

**ĐẠI HỌC QUỐC GIA HÀ NỘI**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**  
**KHOA KỸ THUẬT MÁY TÍNH**



# BÁO CÁO BÀI TẬP LỚN CUỐI KỲ

**Sinh viên thực hiện** : Nguyễn Công Tân 19020620 (Leader)  
Tạ Đình Đức Hiếu 19020543 (Contributor)

**Môn học** : Nhập môn hệ thống nhúng (ELT3240 1)

**Lớp** : K64K1

**GV hướng dẫn** : Nguyễn Kiêm Hùng

# REPORT

## Design and Implementation of a Simple Processor

Ver 2.0

10/01/2022

	Full name	Function		Date
Written by	Nguyễn Công Tân			10/01/2022
Verified by				
Approved byz				

## Document History

Version	Time	Revised by	Description
V1.0	15/11/2021	Nguyễn Kiêm Hùng	Original Version
V2.0	10/01/2022	Nguyễn Công Tân Tạ Đình Đức Hiếu	- Thêm phần code vào phần 5: Modeling, giải thích code, cách thức hoạt động của các <b>component</b> - Thêm kết quả mô phỏng vào phần 6: Simulation

## Table of Contents

<b>1. Introduction .....</b>	<b>4</b>
<b>2. Requirements.....</b>	<b>4</b>
<b>3. Architecture Design.....</b>	<b>6</b>
3.0. <i>FSMD</i> .....	6
3.1. <i>Datapath architecture</i> .....	7
3.2. <i>Controller</i> .....	9
<b>4. Simple System-on-Chip.....</b>	<b>10</b>
<b>5. Modeling.....</b>	<b>11</b>
5.0. <i>File Sys_definition</i> .....	11
5.1. <i>Component chung bộ ghép kênh multiplexer (Mux3to1)</i> .....	12
5.2. <i>Khối Datapath</i> .....	13
a) Tệp các thanh ghi Register File.....	13
b) Bộ tính toán số học ALU.....	134
c) Khối xử lý dữ liệu Datapath.....	15
5.3. <i>Khối Control_unit</i> .....	17
a) Thanh ghi bộ đếm chương trình Program Counter (PC)	17
b) Thanh ghi câu lệnh Instruction Register	18
c) Bộ điều khiển Controller	19
d) Khối điều khiển Control_Unit	26
5.4. <i>Bộ nhớ Dual Port Memory</i> .....	28
5.5. <i>Bộ vi xử lý ( CPU )</i> .....	29
<b>6. Simulation and Synthesis.....</b>	<b>31</b>

## 1. Introduction

(Introduction to the motivation, Objectives, and main Contents of the project)

**Objective:** Vận dụng các kiến thức, kỹ năng đã được học để thiết kế, mô phỏng và thực thi một bộ vi xử lý 16-bit đơn giản.

## 2. Requirements

Bộ vi xử lý hỗ trợ 16 lệnh cơ bản như được liệt kê trong

**Bảng 1: Cấu trúc tập lệnh.**

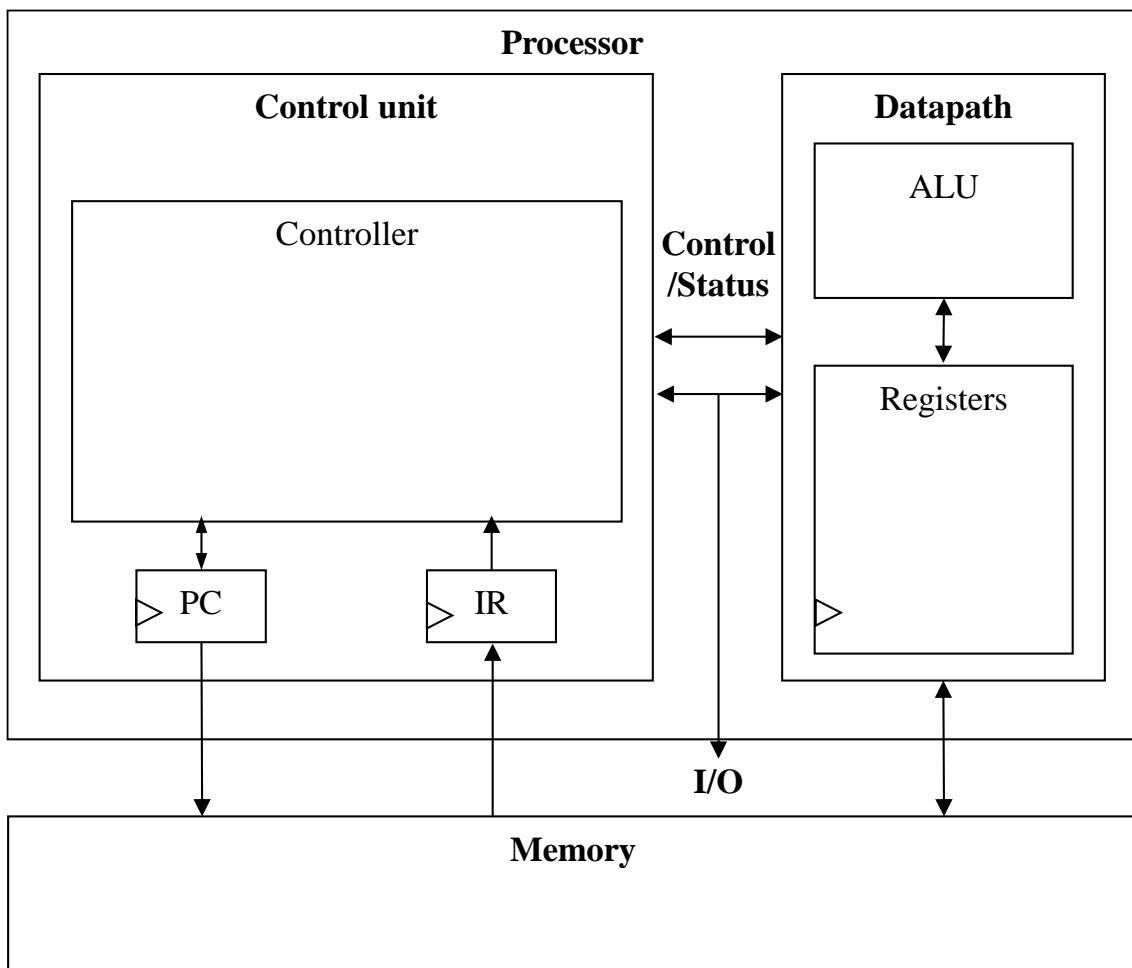
TT	Assembly Instruction	First Byte		Second Byte		Operation
		Opcod e	Operand 1	Operand2		
1	MOV Rn, direct	0000	Rn	direct		Rn = M(direct)
2	MOV direct, Rn	0001	Rn	direct		M(direct) = Rn
3	MOV Rn, @Rm	0010	Rn	Rm		M(Rm) = Rn
4	MOV Rn, #immed	0011	Rn	immediate		Rn = immediate
5	ADD Rn, Rm	0100	Rn	Rm		Rn = Rn + Rm
6	SUB Rn, Rm	0101	Rn	Rm		Rn = Rn - Rm
7	JZ Rn, Addr	0110	Rn	Addr		PC = Addr only if Rn = 0
8	OR Rn, Rm	0111	Rn	Rm		Rn = Rn OR Rm
9	AND Rn, Rm	1000	Rn	Rm		Rn = Rn AND Rm
10	JMP Addr	1010	Rn	Adrr		PC = Addr

Cấu trúc của bộ vi xử lý bao gồm các khối chức năng như trong

**Hình 1**, trong đó:

- Thanh ghi PC (Program Counter): 16-bit, dùng để chứa địa chỉ của lệnh tiếp theo mà bộ vi xử lý sẽ thực hiện
- Thanh ghi IR (Instruction Register): 16-bit, dùng để chứa lệnh mà vi xử lý sẽ thực hiện
- Tệp thanh ghi RF (Register File): 16×16 bit, dùng lưu trữ dữ liệu trong quá trình tính toán của ALU
- ALU (arithmetic and logic unit): hỗ trợ các phép tính trên dữ liệu 16-bit

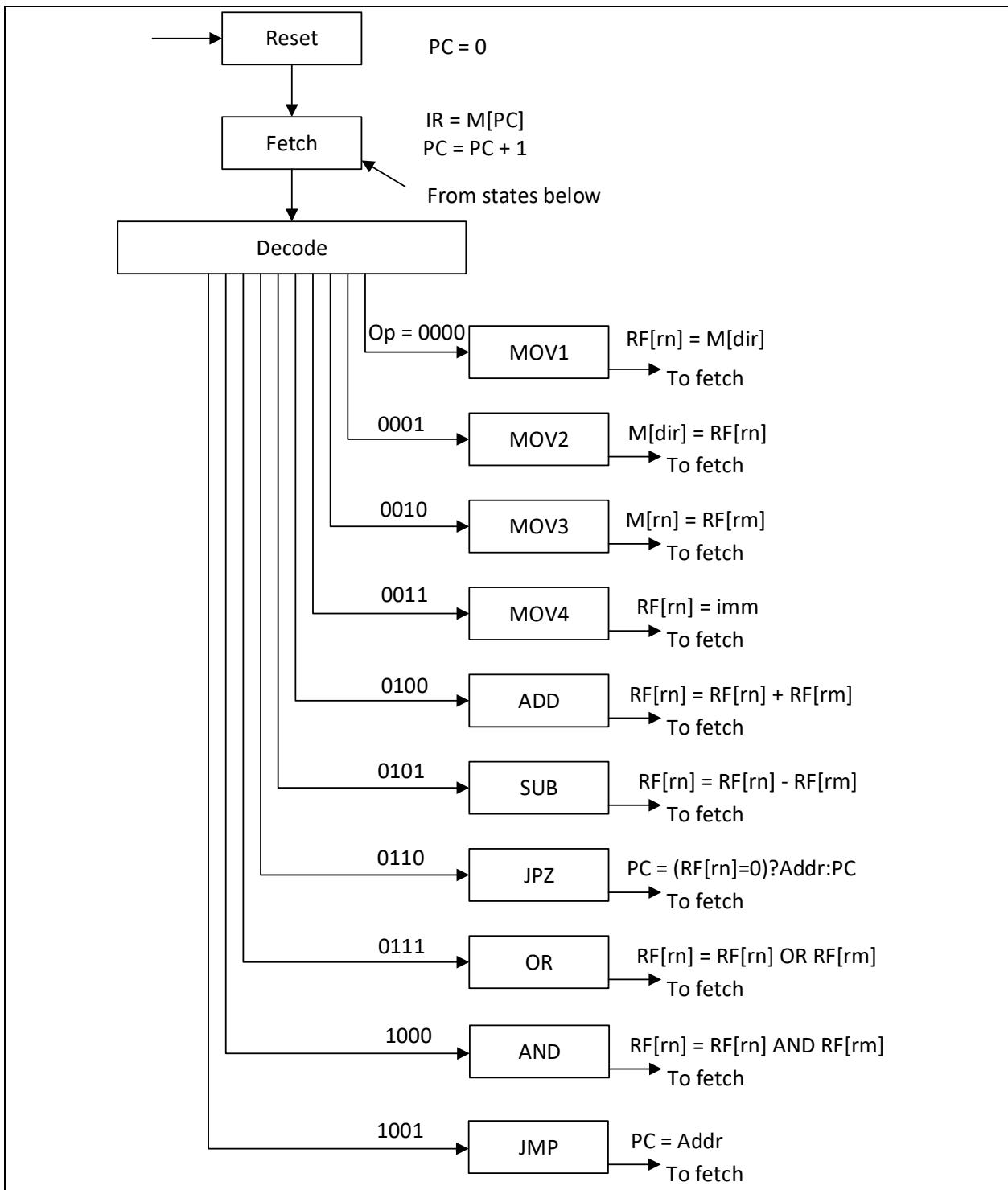
- Bộ nhớ Memory:  $64K \times 16$  bit, dùng để lưu chương trình và dữ liệu cho bộ vi xử lý



