

# MICROPROCESSOR DESIGN PROJECT

CE6302.301

FALL 2021

SUCHITH NATRAJ JAVALI

SXJ200024

## Table of contents

- ❖ Objective
- ❖ Introduction
- ❖ Overview of all the components
- ❖ Assembly level Program run
- ❖ Simulation output of each module
- ❖ Final result output

## Objective

---

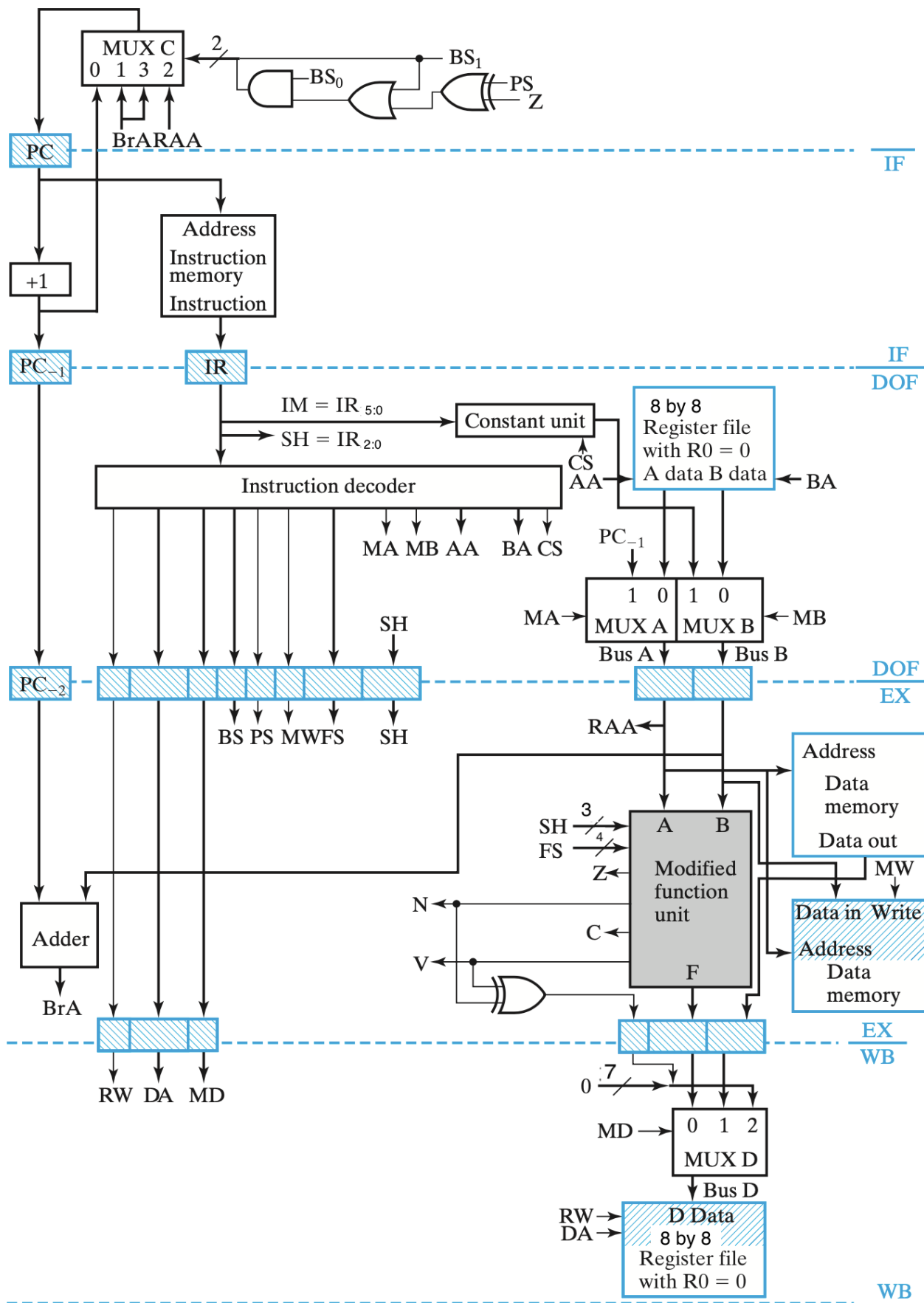
Primary objective of this project is to design 8-bit RISC MCU which includes Program memory, data memory, instruction decoder, Program counter, Arithmetic Logic Unit (ALU), MUX, register file, adders etc. using Verilog through Xilinx Vivado

## Introduction

---

Purpose of the CPU is to decode instructions received from memory and perform transfer, arithmetic, logic, and control operations with data stored in internal registers, memory, or I/O interface units.

The basic design of the MCU is shown in Figure below



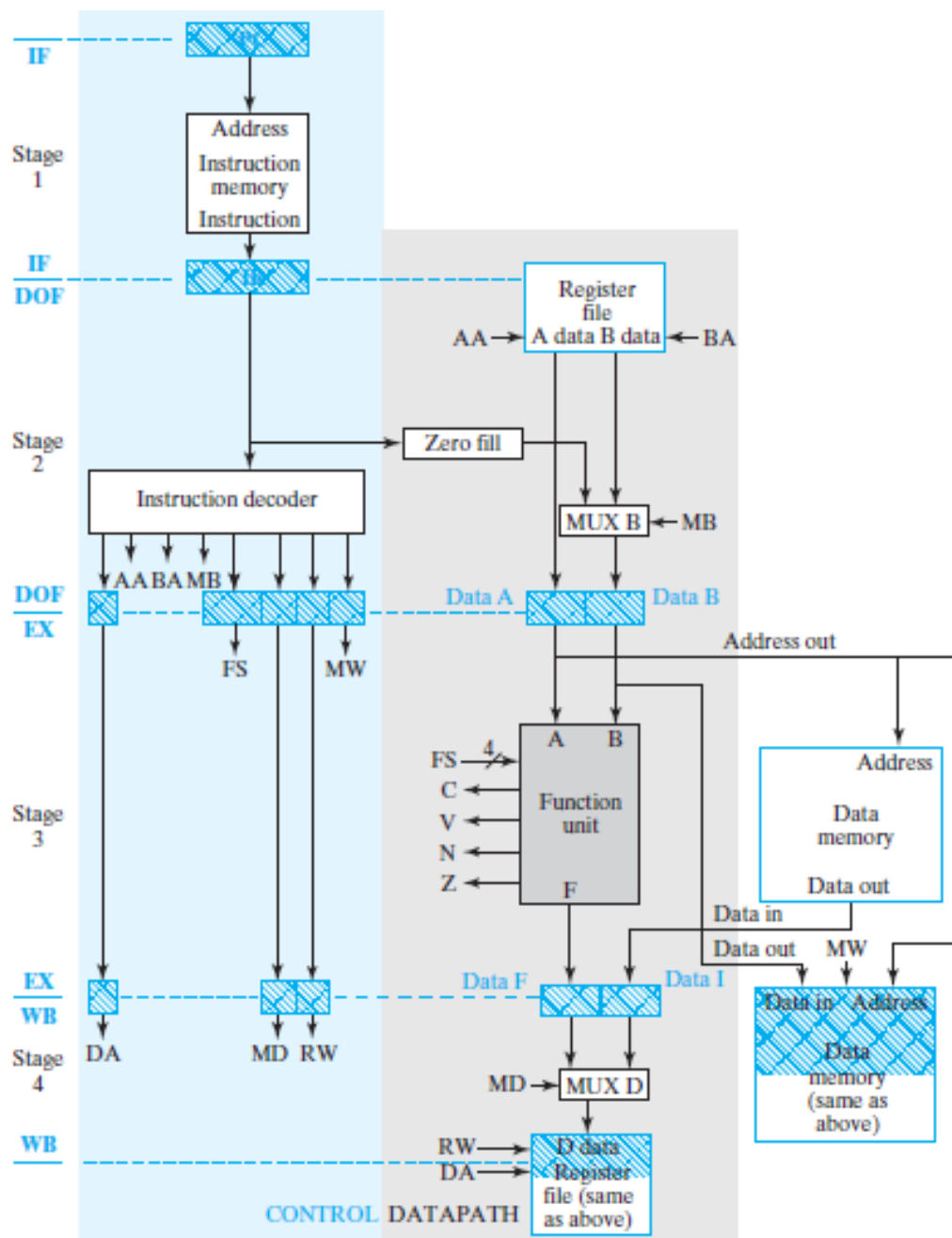
Below fig shows the pipeline stages in the Microprocessor design. We have fetch, decode, execute and write back stages (4 stages). Data flows in the pipeline from one stage to other accomplished by registers.

**Fetch stage**- Fetch the next instruction to be processed

**Decode stage** – Decode the instruction coming from fetch cycle and release control signals

**Execute stage** – Here, all the computations, memory read and write happens

**Write back stage**- In this stage, data store to registers happens after computation

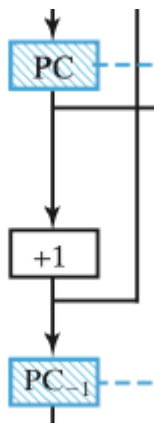


## Overview of the Components used

---

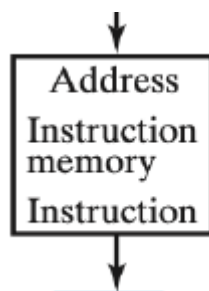
### 1. Program Counter (PC)

It controls the instruction memory. Value of PC is the address of the next instruction to be executed. Unless there is a Jump/Branch instruction, PC always increments by 1.



