TDC_Lab_4. Memory design using VHDL

4.1. Modeling RAM memory with VHDL

RAM memory

- Create a new Project, **Project_23**, and add a VHDL file to describe a generic Write-First RAM memory. (Synchronous Write and Read)
- Set the generics ADDR_WIDTH to 2 bits (2**M) and DATA_WIDTH to 8 bits (N).
- Test the circuit onto Nexys4, use switches and LED.
- Firstly write some data in several address, then read the data.

Project's name: Project_23
Entity's name: "RAM_WF_MxN"
Inputs: DATA_RAM_i, ADDR_RAM_i, WE_i, CLK_i
Outputs: DATA_RAM_o

4.2. DATAPATH and RAM

Adding a RAM to DATAPATH

P24_DATAPATH_0_RAM

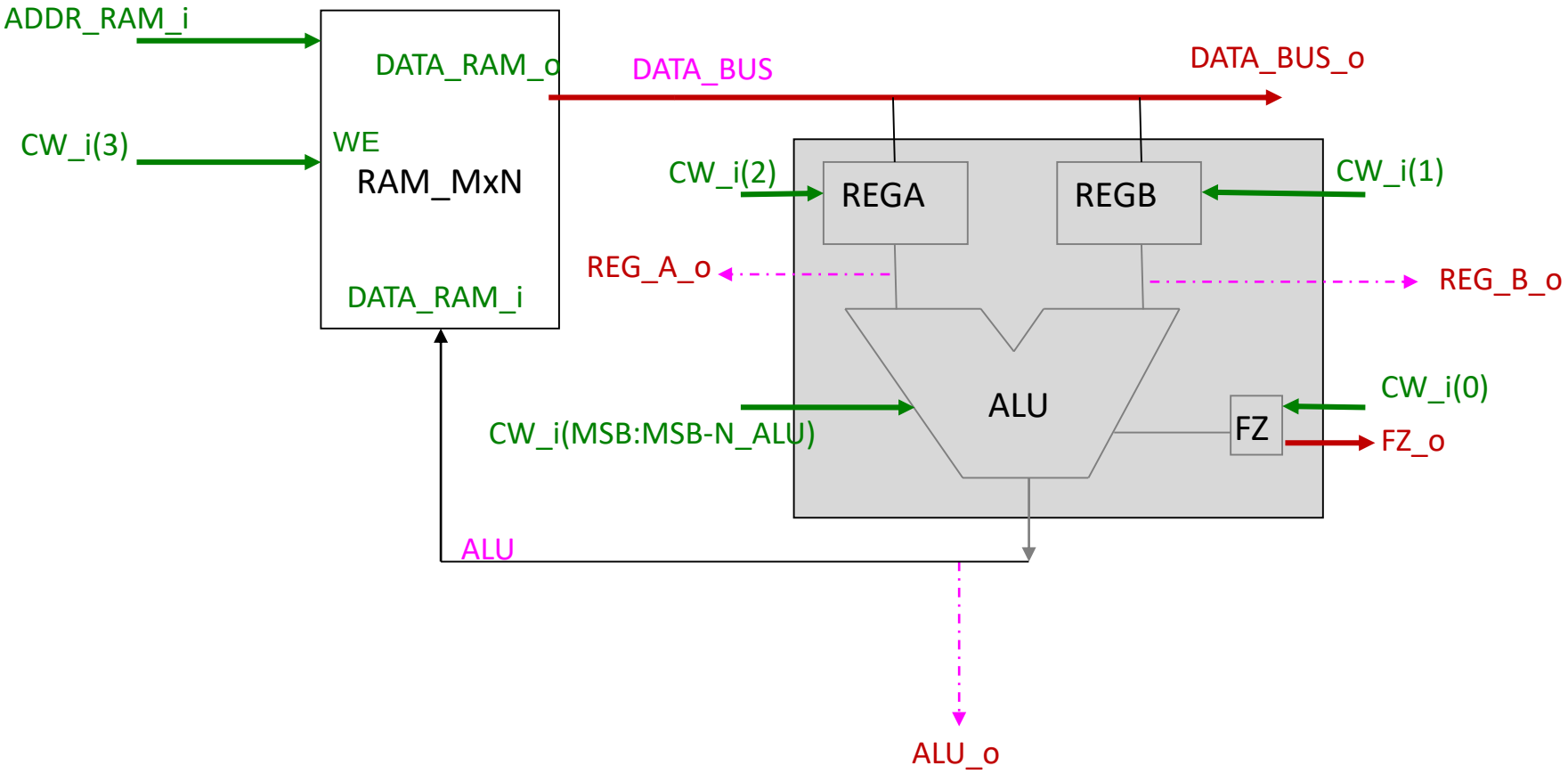
Vivado allows copying projects. Now, copy the P22_Datapath_0_ALU_RA_RB as P24_DATAPATH_0_RAM

Guides and Tips:

- Add to the datapath a generic WF RAM **as a new process**. You will need modify the entity's ports. (Define the RAM as 2**ADDR_RAM_WIDTH x DATA_WIDTH).
- The image in the next slide shows the block-diagram you should get.
- Two slides ahead there is a table containing the values to store in each memory position. Use signal initialization to store the data.

Adding a RAM to DATAPATH: architecture for designing

P24_Datapath_0_RAM



Adding a RAM to DATAPATH: initial values for the operands

Address (4bits) Hex.	Data (4 bits) Binary
0	0101
1	1111
2	1110
3	1110
4	1110
5	0000
6	0001
7	0010
8	0101
9	1111
A	0000
B	0000
C	0000
D	0000
E	0000
F	0000

The next slide shows the format to initialize a signal

Adding a RAM to DATAPATH: Initial values for RAM

First method :

```
signal RAM : ram_type:=("0000", "0001", "0010", "0011", "0000"
, "1111", "0000", "0000", "0000"
, "1111", "1100", "0011", "0010"
, "0001", "0000", "0000");
```

Item 15th

Item 0th

Remember that the RAM is a array of arrays data type.

Adding a RAM to DATAPATH: Initial values for RAM

Second method:

```
signal RAM: ram_type:=
( 0 => "0001",
  1 => "1111",
  2 => "0101",
  others => "0000");
```

Binary format

Index

Third method:

```
  signal RAM: ram_type:=
( 0 => X"1",
  1 => X"F",
  2 => X"5",
  others => "0000");
```

Hexadecimal format

Remaining addresses with the same value

Adding a RAM to DATAPATH: Initial values for RAM

Two options for “others” clausule:

1.1. Set the value for each array in the matrix

```
RAM<=(others => "0000");
```

1.2. Set the value for each bit in the matrix

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

Adding a RAM to DATAPATH: Preparing the testbench

P24_Datapath_0_RAM: Add a testbench which writes the proper values for address and CW in each clock-cycle for each operation.

