

Presentación *Práctica 1*

Arquitectura de Computadores

3º de grado en Ingeniería Informática y

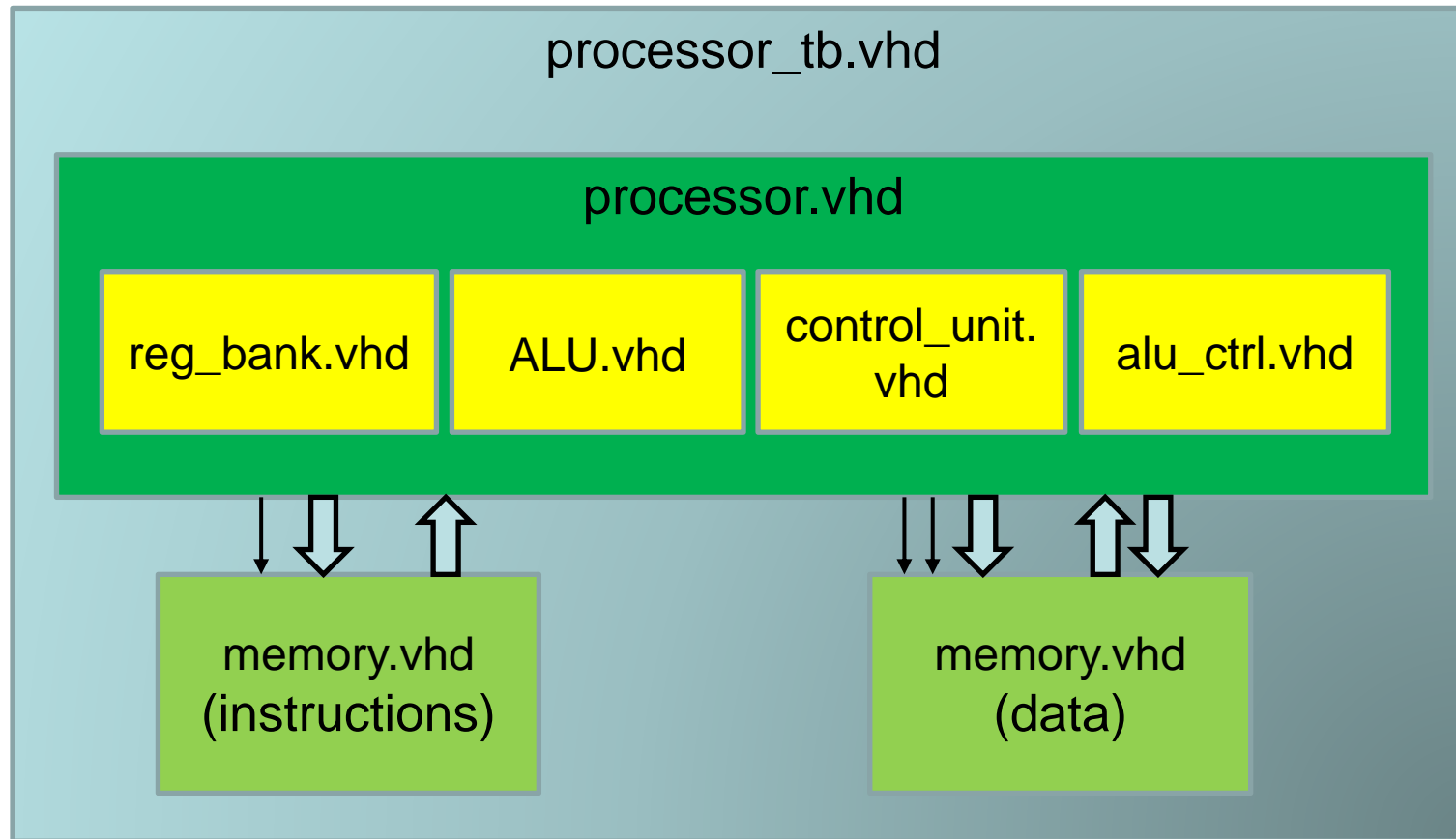
4º de doble grado en Ing. Informática y Matemáticas

Generalidades

- Entorno Linux o Windows
- Simulador Mentor ModelSim o Mentor QuestaSim
- Opcional: para editar usar Xilinx Vivado / Notepad++ / Atom / Sublime / Vim / ...
- En casa: Versión gratuita (**ModelSim PE Student Edition**) de Modelsim o Vivado Simulator