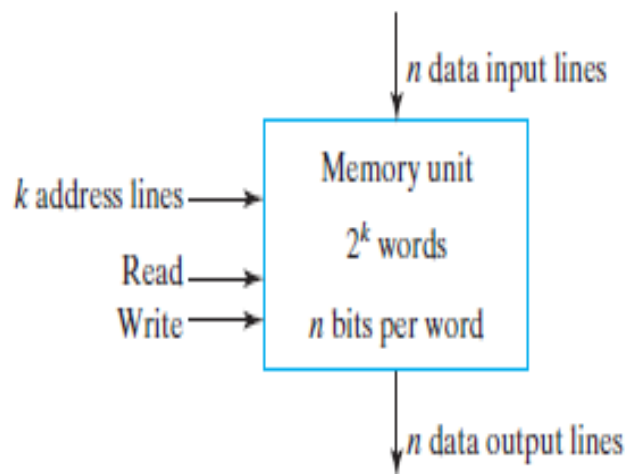
### 2. Instruction memory (ROM)

It stores the program that MCU will run. It takes the input as address and provides the corresponding instruction that needs to be acted upon by MCU. This gives out the 17-bit instruction to the instruction decoder according to the Program counter (PC)



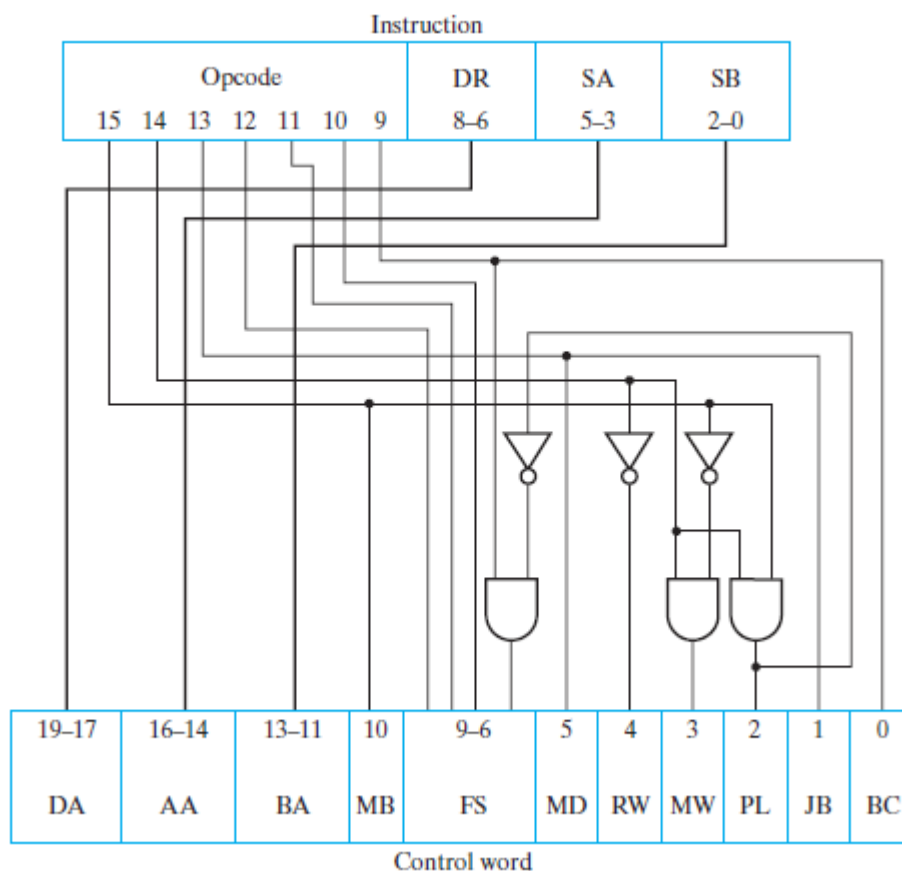
### 3. Data Memory (RAM)

It is used to load and store data in the MCU. It takes the address as input and stores the data through data line when write enable is active. And when loading the data, it takes the address as input and sends out the data from the respective address



## 4. Instruction decoder

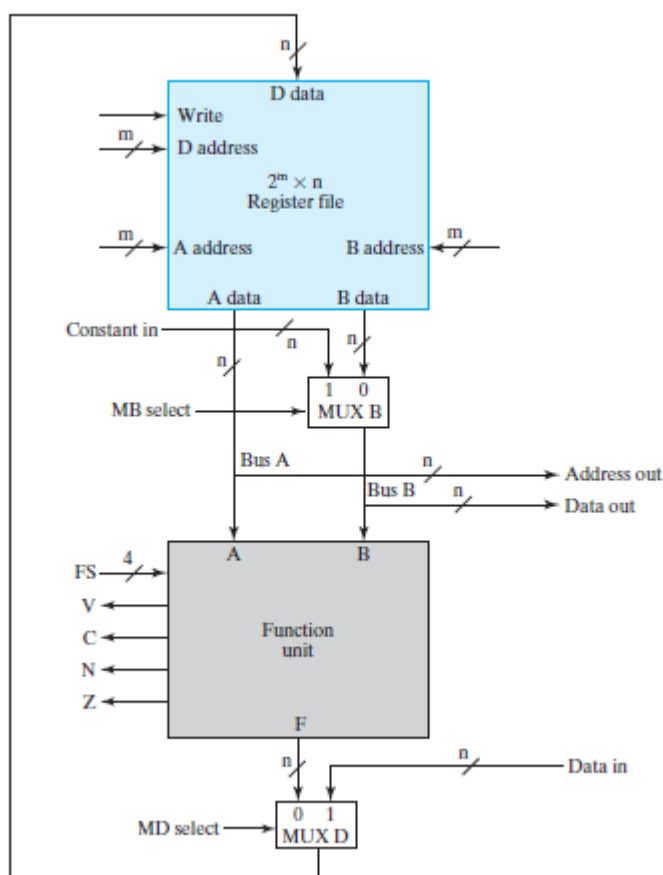
It is a combinational circuit which gives all the control words for the data path. The instruction decoder will send the proper control signals to the register file and any other control logic in order to prepare the data needed in the next stage. In this control-word fields DA, AA, and BA are equal to the instruction fields DR, RA (SA in fig), RB (SB in fig). The remaining control-word fields include data path and data memory control bits MB, MD, RW, and MW. FS is used as function select to work on ALU to select the operation to worked on operands





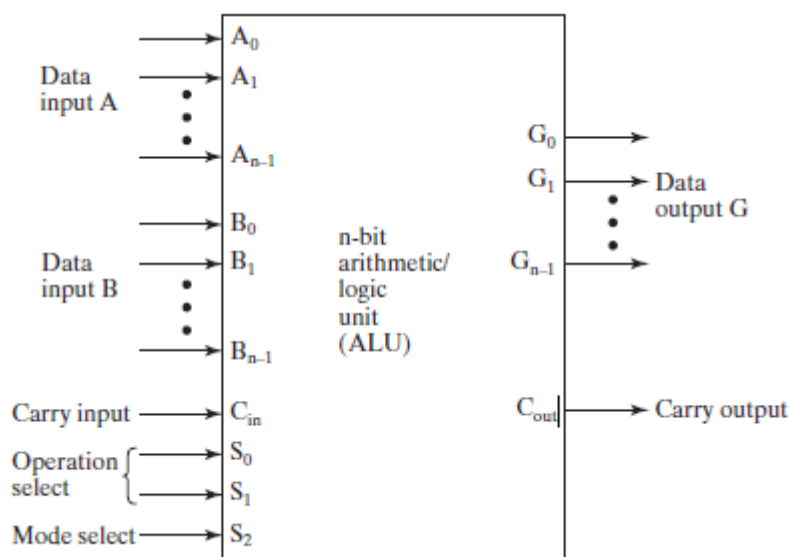
## 5. Register file

In our design, there are 8 registers (R0 – R7). Each of these are used by MCU to perform operations, load/store. R0 is always 000 and can not be changed. All the operations use at least 1 up to 3 registers for their function. A, B, D address is used to access them. Registers are controlled by instruction decoder and eventually data is sent to ALU.



## 6.ALU

Arithmetic and logic unit is where all the arithmetic and logical operations are done like ADD, AND, XOR etc. on the 8 bit input A/B. After the computation, it also sends out 4 flags (Zero, carry, negative, overflow) which gives the information about result computed.

