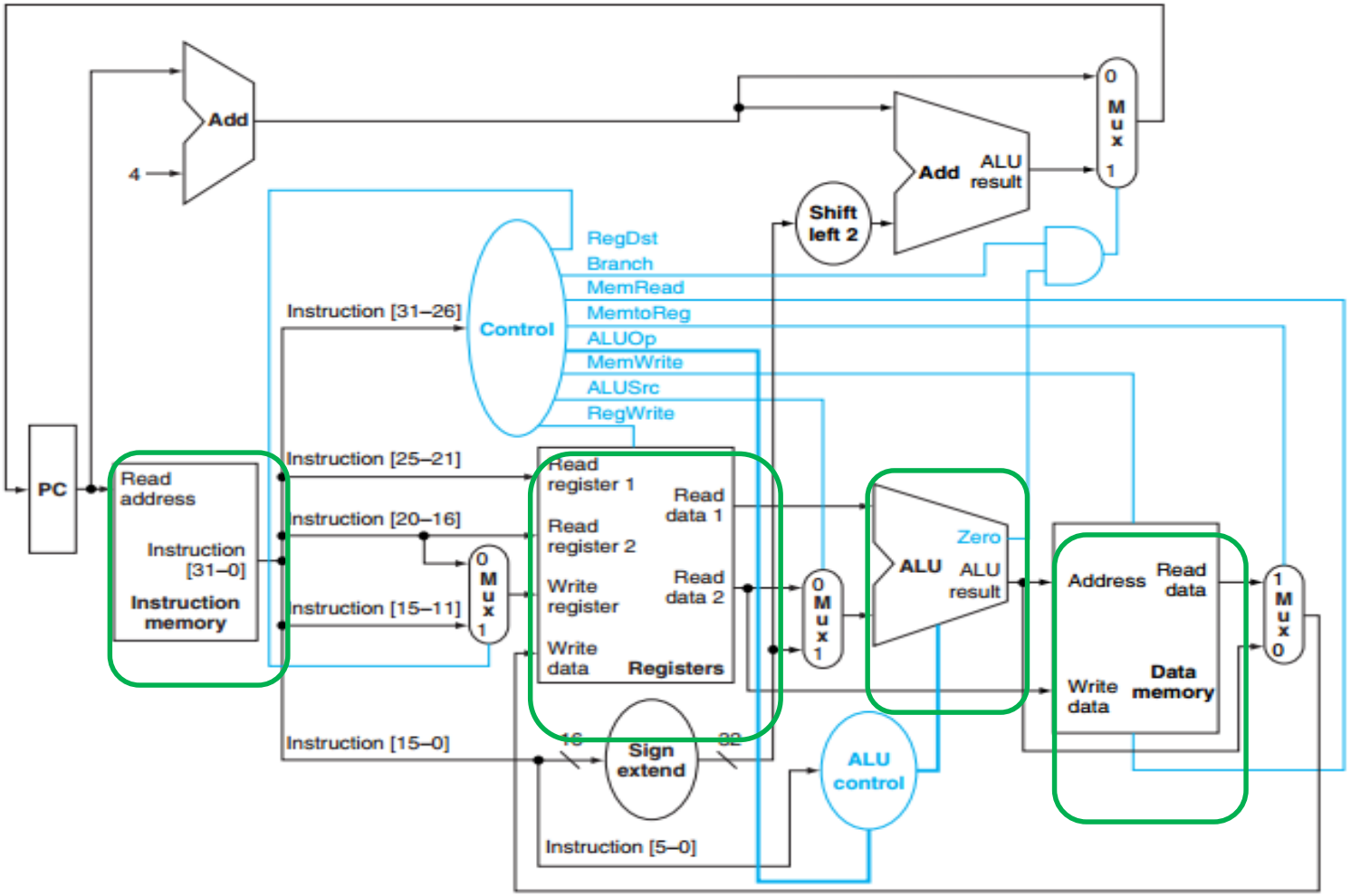
3º de doble grado en Ing. Informática y Matemáticas

Diseño jerárquico



Módulos que se proveen y módulos a desarrollar

MIPS uniciclo



Entidades externas ya desarrolladas

Material Entregado

- 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

processor_tb.vhd	**El banco de pruebas (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

Unidad de control

- Genera las señales de control desde el código de operación de la instrucción
  - Entra el código de operación (OpCode)
  - Salen las señales de control: MemToReg, MemWrite, Branch/Jump, AluCtrl, RegDst, RegWrite, AluOP

- A desarrollar por el alumno. Utilizar la declaración de entidad provista

ALU (Arithmetic Logic Unit)

