THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

CSCE 3301

Dr. Cherif Salama

Rana Taher 900221430

Yasmina Mahdy 900221083

RISC-V Pipelined Processor

Project Report

Outline

Project Description

This project is an implementation of the RISC-V pipelined processor with added support for all the 42 RV32I instructions. The datapath consists of 5 separate stages, namely Instruction Fetch (IF), Instruction Decoding and Register Reading (ID), Execution (EX), Memory Access (MEM), and write back (WB). Each stage is separated by a pipeline register that preserves the needed values for each instruction as it propagates through different stages in the pipeline. The processor follows a von Neumann architecture by using a single memory for both the instruction and the datapath, and avoids possible structural hazards by operating the pipeline at half cycles. For other types of hazards, two separate hazard-handling units were created to handle all hazards arising in the ID and EX stages.

Bonuses

We attempted to implement two bonuses in this project. The first of which was moving the branch control to the ID stage. This made us also move other components like the branch and PC control units, target address adder as well and jalr address adder to the ID stage, as well as creating a new flags unit to assist in determining the branch outcome. Not only that but also the normal forwarding unit that we have for other instructions wouldn't have worked for the branch now that it is in the ID stage, so we created a hazard unit that handles the needed forwarding and stalling for the branch instructions. This hazard unit forwards the results from the execution stage for normal RAW dependencies, but stalls the pipeline for one cycle in case of a load-use dependency.

The second bonus we attempted to implement was running the program on the Nexys FPGA. Unfortunately the bonus, was not fully functional as some of the values needed like the pc_in and pc_out were displayed, but the others such as the write data were not displayed correctly.

Tests

<u>Test Case 1</u>:

Purpose:

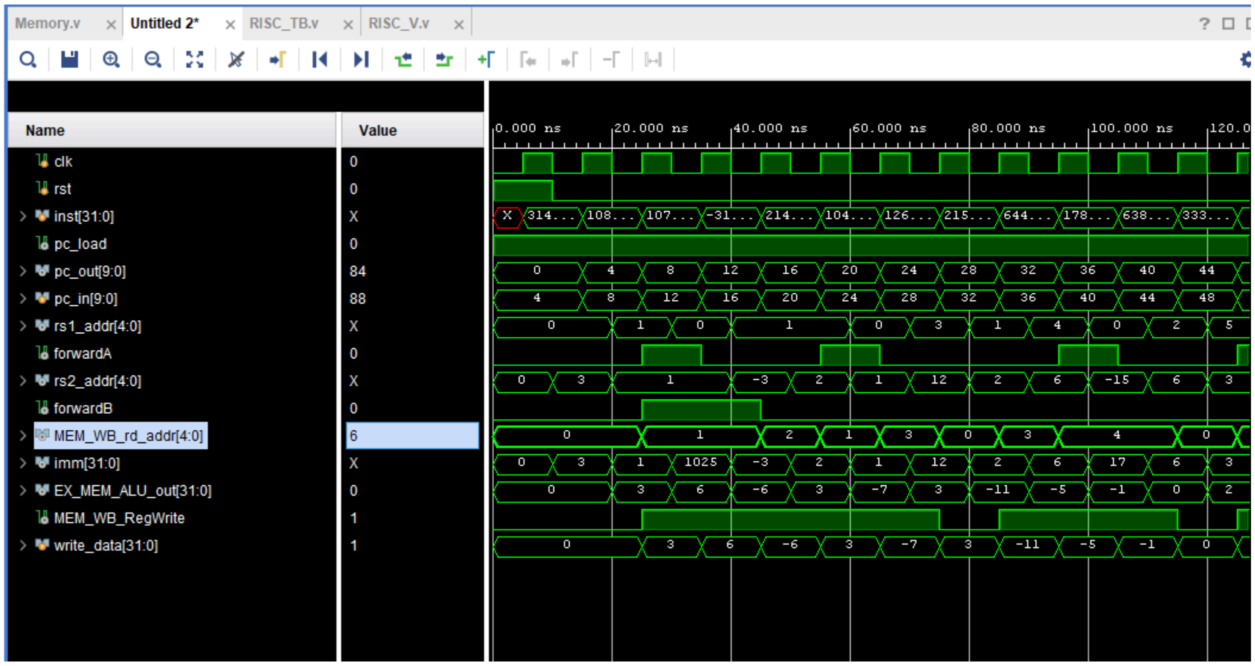
Testing R and I instructions, as well as the syscall instructions (ecall, fence, etc) and RAW dependencies

