DESIGN OF 5-STAGE PIPELINED RISC-V PROCESSOR IN RTL

FINAL PROJECT SUBMISSION OF GROUP - 11

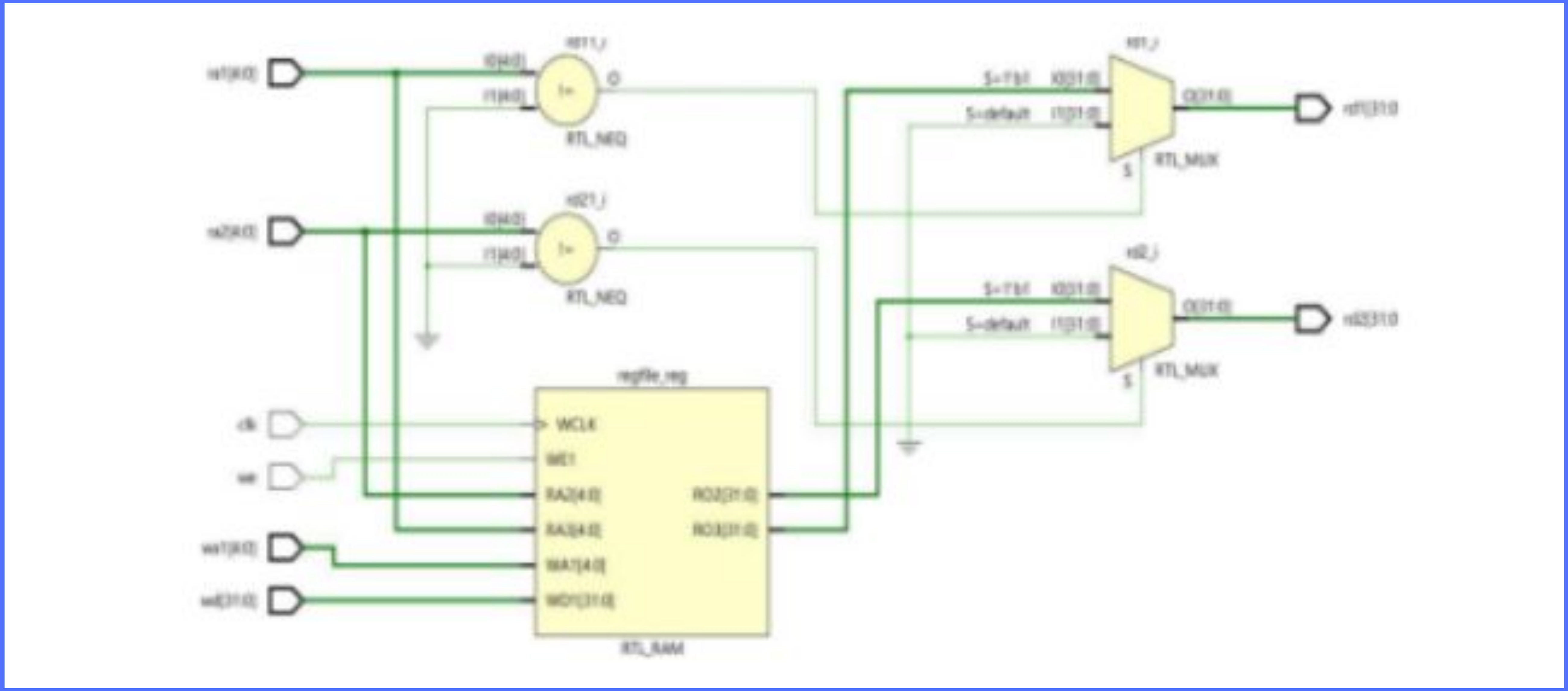
INTRODUCTION

In this project, we implemented the 5 stage[Fetch,Decode,Execute,Memory,Writeback] pipelined RISC-V processor in RTL[Verilog in this case]. Two separate memories have been used for storing instructions & data. To achieve better performance Bypassing and stalling logic is also implemented.

