SC3050

Advanced Computer Architecture

Lab 3 briefing

College of Computing and Data Science

Nanyang Technological University

Objectives

1. Implementation and functional verification of four-stage pipeline architecture for R-type instructions.
2. Design, Implementation and functional verification of five-stage pipeline architecture for R&D-type instructions.

Points to be noted (R type- 4–stage pipeline)

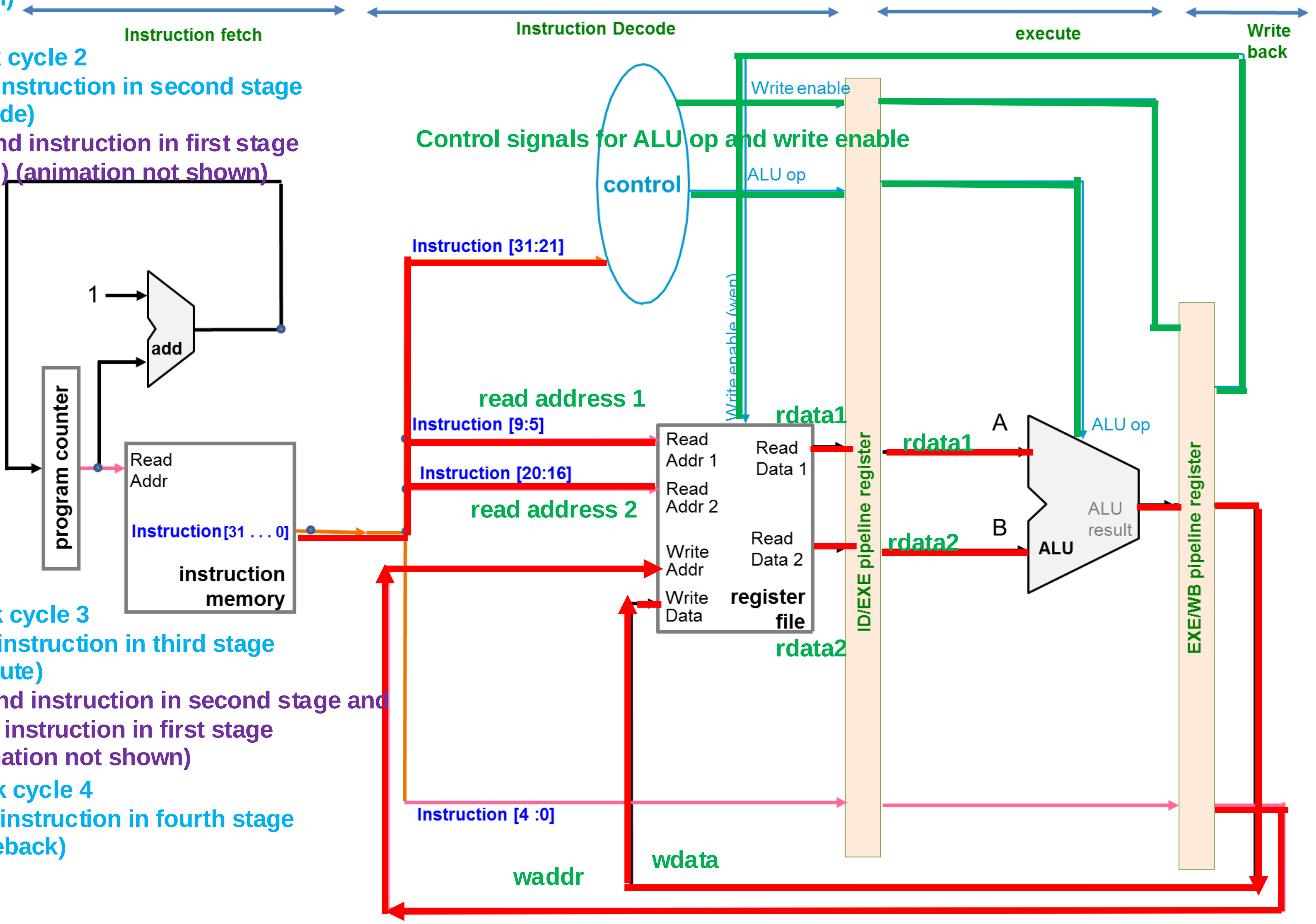
- Program counter available in ‘PC.v’
- Instruction memory available in ‘imemory.v’.
- You need data memory only for LDUR/STUR instructions.
- The ‘imemory.v’ uses file operation and the instructions are accessed from a file.

(Note that memory is clocked and hence we have a clock cycle delay for getting the output from memory.)

- Datapath modules discussed in lab 2 are available at ‘Alu.v’, ‘control.v’ and ‘regfile.v’. In order to initialize ‘ regfile.v’, we have assigned a value inside ‘regfile.v’ itself.
- The ‘ID_EXEstage.v’ is the pipeline register between decode and execute section and “EXE_WBstage.v” is the pipeline register between execute and write back section
- Please note that you need to pass your write address to the pipeline register as write address should come at the same time of write data.

