Introducción *Práctica 1*

Simular el procesador RISC-V en versión uniciclo y segmentado

Curso 2024-2025

Arquitectura de Computadores

3º de grado en Ingeniería Informática y

3º de doble grado en Ing. Informática y Matemáticas

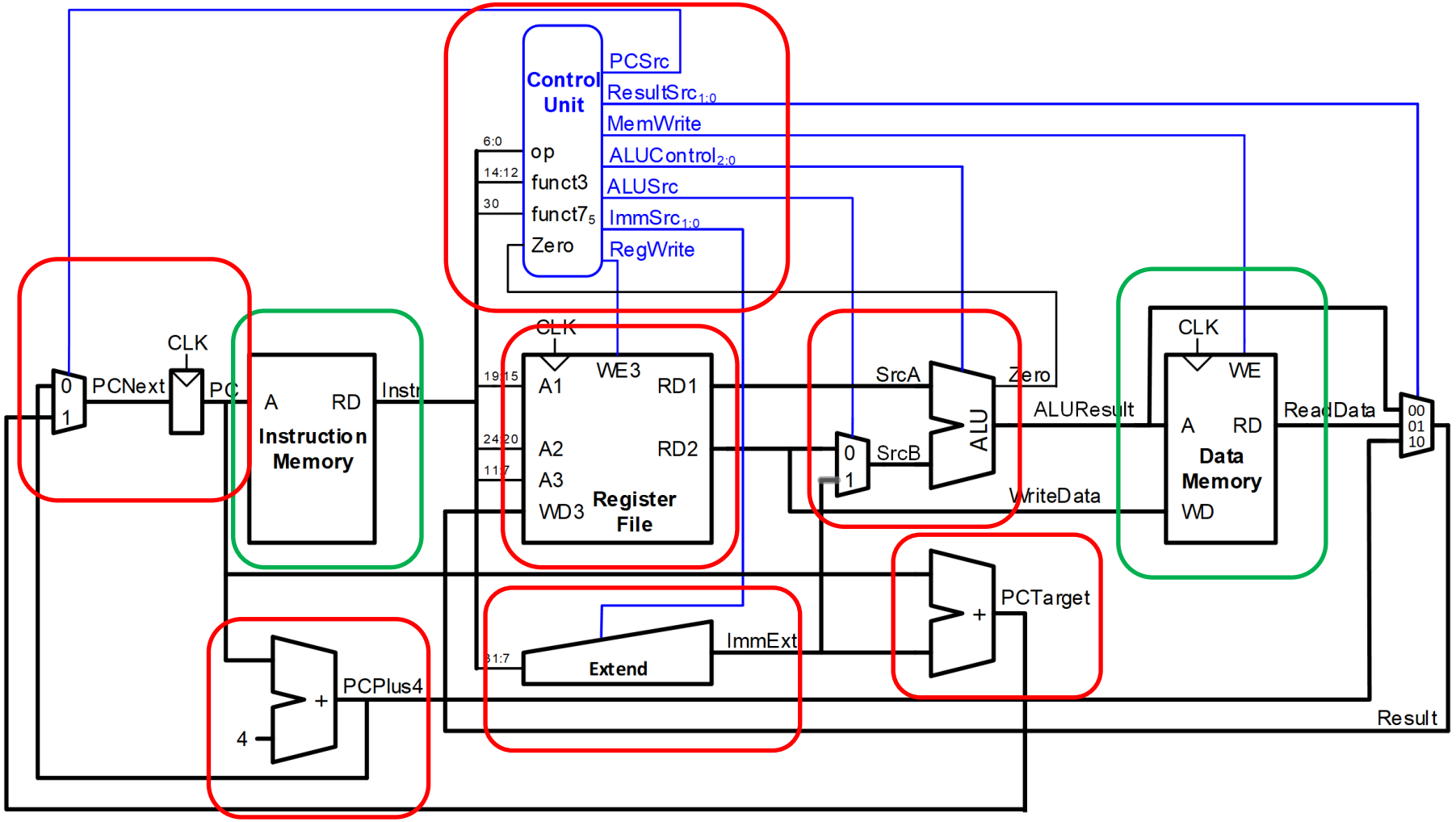
Generalidades

- Simular el procesador RISC-V en versión uniciclo/segmentado
- Simulador Siemens EDA (Mentor) QuestaSim
- Entorno Linux
- Ensamblar código con RARS

- Opcional: para editar VHDL usar Visual Studio Code / Xilinx Vivado / Notepad++ / Atom / Sublime / Vim /…
- En casa: Versión gratuita (**Intel FPGA**) de QuestaSim o Vivado Simulator o Máquina Virtual provista por la asignatura

RISC-V uniciclo

(figura libro Harris usado en EC (estructura Computadores))

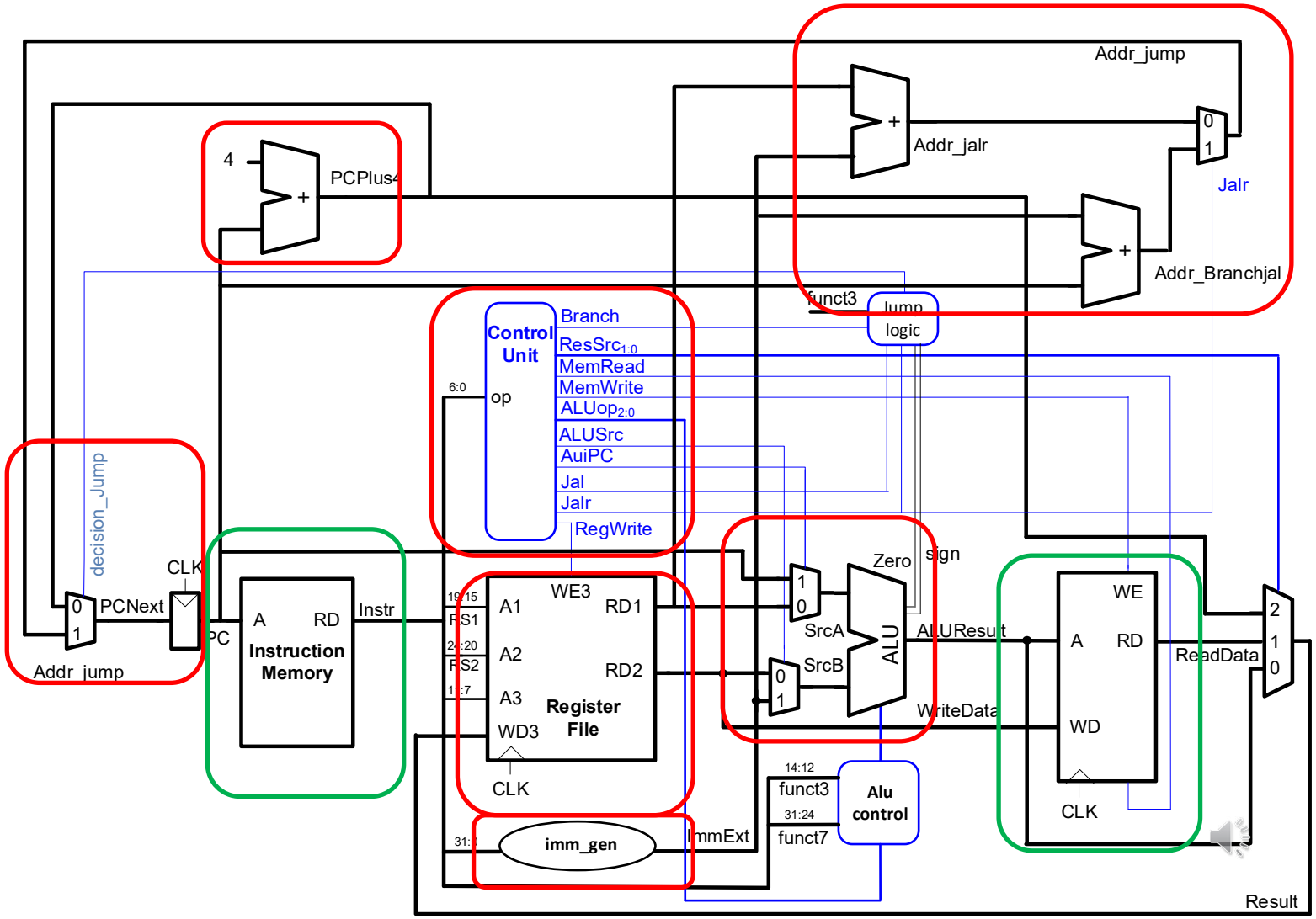


Entidades externas al procesador

Entidades (módulos) separados

RISC-V uniciclo (Arquitectura de Ordenadores 2024)

- Es el mismo procesador RISC-V en versión uniciclo.
- Existen algunas diferencias menores dado que implementa más instrucciones que el usado en EC.
- Algunas diferencias:
  - Nombre de las señales (cables) diferentes
  - Unidad de control. Separada en 3 partes: Control Unit + Alu Control + Jump Logic
  - Multiplexor extra en la entrada SrcA de la ALU para soportar instrucción AuiPC (add upper immediate to PC)
  - El módulo imm_gen (generador del operando inmediato) antes “extend” coje la instrucción completa