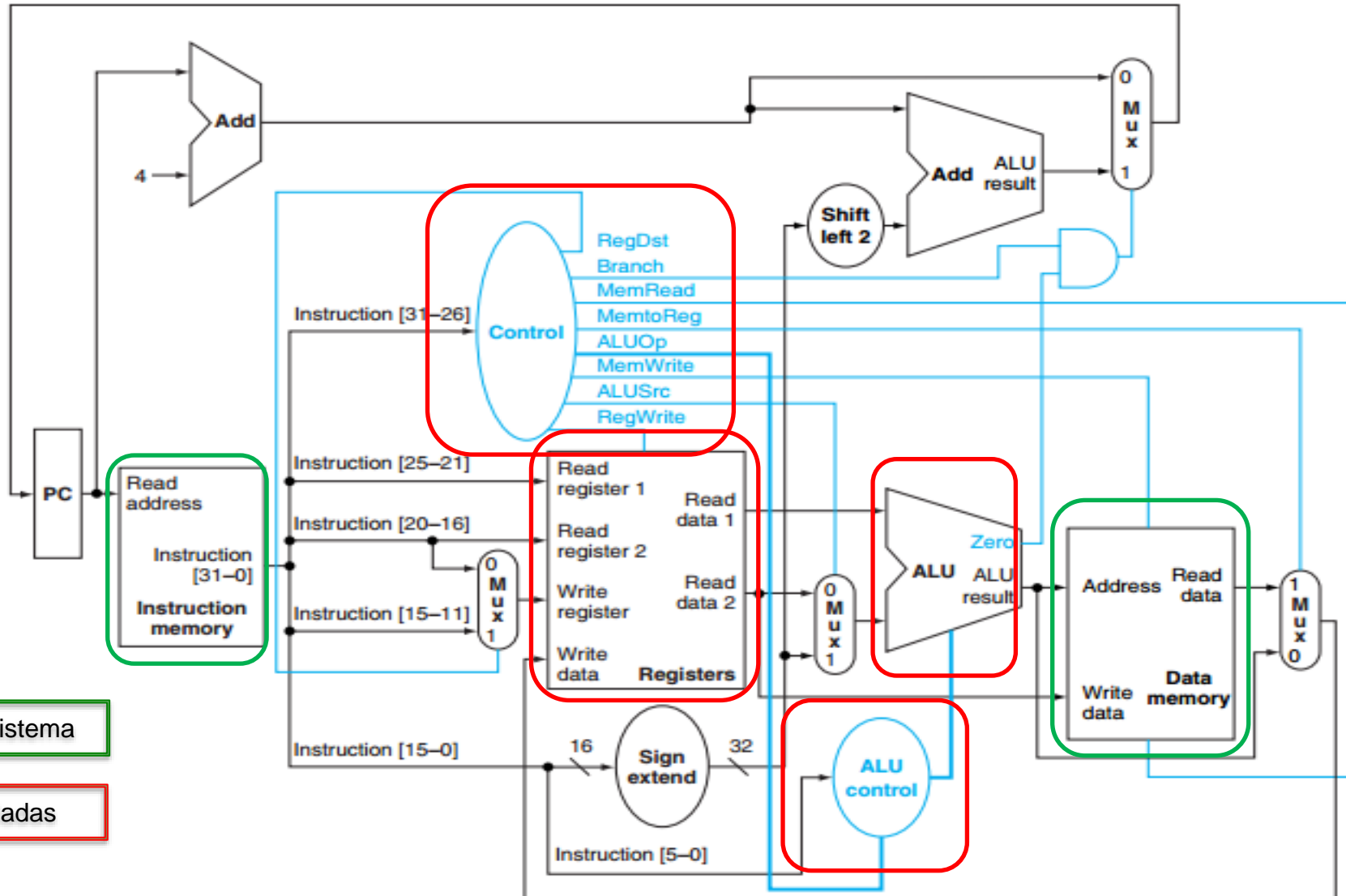


Diseño Jerárquico



Módulos que se proveen y han de ser desarrollados

MIPS uniciclo



Material Entregado

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

processor.vhd	procesador, versión simple uniciclo
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 para ModelSim

ALU (Arithmetic Logic Unit)

- Debe ser capaz de ejecutar el conjunto de instrucciones de esta versión reducida de MIPS que se implementarán
- Las señales de control las genera el bloque combinacional “**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)

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
- Aunque ya está desarrollado el modulo hay que verificar que se puedan ejecutar todas las instrucciones pedidas. Caso contrario habrá que agregarlas

Simular el sistema

- Se utilizará un simulador HDL
 - En el lab Questasim en Linux o ModelSim en Windows
 - Fuera de la EPS, ModelSim versión “student” o VivadoSimulator
- Para simular el sistema es necesario
 - Código del procesador (y sus componentes)
 - Testbench (procesor_tb.vhd y modelos de las memorias de instrucciones y datos memory.vhd)
 - Contenido de las memorias (ficheros de texto plano “instrucciones” y “datos”). Ver como generar a continuación.

Simulación: generar instrucciones y datos

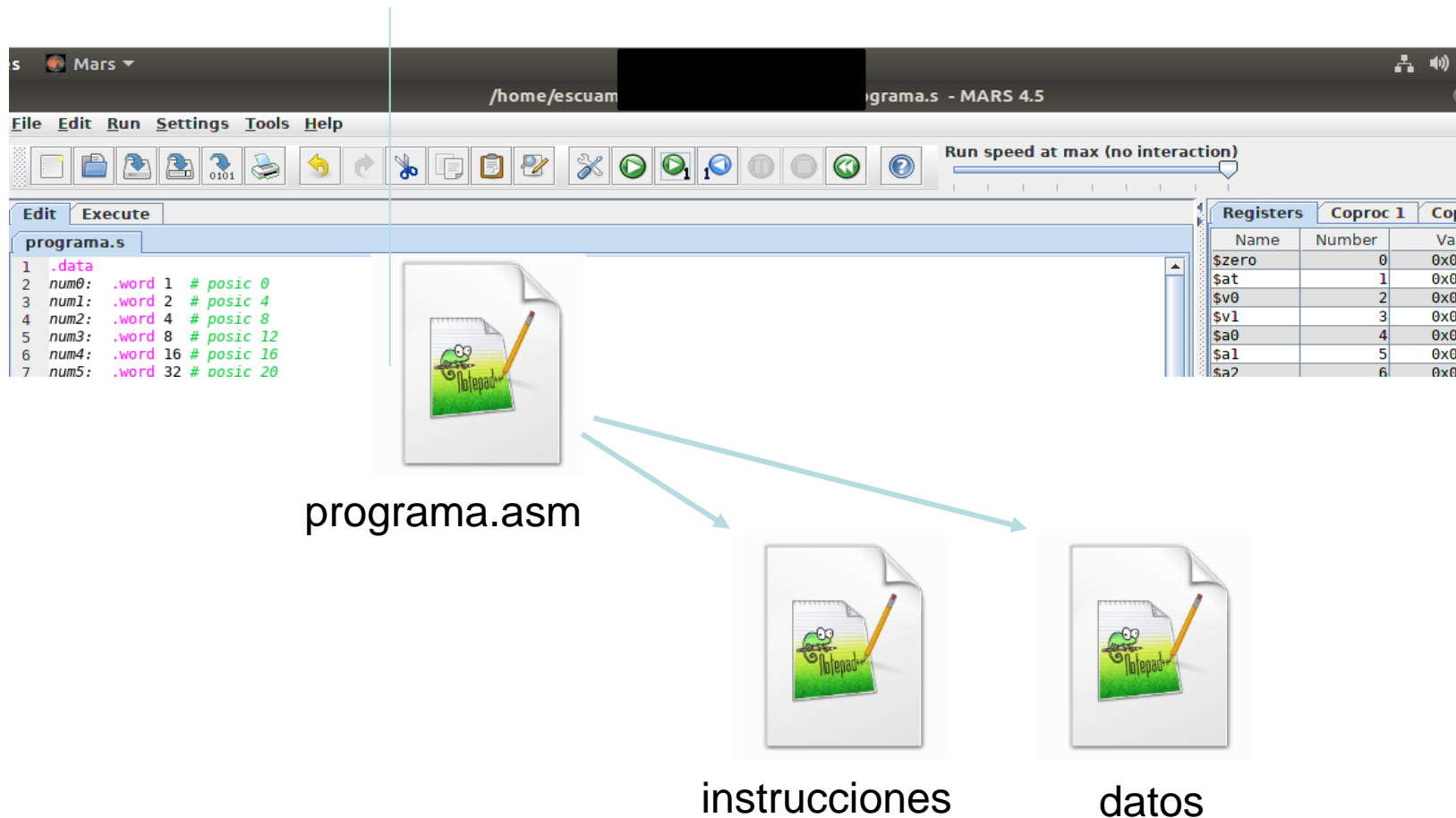
En Windows:

