# *Tutoría no presencial 15-Abril-2020*

Escuela Politécnica Superior - UAM

# Microsoft Teams

- Empezamos a las 11:05
- **Desactivad el audio** mientras no preguntéis
- Estaría bien que **algun@s** tuvierais el vídeo activado para ver si seguís las explicaciones. Los profesores necesitamos ver vuestras reacciones. Con que haya 5-10 es suficiente. ¡Gracias!
- **¡PREGUNTAD!** La única pregunta mala es la que no se hace.

# Práctica 4: Diseño del Microprocesador MIPS

- Realización y evaluación: **INDIVIDUAL** (Peso: 50%)
- Tiempo de realización: **4 semanas**
- Tutorías dudas→ Cada uno con su profesor
- Examen : 13 de Mayo (**convocatoria única**). Arquitectura alternativa
- Soluciones: 29 de Mayo
- Contenido:
  - **Ejercicio 1:** Adaptación al juego de instrucciones presente.
  - **Ejercicio 2:** Diseño de la unidad de control.
  - **Ejercicio 3:** Diseño completo del microprocesador.

# Ejercicio 1

- Ejercicio 2 de la práctica 3 → *Vectores.asm* (Si tiene errores→ ¡CORREGIDLOS!)
- **Adaptarlo** usando el nuevo juego de instrucciones.
- MARS→ Misma funcionalidad que antes.
- Nuevos datos:

| Posición de memoria | Etiqueta | Valores |
|---|---|---|
| x2000 | N | 6 |
| x2004 | A | 2,4,6,8,10,12 |
| x201C | B | -1,-5,4,10,1,-5 |
| x2034 | C | |

| OPCode | Nombre | Descripción | Operación |
|---|---|---|---|
| 000000 | R-Type | Instrucciones con formato R-Type | Varias. Se verán en la siguiente tabla |
| 000010 | j | Salto incondicional | PC = JTA |
| 000100 | beq | Bifurca si igual (Z = 1) | Si ([rs] == [rt]); PC =BTA |
| 001000 | addi | Suma con dato inmediato | [rt] = [rs] + SigImm |
| 001100 | andi | AND con dato inmediato | [rt] = [rs] & ZeroImm |
| 001101 | ori | OR con dato inmediato | [rt] = [rs] \| ZeroImm |
| 100011 | lw | Lee una palabra de memoria | MEM ([rs]+SigImm) => [rt] |
| 101011 | sw | Escribe una palabra en memoria | [rt] => MEM ([rs]+SigImm) |
| 001010 | slti | Set on less than (inmediato) | [rs]<[SigImm] ? [rt]=1 : [rt]=0 |

R-TYPE

| Funct | Nombre | Descripción | Operación |
|---|---|---|---|
| 100100 | and | Función AND | [rd] = [rs] & [rt] |
| 100000 | add | Sumar | [rd] = [rs] + [rt] |
| 100010 | sub | Restar | [rd] = [rs] - [rt] |
| 100110 | xor | Función XOR | [rd] = [rs] XOR [rt] |
| 100101 | or | Función OR | [rd] = [rs] \| [rt] |
| 101010 | slt | Set on less than | [rs]<[rt] ? [rd]=1 : [rd]=0 |

# Ejercicio 1

Edit | Execute

**Text Segment**

| Bkpt | Address | Code | Basic | | Source |
|---|---|---|---|---|---|
| | 0x00000000 | 0x8c042078 | lw $4,8312($0) | 18: | lw $4, N # Se carga N, que es el número de elementos del vector |
| | 0x00000004 | 0x00042080 | sll $4,$4,2 | 21: | sll $4, $4, 2 # Por 4, y en vez de utilizar i se utiliza i*4, que es el offset dentro del vector |
| | 0x00000008 | 0x00a52826 | xor $5,$5,$5 | 24: | xor $5, $5, $5 # Inicializa $5 a 0 |
| | 0x0000000c | 0x10850007 | beq $4,$5,7 | 27: | inicioBucle: beq $4,$5, fin |
| | 0x00000010 | 0x8ca12000 | lw $1,8192($5) | 30: | lw $1, A($5) # Carga del dato a[$5] en $1 |
| | 0x00000014 | 0x8ca22028 | lw $2,8232($5) | 33: | lw $2, B($5) # Carga del dato b[$5] en $2 |
| | 0x00000018 | 0x00021080 | sll $2,$2,2 | 36: | sll $2, $2, 2 # b[i]*4 en $3 |
| | 0x0000001c | 0x00221820 | add $3,$1,$2 | 39: | add $3, $1, $2 # Ahora $4 vale a[i]+b[i]*4 |
| | 0x00000020 | 0xaca32050 | sw $3,8272($5) | 42: | sw $3, C($5) # Se guarda el resultado en c[$5] |
| | 0x00000024 | 0x20a50004 | addi $5,$5,4 | 45: | addi $5, $5, 4 |
| | 0x00000028 | 0x08000003 | j 0x0000000c | 48: | j inicioBucle |
| | 0x0000002c | 0x0800000b | j 0x0000002c | 51: | fin: j fin |