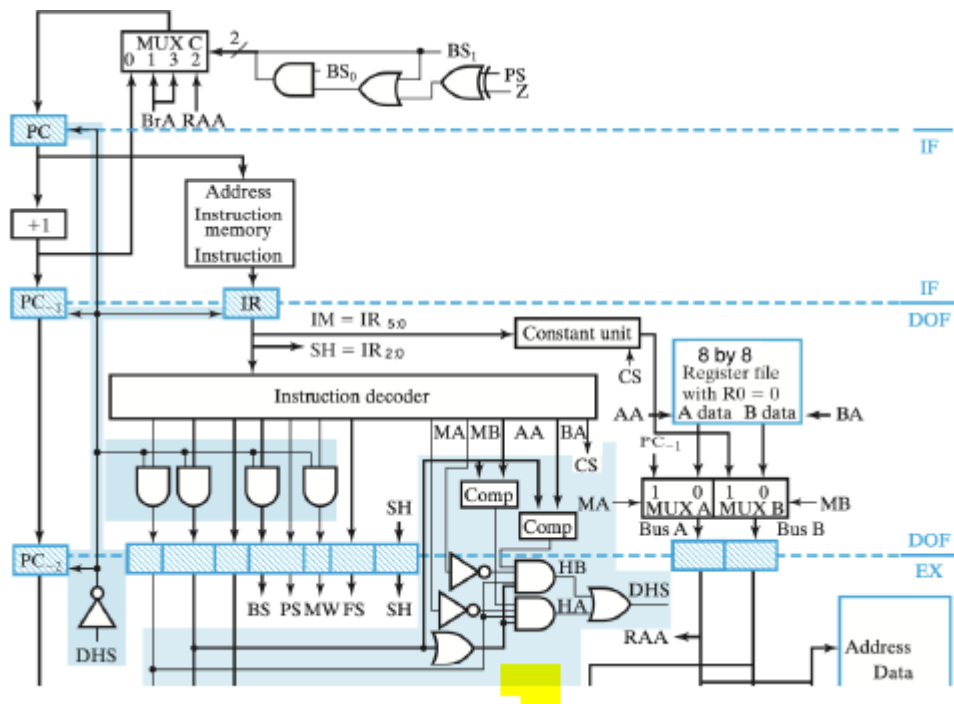


## Data Hazard & Branch detection

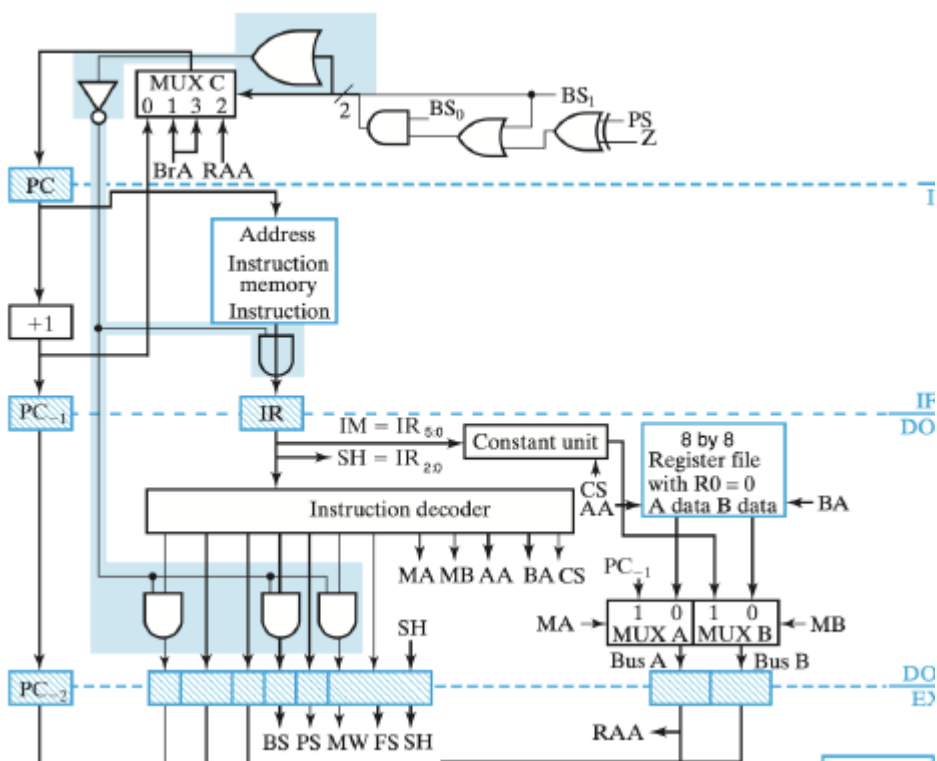
In pipeline architecture, it takes few cycles to write back to memory.

Because of this, if the data is being read from register file which has previous data, there will be wrong computation. So we have designed circuits to handle this

## Data hazard circuit



## Branch Detection circuit




## Simulation output of all the components

---

### 1. Instruction memory (ROM)

Input: `addr_line` - 8-bit address line

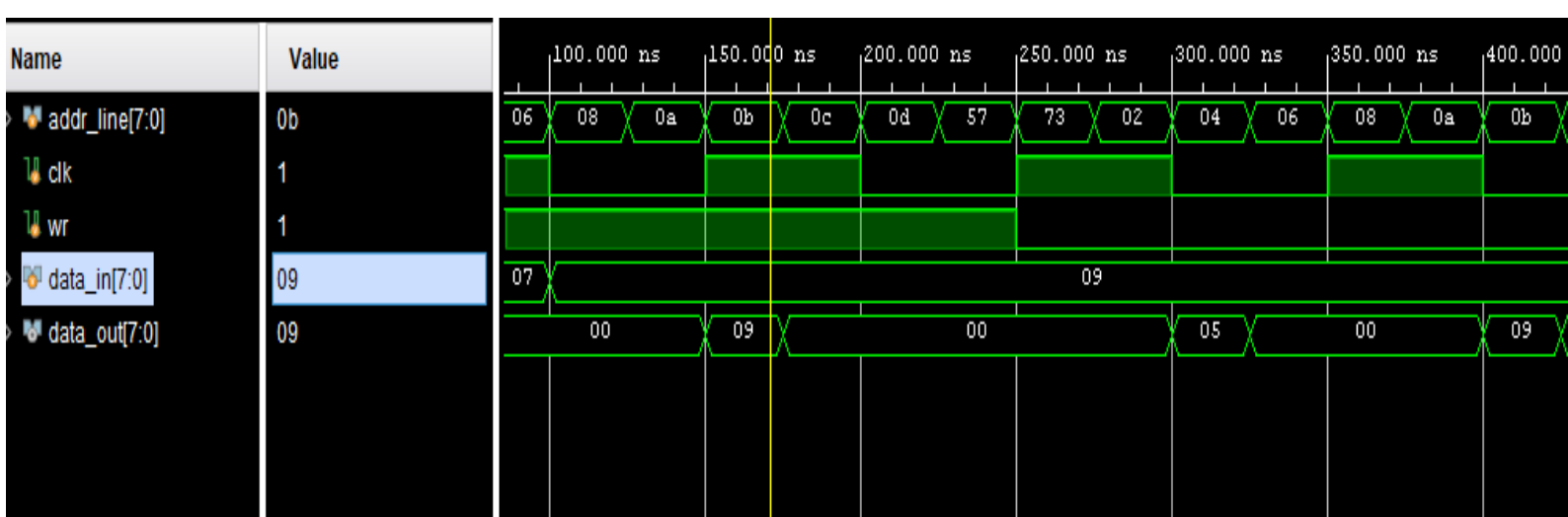
Output: `instr_line`- 17-bit instruction output fed into the instruction decoder

Name	Value	0.000 ns	100.000 ns	200.000 ns	300.000 ns	400.000 ns	500.000 ns					
>  addr_line[7:0]	01	00	01	02	03	04	05	06	07	08	09	0a
>  instr_line[16:0]	17404	17201	17404	0a656	19858	08b03	1e358	0c558	1aec0	18dc0	040b0	1b280

## 2. Data Memory (RAM)

**Input:** Clock signal(clk), 8-bit address line(addr\_line), Write enable(wr), 8-bit data input(data\_in) during write

**Output:** 8-bit data output(data\_out) from the memory



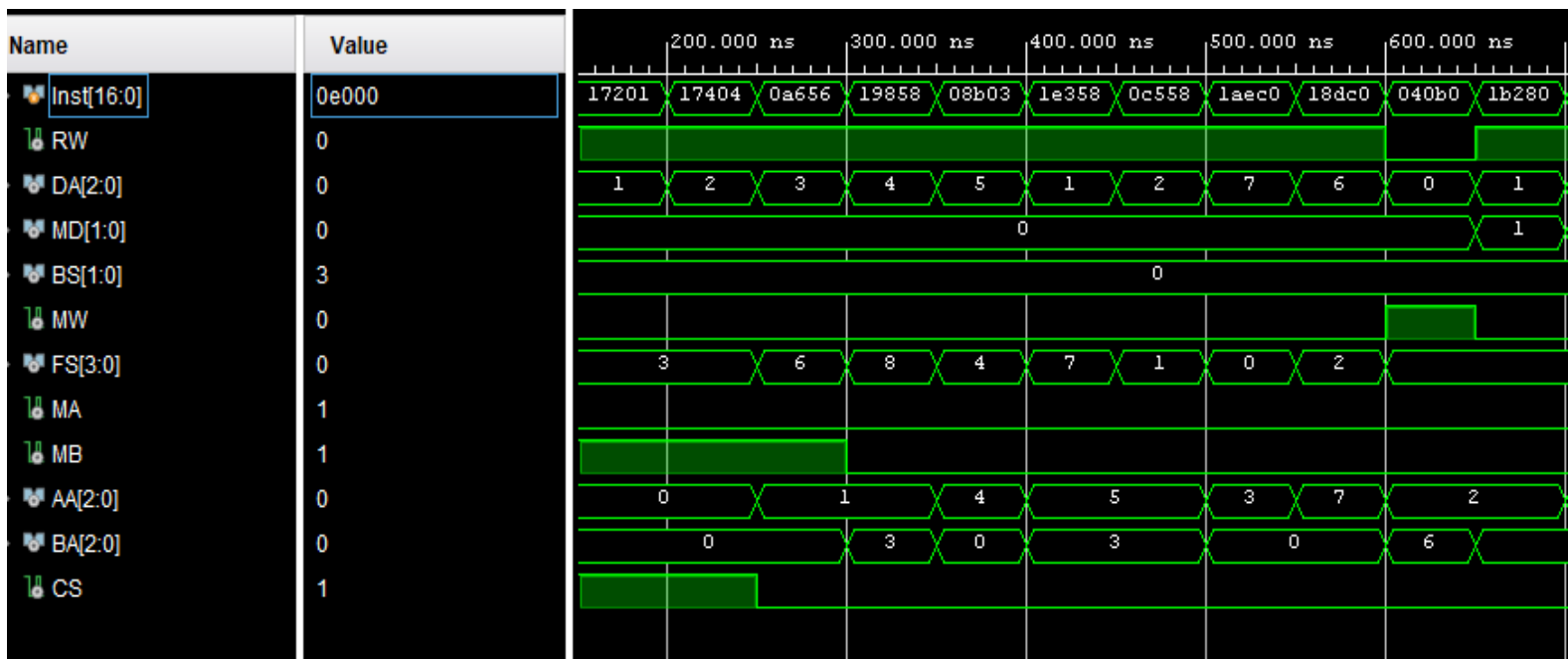
### 3. Instruction decoder

Input : [16:0]Inst — 17 bit instruction from the program memory

Output :