- Ejecutar el fichero “arqo_comp.bat”
- Requiere “programa.asm” de entrada
- Genera “instrucciones” y “datos” de salida

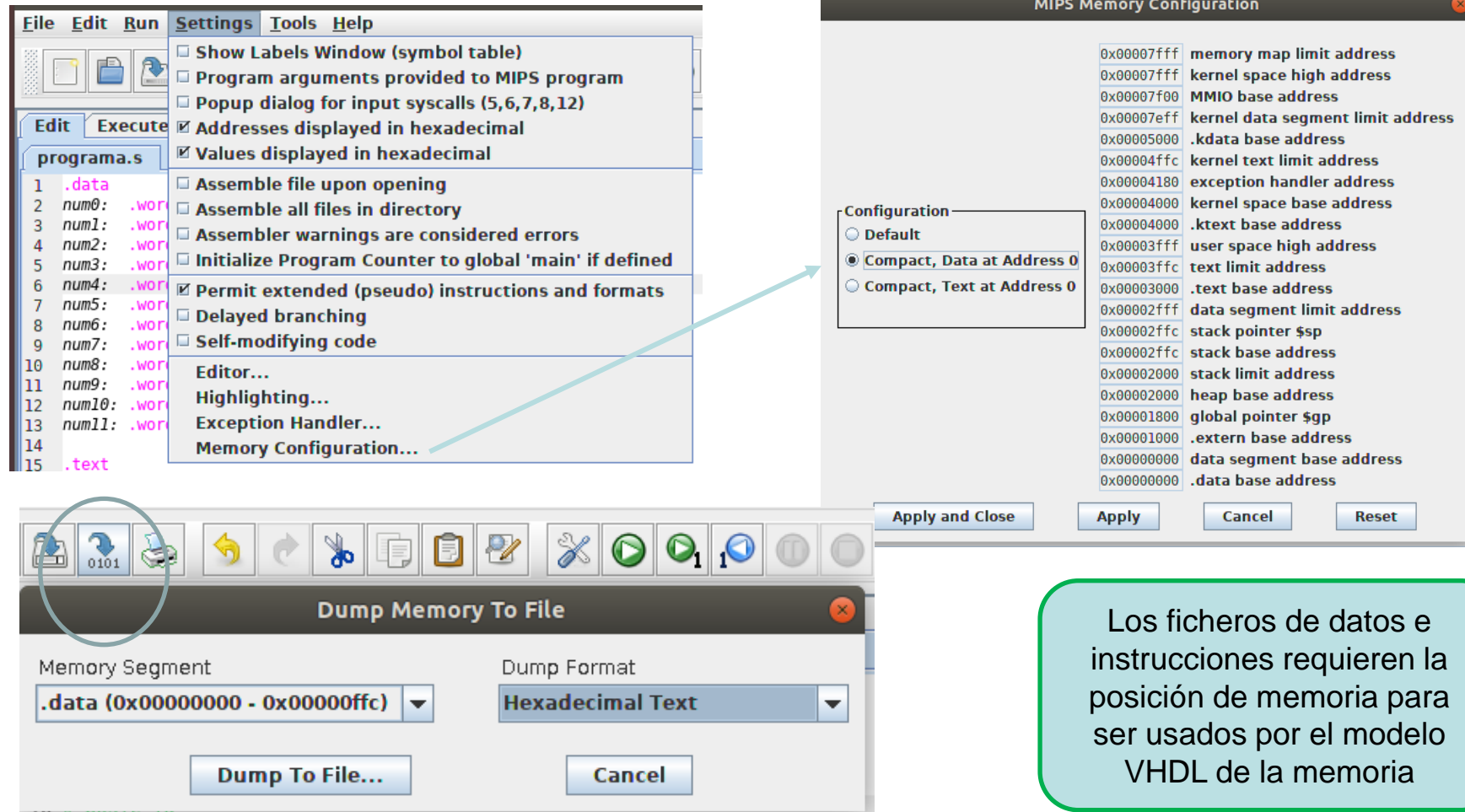


Simulación: generar instrucciones y datos

En Linux (o Windows) también se puede usar el simulador **MARS** (*MIPS Assembler and Runtime Simulator*) u otro compilador online