**Data Segment**

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) | Value (+14) | Value (+18) | Value (+1c) |
|---|---|---|---|---|---|---|---|---|
| 0x00002000 | 2 | 2 | 4 | 6 | 5 | 6 | 7 | 8 |
| 0x00002020 | 9 | 10 | -1 | -5 | 4 | 10 | 1 | -2 |
| 0x00002040 | 5 | 10 | -10 | 0 | 0 | 0 | 0 | 0 |
| 0x00002060 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| 0x00002080 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000020a0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000020c0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000020e0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x00002100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x00002120 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x00002140 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x00002160 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x00002180 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000021a0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000021c0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x000021e0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

0x00002000 (.data) | Hexadecimal Addresses | Hexadecimal Values | ASCII

# Ejercicio 1

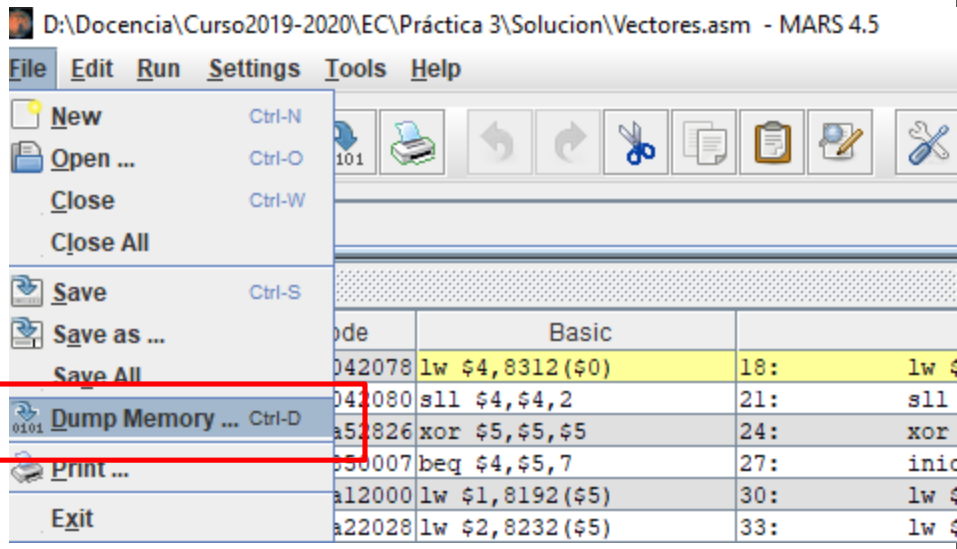
Execute→File→Dump Memory

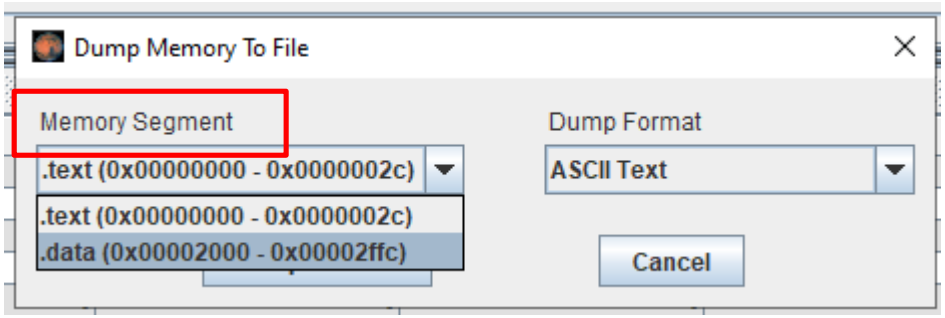
- **MemDataVectores.vhd**→.data

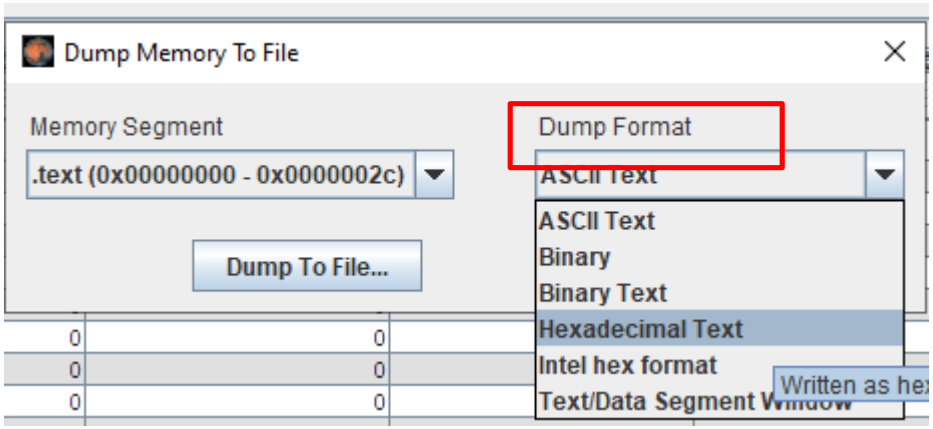
  (Plantilla MemDataMIPS.vhd)

- **MemProgVectores.vhd**→.text