Clock cycle 1
first instruction in first stage
(fetch)

Clock cycle 2
First instruction in second stage
(decode)
Second instruction in first stage
(fetch) (animation not shown)

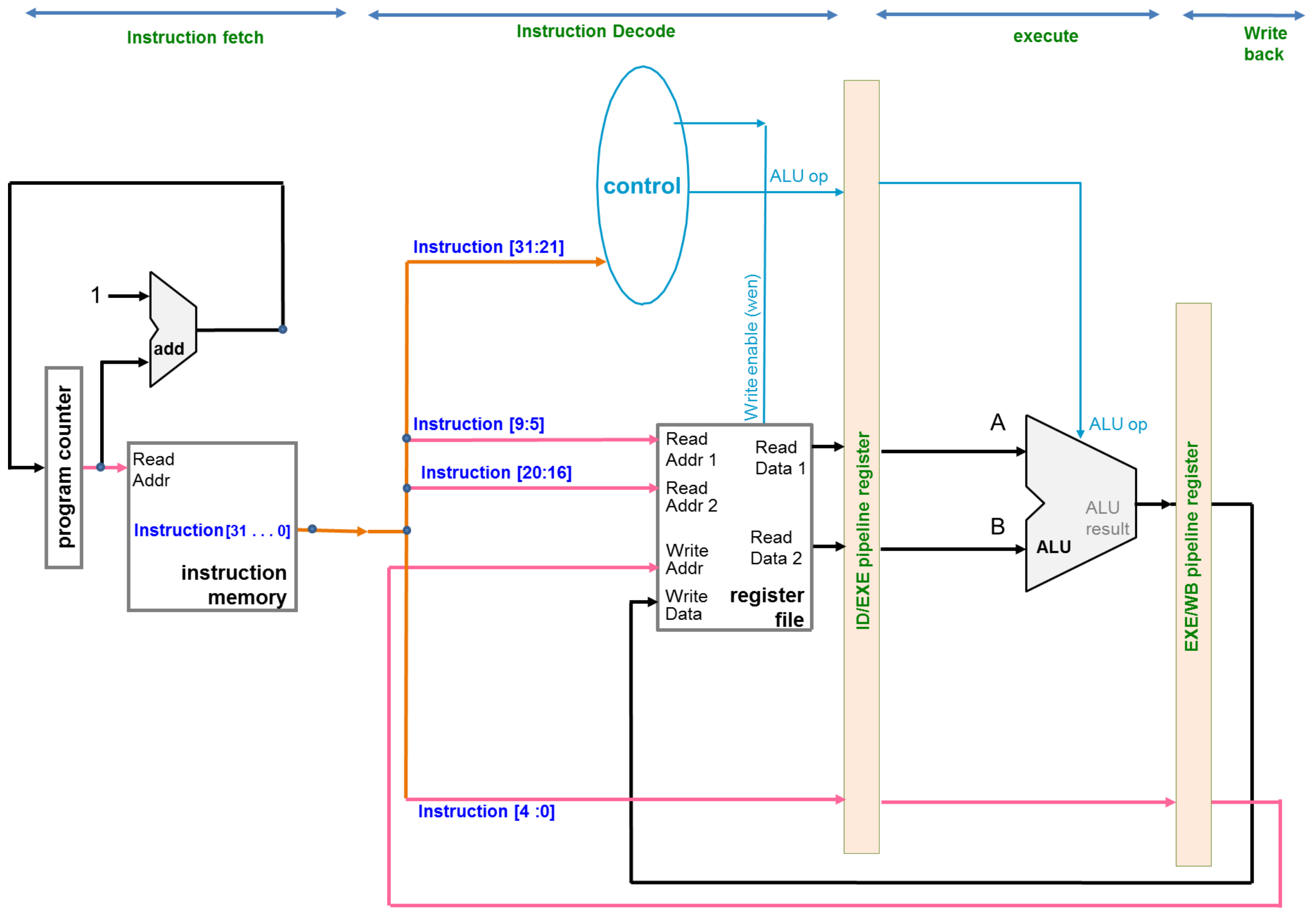


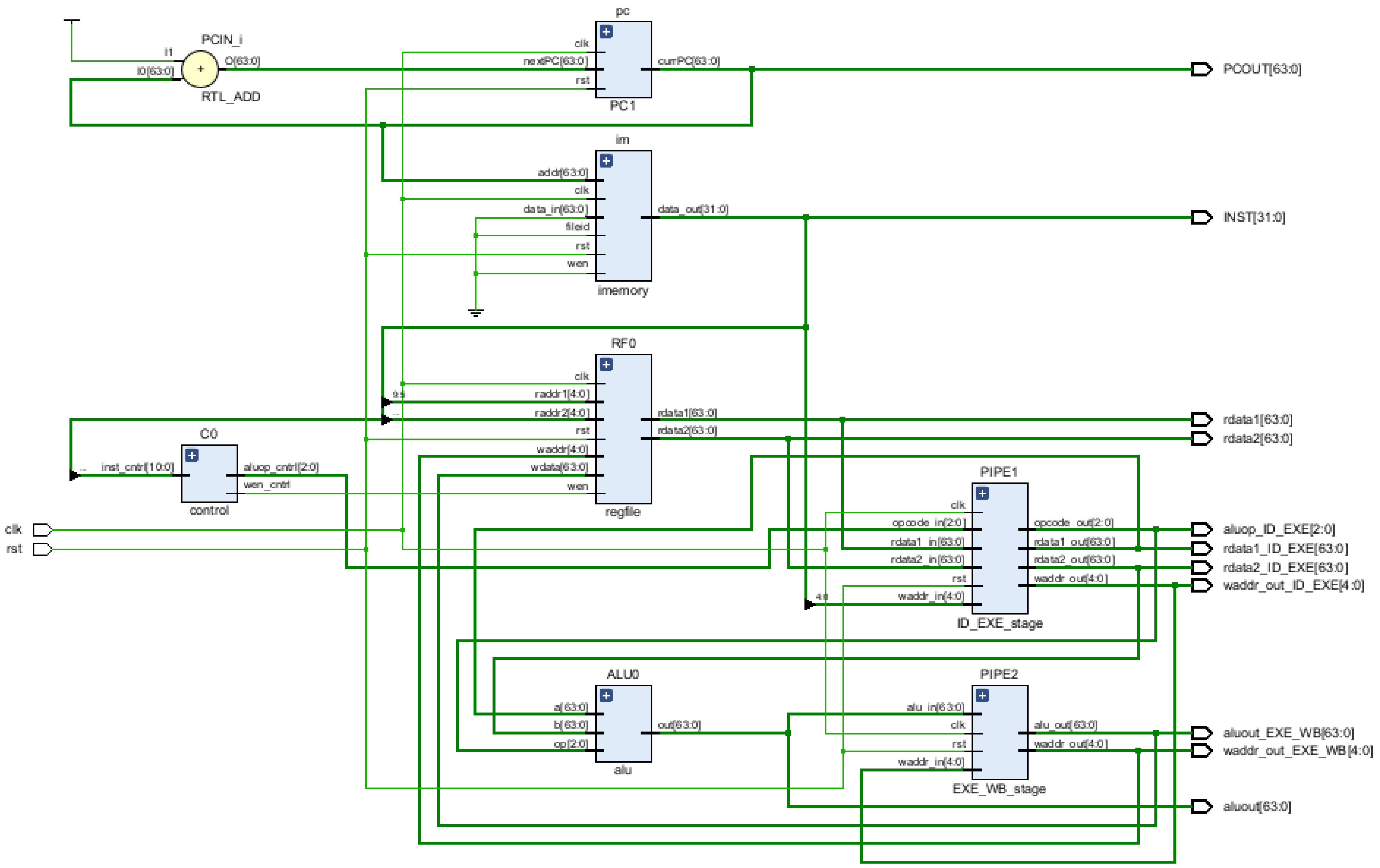
Clock cycle 3
First instruction in third stage
(execute)
Second instruction in second stage and
third instruction in first stage
(animation not shown)