MARS como compilador

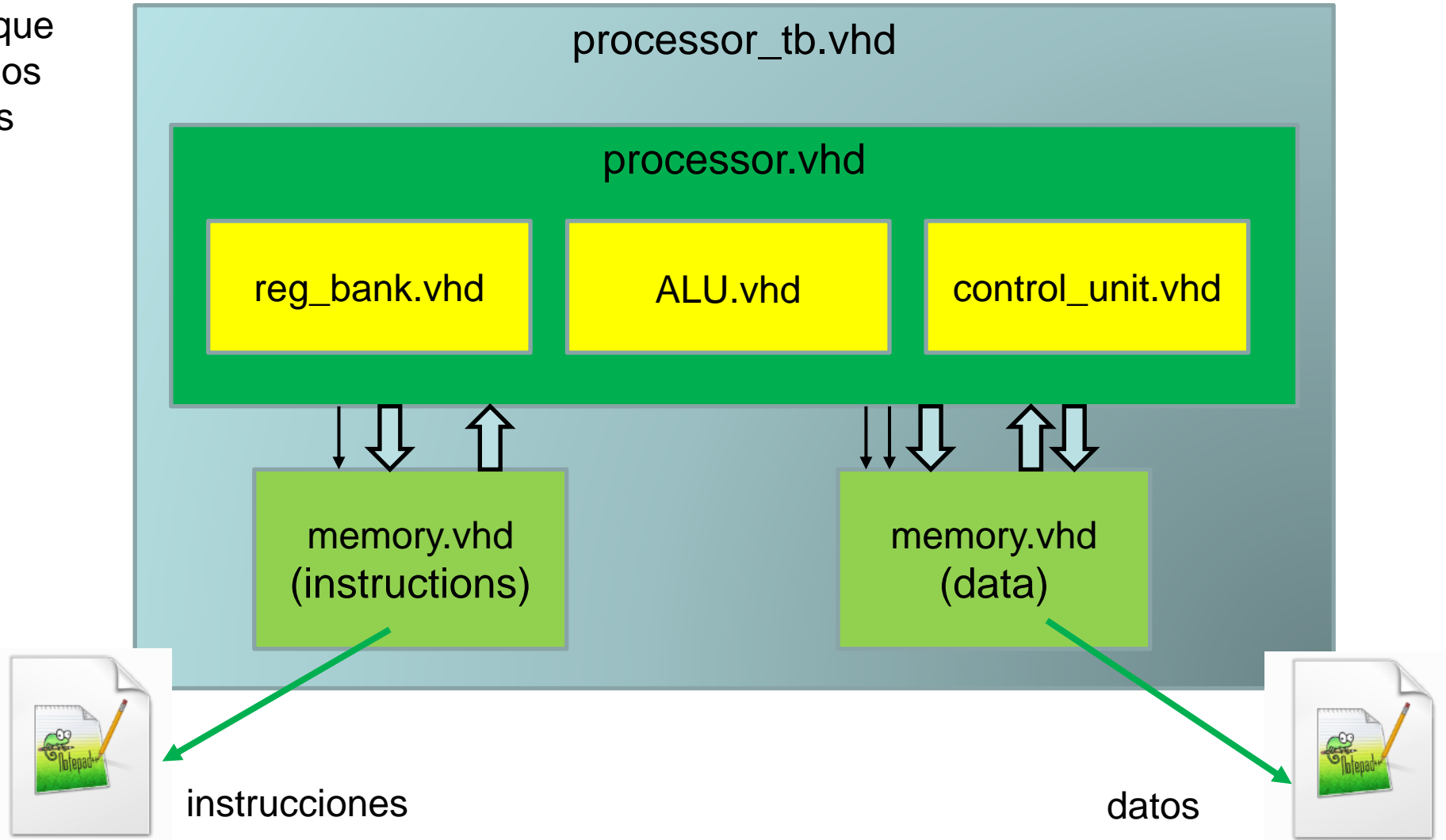


También se pueden generar por línea de comandos

```
java -jar Mars4_5.jar a dump .data HexText hexdata.txt arq0_p1/sim/programa.s
java -jar Mars4_5.jar a dump .text HexText hextext.txt arq0_p1/sim/programa.s
```