**Hình 1: Cấu trúc các khối chức năng cơ bản của bộ vi xử lý**

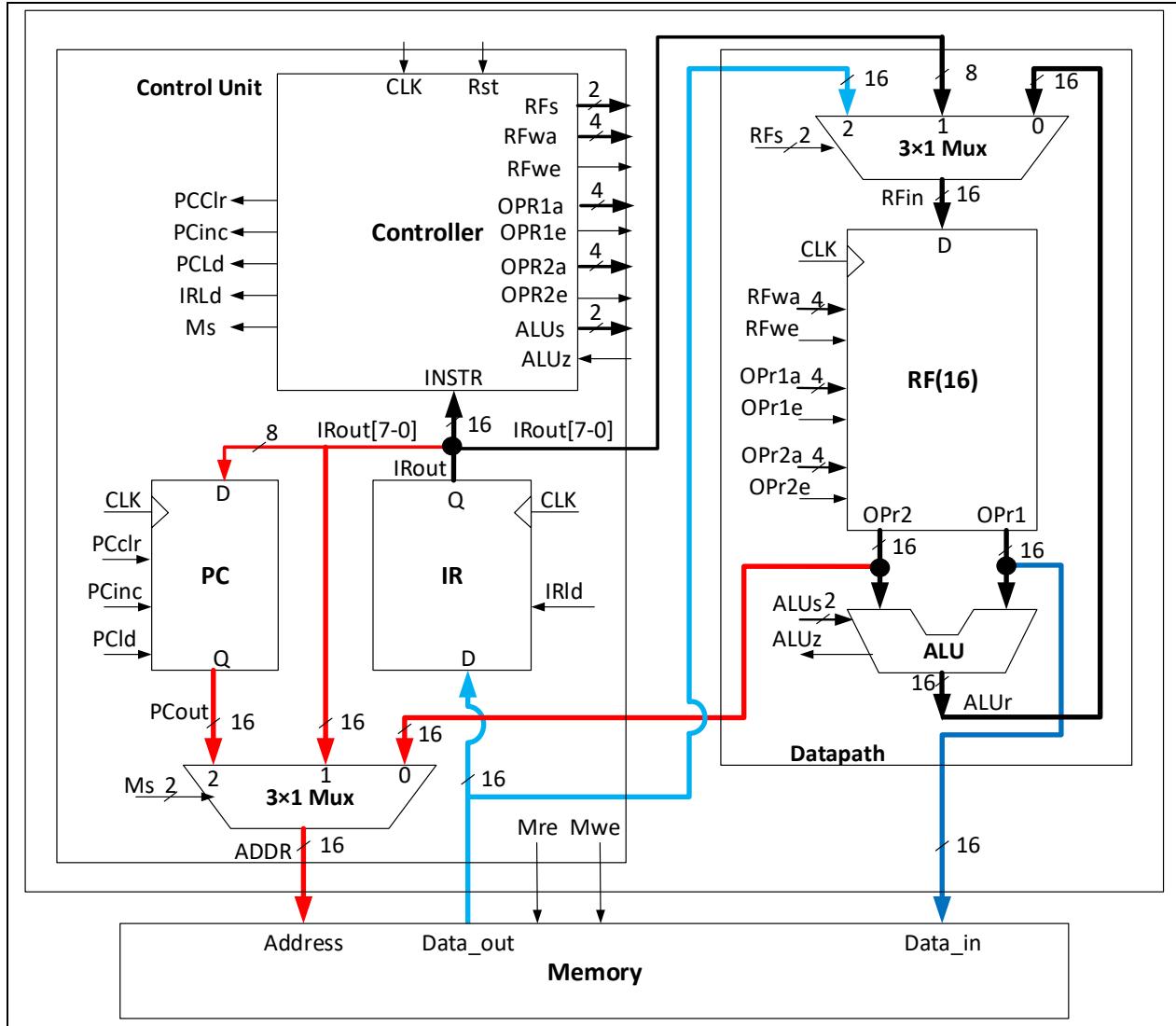
### 3. Architecture Design

#### 3.0. FSMD

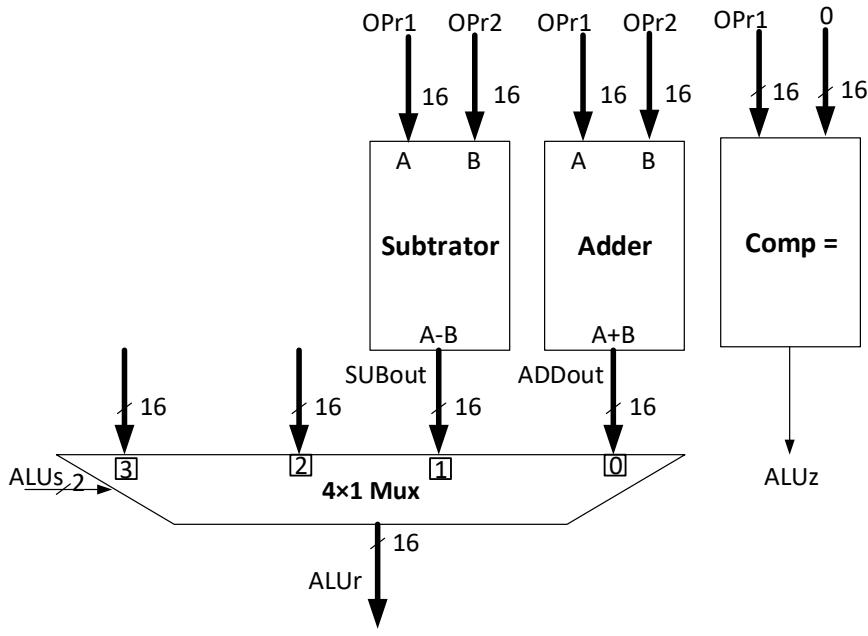


Hình 2: Máy trạng thái FSMD

### 3.1. Datapath architecture

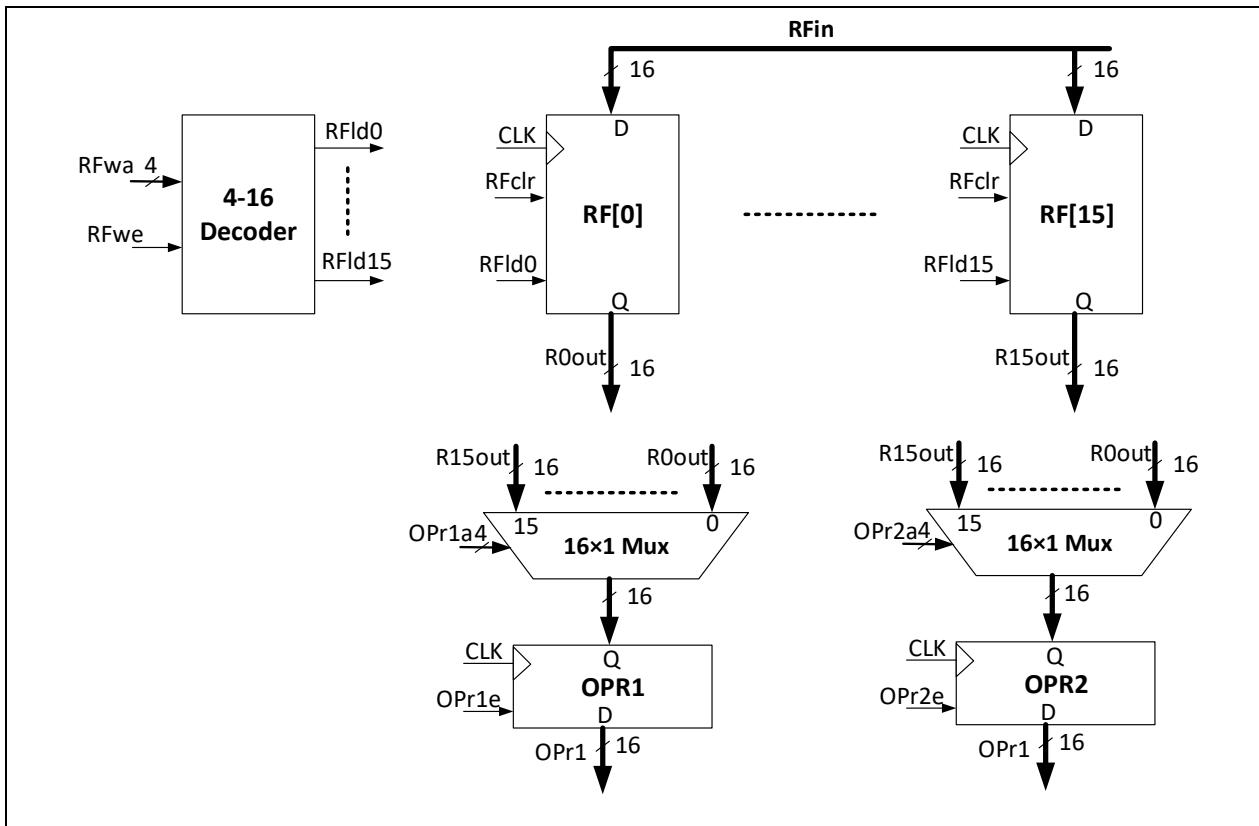


Hình 3: Cấu trúc Datapath



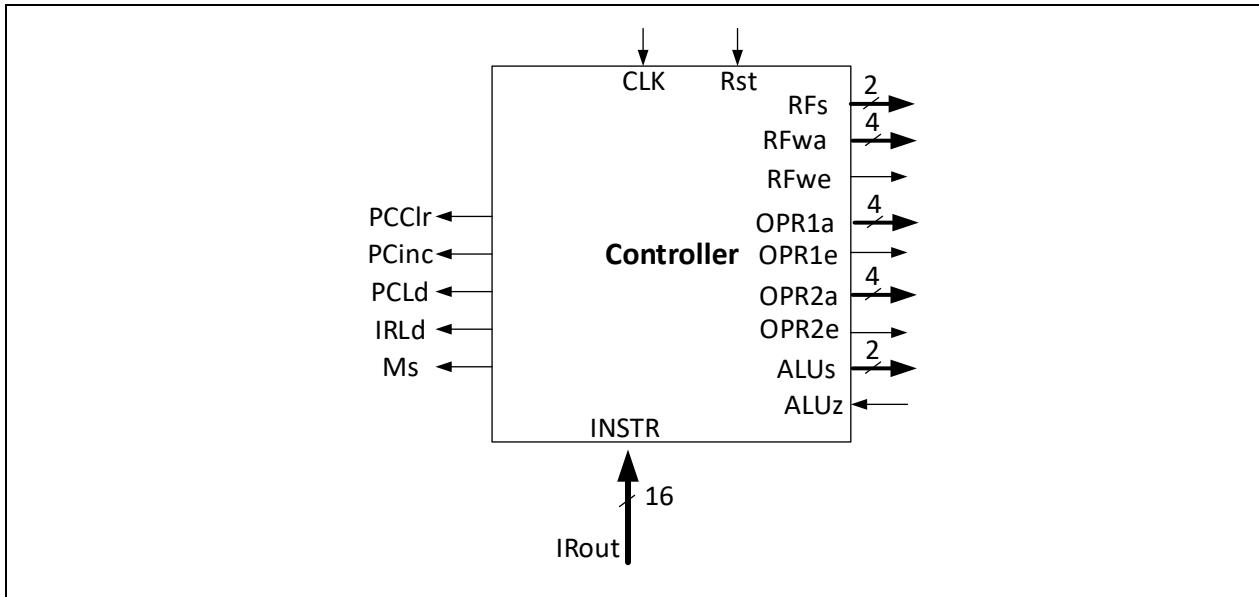
**Hình 5: Các phép tính được hỗ trợ bởi ALU**

ALUs	ALUr
00	$OPr1 + OPr2$
01	$OPr1 - OPr2$
10	$OPr1 \text{ OR } OPr2$
11	$OPr1 \text{ AND } OPr2$

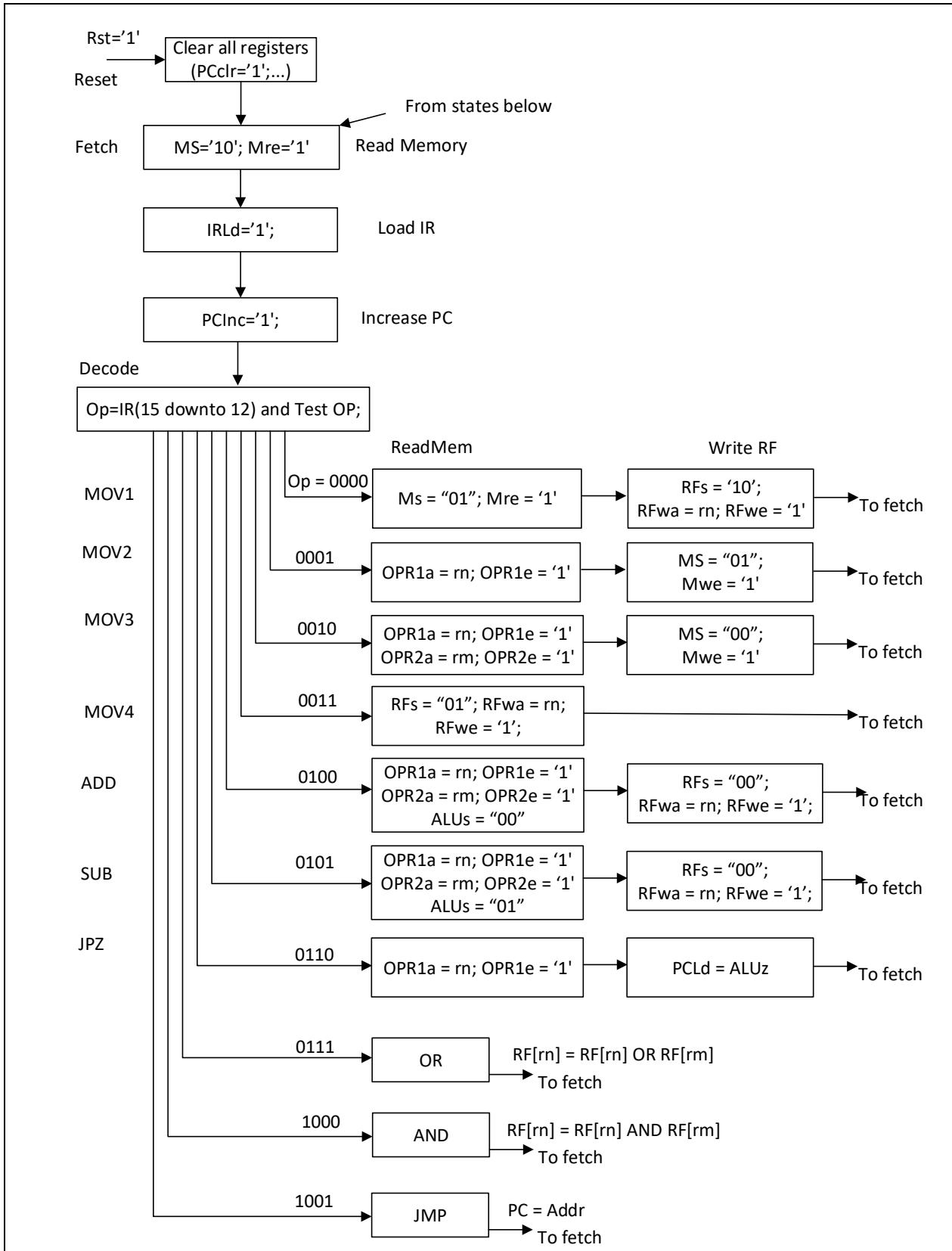


**Hình 6: Tệp thanh ghi RF.**

### 3.2. Controller

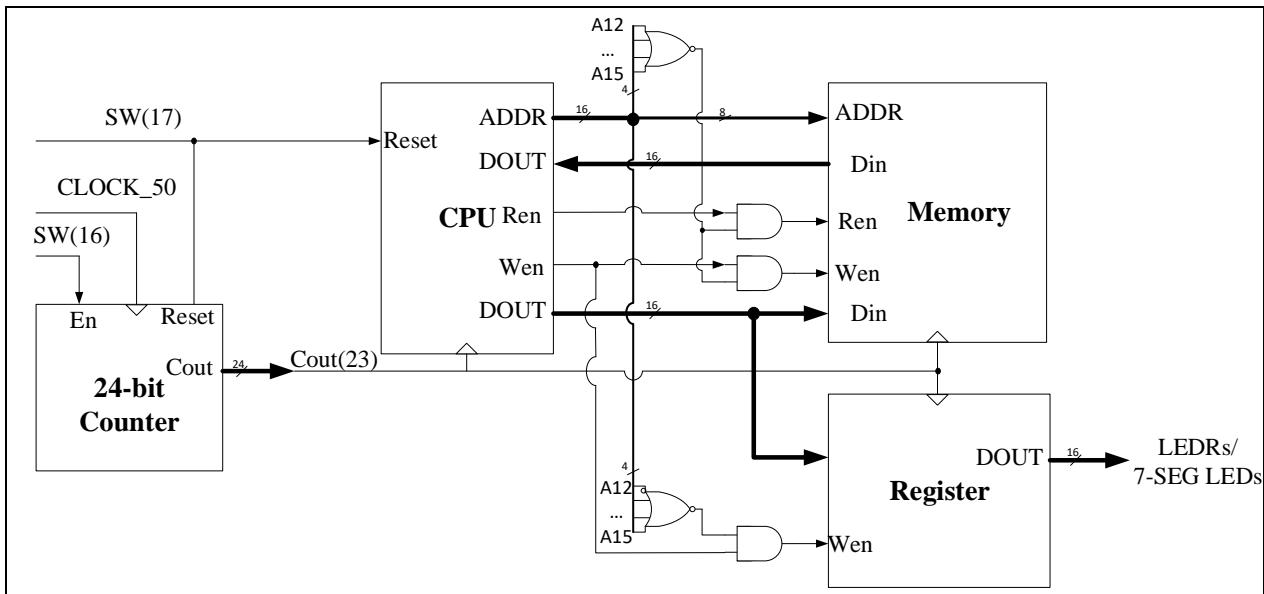


**Hình 4: Giao diện ghép nối vào/ra của controller**



Hình 5: Mô hình máy trạng thái FSM của bộ điều khiển.

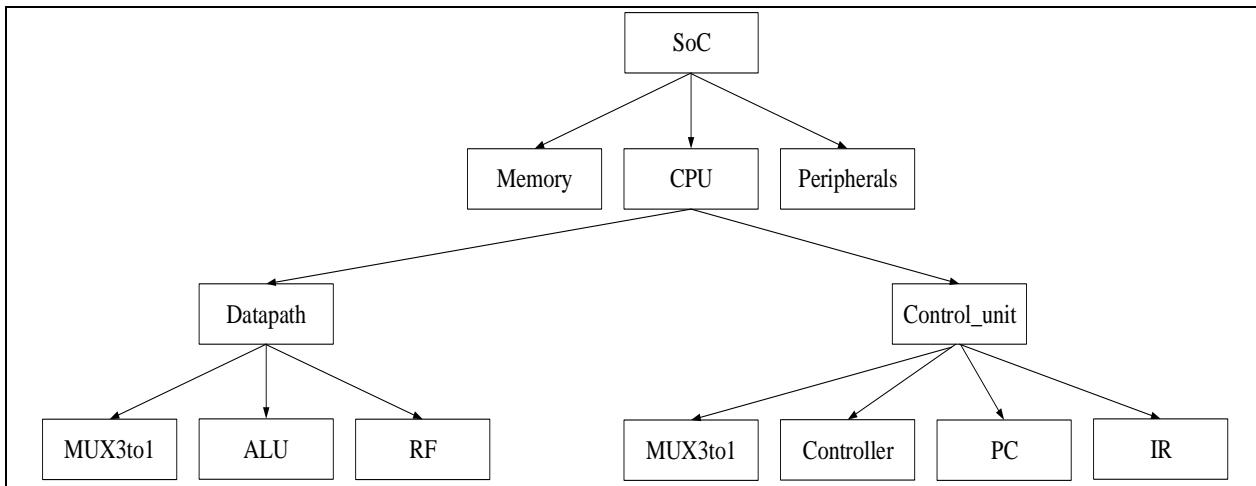
#### 4. Simple System-on-Chip



Hình 6. Thiết kế một hệ thống SoC đơn giản sử dụng CPU phần trên.

