RTL Design of the 5 Stage Pipelined Architecture of RISC V Processor

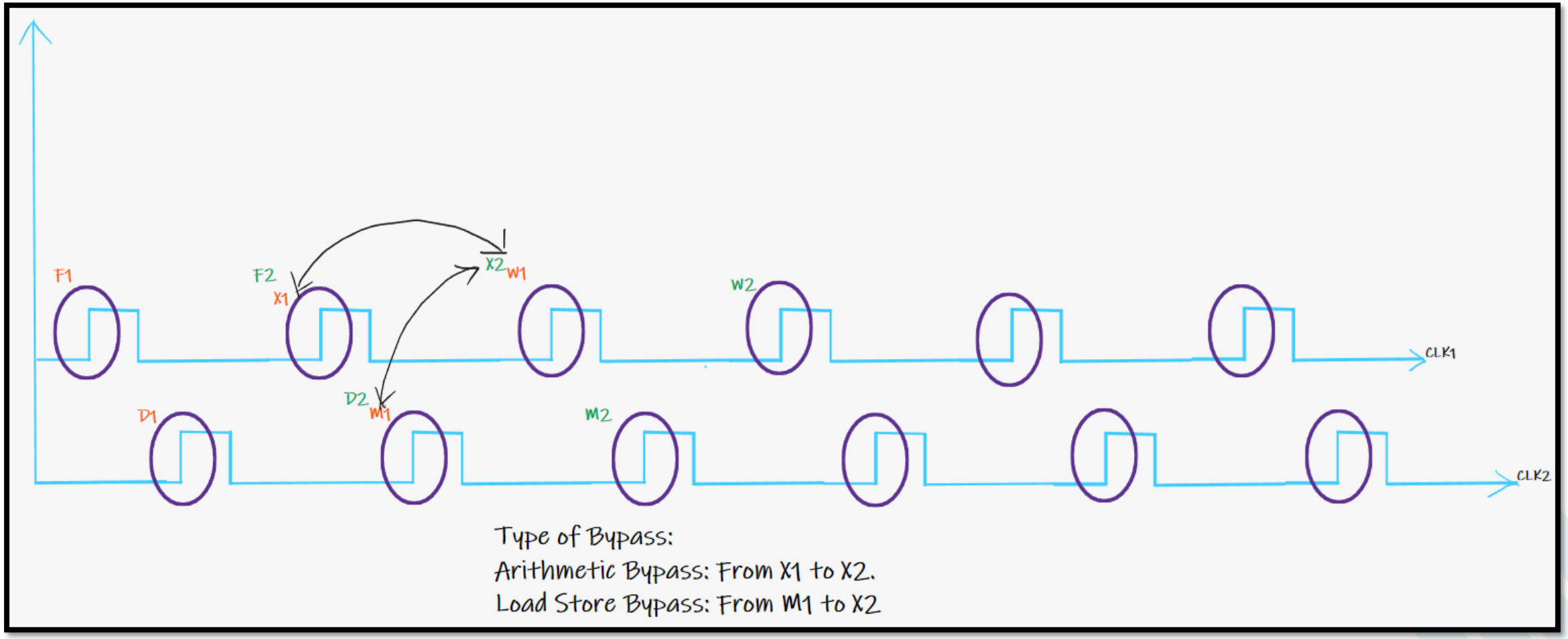
Group Number: 4

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

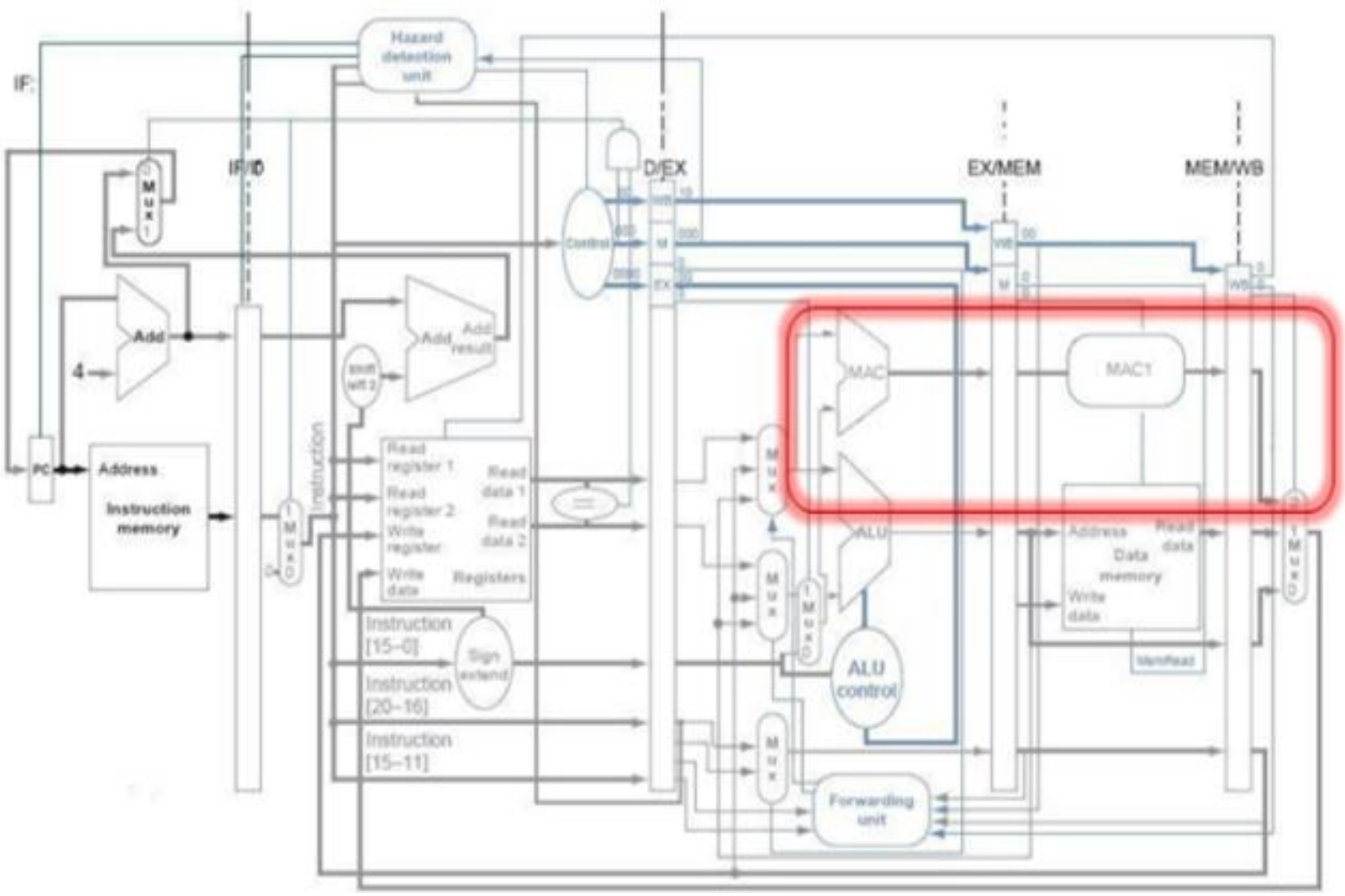
Group Members :

- Aakash Gupta – MT21150
- Sachi Garg– MT21206
- NK Vaishnav – MT21157

Explanation for usage of two clocks



RISC-V Architecture with MAC unit



Source: Design of MIPS Processor Supporting MAC with FPGA Munju Lee, Taewoo Han School of Electrical and Electronic Engineering College of Engineering

Pipeline Stages Used

Five stages, one step per stage

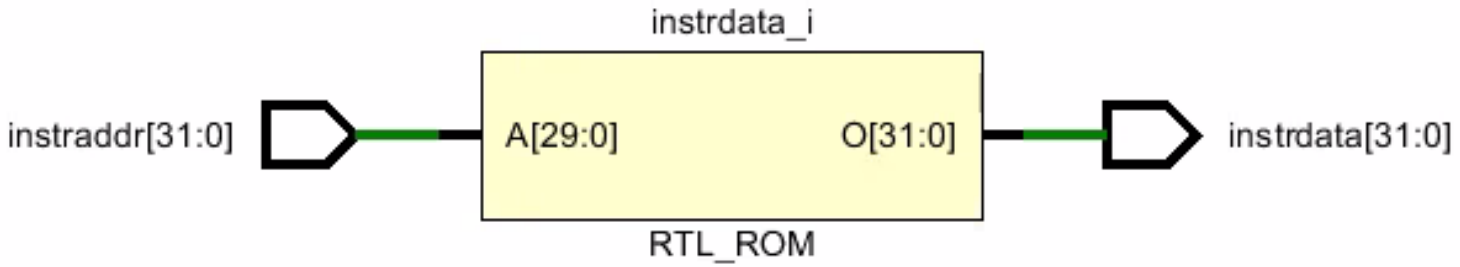
- *IF Stage* : Instruction fetch from memory
- *ID Stage* : Instruction decode & register read
- *EX Stage* : Execute operation or calculate address
- *MEM Stage* : Access memory operand
- *WB Stage* : Write result back to register