  (Plantilla MemProgMIPS.vhd)



# Ejercicio 1

MemDataMIPS.vhd

```
28    -- 4 GB son 1 gigapalabras, pero el simulador no deja tanta
29    -- Dejamos 64 kB (16 kpalabras), usamos los 16 LSB
30    type Memoria is array (0 to (2**14)-1) of signed(31 downto 0
31    signal memData : Memoria;
32
33  begin
34
35      -- Proceso para la escritura síncrona en la memoria de dat
36  EscrituraMemData: process(all)
37      begin
38          -- Con el reset activo, se inicializa a ceros todas la
39      if NRst = '0' then
40          for i in 0 to (2**14)-1 loop
41              memData(i) <= (others => '0');
42          end loop;
43
44          -- Código para la escritura de los datos iniciales que
45
46          -- Se cargan a partir de una dirección determinada, en
47          -- Para que se entienda como un índice entero, la dire
48          -- Como cada dato ocupa 4 bytes, el índice entero debe
49
50  --      ************************************************
51                  memData(16#2000#/4) <= x"0000000A";
52                  memData(16#2004#/4) <= x"00000009";
53                  memData(16#2008#/4) <= x"00000009";
54  --      ************************************************
55
56
57      -- Código para la escritura síncrona en la memoria de dato
58      elsif rising_edge(Clk) then
59          -- En este caso se escribe por flanco de bajada para q
60          -- a mitad de ciclo y todas las señales estén estables
```