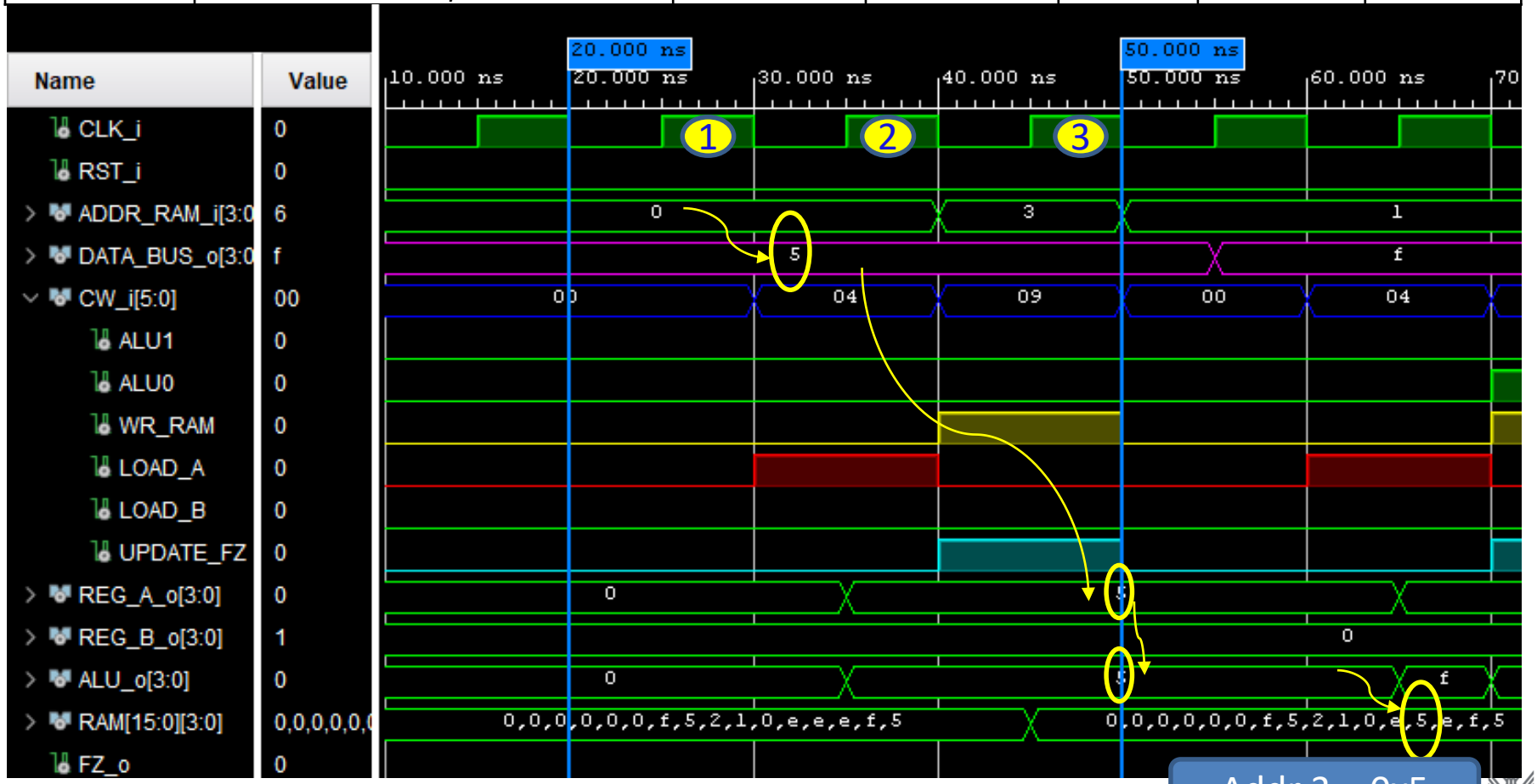
	MOV A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	*MOV operation, set B address RAM and update FZ*	00	1	0	0	1

```
-- 1) MOV A,B ---------------------
-- CLK1: Read OPE_A in RAM(0)
     ADDR_RAM_i   <= "0000";
     CW_i         <= "000000";
     wait for 10 ns;
-- CLK2: Load OPE_A
     CW_i         <= "000100";
     wait for 10 ns;
-- CLK3:Select ALU operation and FZ
     -- Write result in RAM (3)
     ADDR_RAM_i   <= "0011";
     CW_i         <= "001001";
     wait for 10 ns;
```

The readout is available one clk-cycle after.

Adding a RAM to DATAPATH: Preparing the testbench

	MOV A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	*MOV operation, set B address RAM and update FZ*	00	1	0	0	1