- Debe ser capaz de ejecutar el conjunto de instrucciones que se implementarán
- Las señales de control las general el bloque conbinacional **“ALU CONTROL”**
- Aunque ya está desarrollado el modulo hay que verificar que se puedan ejecutar todas las instrucciones pedidas. Caso contrario habrá que agregarlas

Ya desarrollado
(alu.vhd)

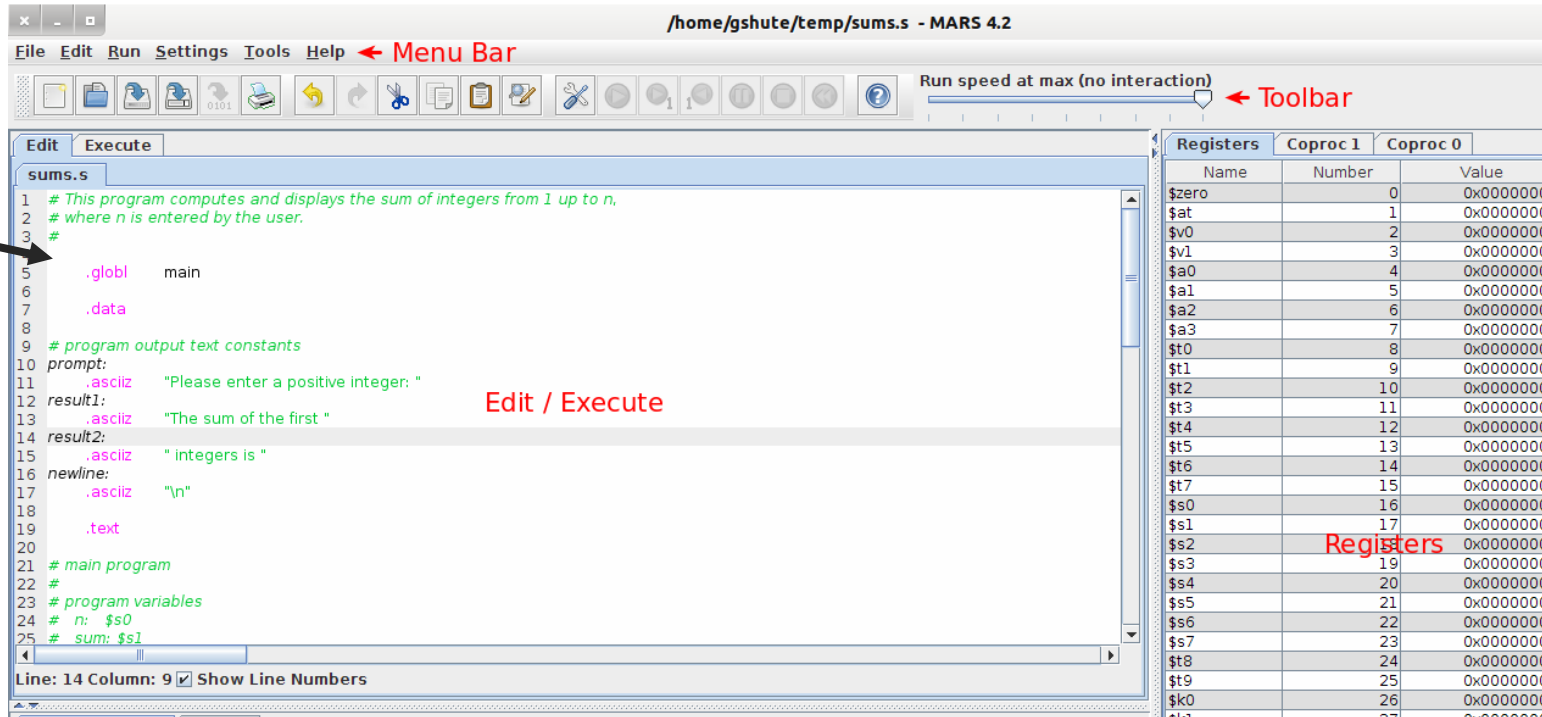
Banco de registros

- **32 registros**
- **Lectura asíncrona**
- **Escritura síncrona**
  - Flanco de bajada
  - El resto de registros del procesador en flanco de subida
- **Registro 0**
  - Siempre vale 0
  - Escrituras sin efecto

Ya desarrollado (reg_bank.vhd)