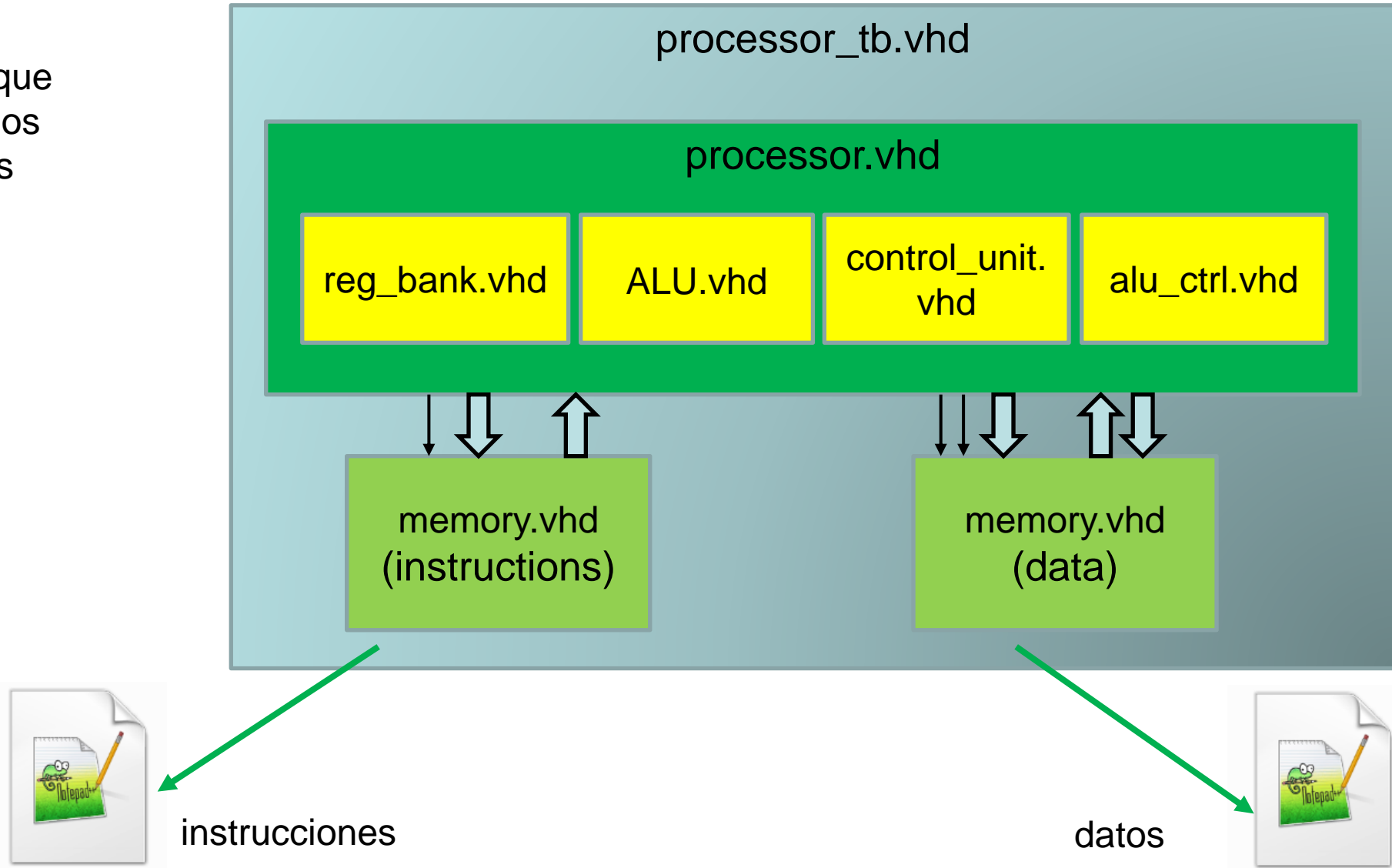
Simulación

Asegurarse que
están todos los
componentes
descriptos



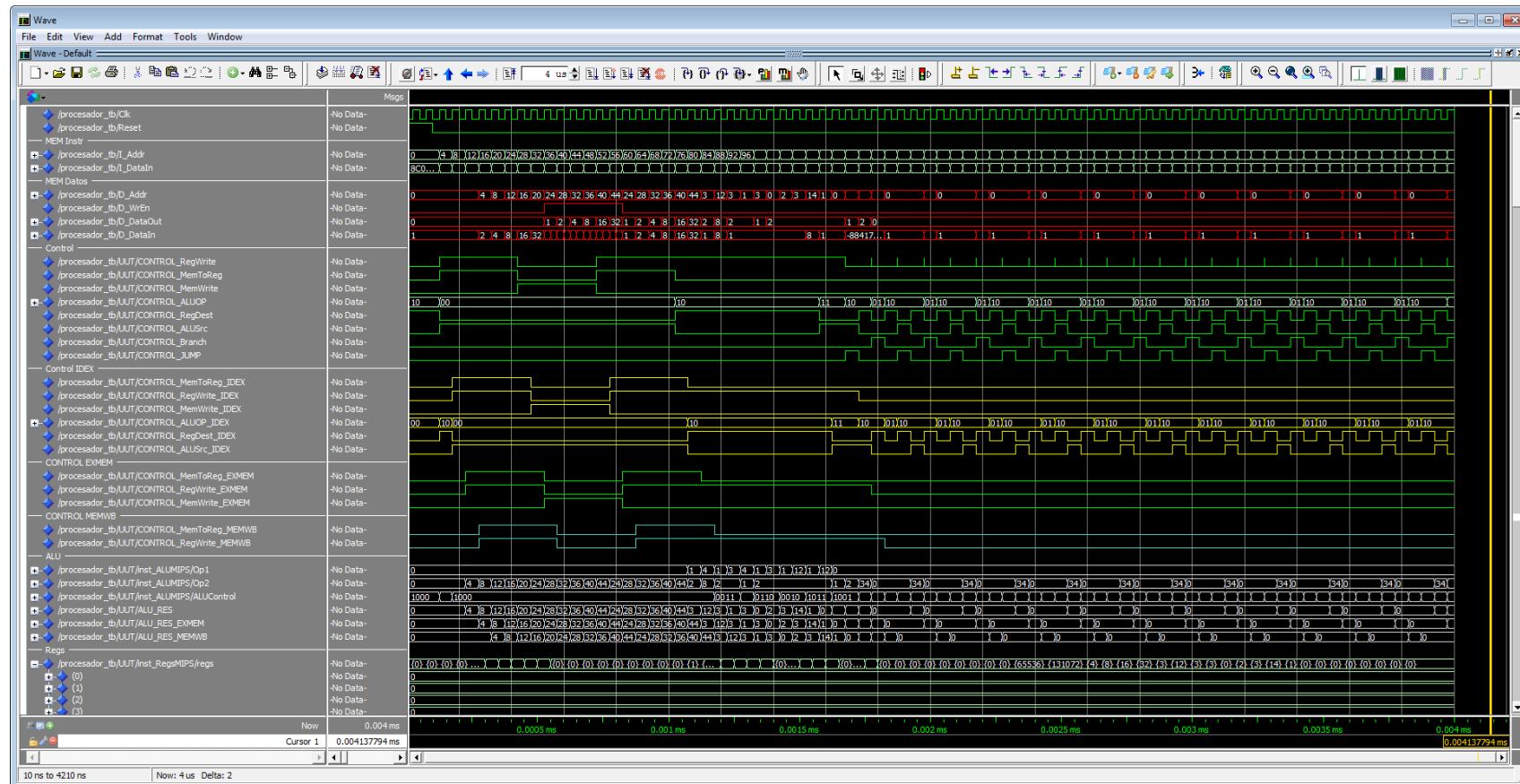
Simulación

Asegurarse que
están todos los
componentes
descriptos



Simulación

Lanzar el script de simulación “do runsim_Arq.do” y analizar el contenido de la forma de ondas.
Recuerda de guardar la forma de ondas para futuras simulaciones