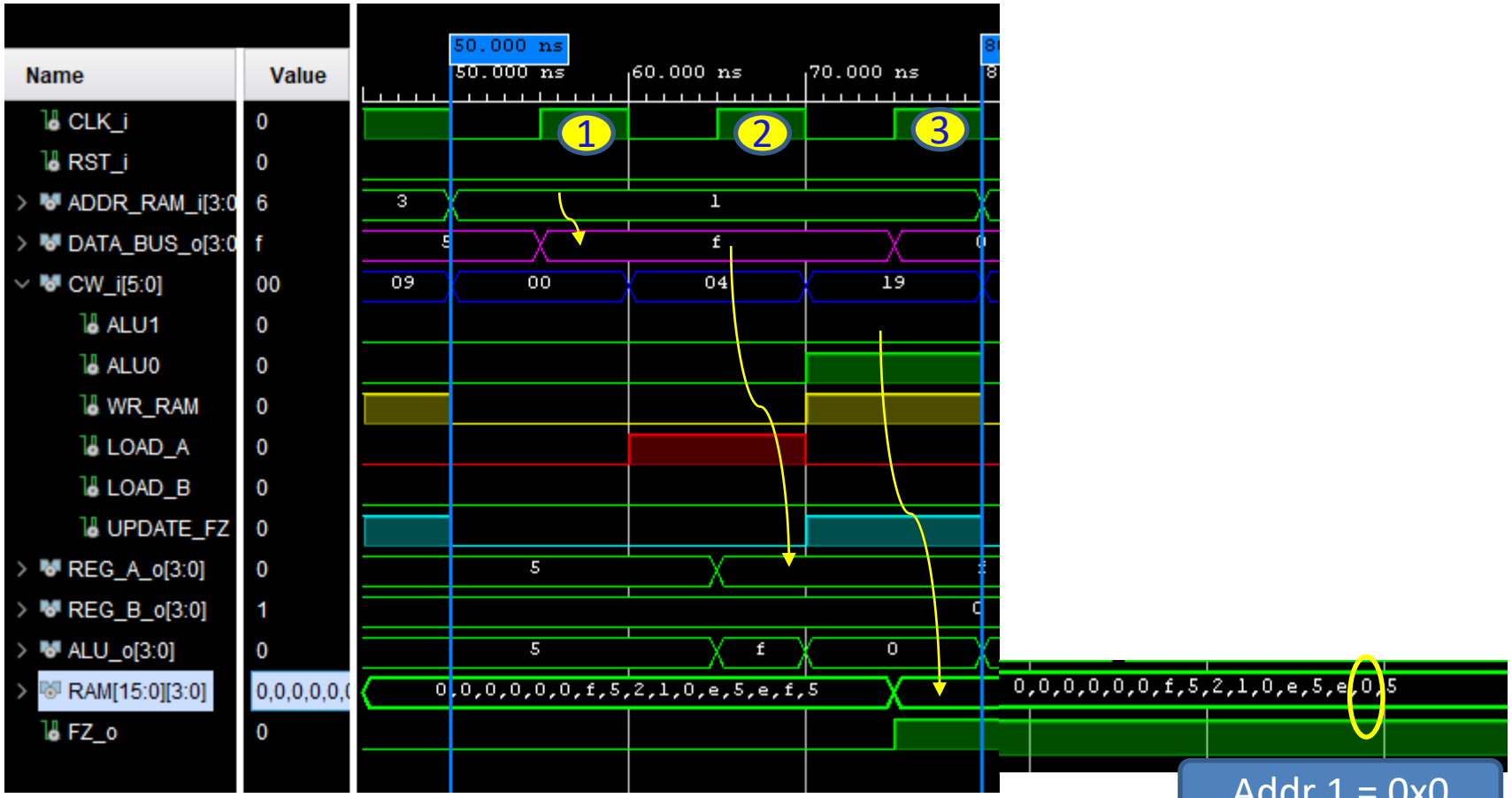
Adding a RAM to DATAPATH: Preparing the testbench

	INC A	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	*INC operation and update FZ*	01	1	0	0	1

```
-- 2) INC A---------------------
-
-- CLK1: Read OPE_A in RAM(1)
     ADDR_RAM_i  <= "0001";
     CW_i        <= "000000";
     wait for 10 ns;
-- CLK2: Load OPE_A
     CW_i        <= "000100";
     wait for 10 ns;
-- CLK3: Select OPE_ALU and FZ
     -- Write result in RAM(1)
     CW_i        <= "011001";
     wait for 10 ns;
```

Adding a RAM to DATAPATH: Preparing the testbench

	INC A	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	*INC operation and update FZ*	01	1	0	0	1



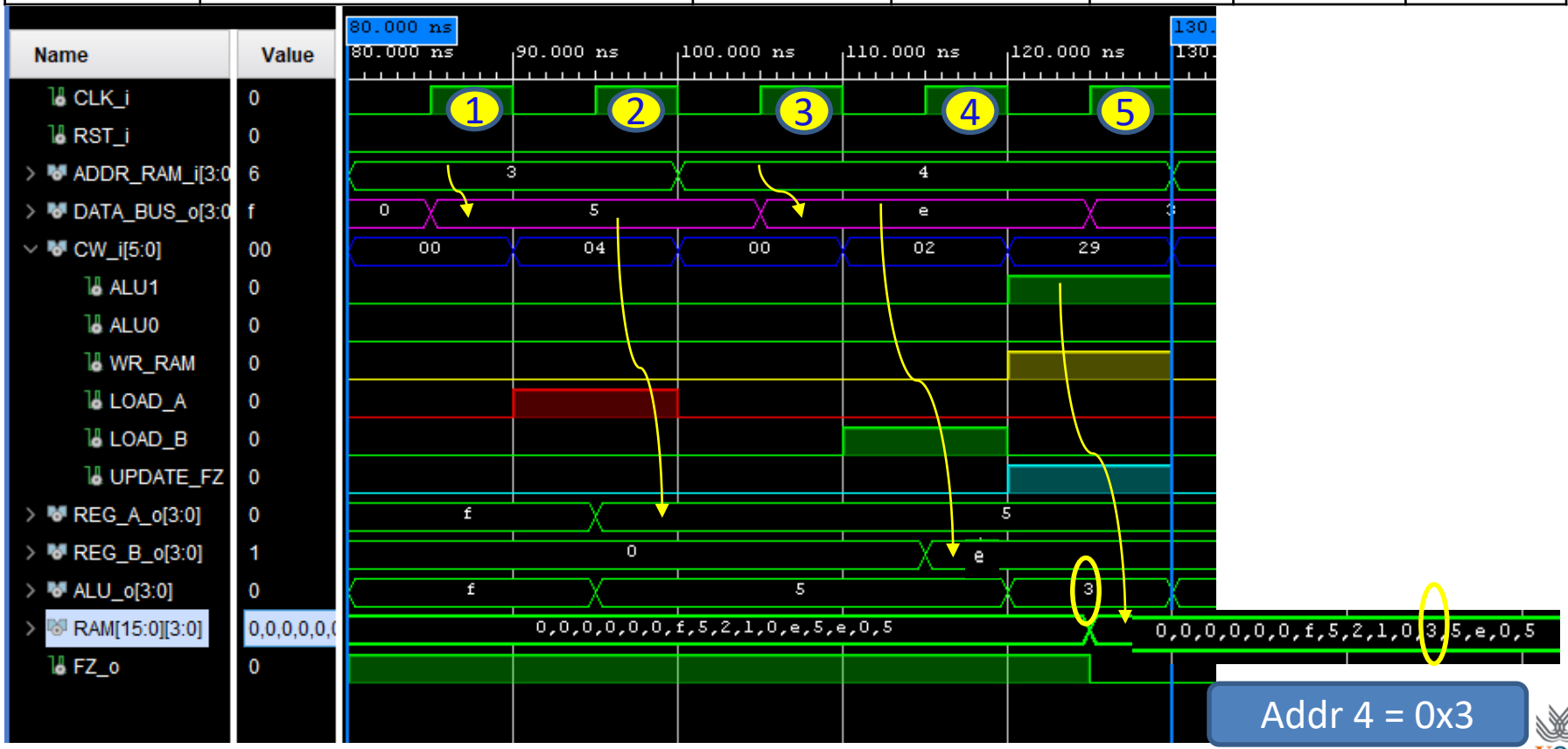
Adding a RAM to DATAPATH: Preparing the testbench