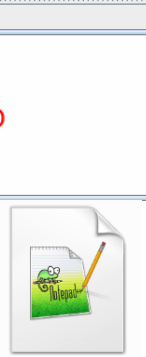
Simulación: generar instrucciones y datos

programa.asm



Para la P1 ya están generados

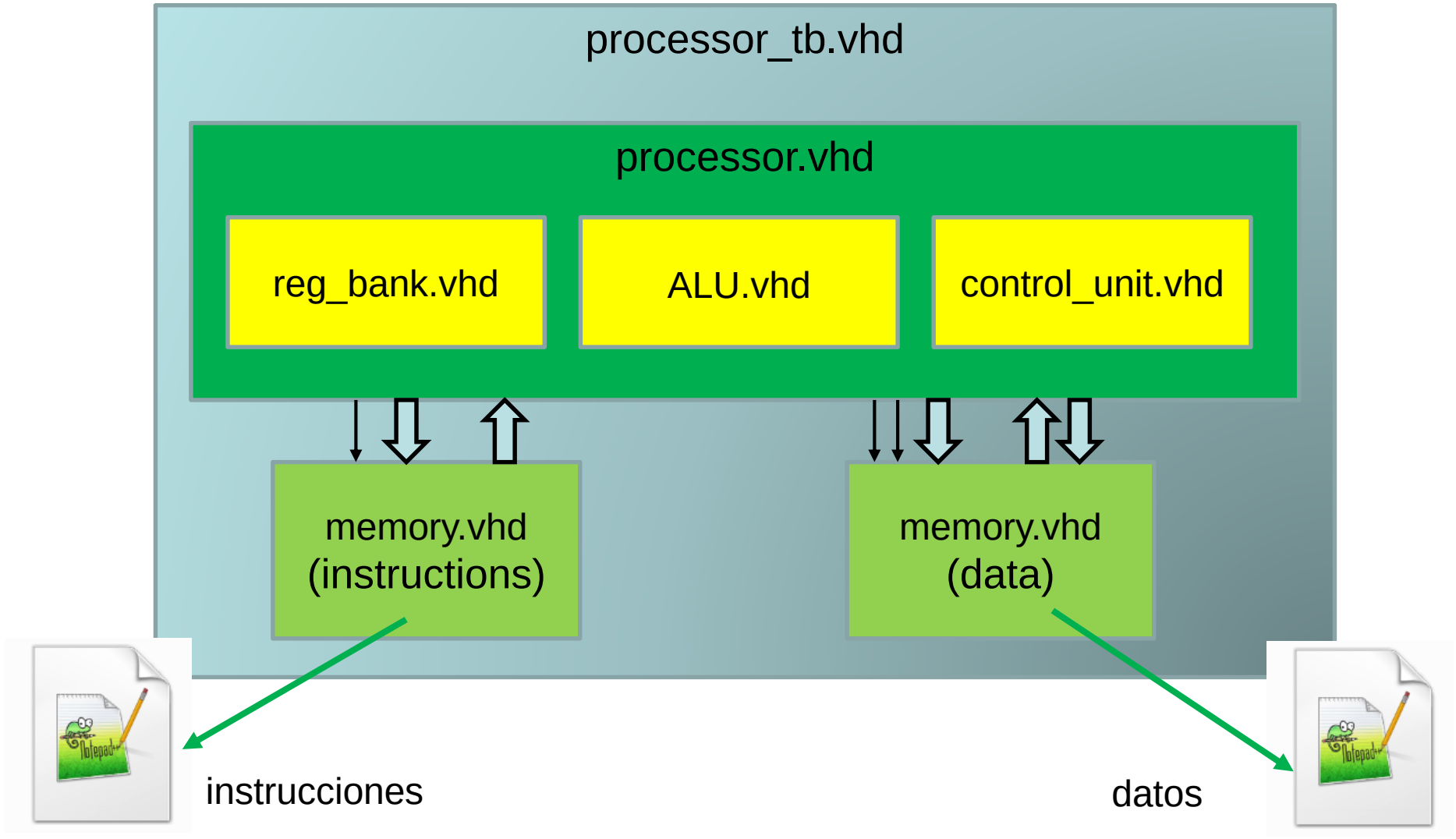
instrucciones