Clock cycle 4
First instruction in fourth stage
(writeback)

Here the waddr is coming at the same clock cycle (4th cycle) as wdata to the regfile.
If waddr came early (2nd cycle) to regfile then the data will be written to a different location.
Hence we need to pipeline the waddr.

Please note that the ‘write enable’ is not taken to the pipeline (ID/EXE) in the given code basically because for all your instructions (R type) need ‘write enable =1’. Hence that was made as a constant for easier programming. But if you have instruction like store and branch where you don’t want data to be written to a ‘regfile’ you may have to take ‘write enable’ signal as a variable and need to take it through pipeline.





PCIN_i
I1
I0[63:0]
O[63:0]
RTL_ADD
pc
clk
nextPC[63:0]
rst
currPC[63:0]
PC1
PCOUT[63:0]
im
addr[63:0]
clk
data_in[63:0]
fileid
rst
wen
data_out[31:0]
imemory
INST[31:0]
RF0
clk
raddr1[4:0]
raddr2[4:0]
rst
waddr[4:0]
wdata[63:0]
wen
rdata1[63:0]
rdata2[63:0]
regfile
rdata1[63:0]
rdata2[63:0]
C0
inst_cntrl[10:0]
aluop_cntrl[2:0]
wen_cntrl
control
clk
rst
PIPE1
clk
opcode_in[2:0]
rdata1_in[63:0]
rdata2_in[63:0]
rst
waddr_in[4:0]
opcode_out[2:0]
rdata1_out[63:0]
rdata2_out[63:0]
waddr_out[4:0]
ID_EXE_stage
aluop_ID_EXE[2:0]
rdata1_ID_EXE[63:0]
rdata2_ID_EXE[63:0]
waddr_out_ID_EXE[4:0]
ALU0
a[63:0]
b[63:0]
op[2:0]
out[63:0]
alu
PIPE2
alu_in[63:0]
clk
rst
waddr_in[4:0]
alu_out[63:0]
waddr_out[4:0]
EXE_WB_stage
aluout_EXE_WB[63:0]
waddr_out_EXE_WB[4:0]
aluout[63:0]

Testing the R datapath (fig.1)

- Hardcoded in “regfile.v”

regdata[1] <=3;

regdata[2] <=2;Rest all the registers from 0-31 are initialized to zero

Inside memory.v, we have the text file having the instruction

64 bit PC address (in hex)	32 bit instruction from IMEM	meaning
0000000000000000	001F03FF	NOP
0000000000000001	00040023	ADD X3, X1, X4
0000000000000002	00220024	SUB X4, X1, X2
0000000000000003	00440027	AND X7, X1, X4
0000000000000004	006200E8	XOR X8, X7, X2
0000000000000005	001F03FF	NOP
0000000000000006	001F03FF	NOP

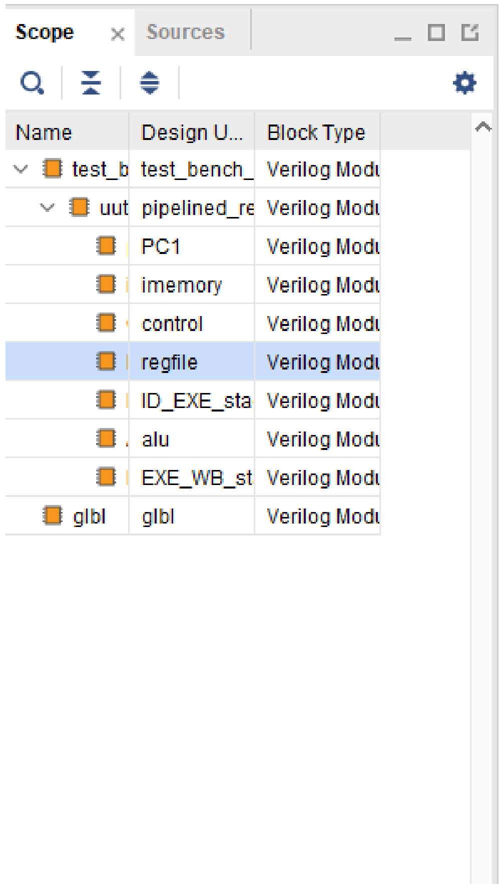
ADD X2, X1, X4= [X2] <= [X1] + [X4]