RW,[2:0]DA,[1:0]MD,[1:0]BS,PS,MW,[3:0]FS,MA,MB,[2:0]AA,[2:0]BA,CS

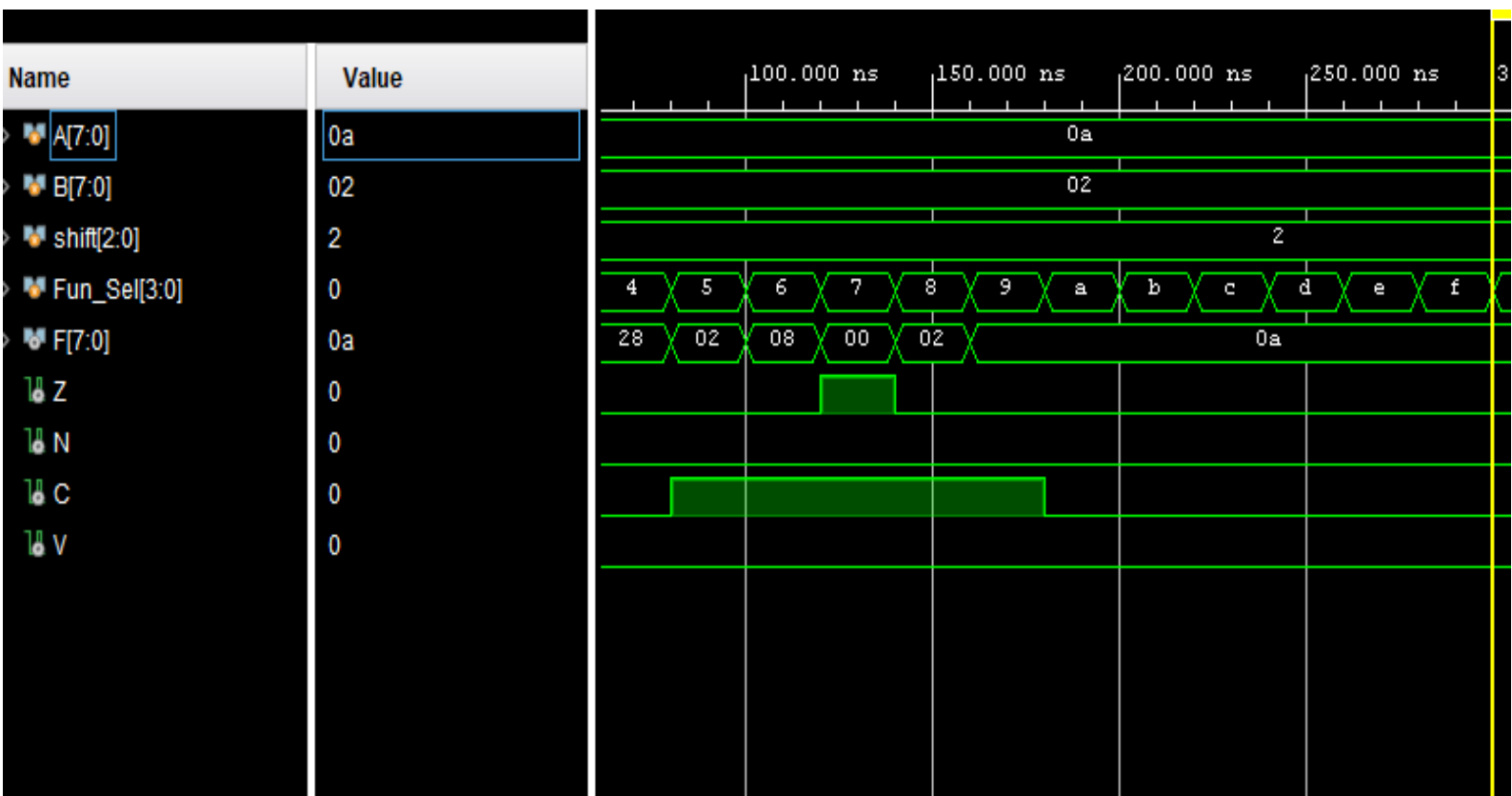
Various output data to different components in the circuit



## 4.Arithmetic Logic Unit (ALU)

**Input:** [7:0] A,B, ( 8-bit Inputs ) , [2:0]shift (shift bits), Fun\_Sel  
(Function select to perform a specific operation on the operands)

**Output:** F, ( ALU 8-bit Output), and 4 flags Z(zero), N(negative),  
C(carry), V(overflow)

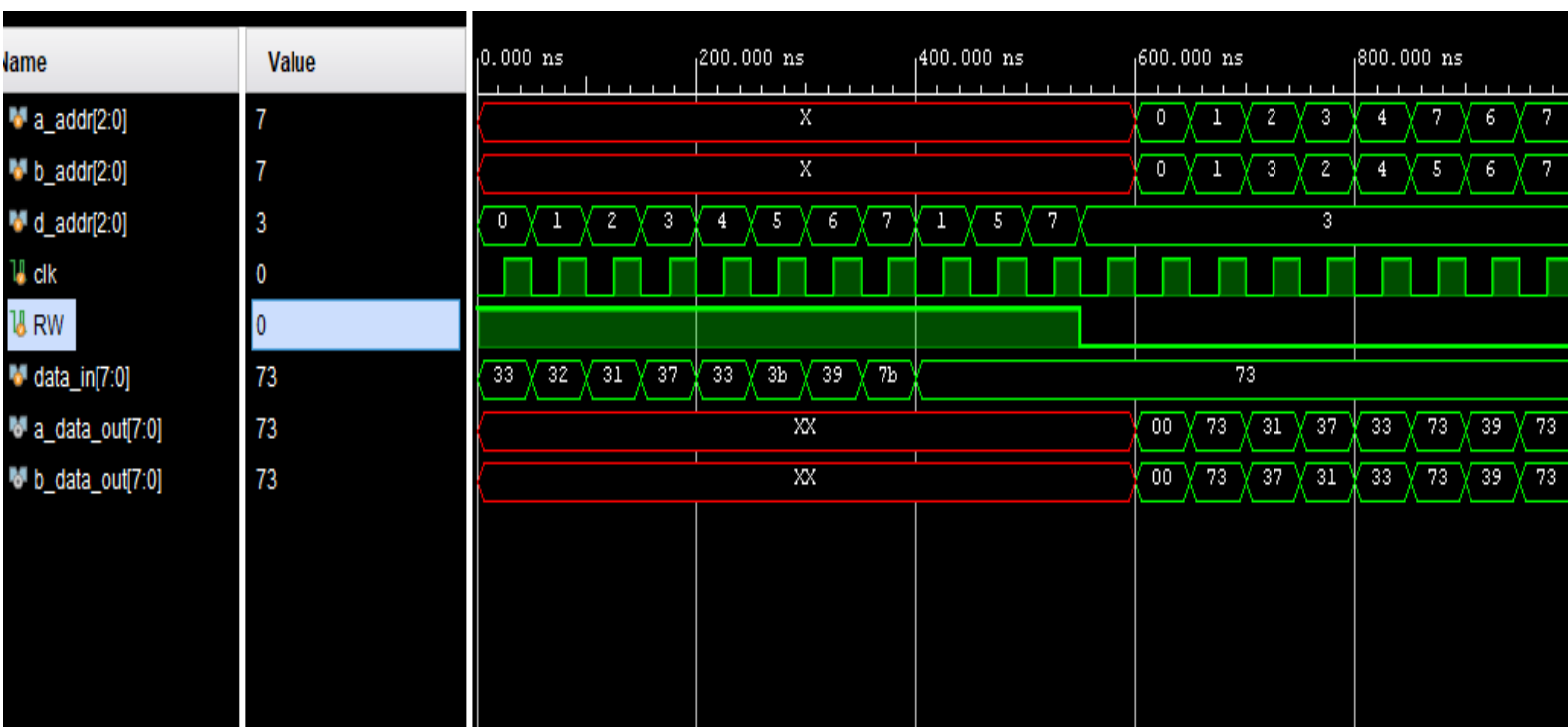


## 5. Register File

**Input:** clk(clock signal), RW(write enable), [2:0]a\_addr(a address), [2:0] b\_addr(b address), [2:0]d\_addr (d address), [7:0] data\_in(input data)