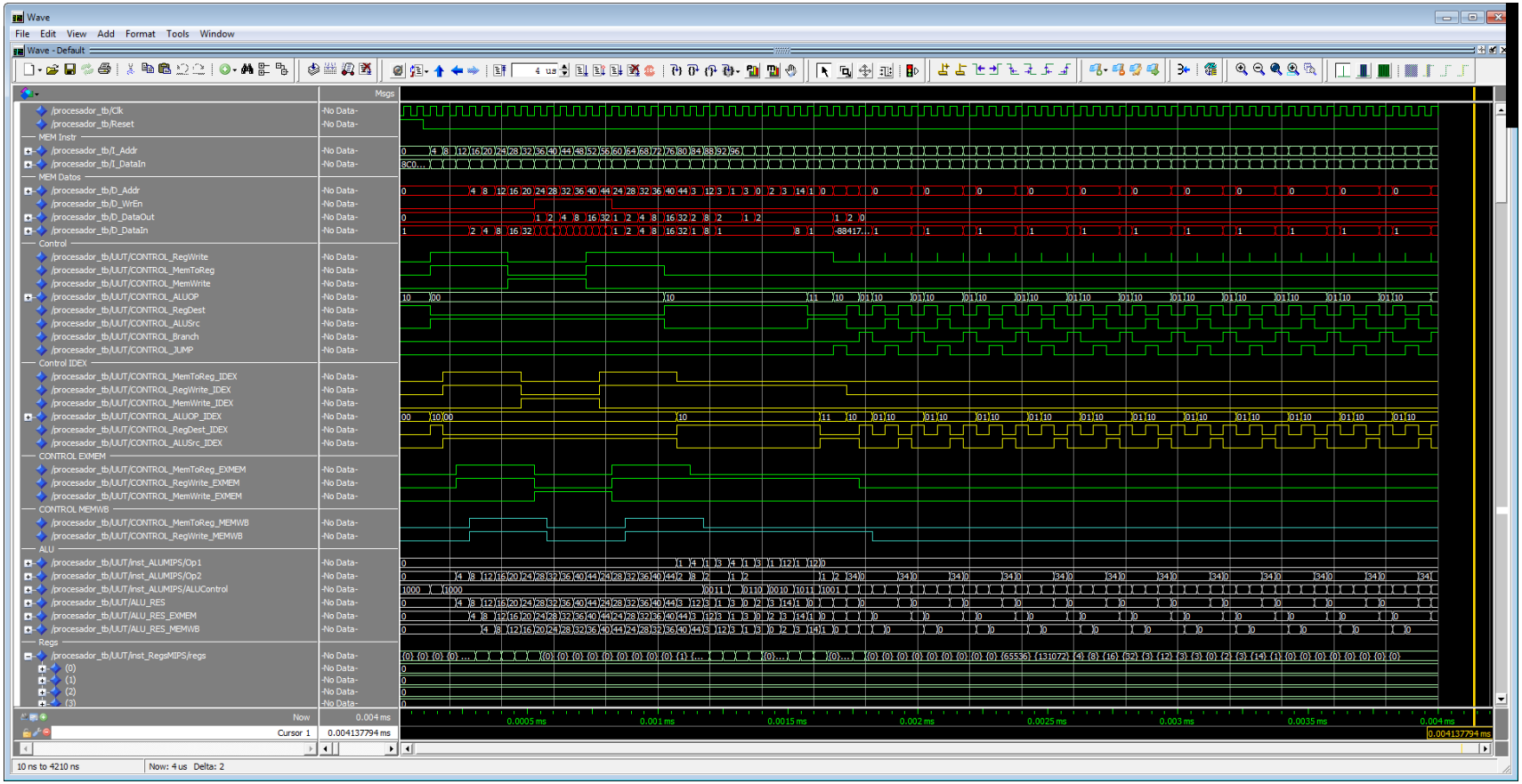
datos

En P2 utilizaremos MARS

Simulación


processor_tb.vhd
processor.vhd
reg_bank.vhd
ALU.vhd
control_unit.vhd
memory.vhd
(instructions)
memory.vhd
(data)
instrucciones
datos