As reg[1] has value 3 and reg[4] has value 0, the result stored in reg[3] =3

The final state of registers

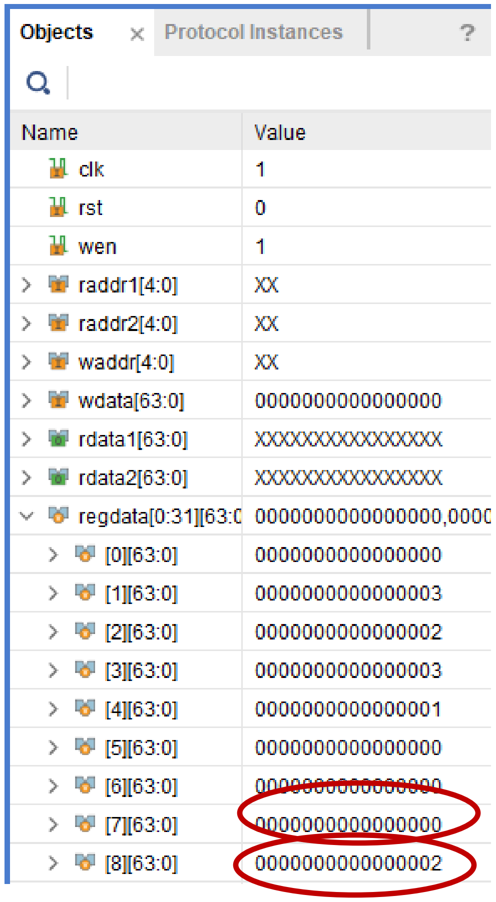
“imem_testw.mem”

64 bit PC address	32 bit instruction from IMEM	meaning
@0	001F03FF	NOP
@1	00040023	ADD X3, X1, X4
@2	00220024	SUB X4, X1, X2
@3	00440027	AND X7, X1, X4
@4	006200E8	XOR X8, X7, X2
@5	001F03FF	NOP
@6	001F03FF	NOP



Scope | Sources

Name	Design U...	Block Type
test_b	test_bench_	Verilog Modu
uut	pipelined_re	Verilog Modu
	PC1	Verilog Modu
	imemory	Verilog Modu
	control	Verilog Modu
	regfile	Verilog Modu
	ID_EXE_sta	Verilog Modu
	alu	Verilog Modu
	EXE_WB_st	Verilog Modu
glbl	glbl	Verilog Modu



Objects | Protocol Instances

Name	Value
clk	1
rst	0
wen	1
raddr1[4:0]	XX
raddr2[4:0]	XX
waddr[4:0]	XX
wdata[63:0]	0000000000000000
rdata1[63:0]	XXXXXXXXXXXXXXXX
rdata2[63:0]	XXXXXXXXXXXXXXXX
regdata[0:31][63:0	0000000000000000,0000
[0][63:0]	0000000000000000
[1][63:0]	0000000000000003
[2][63:0]	0000000000000002
[3][63:0]	0000000000000003
[4][63:0]	0000000000000001
[5][63:0]	0000000000000000
[6][63:0]	0000000000000000
[7][63:0]	0000000000000000
[8][63:0]	0000000000000002

Initialization – [X1]=3, [X2]=2

1. No change
2. [X3] =3+0=3
3. [X4]=3-2=1
4. [X7]=3(AND)1=1
5. [X8]=1(XOR)2=3
6. No change
7. No change

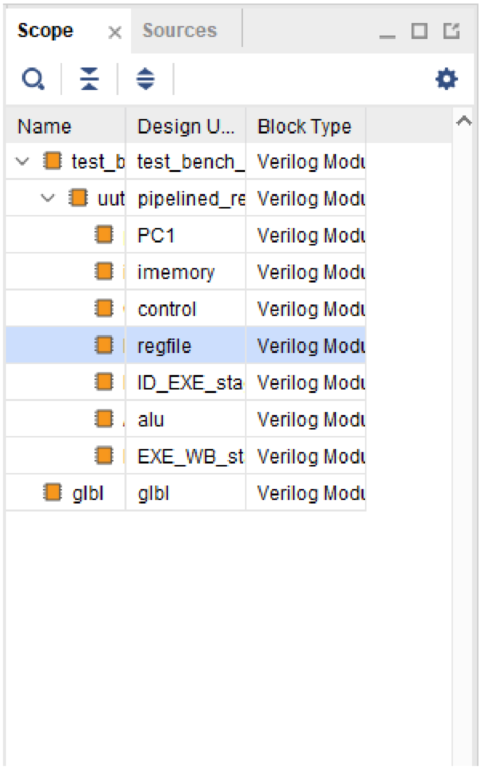
Are you getting the above result?

Most likely No. Why?
Can we correct the issues by adding NOPs?