## 5. Modeling

(Viết mã nguồn VHDL của project ở đây hoặc trong Appendix A: VHDL Code.)

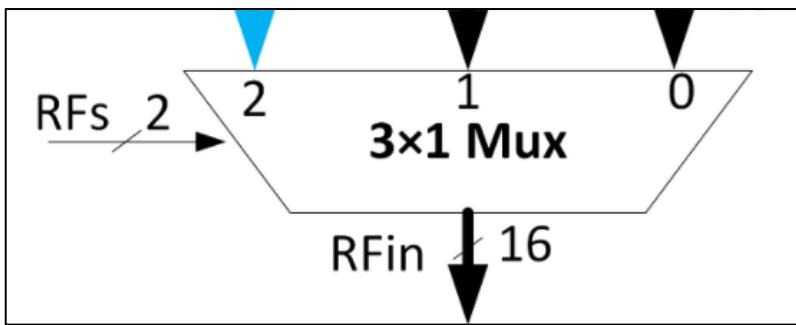


Hình 7. Tô chức của các tệp VHDL

### 5.0. File Sys\_definition

File này có tác dụng như một thư viện, để khai báo tất cả các **component** có trong Project này, khi cần sử dụng một **component** trong 1 file khác thì chỉ cần khai báo file **Sys\_definition**, sau đó chỉ cần port map, không cần khai báo lại file được sử dụng dưới dạng COMPONENT nữa. Tất cả các **component** trong Project này sẽ được khai báo trong **Sys\_definition**

## 5.1. Component chung bộ ghép kênh multiplexer (Mux3to1)



Ý tưởng code:

- Nhìn vào hình ta có thể thấy rằng có 3 tín hiệu input, 1 tín hiệu control có 2 bit để chọn và có 1 tín hiệu output
- Với giá trị của tín hiệu điều khiển thì chọn ra 1 tín hiệu input để chuyển ra tín hiệu output

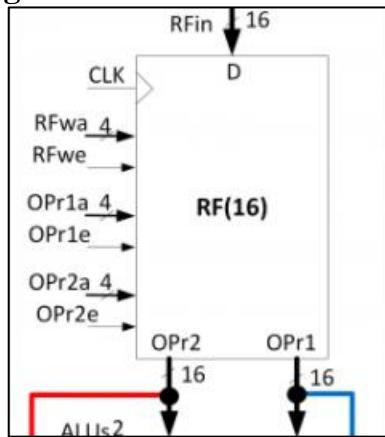
Triển khai code:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  USE ieee.numeric_std.all ;
4  use IEEE.std_logic_unsigned.all;
5  use work.Sys_Definition.all;
6
7  ENTITY mux3to1 IS
8      GENERIC ( DATA_WIDTH : integer := 16);
9      PORT (A, B, C: IN      std_logic_vector (DATA_WIDTH-1 downto 0);
10             SEL : IN      std_logic_vector (1 downto 0);
11             Z : OUT      std_logic_vector (DATA_WIDTH-1 downto 0)
12         );
13 END mux3to1;
14
15 ARCHITECTURE bev OF mux3to1 IS
16 BEGIN
17     proc_mux: PROCESS(A, B, C, SEL)
18     BEGIN
19         IF SEL = "00" THEN
20             Z <= C;
21         ELSIF SEL = "01" THEN
22             Z <= B;
23         ELSIF SEL = "10" THEN
24             Z <= A;
25         ELSE Z <= A;
26         END IF;
27     END PROCESS proc_mux;
28 END bev;
```

## 5.2. Khối Datapath

Khối Datapath bao gồm Register File và ALU

### a) Tệp các thanh ghi Register File



#### Ý tưởng code:

- Đầu tiên ta sẽ khai báo có 16 register mỗi register có 16bit khởi tạo bằng x'0000'
- Đến đây mình sẽ chia ra làm 2 process: 1 process tương ứng với write vào RF, 1 process là read từ RF
- Write\_process:
  - o Khi nào có xung sườn dương CLK và có RFwe= '1' thì sẽ chuyển dữ liệu từ RFin vào thanh ghi trong RF có địa chỉ là RFwa
- Read\_process:
  - o Khi nào mà Opr1e= '1' thì mình sẽ đẩy dữ liệu trong RF có địa chỉ là OPR1a ra bus OPR1
  - o Khi nào mà Opr2e= '1' thì mình sẽ đẩy dữ liệu trong RF có địa chỉ là OPR2a ra BUS OPR2

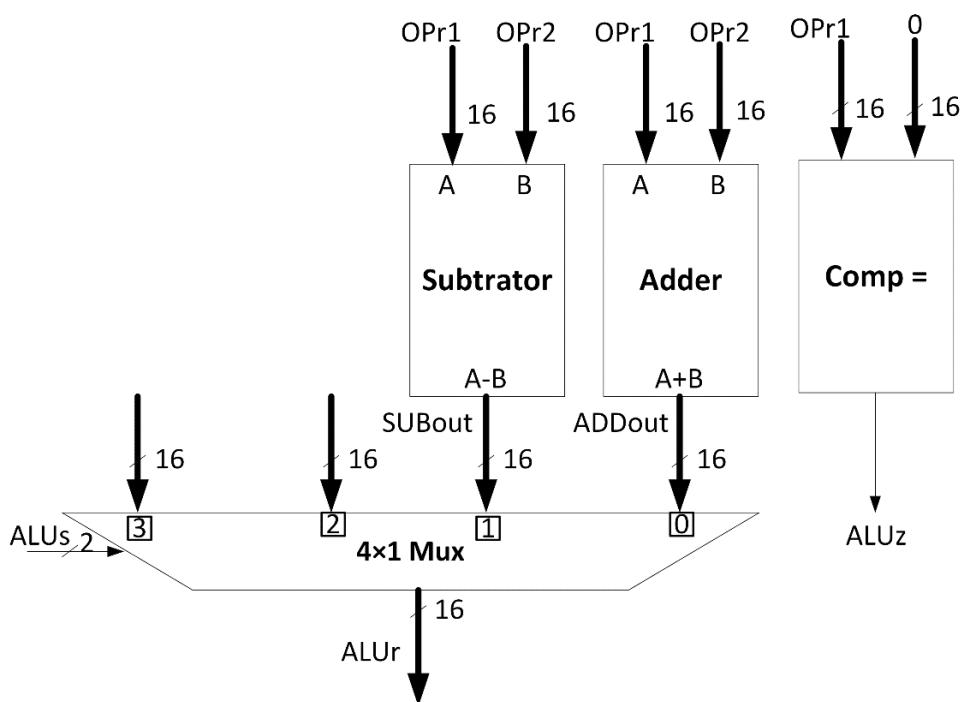
## Triển khai Code thiết kế

```

1  library ieee;
2  use ieee.STD_LOGIC.all;
3  use ieee.numeric_std.all;
4  -----
5  entity register_file is
6  port (
7    clk: in STD_LOGIC;
8    --RFclr: in STD_LOGIC;
9    RFwe, OPr1e, OPr2e: in STD_LOGIC;
10   RFwa, OPr1a, OPr2a: in STD_LOGIC_VECTOR(3 downto 0);
11   RFin: in STD_LOGIC_VECTOR(15 downto 0);
12   OPr1, OPr2: out STD_LOGIC_VECTOR(15 downto 0)
13 );
14 end register_file;
15 -----
16 architecture register_file_arch of register_file is
17 type reg_type is array(0 to 15) of STD_LOGIC_VECTOR(15 downto 0);
18 signal reg_array: reg_type := (others=>(others => '0'));
19 begin
20   write_proc: process(clk, RFwe)
21   begin
22     if clk'event and clk='1' and RFwe ='1' then
23       reg_array(to_integer(unsigned(RFwa))) <= RFin;
24     end if;
25   end process write_proc;
26   --
27   read_proc : process (OPr1e, OPr2e, reg_array, OPr1a, OPr2a)
28   begin
29     if OPr1e = '1' then
30       OPr1 <= reg_array(to_integer(unsigned(OPr1a)));
31     end if;
32     if OPr2e = '1' then
33       OPr2 <= reg_array(to_integer(unsigned(OPr2a)));
34     end if;
35   end process;
36   --
37 end register_file_arch;

```

### b) Bộ tính toán số học ALU



## Ý tưởng code:

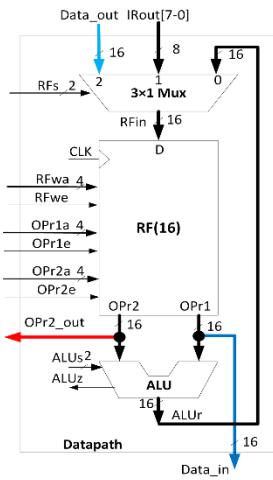
- Từ cái sơ đồ thì ta có thể thấy là ALUs chính là giá trị chọn cho ALU biết nên thực hiện phép tính gì. Vì vậy ta sẽ đưa ALUs vào lệnh case...when để xét. Kết quả sẽ được đưa ra lối ra ALUr.
- Đới với flag ALUz thì mình sẽ so sánh OPR1 với x ‘0000’ nếu bằng thì sẽ cho ALUz = ‘1’ nếu không thì sẽ cho ALUz = ‘0’.

## Triển khai Code thiết kế

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.STD_LOGIC_signed.all;
4  use IEEE.std_logic_arith.all;
5  use work.sys_definition.all;
6
7  ENTITY alu IS
8    port (
9      Opl, Op2: in STD_LOGIC_VECTOR(15 downto 0) ;
10     ALUs: in STD_LOGIC_VECTOR(1 downto 0);
11     ALUz: out STD_LOGIC; -- Zero flag
12     ALUr: out STD_LOGIC_VECTOR(15 downto 0)
13   );
14 END alu;
15
16 ARCHITECTURE alu_arch of alu is
17   signal result: STD_LOGIC_VECTOR(15 downto 0);
18 begin
19   begin
20     process(ALUs, Opl, Op2)
21     begin
22       case ALUs is
23         when "00" =>
24           result <= Opl + Op2; --add
25         when "01" =>
26           result <= Opl - Op2; --subtract
27         when "10" =>
28           result <= Opl and Op2;
29         when others =>
30           result <= Opl or Op2;
31       end case;
32     end process;
33     ALUz <= '1' when Opl = x"0000" else '0';
34     ALUr <= result;
35   end alu_arch;
```

### c) Khối xử lý dữ liệu Datapath

Khối xử lý dữ liệu **Datapath** thực hiện tất cả phép tính được yêu cầu bởi bộ vi xử lý. **Datapath** được cấu tạo bởi các mạch để thực hiện các toán tử số học (cộng, trừ, nhân, so sánh ...) và các phép tính logic (and, or, ..) thông qua bộ **ALU**, bộ hợp kêt **Mux3to1** để điều hướng luồng dữ liệu, tập các thanh ghi **RF** để lưu dữ liệu và kết quả trung gian trong quá trình vi xử lý hoạt động



## Ý tưởng thiết kế code:

- Bây giờ mình đã thiết kế xong các **component RF, ALU, MUX** cần thiết cho **datapath** bây giờ mình sẽ nối các đầu vào đầu ra của các linh kiện đầy bằng lệnh **port map**. Mình sẽ port map các dây tương ứng với 3 component trên như hình.

## Code thiết kế

```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use work.sys_definition.all;
5
6  entity datapath is
7      Generic (
8          DATA_WIDTH : integer := 16; -- Data Width
9          ADDR_WIDTH : integer := 16 -- Address width
10     );
11
12     port (
13         --nReset : in STD_LOGIC;
14         clk : in std_logic;
15         IR_out: in std_logic_vector (DATA_WIDTH - 1 downto 0);
16         Data_out: in std_logic_vector (DATA_WIDTH - 1 downto 0);
17         RFs_net : in std_logic_vector (1 downto 0);
18         RFwa_net, OPr1a_net, OPr2a_net : in std_logic_vector (3 downto 0);
19         RFwe_net, OPr1e_net, OPr2e_net: in std_logic;
20
21         OPr2_out : buffer std_logic_vector (DATA_WIDTH - 1 downto 0);
22         Data_in : buffer std_logic_vector (DATA_WIDTH - 1 downto 0);
23         ALUs_net : in std_logic_vector (1 downto 0);
24         ALUz_net : out std_logic;
25         ALUr_net : buffer std_logic_vector (DATA_WIDTH - 1 downto 0)
26     );
27 end datapath;
28
29 architecture struct of datapath is
30     signal ALUr_net : std_logic_vector (DATA_WIDTH - 1 downto 0);
31     signal OPr1_net : std_logic_vector (DATA_WIDTH - 1 downto 0);
32     signal RFin_net : std_logic_vector (DATA_WIDTH - 1 downto 0);
33 begin
34     -- write your code here
35     -- mux3to1-----
36     MUX_U: mux3to1 port map
37     (
38         SEL => RFs_net,
39         A => Data_out,
40         B => IR_out,
41         C => ALUr_net,
42         Z => RFin_net
43     );
44     -- register file-----
45     RF_U: register_file port map
46     (
47         clk => clk,
48         RFin => RFin_net,
49         RFwa => RFwa_net,
50         RFwe => RFwe_net,
51         OPr1a => OPr1a_net,
52         OPr1e => OPr1e_net,
53         OPr2a => OPr2a_net,
54         OPr2e => OPr2e_net,
55         OPr2 => OPr2_out,
56         OPr1 => Data_in
57     );
58     -- alu-----
59     ALU_U: alu port map
60     (
61         Opl => Data_in,
62         Op2 => OPr2_out,
63         ALUs => ALUs_net,
64         ALUz => ALUz_net,
65         ALUr => ALUr_net
66     );
67 end struct;

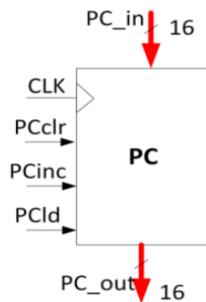
```

### 5.3. Khối Control\_unit

Khối điều khiển **Control\_Unit** bao gồm bộ điều khiển **Controller**, Thanh ghi bộ đếm chương trình **PC**, thanh ghi câu lệnh **IR**

#### a) Thanh ghi bộ đếm chương trình Program Counter (PC)

- Có độ rộng là 16-bit, dùng để chứa địa chỉ của lệnh tiếp theo mà bộ vi xử lý sẽ thực hiện



Bảng chân lý

Pcclr	PCld	PCincr	CLK	PC_in	PC_out
1	X	X	X	X	0
0	1	X	↑	-	PC_in
0	0	1	↑	X	PC_out + 1
0	0	0	↑	X	PC_out

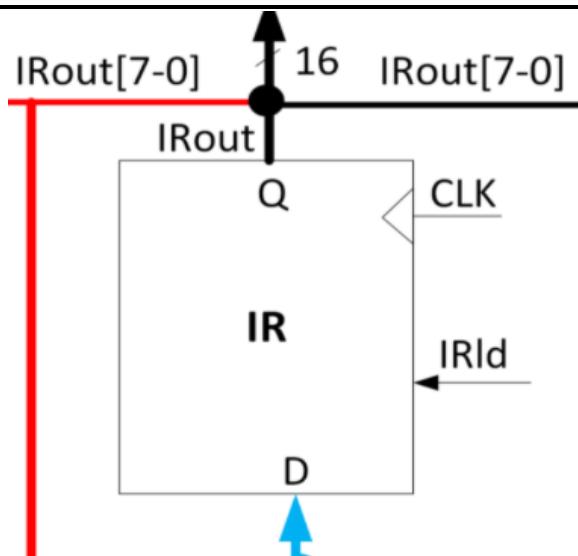
**Ý tưởng code:** Nhìn vào bảng chân lý ta thấy rằng:

- Cứ khi nào có signal PCclr = '1' thì PC\_out = 0 mà không cần quan tâm đến signal khác
- Nếu PCclr = '0' thì khi mà có xung clk sùm dương cộng với:
  - PCld = '1' thì dù cho PCincr bằng mấy thì PC\_out = PC\_in
  - PCld= '0' và PCincr= '1' thì tăng PC\_out lên 1

## Triển khai Code thiết kế

```
1  Library ieee;
2  Use ieee.std_logic_1164.all;
3  Use ieee.std_logic_unsigned.all;
4  Use ieee.std_logic_arith.all;
5
6  entity pc is
7      port(
8          clk : in std_logic;
9          PC_in : in std_logic_vector(15 downto 0);
10         PCclr : in std_logic; --Active low reset signal
11         PCld : in std_logic;
12         PCinc : in std_logic;
13         PC_out: out std_logic_vector(15 downto 0)
14     );
15 end pc;
16
17 architecture pc_arch of pc is
18     signal pc_temp: STD_LOGIC_VECTOR(15 downto 0):=x"0000";
19 begin
20     process(clk, PCclr)
21     begin
22         IF PCclr='1' THEN
23             pc_temp <=x"0000";
24         ELSIF (clk'event and clk='1') THEN
25             IF PCld='1' THEN
26                 pc_temp <= PC_in;
27             ELSIF PCinc='1' THEN
28                 pc_temp <= pc_temp + x"0001";
29             END IF;
30         END IF;
31     END PROCESS;
32
33     PC_out <= pc_temp;
34 end pc_arch;
```

### b) Thanh ghi câu lệnh Instruction Register



## Bảng chân lý

IRld	CLK	IR_in	IR_out
1	↑	-	IR_in
0	↑	X	IR_out

**Ý tưởng code:** Nhìn vào bảng chân lý ta thấy rằng:

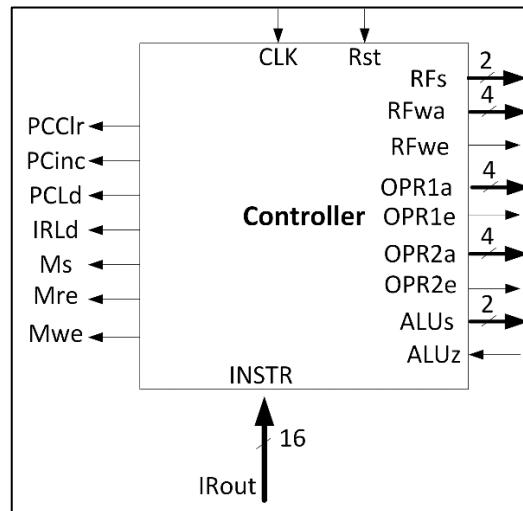
- Với mỗi xung lên CLK thì khi mà IRld = '1' thì IR\_out = IR\_in còn không thì vẫn giữ nguyên IR\_out

## Triển khai Code thiết kế

```
1  Library ieee;
2  Use ieee.std_logic_1164.all;
3  -----
4  entity ir is
5  port(
6      clk : in std_logic;
7      IR_in : in std_logic_vector(15 downto 0);
8      IRld : in std_logic;
9      IR_out: out std_logic_vector(15 downto 0)
10     );
11 end ir;
12 -----
13 architecture ir_arch of ir is
14 begin
15 begin
16 process(clk, IRld)
17 begin
18 IF clk'event and clk='1' and IRld='1' THEN
19     IR_out <= IR_in;
20 END IF;
21 END PROCESS;
22 end ir_arch;
```

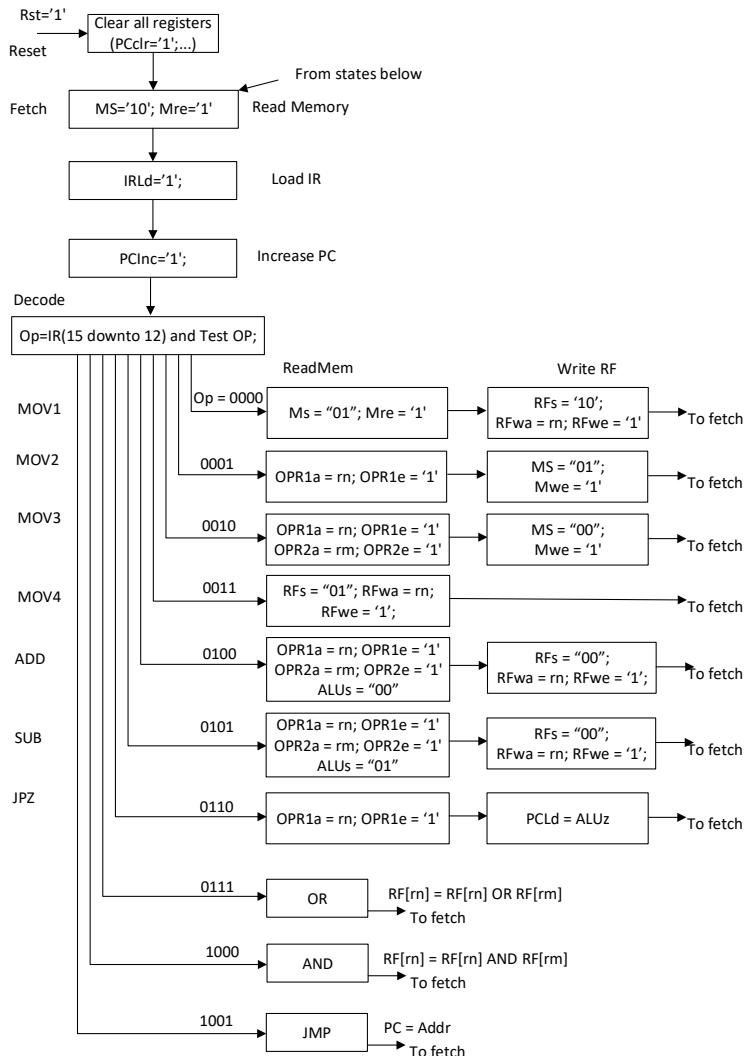
### c) Bộ điều khiển Controller

- Bộ Controller điều khiển quá trình đọc lệnh từ bộ nhớ vào thanh ghi IR sau đó tiến hành giải mã lệnh và tạo ra các tín hiệu điều khiển hoạt động của datapath nhằm thực hiện lệnh được nạp trong IR.
- Và đây cũng là bộ làm mất nhiều thời gian của bọn em nhất mất tận hơn 500 dòng code nên em sẽ đưa vào những cái cốt lõi của code vào báo cáo.



### Các bước thiết kế bộ Controller:

- Chúng ta chú ý là quá trình thực hiện các lệnh của vi xử lý gồm **5 stage** cơ bản sau
  - o **Fetch Instruction:** Đọc lệnh vào IR
  - o **Decode Instruction:** Giải mã lệnh và xác định CPU nên thực hiện thao tác nào
  - o **Fetch Operands:** Nạp toán hạng
  - o **Execute Operation:** Thao tác trên ALU và lưu kết quả ra bộ nhớ
  - o **Store Results:** Lưu nội dung thanh ghi vào bộ nhớ
- Bộ điều khiển được thiết kế dưới dạng máy trạng thái, được chuyển trạng thái liên tục sau mỗi **CLOCK CYCLE**, Bộ **Controller** sẽ nhận 2 lối vào tín hiệu điều khiển đó là **Clock** và **Reset**, 1 lối vào mang dữ liệu câu lệnh đó là **INSTR**

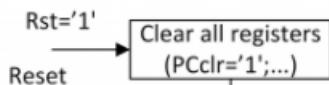


- **5 stage** thực hiện 1 câu lệnh sẽ được **chia nhỏ** ra thành các trạng thái

```

26 type state is
27   (
28     reset, fetch, decoder, loadIR, incPC,
29     readMem_1, readMem_2, readMem_3, readMem_4, readMem_add, readMem_sub, readMem_jpz, readMem_or, readMEM_and, jmp,
30     writeRF_1, writeRF_2, writeRF_3, writeRF_add, writeRF_sub, writeRF_jpz, writeRF_or, writeRF_and
31   );
32   signal current_state : state;
  
```

- **Reset** là trạng thái khởi tạo, toàn bộ tín hiệu ra được đưa về 0

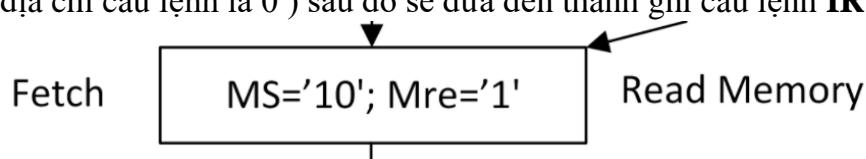


```

121    case current_state is
122    -----
123    when reset =>
124        -- program counter control signal
125        PCclr <= '1'; -- active
126        PCld <= '0';
127        PCinc <= '0';
128        -- instruction register control signal
129        IRld <= '0';
130        -- dual port memory signal
131        Mre <= '0';
132        Mwe <= '0';
133        -- register file control signal
134        RFwe <= '0';
135        OPrle <= '0';
136        OPr2e <= '0';

```

- o **Fetch** là trạng thái nạp địa chỉ câu lệnh vào **MEM** để đọc, khi này thì bộ **Mux3to1** ở khối **Control\_Unit** sẽ chọn đầu vào từ **PC** là dữ liệu để đưa đến lối vào **ADDR** của bộ nhớ ( lần nạp đầu tiên sau **Reset** thì giá trị của địa chỉ câu lệnh là 0 ) sau đó sẽ đưa đến thanh ghi câu lệnh **IR**



```

137 when fetch =>
138     -- select signal
139     Ms <= "10"; --active
140     -- program counter control signal
141     PCclr <= '0';
142     PCld <= '0';
143     PCinc <= '0';
144     -- instruction register control signal
145     IRld <= '0';
146     -- dual port memory signal
147     Mre <= '1'; --active
148     Mwe <= '0';
149     -- register file control signal
150     RFwe <= '0';
151     OPrle <= '0';
152     OPr2e <= '0';

```

- o **IRld** là trạng thái mà thanh ghi câu lệnh sẽ nạp câu lệnh từ trong thanh ghi vào trong khối

IRld='1';	Load IR
-----------	---------

```

153  when loadIR =>
154      -- program counter control signal
155      PCclr <= '0';
156      PCld <= '0';
157      PCinc <= '0';
158      -- instruction register control signal
159      IRld <= '1';-- active
160      -- dual port memory signal
161      Mre <= '0';
162      Mwe <= '0';
163      -- register file control signal
164      RFwe <= '0';
165      OPrle <= '0';
166      OPr2e <= '0';

```

- **PCinc** là trạng thái mà thanh ghi **PC** đếm giá trị địa chỉ tăng **thêm 1**, tại thời điểm này thì giá trị vẫn đang được giữ trong thanh ghi

PCinc='1';	Increase PC
------------	-------------

```

167  when incPC =>
168      -- program counter control signal
169      PCclr <= '0';
170      PCld <= '0';
171      PCinc <= '1';--active
172      -- instruction register control signal
173      IRld <= '0';--
174      -- dual port memory signal
175      Mre <= '0';
176      Mwe <= '0';
177      -- register file control signal
178      RFwe <= '0';
179      OPrle <= '0';
180      OPr2e <= '0';

```

- **Decoder** là trạng thái xử lí câu lệnh được nạp vào, xác định và xử lí **Opcode** để phân biệt loại câu lệnh, **Rn**, **Rm** và địa chỉ của câu lệnh nhảy (nếu có)

begin	opcode <= IRout(15 downto 12);
-------	--------------------------------

```

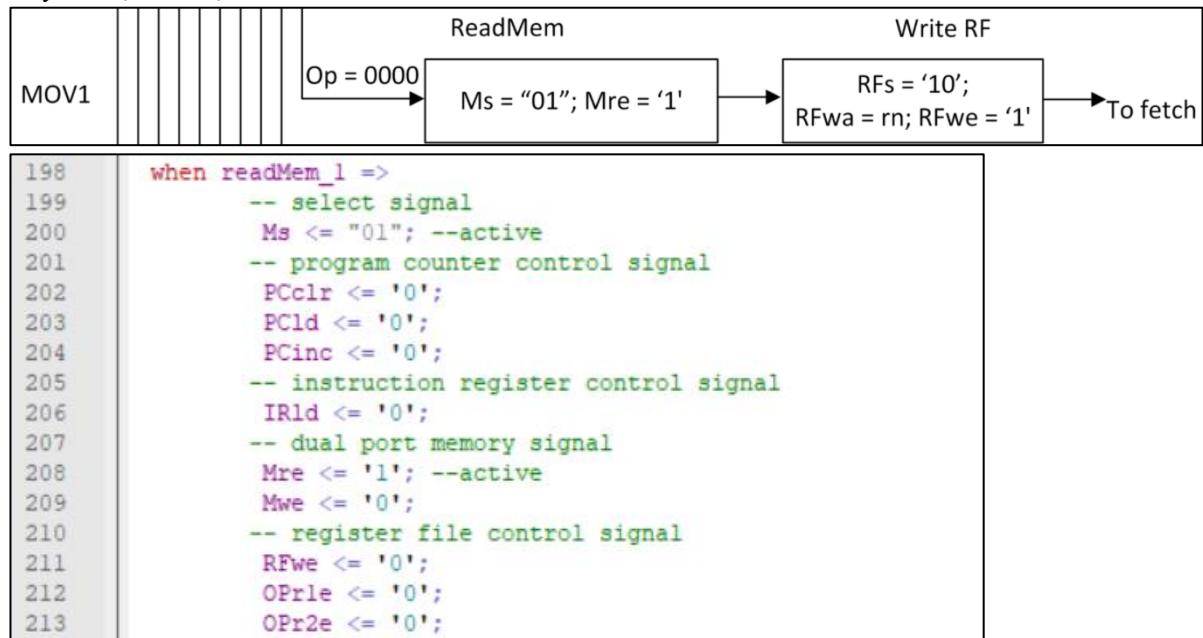
38  begin
39      opcode <= IRout(15 downto 12);

181  when decoder =>
182      -- program counter control signal
183      PCclr <= '0';
184      PCld <= '0';
185      PCinc <= '0';
186      -- instruction register control signal
187      IRld <= '0';
188      -- dual port memory signal
189      Mre <= '0';
190      Mwe <= '0';
191      -- register file control signal
192      RFwe <= '0';
193      OPrle <= '0';
194      OPr2e <= '0';
195      -----
196      opr1 <= IRout(11 downto 8); --active
197      opr2 <= IRout(7 downto 0); --active