Instructions Set

INSTRUCTION		Function7				Function3		Opcode	
		31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
1	AND	00000	00	rs2	rs1	111	rd	01100	11
2	OR	00000	00	rs2	rs1	110	rd	01100	11
3	ADD	00000	00	rs2	rs1	000	rd	01100	11
4	SUB	01000	00	rs2	rs1	000	rd	01100	11
5	ADDI	imm[11:0]			rs1	000	rd	00100	11
6	BEQ	offset[12|10:5]		rs2	rs1	000	offset[4:1|11]	11000	11
7	LW	offset[11:0]			rs1	010	rd	00000	11
8	SW	offset[11:5]		rs2	rs1	010	offset[4:0]	01000	11
9	SLL	00000	00	rs2	rs1	001	rd	01100	11
10	SRA	00000	00	rs2	rs1	101	rd	01100	11
11*	MAC	00000	00	rs2	rs1	000	rd	11111	11

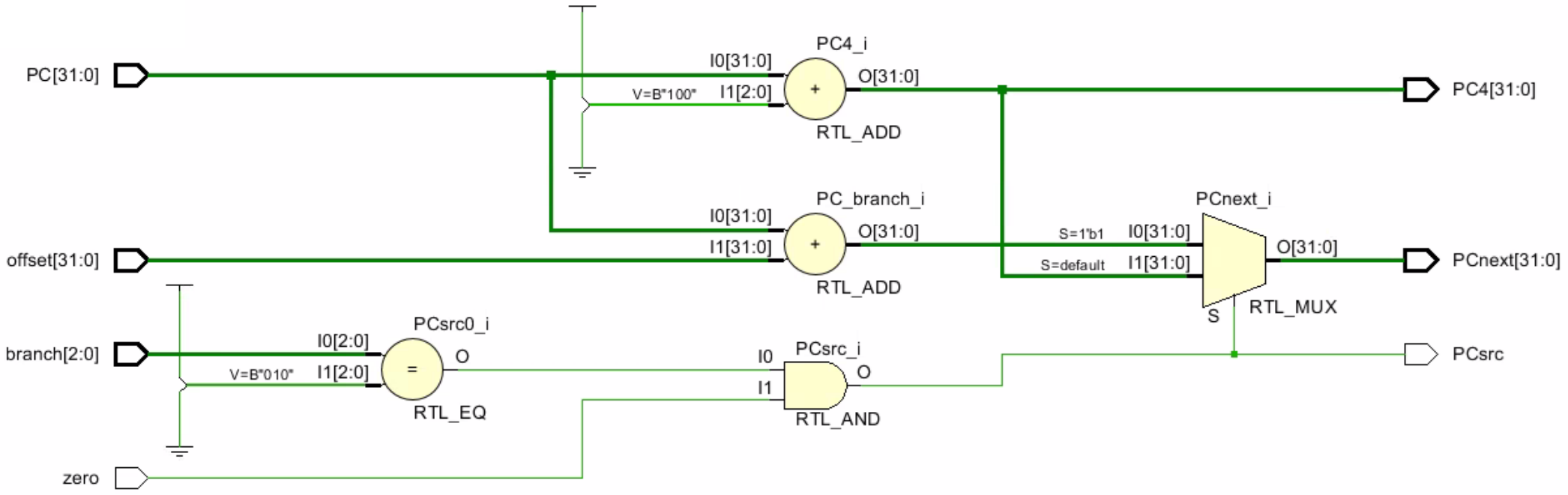
Instruction Memory

Instruction memory used in fetch stage to store the instructions to be executed. It has single read port which take 30 bit instruction address input, *instraddr*, and read data from that address onto the read data output, *instrdata*.

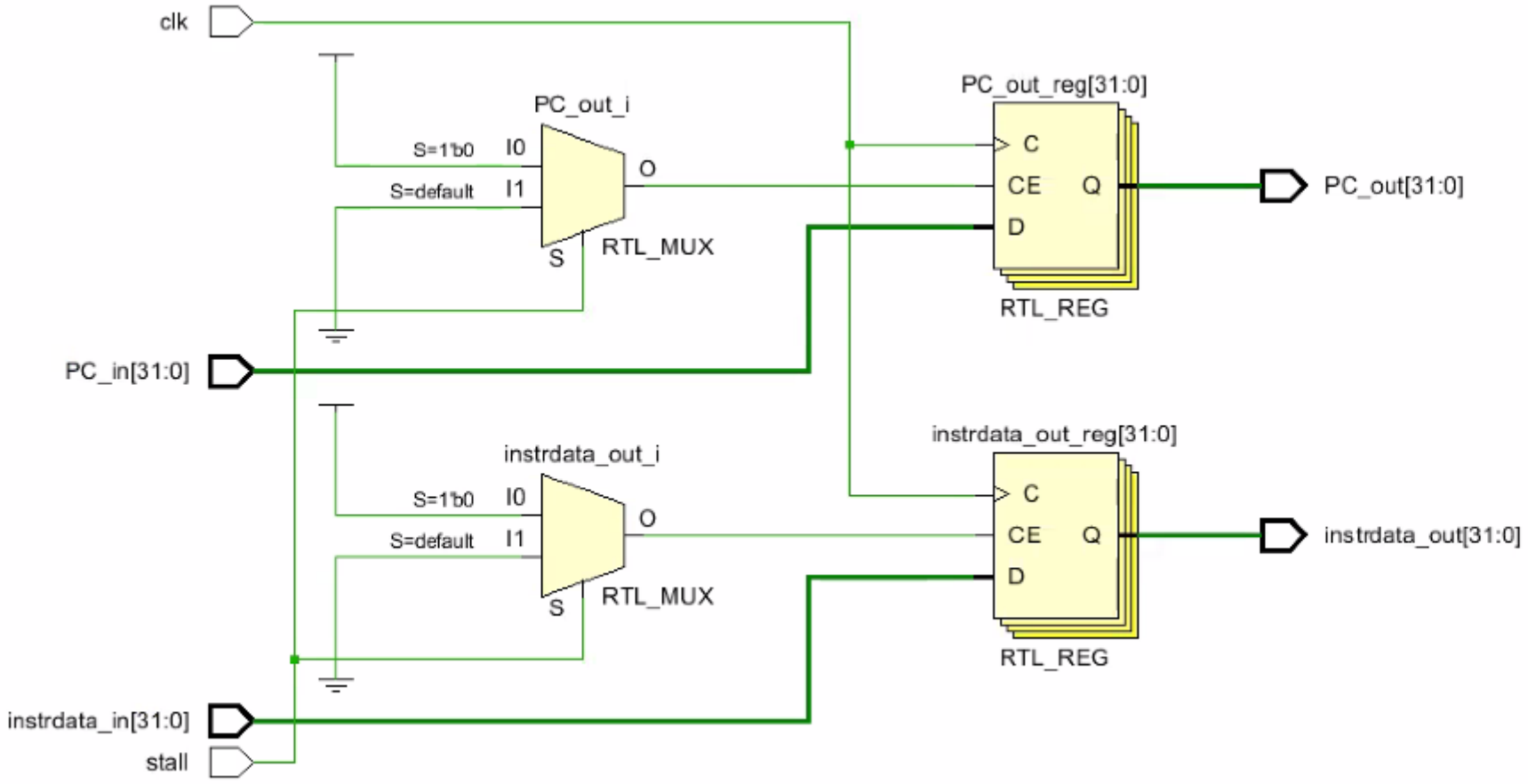


Program Counter

Program Counter is a register in computer processor that store the address of the instruction that is being executed at present. And as the instruction being fetched the program counter store the next address of instruction in the sequence

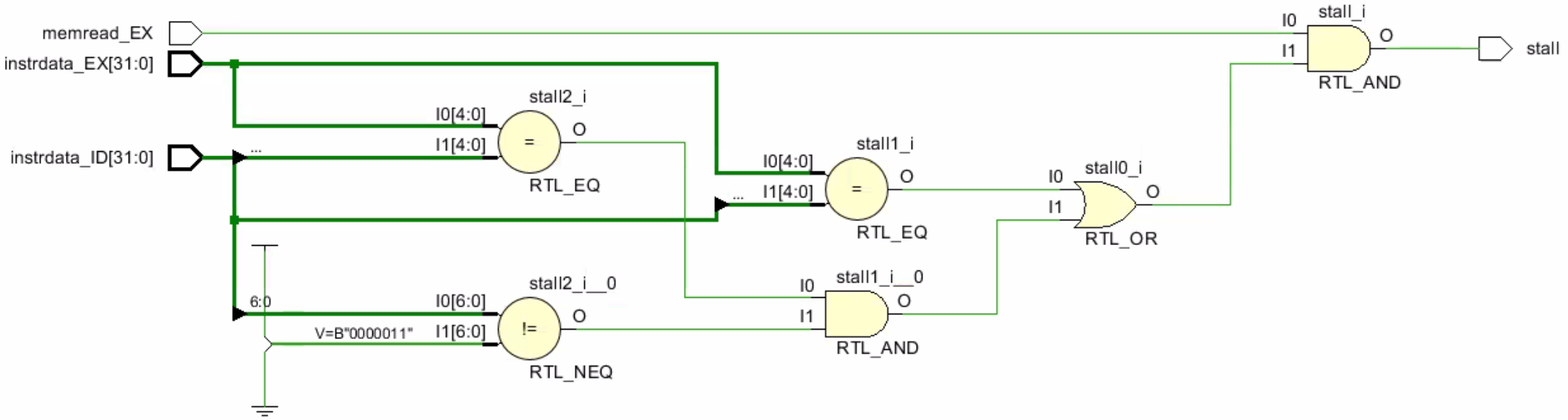


IF ID


clk
PC_out_i
S=1'b0
I0
S=default
I1
O
S
RTL_MUX
PC_out_reg[31:0]
C
CE
Q
D
RTL_REG
PC_out[31:0]
PC_in[31:0]
instrdata_out_i
S=1'b0
I0
S=default
I1
O
S
RTL_MUX
instrdata_out_reg[31:0]
C
CE
Q
D
RTL_REG
instrdata_out[31:0]
instrdata_in[31:0]
stall