GROUP MEMBERS

- Anay Chauhan (2021013)
- Aditya Mishra (2021125)
- Gaurav Gupta (2021148)
- Himanshu Shekhar (202152)
- Utkarsh (2021215)

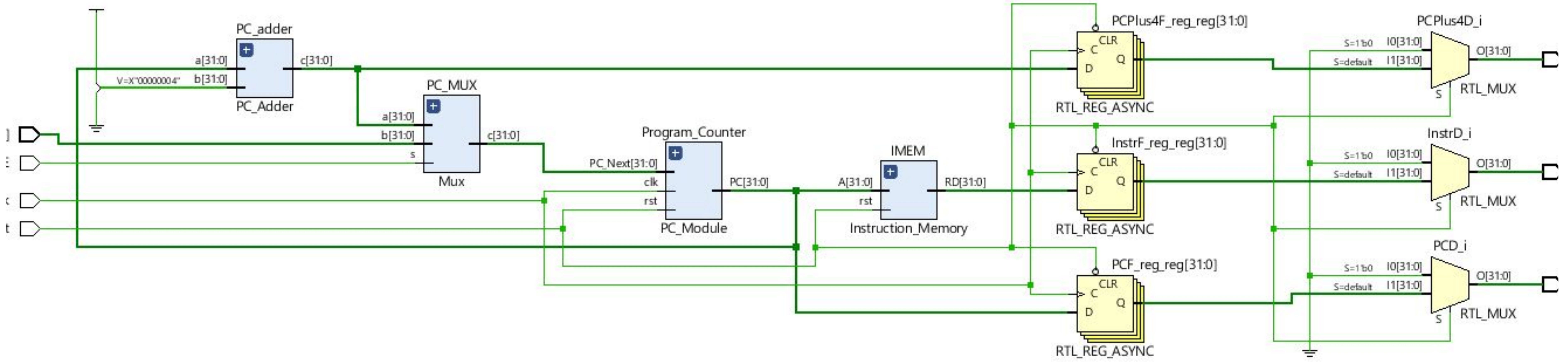
CONTENT INDEX



RTL

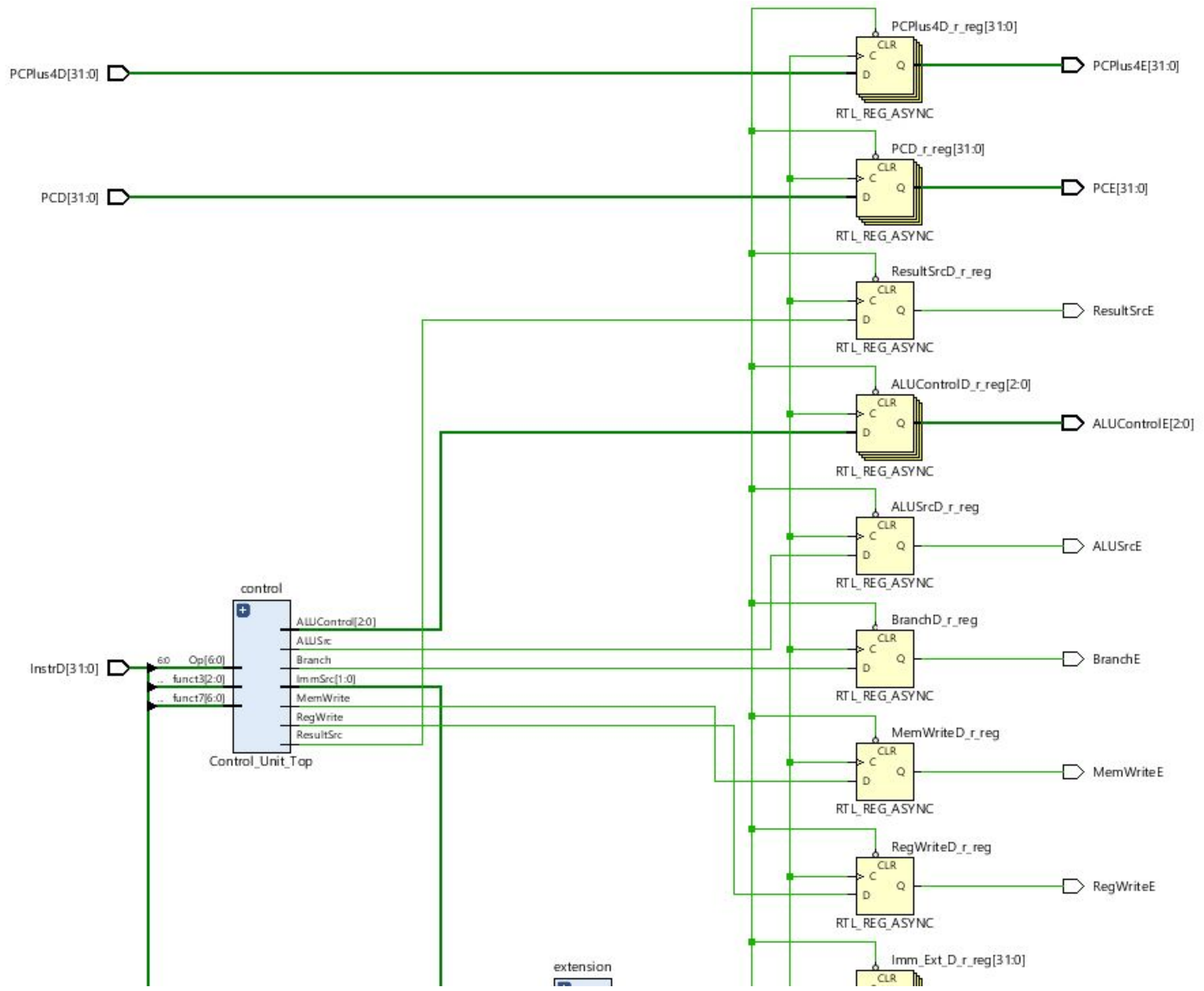
SCHEMATICS