**Output:** [7:0]a\_data\_out(data output from a),  
[7:0]b\_data\_out(output data from b)

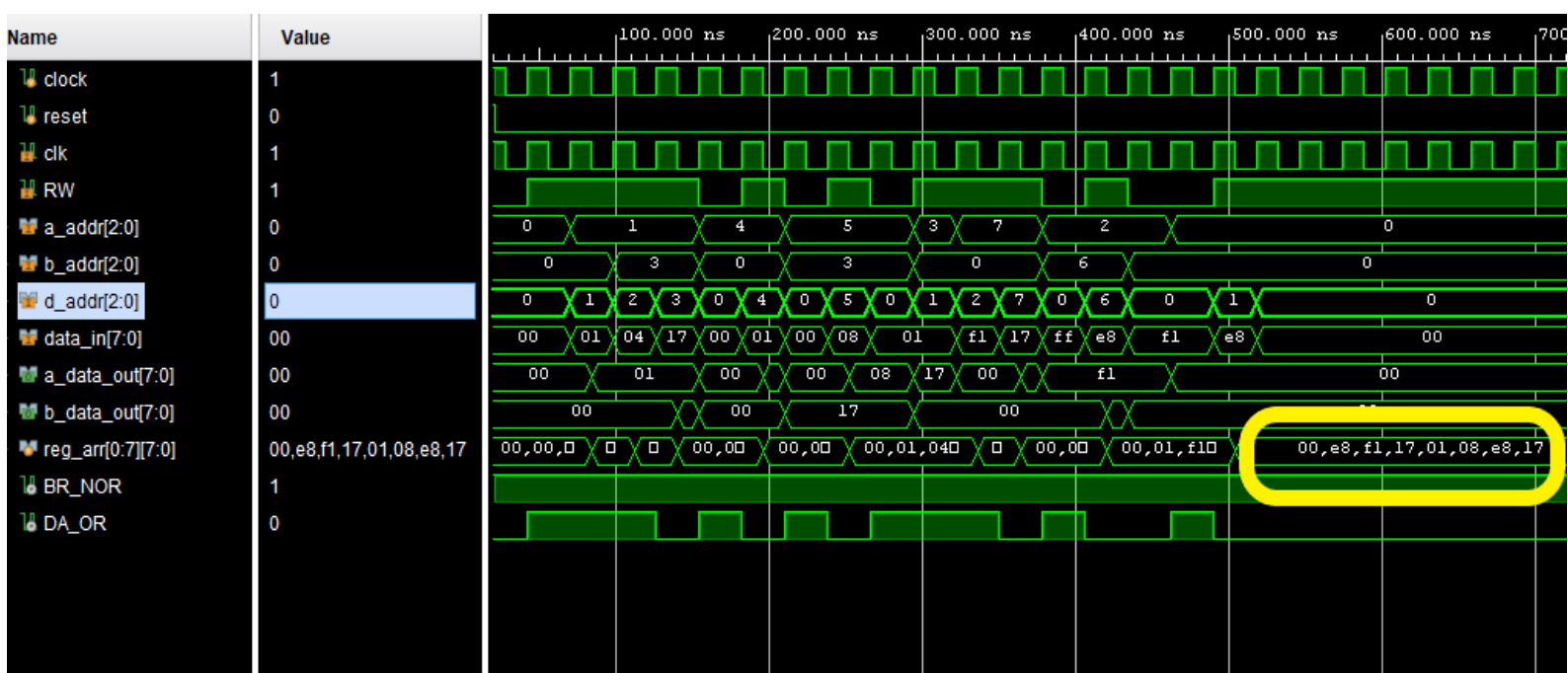
Data is written back in the negative edge of the clock in the register





## Final output which handles Data and Branch hazards

Below simulation output shows the various signals in the microprocessor. Each of them will be explained in the next section



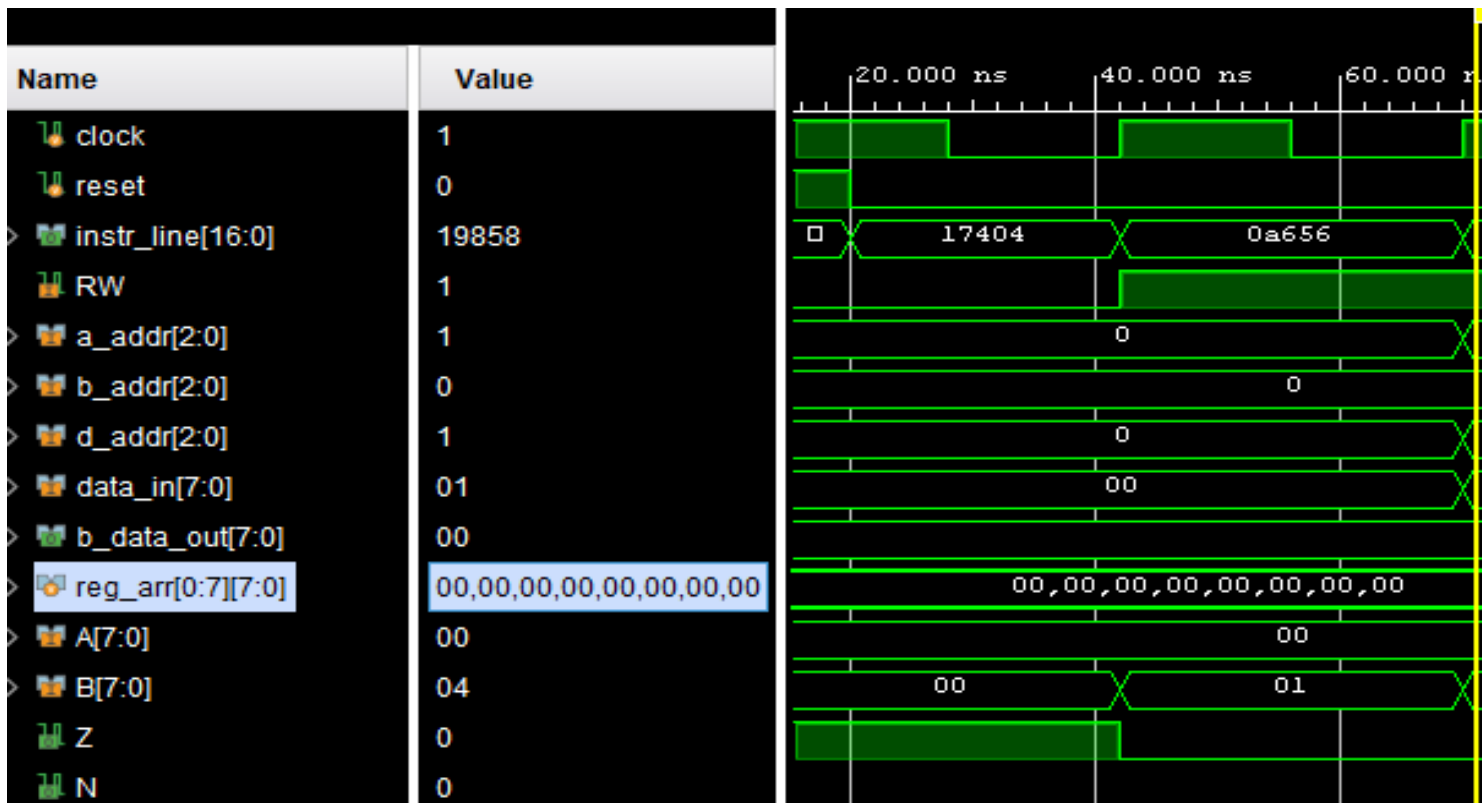
## Program run on the designed 8-bit MCU

```
R0 + 1 -> R1      // Add Immediate
R0 + 4 -> R2      // Add Immediate
R1 (+)l 16h -> R3  // XOR Immediate
R1 & R3 -> R4     // AND operation
R4 LSL (3) -> R5   // Logical Shift left operation
R5 < R3 -> R1      // Set if less than
R5 - R3 -> R2      // subtraction (2's compliment)
R3 -> R7          // Move operation
(~R7) -> R6       // 1's Compliment
R6 -> mem[R2]     // Memory Store
mem [R2] -> R1    // Memory Load
```

## Step by step calculation and Result

There are in total 8 registers R0-R7

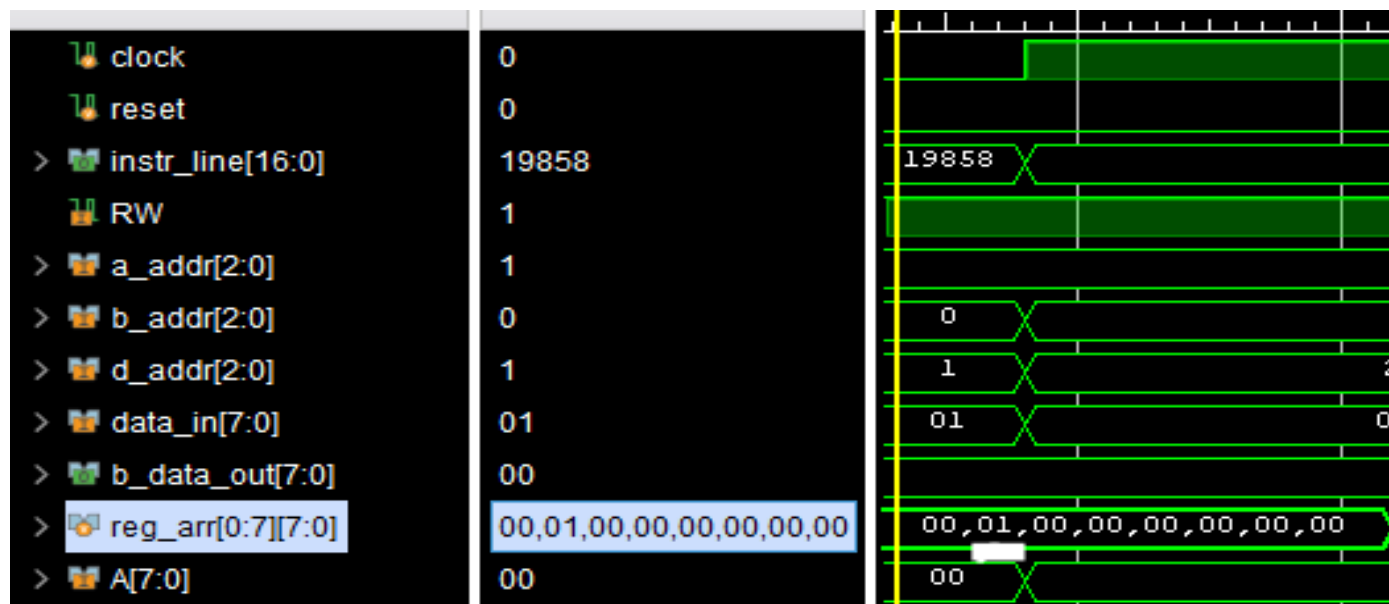
Initially all the registers will have value = 0