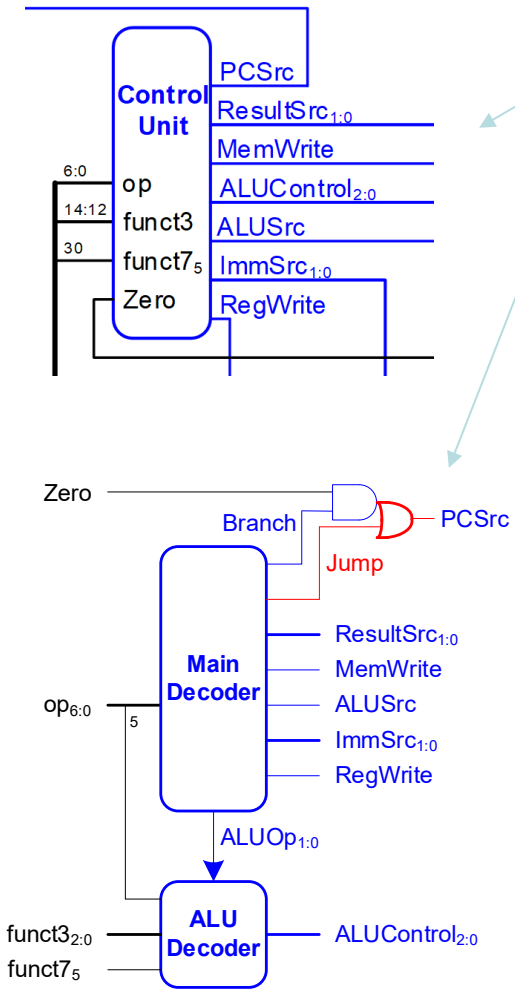


Entidades externas al procesador

Entidades (módulos) separados

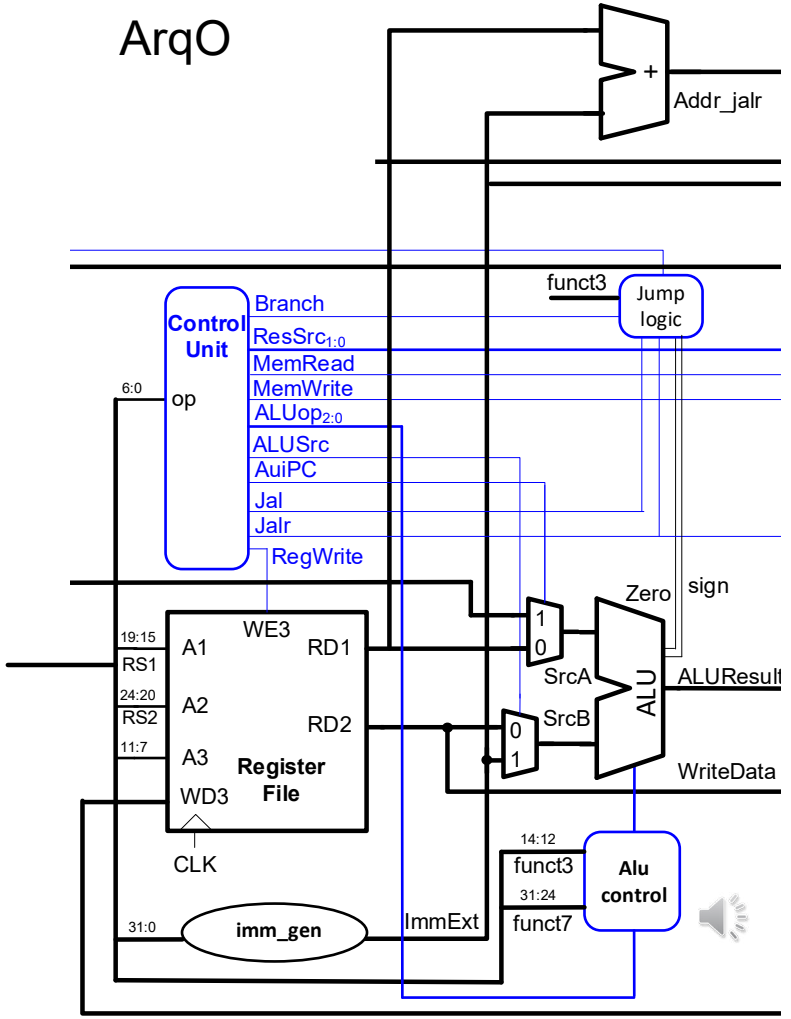
RISC-V uniciclo (unidad de control)

EC



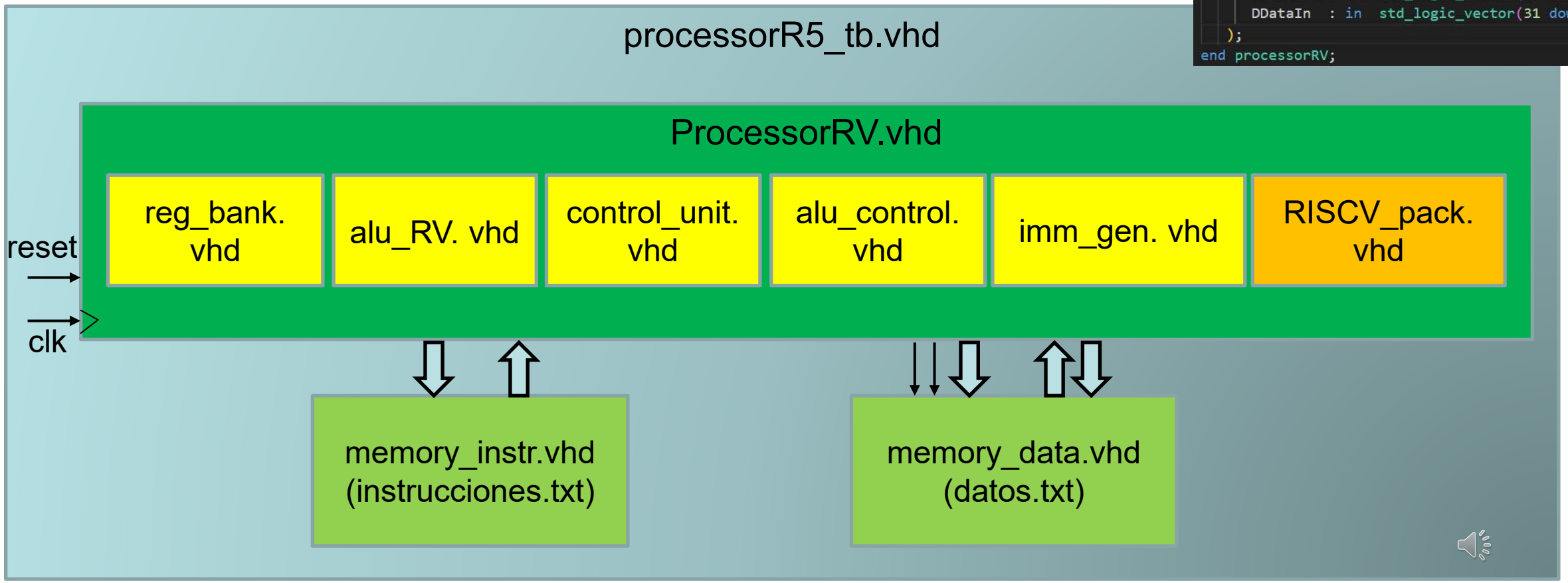
- En la implementación de EC la unidad de control tiene 3 bloques que están dentro de la “control Unit”
- En ArqO las tres partes están por separado (simplifica conceptos en la segmentación)
  - Control Unit. Decodifica instrucción basado en los 7 bits del código de operación (codop)
  - Alu control. En función de ALUOP y func3 funt7 determina operación de la ALU
  - Jump Logic

ArqO



Diseño Jerárquico

```
entity processorRV is
    port(
        Clk      : in  std_logic;
        Reset    : in  std_logic;
        -- Instruction memory
        IAddr    : out std_logic_vector(31 downto 0);
        IDataIn  : in  std_logic_vector(31 downto 0);
        -- Data memory
        DAddr    : out std_logic_vector(31 downto 0);
        DRdEn    : out std_logic;
        DWrEn    : out std_logic;
        DDataOut : out std_logic_vector(31 downto 0);
        DDataIn  : in  std_logic_vector(31 downto 0)
    );
end processorRV;
```



RISC-V pipeline

- Versión pipeline (segmentada) sin control de riesgos
  - Usa la misma unidad de control
  - Las señales de control se van descartando conforme avanzan en el pipeline