	ADD A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	Set RAM address (OPE_B)	00	0	0	0	0
4	*Load OPE_B*	00	0	0	1	0
5	*ADD operation and update FZ*	10	1	0	0	1

```
-- 3) ADD A,B--------------------
-- CLK1: Read OPE_A in RAM(3)
      ADDR_RAM_i  <= "0011";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK2: Load OPE_A
      CW_i        <= "000100";
      wait for 10 ns;
-- CLK3: Read OPE_B in RAM(4)
      ADDR_RAM_i  <= "0100";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK4: Load OPE_B
      CW_i        <= "000010";
      wait for 10 ns;
-- CLK5: Select OPE_ALU and FZ
      -- Write result in RAM(4)
      CW_i        <= "101001";
      wait for 10 ns;
```

Adding a RAM to DATAPATH: Preparing the testbench

	ADD A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	Set RAM address (OPE_B)	00	0	0	0	0
4	*Load OPE_B*	00	0	0	1	0
5	*ADD operation and update FZ*	10	1	0	0	1



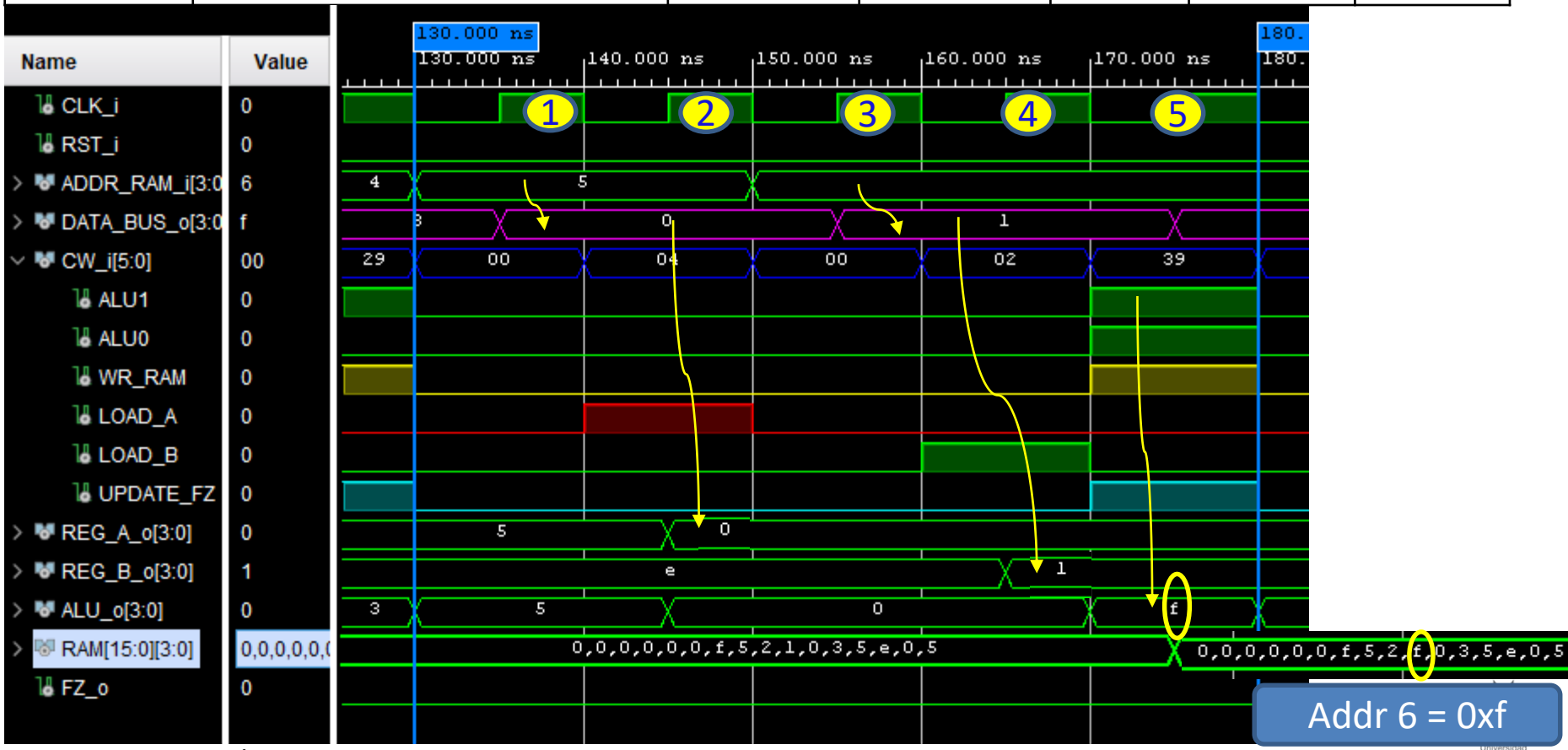
Adding a RAM to DATAPATH: Preparing the testbench

	SUB A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	Set RAM address (OPE_B)	00	0	0	0	0
4	*Load OPE_B*	00	0	0	1	0
5	*SUB operation and update FZ*	11	1	0	0	1

```
-- 4) SUB A,B--------------------
-- CLK1: Read OPE_A in RAM(5)
      ADDR_RAM_i  <= "0101";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK2: Load OPE_A
      CW_i        <= "000100";
      wait for 10 ns;
-- CLK3: Read OPE_B in RAM(6)
      ADDR_RAM_i  <= "0110";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK4: Load OPE_B
      CW_i        <= "000010";
      wait for 10 ns;
-- CLK5: Select OPE_ALU and FZ
      -- Write result in RAM(6)
      CW_i        <= "111001";
      wait for 10 ns;
```

Adding a RAM to DATAPATH: Preparing the testbench

	SUB A,B	ALU_OP	WRITE_RAM	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(3)	CW(2)	CW(1)	CW(0)
1	Set RAM address (OPE_A)	00	0	0	0	0
2	*Load OPE_A*	00	0	1	0	0
3	Set RAM address (OPE_B)	00	0	0	0	0
4	*Load OPE_B*	00	0	0	1	0
5	*SUB operation and update FZ*	11	1	0	0	1