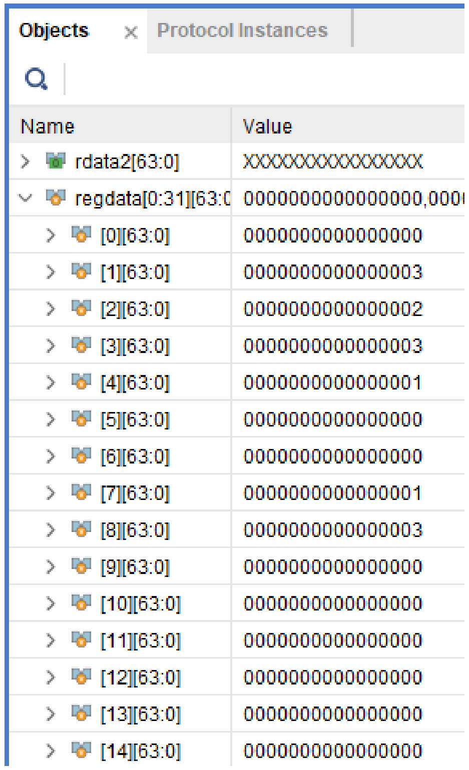
“imem_txtw.mem”

64 bit PC address (in hex)	32 bit instruction from IMEM	meaning
0000000000000000	001F03FF	NOP
0000000000000001	00040023	ADD X3, X1, X4
0000000000000002	00220024	SUB X4, X1, X2
0000000000000003	00440027	AND X7, X1, X4
0000000000000004	006200E8	XOR X8, X7, X2
0000000000000005	001F03FF	NOP
0000000000000006	001F03FF	NOP

64 bit PC address (in hex)	32 bit instruction from IMEM	meaning
0000000000000000	001F03FF	NOP
0000000000000001	00040023	ADD X3, X1, X4
0000000000000002	00220024	SUB X4, X1, X2
0000000000000003	001F03FF	NOP
0000000000000004	00440027	AND X7, X1, X4
0000000000000005	001F03FF	NOP
0000000000000006	006200E8	XOR X8, X7, X2
0000000000000007	001F03FF	NOP
0000000000000008	001F03FF	NOP

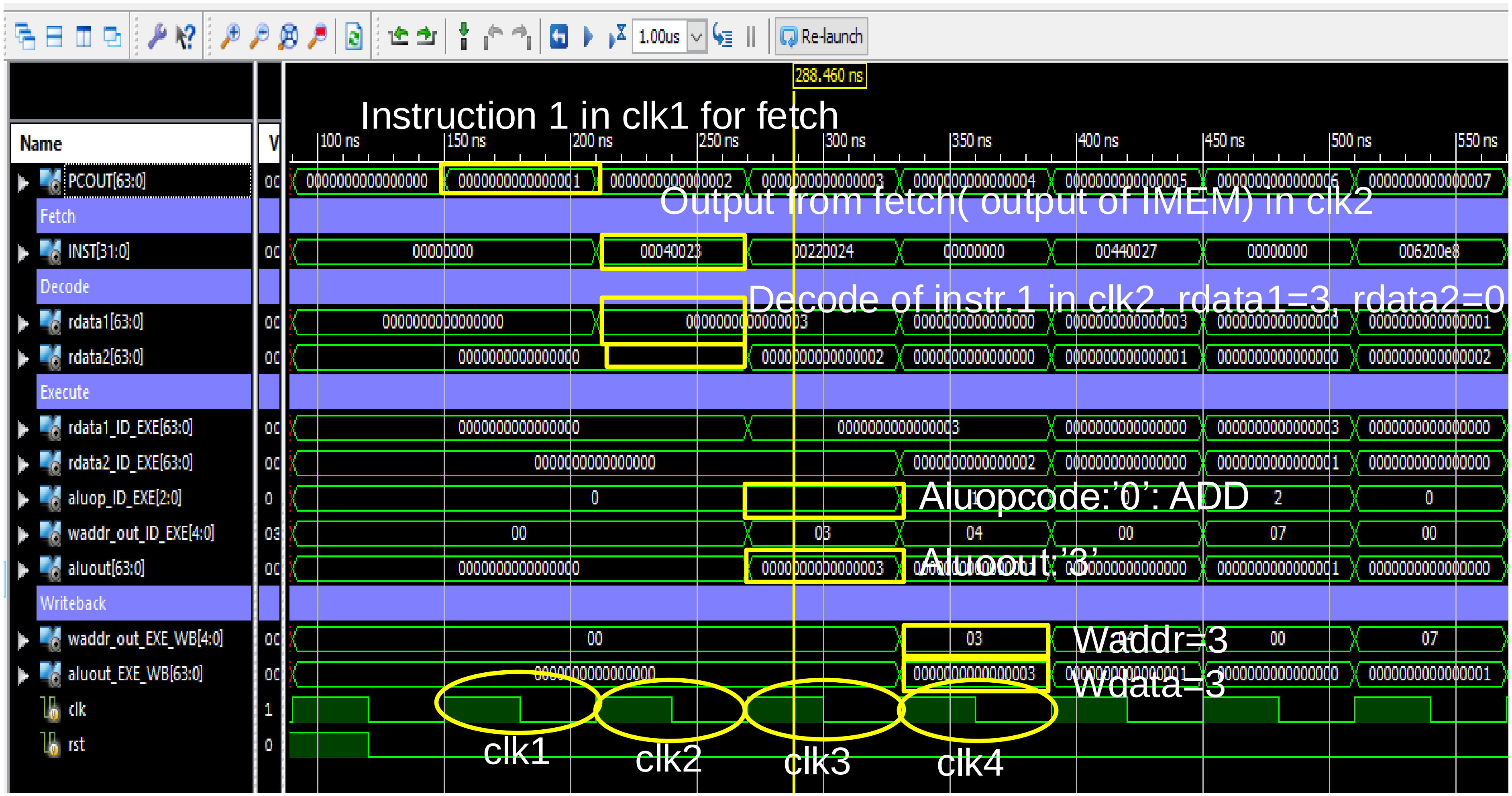


Name	Design U...	Block Type
test_b	test_bench_	Verilog Modu
uut	pipelined_re	Verilog Modu
	PC1	Verilog Modu
	imemory	Verilog Modu
	control	Verilog Modu
	regfile	Verilog Modu
	ID_EXE_sta	Verilog Modu
	alu	Verilog Modu
	EXE_WB_st	Verilog Modu
glbl	glbl	Verilog Modu