Fetch Stage



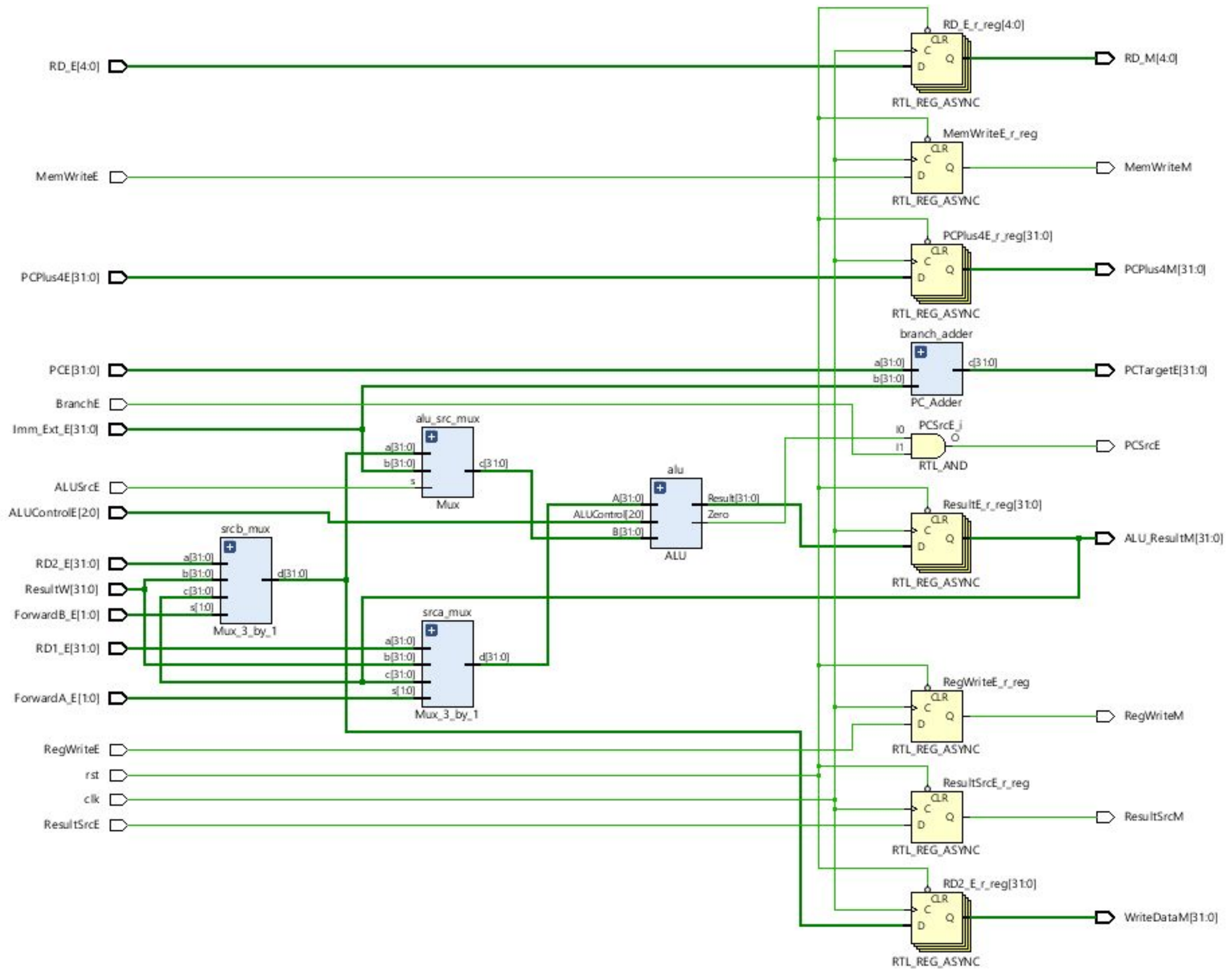
Fetch is the 1st stage of a processor pipeline. It uses a program counter that helps in fetching the instruction code from the instruction memory. The program counter is incremented by 4 after every instruction.