Simulación


Wave
File Edit View Add Format Tools Window
Wave - Default
Msgs
/procesador_tb/Clk
/procesador_tb/Reset
MEM Instr
/procesador_tb/I_Addr
/procesador_tb/I_DataIn
MEM Datos
/procesador_tb/D_Addr
/procesador_tb/D_WrEn
/procesador_tb/D_DataOut
/procesador_tb/D_DataIn
Control
/procesador_tb/UUT/CONTROL_RegWrite
/procesador_tb/UUT/CONTROL_MemToReg
/procesador_tb/UUT/CONTROL_MemWrite
/procesador_tb/UUT/CONTROL_ALUOP
/procesador_tb/UUT/CONTROL_RegDest
/procesador_tb/UUT/CONTROL_ALUSrc
/procesador_tb/UUT/CONTROL_Branch
/procesador_tb/UUT/CONTROL_JUMP
Control IDEX
/procesador_tb/UUT/CONTROL_MemToReg_IDEX
/procesador_tb/UUT/CONTROL_RegWrite_IDEX
/procesador_tb/UUT/CONTROL_MemWrite_IDEX
/procesador_tb/UUT/CONTROL_ALUOP_IDEX
/procesador_tb/UUT/CONTROL_RegDest_IDEX
/procesador_tb/UUT/CONTROL_ALUSrc_IDEX
CONTROL EXMEM
/procesador_tb/UUT/CONTROL_MemToReg_EXMEM
/procesador_tb/UUT/CONTROL_RegWrite_EXMEM
/procesador_tb/UUT/CONTROL_MemWrite_EXMEM
CONTROL MEMWB
/procesador_tb/UUT/CONTROL_MemToReg_MEMWB
/procesador_tb/UUT/CONTROL_RegWrite_MEMWB
ALU
/procesador_tb/UUT/inst_ALUMIPS/Op1
/procesador_tb/UUT/inst_ALUMIPS/Op2
/procesador_tb/UUT/inst_ALUMIPS/ALUControl
/procesador_tb/UUT/ALU_RES
/procesador_tb/UUT/ALU_RES_EXMEM
/procesador_tb/UUT/ALU_RES_MEMWB
Regs
/procesador_tb/UUT/inst_RegsMIPS/regs
(0)
(1)
(2)
(3)
Now
0.004 ms
Cursor 1
0.004137794 ms
0.0005 ms
0.001 ms
0.0015 ms
0.002 ms
0.0025 ms
0.003 ms
0.0035 ms
0.004 ms
10 ns to 4210 ns
Now: 4 us Delta: 2

Consejos de desarrollo

- Codificación
  - No crear nuevos componentes. Multiplexores y registros se pueden describir con procesos o asignaciones concurrentes
- EDITOR: Xilinx ISE
  - Check syntax (Detección de errores de VHDL)
  - Synthetize (Detección de errores “HW” (latches, …))
- Simulador: Mentor ModelSIM
  - Integrado en Xilinx ISE
  - **File -> Save -> wave.do**
  - **File -> Load -> wave.do**
    - Permite guardar el formato de vuestra simulación
      - » Agrupaciones, formatos de representación, colores, …

Consejos antes de empezar

1. Leer el guion completo de la práctica

2. Mirar en el libro de la teoría el sistema a implementar

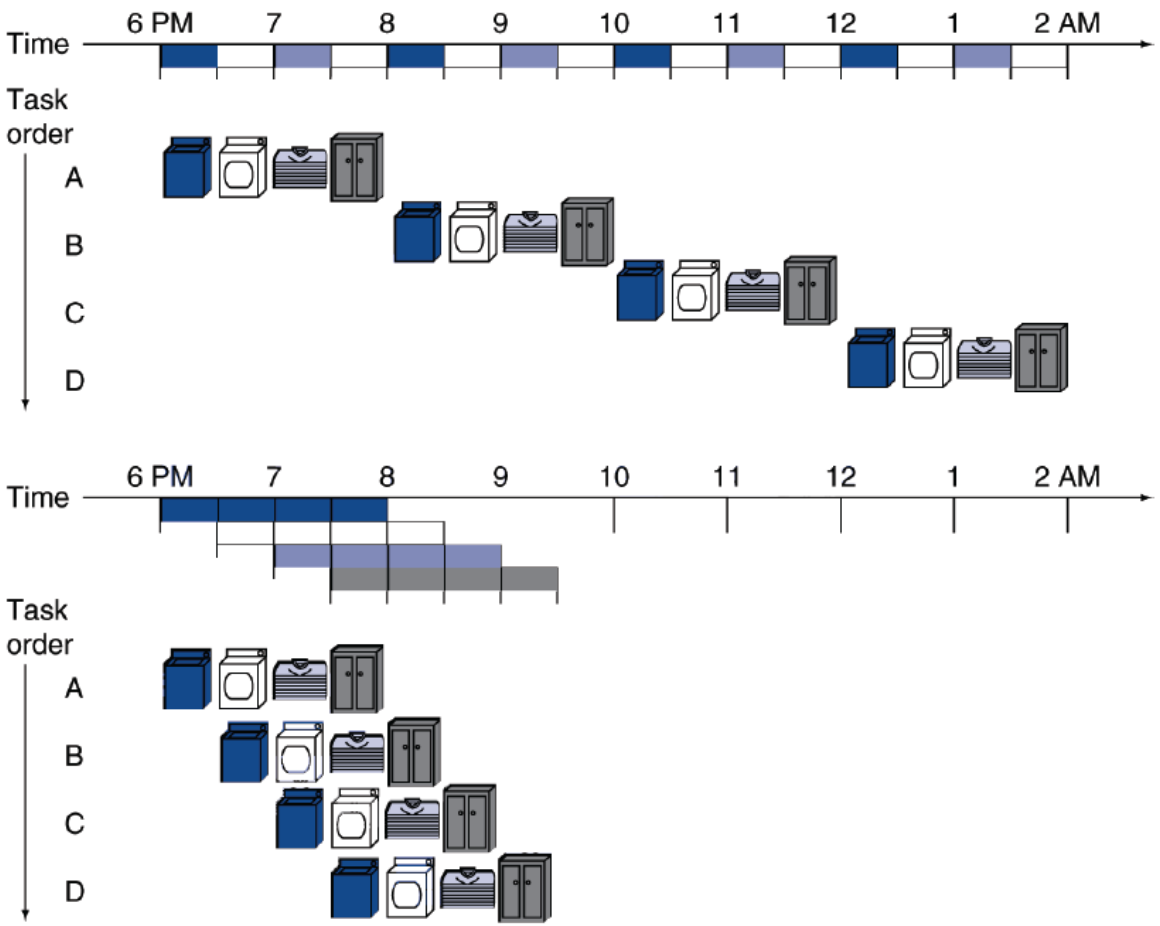
3. Preguntar al profesor lo que no quede claro