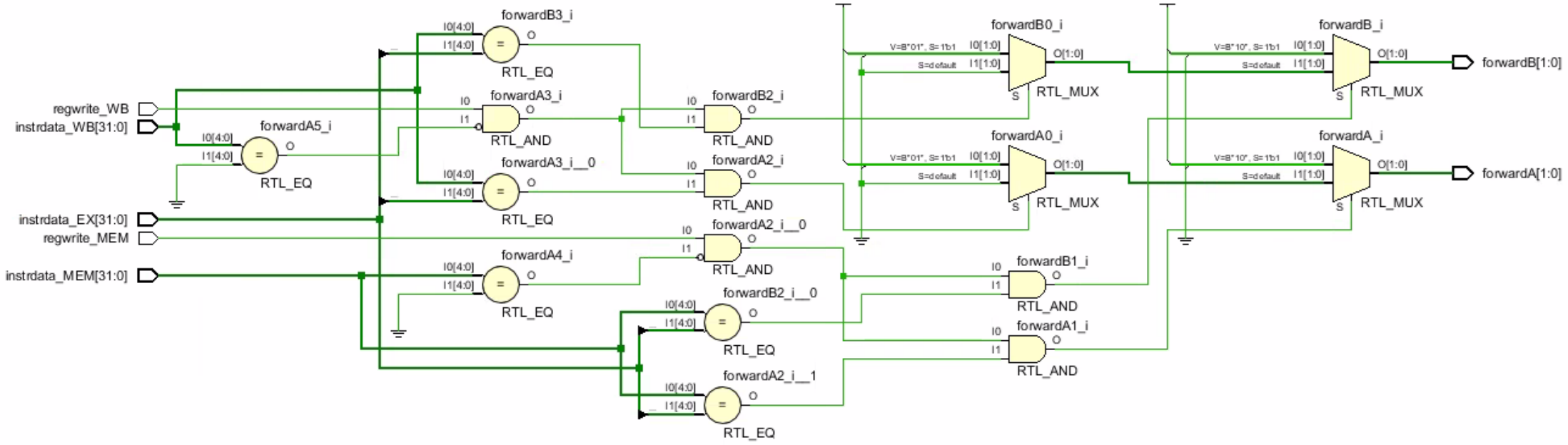
Stalling

Stalling is used to avoid hazards associated with the pipelining. But using the stall mechanism increases the delay of the processor therefore to avoid stalling as much as we can, we use bypassing/forwarding.

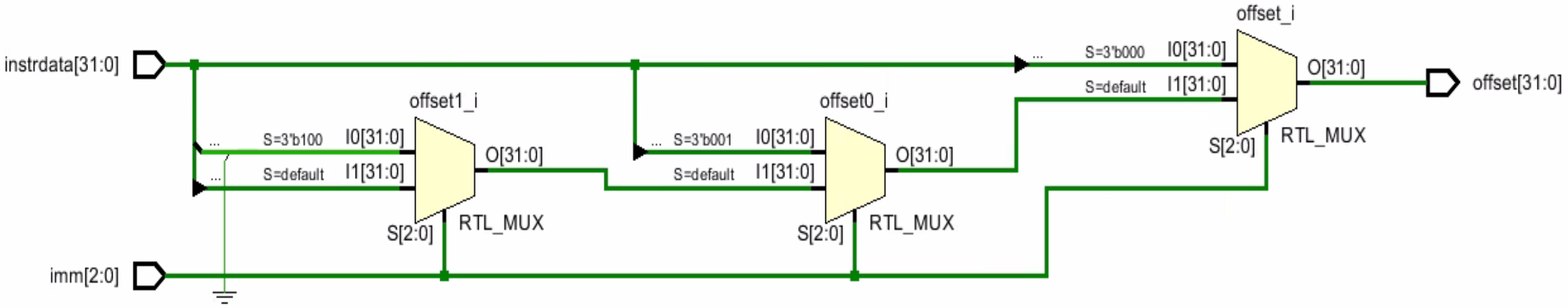


Forwarding

Forwarding is sufficient to solve RAW data hazards when the result is computed in the Execute stage of
an instruction, because its result can then be forwarded to the Execute stage of the next instruction.