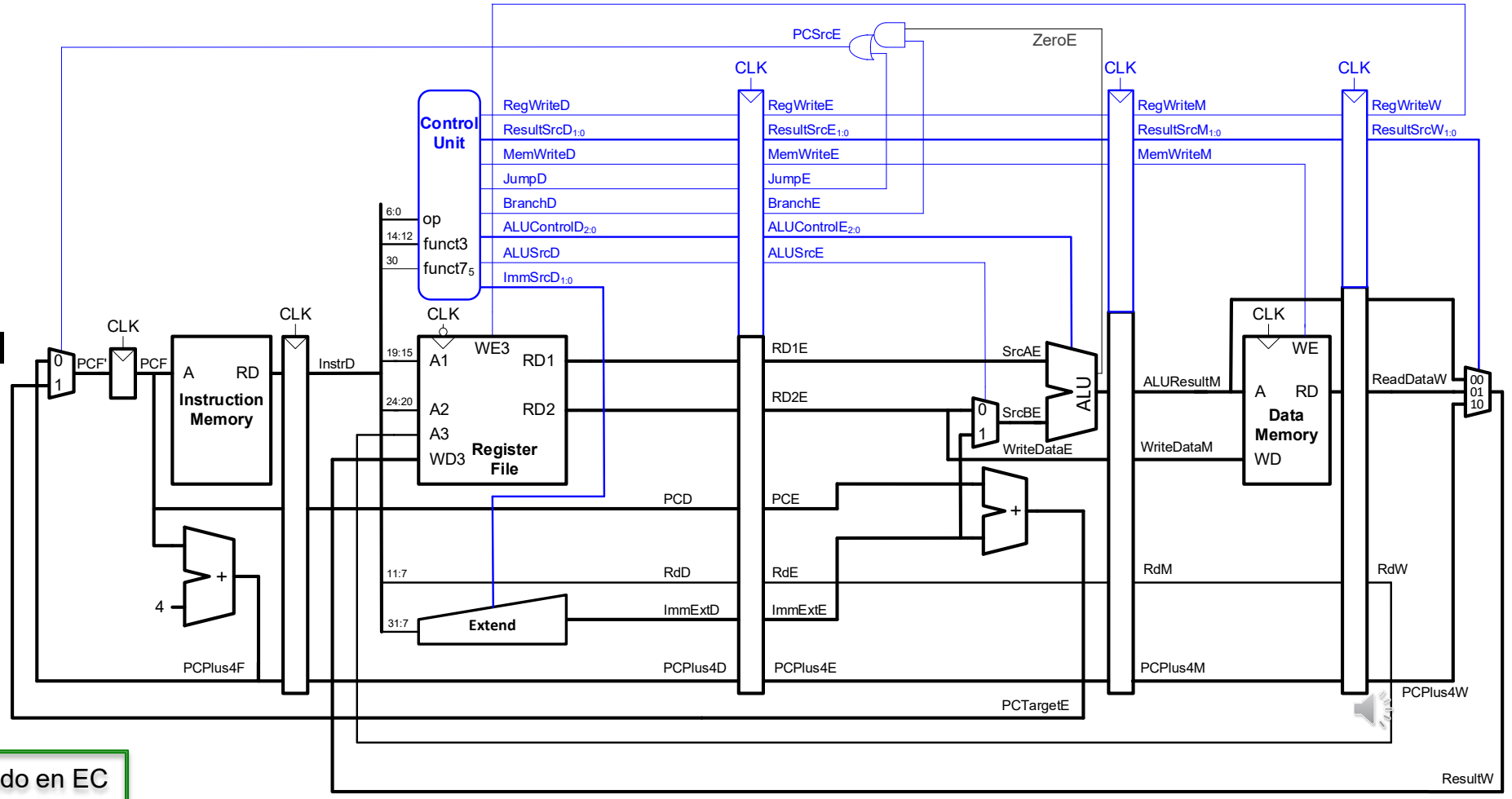
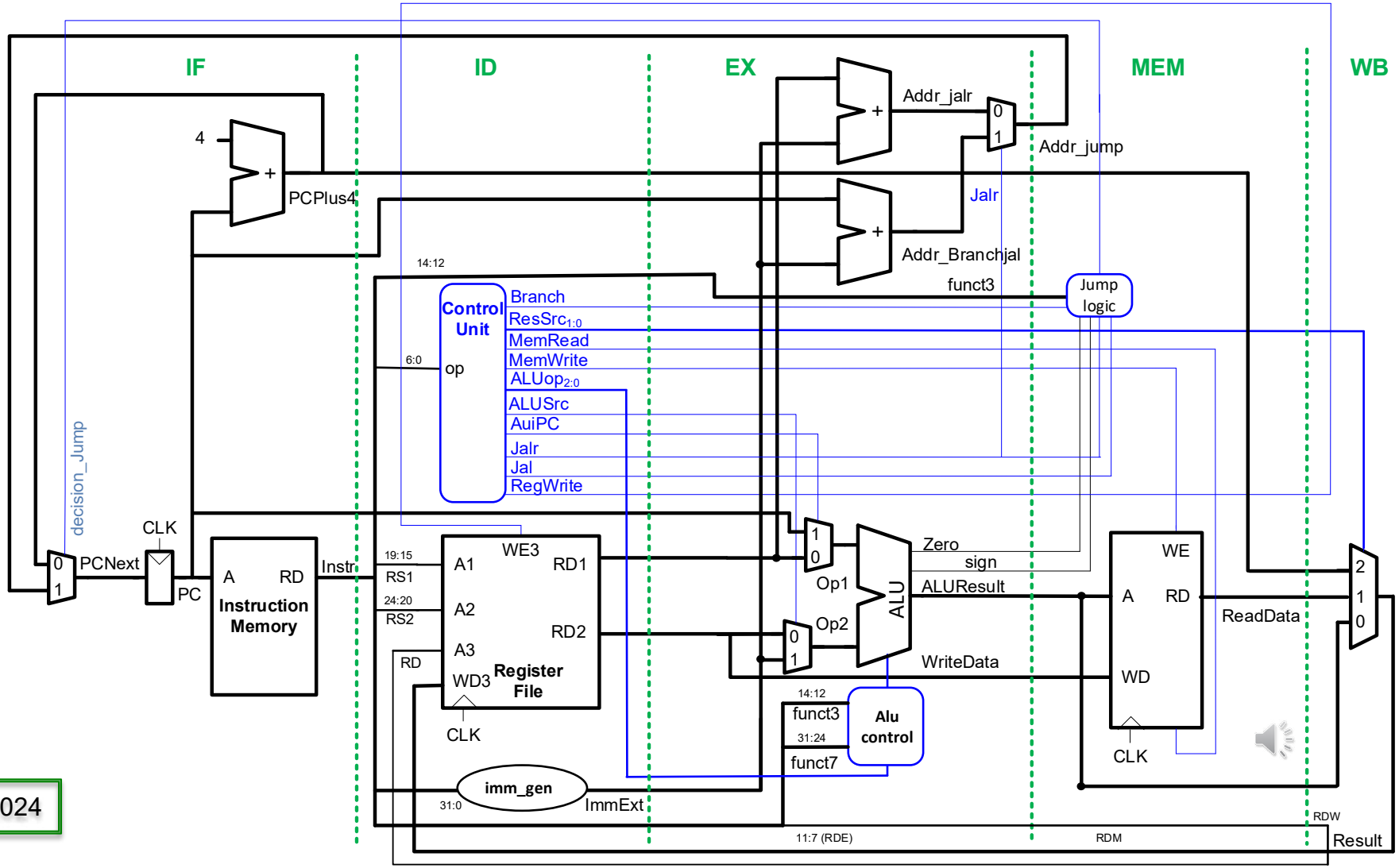


Figura Harris usado en EC

RISC-V pipeline

- Algunas diferencias
  - Líneas verdes indican “líneas equitempo-rales” donde se colocan registros FF
  - Unidad de control dividida en 3 partes. Control Unit (ID), Alu control (EX) y Jump Logic (MEM)
  - El salto se resuelve en la etapa MEM

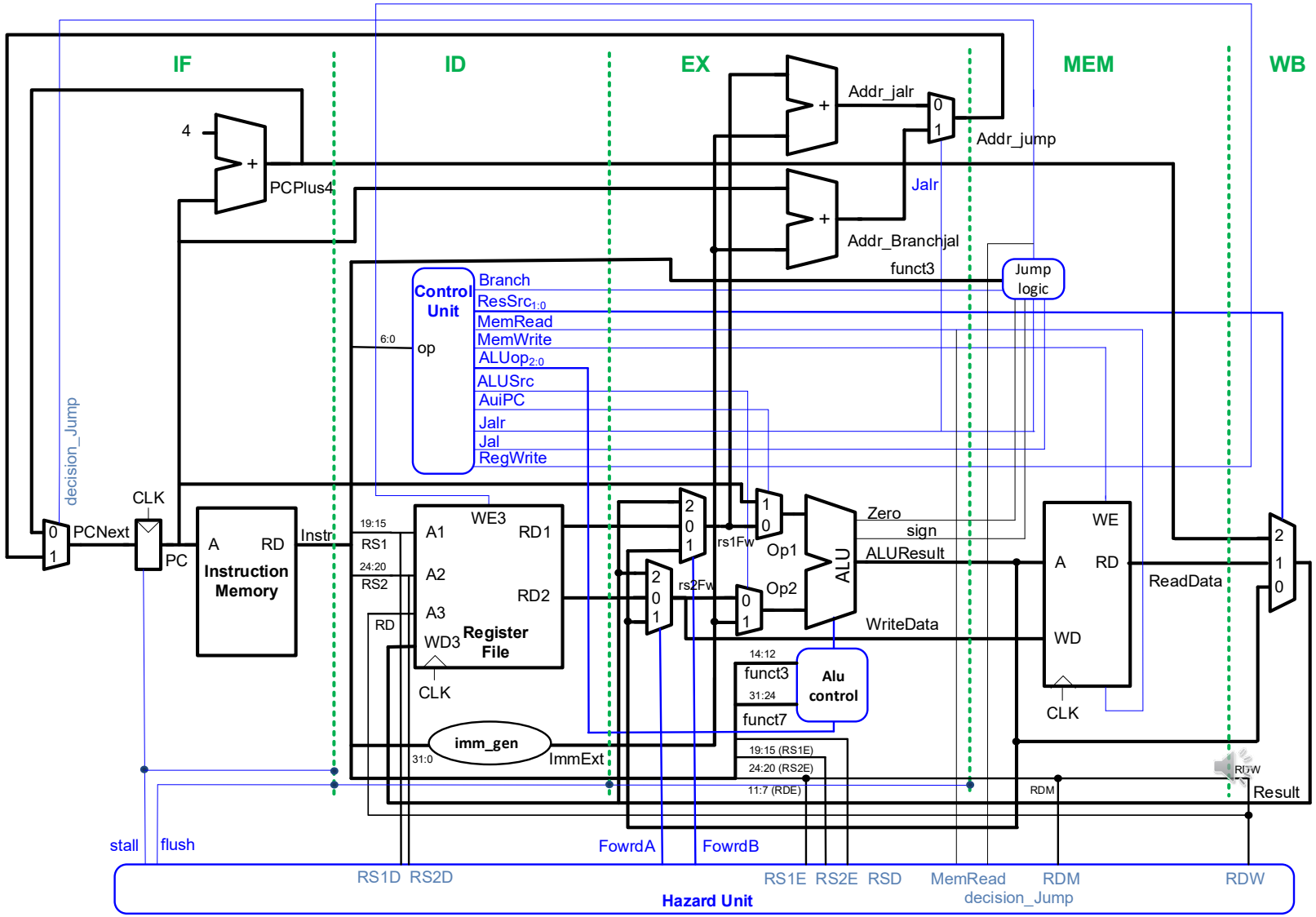


Versión usada en ArqO 2024

Objetivo de la Práctica:

RISC-V pipeline con soporte de riesgos

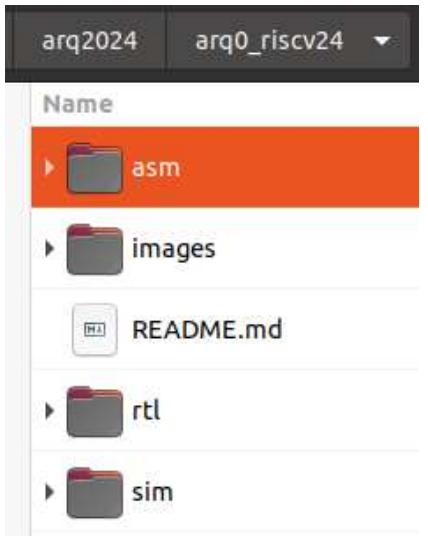
- Dotar al procesador RISC-V de la lógica de resolución de riesgos de datos y de control.
  - Adelantamiento de datos
  - Riesgos del tipo Load-Use
  - Ejecutar correctamente los saltos (quitar del pipeline las instrucciones que no se deben ejecutar)



Versión usada en ArqO 2024

Material de Partida

- Se entrega un fichero zip que contiene la descripción VHDL del procesador RISC-V uniciclo y segmentado (pipeline)
  - El directorio ***rtl/*** : contiene el código del procesador
  - Directorio ***sim/*** : contiene lo necesario para simular en QuestaSim
  - En ***asm/*** ejemplos de código ensamblador para RARS
  - En ***images/*** gráficos (en pdf) de las rutas de datos de la versión uniciclo, pipeline sin soporte de riesgo y pipeline con soporte de riesgos



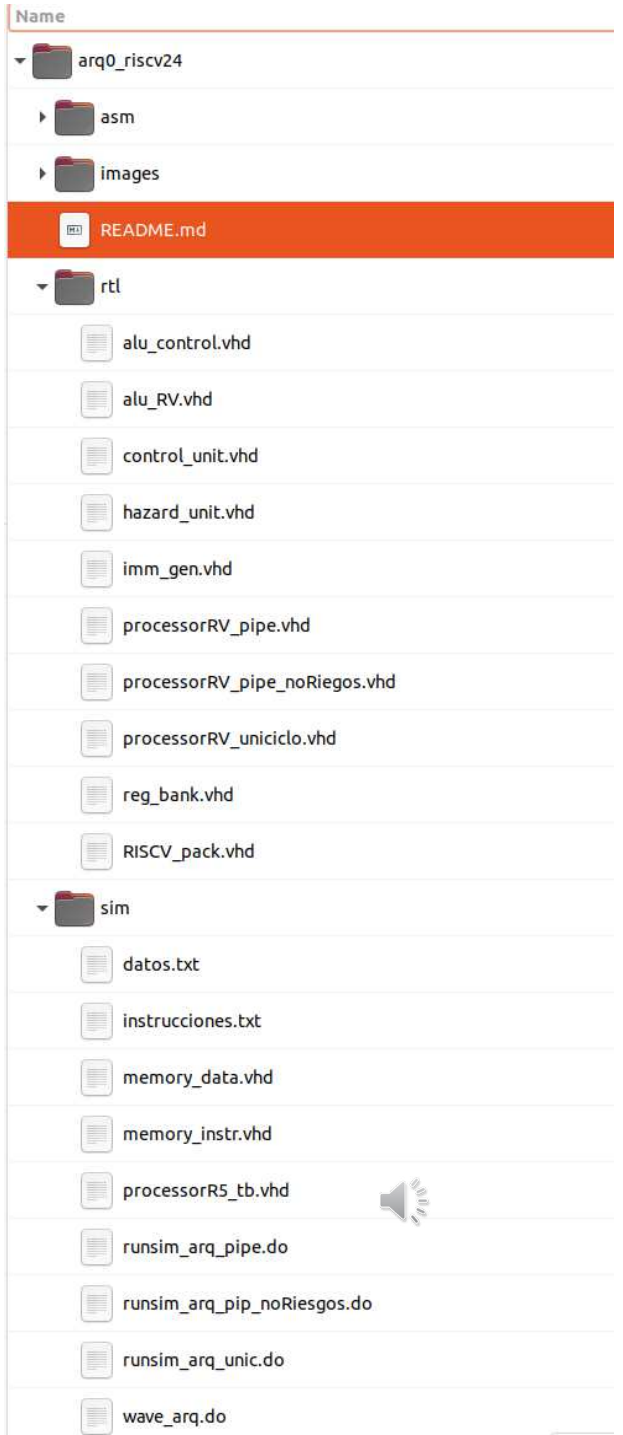
Material Entregado

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

processorRV_uniciclo.vhd	**Procesador uniciclo (top Level)**
processorRV_pipeNoRiesgos.vhd	Procesador Pipeline sin soporte de riesgos
processorRV_Pipe.vhd	Procesador Pipeline para completar por el estudiante
alu_control.vhd	Control de la ALU, completo
alu_RV.vhd	ALU para RISC-V, completa
control_unit.vhd	Unidad de control, completo
hazard_Unit.vhd	Unidad de riesgos a completar por el estudiante
imm_gen.vhd	Generador de inmediato, completo
reg_bank.vhd	Banco de registros, completo
RISCV_pack.vhd	Package con definiciones comunes

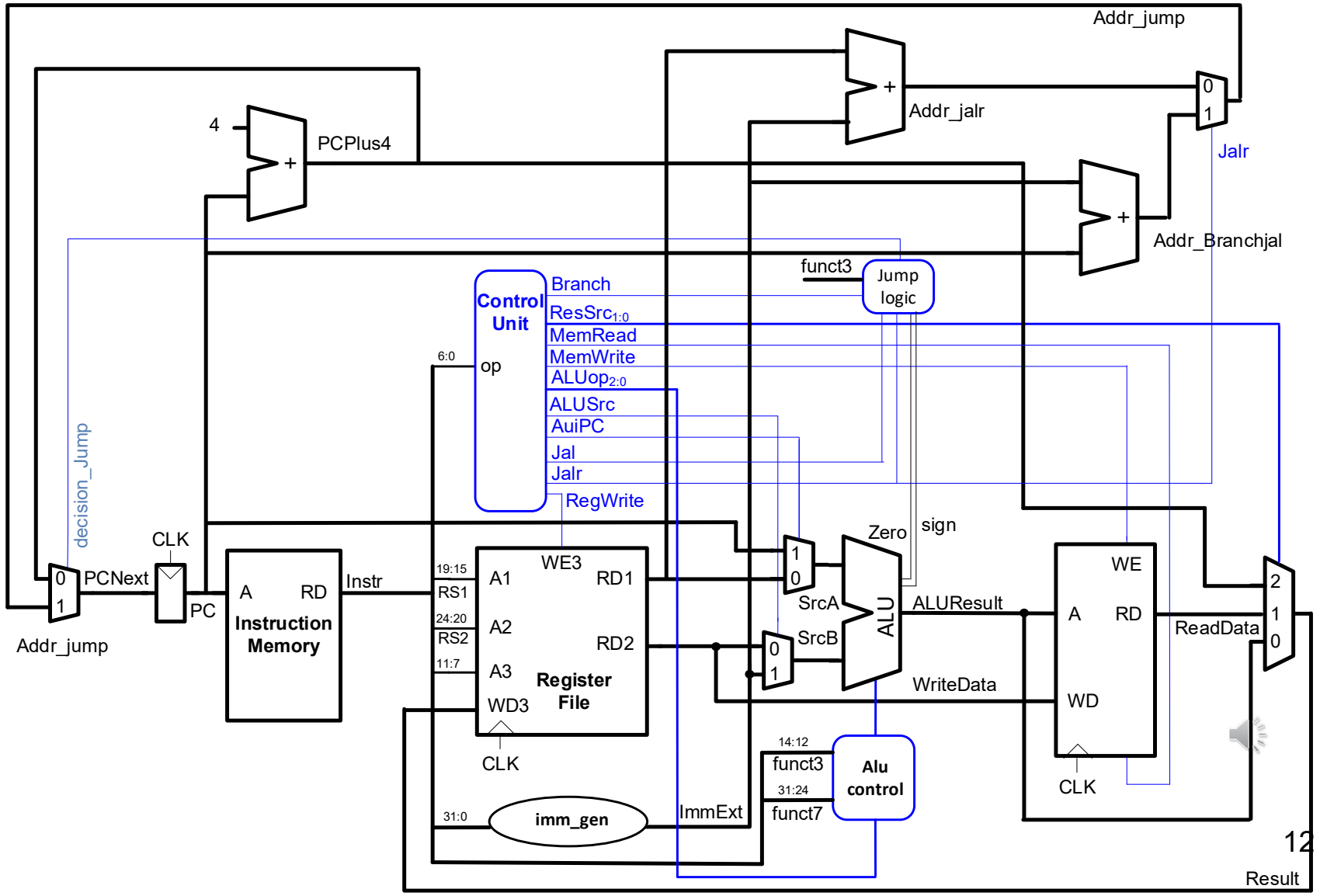
- directorio ***sim/*** : contiene lo necesario para simular

processorR5_tb.vhd	Testbench. Prueba el diseño
datos.txt	Fichero de datos para la memoria de datos
Instrucciones.txt	Fichero de datos para la memoria de instrucciones
memory_datos.vhd	Modelo simple de memoria síncrona de datos
memory_instr.vhd	Modelo simple de memoria síncrona de instrucciones
runsim_arq_pipe.do	Script de simulación para QuestaSim Procesador pipeline
runsim_arq_pipNoriesgo.do	Script de simulación Procesador pipeline sin solución risgos
runsim_arq_unic.do	Script de simulación para QuestaSim Procesador uniciclo
wave.do	Script de configuración de ondas paras ModelSim/Questa