Presentación *Práctica 1*
2do ejercicio

Arquitectura de Computadores

Microprocesador segmentado

Segmentación. Reutilizar recursos



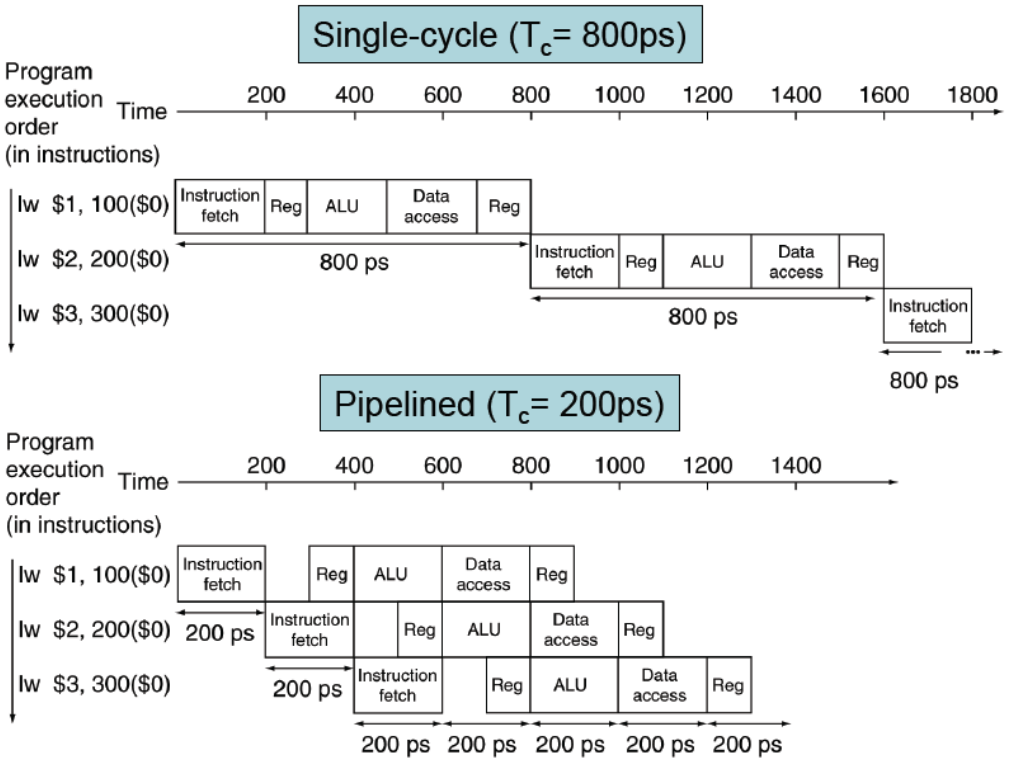
*D.A. Patterson y J.L. Hennessy “Computer Organization and Design: The Hardware/Software Interface” 4th edition.