Decode Stage



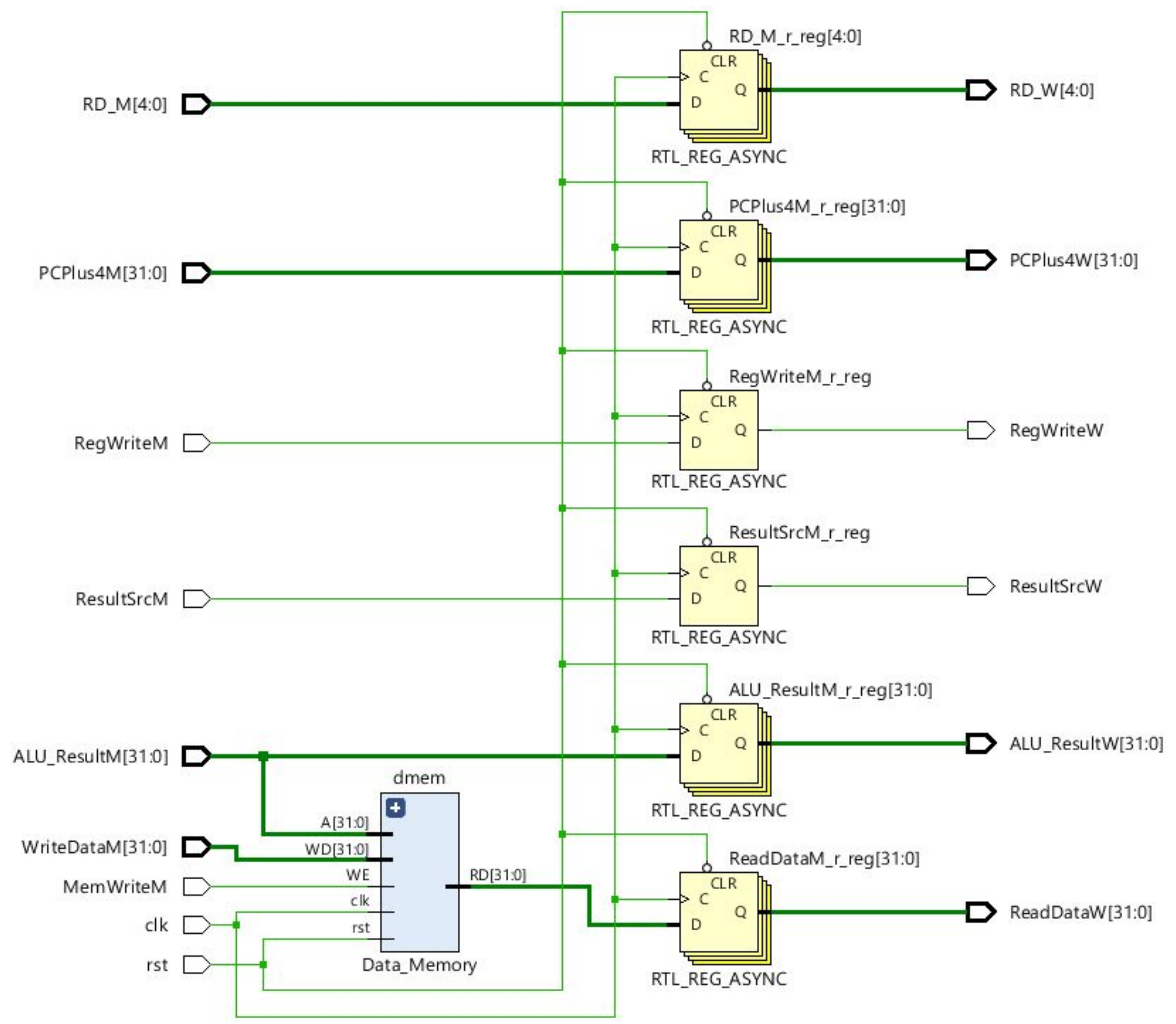
In the decode stage the instruction that has been fetched is decoded and the values are fetched from the registers mentioned in the instruction.