```

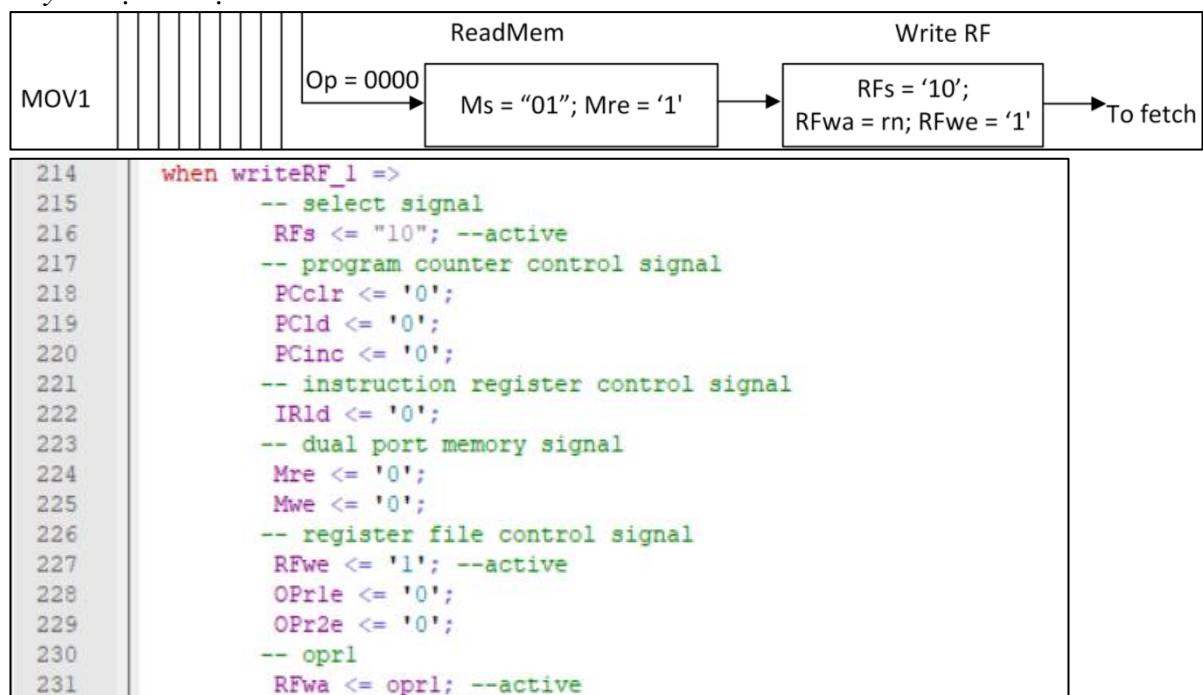
- **ReadMem** là trạng thái xử lý dữ liệu từ câu lệnh, sau đó điều khiển thanh bộ **Mux3to1** ở khối **Datapath** sẽ chọn lối vào phù hợp, đưa vào tệp các thanh ghi **RF**, chọn lối vào cho bộ tính toán **ALU**, và chọn phép toán phù hợp với câu lệnh bộ tín toán **ALU** thực hiện, kết quả của bộ tính toán **ALU** sẽ được phản hồi về lại khối **Datapath** (tùy vào loại câu lệnh)

*Lấy ví dụ câu lệnh MOVI*



- **WriteRF** là trạng thái thực hiện ghi dữ liệu ( ghi vào tệp thanh ghi hoặc ghi vào bộ nhớ hoặc không ghi tùy vào loại câu lệnh )

### Lấy ví dụ câu lệnh *MOV1*

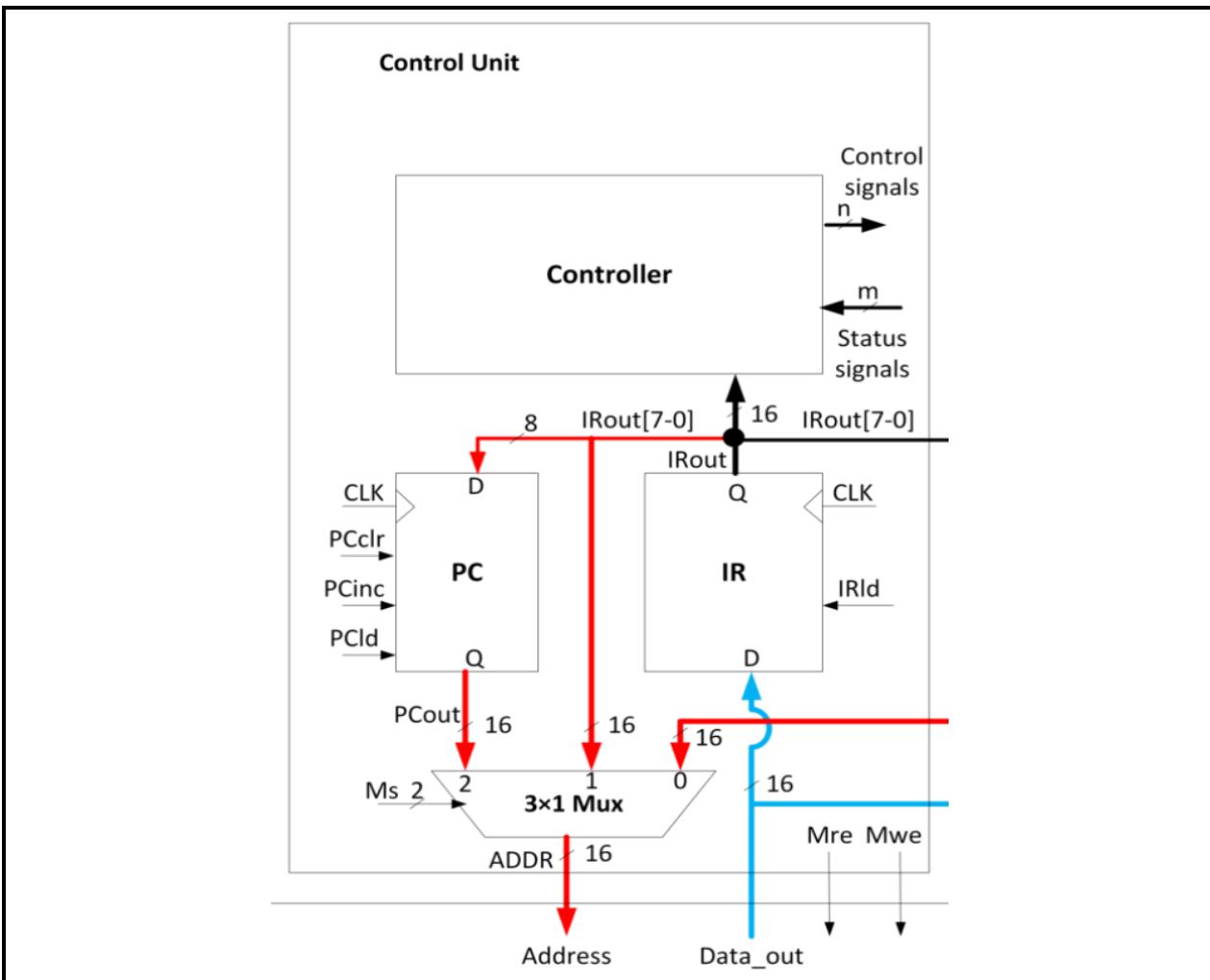


➔ Để đảm bảo bài báo cáo không bị dài và bởi phần code của các **10 câu lệnh này** là tương tự nhau thiết kế theo sơ đồ máy trạng thái ở **Hình 2** ở trên bạn em sẽ để trong file **controller.vhd** của thư mục **SourceCode**

- Sau khi thực hiện xong câu lệnh ( tức là xong trạng thái **WriteRF** ) thì khôi điểu khiển sẽ quay trở lại trạng thái **Fetch** để nạp câu lệnh tiếp theo, lúc này thì dựa vào câu lệnh trước, lối vào **ADDR** sẽ được chọn từ lối ra của thanh ghi hoặc địa chỉ ở câu lệnh hoặc là lối

102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119	<pre> when writeRF_1 =&gt;     current_state &lt;= fetch; when writeRF_2 =&gt;     current_state &lt;= fetch; when writeRF_3 =&gt;     current_state &lt;= fetch; when writeRF_add =&gt;     current_state &lt;= fetch; when writeRF_sub =&gt;     current_state &lt;= fetch; when writeRF_jpz =&gt;     current_state &lt;= fetch; when writeRF_or =&gt;     current_state &lt;= fetch; when writeRF_and =&gt;     current_state &lt;= fetch; when others =&gt; end case;</pre>
--	--

#### d) Khối điều khiển Control\_Unit



Ý tưởng thiết kế:

- Bây giờ mình đã thiết kế xong **4 component**: PC, IR, MUX, Controller. Ta sẽ port map các tín hiệu vào ra lại với nhau tương ứng với các **port component** trong file Sys\_Definition.

## Code thiết kế

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.std_logic_arith.all;
4 use work.sys_definition.all;
5 -----
6 entity control_unit is
7   Generic (
8     DATA_WIDTH : integer := 16; -- Data Width
9     ADDR_WIDTH : integer := 16 -- Address width
10    );
11  port (
12    clk : in STD_LOGIC;
13    Rst : in std_logic;
14    Data_out : in std_logic_vector (DATA_WIDTH -1 downto 0);
15    ALUz_net : in std_logic;
16    OPr2_net : in std_logic_vector (DATA_WIDTH -1 downto 0);
17    ADDR : out std_logic_vector (DATA_WIDTH -1 downto 0);
18    I Rout_net : buffer std_logic_vector (DATA_WIDTH -1 downto 0);
19    I Rout_16 : buffer std_logic_vector (DATA_WIDTH -1 downto 0);
20    -- X"00" + IR (7 downto 0)
21    RFs_net : out std_logic_vector (1 downto 0);
22    RFwa_net, OPr1a_net, OPr2a_net : out std_logic_vector (3 downto 0);
23    RFwe_net, OPr1e_net, OPr2e_net : out std_logic;
24    ALUs_net : out std_logic_vector (1 downto 0);
25    Mre_net, Mwe_net : out std_logic
26    );
27 end entity;
28 -----
29 architecture beh of control_unit is
30   signal Ms_net : std_logic_vector (1 downto 0);
31   --signal I Rout_wire : std_logic_vector (DATA_WIDTH -1 downto 0);
32   signal IR_out_8 : std_logic_vector (7 downto 0);
33   signal PCclr_net, PCinc_net, PCld_net, IRld_net : std_logic;
34   signal PCout_net : std_logic_vector (DATA_WIDTH -1 downto 0);
35 begin
36   -- immediate block
37   IR_out_8 <= I Rout_net (7 downto 0);
38   I Rout_16 (7 downto 0) <= IR_out_8;
39   I Rout_16 (DATA_WIDTH -1 downto 8) <= x"00";
40   -----
41   CONTROLLER_U : controller port map
42   (
43     clk => clk,
44     nReset => Rst,
45     I Rout => I Rout net.
46     ALUz => ALUz_net,
47     RFs => RFs_net,
48     ALUs => ALUs_net,
49     Ms => Ms_net,
50     PCclr => PCclr_net,
51     PCld => PCld_net,
52     PCinc => PCinc_net,
53     IRld => IRld_net,
54     Mre => Mre_net,
55     Mwe => Mwe_net,
56     RFwe => RFwe_net,
57     OPr1e => OPr1e_net,
58     OPr2e => OPr2e_net,
59     RFwa => RFwa_net,
60     OPr1a => OPr1a_net,
61     OPr2a => OPr2a_net
62   );
63   IR_U : ir port map
64   (
65     clk => clk,
66     IR_in => Data_out,
67     IRld => IRld_net,
68     IR_out => I Rout_net
69   );
70   PC_U : pc port map
71   (
72     clk => clk,
73     PC_in => I Rout_16,
74     PCclr => PCclr_net,
75     PCinc => PCinc_net,
76     PCld => PCld_net,
77     PC_out => PCout_net
78   );
79   MUX_U: mux3tol port map
80   (
81     SEL => Ms_net,
82     A => PCout_net,
83     B => I Rout_16,
84     C => OPr2_net,
85     Z => ADDR
86   );
87 end architecture;
```

## 5.4. Bộ nhớ Dual Port Memory

Address	Data_out	Data_in
<b>Program Memory (ROM): 18 từ 16-bit từ địa chỉ 0x000 đến 0x011</b>		
<b>Data Memory (RAM): 237 từ 16-bit từ 0x012 đến 0xOFF</b>		
<b>Cổng GPIO 16-bit ánh xạ bộ nhớ tại địa chỉ 0x100</b>		

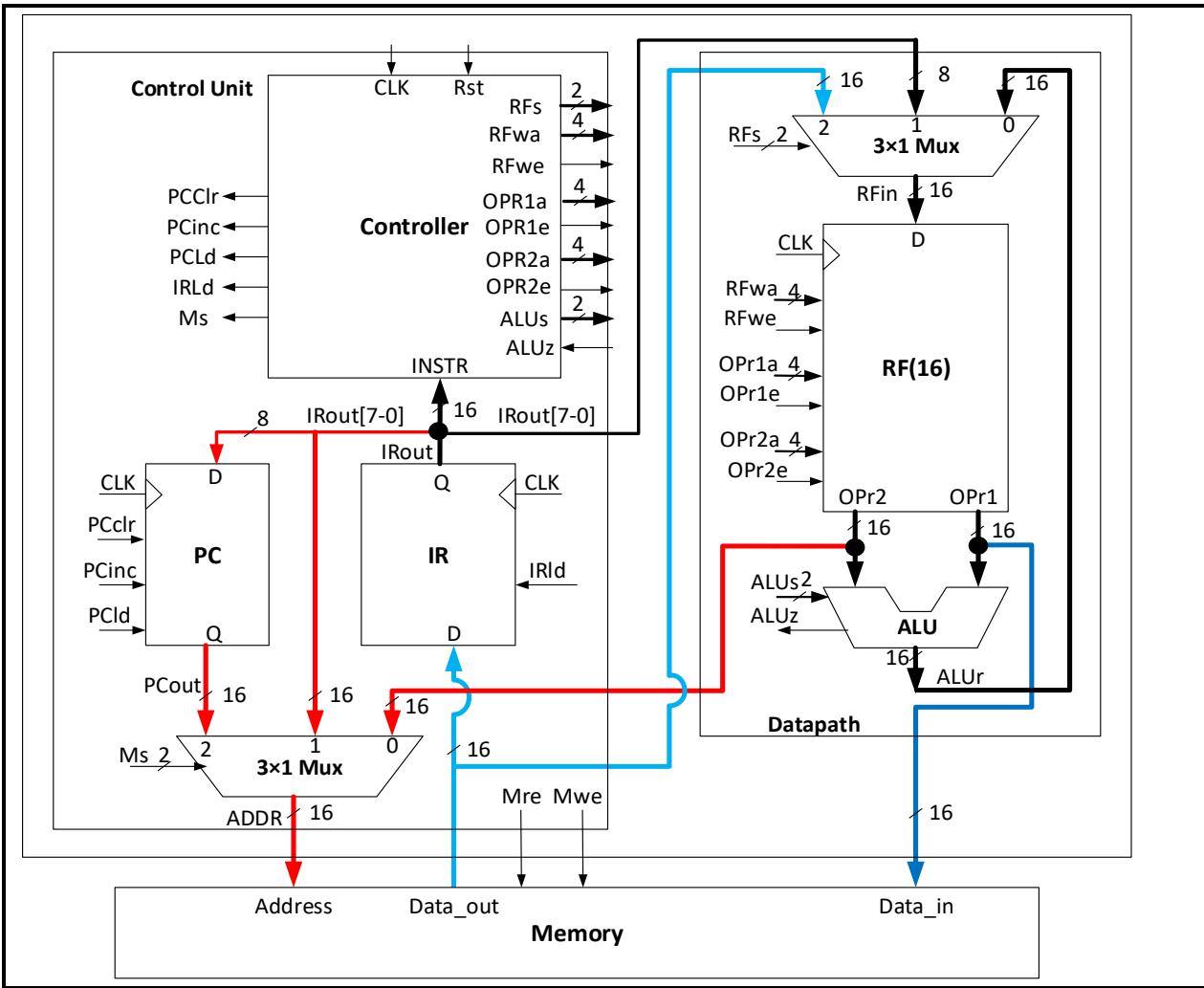
### Code thiết kế

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  -----
5  entity dpmem is
6      generic (
7          DATA_WIDTH : integer := 16; -- Word Width
8          ADDR_WIDTH : integer := 16 -- Address width
9      );
10     port (
11         clk : in std_logic;
12         nReset : in std_logic; -- Reset input
13         addr : in std_logic_vector(ADDR_WIDTH -1 downto 0); -- Address
14         -- Writing Port
15         Wen : in std_logic; -- Write Enable
16         Datain : in std_logic_vector(DATA_WIDTH -1 downto 0) := (others => '0'); -- Input Data
17         -- Reading Port
18         Ren : in std_logic; -- Read Enable
19         Dataout : out std_logic_vector(DATA_WIDTH -1 downto 0) -- Output data
20     );
21 end dpmem;
22 -----
23 architecture dpmem_arch of dpmem is
24     type DATA_ARRAY is array (integer range <>) of std_logic_vector(DATA_WIDTH -1 downto 0); -- Memory Type
25     signal M : DATA_ARRAY(0 to (2^ADDR_WIDTH) -1) := (others => (others => '0')); -- Memory model
26     constant PM_Size : Integer := 18; --18
27     constant PM : DATA_ARRAY(0 to PM_Size-1) :=
28     (
29         -- Machine code
30         X"0911", -- 0: Mov R9, 17 => R9 = M(17)
31         X"1980", -- 1: Mov 128, R9 => M(128) = R9
32         X"3210", -- 2: Mov R2, #16 => R2 = 16
33         X"2290", -- 3: Mov R2, @R9 => M(R9) = R2
34         X"4920", -- 4: ADD R9, R2 => R9 = R9 + R2
35         X"5920", -- 5: SUB R9, R2 => R9 = R9 - R2
36         X"6201", -- 6: JZ R2,1
37         -- for (int i = 10; i! = 0; i--) total += 1
38         X"3000", -- 7: Start : Mov R0, #R0 // total = 0
39         X"310A", -- 8:           Mov R1, #0 // i = 10
40         X"3201", -- 9:           Mov R2, #1 // constant 1
41         X"3300", -- 10:          Mov R3, #0 // constant 0
42         X"6110", -- 11: Loop : JZ R1. Next // Done if i = 0
43         X"4010", -- 12:           ADD R0, R1 // total += i
44         X"2090", -- 13:           Mov R0, @R9 // M(R9) = R0 = total
45         X"5120", -- 14:           SUB R1, R2 // i--
46         X"630B", -- 15:           JZ R3, Loop // total += i
47         X"6307", -- 16: Next JZ R3, Start// total += i
48         X"0100" -- 17: Addr-table // address of external _SEGMENT
49     );
50     -----
51 begin
52     -- Read/Write process
53     RW_Proc : process (clk, nReset)
54     begin
55         if nReset = '0' then
56             Dataout <= (others => '0');
57             M(0 to PM_Size-1) <= PM; -- initialize program memory
58         elsif (clk'event and clk = '1') then -- rising clock edge
59             if Wen = '1' then
60                 M(conv_integer(addr)) <= Datain; -- ensure that data cant overwrite on program --// +PM_Size
61             else
62                 if Ren = '1' then
63                     Dataout <= M(conv_integer(addr));
64                 else
65                     Dataout <= (others => 'Z');
66                 end if;
67             end if;
68         end if;
69     end process RW_Proc;
70 end dpmem_arch;