MemProgMIPS.vhd

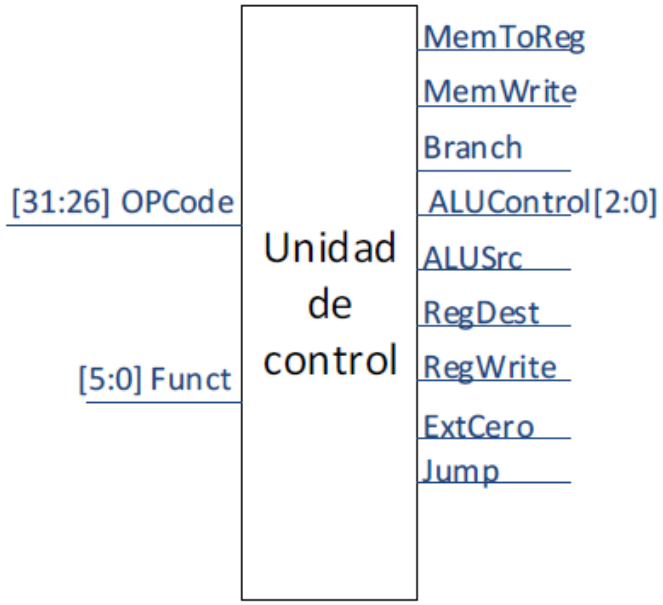
```
29          -- La memoria devuelve un valor para cada dirección.
30          -- Estos valores son los códigos de programa de cada inst
31          -- Código para la escritura de los datos iniciales quedeb
32
33
34          case MemProgAddr is
35  -- ************************************************************
36              when x"00000000" => MemProgData <= x"20010004";
37              when x"00000004" => MemProgData <= x"3402000f";
38              when x"00000008" => MemProgData <= x"30430004";
39              when x"0000000C" => MemProgData <= x"2064ffec";
40              when x"00000010" => MemProgData <= x"28057fff";
41              when x"00000014" => MemProgData <= x"2805ffff";
42              when x"00000018" => MemProgData <= x"0081302a";
43              when x"0000001C" => MemProgData <= x"8c072000";
44              when x"00000020" => MemProgData <= x"8c282000";
45              when x"00000024" => MemProgData <= x"8c092008";
46              when x"00000028" => MemProgData <= x"01075022";
47              when x"0000002C" => MemProgData <= x"10240001";
48              when x"00000030" => MemProgData <= x"11090001";
49              when x"00000034" => MemProgData <= x"00005020";
50              when x"00000038" => MemProgData <= x"214b0002";
51              when x"0000003C" => MemProgData <= x"00016020";
52              when x"00000040" => MemProgData <= x"ac0c200c";
53              when x"00000044" => MemProgData <= x"8c0d200c";
54              when x"00000048" => MemProgData <= x"00437026";
55              when x"0000004C" => MemProgData <= x"01c07024";
56              when x"00000050" => MemProgData <= x"00227825";
57              when x"00000054" => MemProgData <= x"08000015";
58  -- ************************************************************
59              when others => MemProgData <= x"0[illegible]"; -- Resto d
60          end case;
61      end process EscrituraMemProg;
```

**MemProgVectores.vhd** y **MemDataVectores.vhd** no se podrán comprobar. Se usarán en el Ejercicio 3

# Ejercicio 2

1. Completar la tabla III con las instrucciones de la tabla II. X, 1 o 0. Consejo: añadir en la tabla las columnas Opcode y Funct. → COMPROBACIÓN POR VUESTRO PROFESOR
2. Implementarla en VHDL
3. Comprobar con el testbench “UnidadControlTb.vhd”

| INSTRUCCIÓN | MemToReg | MemWrite | Branch | ALUControl | ALUSrc | RegDest | RegWrite | ExtCero | Jump |
|---|---|---|---|---|---|---|---|---|---|
| R-Type | | | | | | | | | |
| Lw | | | | | | | | | |
| Sw | | | | | | | | | |
| Beq | | | | | | | | | |
| Lógicas Inm | | | | | | | | | |
| Aritméticas Inm | | | | | | | | | |
| J | | | | | | | | | |



# Ejercicio 3

Tenemos:

- **ALU** (ALUMIPS.vhd) (instanciar)
- **GPR** (RegsMIPS.vhd) (instanciar)
- **Unidad de Control** (instanciar) (UnidadControl.vhd)
- **Memoria de programa**