Name	Value
rdata2[63:0]	XXXXXXXXXXXXXXXX
regdata[0:31][63:0	0000000000000000,000
[0][63:0]	0000000000000000
[1][63:0]	0000000000000003
[2][63:0]	0000000000000002
[3][63:0]	0000000000000003
[4][63:0]	0000000000000001
[5][63:0]	0000000000000000
[6][63:0]	0000000000000000
[7][63:0]	0000000000000001
[8][63:0]	0000000000000003
[9][63:0]	0000000000000000
[10][63:0]	0000000000000000
[11][63:0]	0000000000000000
[12][63:0]	0000000000000000
[13][63:0]	0000000000000000
[14][63:0]	0000000000000000

The simulation time window can be explained as shown in next slide



Instruction 1 in clk1 for fetch

Output from fetch(output of IMEM) in clk2

Decode of instr.1 in clk2, rdata1=3, rdata2=0

Aluopcode:'0': ADD

Aluoout:'3'

Waddr=3

Wdata=3

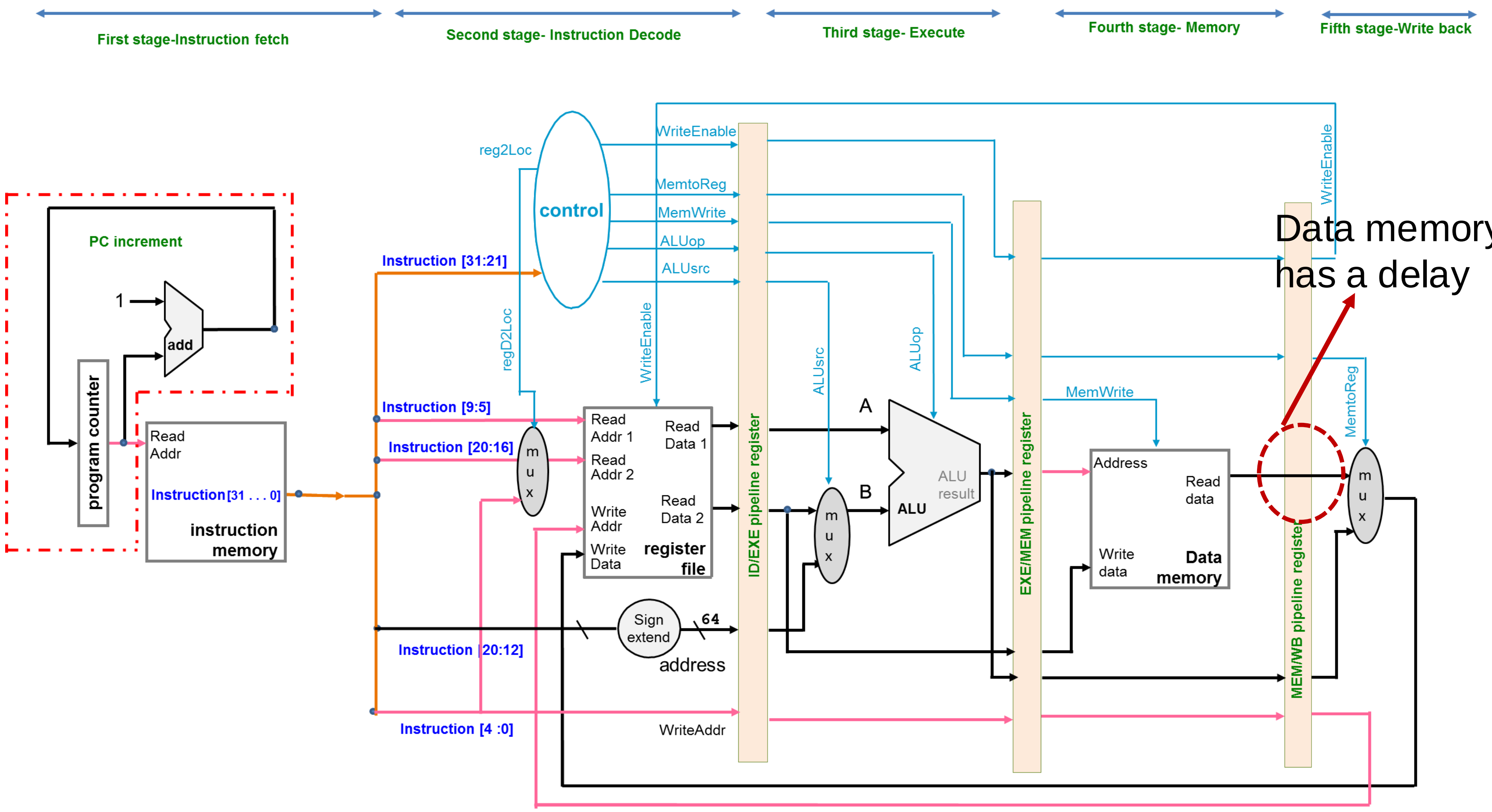
```
PCOUT[63:0]            0000000000000000  0000000000000001  0000000000000002  0000000000000003  0000000000000004  0000000000000005  0000000000000006  0000000000000007
Fetch
INST[31:0]             00000000  00040023  00220024  00000000  00440027  00000000  006200e8
Decode
rdata1[63:0]           0000000000000000  0000000000000003  0000000000000000  0000000000000003  0000000000000000  0000000000000001
rdata2[63:0]           0000000000000000  0000000000000002  0000000000000000  0000000000000001  0000000000000000  0000000000000002
Execute
rdata1_ID_EXE[63:0]    0000000000000000  0000000000000003  0000000000000000  0000000000000003  0000000000000000
rdata2_ID_EXE[63:0]    0000000000000000  0000000000000002  0000000000000000  0000000000000001  0000000000000000
aluop_ID_EXE[2:0]      0  1  1  2  0
waddr_out_ID_EXE[4:0]  00  03  04  00  07  00
aluout[63:0]           0000000000000000  0000000000000003  0000000000000000  0000000000000000  0000000000000001  0000000000000000
Writeback
waddr_out_EXE_WB[4:0]  00  03  04  00  07
aluout_EXE_WB[63:0]    0000000000000000  0000000000000003  0000000000000001  0000000000000000  0000000000000001
clk                    1
rst                    0
```