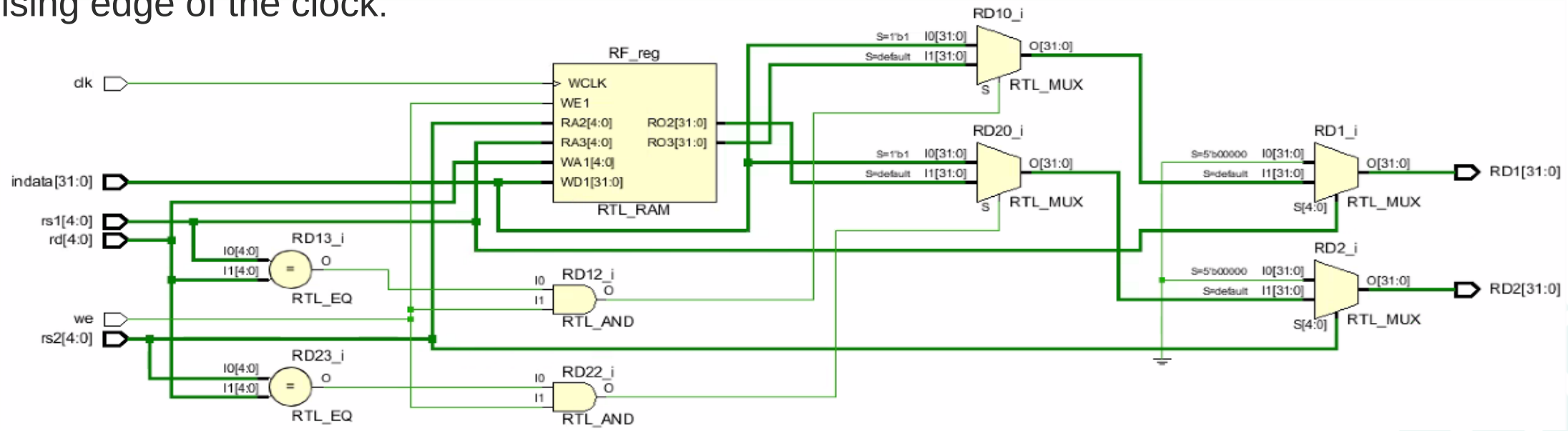


Offset Module



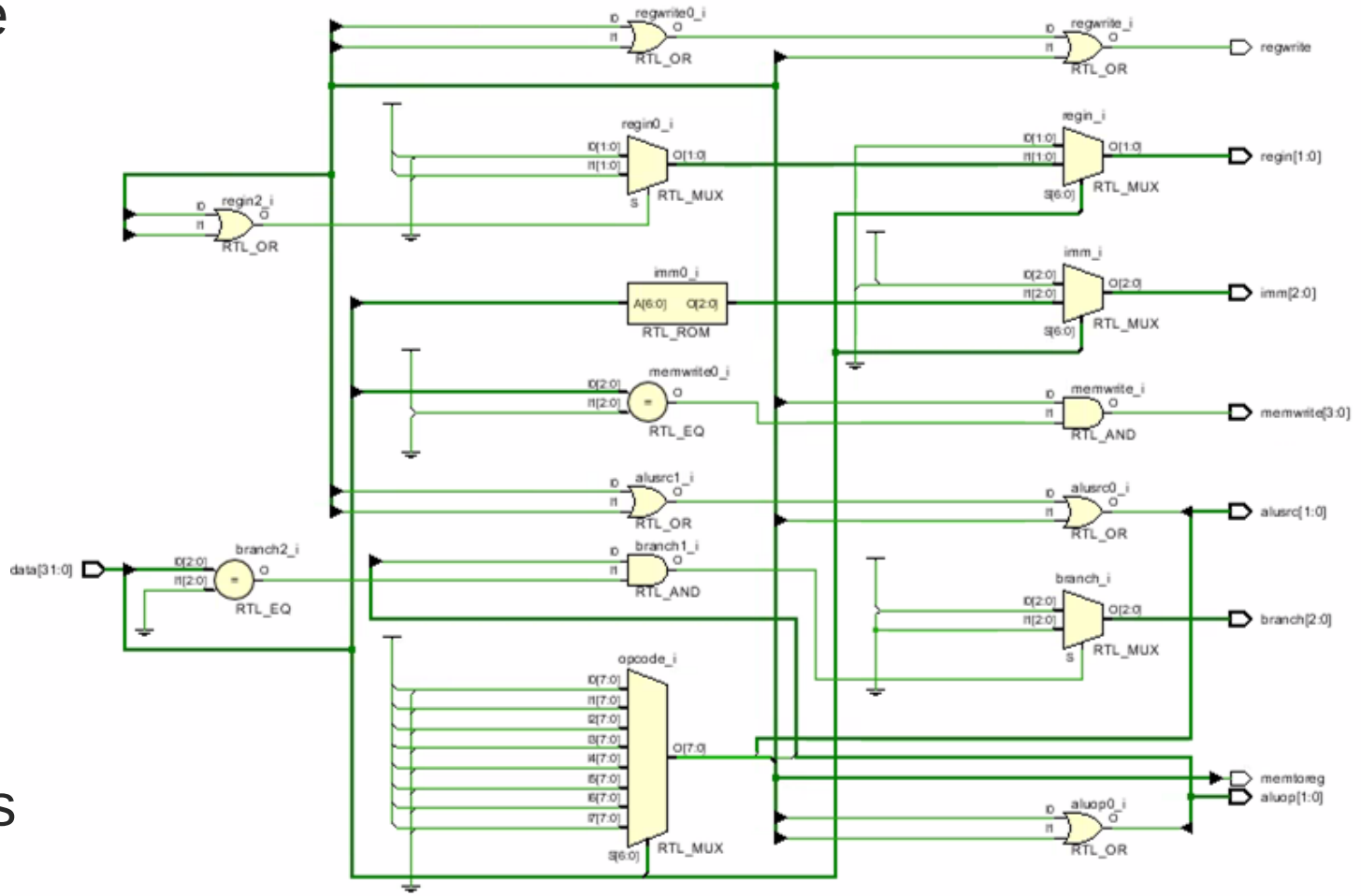
Register File

- The 32-element × 32-bit *register file* has two read ports and one write port.
- The read ports take 5-bit address inputs, *rs1* and *rs2*, each specifying one of $2^5 = 32$ registers as source
  operands.
- They read the 32-bit register values onto read data outputs *RD1* and *RD2*, respectively.
- The write port takes a 5-bit address input, *rd*; a 32-bit write data input, *indata*; a write enable input, *we*;
  and a clock. If the write enable is 1, the register file writes the data into the specified register on the rising edge of the clock.



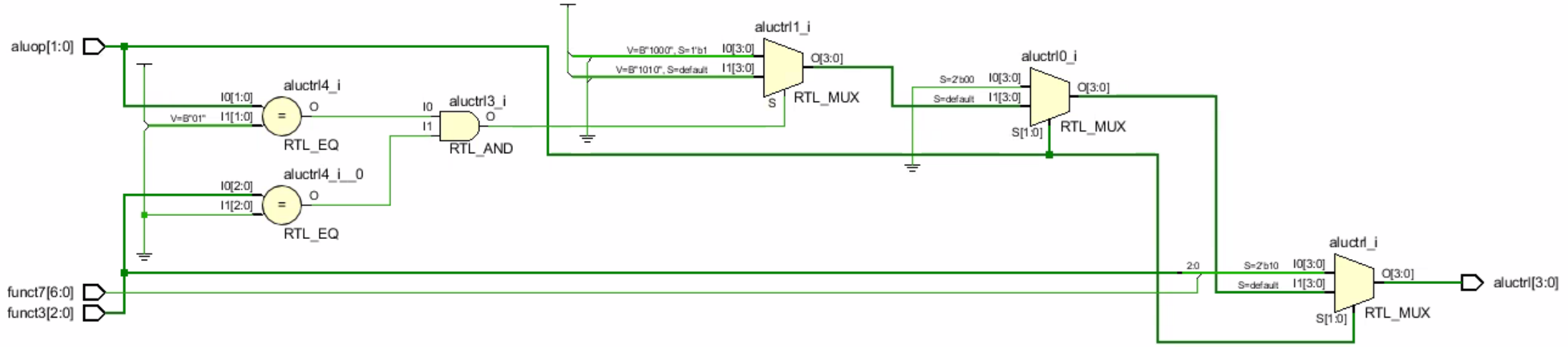
Control Unit

- Configuring control is one of the first step after decoding the instruction.
- To make CPU implementation simple, RISC-V defines bits [6:0] that needs to be used by control unit. Bit [6:0] defines type of operation and control units configures all multiplexers to set required data flow.



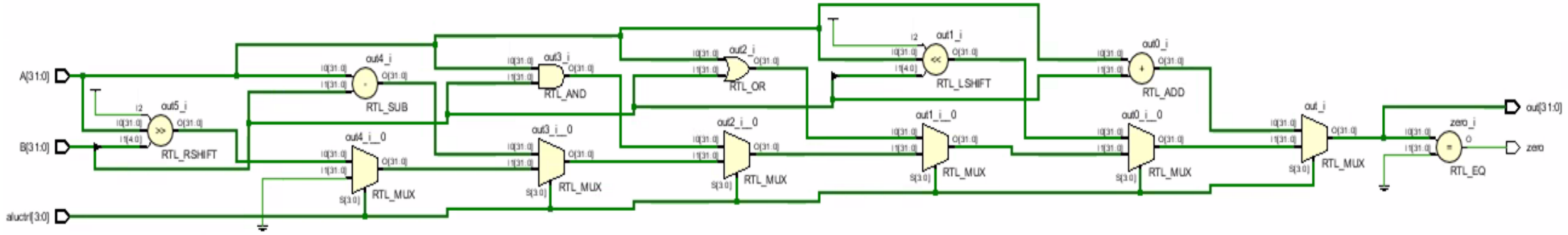
ALU Control

- ALU requires control unit different from actual control unit.
- ALU control unit keeps the architecture modular and makes it easy to include additional instruction types.