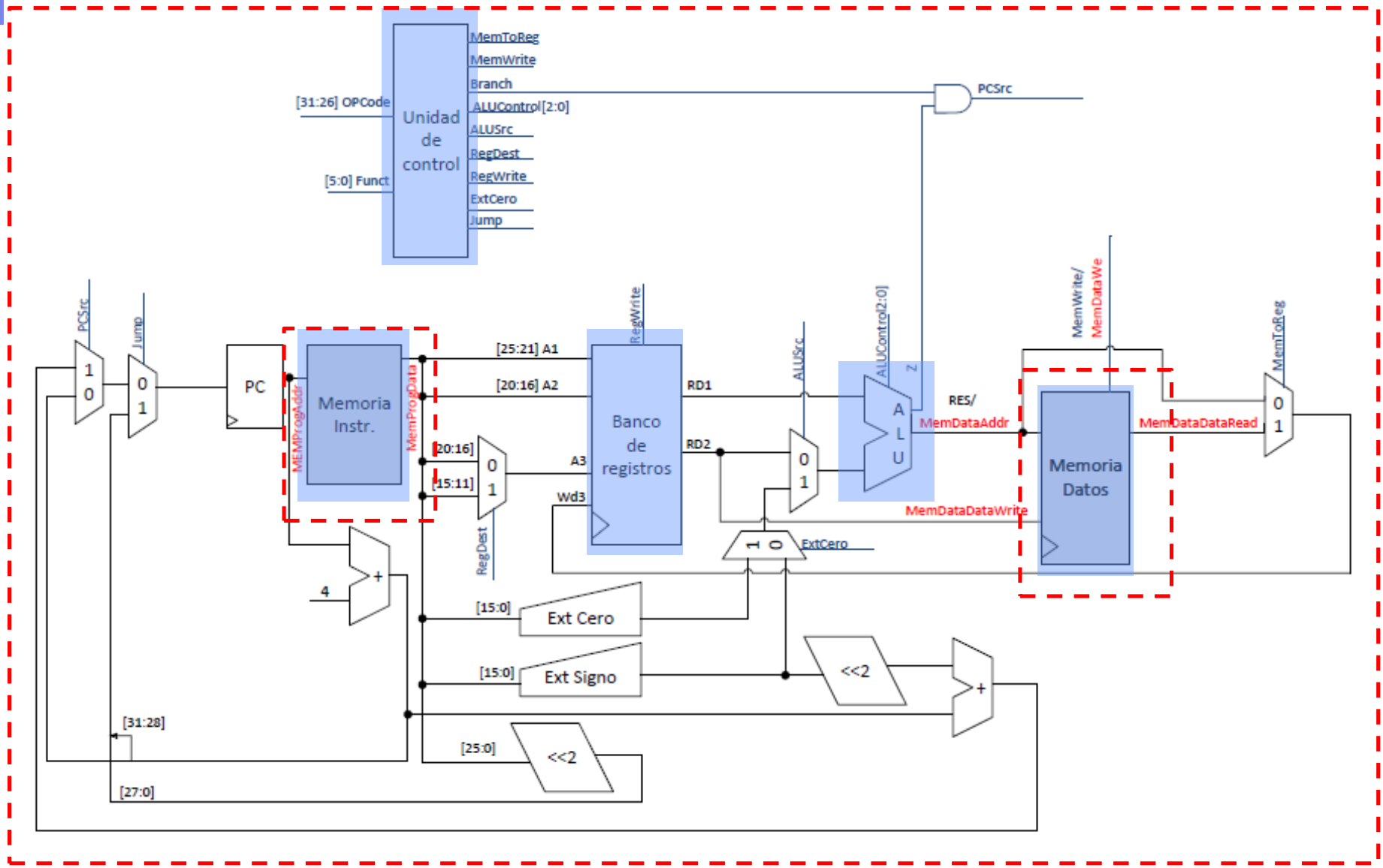
  (MemProgMIPS.vhd)
- **Memoria de datos**

  (MemDataMIPS.vhd)