Segmentación 5 etapas del MIPS

- Etapa IF (*Instruction Fetch*): Captura de instrucción.
- Etapa ID (*Instruction Decode*): Decodificación de instrucción.
- Etapa EX (*Execute*): Ejecución de instrucción (ALU) o cálculo de dirección para lw y sw.
- Etapa MEM (*Memory*): Lectura o escritura en memoria.
- Etapa WB (*Write Back*): Escritura en banco de registros

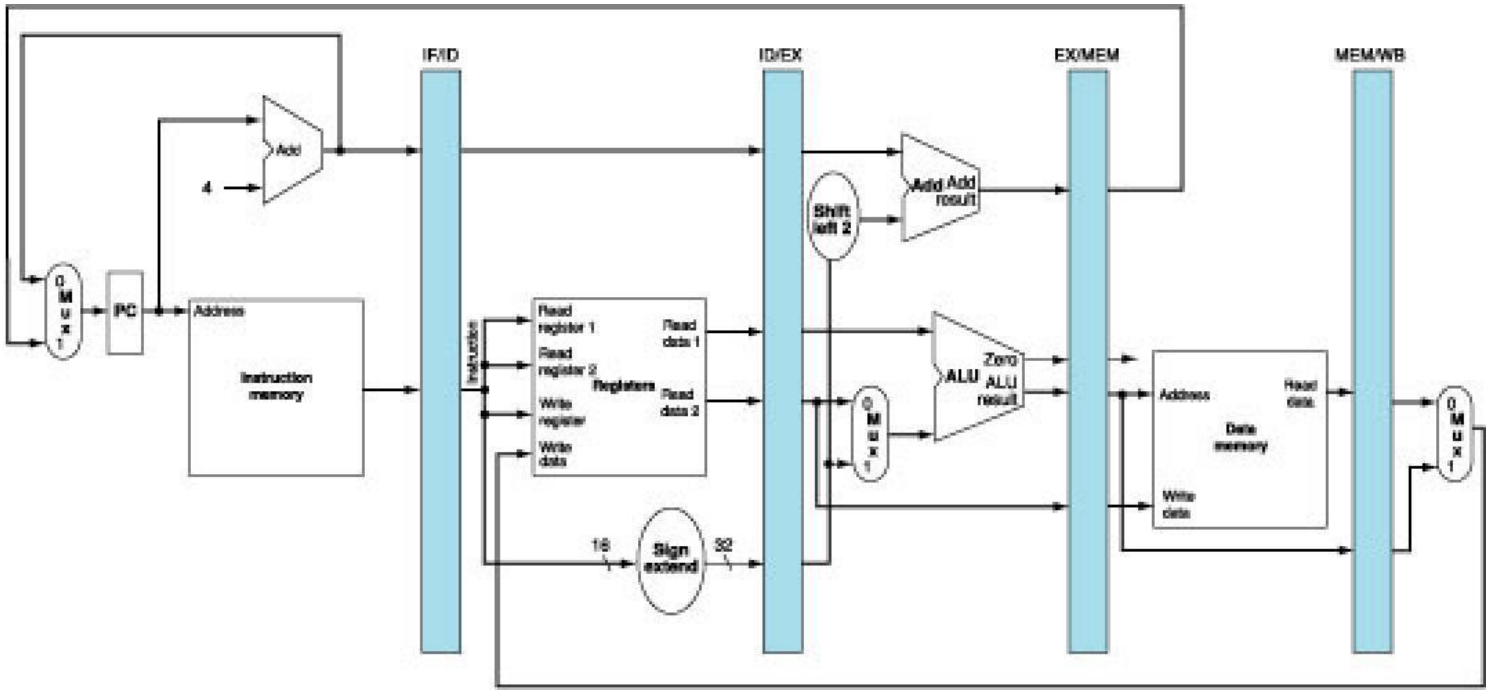
Segmentación en MIPS

IF	Captura instrucción.
ID	Decodificación instrucción.
	Lectura GPR.
	Cálculos saltos beq y jump.
EX	Ejecución.
	Cálculo dirección sw y lw.
MEM	Acceso memoria de datos.
WB	Escritura GPR.



*D.A. Patterson y J.L. Hennessy “Computer Organization and Design: The Hardware/Software Interface” 4th edition.

Segmentación en MIPS



Vista simplificada de la segmentación en 5 etapas

Separando etapas: crear registros

```
PC_counter: process(clk,reset)
begin
    if reset = '1' then
        pcmas4_ID <= (others=>'0');
        intruccion_ID <= (others=>'0');
    elsif rising_edge(clk) then
        pcmas4_ID <= pcmas4_IF;
        instrucción_ID <= instrucción_IF;
    end if;
end process;
```

¡Proceso síncrono!