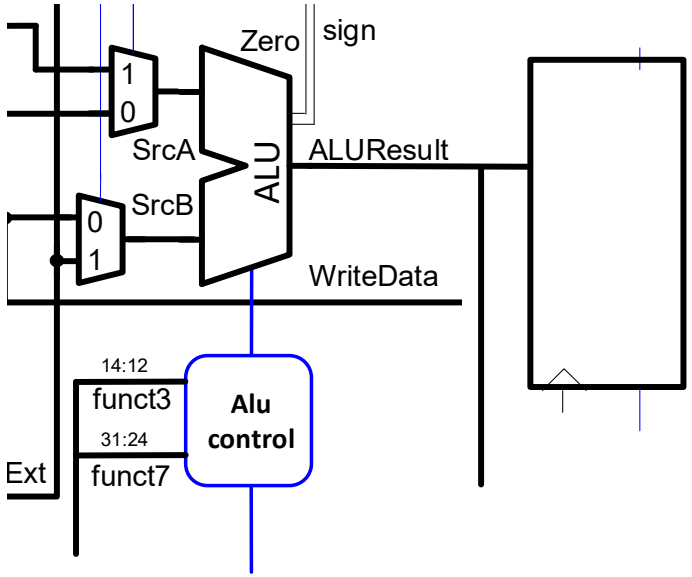
RISC-V descripción de los componentes

- A continuación, se describen los componentes del sistema.
- La versión uniciclo y pipeline utilizan los mismos componentes:
  - ALU (Unidad Aritmético Lógica)
  - Control Unit (unidad de control)
  - Register File (Banco de registros)
  - Imm_gen
  - Alu_control
  - Jump logic



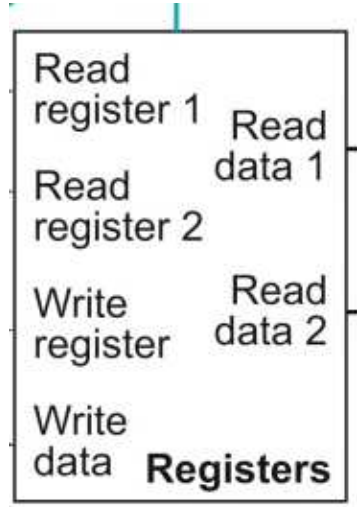
ALU (Arithmetic Logic Unit)

- Es capaz de ejecutar as operaciones (suma, resta, and, etc) del conjunto de instrucciones de esta versión reducida de RISC-V
- Las señales de control las genera el bloque combinacional **“ALU CONTROL”**




```
entity alu_RV is
    port (
        OpA     : in  std_logic_vector (31 downto 0); -- Operando A
        OpB     : in  std_logic_vector (31 downto 0); -- Operando B
        Control : in  std_logic_vector ( 3 downto 0); -- Codigo de control=op. a ejecutar
        Result  : out std_logic_vector (31 downto 0); -- Resultado
        SignFlag: out std_logic;                      -- Sign Flag
        carryOut: out std_logic;                      -- Carry bit
        ZFlag   : out std_logic                       -- Flag Z
    );
end alu_RV;
```

Banco de registros

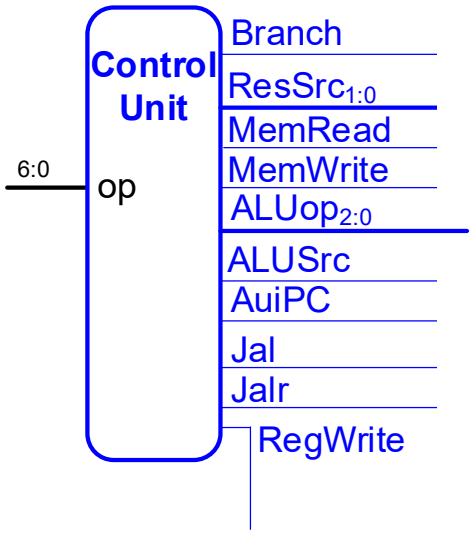


- **32 registros**
- **Lectura asíncrona**
- **Escritura síncrona**
  - Flanco de bajada (soluciona adelantamientos desde WB)
  - El resto de los registros del procesador siempre en flanco de subida
- **Registro 0**
  - Siempre vale 0
  - Escrituras sin efecto

```
entity reg_bank is
   port (
      Clk   : in std_logic; -- Reloj activo en flanco de subida
      Reset : in std_logic; -- Reset asíncrono a nivel alto
      A1    : in std_logic_vector(4 downto 0);   -- Dirección para el puerto Rd1
      Rd1   : out std_logic_vector(31 downto 0); -- Dato del puerto Rd1
      A2    : in std_logic_vector(4 downto 0);   -- Dirección para el puerto Rd2
      Rd2   : out std_logic_vector(31 downto 0); -- Dato del puerto Rd2
      A3    : in std_logic_vector(4 downto 0);   -- Dirección para el puerto Wd3
      Wd3   : in std_logic_vector(31 downto 0);  -- Dato de entrada Wd3
      We3   : in std_logic -- Habilitación de la escritura de Wd3
   );
end reg_bank;
```

Unidad de control

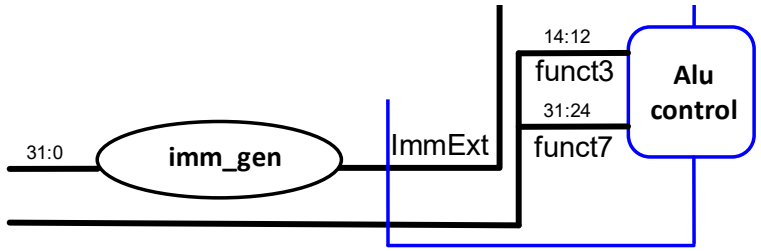
- Genera las señales de control desde el código de operación
  - Entra el código de operación (OpCode) de la instrucción
  - Salen las señales de control: Branch, ResultScr, MemRead, MemWrite, AluOP, AluScr, AuiPC, ins_jalr, RegWrite




```
entity control_unit is
    port (
        -- Entrada = codigo de operacion en la instruccion:
        OpCode   : in  std_logic_vector (6 downto 0);
        -- Seniales para el PC
        Branch     : out  std_logic;                     -- 1 = Ejecutandose instruccion branch
        Ins_Jal    : out  std_logic;                     -- 1 = jal , 0 = otra instruccion,
        Ins_Jalr   : out  std_logic;                     -- 1 = jalr, 0 = otra instruccion,
        -- Seniales relativas a memoria y seleccion dato escritura registros
        ResultSrc: out  std_logic_vector(1 downto 0);  -- 00 salida Alu; 01 = salida de la mem.; 10 PC_plus4
        MemWrite : out  std_logic;                     -- Escribir la memoria
        MemRead  : out  std_logic;                     -- Leer la memoria
        -- Seniales para la ALU
        ALUSrc     : out  std_logic;                     -- 0 = oper.B es registro, 1 = es valor inm.
        AuiPc      : out  std_logic;                     -- 1 = AuiPC (use PC in add), 0 = reg1.
        ALUOp      : out  std_logic_vector (2 downto 0); -- Tipo operacion para control de la ALU
        -- Seniales para el GPR
        RegWrite : out  std_logic                      -- 1=Escribir registro
    );
end control_unit;
```

ALU control, Immediate Gen y Jump Logic

- alu_control: Genera la operación efectiva a realizar por la ALU
- imm_gen: genera el operando inmediato en caso de ser necesario
- Jump Logic: no es una entidad, es código en el procesador




```
entity alu_control is
    port (
        -- Entradas:
        ALUOp  : in std_logic_vector (2 downto 0); -- Codigo de control desde la unidad de control
        Funct7 : in std_logic_vector (6 downto 0); -- Campo "funct7" de la instruccion (I(31:25))
        Funct3 : in std_logic_vector (2 downto 0); -- Campo "funct3" de la instruccion (I(14:12))
        -- Salida de control para la ALU:
        ALUControl : out std_logic_vector (3 downto 0) -- Define operacion a ejecutar por la ALU
    );
end alu_control;
```

```
-- Jump Logic: decide si saltar o sino usa PC+4
decision_Jump  <= Ctrl_Jal_ME or Ctrl_Jalr_ME or (Ctrl_Branch_ME and branch_true);
branch_true    <= '1' when ( ((Funct3_ME = BR_F3_BEQ) and (Alu_ZERO_ME = '1')) or
                             ((Funct3_ME = BR_F3_BNE) and (Alu_ZERO_ME = '0')) or
                             ((Funct3_ME = BR_F3_BLT) and (Alu_SIGN_ME = '1')) or
                             ((Funct3_ME = BR_F3_BGE) and (Alu_SIGN_ME = '0')) ) else
                  '0';
```