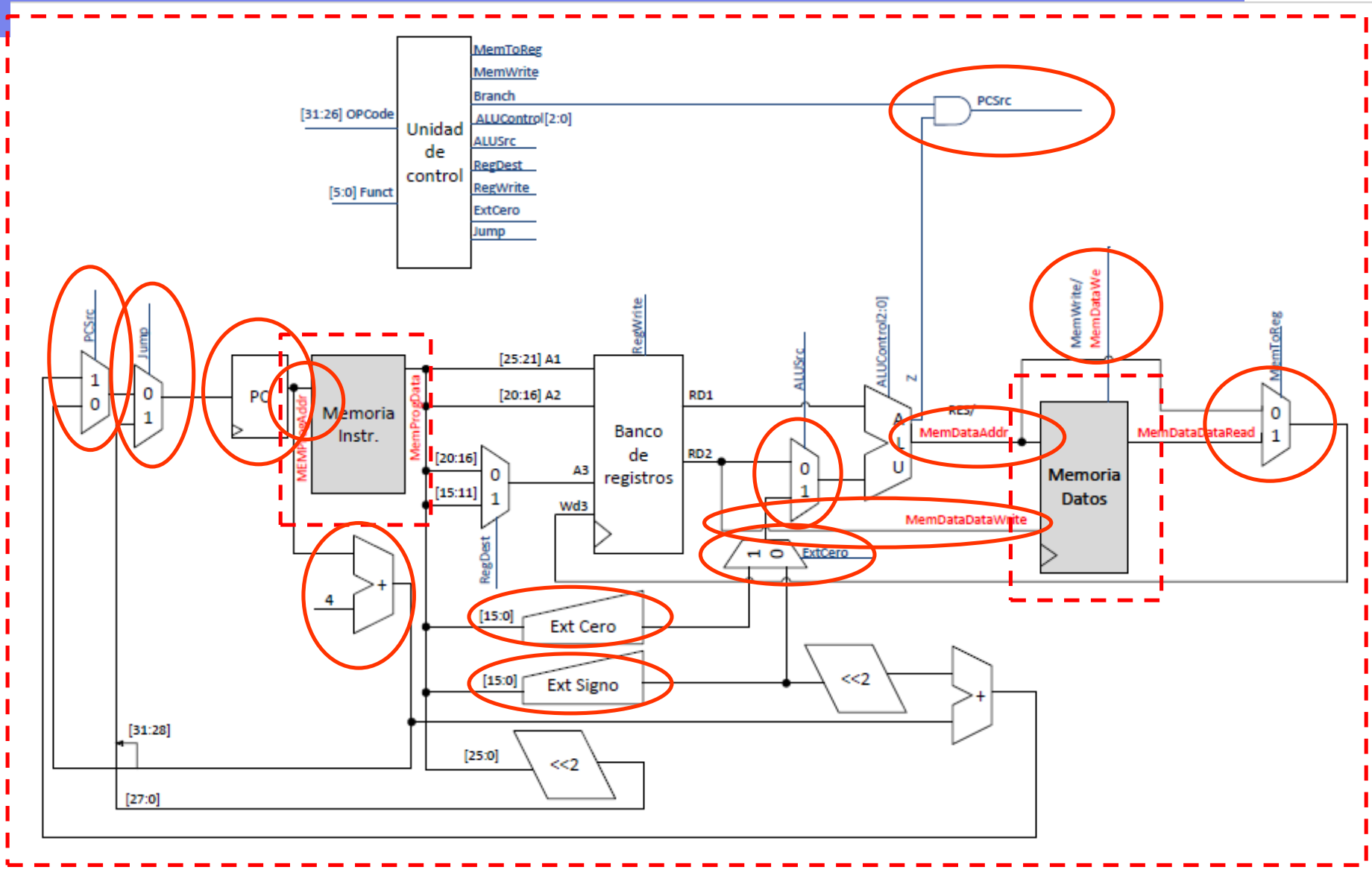
Implementar:

- **Microprocesador**(MicroMIPS.vhd):
  - Entradas:
    - Low Active Reset (NRst)
    - Clock (Clk)
    - MemProgData
    - MemDataDataRead
  - Salidas:
    - MemDataWe
    - MemProgAddr
    - MemDataAddr
    - MemDataDataWrite
- **Resto conexiones y multiplexores**

# Ejercicio 3


Unidad de control
[31:26] OPCode
[5:0] Funct
MemToReg
MemWrite
Branch
ALUControl[2:0]
ALUSrc
RegDest
RegWrite
ExtCero
Jump
PCSrc
PC
Memoria Instr.
MEMProgAddr
MemProgData
[25:21] A1
[20:16] A2
[20:16]
[15:11]
A3
Wd3
Banco de registros
RD1
RD2
RegWrite
ALUSrc
ALUControl2:0]
A L U
Z
RES/ MemDataAddr
MemDataDataWrite
MemWrite/ MemDataWe
Memoria Datos
MemDataDataRead
MemToReg
RegDest
ExtCero
4
[15:0]
Ext Cero
[15:0]
Ext Signo
<<2
[31:28]
[25:0]
<<2
[27:0]
Señales de control
Entradas/Salidas
Datos