1.  $R0 + 1 \rightarrow R1$

$R0 = 00000000$

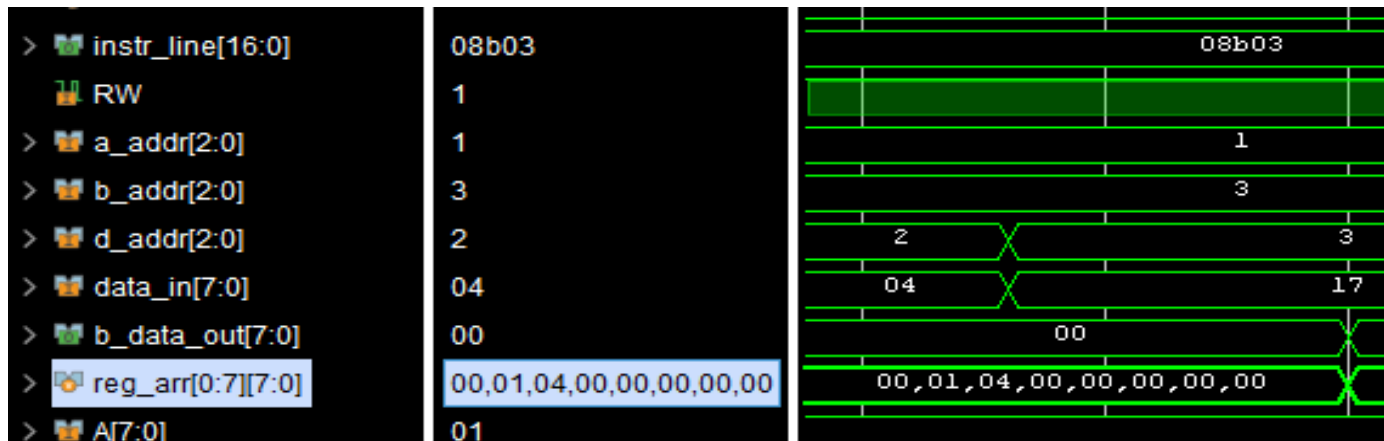
$R1 = 00000001 = 1h$



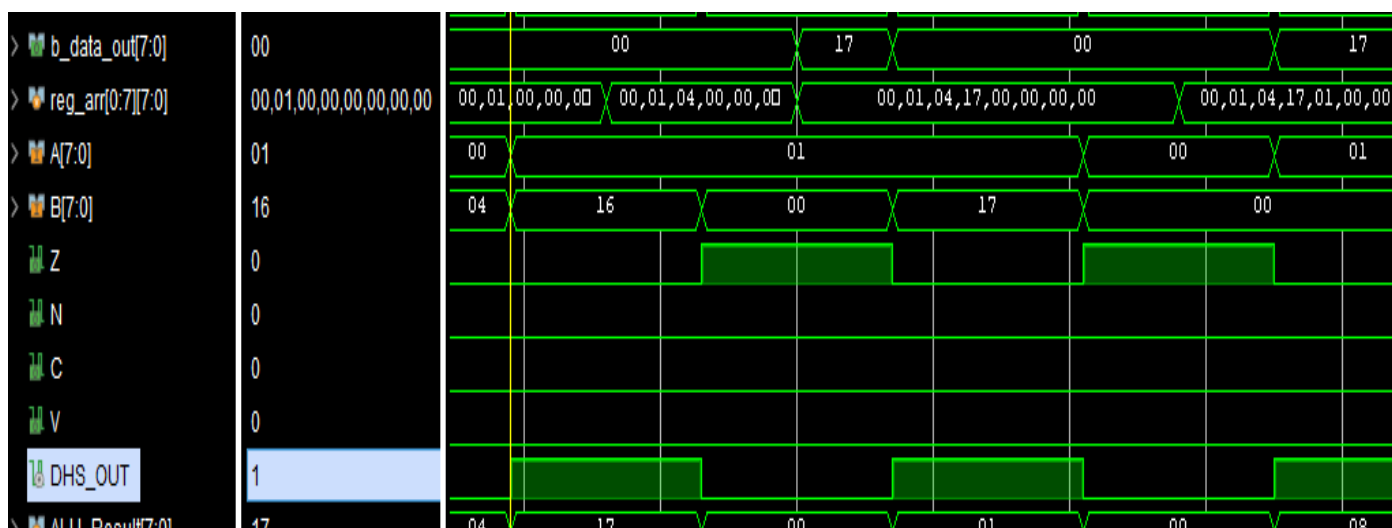
2.  $R0 + 4 \rightarrow R2$

$R0 = 00000000$

$R2 = 00000100 = 4h$



## DATA HAZARD!

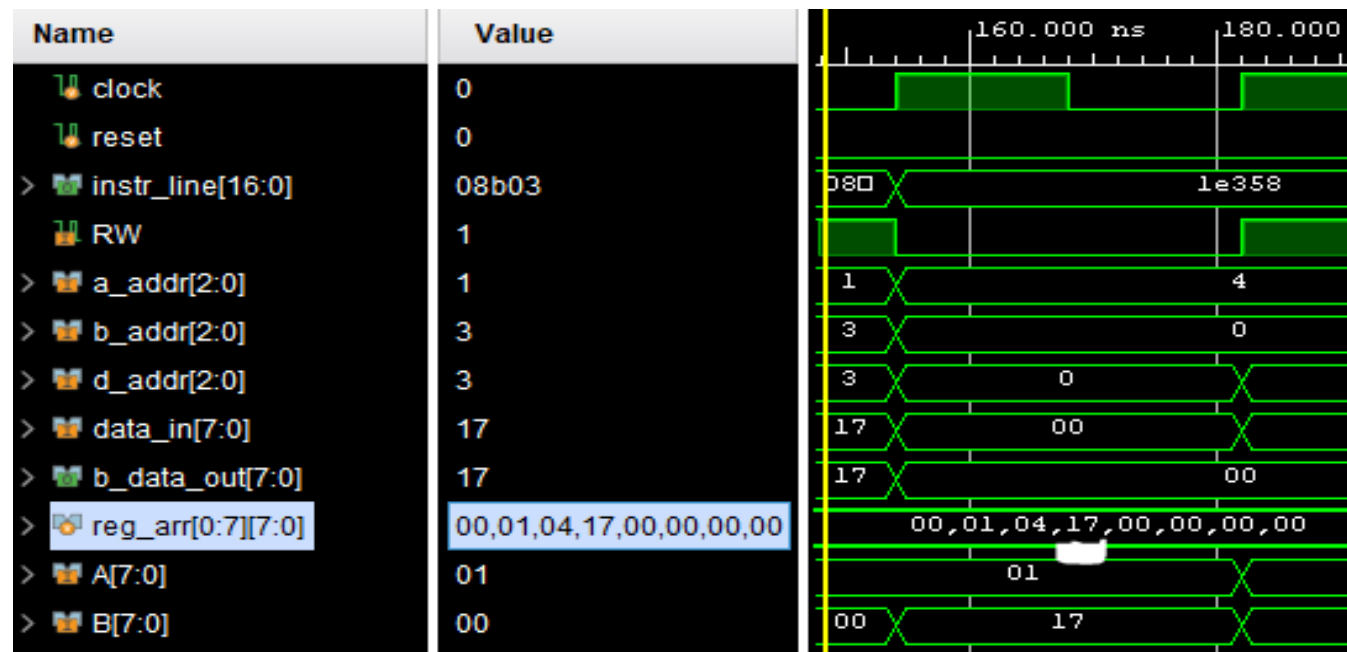


Data hazards is detected after 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> instruction which can be seen in DHS\_OUT to stall the pipeline