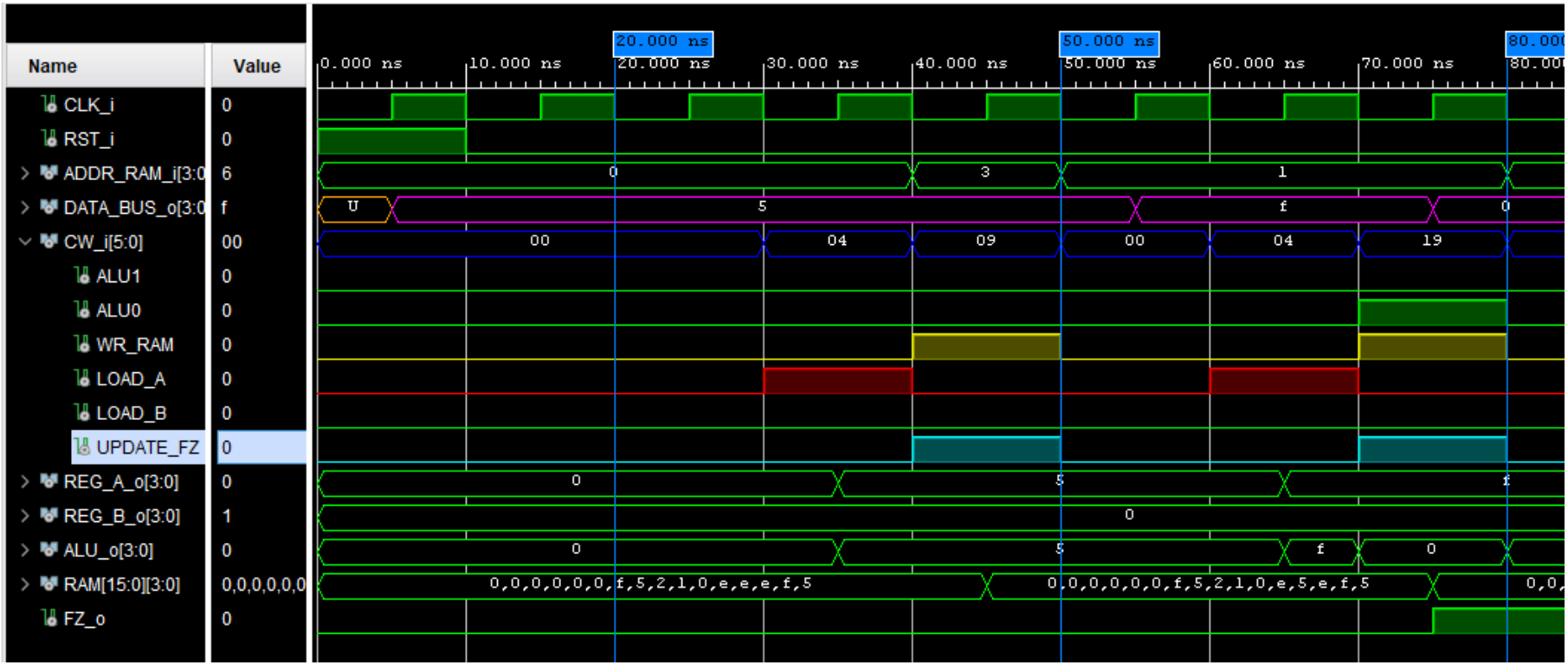
Adding a RAM to DATAPATH: the complete TB

- Add a testbench which sets the proper values for address and CW in each clock-cycle for each operation.

```
stimulus: process
  begin
  CW_i         <= "000000";
  ADDR_RAM_i   <= "0000";
  RST_i <= '1';
  wait for 10 ns;
  RST_i <= '0';
  wait for 10 ns;
-- 1) MOV A,B ---------------------
-- CLK1: Read OPE_A in RAM(0)
    ADDR_RAM_i   <= "0000";
    CW_i         <= "000000";
    wait for 10 ns;
-- CLK2: Load OPE_A
    CW_i         <= "000100";
    wait for 10 ns;
-- CLK3:Select ALU operation and FZ
    -- Write result in RAM (3)
    ADDR_RAM_i   <= "0011";
    CW_i         <= "001001";
    wait for 10 ns;
```

```
-- 2) INC A-----------------------
-- CLK1: Read OPE_A in RAM(1)
     ADDR_RAM_i   <= "0001";
     CW_i         <= "000000";
     wait for 10 ns;
-- CLK2: Load OPE_A
     CW_i         <= "000100";
     wait for 10 ns;
-- CLK3: Select OPE_ALU and FZ
     -- Write result in RAM(1)
     CW_i         <= "011001";
     wait for 10 ns;
```