# Ejercicio 3

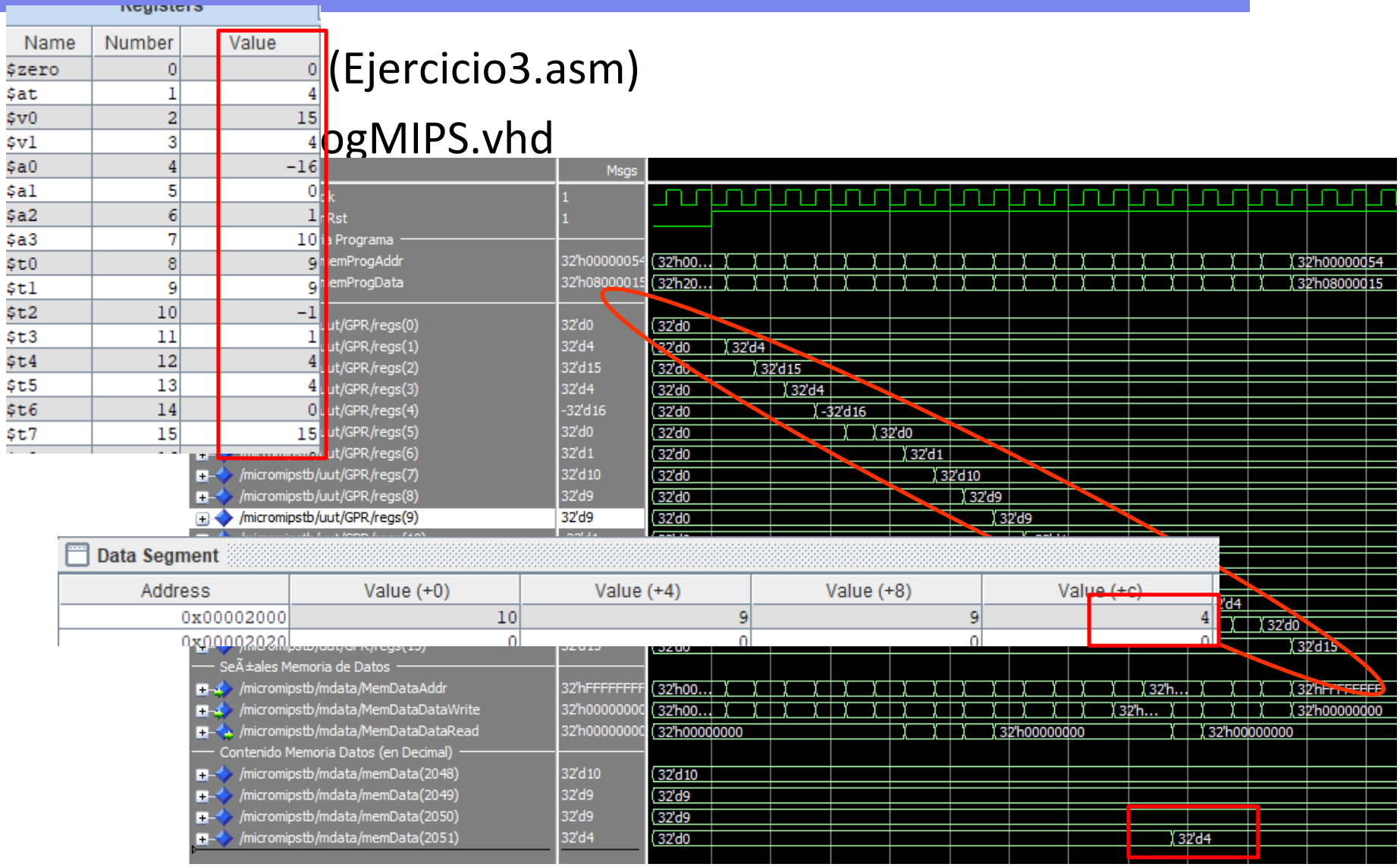



Señales de control  Entradas/Salidas  Datos

# Ejercicio 3: DOBLE VALIDACIÓN

(Ejercicio3.asm)