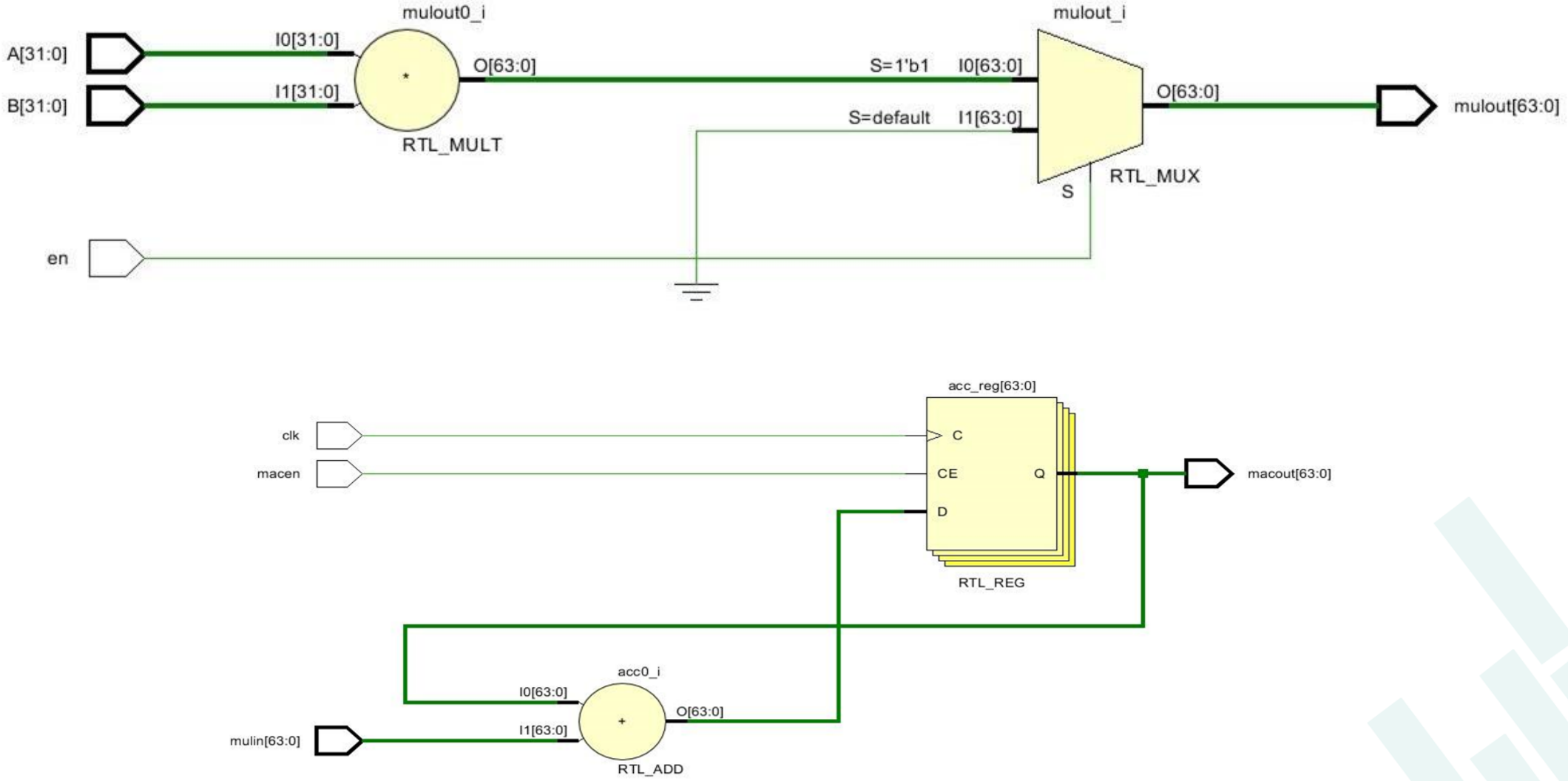


ALU

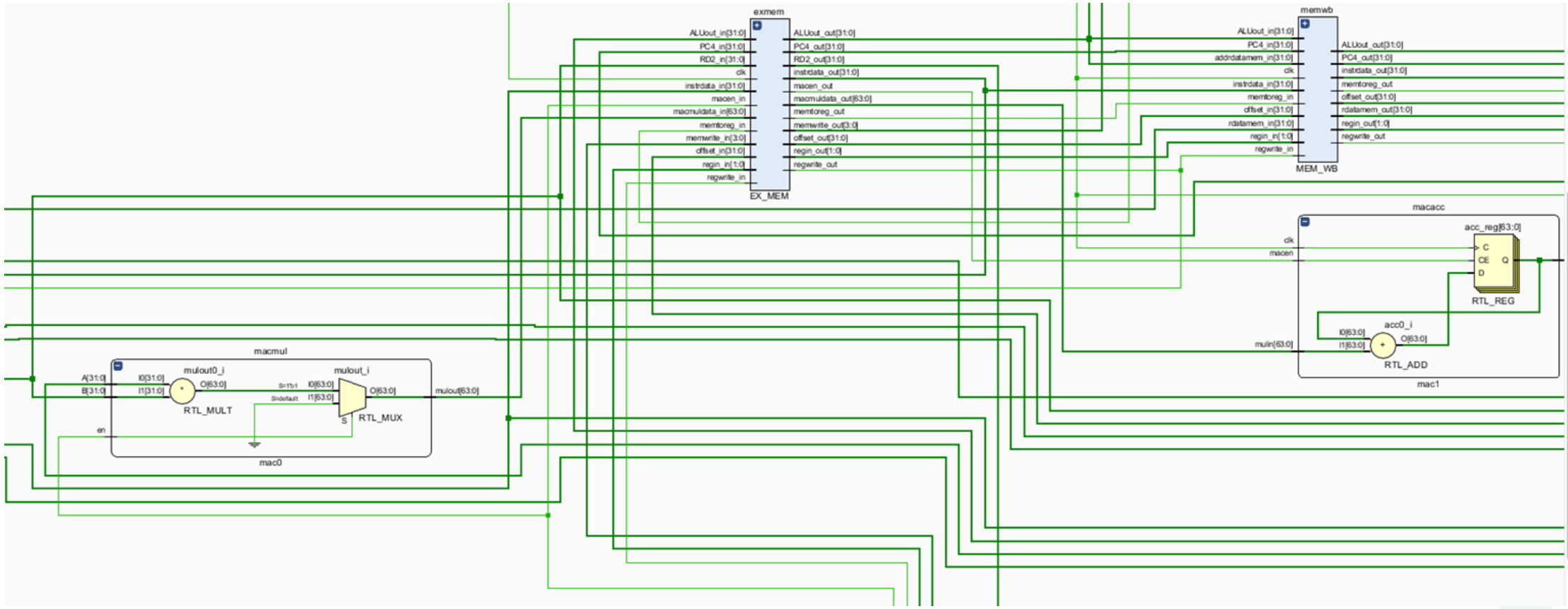
- ALU (Arithmetic logic unit) is a combinational circuit which takes two 32 bit data words
- A and B as input and perform logical and arithmetic operation on A and B and generate 32 bit output



MAC Unit

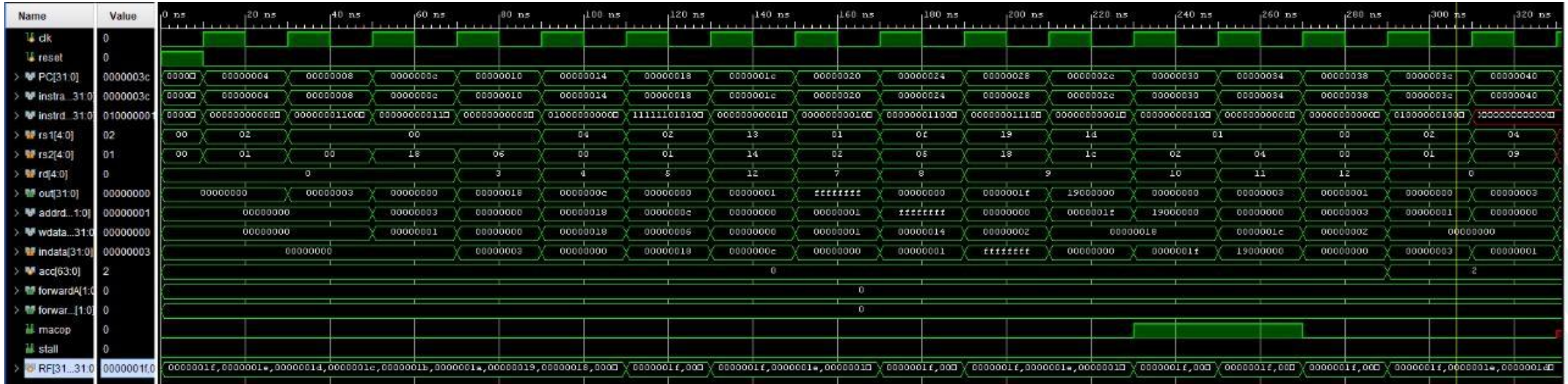

mulout0_i
A[31:0]
B[31:0]
I0[31:0]
I1[31:0]
*
O[63:0]
RTL_MULT
mulout_i
S=1'b1
I0[63:0]
S=default
I1[63:0]
O[63:0]
S
RTL_MUX
mulout[63:0]
en
acc_reg[63:0]
clk
macen
C
CE
D
Q
RTL_REG
macout[63:0]
acc0_i
I0[63:0]
I1[63:0]
+
O[63:0]
mulin[63:0]
RTL_ADD

MAC unit in pipelined RISC-V

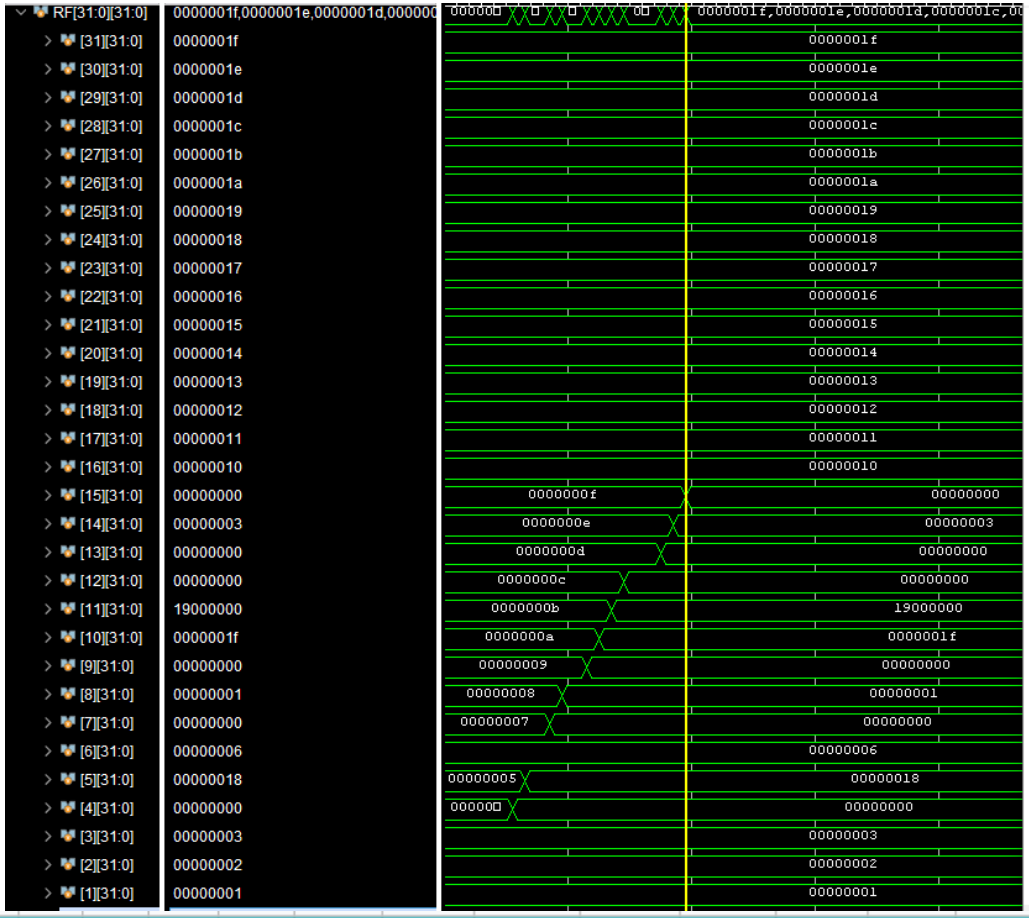


Normal Testcase

	Instruction Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00000000000010001000000110110011	Add R3,R2,R1	F	D	X	M	W															
00000000000000000010001000000011	lw R4,0(R0)		F	D	X	M	W														
00000001100000000000001010010011	addi R5,24(R0)			F	D	X	M	W													
00000000011000000010011000100011	sw R6,12(R0)				F	D	X	M	W												
00000000000000010001000111000011	lw R7,0(R4)					F	D	X	M	W											
01000000000100010000010000110011	sub R8,R2,R1						F	D	X	M	W										
11111101010010011000010011100011	beq R19,R20							F	D	X	M	W									
00000000001000001111010010110011	and R9,R1,R2								F	D	X	M	W								
00000000010101111110010100110011	or R10,R15,R5									F	D	X	M	W							
00000001100011001001010110110011	sll R11,R25,R24										F	D	X	M	W						
00000001110011101101011000110011	sra R12,R29,R28											F	D	X	M	W					
00000000001000001000000001111111	mac R2,R1												F	D	X	M	W				
00000000010000001000000001111111	mac R4,R1													F	D	X	M	W			
00000000000000000010011010000011	lw R13,0(R0)														F	D	X	M	W		
00000000000100010000011100110011	add r14,r2,r1															F	D	X	M	W	
01000000100100100000011110110011	sub r15,r4,r9																F	D	X	M	W