Assembly

```
addi x1, x0, 3  # x1 = 3
add x1, x1, x1 # x1 = 6
sub x2, x0, x1 # x2 = -6
addi x1, x1, -3 # x1 = 3
xor x3, x1, x2  # x3 = -7
```

```
ebreak
xori x3, x3, 12 # x3 = -11
or x4, x1, x2 # x4 = -5
ori x4, x4, 6 # x4 = -1
fence 1, 1
andi x5, x2, 6 # x5 = 2
and x5, x5, x3 # x5 = 0
slli x4, x4, 1 # x4 = -2
srai x4, x4, 1 # x4 = -1
srli x4, x4, 1 # x4 = really big number
fence.i
slt x6, x3, x2 # x6 = 1
sltu x6, x3, x6 # x6 = 0
slti x5, x2, 0 # x5 = 0
sltiu x6, x5, -1 # x6 = 1
ecall
```

Waveform:



Name	Value
clk	0
rst	0
inst[31:0]	X
pc_load	0
pc_out[9:0]	84
pc_in[9:0]	88
rs1_addr[4:0]	X
forwardA	0
rs2_addr[4:0]	X
forwardB	0
MEM_WB_rd_addr[4:0]	6
imm[31:0]	X
EX_MEM_ALU_out[31:0]	0
MEM_WB_RegWrite	1
write_data[31:0]	1

Fig. 1. *Snapshot of the simulation for test 1 with some relevant signals*

Expected output

Order	Instruction	Rs1	Rs2	Imm	Write Data	Other
1	addi x1, x0, 3	0	x	3	x1 = 3	
2	add x1, x1, x1	3	3	x	x1 = 6	Double RAW dependecy on i1
3	sub x2, x0, x1	0	6	x	x2 = -6	RAW dependency on i2
4	addi x1, x1, -3	6	x	-3	x1 = 3	
5	xor x3, x1, x2	3	-6	x	x3 = -7	Raw depedency with i4
6	ebreak	x	x	x	x	nop
7	xori x3, x3, 12	-7	x	12	x3 = -11	Raw depenedcy with i5
8	or x4, x1, x2	3	-6	x	x4 = -5	
9	ori x4, x4, 6	-5	x	6	x4 = -1	Raw dependency with i7
10	fence 1, 1	x	x	x	x	nop
11	andi x5, x2, 6	-6	x	6	x5 = 2	
12	and x5, x5, x3	2	-11	x	x5 = 0	Raw dependency with i11
13	slli x4, x4, 1	-1	x	1	x4 = -2	
14	srai x4, x4, 1	-2	x	1	x4 = -1	Raw dependency with i13
15	srli x4, x4, 1	-1	x	1	x4 = 2147483647	Raw dependency with i14
16	fence.i	x	x	x	x	nop

17	slt x6, x3, x2	-11	-6	x	x6 = 1	
18	sltu x6, x3, x6	-11	1	x	x6 = 0	Raw dependency with i17
19	slti x5, x2, 0	-6	x	0	x5 = 1	
20	sltiu x6, x5, -1	1	x	-1	x6 = 1	Raw dependency with i19
21	ecall	x	x	x	x	Stop program

Test Case 2:

Purpose:

Testing store and load instructions, as well as the load hazards (treated as normal RAW dependencies due to implementation details)

Assembly:

```
addi x12, x0, 512
addi x1, x0, -6
addi x2, x0, -12
addi x3, x0, -14

sb x1, 0(x12)
sh x2, 1(x12)
sw x3, 3(x12)

lb x4, 0(x12) # x4 = -6
addi x9, x4, 0 # x9 = -6
lbu x5, 0(x12) # x5 = 250
addi x9, x5, 0 # x9 = 250
lh x6, 1(x12) # x6 = -12
addi x9, x6, 0 # x9 = -12
lhu x7, 1(x12) # x7 = 65524
addi x9, x7, 0 # x9 = 65524
lw x8, 3(x12) # x8 = -14
addi x9, x8, 0 # x9 = -14
ecall
```