Consejos de desarrollo

- Codificación
 - No crear nuevos componentes. Multiplexores y registros se pueden describir con procesos o asignaciones concurrentes
- EDITOR: Si usas Xilinx ISE o Xilinx Vivado
 - Check syntax (Detección de errores de VHDL)
 - Synthesize (Detección de errores “HW” (latches, ...))
- Simulador: Mentor ModelSim/QuestaSim
 - Desde el directorio sim, en la consola del simulador **do runsim_Arq.do**
 - **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

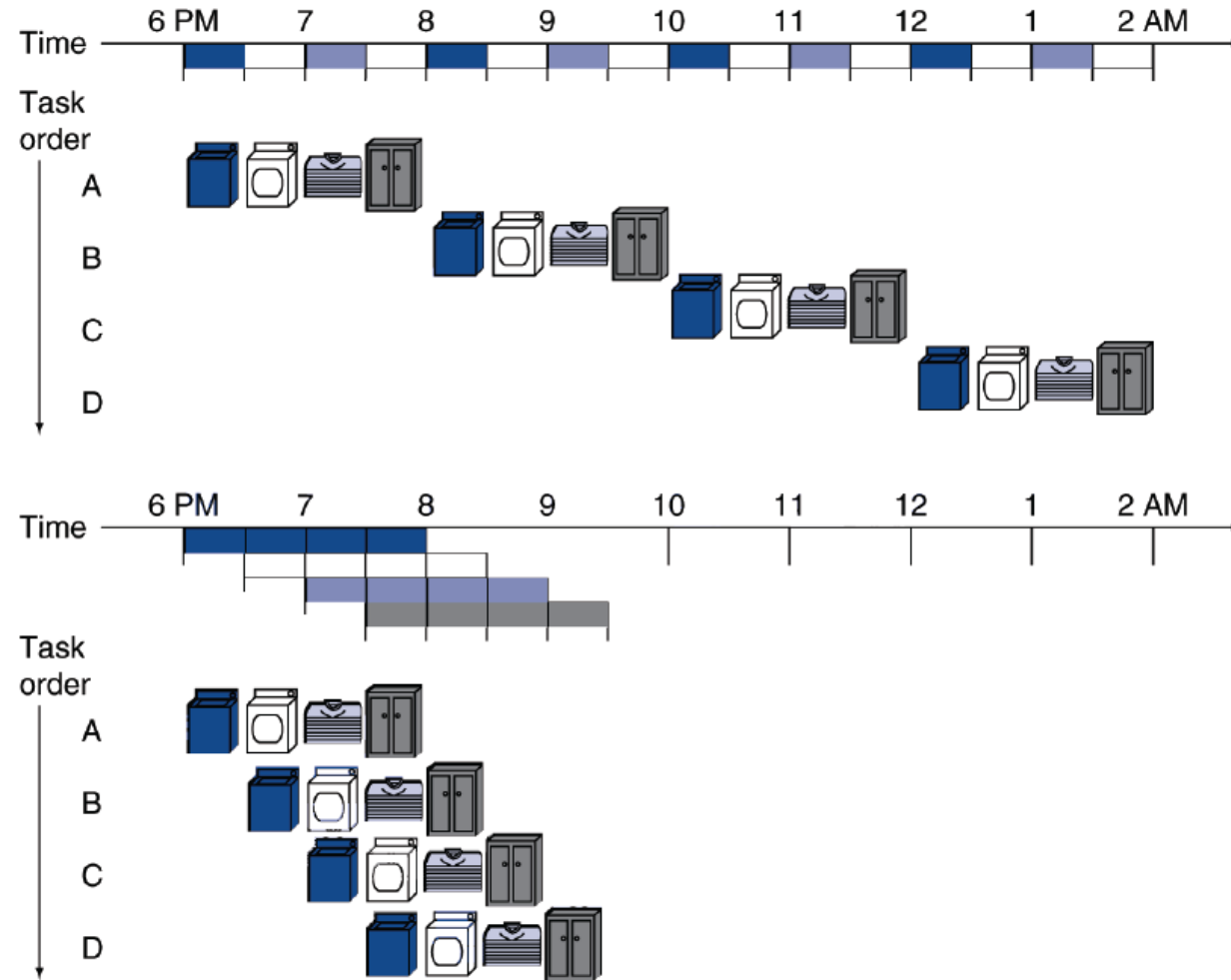


Presentación *Práctica 1*
3^{er} ejercicio

Arquitectura de Computadores