```

## 5.5. Bộ vi xử lý ( CPU )



Sau khi đã thiết kế xong các thành phần của bộ vi xử lý, bây giờ sẽ ghép các thành phần lại với nhau để có được 1 bộ vi xử lý hoàn chỉnh

2 lõi vào điều khiển đó là tín hiệu **Clock** và tín hiệu **Reset**

5 lõi ra

- **IR\_out** để xem câu lệnh được thực hiện
- **ALU\_out** để xem kết quả sau khi được tính toán
- **Mre** và **Mwe** để xem tín hiệu đọc và tín hiệu ghi dữ liệu từ bộ nhớ
- **Data\_in\_wire** và **Data\_out\_wire** để xem dữ liệu được đưa vào bộ nhớ và dữ liệu được lấy ra từ bộ nhớ
- **Addr\_out** để xem địa chỉ câu lệnh được chọn

## Code thiết kế

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use work.sys_definition.all;
5  -----
6  entity cpu is
7    Generic (
8      DATA_WIDTH : integer := 16; -- Data Width
9      ADDR_WIDTH : integer := 16 -- Address width
10     );
11   port (
12     nReset : in STD_LOGIC; -- low active reset signal
13     clk : in STD_LOGIC; -- Clock
14     Addr_out : buffer STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
15     ALU_out : out STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
16     IR_out : buffer STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
17     Data_in_wire, Data_out_wire : buffer STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
18     Mre, Mwe : buffer std_logic
19   );
20 end cpu;
21 -----
22 architecture struc of cpu is
23 begin
24   signal IRout_16_net : std_logic_vector (DATA_WIDTH -1 downto 0);
25   signal OPr2_out_wire : std_logic_vector (DATA_WIDTH -1 downto 0);
26   signal RFs_wire, ALUs_wire : std_logic_vector (1 downto 0);
27   signal RFwa_wire, OPr1a_wire, OPr2a_wire : std_logic_vector (3 downto 0);
28   signal RFwe_wire, OPr1e_wire, OPr2e_wire, ALUz_wire : std_logic;
29   -----
30   begin
31     --control unit-----
32     CTRL_U: control_unit port map
33     (
34       clk => clk,
35       Rst => nReset,
36       Data_out => Data_out_wire,
37       ALUz_net => ALUz_wire,
38       OPr2_net => OPr2_out_wire,
39       ADDR => Addr_out,
40       IRout_net => IR_out,
41       IRout_16 => IRout_16_net,
42       RFs_net => RFs_wire,
43       RFwa_net => RFwa_wire,
44       OPr1a_net => OPr1a_wire,
45       OPr2a_net => OPr2a_wire,
46
47       RFwe_net => RFwe_wire,
48       OPr1e_net => OPr1e_wire,
49       OPr2e_net => OPr2e_wire,
50       ALUz_net => ALUz_wire,
51       Mre_net => Mre,
52       Mwe_net => Mwe
53     );
54   -- datapath -----
55   DP_U : datapath port map
56   (
57     clk => clk,
58     IR_out => IRout_16_net,
59     Data_out => Data_out_wire,
60     RFs_net => RFs_wire,
61     RFwa_net => RFwa_wire,
62     OPr1a_net => OPr1a_wire,
63     OPr2a_net => OPr2a_wire,
64     RFwe_net => RFwe_wire,
65     OPr1e_net => OPr1e_wire,
66     OPr2e_net => OPr2e_wire,
67     OPr2_out => OPr2_out_wire,
68     Data_in => Data_in_wire,
69     ALUs_net => ALUs_wire,
70     ALUz_net => ALUz_wire,
71     ALU_out => ALU_out
72   );
73   -- dual port memory --
74   MEM_U: dpmem port map
75   (
76     clk => clk,
77     nReset => nReset,
78     addr => Addr_out,
79     Wen => Mwe,
80     Datin => Data_in_wire,
81     Ren => Mre,
82     Dataout => Data_out_wire
83   );
84 end struc;
```

## 6. Simulation and Synthesis

### File testbench để kiểm tra hoạt động của bộ vi xử lý

Có hai tín hiệu điều khiển vào được viết trong 2 process để kiểm tra từng phần

Tín hiệu vào **Reset** được kiểm tra ở **60 ns** giây đầu

Tín hiệu vào **Clock** với chu kỳ **20 ns**, được kiểm tra liên tục, sử dụng vòng lặp để có thể kiểm tra đến giá trị mình muốn

### Code thiết kế