clk1 clk2 clk3 clk4

Wdata produced in clk 4 and written to regfile address 3

Four clk cycles required for the full operation of R-type
Four stage pipeline

5-stage pipelined architecture for R&D instructions



Both IMEM and DMEM provide the output after one clock cycle. This provides an imaginative pipeline after instruction fetch. It is also the reason why the output of DMEM is not passed through the last pipeline stage .

Points to be noted (R&D type- 5–stage pipeline)

- The Verilog file ‘pipelined_five_stage.v’ (provided) is the top module
  - consists of ‘PC.v’, ‘regfile.v’, ‘control.v’, ‘imemory.v’ , ‘alu.v’ , ‘dmemory.v’, ‘ID_EXEstage.v’, ‘EXE_MEMstage.v’, and ‘MEM_WBstage.v’.
  - The ‘ID_EXEstage.v’, ‘EXE_MEMstage.v’, and ‘MEM_WBstage.v’ are the pipeline registers between decode /execute, execute/datamemory and datamemory/writeback section respectively.
- You can note that we are able to operate register type of instructions (ADD, SUB, AND, XOR, ORR) along with (D type) LDUR and STUR. In order to initialize the registers in the register file, data from the memory can be utilized.

(Note that imemory and dmemory is clocked and hence we have a clock cycle delay for getting the output from memory.)

Testing 5-stage pipelined architecture for R&D instruction

instructions in the IMEM

PC address	32 bit instruction from IMEM	meaning
@0	001F03FF	NOP
@1	00A00045	LDUR X5, [X2,#0]
@2	00A02026	LDUR X6, [X1,#2]
@3	000600A5	ADD X5, X5, X6
@4	00C02045	STUR X5, [X2,#2]

This has dependencies and definitely need NOP to be inserted to remove that

Data in the DMEM

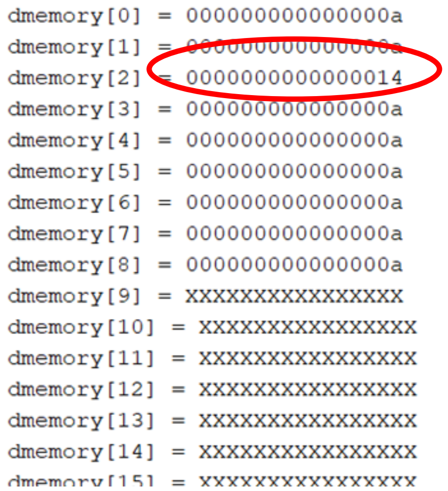
64 bit address (in hex)	64 bit data from DMEM
@0	000000000000000A
@1	000000000000000A
@2	000000000000000A
@3	000000000000000A
@4	000000000000000A
@5	000000000000000A
@6	000000000000000A
@7	000000000000000A

Initialization – all registers zero

1. No change
2. [X5] <=mem[0+0]=A
3. [X6] <=mem[0+2]=A
4. [X5]=A+A=14 (in HEX)
5. [X5]=>mem[0+2]=14

Finally X5 should have 14 as the value

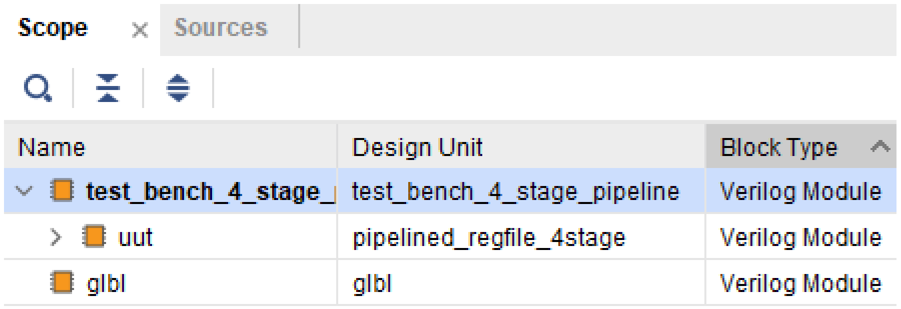
Using STUR instruction, we can write back to DMEM register.
Note that we are not writing back to the text file ‘dmem_test0.mem’.
Hence the update is only visible at the DMEM register as shown in the simulation window represented below.
X5 has the result of [A+A]=[14] (in hexadecimal).