ogMIPS.vhd



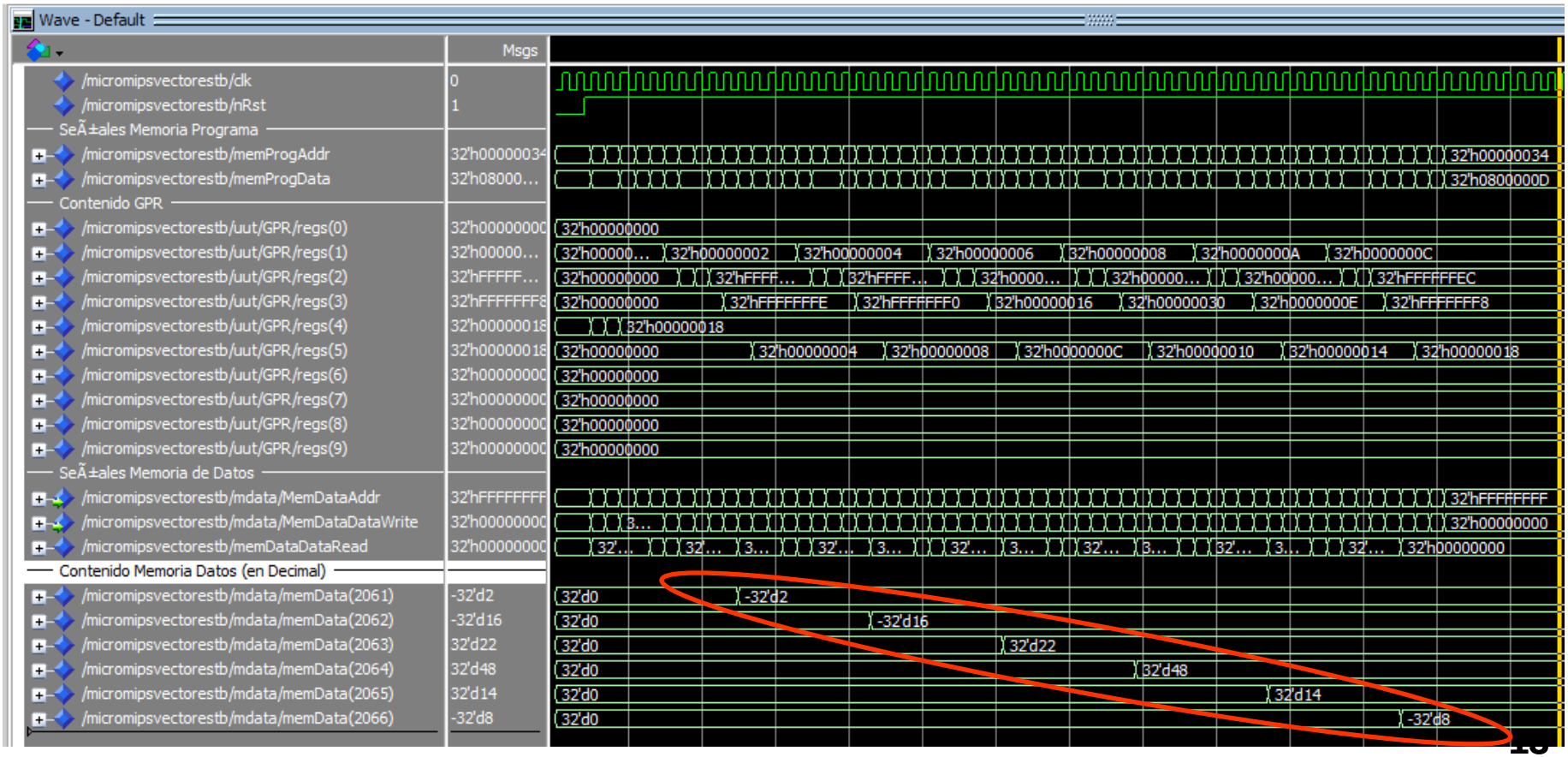
# Ejercicio 3: DOBLE VALIDACIÓN

**B. Vectores:** (Vectores asm-Ejercicio 1)

- MemDataVectores.vhd

Data Segment

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) | Value (+14) | Value (+18) | Value (+1c) |
|---|---|---|---|---|---|---|---|---|
| 0x00002000 | 6 | 2 | 4 | 6 | 8 | 10 | 12 | -1 |
| 0x00002020 | -5 | 4 | 10 | 1 | -5 | -2 | -16 | 2 |
| 0x00002040 | 43 | 14 | -8 | 0 | 0 | 0 | 0 | 0 |
| 0x00002060 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



# Tutoría

## ¿DUDAS?

- Las tutorías serán útiles en la medida en que preguntéis