Normal Test Case modeling verification with interpreter



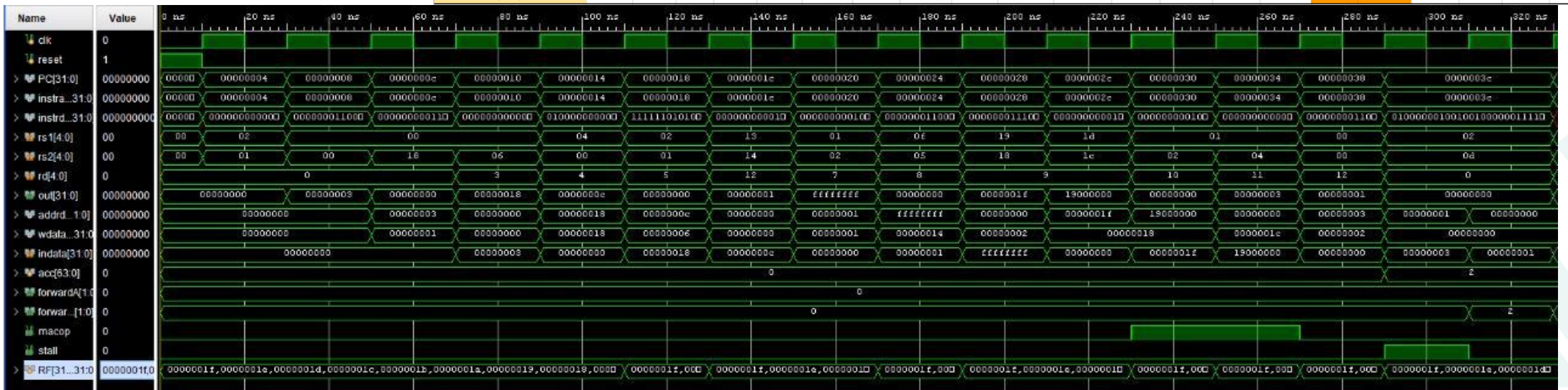

```
Register Expected Results
## expect[0] = 0x00000000
## expect[1] = 0x00000001
## expect[2] = 0x00000002
## expect[3] = 0x00000003
## expect[4] = 0x00000000
## expect[5] = 0x00000018
## expect[6] = 0x00000006
## expect[7] = 0x00000000
## expect[8] = 0x00000001
## expect[9] = 0x00000000
## expect[10] = 0x0000001f
## expect[11] = 0x19000000
## expect[12] = 0x00000000
## expect[13] = 0x00000000
## expect[14] = 0x00000003
## expect[15] = 0x00000000
## expect[16] = 0x00000010
## expect[17] = 0x00000011
## expect[18] = 0x00000012
## expect[19] = 0x00000013
## expect[20] = 0x00000014
## expect[21] = 0x00000015
## expect[22] = 0x00000016
## expect[23] = 0x00000017
## expect[24] = 0x00000018
## expect[25] = 0x00000019
## expect[26] = 0x0000001a
## expect[27] = 0x0000001b
## expect[28] = 0x0000001c
## expect[29] = 0x0000001d
## expect[30] = 0x0000001e
## expect[31] = 0x0000001f
```

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Instruction Fetch	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15				
Instruction Decode		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15			
Execute			I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15		
Memory access				I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	
Write Back					I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15

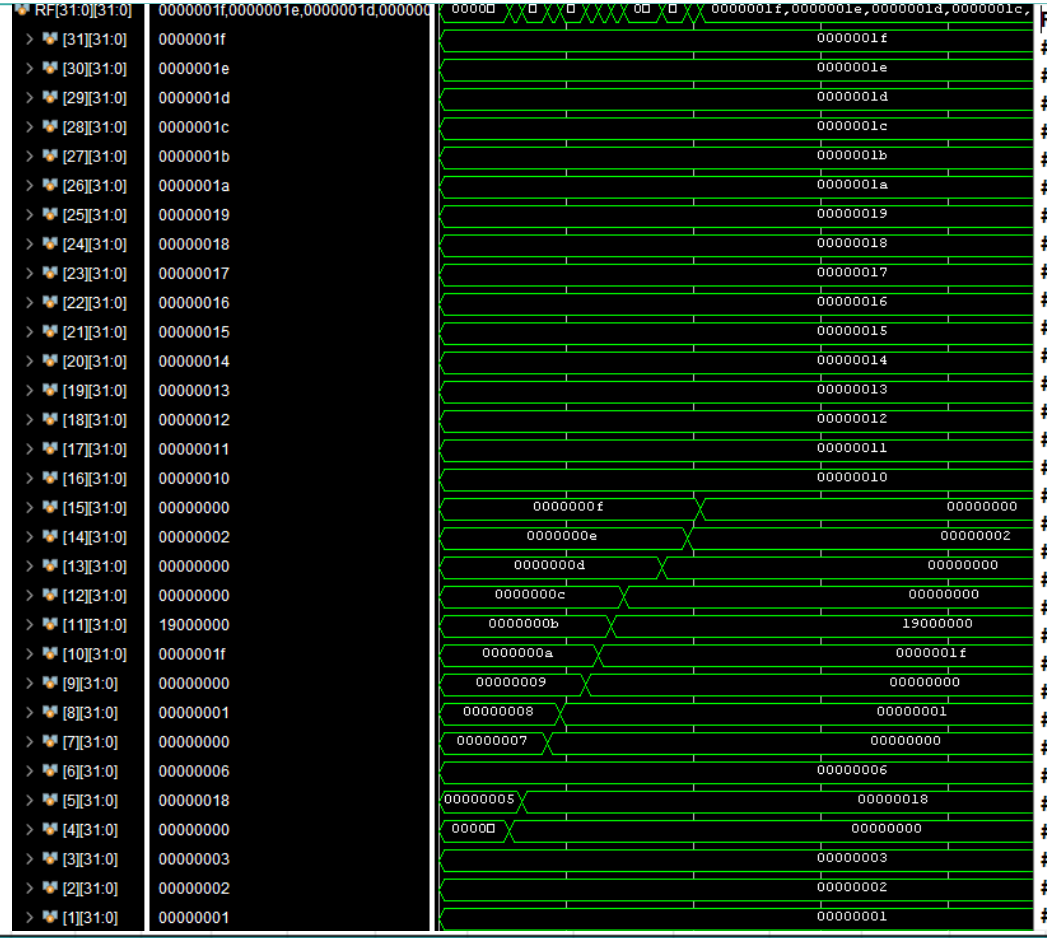
Stalling

```
00000000000010001000000110110011
00000000000000000010001000000011
00000001100000000000001010010011
00000000011000000010011000100011
00000000000000010001000111000011
01000000000100010000010000110011
11111101010010011000010011100011
00000000001000001111010010110011
00000000010101111111001010011011
00000001100011001001010110110011
00000001110011101101011000110011
00000000001000001000000001111111
00000000010000001000000001111111
00000000000000000010011010000011
00000000110100010000011100110011
01000000100100100000011110110011
```

Instruction Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Add R3,R2,R1	F	D	X	M	W																
lw R4,0(R0)		F	D	X	M	W															
addi R5,24(R0)			F	D	X	M	W														
sw R6,12(R0)				F	D	X	M	W													
lw R7,0(R4)					F	D	X	M	W												
sub R8,R2,R1						F	D	X	M	W											
beq R19,R20							F	D	X	M	W										
and R9,R1,R2								F	D	X	M	W									
or R10,R15,R5									F	D	X	M	W								
sll R11,R25,R24										F	D	X	M	W							
sra R12,R29,R28											F	D	X	M	W						
mac R2,R1												F	D	X	M	W					
mac R4,R1													F	D	X	M	W				
lw R13,0(R0)														F	D	X	M	W			
add r14,r2,r13															F	D	D	X	M	W	
sub r15,r4,r9																F	F	D	X	M	W