### 3. R1 (+) 16h -> R3

R1 = 00000001 = 1h

R3 = 00 010111 = 17h

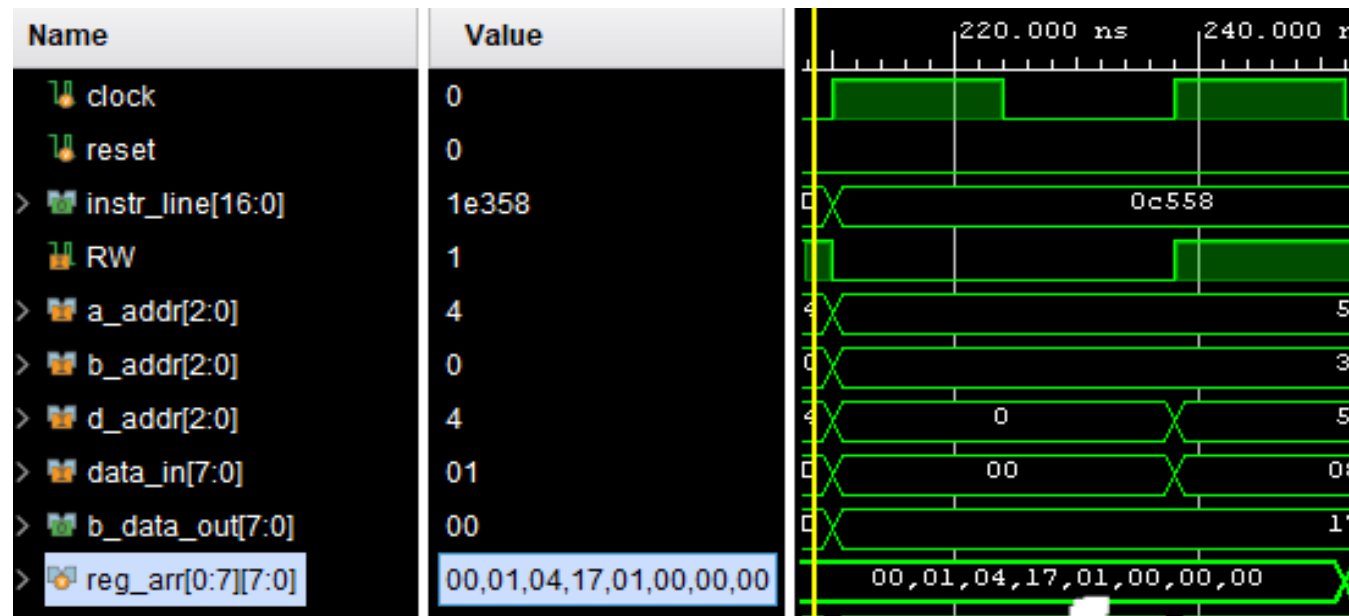


## 4. R1 &amp; R3 -&gt; R4

R1 = 00000001 = 1h

R3 = 00 010111 = 17h

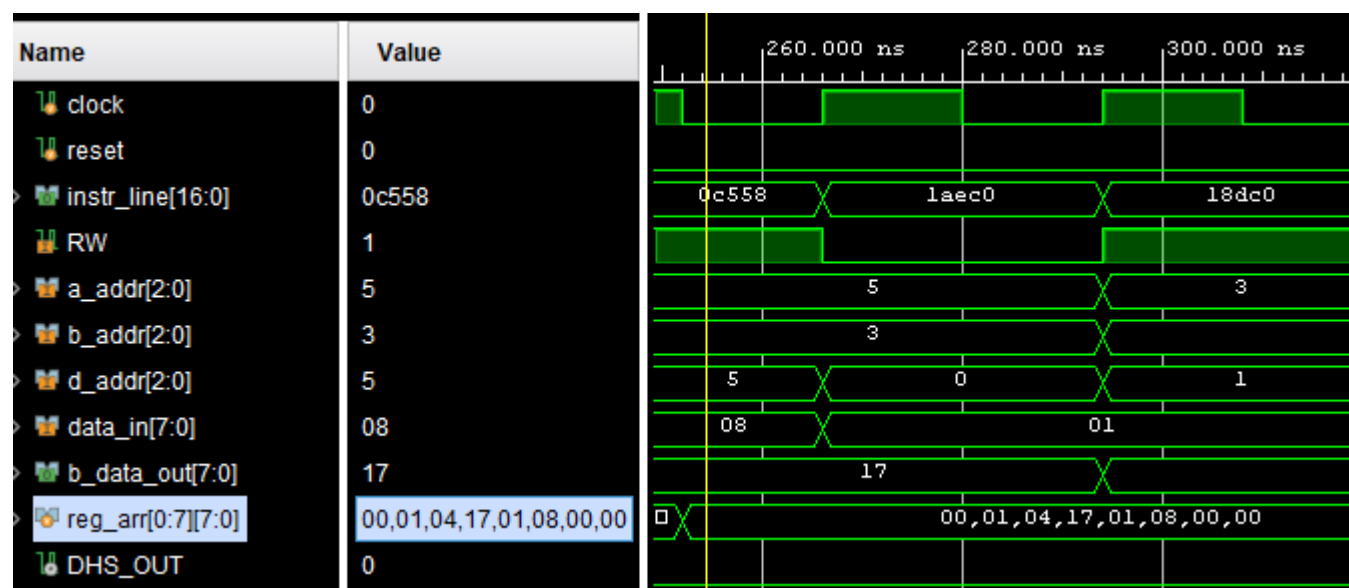
R4 = 00000001 = 1h



## 5. R4 LSL (3) -&gt; R5 , SH = 3

R4 = 00000001 = 1h

R5 = 00000100 = 8h

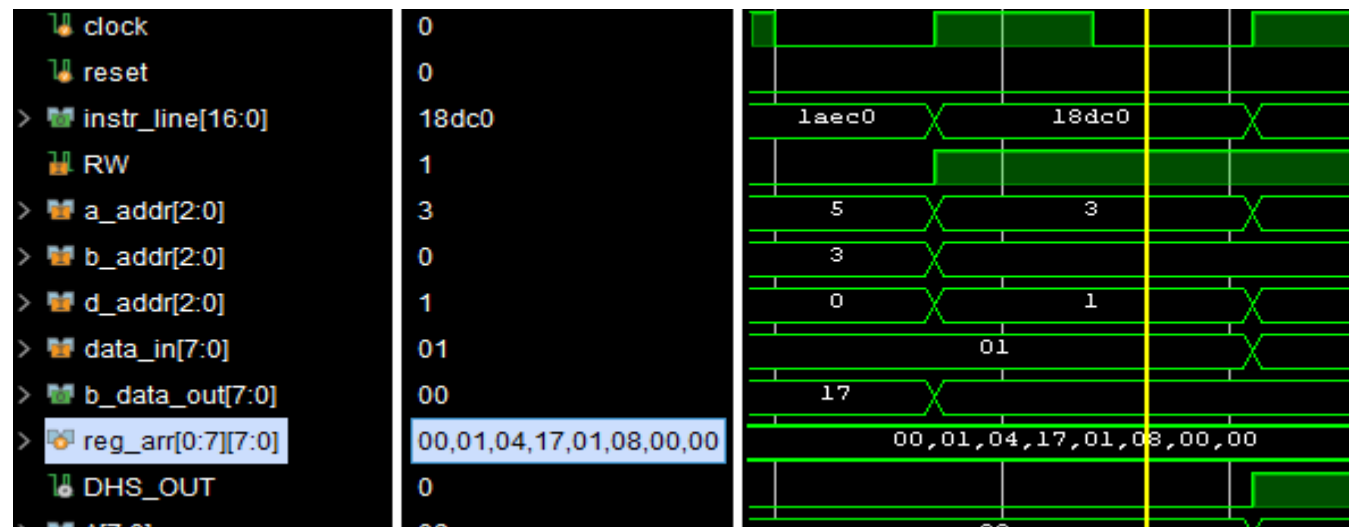


## 6. R5 < R3 -> R1

R5 = 00000100 = 8h

R3 = 00 010111 = 17h

R1 = 00000001 = 1h

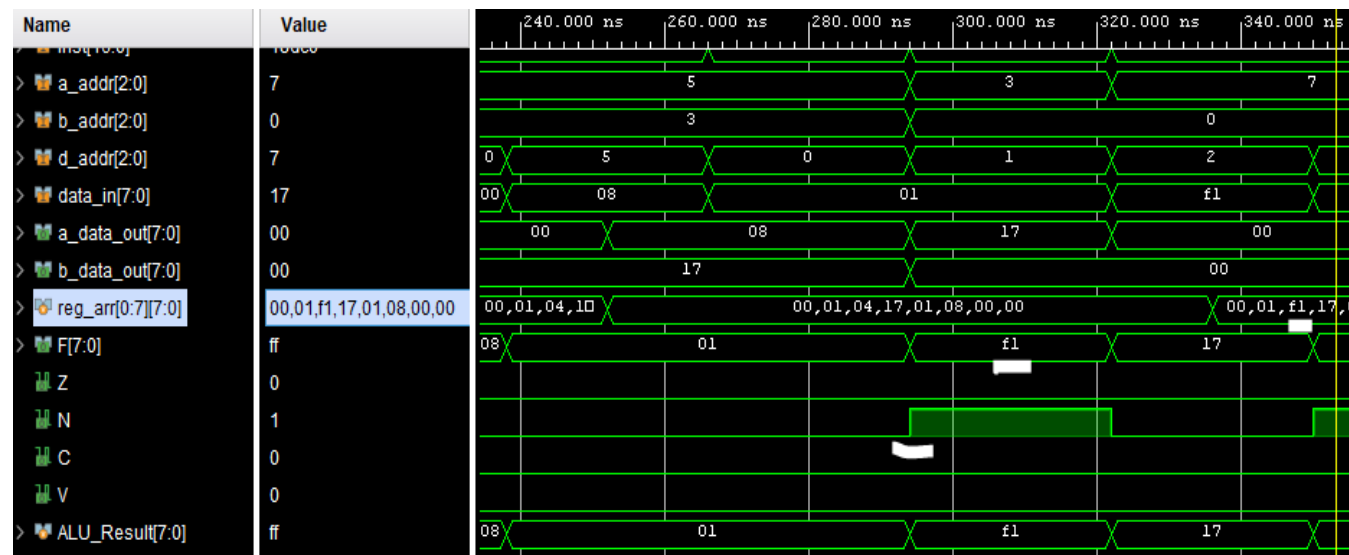


## 7. R5 - R3 -> R2

R5 = 00000100 = 8h

R3 = 00010111 = 17h

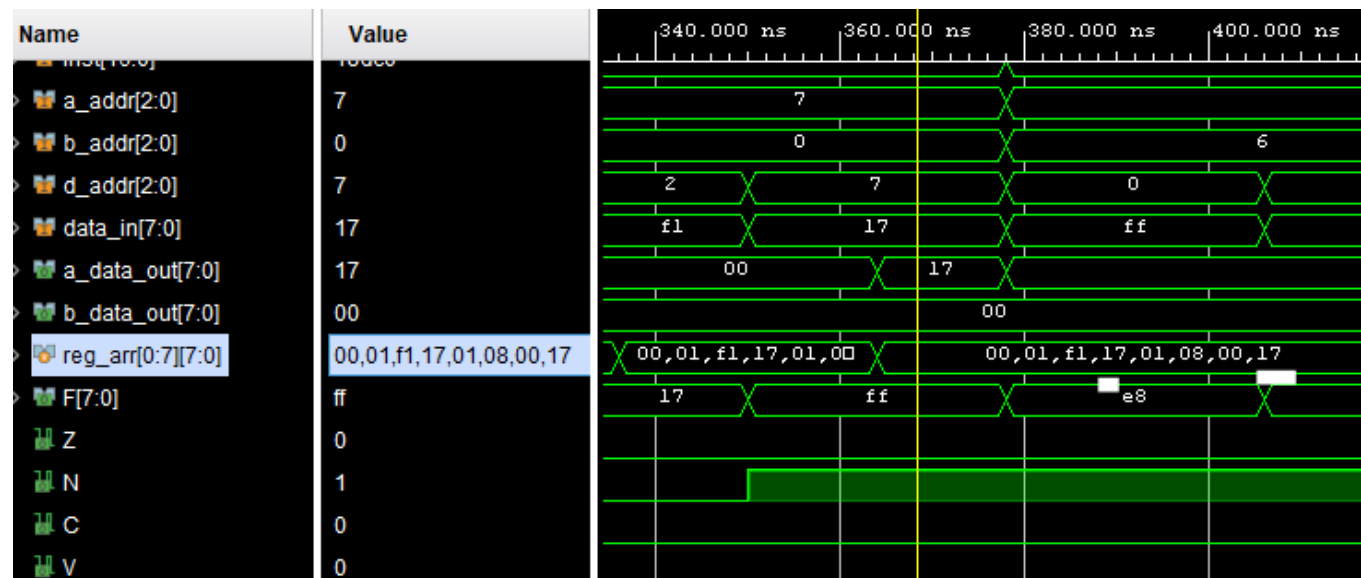
R2 = 11110001 = f1h, N=1



## 8. R3 -&gt; R7

R3 = 00010111 = 17h

R7 = 00010111 = 17h



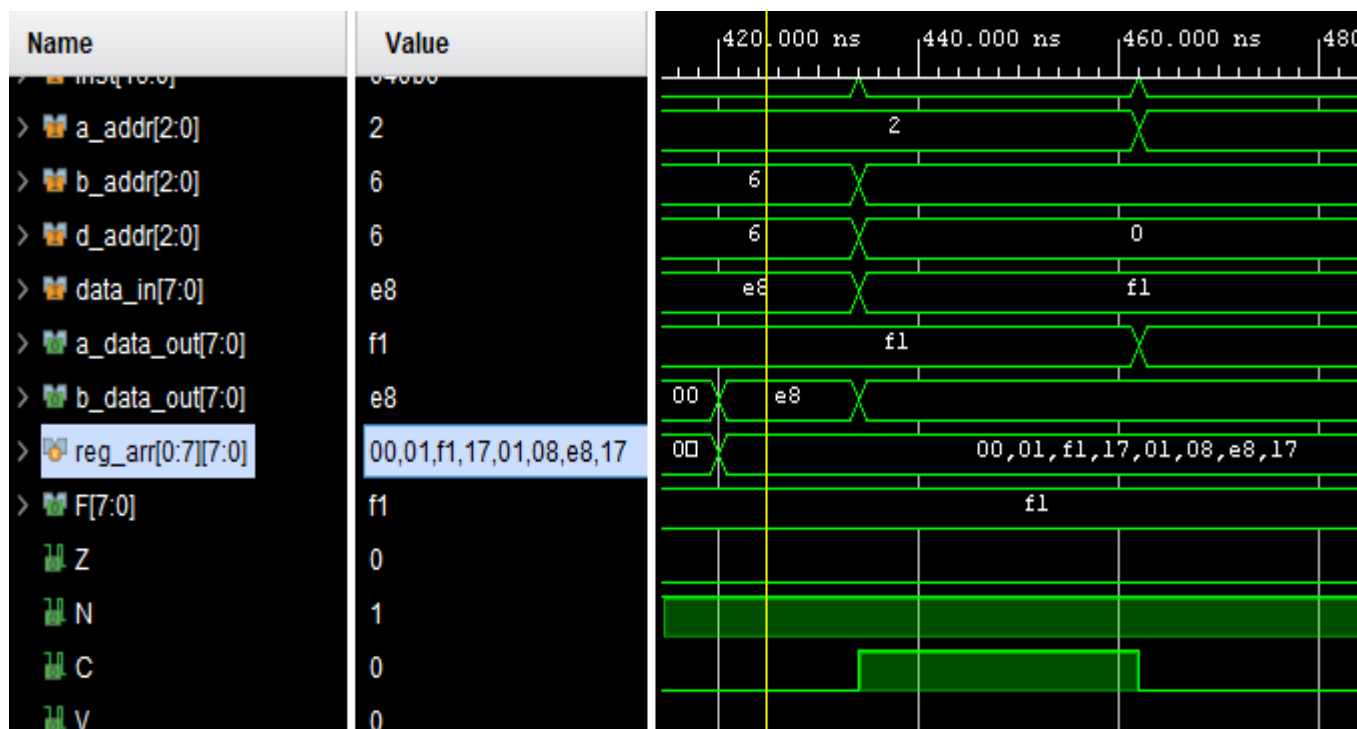
## 9. (~R7) -&gt; R6

R7 = 00010111 = 17h

~R7 = 11101000 = e8h

R6 = 11101000 = e8h





## 10. R6 -> mem[R2] STORE

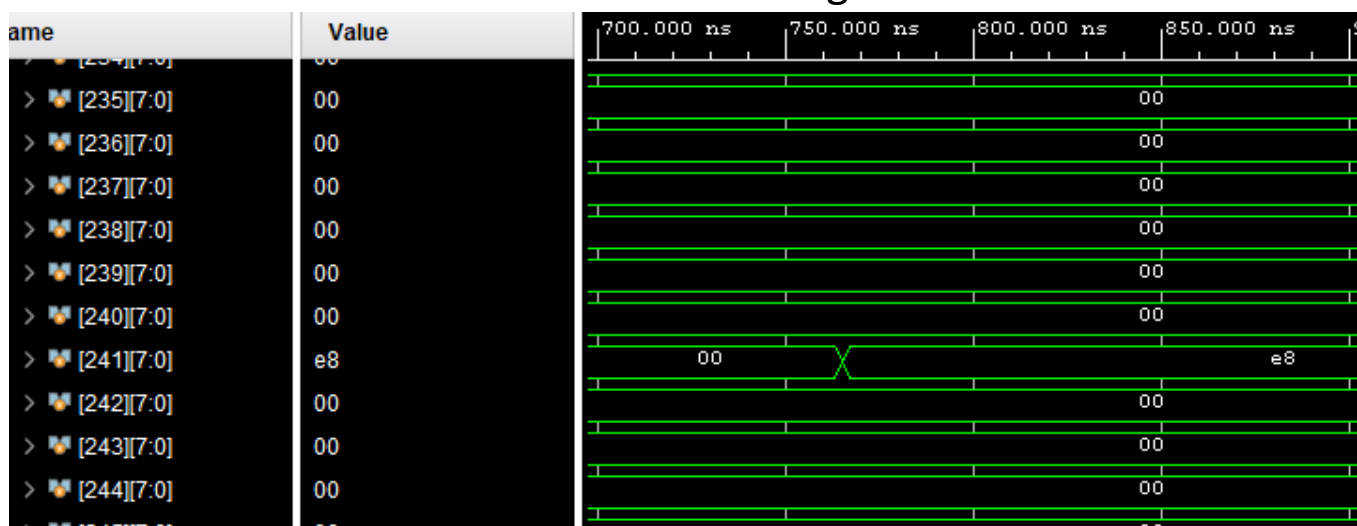
Data in register will be stored in address pointed by data in R2