Microprocesador segmentado

Segmentación. Reutilizar recursos

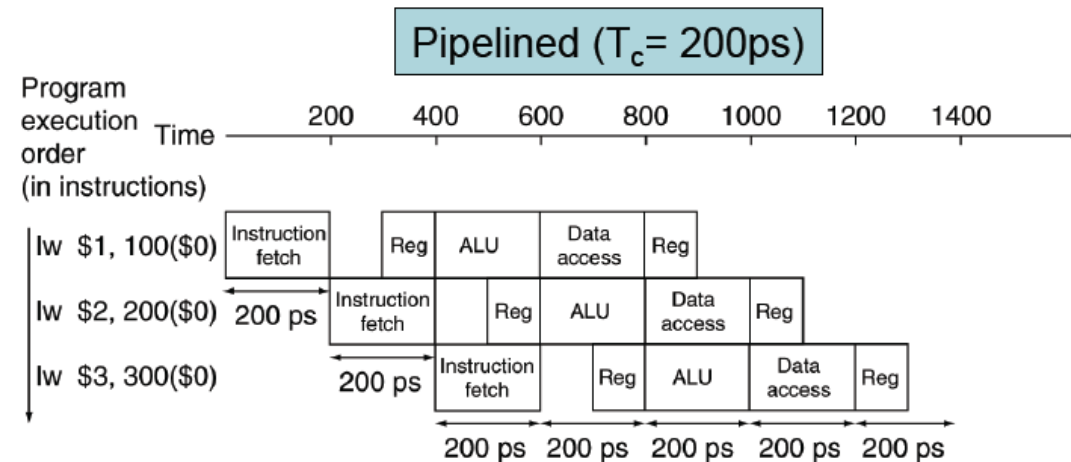
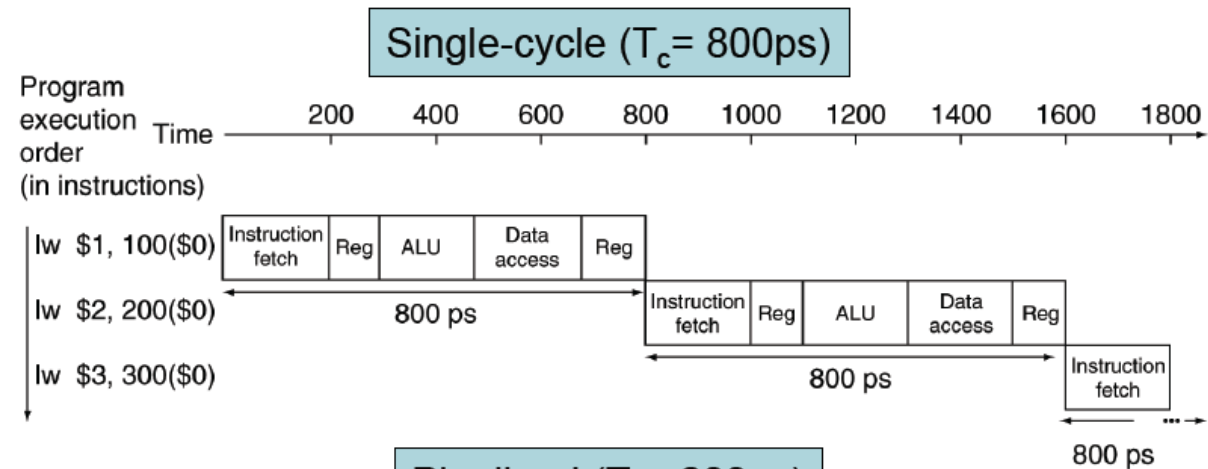


Segmentación 5 etapas del MIPS

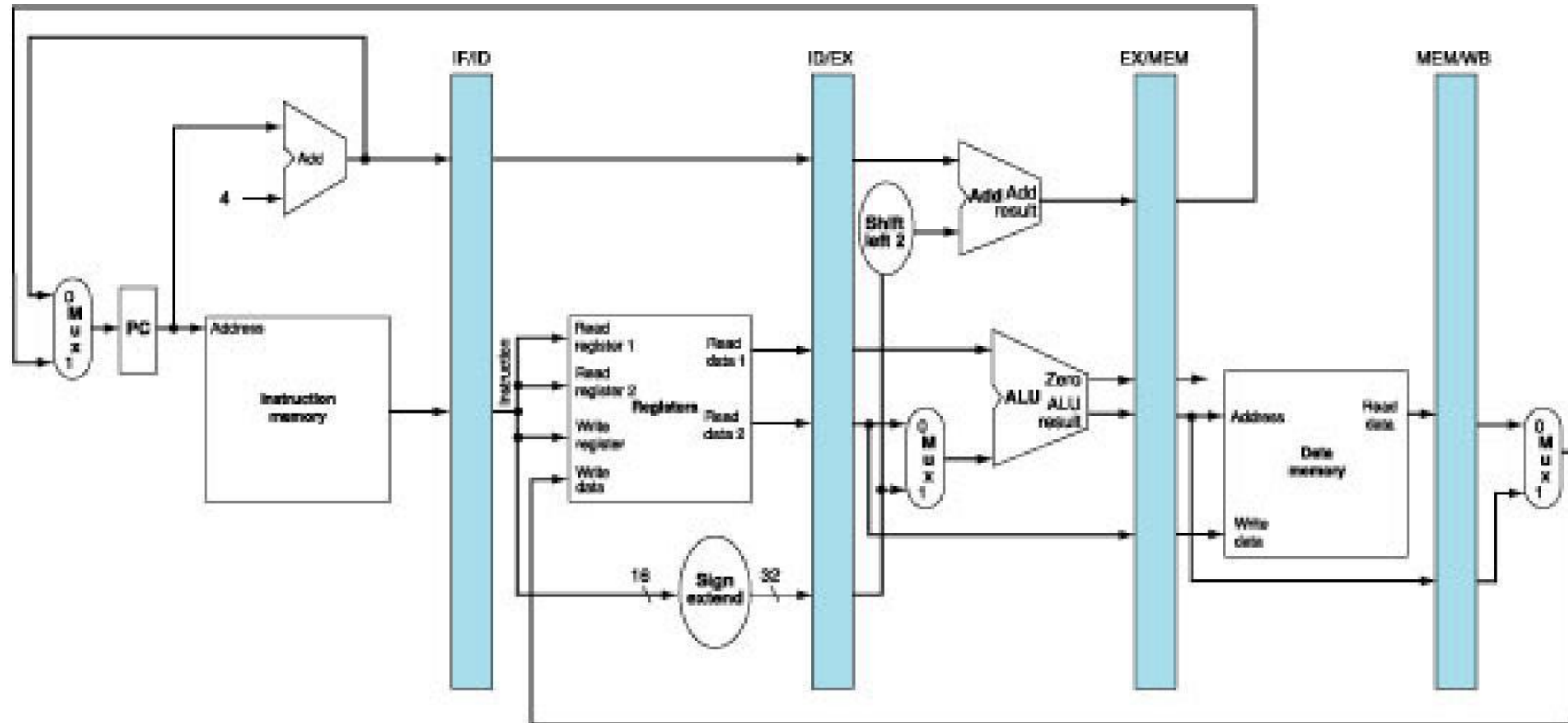
- Etapa IF (*Instruction Fetch*): Captura de instrucción.
- Etapa ID (*Instruction Decode*): Decodificación de instrucción.
- Etapa EX (*Execute*): Ejecución de instrucción (ALU) o cálculo de dirección para lw y sw.
- Etapa MEM (*Memory*): Lectura o escritura en memoria.
- Etapa WB (*Write Back*): Escritura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