Execute Stage



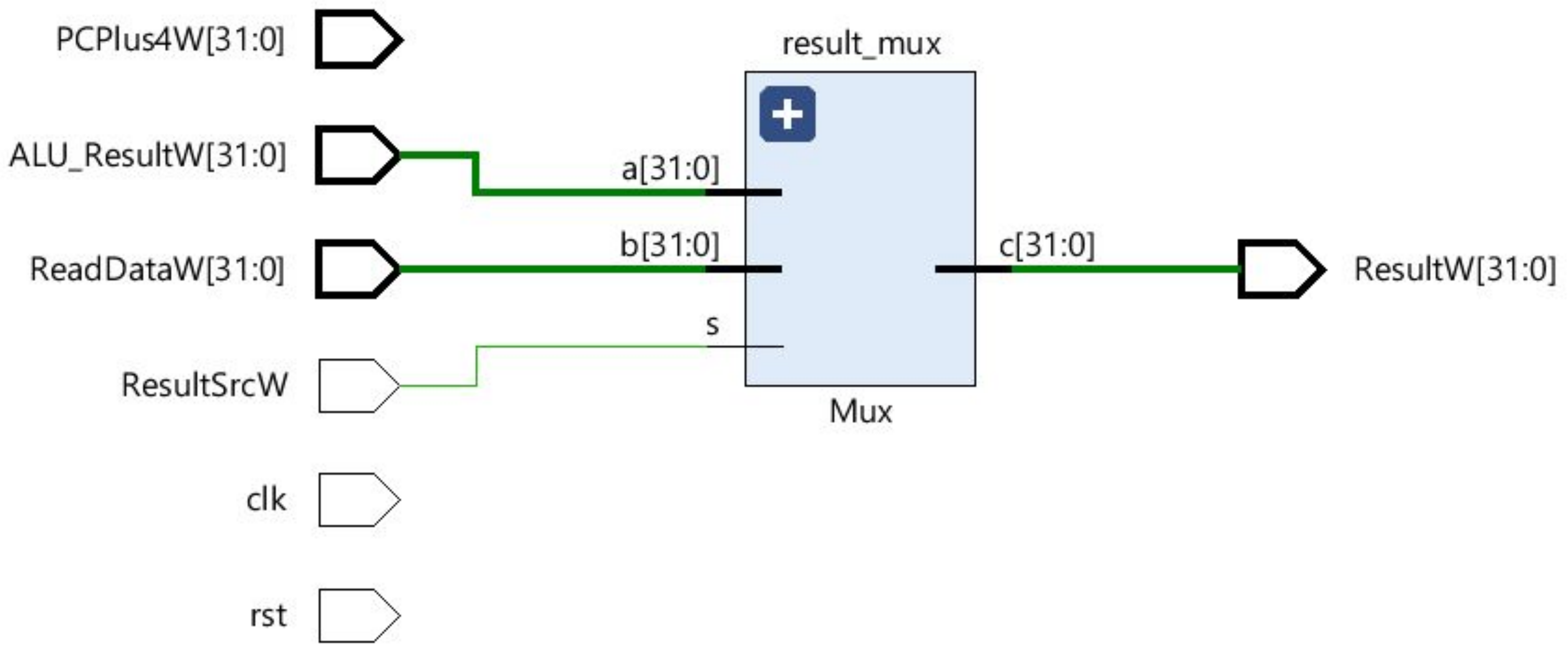
This stage will get the values from the decode stage & then perform the operations according to instruction.

Memory Stage



Memory stage in the processor will perform the operations of memory access & forward the signals which are not related with memory.

Writeback Stage



Writeback stage is for perform the write operations on The registers where we have to store the new value.

2 BIT BRANCH PREDICTOR

2-bit predictor: This predictor changes prediction only on two successive mispredictions. Two bits are maintained in the prediction buffer and there are four different states. Two states corresponding to a taken state and two corresponding to not taken state. The state diagram of such a predictor is given below.

Advantage of this approach is that a few atypical branches will not influence the prediction (a better measure of —the common casell). Only when we have two successive mispredictions, we will switch prediction. This is especially useful when multiple branches share the same counter (some bits of the branch PC are used to index into the branch predictor).