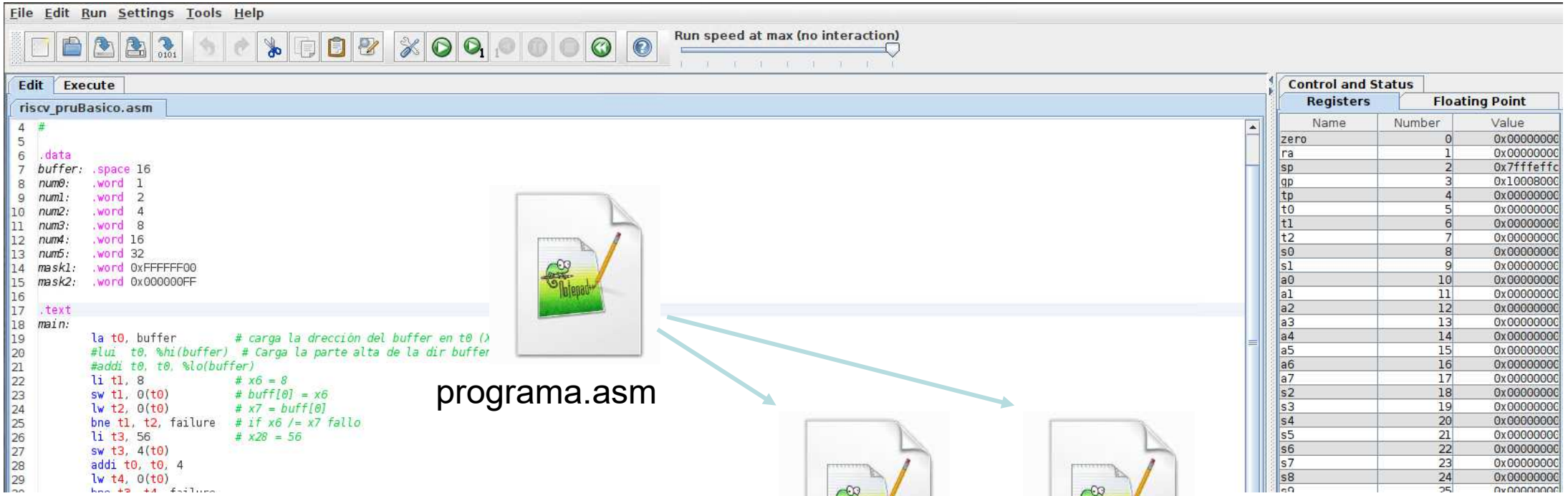
```
entity Imm_Gen is
    port (
        instr     : in std_logic_vector(31 downto 0);
        imm       : out std_logic_vector(31 downto 0)
    );
end entity Imm_Gen;
```

Simular el procesador

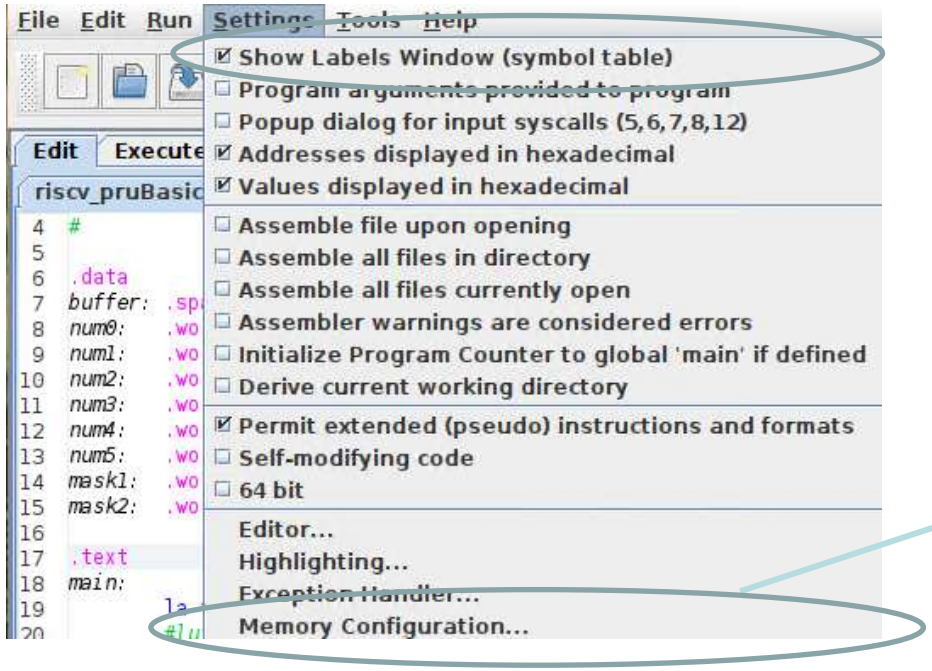
- Se utilizará un simulador HDL (Hardware Descripction Language)
  - En los laboratorios QuestaSim en Linux
  - Fuera de la EPS, QuestaSim (Intel FPGA) o Máquina Virtual
- Para simular el sistema es necesario
  - Código del procesador (y sus componentes)
  - Testbench (procesorRV_tb.vhd) y modelos de las memorias de instrucciones y datos (memory_instr.vhd y memory_data.vhd)
  - Contenido de las memorias (ficheros de texto plano "instrucciones.txt" y "datos.txt"). Ver como generar a continuación.

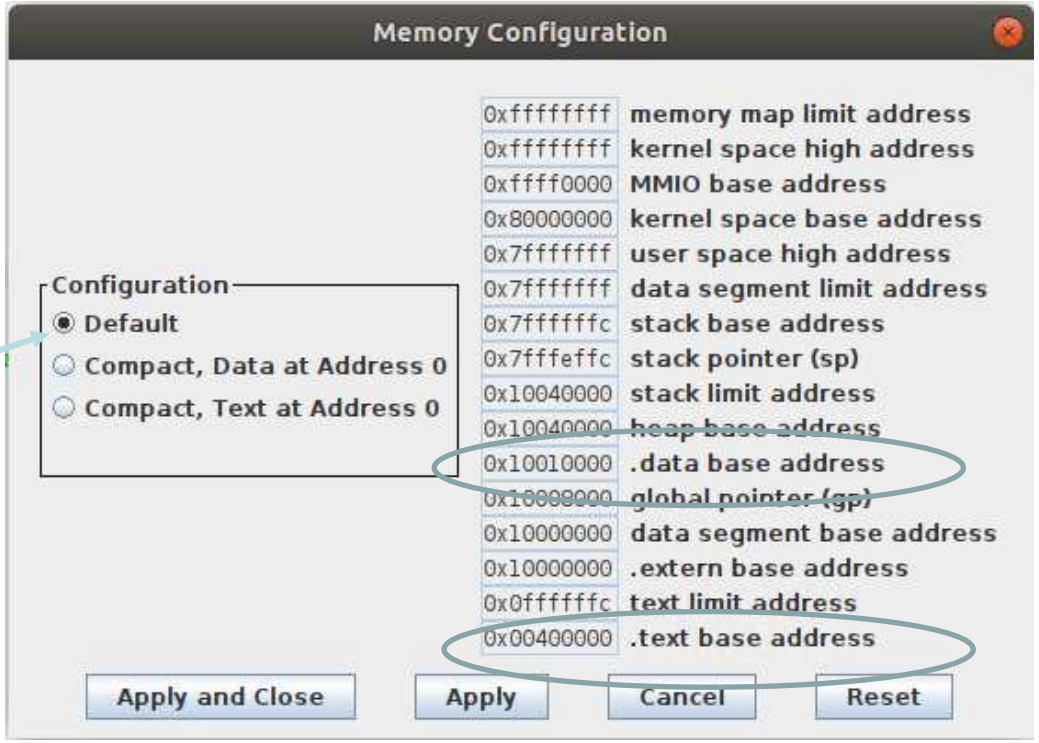
Simulación: generar instrucciones y datos

En Linux (o Windows) utilizaremos el simulador ***RARS*** *(RISC-V Assembler and Runtime Simulator)* para generar el contenido de las memorias



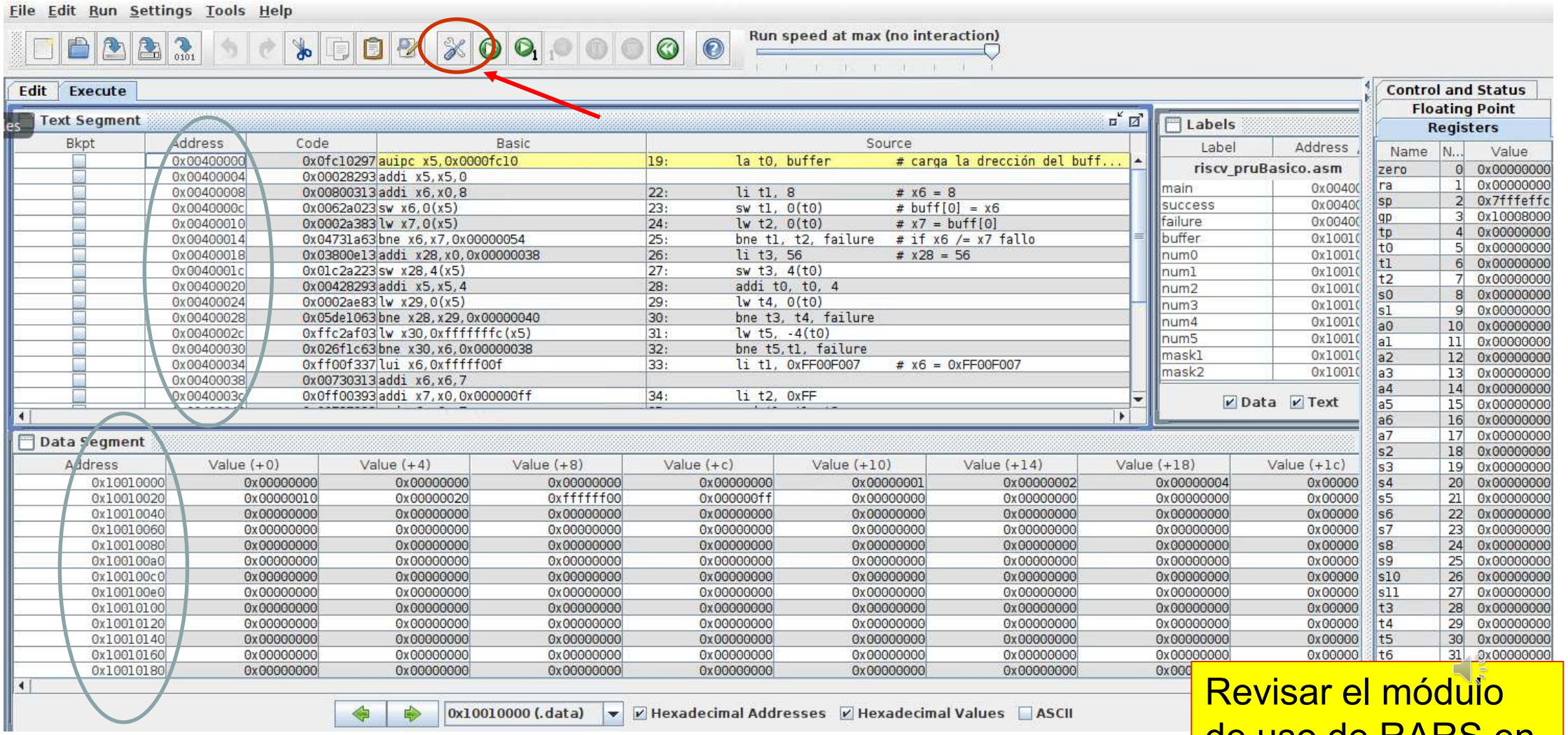
RARS como ensamblador y compilador





- Seleccionar ver las etiquetas (show labels)
- *Dejar la configuración de memoria por defecto:*
  Esto es datos (.data) en dirección 0x1001_0000 y código (.text) en dirección 0x0040_0000