Adding a RAM to DATAPATH: Complete simulation



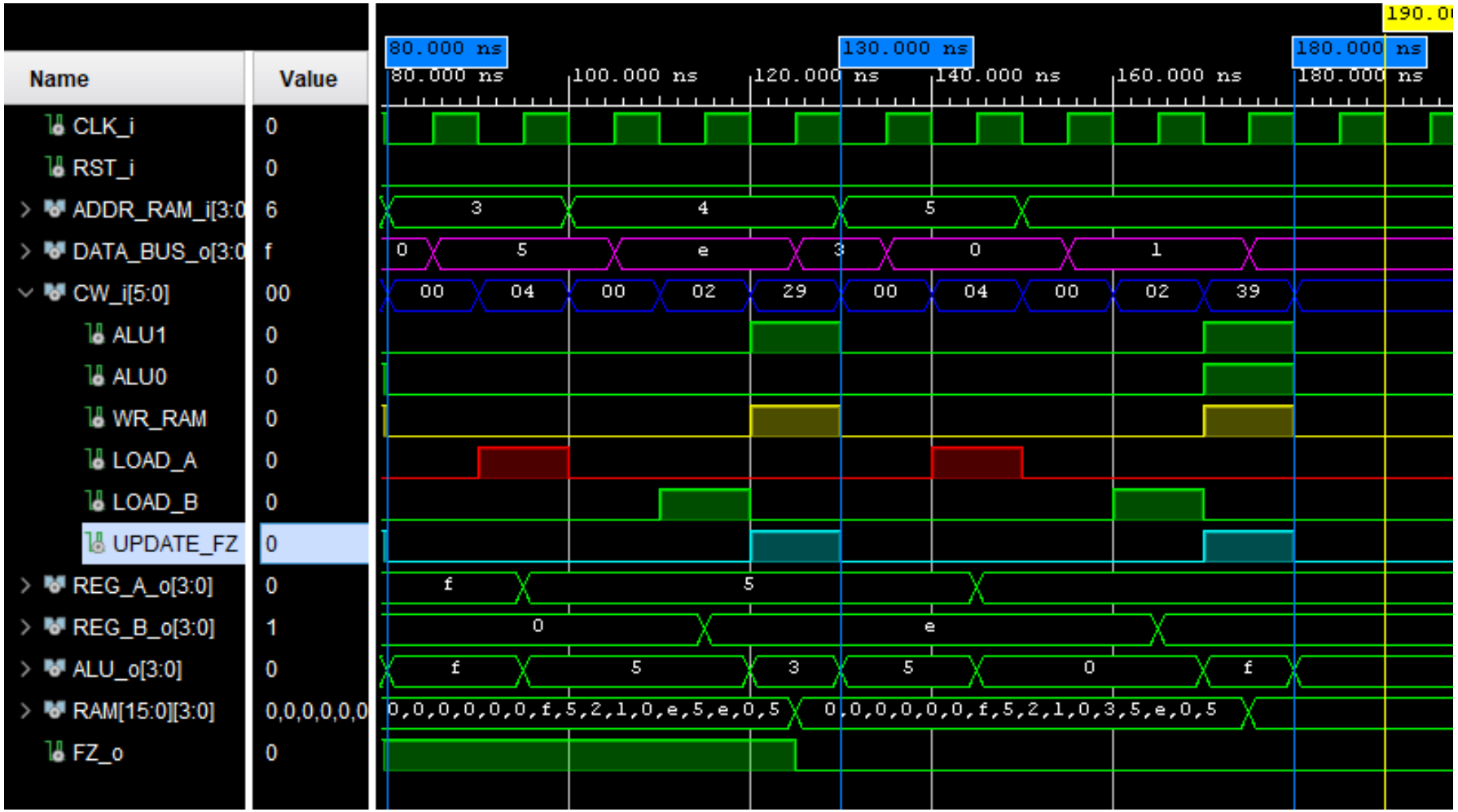
Adding a RAM to DATAPATH

- Add a testbench which sets the proper values for address and CW in each clock-cycle for each operation.

```
-- 3) ADD A,B--------------------
-- CLK1: Read OPE_A in RAM(3)
      ADDR_RAM_i  <= "0011";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK2: Load OPE_A
      CW_i        <= "000100";
      wait for 10 ns;
-- CLK3: Read OPE_B in RAM(4)
      ADDR_RAM_i  <= "0100";
      CW_i        <= "000000";
      wait for 10 ns;
-- CLK4: Load OPE_B
      CW_i        <= "000010";
      wait for 10 ns;
-- CLK5: Select OPE_ALU and FZ
      -- Write result in RAM(4)
      CW_i        <= "101001";
      wait for 10 ns;
```

```
-- 4) SUB A,B --------------------
      -- Read OPE_A in RAM(5)
      ADDR_RAM_i  <= "0101";
      CW_i        <= "000000";
      wait for 10 ns;
      -- Load OPE_A
      CW_i        <= "000100";
      wait for 10 ns;
      --- Read OPE_B in RAM(6)
      ADDR_RAM_i  <= "0110";
      CW_i        <= "000000";
      wait for 10 ns;
       -- Load OPE_B
      CW_i        <= "000010";
      wait for 10 ns;
      -- Select OPE_ALU and FZ
      -- Write result in RAM(6)
      CW_i        <= "111001";
      wait for 10 ns;
      CW_i        <= "000000";
      wait for 10 ns;
```

Adding a RAM to DATAPATH



4.3. Adding a ROM (Program memory) to Didacomp

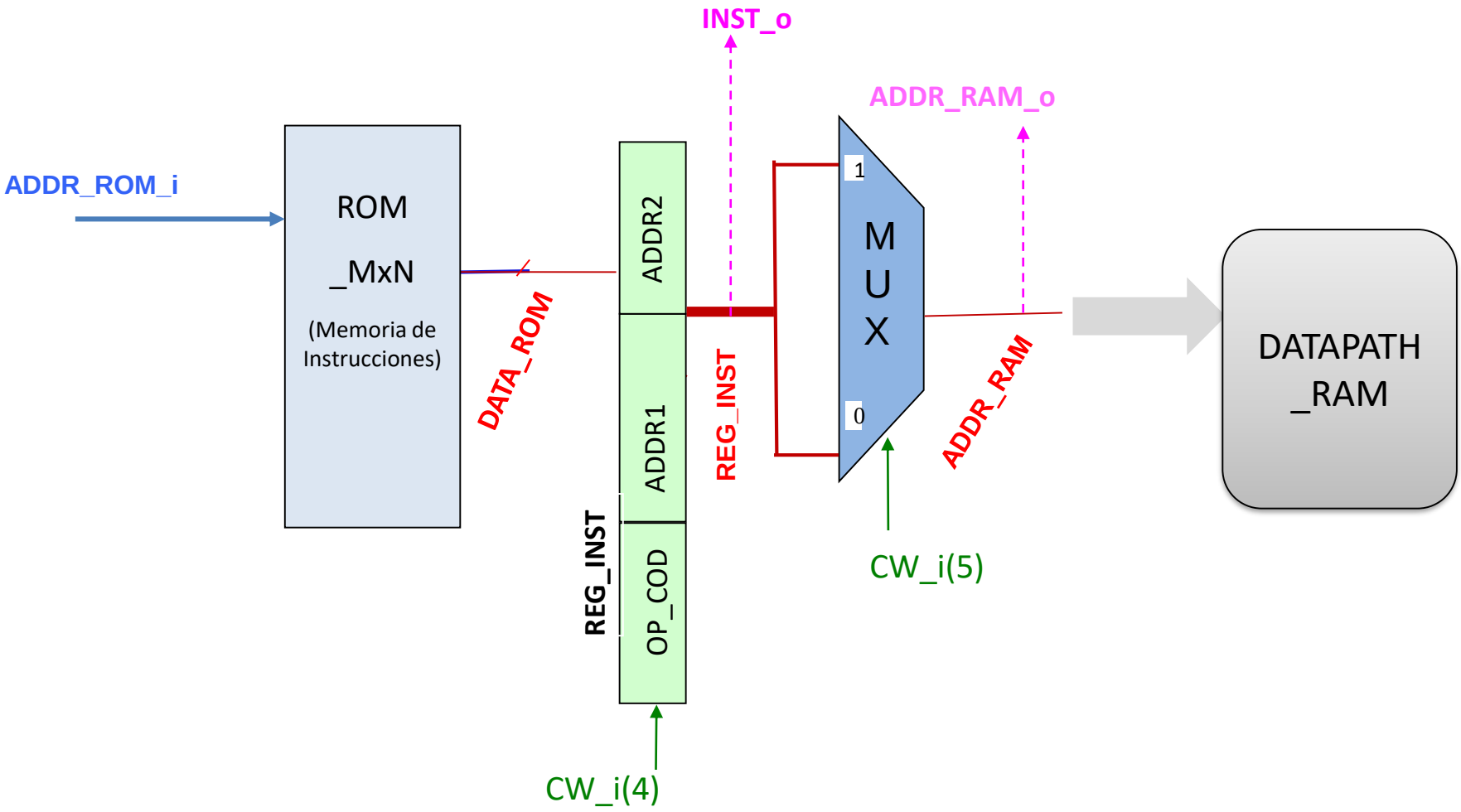
Adding a ROM memory (Program memory)

Create a new Project named P25_DATA_PATH_0_ROM and add one source called "DATAPATH_0_ROM". Copy the code from the file in Project "P24_DATAPATH_0_RAM". Then add the **ROM memory** (instructions), the **instruction register** and the **MUX** for selecting the address in the RAM (operands) as shown in the schematic.