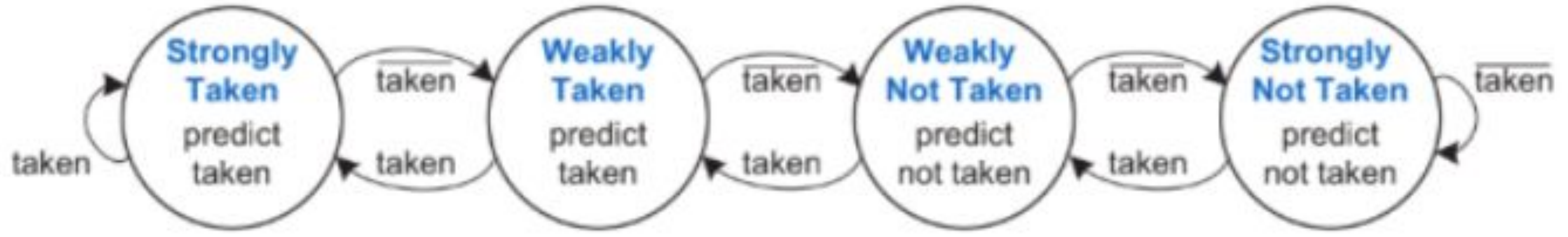


Figure 7.67 Two-bit branch predictor state transition diagram

EXTRA INSTRUCTIONS

LOADNOC

LoadNoc is used to store immediate integer value to a memory mapped register. The memory mapped register stores a memory location and the final argument of the instruction acts as offset to the location.

Usage:

```
        LOADNOC R1,R2,#8
```

The immediate value (r1) is stored at memory address (r2) + offset.

STORENOC

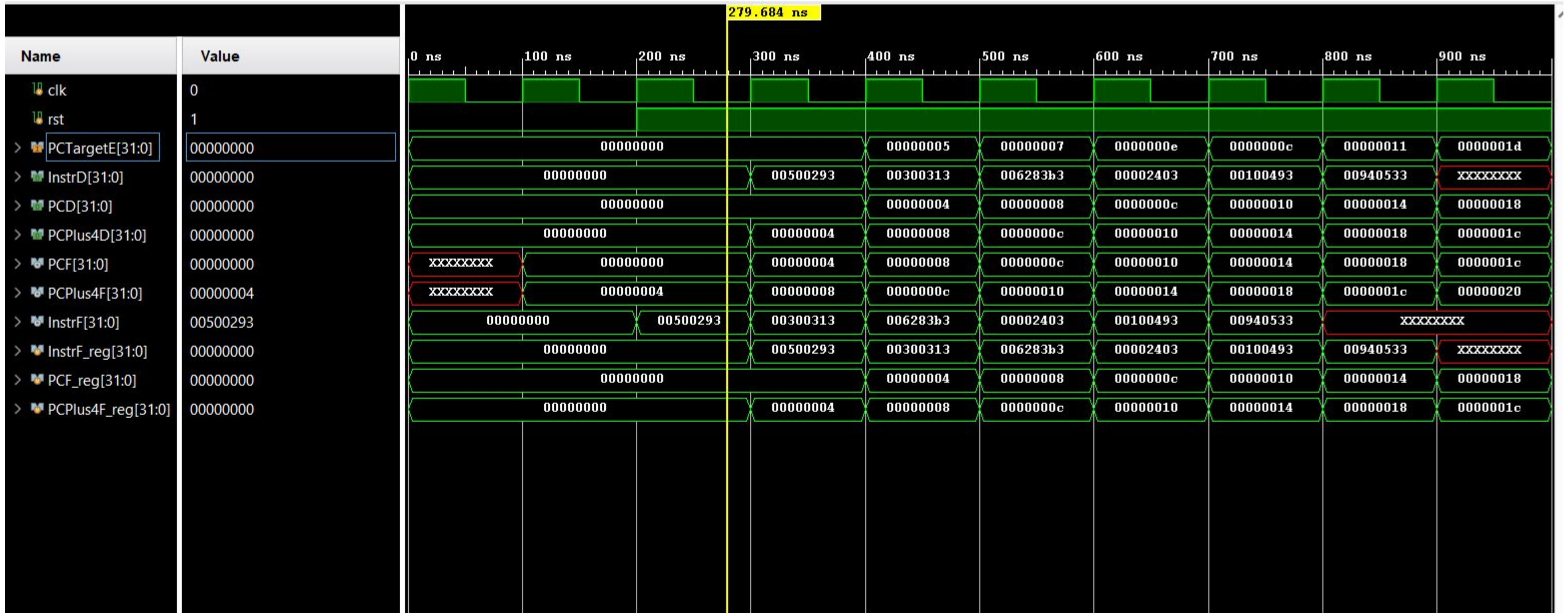
StoreNoc is similar to LoadNoc but in this the integer one is always stored at a hardcoded memory location. It takes no arguments. For example, when StoreNoc is used, the memory mapped register with our hardcoded address say, 0x4000 will be written with the value 1.