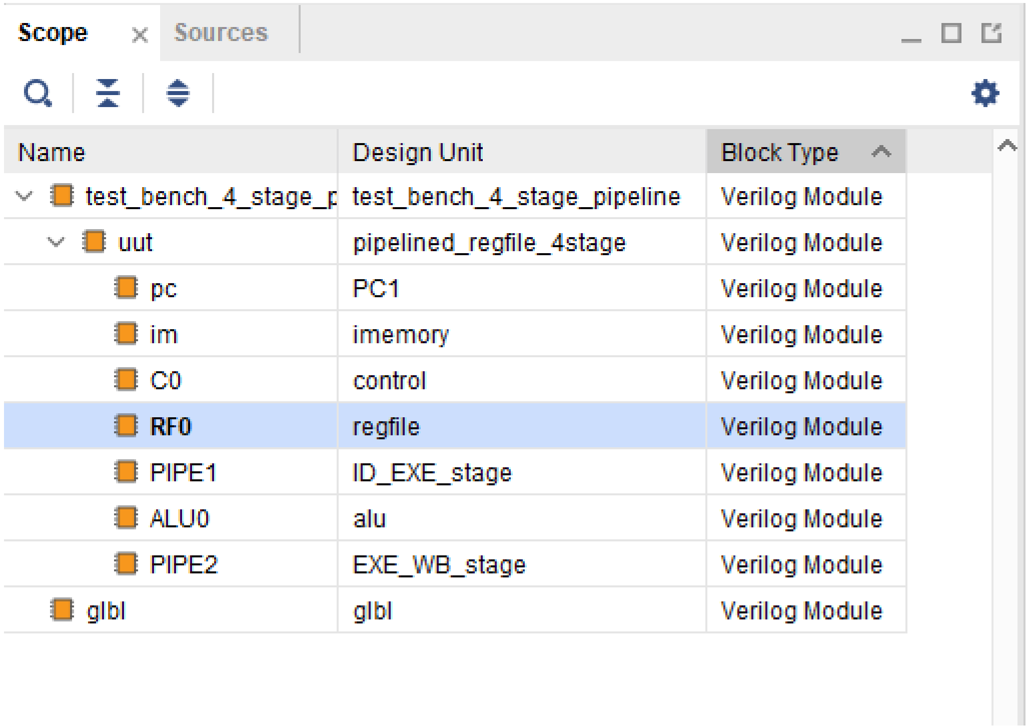
```
dmemory[0]  = 000000000000000a
dmemory[1]  = 000000000000000a
dmemory[2]  = 0000000000000014
dmemory[3]  = 000000000000000a
dmemory[4]  = 000000000000000a
dmemory[5]  = 000000000000000a
dmemory[6]  = 000000000000000a
dmemory[7]  = 000000000000000a
dmemory[8]  = 000000000000000a
dmemory[9]  = XXXXXXXXXXXXXXXX
dmemory[10]  = XXXXXXXXXXXXXXXX
dmemory[11]  = XXXXXXXXXXXXXXXX
dmemory[12]  = XXXXXXXXXXXXXXXX
dmemory[13]  = XXXXXXXXXXXXXXXX
dmemory[14]  = XXXXXXXXXXXXXXXX
dmemory[15]  = XXXXXXXXXXXXXXXX
```

Things to do

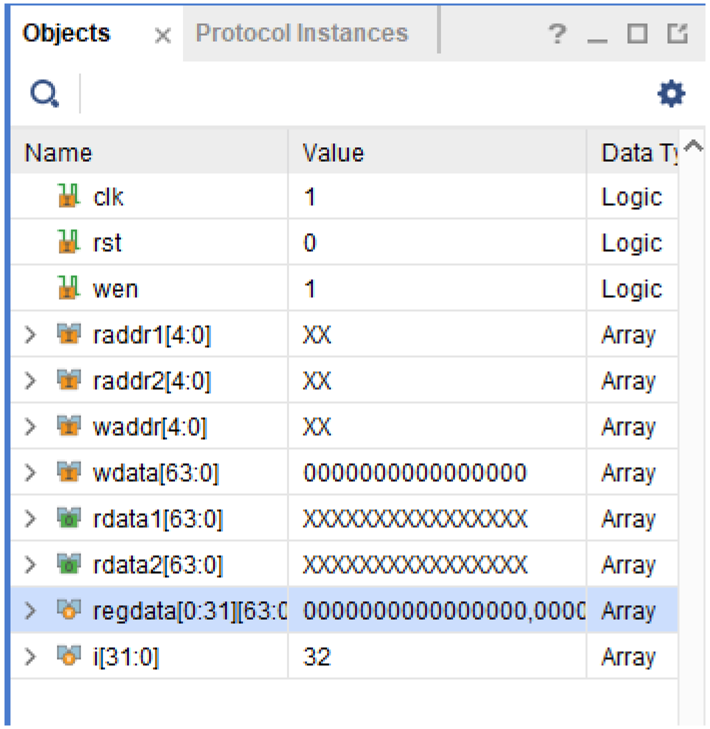
- Insert NOPs where ever necessary
- The final Dmemory should be like the above figure.
- You can synthesize the 5 stage architecture to fine the area and delay complexity



Name	Design Unit	Block Type
test_bench_4_stage_	test_bench_4_stage_pipeline	Verilog Module
uut	pipelined_regfile_4stage	Verilog Module
glbl	glbl	Verilog Module



Name	Design Unit	Block Type
test_bench_4_stage_p	test_bench_4_stage_pipeline	Verilog Module
uut	pipelined_regfile_4stage	Verilog Module
pc	PC1	Verilog Module
im	imemory	Verilog Module
C0	control	Verilog Module
RF0	regfile	Verilog Module
PIPE1	ID_EXE_stage	Verilog Module
ALU0	alu	Verilog Module
PIPE2	EXE_WB_stage	Verilog Module
glbl	glbl	Verilog Module



Name	Value	Data Ty
clk	1	Logic
rst	0	Logic
wen	1	Logic
raddr1[4:0]	XX	Array
raddr2[4:0]	XX	Array
waddr[4:0]	XX	Array
wdata[63:0]	0000000000000000	Array
rdata1[63:0]	XXXXXXXXXXXXXXXX	Array
rdata2[63:0]	XXXXXXXXXXXXXXXX	Array
regdata[0:31][63:0	0000000000000000,0000	Array
i[31:0]	32	Array