RARS como ensamblador y simulador



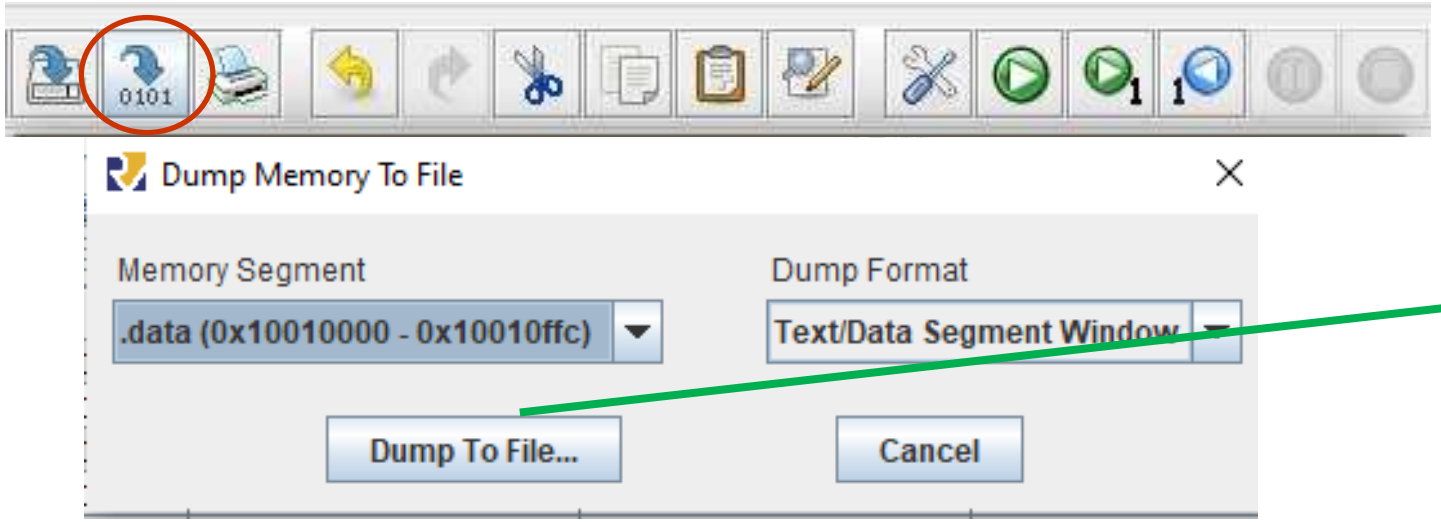
Ensamblar del código. Revisar las direcciones de instrucciones y datos

Revisar el módulo de uso de RARS en caso de dudas

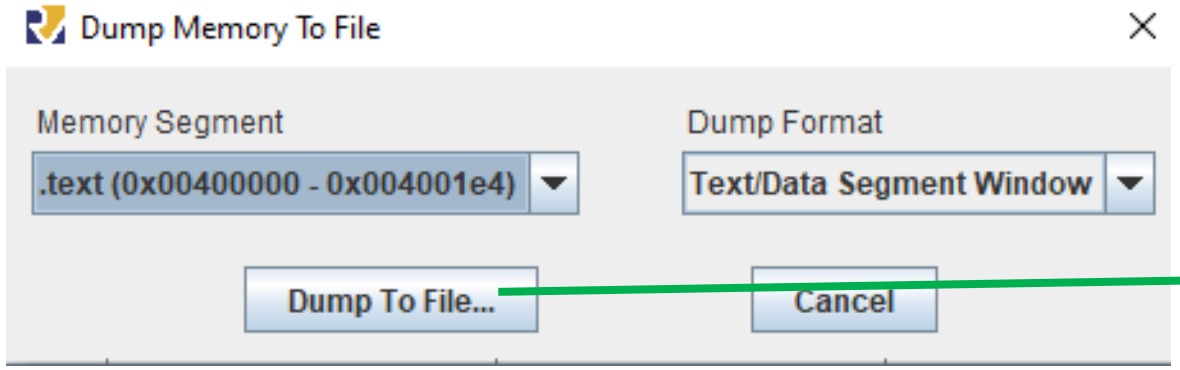
RARS memoria de instrucciones y datos

Exportar el código (dump machine code)

Hacerlo una vez para el código del programa (instrucciones)
Y otra para el contenido de la memoria del programa (datos)



Datos.txt



Instrucciones.txt

RARS memoria de instrucciones y datos

sim > instrucciones.txt

```
1   Address     Code        Basic                    Line Source
2
3   0x00400000  0x0fc10297  auipc x5,0x0000fc10      19    la t0, buffer
4   0x00400004  0x00028293  addi x5,x5,0
5   0x00400008  0x00800313  addi x6,x0,8            22    li t1, 8              # x6 = 8
6   0x0040000c  0x0062a023  sw x6,0(x5)              23    sw t1, 0(t0)          # buff[0] = x6
7   0x00400010  0x0002a383  lw x7,0(x5)              24    lw t2, 0(t0)          # x7 = buff[0]
8   0x00400014  0x04731a63  bne x6,x7,0x00000054     25    bne t1, t2, failure   # if x6 /= x7 fallo
9   0x00400018  0x03800e13  addi x28,x0,0x00000038   26    li t3, 56             # x28 = 56
10  0x0040001c  0x01c2a223  sw x28,4(x5)             27    sw t3, 4(t0)
11  0x00400020  0x00428293  addi x5,x5,4            28    addi t0, t0, 4
12  0x00400024  0x0002ae83  lw x29,0(x5)             29    lw t4, 0(t0)
13  0x00400028  0x05de1063  bne x28,x29,0x00000040   30    bne t3, t4, failure
14  0x0040002c  0xffc2af03  lw x30,0xfffffffc(x5)    31    lw t5, -4(t0)
15  0x00400030  0x026f1c63  bne x30,x6,0x00000038    32    bne t5,t1, failure
16  0x00400034  0xff00f337  lui x6,0xffff00f         33    li t1, 0xFF00F007     # x6 = 0xFF00F007
17  0x00400038  0x00730313  addi x6,x6,7
18  0x0040003c  0x0ff00393  addi x7,x0,0x000000ff    34    li t2,
19  0x00400040  0x00737333  and x6,x6,x7             35    and t1
20  0x00400044  0x00700e13  addi x28,x0,7           36    li t3,
21  0x00400048  0x03c31063  bne x6,x28,0x00000020    37    bne t1
22  0x0040004c  0x000013b7  lui x7,1                 38    li t2,
23  0x00400050  0xfff38393  addi x7,x7,0xffffffff
24  0x00400054  0xfff3c393  xori x7,x7,0xffffffff    39    not t2
25  0x00400058  0xfffffe37  lui x28,0x000fffff       40    lui t3
26  0x0040005c  0x01c39663  bne x7,x28,0x0000000c    41    bne t2
27  0x00400060  0x00000263  beq x0,x0,0x00000004     42    beq ze
28  0x00400064  0x00000063  beq x0,x0,0x00000000     44    beq ze
29  0x00400068  0x00000063  beq x0,x0,0x00000000     46    beq ze
30
```