Inicialización

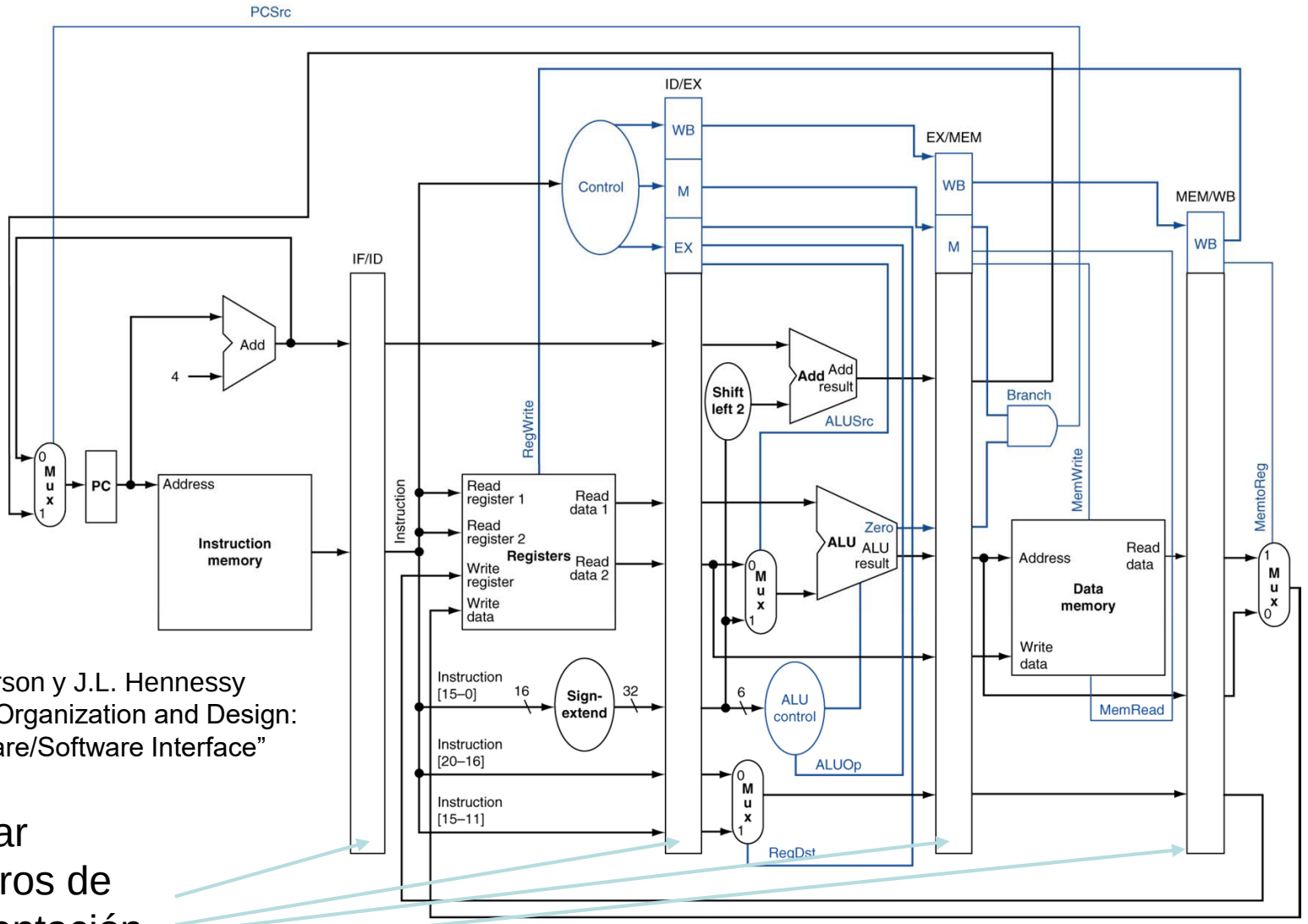
Propagación de valores

Separando etapas: crear registros con habilitación

```
PC_counter: process(clk,reset)
begin
    if reset = ‘1’ then
        pcmas4_ID <= (others=>’0’);
        intruccion_ID <= (others=>’0’);
    elsif rising_edge(clk) and enable_IF_ID=‘1’ then
        pcmas4_ID <= pcmas4_IF;
        instrucción_ID <= instrucción_IF;
    end if;
end process;
enable_IF_ID <= ‘1’;
```

En P1 nunca se para el pipeline, pero será necesario en P2

MIPS Segmentado (Pipelined)



*D.A. Patterson y J.L. Hennessy
“Computer Organization and Design:
The Hardware/Software Interface”

Agregar
Registros de
Segmentación

Recomendaciones

1. Documentar el código y usar nombres de señales descriptivos. Mejor seguir nomenclatura del libro

2. Hacer checkeos sintácticos y síntesis antes de simular.

   Si lo que habéis descripto no puede transformarse en HW difícilmente funcionará

3. Simular y entender que se está simulando

4. Este diseño es la base para la práctica 2. Asegurarse que realmente funciona.

5. Preguntar al profesor lo que no quede claro (para eso está allí)