Note: The schematic and the ROM contents are provided in the following slides

Adding ROM memory, instruction register and address RAM selection

P25_DATA_PATH_0_ROM



Contents of ROM memory

P25_DATA_PATH_ROM → ROM (instructions) contents

Dir. Hex	Instruction (12 bits)	Instruction functions
0	0010 1000 0011	ADD M[8],M[3] --> M[3]
1	0000 0001 0100	MOV M[1],M[4]
2	0001 0000 1001	INC M[0]
3	0011 0001 0001	SUB M[1],M[1] --> M[1] ; FZ=1
4	0100 0011 0111	BRZ [7] --> PC = 7
5	0000 0100 0000	
6	0010 0001 0010	
7	0010 0001 0010	ADD M[1],M[2] --> M[2]
8	0001 0000 0100	
…	0000 0000 0000	

Contents of ROM memory

P25_DATA_PATH_ROM

Add a testbench for checking the fisrt instruction (Address 0x003) stored in the ROM program memory (SUB M[1],M[1] --> M[1]).

Tips:

- *You can base on the TB of the P24 Project (Lab_4 page 15). Take into account more bits are required for "CW_i", and a new input, "ADDR_ROM_i" must be included instead of "ADDR_RAM_i"*

- *The needed stimuli for each CLK cycle is similar to that in the Bus control table written in "Tema 4_1_Diseño de un procesador: el camino de datos" (page 23). (Notice that in this case, 6 clk-cycles, instead 5, are needed to complete the actions related to the SUB instruction. A different value must be applied to the ALU selection lines in CW_i).*

4.4. Entire microarchitecture of Didacomp

Entire microarchitecture of Didacomp

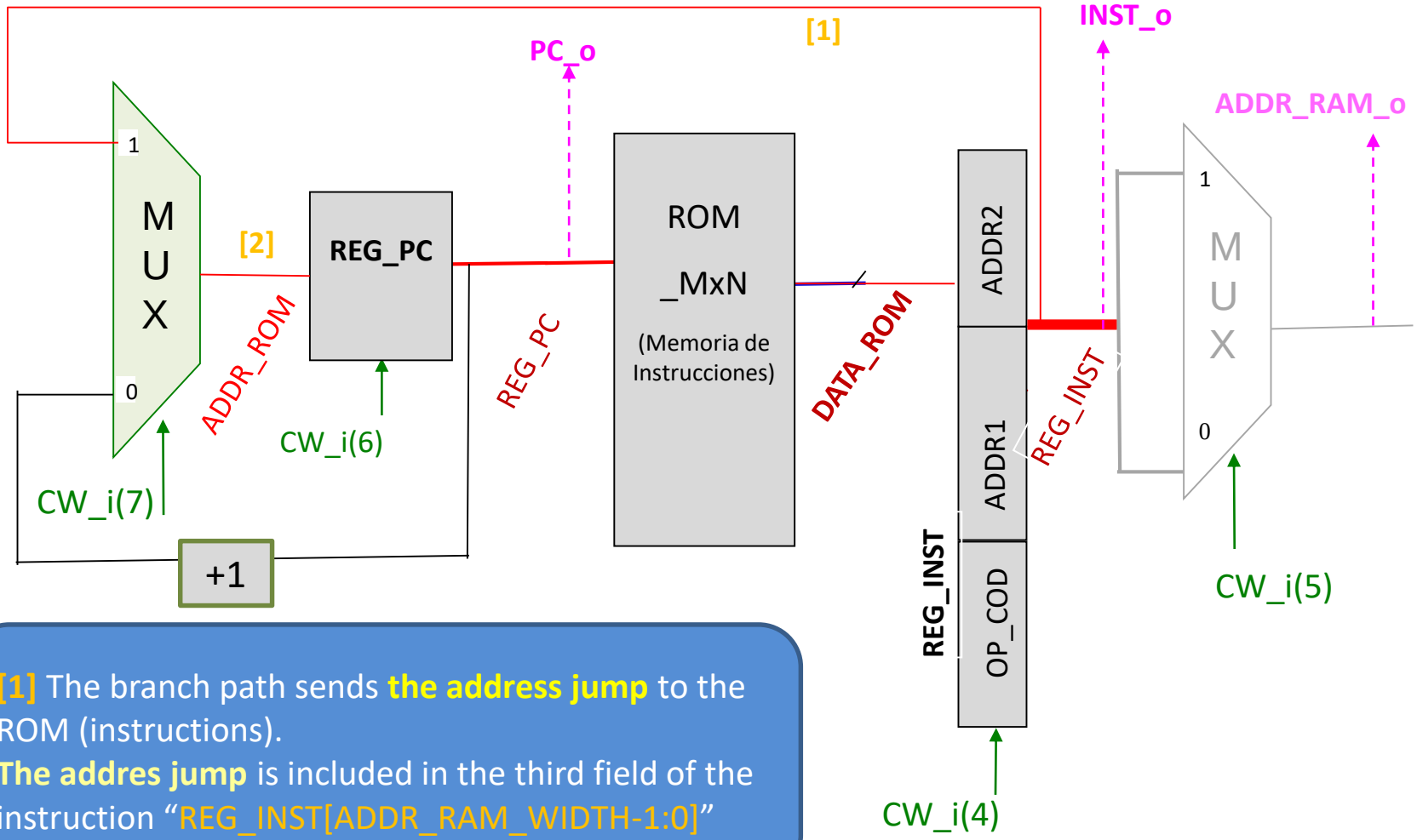
Create a new Project named P26_DIDACOMP_uArq and add one source called "DATAPATH_0". Copy the code from the file "Datapath_0" (P25_DATAPATH_0_ROM). Then add the Program Counter (PC) and the jump structure as shown in the schematic.

Note: The values for the control Word (CW) are available in the lecture 4_1.