```
1  library ieee;
2  use ieee.std_logic_arith.all;
3  use ieee.std_logic_1164.all;
4  use ieee.STD_LOGIC_UNSIGNED.all;
5  use work.sys_definition.all;
6  use std.textio.all;
7  -----
8  entity cpu_tb is
9  end cpu_tb;
10 
11 architecture behavior of cpu_tb is
12   constant CLKTIME : time := 20 ns;
13 
14   signal nReset : STD_LOGIC; -- INPUT low active reset signal
15   signal clk : STD_LOGIC; -- INPUT Clock
16   signal Addr_out : STD_LOGIC_VECTOR (ADDR_WIDTH -1 downto 0);
17   signal ALU_out : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
18   signal IR_out : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
19   signal Data_in_wire, Data_out_wire : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
20   signal Mre, Mwe : std_logic;
21 
22 begin
23 
24  -- clock generation
25  --  clk <= not clk after CLKTIME/2;
26  -- UUT component
27  dut : cpu
28  generic map
29  (
30    DATA_WIDTH => 16, -- Data Width
31    ADDR_WIDTH => 16 -- Address width
32  )
33  port map
34  (
35    nReset => nReset,
36    clk => clk,
37    Addr_out => Addr_out,
38    ALU_out => ALU_out,
39    IR_out => IR_out,
40    Mre => Mre,
41    Mwe => Mwe,
42    Data_in_wire => Data_in_wire,
43    Data_out_wire => Data_out_wire
44  );
45 
46  -- Read process
47  clk_proc : process
48  begin
49    clk <= '0';
50    wait for CLKTIME/2;
51    clk <= '1';
52    wait for CLKTIME/2;
53  end process;
54  --
55  rst_proc : process
56  begin
57    nReset <= '0';
58    wait for CLKTIME * 3;
59    nReset <= '1';
60    wait;
61  end process;
62 end behavior;
```

### Các lệnh để kiểm tra hoạt động của bộ vi xử lý

#### Test 7 câu lệnh:

- X”0911”, — 0: Mov R9,0x11 => R9 = M(0x11)
- X”1980”, — 1: Mov 0x80,R9 => M(0x80) = R9
- X”3210”, — 2: MOV R2,#0x10 => R2 = 0x10
- X”2290”, — 3: Mov R2,@R9 => M(R9) = R2
- X”4920”, — 4: ADD R9,R2 => R9 = R9 + R2
- X”5920”, — 5: SUB R9,R2 => R9 = R9 – R2
- X”6201”, — 6: JZ R2,1

## Chạy thử một chương trình tính tổng từ 1 đến 10

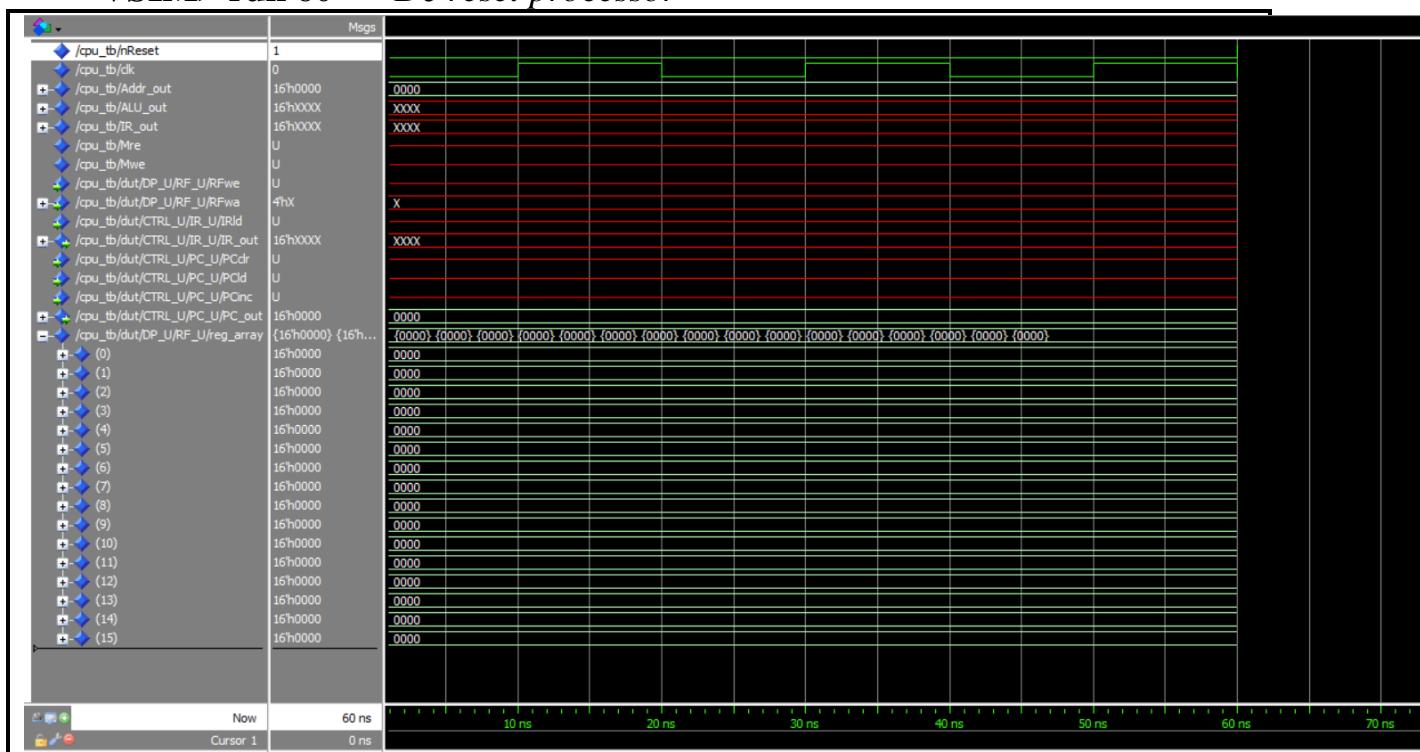
```
— for (iny i=10; i!=0;i-) total += 1;  
X”3000”, — 7: Start: Mov R0,#0 //total = 0  
X”310A”, — 8: Mov R1,#10 //i=10  
X”3201”, — 9: Mov R2,#1 //constant 1  
X”3300”, — 10: Mov R3,#0 //constant 0  
X”6110”, — 11: Loop: JZ R1,Next //Done if i=0  
X”4010”, — 12: ADD R0,R1 //total +=i  
X”2090”, — 13: MOV R0,@R9 // M(R9) = R0 = total: display the total on  
GPIO  
X”5120”, — 14: SUB R1,R2 //i–  
X”630B”, — 15: JZ R3,Loop //total +=i  
X”6307”, — 16: Next JZ R3,Start //total +=i  
X”0100” — 17: 0x100 // GPIO address
```

## Quá trình thực hiện

### a) Test tập lệnh

Tại cửa sổ Transcript của ModelSIM lần lượt gõ các lệnh run như sau:

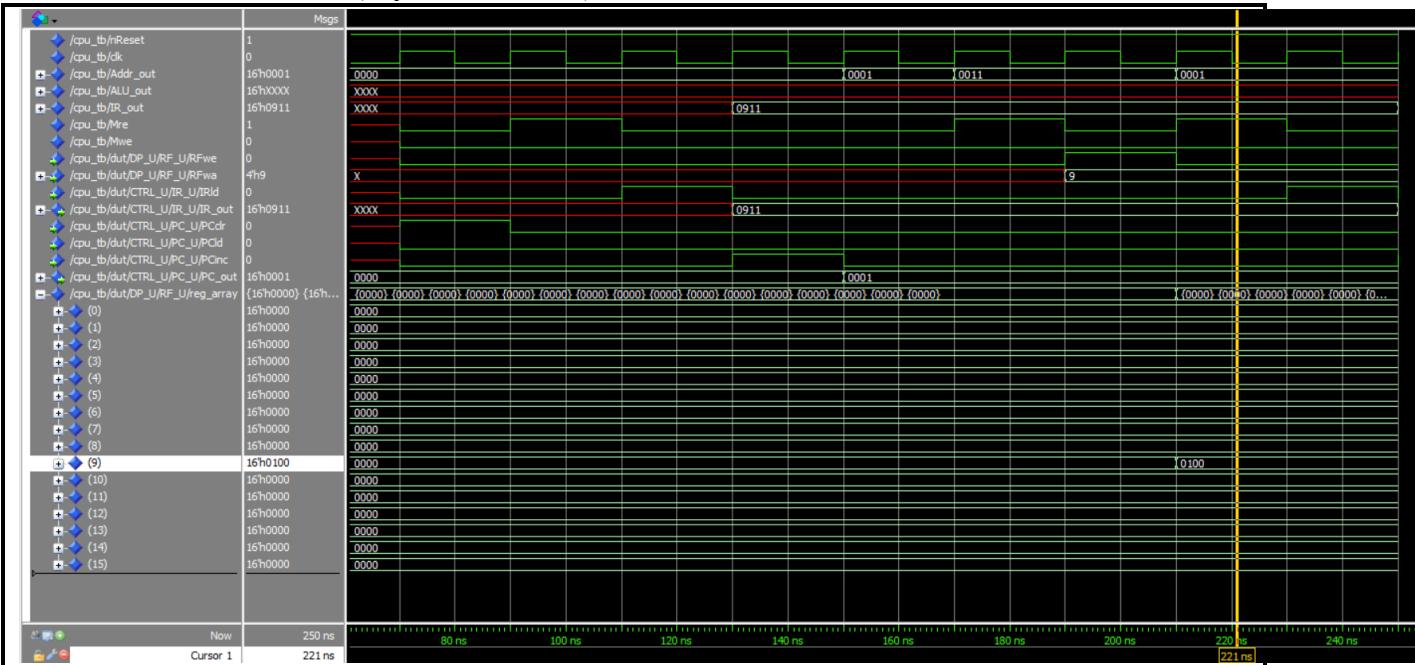
- VSIM> run 60 -- Để reset processor



Tín hiệu reset = 0 thì tất cả các tín hiệu khác đều bằng 0

Giá trị thanh ghi khi đó được reset lại về giá trị x”0000”

- VSIM> run 190 (*Lệnh 0- MovI*)

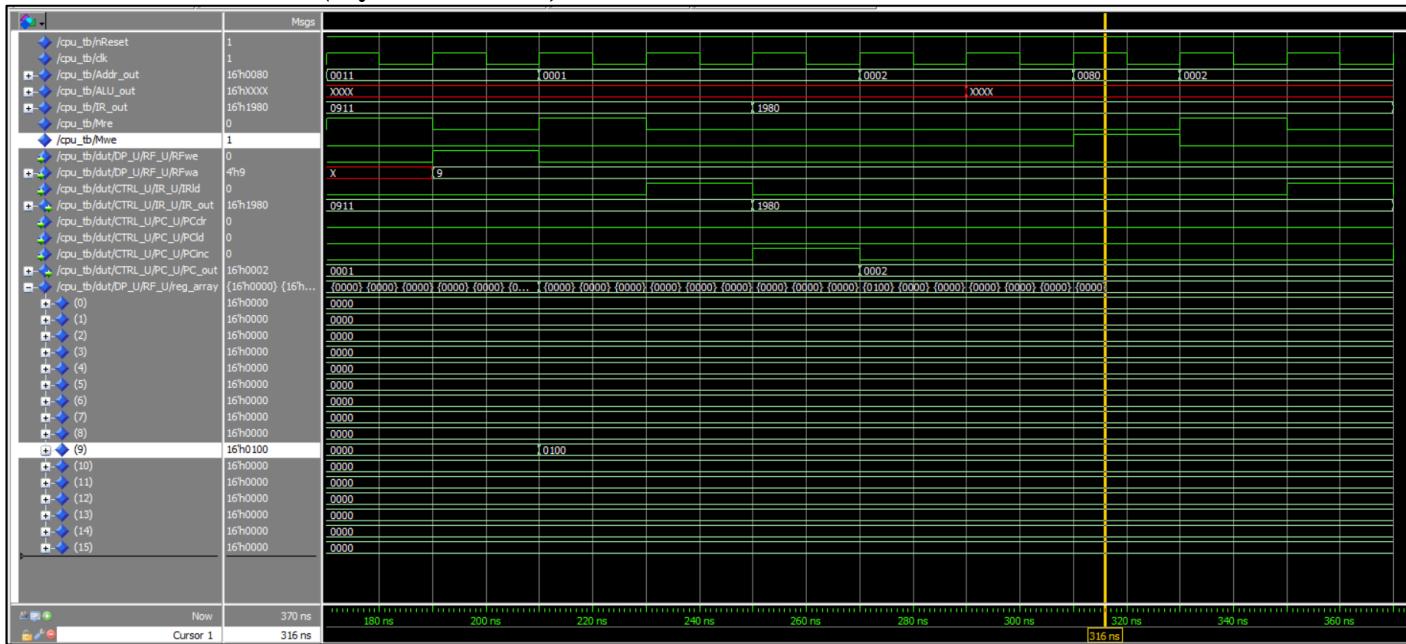


Để chạy câu lệnh đầu tiên:

X"0911", — 0: Mov R9,0x11 => R9 = M(0x11)

Sau khi chạy xong thì ta đã thấy R9 đã nạp giá trị trong MEM có địa chỉ là 0x11

- VSIM> run 120 (*Lệnh 1- Mov2*)



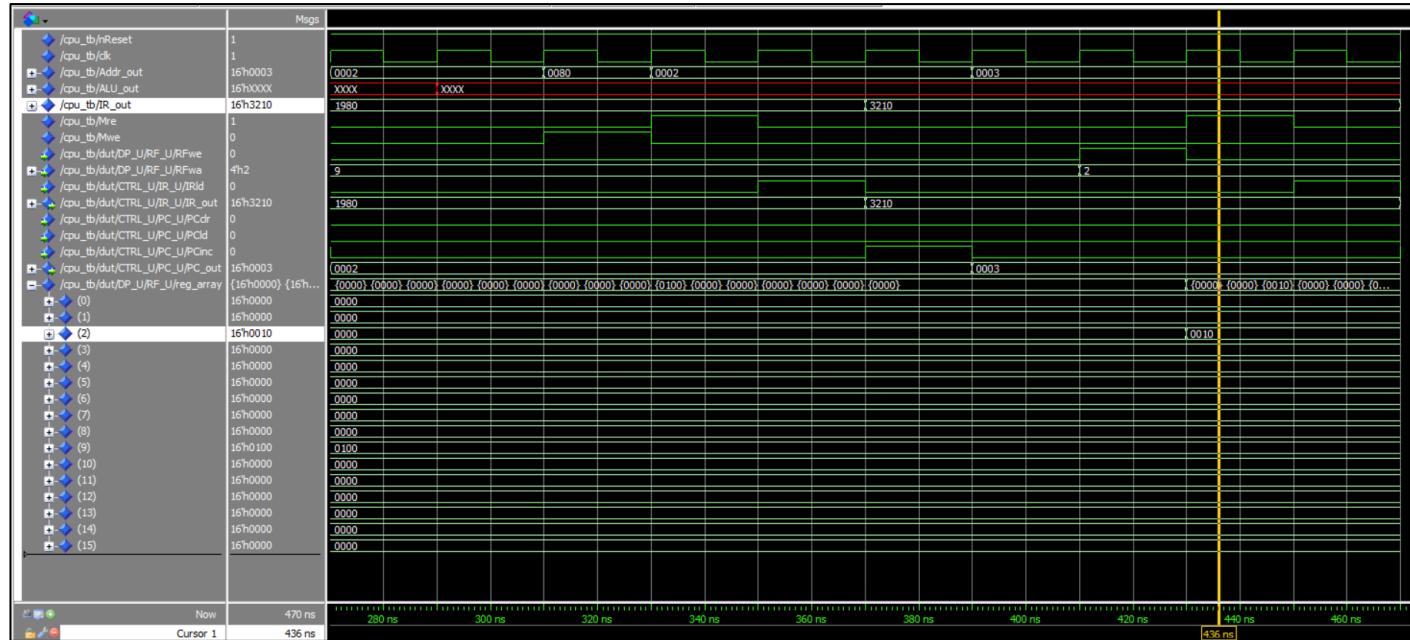
Để kiểm tra hoạt động của lệnh thứ 2:

X"1980", — 1: Mov 0x80,R9 => M(0x80) = R9

- Ta có thể thấy rằng trong quá trình thực hiện lệnh này thì có 1 giai đoạn Mwe = 1 mục đích để kết quả của R9 vào **MEMORY**
- Kiểm tra trong **MEMORY** ta đã thấy tại địa chỉ 0x80 đã được ghi giá trị là 0x100

Memory Data - /cpu_tb/dut/MEM_U/M - Default																	
00000000	0911	1980	3210	2290	4920	5920	6201	3000	310A	3201	3300	6110	4010	2090	5120	630B	
00000010	6307	0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00000080	0100	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

### • VSIM> run 100 (Lệnh 2- Mov4)



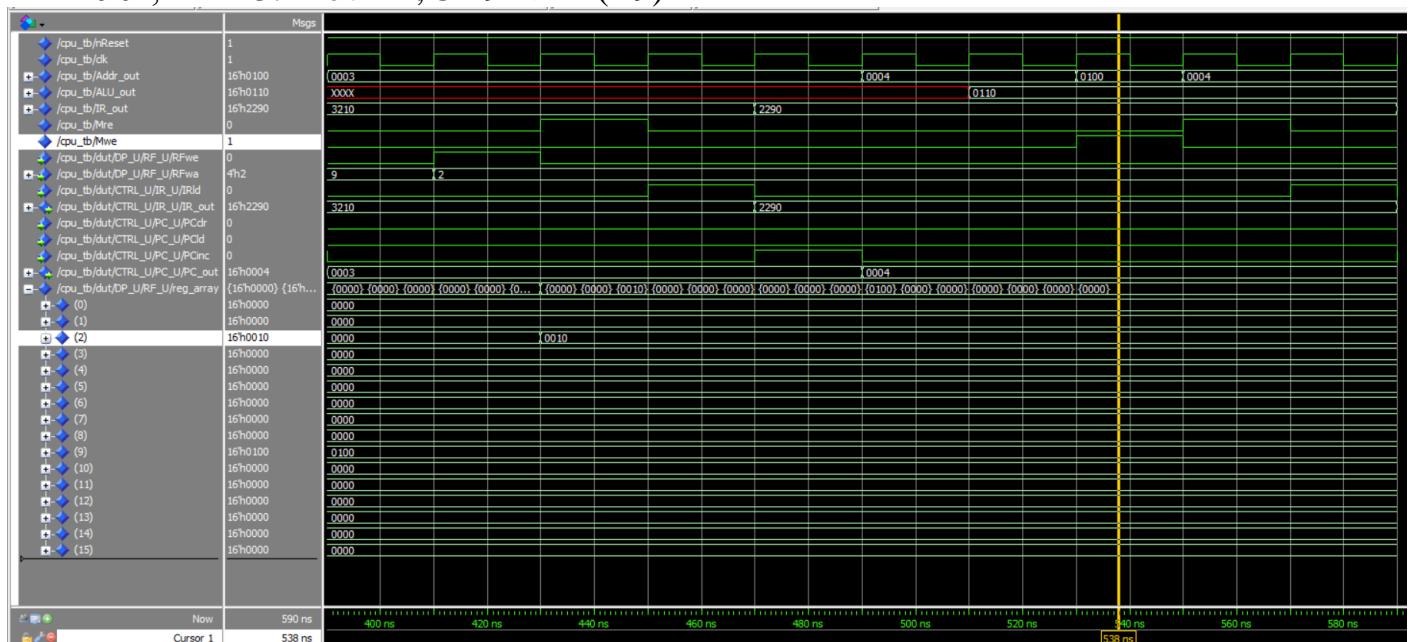
Để kiểm tra lệnh thứ 3:

X”3210”, — 2: MOV R2,#0x10 => R2 = 0x10

- Kết quả chạy ta có thể thấy rãnh thanh ghi R2 đã được gán giá trị là 0x10

### • VSIM> run 120 (Lệnh 3- Mov3)

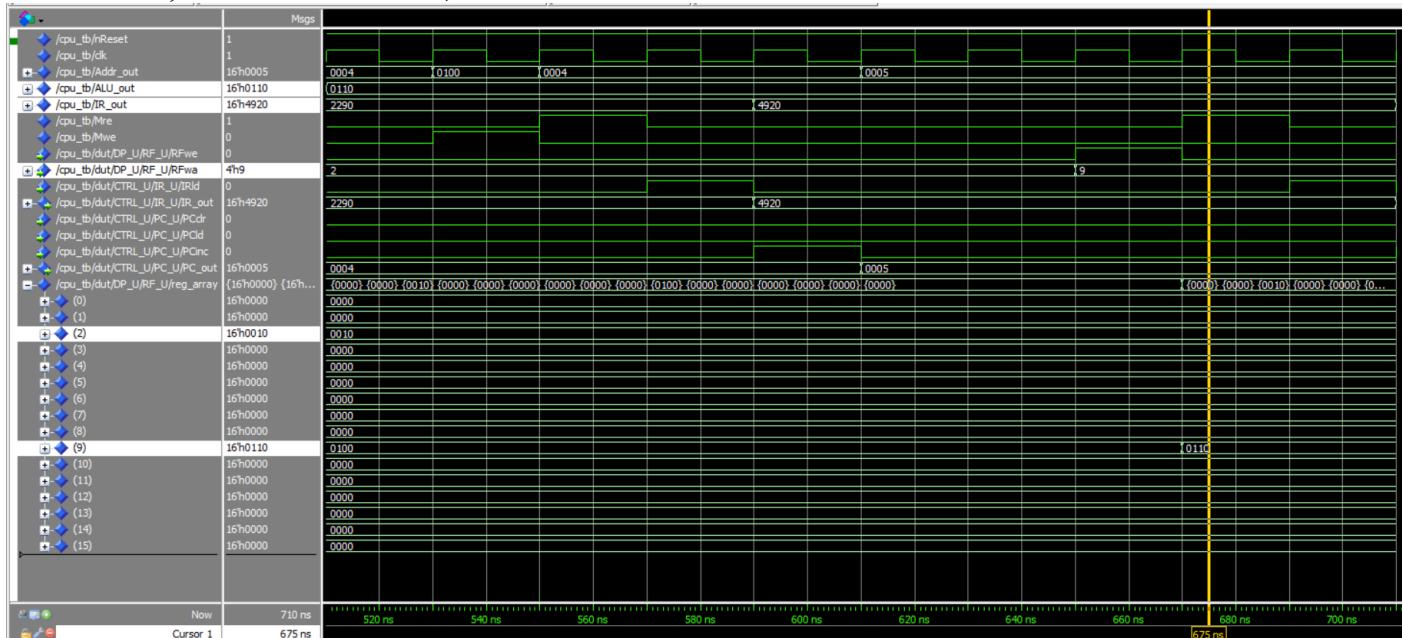
X”2290”, — 3: Mov R2,@R9 => M(R9) = R2



- Đây là câu lệnh ghi giá trị của thanh ghi R2 (0x10) ra bộ nhớ có địa chỉ là R9 (0x100)
- Check MEMlist vào đã thấy kết quả 0x10 được lưu ra ô nhớ 0x100

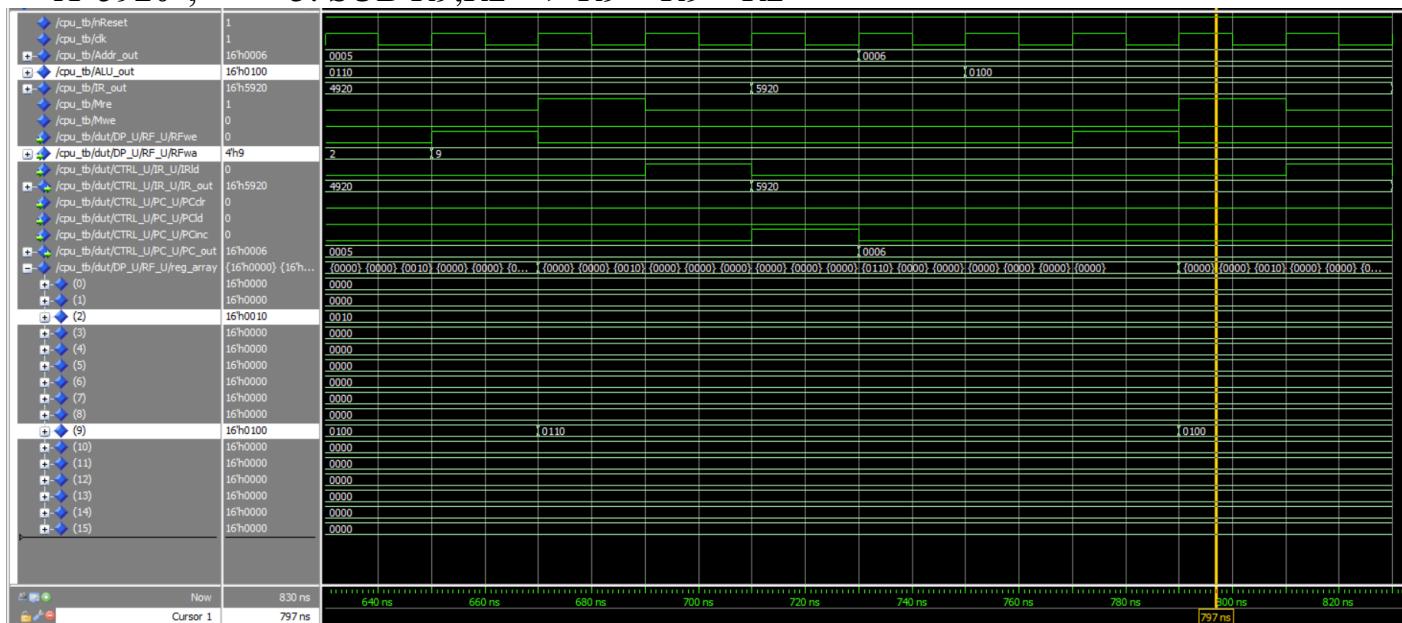
000000e0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

- VSIM> run 120 (**Lệnh 4 - Add**)  
X”4920”, — 4: ADD R9,R2 => R9 = R9 + R2



- Kết quả của phép tính cộng này là 0x110 nhìn vào giá trị của thanh ghi R9 cho thấy câu lệnh đã thực hiện đúng.