In this program we have R6 = e8h

R2 = f1 h => 241d

so e8h will be stored in 241<sup>st</sup> location in the data memory

This is as shown in the below simulation figure

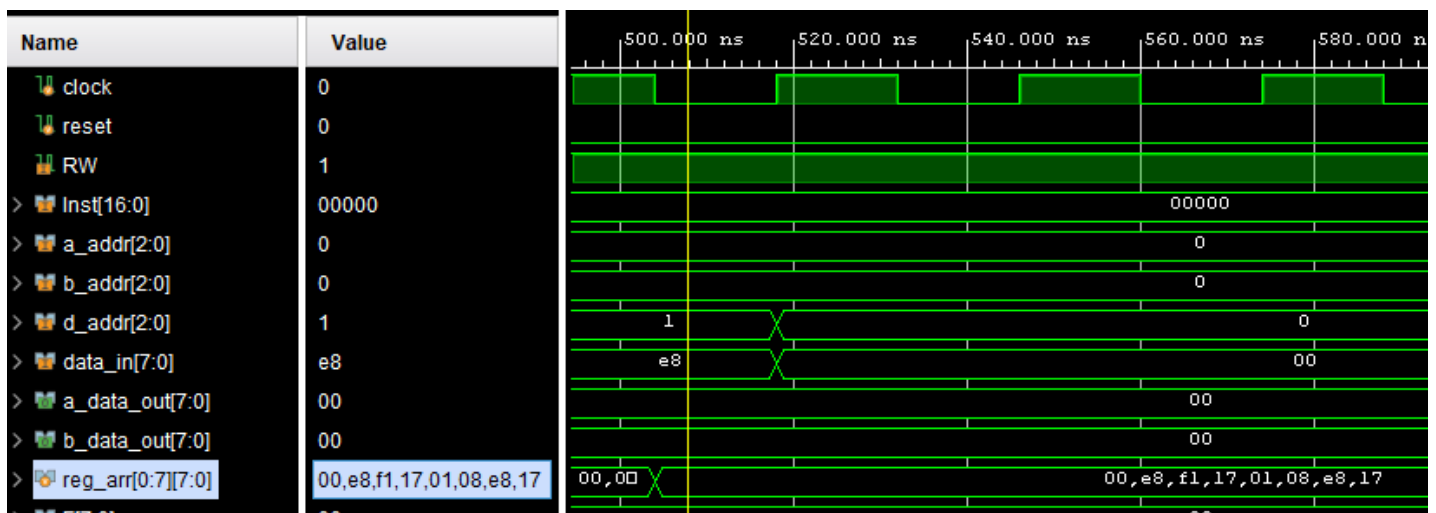


## 11. mem [R2] -> R1

The data pointed by R2 will be moved to R1

R2 points to location 241d which has value e8h

e8 will be moved to R1 and is as shown in the figure



## Result

After designing all the components needed for the 8-bit RISC MCU, and running the sample program, we have observed the expected outcome in the simulation waveform.

When there was a data hazard and branch hazard during pipeline, stall was introduced through the form of circuit and hence overcome the hazards

For this part of the project, design was done in Xilinx Vivado and output waveform is observed at each stage and at final stage.

## References

---

M. Morris Mano, Charles R. Kime, Tom Martin - Logic & Computer Design Fundamentals (5th Edition)-Prentice Hall (2015)