In the same file (DATAPATH_0.vhd) add the VHDL code to infer a second source to provide an address to ROM (instructions)

Estructura de salto de DidaComp

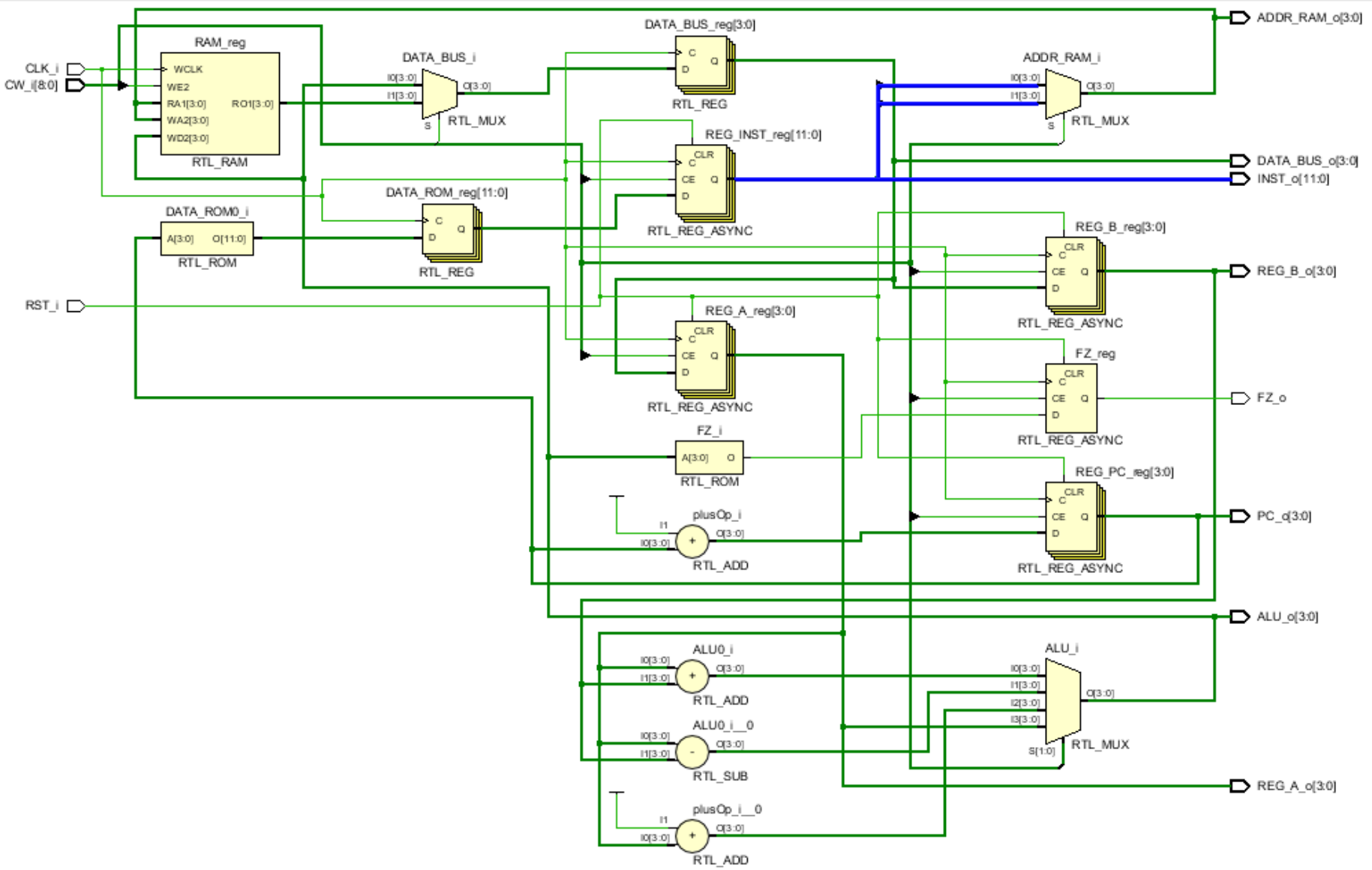
[2] Now, there are two sources of address for the ROM (instructions)



[1] The branch path sends **the address jump** to the ROM (instructions).
The addres jump is included in the third field of the instruction "REG_INST[ADDR_RAM_WIDTH-1:0]"

Entire microarchitecture of Didacomp

Open the RTL Schematic and make sure you have something similar to the below image:



Didacomp microarquitecture

Create a new Project **P27_Didacomp_uArq_Nexys4**. Add a module called "TOP" to connect the "DIDACOMP_uArq" and "Disp7Seg_8ON", to use the 7-segment displays to show the values in different points of Didacomp uArq.

- To avoid bouncing in the pushbuttons, add the component "RISING_EDGE".
- In order to test the design onto ***Nexys4***, add a constraints file to associate the ports with the below peripherals:

  - Sw15-14 → CW_i(9),CW_(8) → ALU
  - Sw3-0 → CW_i(7), CW_i(5), CW_i(4), CW_i(3)
  - BTN → Down: CW_i(6) Left: CW_i(2)
    Up: CW_i(1) Right: CW_i(0)
    Central: RST_i

  - Led 11-0 → INST_o
  - Led 15 → FZ_o

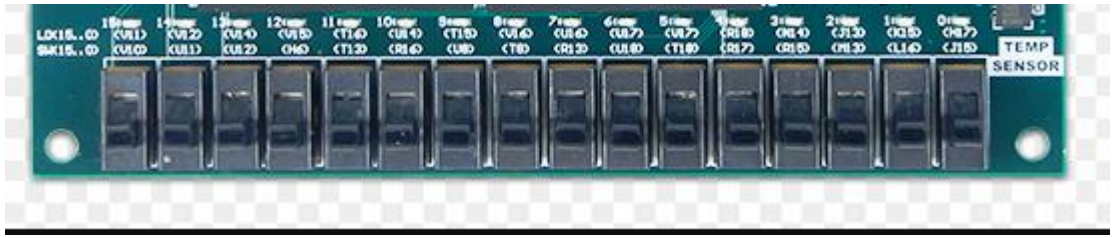
  - Display5 → PC_o
  - Display4 → ADDR_RAM_o
  - Display3 → DATABUS_o
  - Display2 → REG_A_o
  - Display1 → REG_B_o
  - Display0 → ALU_o

- Do not forget to add the clock constraint.

Microarquitectura de Didacomp

Check the design in Nexys4 executing the whole program contained in the ROM and therefore testing the complete ISA.

TIP: In order to help you to recall the function of each switch or button, you may write the functions in a paper and to place it below the peripherals.



REG_INST
R/W