Waveform:

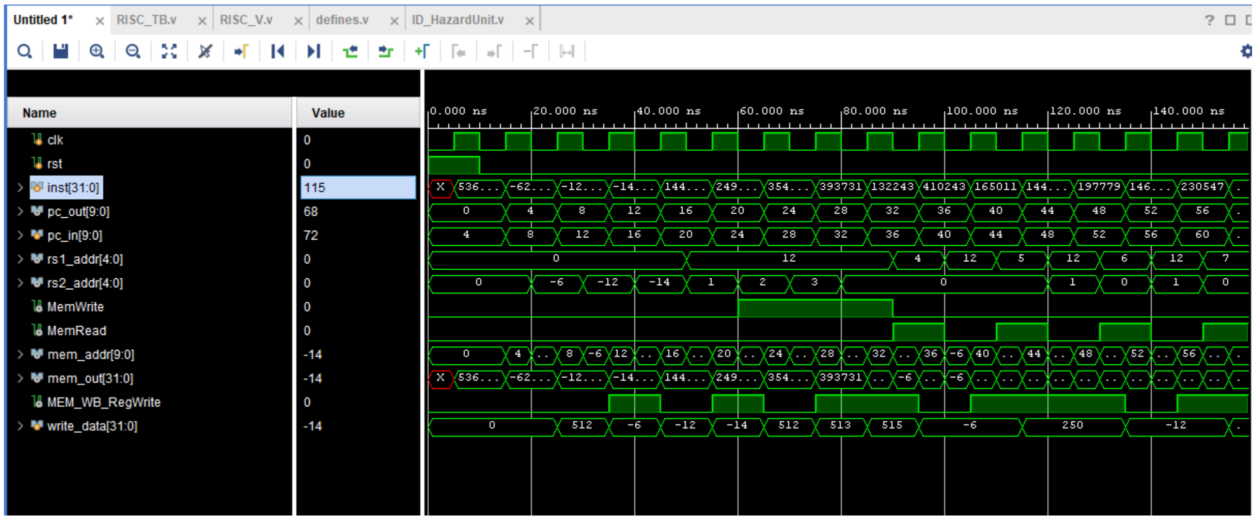


Fig. 2. *Snapshot of the simulation for test 2 with some relevant signals*

Expected output:

Order	Instruction	Rs1	Rs2	Imm	Write Data	Other
1	addi x12, x0, 1024	0	x	512	x12 = 512	
2	addi x1, x0, -6	0	x	-6	x1 = -6	
3	addi x2, x0, -12	0	x	-12	x2 = -12	
4	addi x3, x0, -14	0	x	-14	x3 = -14	
5	sb x1, 0(x12)	512	-6	0	x	Mem[512] = -6
6	sh x2, 1(x12)	512	x	1	x	Mem[513] = -12
7	sw x3, 3(x12)	512	x	3	x	Mem[515] = -14
8	lb x4, 0(x12) # x4 = -6	512	x	0	x4 = -6	Reads from mem[512]
9	addi x9, x4, 0 # x9 = -6	-6	x	0	x9 = -6	Raw dependencies with i8
10	lbu x5, 0(x12)	512	x	0	x5 = 250	Reads from

						mem[512]
11	addi x9, x5, 0	250	x	0	x9 = 250	Raw dependencies with i10
12	lh x6, 1(x12)	512	x	1	x6 = -12	Reads from mem[513]
13	addi x9, x6, 0	-12	x	0	x9 = -12	Raw dependencies with i12
14	lhu x7, 1(x12)	512	x	1	x7 = 65524	Reads from mem[513]
15	addi x9, x7, 0	65524	x	0	x9 = 65524	Raw dependencies with i14
16	lw x8, 3(x12)	512	x	3	x8 = -14	Reads from mem[515]
17	addi x9, x8, 0	-14	x	0	x9 = -14	Raw dependencies with i16
18	ecall	x	x	x	x	Stop program