Stalling testcase verification with interpreter



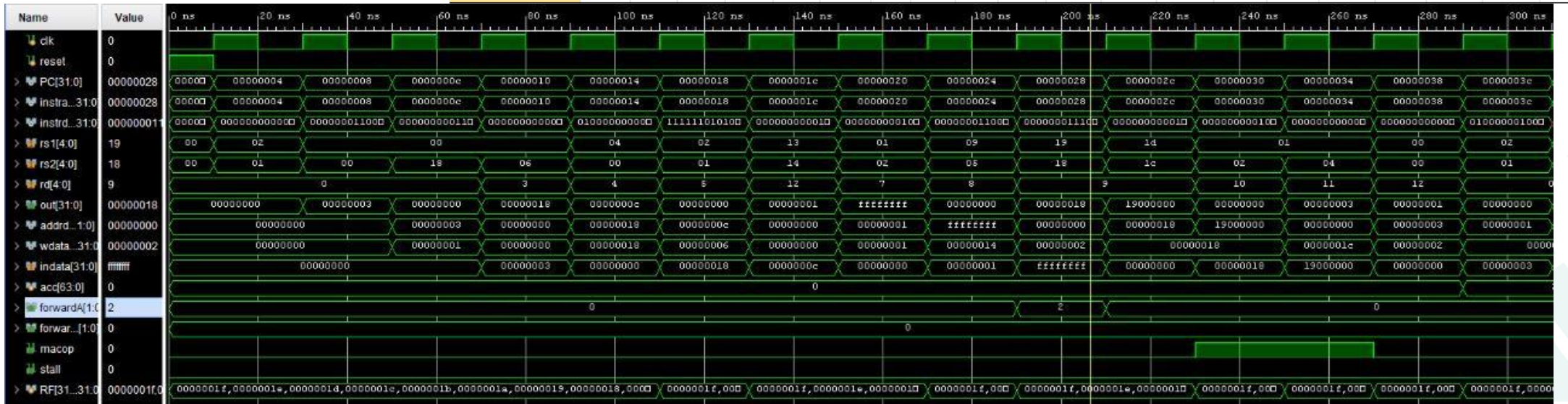

```
Register Expected Results
## expect[0] = 0x00000000
## expect[1] = 0x00000001
## expect[2] = 0x00000002
## expect[3] = 0x00000003
## expect[4] = 0x00000000
## expect[5] = 0x00000018
## expect[6] = 0x00000006
## expect[7] = 0x00000000
## expect[8] = 0x00000001
## expect[9] = 0x00000000
## expect[10] = 0x0000001f
## expect[11] = 0x19000000
## expect[12] = 0x00000000
## expect[13] = 0x00000000
## expect[14] = 0x00000002
## expect[15] = 0x00000000
## expect[16] = 0x00000010
## expect[17] = 0x00000011
## expect[18] = 0x00000012
## expect[19] = 0x00000013
## expect[20] = 0x00000014
## expect[21] = 0x00000015
## expect[22] = 0x00000016
## expect[23] = 0x00000017
## expect[24] = 0x00000018
## expect[25] = 0x00000019
## expect[26] = 0x0000001a
## expect[27] = 0x0000001b
## expect[28] = 0x0000001c
## expect[29] = 0x0000001d
## expect[30] = 0x0000001e
## expect[31] = 0x0000001f
```

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Instruction Fetch	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	X	I15				
Instruction Decode		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15			
Execute			I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15		
Memory access				I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15	
Write Back					I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15

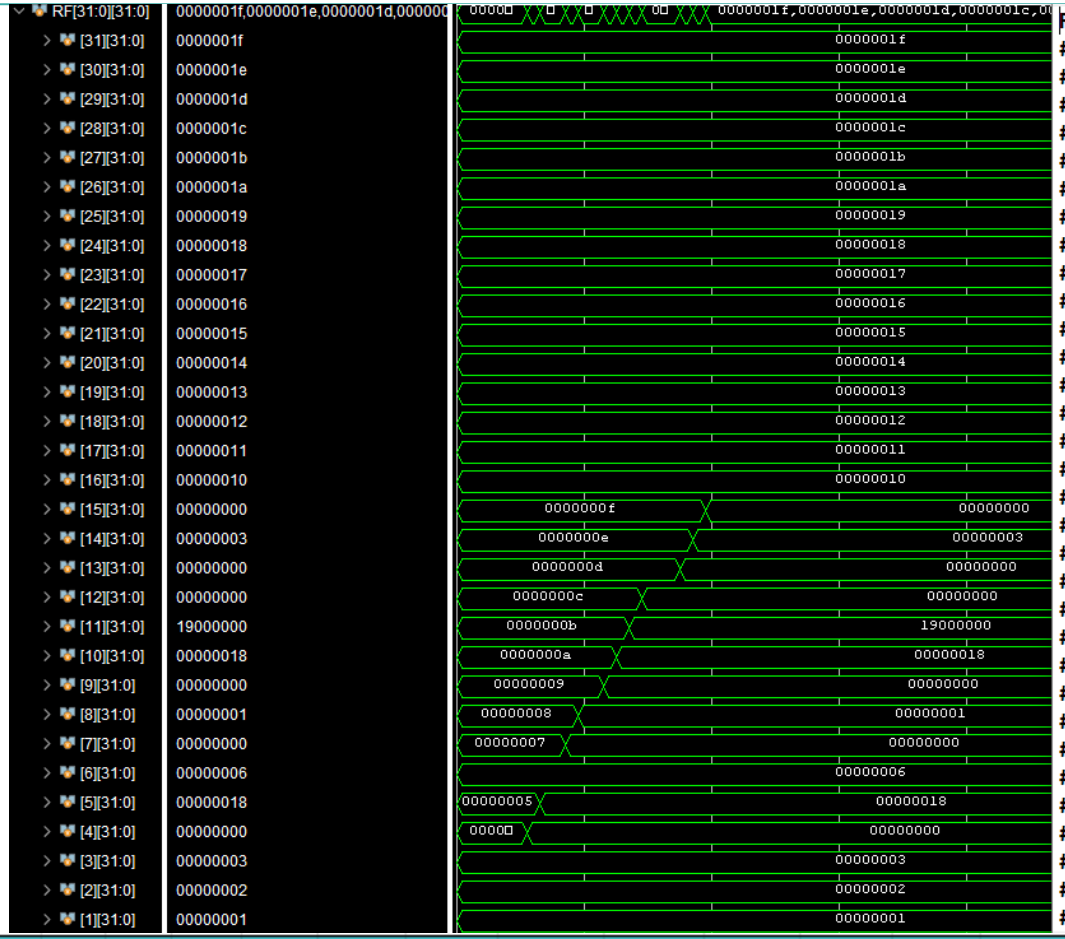
Forwarding

```
00000000000010001000000110110011
00000000000000000010001000000011
00000001100000000000001010010011
00000000011000000010011000100011
00000000000000010001000111000011
01000000000100010000010000110011
11111101010010011000010011100011
00000000001000001111010010110011
00000000010101001110010100110011
00000001100011001001010110110011
00000001110011101101011000110011
00000000001000001000000001111111
00000000010000001000000001111111
00000000000000000010011010000011
00000000000100010000011100110011
01000000100100100000011110110011
```

Instruction Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Add R3,R2,R1	F	D	X	M	W															
lw R4,0(R0)		F	D	X	M	W														
addi R5,24(R0)			F	D	X	M	W													
sw R6,12(R0)				F	D	X	M	W												
lw R7,0(R4)					F	D	X	M	W											
sub R8,R2,R1						F	D	X	M	W										
beq R19,R20							F	D	X	M	W									
and R9,R1,R2								F	D	X	M	W								
or R10,R9,R5									F	D	X	M	W							
sll R11,R25,R24										F	D	X	M	W						
sra R12,R29,R28											F	D	X	M	W					
mac R2,R1												F	D	X	M	W				
mac R4,R1													F	D	X	M	W			
lw R13,0(R0)														F	D	X	M	W		
add r14,r2,r3															F	D	X	M	W	
sub r15,r4,r9																F	D	X	M	W



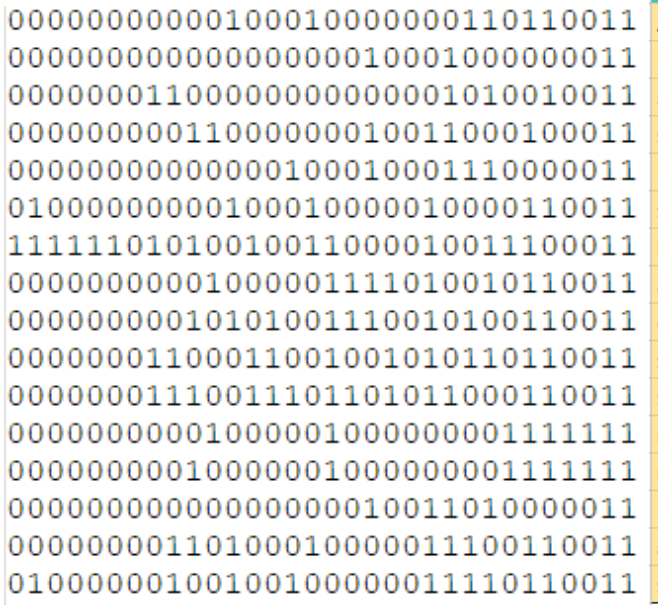
Forwarding testcase verification with interpreter