Ejemplos del código exportado

Datos.txt

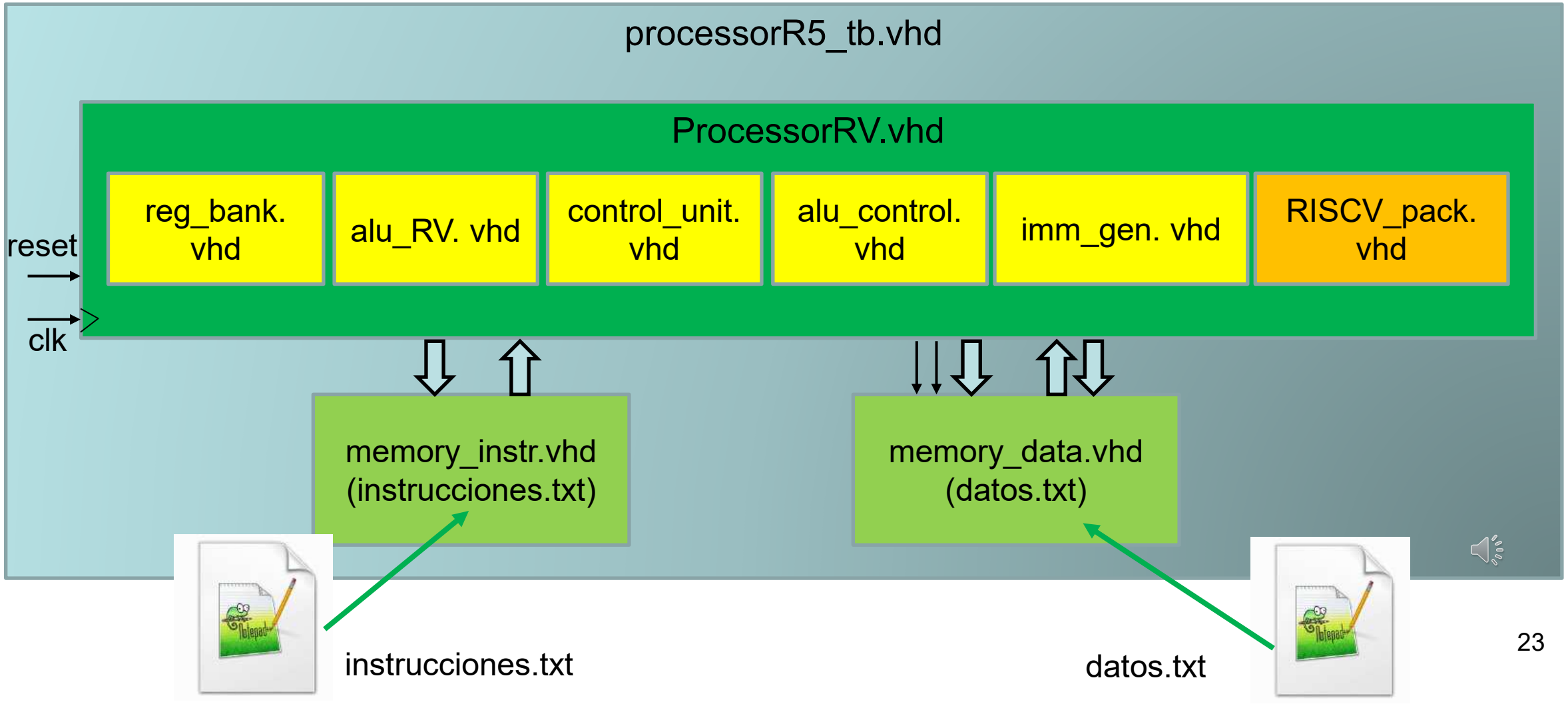
sim > datos.txt

```
1   0x10010000  0x00000008 0x00000000 0x00000000 0x00000000 0x00000000 0x6f626946 0x6363616e 0x30203a69
2   0x10010020  0x20003120 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
3   0x10010040  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
4   0x10010060  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
5   0x10010080  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
6   0x100100a0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
7   0x100100c0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
8   0x100100e0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
9   0x10010100  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
10  0x10010120  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
11  0x10010140  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
12  0x10010160  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
13  0x10010180  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
14  0x100101a0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
15  0x100101c0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
16  0x100101e0  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
17  0x10010200  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
18  0x10010220  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
19  0x10010240  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
20  0x10010260  0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
```

Instrucciones.txt

Simulación

Asegurarse que están todos los componentes en los directorios específicos (/rtl y /sim)



Simulación RTL: ejecutar script

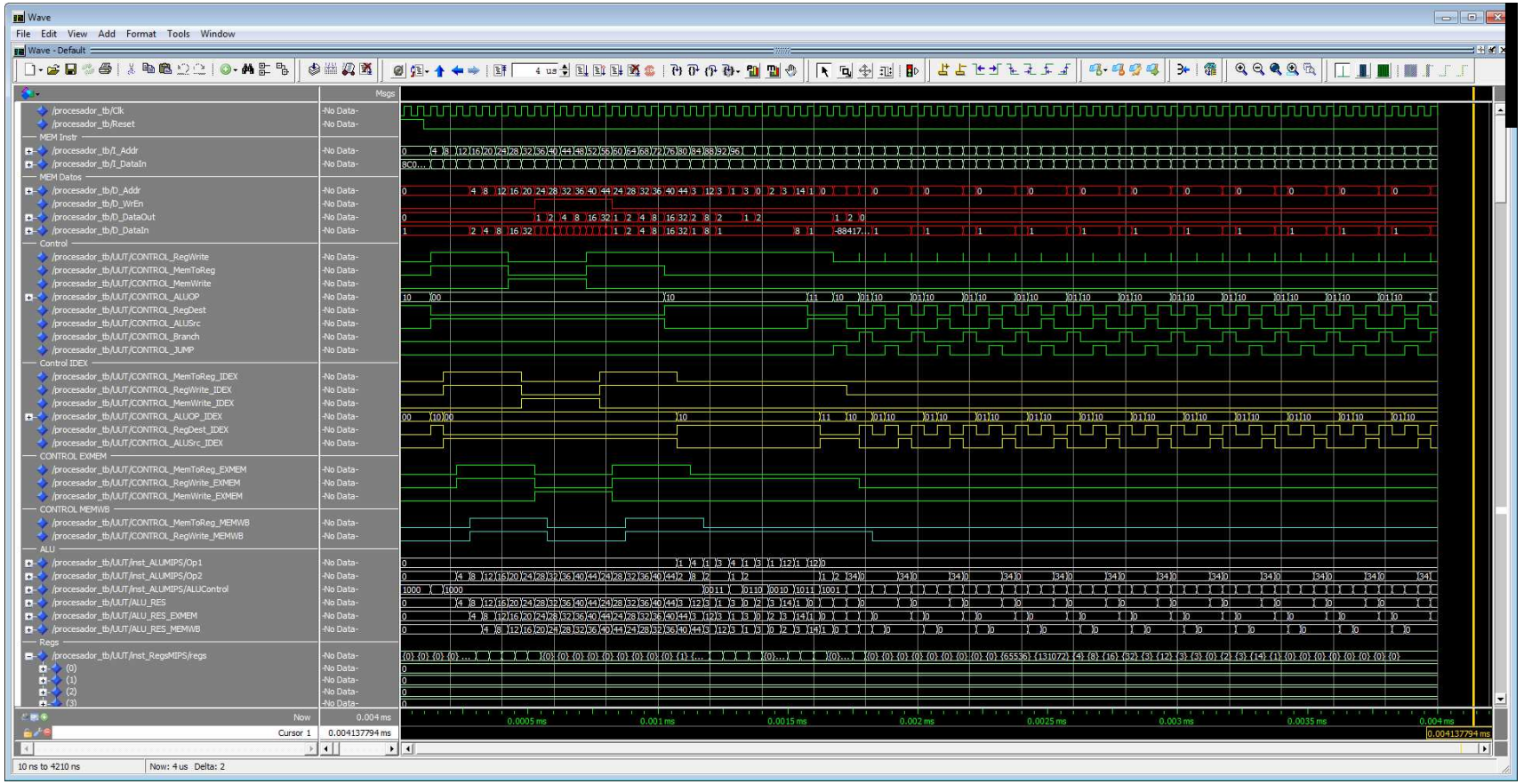
- Ejecutar QuestaSim desde línea de comandos

```
arq@osboxes:~$ vsim
```

- Desde la consola de QuestaSim (modelSim)
- Ir a la carpeta donde está el script (extensión .do)
  - cd arqo2024/sim/ (o usar file-> change directory …)
  - Asegurarse estar donde queremos (pwd)
- Ejecutar el script (comando “do”)
  - **do** runsim_arq_xxx.do

Simulación

Desde de directorio **\sim** lanzar el script de simulación “do runsim_arq_xxx.do” y analizar el contenido de la forma de ondas.
Recuerda de guardar la forma de ondas para futuras simulaciones



¿Qué hace el script?

```
-----------------------------------------------------------------
# Script QuestaSim para la simulacion del procesador Risc V ArqO 2022
-----------------------------------------------------------------
# Crear library, borrando cualquier compilacion previa:
if [file exists work] {vdel -lib work -all }
vlib work

# Compilar RTL:
vcom -work work -2008 ../rtl/RISCV_pack.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/reg_bank.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/alu_RV.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/alu_control.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/control_unit.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/Imm_Generator.vhd
vcom -work work -2008 -explicit -check_synthesis ../rtl/processorRV.vhd

# Compilar testbench:
vcom -work work -2008 -explicit memory_data.vhd
vcom -work work -2008 -explicit memory_instr.vhd
vcom -work work -2008 -explicit processorR5_tb.vhd
---

# Lanzar la simulacion, hast
run -all
-----------------------------------------------------------------
```

- Borra compilación Previa
- Crea biblioteca work
- Compila los fuentes del procesador (vcom)
- Compila los fuentes del simulación (vcom)
- Elabora el diseño (lanza simulación) usando processor_tb como top
  - -g para los generics del toplevel
- Abre las formas de onda del archivo wave_Arq.do
- Simula hasta el final (run -all)

¿Qué hace el script?

```
# Elaboracion:
vsim -gINIT_FILENAME_INST="instrucciones.txt"
     -gINIT_FILENAME_DATA="datos.txt"
      -gN_CYCLES=150 processorRV_tb

# Opcion para guardar todas las ondas:
log -r /*

# Mostrar las ondas:
do wave_arq.do

# Opcion del simulador para evitar warnings tipicos en tiempo 0 :
set StdArithNoWarnings 1
run 0 ns
set StdArithNoWarnings 0

# Lanzar la simulacion, hasta que pare sola:
run -all
```

- Borra compilación Previa
- Crea biblioteca work
- Compila los ficheros fuentes del procesador (vcom)
- Compila los fuentes del simulación (vcom)
- Elabora el diseño (lanza simulación) usando processor_tb como top
  - -g para los generics del toplevel
- Abre las formas de onda del archivo wave_arq.do
- Simula hasta el final (run -all)

Consejos antes de empezar

1. Leer el guion completo de la práctica

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

3. Preguntar al profesor lo que no quede claro