Usage:

```
        STORENOC
```

RESULT AND OBSERVATION

Testbench Simulation results

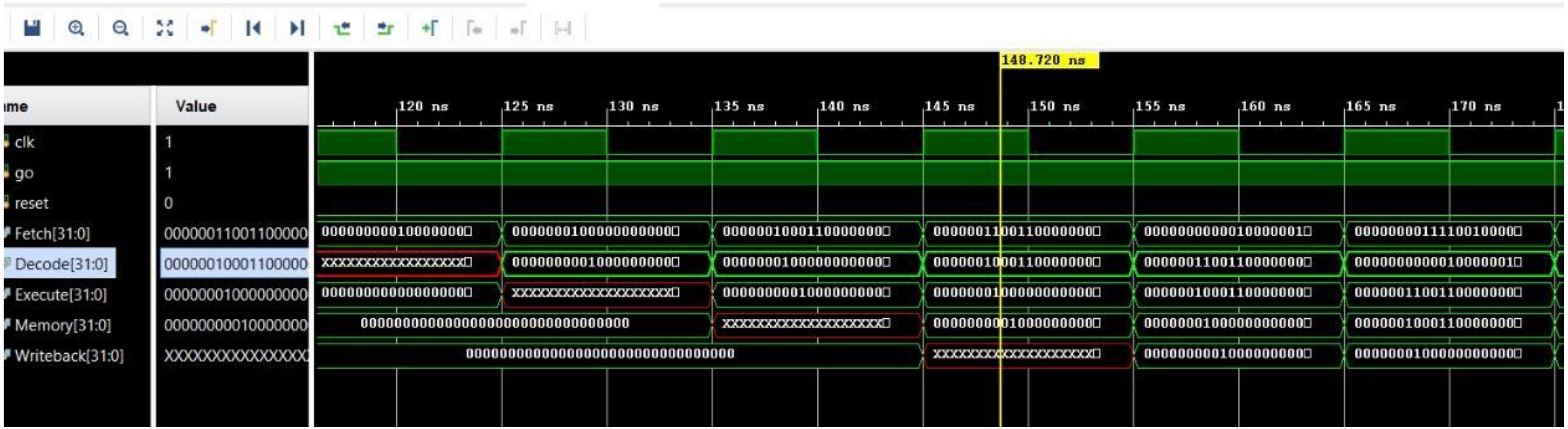


279.684 ns

0 ns | 100 ns | 200 ns | 300 ns | 400 ns | 500 ns | 600 ns | 700 ns | 800 ns | 900 ns