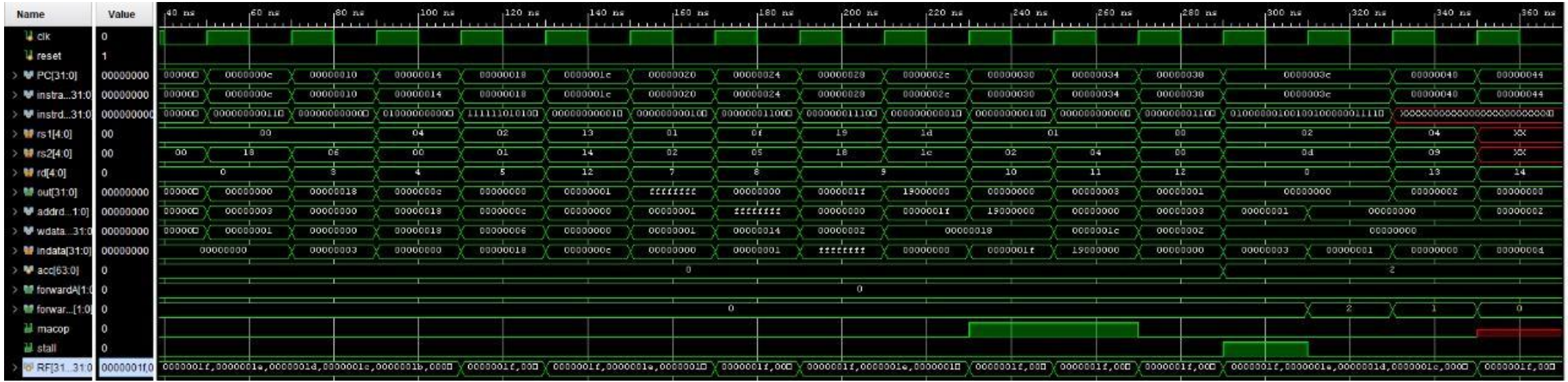

```
Register Expected Results
## expect[0] = 0x00000000
## expect[1] = 0x00000001
## expect[2] = 0x00000002
## expect[3] = 0x00000003
## expect[4] = 0x00000000
## expect[5] = 0x00000018
## expect[6] = 0x00000006
## expect[7] = 0x00000000
## expect[8] = 0x00000001
## expect[9] = 0x00000000
## expect[10] = 0x00000018
## expect[11] = 0x19000000
## expect[12] = 0x00000000
## expect[13] = 0x00000000
## expect[14] = 0x00000003
## expect[15] = 0xfffffffe
## expect[16] = 0x00000010
## expect[17] = 0x00000011
## expect[18] = 0x00000012
## expect[19] = 0x00000013
## expect[20] = 0x00000014
## expect[21] = 0x00000015
## expect[22] = 0x00000016
## expect[23] = 0x00000017
## expect[24] = 0x00000018
## expect[25] = 0x00000019
## expect[26] = 0x0000001a
## expect[27] = 0x0000001b
## expect[28] = 0x0000001c
## expect[29] = 0x0000001d
## expect[30] = 0x0000001e
## expect[31] = 0x0000001f
```

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Instruction Fetch	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15				
Instruction Decode		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15			
Execute			I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15		
Memory access				I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	
Write Back					I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15

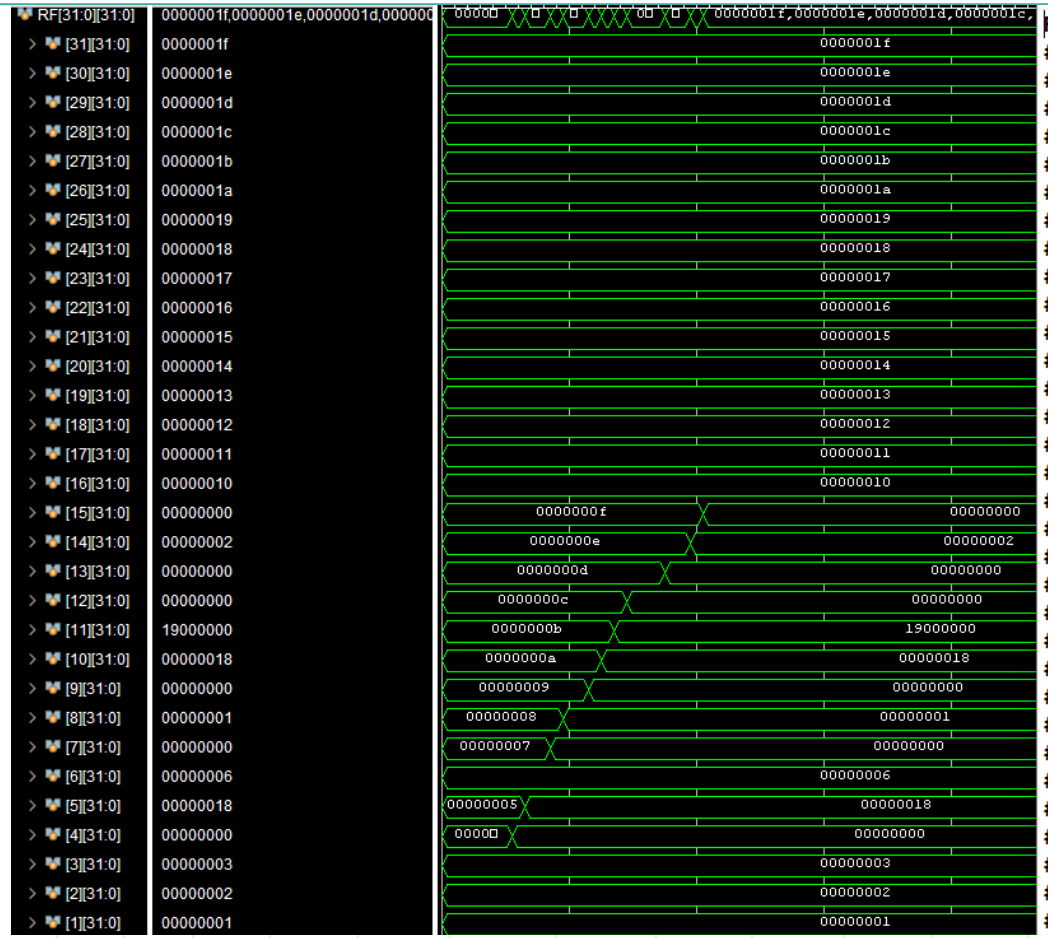
Normal case + Stall + Forwarding



Instruction Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Add R3,R2,R1	F	D	X	M	W																
lw R4,0(R0)		F	D	X	M	W															
addi R5,24(R0)			F	D	X	M	W														
sw R6,12(R0)				F	D	X	M	W													
lw R7,0(R4)					F	D	X	M	W												
sub R8,R2,R1						F	D	X	M	W											
beq R19,R20							F	D	X	M	W										
and R9,R1,R2								F	D	X	M	W									
or R10,R9,R5									F	D	X	M	W								
sll R11,R25,R24										F	D	X	M	W							
sra R12,R29,R28											F	D	X	M	W						
mac R2,R1												F	D	X	M	W					
mac R4,R1													F	D	X	M	W				
lw R13,0(R0)														F	D	X	M	W			
add r14,r2,r13															F	D	D	X	M	W	
sub r15,r4,r9																F	F	D	X	M	W



All testcase verification with interpreter




```
Register Expected Results
## expect[0] = 0x00000000
## expect[1] = 0x00000001
## expect[2] = 0x00000002
## expect[3] = 0x00000003
## expect[4] = 0x00000000
## expect[5] = 0x00000018
## expect[6] = 0x00000006
## expect[7] = 0x00000000
## expect[8] = 0x00000001
## expect[9] = 0x00000000
## expect[10] = 0x00000018
## expect[11] = 0x19000000
## expect[12] = 0x00000000
## expect[13] = 0x00000000
## expect[14] = 0x00000002
## expect[15] = 0xfffffffe
## expect[16] = 0x00000010
## expect[17] = 0x00000011
## expect[18] = 0x00000012
## expect[19] = 0x00000013
## expect[20] = 0x00000014
## expect[21] = 0x00000015
## expect[22] = 0x00000016
## expect[23] = 0x00000017
## expect[24] = 0x00000018
## expect[25] = 0x00000019
## expect[26] = 0x0000001a
## expect[27] = 0x0000001b
## expect[28] = 0x0000001c
## expect[29] = 0x0000001d
## expect[30] = 0x0000001e
## expect[31] = 0x0000001f
```

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Instruction Fetch	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	X	I15				
Instruction Decode		I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15			
Execute			I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15		
Memory access				I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15	
Write Back					I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	X	I14	I15

Result

- From the simulation performed it has been seen that the stalling, forwarding, branching operation has shown correct functionality using the 5-stage of pipelining.
- And the forwarding was also performed from the memory and writeback to execution stage.
- Finally MAC unit was also included in the RISC-V architecture using 2-stages

Reference

- https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#ebreak
- https://riscv.org/technical/specifications/
- https://www.sciencedirect.com/topics/computer-science/instruction-memory
- https://www.cs.colostate.edu
- Design of MIPS Processor Supporting MAC with FPGA Munju Lee, Taewoo Han School of Electrical and Electronic Engineering College of Engineering