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');  
    instruccion_ID <= (others=>'0');
```

```
  elsif rising_edge(clk) then
```

```
    pcmas4_ID <= pcmas4_IF;  
    instruccion_ID <= instruccion_IF;
```

```
  end if;
```

```
end process;
```

¡Proceso
síncrono!

Inicialización

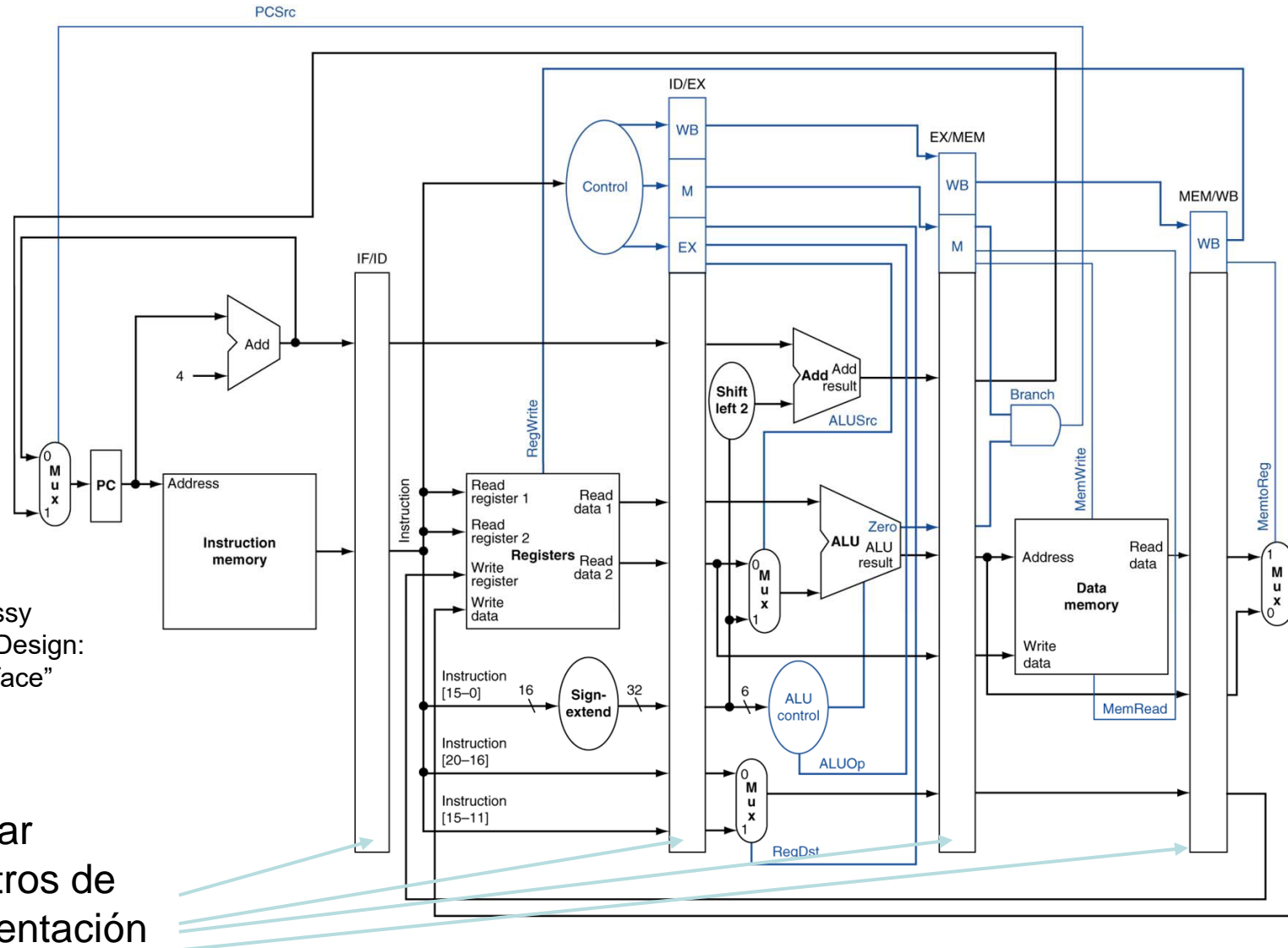
Propagación de valores

Separando etapas: crear registros con habilitación

```
PC_counter: process(clk,reset)
begin
    if reset = '1' then
        pcmas4_ID <= (others=>'0');
        instruccion_ID <= (others=>'0');
    elsif rising_edge(clk) and enable_IF_ID='1' then
        pcmas4_ID <= pcmas4_IF;
        instruccion_ID <= instruccion_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 chequeos sintácticos y síntesis antes de simular (si usas Vivado).
Si lo que habéis descrito 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)