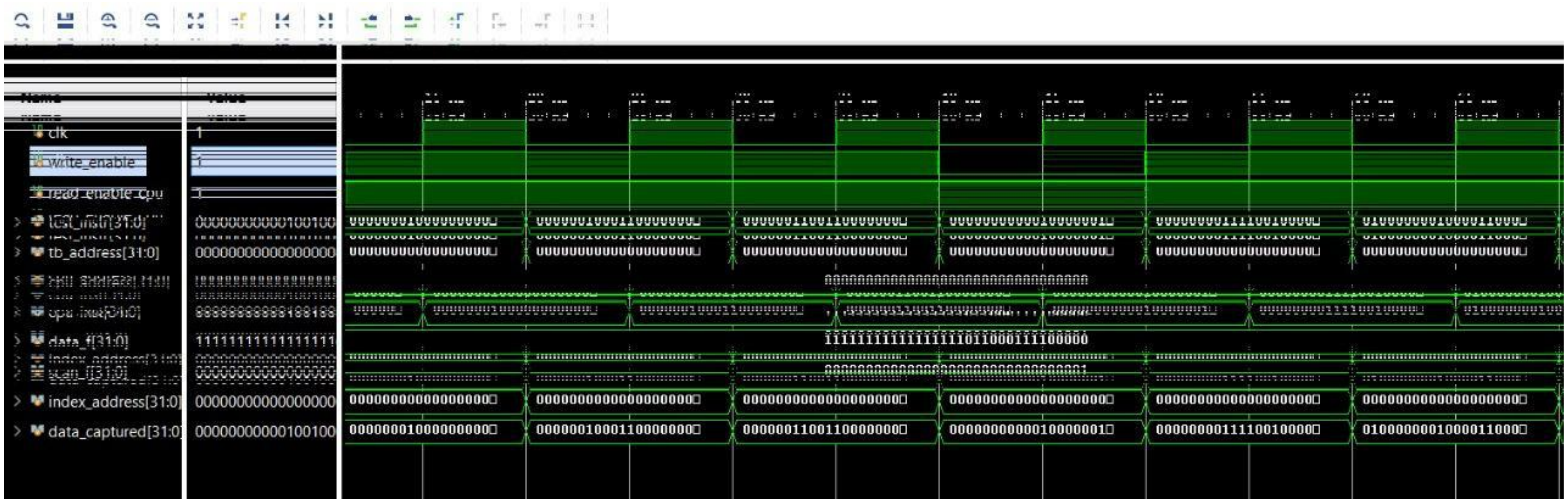
Name	Value	
clk	0	
rst	1	
PCTargetE[31:0]	00000000	00000000, 00000005, 00000007, 0000000e, 0000000c, 00000011, 0000001d
InstrD[31:0]	00000000	00000000, 00500293, 00300313, 006283b3, 00002403, 00100493, 00940533, XXXXXXXX
PCD[31:0]	00000000	00000000, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018
PCPlus4D[31:0]	00000000	00000000, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018, 0000001c
PCF[31:0]	00000000	XXXXXXXX, 00000000, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018, 0000001c
PCPlus4F[31:0]	00000004	XXXXXXXX, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018, 0000001c, 00000020
InstrF[31:0]	00500293	00000000, 00500293, 00300313, 006283b3, 00002403, 00100493, 00940533, XXXXXXXX
InstrF_reg[31:0]	00000000	00000000, 00500293, 00300313, 006283b3, 00002403, 00100493, 00940533, XXXXXXXX
PCF_reg[31:0]	00000000	00000000, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018
PCPlus4F_reg[31:0]	00000000	00000000, 00000004, 00000008, 0000000c, 00000010, 00000014, 00000018, 0000001c

Testbench Simulation results



```
                 148.720 ns
ame            Value               120 ns  125 ns  130 ns  135 ns  140 ns  145 ns  150 ns  155 ns  160 ns  165 ns  170 ns
clk            1
go             1
reset          0
Fetch[31:0]    0000001100110000    000000000100000000 | 00000000100000000000000 | 000000100011000000000 | 000000110011000000000 | 000000000000100000001 | 000000000111100100000
Decode[31:0]   0000001000110000    XXXXXXXXXXXXXXXXXXX | 000000000100000000000 | 00000000100000000000000 | 000000100011000000000 | 000000110011000000000 | 000000000000100000001
Execute[31:0]  0000000100000000    00000000000000000000 | XXXXXXXXXXXXXXXXXXXXXX | 000000000100000000000 | 00000000100000000000000 | 000000100011000000000 | 000000110011000000000
Memory[31:0]   0000000001000000    00000000000000000000000000000000 | XXXXXXXXXXXXXXXXXXXXXX | 000000000100000000000 | 00000000100000000000000 | 000000100011000000000
Writeback[31:0] XXXXXXXXXXXXXXX    00000000000000000000000000000000 | XXXXXXXXXXXXXXXXXXXXXX | 000000000100000000000 | 00000000100000000000000
```

Testbench Simulation results



```
Name                  Value
clk                   1
write_enable          1
read_enable_cpu       1
test_instr[31:0]      0000000000000100100
tb_address[31:0]      0000000000000000000
data_f[31:0]          1111111111111111
index_address[31:0]   0000000000000000000
scan_i[31:0]          0000000000000000000
index_address[31:0]   0000000000000000000
data_captured[31:0]   0000000000000100100
```

```
00000000000000000000000000000000
11111111111111111111011000111100000
00000000000000000000000000000001
```

```
index_address[31:0]:  000000000000000000 | 00000000000000000000 | 00000000000000000000 | 00000000000000000000 | 00000000000000000000 | 0000000000000000000
data_captured[31:0]:  000000010000000000 | 00000010001100000000 | 00000011001100000000 | 00000000000100000001 | 00000000111100100000 | 0100000001000011000
```