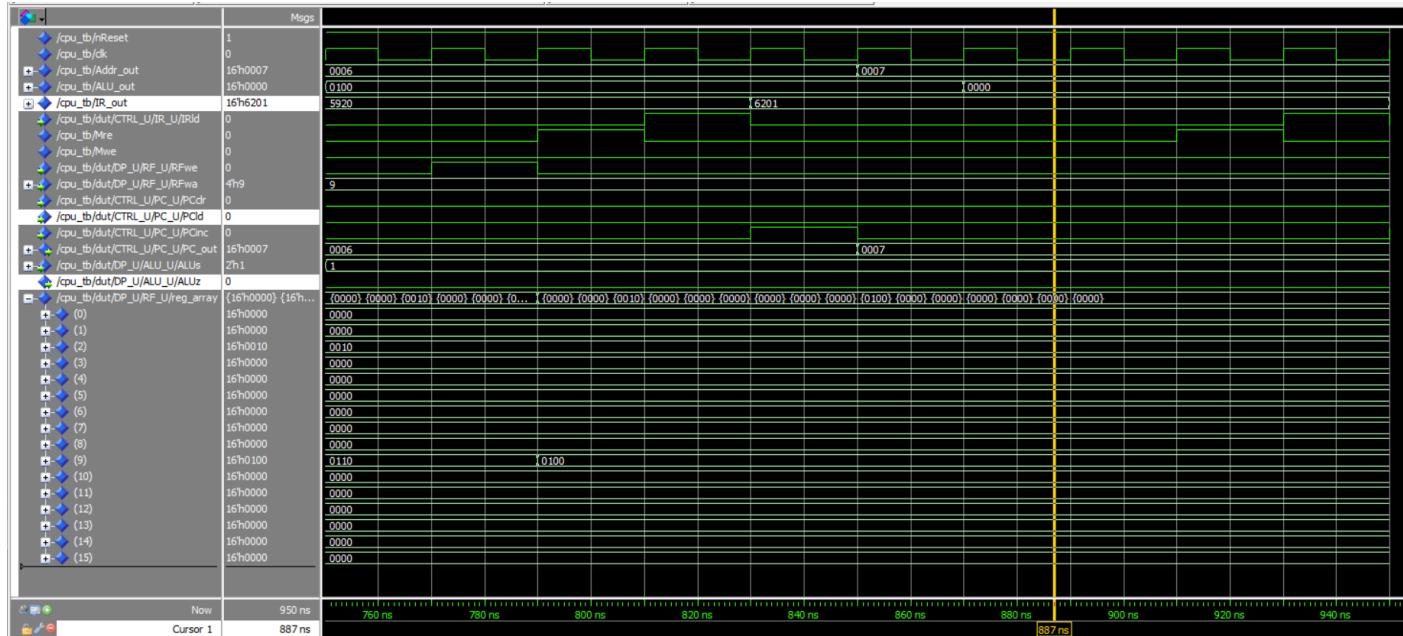
- VSIM> run 120 (**Lệnh 5 - Sub**)  
X”5920”, — 5: SUB R9,R2 => R9 = R9 – R2



- Kết quả của phép tính là **0x100** và bộ vi xử lý đã đưa ra đúng kết quả ra thanh ghi **R9**

- VSIM> run 120 (*Lệnh 6 JZ*)

X"6201", — 6: JZ R2,1



- Nhận thấy  $R2=0x10$  khác 0 nên lệnh này nó sẽ không nhảy tới địa chỉ 1
  - Theo mô phỏng sóng ta thấy được  $ALUz='0'$  nên  $PCld = '0'$  và sẽ không nạp địa chỉ 1 vào

⇒ Sau khi chạy hết 7 câu lệnh thì đã có thể kết luận là bộ vi xử lý đã hoạt động, dữ liệu được đọc ra, sau đó được ghi vào thanh ghi R9, sau đó thanh ghi R2 được gán giá trị 0x10.

b) Test chương trình tính tổng từ 1 đến 10

Bây giờ ta sẽ dùng bộ vi xử lý này chạy một chương trình tính tổng các số từ 1 đến 10 đã được lưu ở address 7 đến 16. Tiếp tục gõ vào transcript dòng lệnh sau:

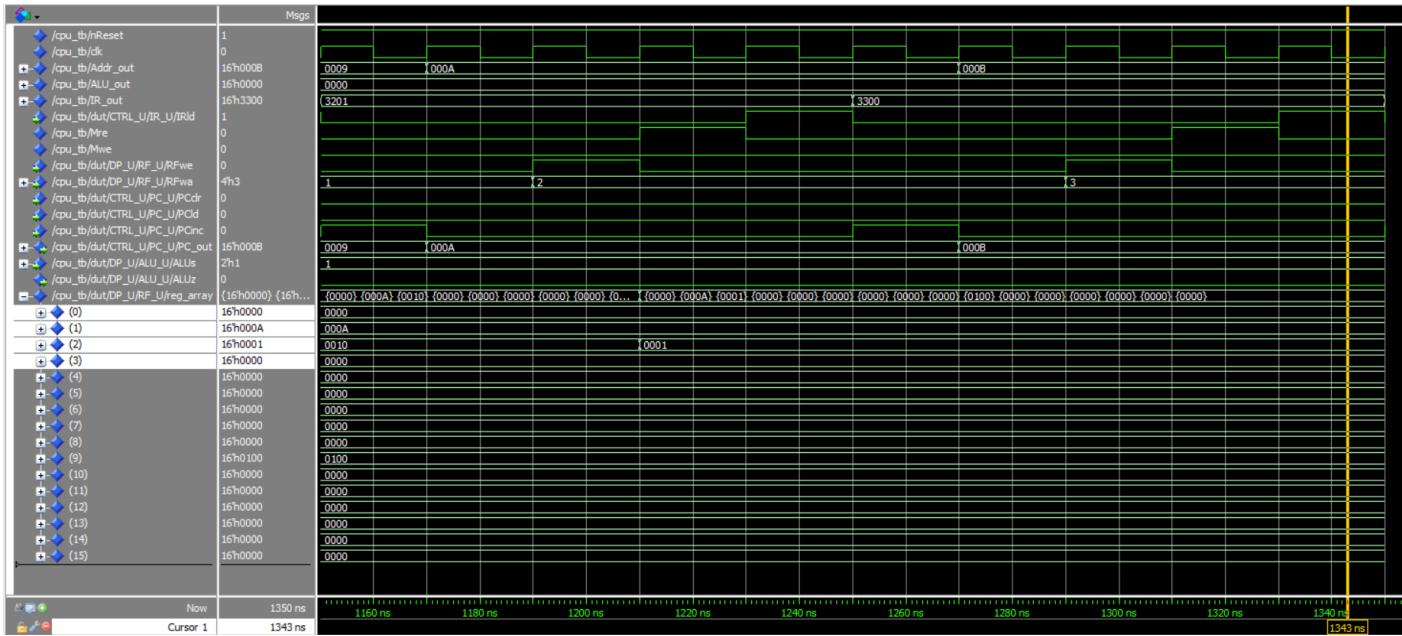
- VSIM> run 400 (Lệnh 7-10 để lưu các hằng số)

X"3000", — 7: Start: Mov R0,0x0 //total = 0

X”310A”, — 8: Mov R1,0xA //i=10

X"3201", -9: Mov R2,0x1 //constant 1

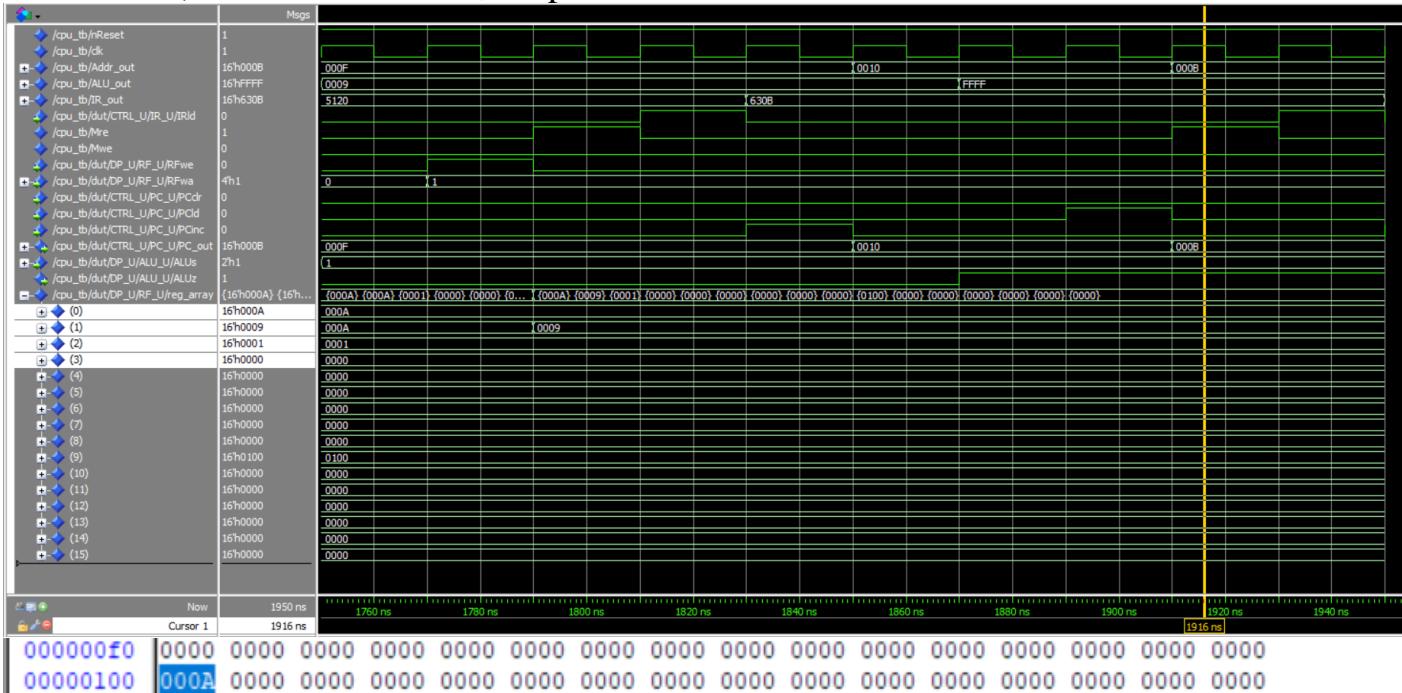
X"3300", -10: Mov R3,0x0 //constant 0



- Kết quả của 4 thanh ghi đã được như mong đợi

### • VSIM> run 600 (Lệnh 11-15 Vòng lặp lần 1)

X”6110”, — 11: Loop: JZ R1,Next //Done if i=0  
 X”4010”, — 12: ADD R0,R1 //total +=i  
 X”2090”, — 13: MOV R0,@R9 // M(R9) = R0  
 X”5120”, — 14: SUB R1,R2 //i–  
 X”630B”, — 15: JZ R3,Loop //total +=i



- Sau lần lặp đầu tiên thì trong thanh ghi R0= 0xA và M(0x100) = 0xA

- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 2*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0013 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 3*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	001B 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 4*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0022 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 5*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0028 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 6*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	002D 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 7*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0031 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 8*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0034 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 9*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0036 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- VSIM> run 600 (*Lệnh 11-15 Vòng lặp lần 10*)
 

000000f0	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000100	0037 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Sau khi chạy xong các lệnh trên, kết quả cuối cùng là X”0037” được lưu ở thanh ghi R0 , vòng lặp kết thúc

Kiểm tra Memory list, sau khi hoàn thành lệnh lặp thì giá trị lưu ở ô nhớ có địa chỉ **0x100** là **0x37**, đúng như tính toán

00000000	0911 1980 3210 2290 4920 5920 6201 3000
00000008	310A 3201 3300 6110 4010 2090 5120 630B
00000010	6307 0100 0000 0000 0000 0000 0000 0000
00000018	0000 0000 0000 0000 0000 0000 0000 0000
00000020	0000 0000 0000 0000 0000 0000 0000 0000
00000028	0000 0000 0000 0000 0000 0000 0000 0000
00000030	0000 0000 0000 0000 0000 0000 0000 0000
00000038	0000 0000 0000 0000 0000 0000 0000 0000
00000040	0000 0000 0000 0000 0000 0000 0000 0000
00000048	0000 0000 0000 0000 0000 0000 0000 0000
00000050	0000 0000 0000 0000 0000 0000 0000 0000
00000058	0000 0000 0000 0000 0000 0000 0000 0000
00000060	0000 0000 0000 0000 0000 0000 0000 0000
00000068	0000 0000 0000 0000 0000 0000 0000 0000
00000070	0000 0000 0000 0000 0000 0000 0000 0000
00000078	0000 0000 0000 0000 0000 0000 0000 0000
00000080	0100 0000 0000 0000 0000 0000 0000 0000
00000088	0000 0000 0000 0000 0000 0000 0000 0000
00000090	0000 0000 0000 0000 0000 0000 0000 0000
00000098	0000 0000 0000 0000 0000 0000 0000 0000
000000a0	0000 0000 0000 0000 0000 0000 0000 0000
000000a8	0000 0000 0000 0000 0000 0000 0000 0000
000000b0	0000 0000 0000 0000 0000 0000 0000 0000
000000b8	0000 0000 0000 0000 0000 0000 0000 0000
000000c0	0000 0000 0000 0000 0000 0000 0000 0000
000000c8	0000 0000 0000 0000 0000 0000 0000 0000
000000d0	0000 0000 0000 0000 0000 0000 0000 0000
000000d8	0000 0000 0000 0000 0000 0000 0000 0000
000000e0	0000 0000 0000 0000 0000 0000 0000 0000
000000e8	0000 0000 0000 0000 0000 0000 0000 0000
000000f0	0000 0000 0000 0000 0000 0000 0000 0000
000000f8	0000 0000 0000 0000 0000 0000 0000 0000
00000100	0037 0000 0000 0000 0000 0000 0000 0000
00000108	0000 0000 0000 0000 0000 0000 0000 0000