Test Case 3:

Purpose:

Testing all branch instructions, when taken and not taken

Assembly:

```
addi x1, x0, 1 # x1 = 1
addi x2, x0, 1 # x2 = 1
addi x3, x0, -1 # x3 = -1
addi x4, x0, -2 # x4 = -2

beq x0, x0, L1 # branch taken
add x2, x2, x2 # skipped
```

```
L1:
beq x0, x1, L2 # branch not taken
add x1, x1, x1 # x1 = 2

blt x2, x1, L2 # taken after stall
add x2, x2, x2 # skipped
L2:
blt x1, x0, L3 # not taken
add x1, x1, x1 # x1 = 4

bge x0, x0, L3 # taken
add x2, x2, x2 # skipped
L3:
bge x0, x1, L4 # not taken
add x1, x1, x1 # x1 = 8

bne x1, x2, L4 # taken after stall
add x2, x2, x2 # skipped

L4:
bne x0, x0, L5 # not taken
add x1, x1, x1 # x1 = 16

bltu x0, x3, L5 # taken
add x2, x2, x2 # skipped

L5:
bltu x3, x4, L6 # not taken
add x1, x1, x1 # x1 = 32

bgeu x3, x1, L6 # taken after stall
add x2, x2, x2 # skipped

L6:
bgeu x2, x3, L1 # not taken
add x1, x1, x1 # x1 = 64
add x5, x1, x0 # x5 = 64
addi x5, x5, 1 # x5 = 65
beq x1, x5, L7 # not taken after stall
Ecall
```

L7:
addi x1, x1, 0

Waveform:

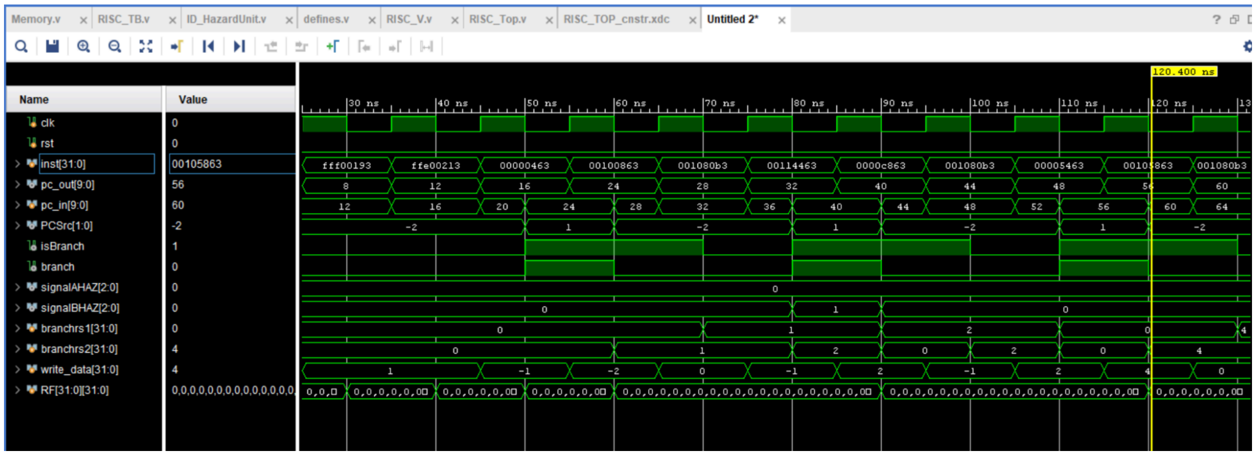


Fig. 3. *Snapshot of the simulation for test 3 with some relevant signals*

Expected output:

Order	Instruction	Rs1	Rs2	Imm	Write Data	Other
1	addi x1, x0, 1	0	x	1	x1 = 1	
2	addi x2, x0, 1	0	x	1	x2 = 1	
3	addi x3, x0, -1	0	x	-1	x3 = -1	
4	addi x4, x0, -2	0	x	-2	x4 = -2	
5	beq x0, x0, L1	0	0	8	x	Branch taken
6	beq x0, x1, L2	0	1	16	x	Branch not taken
7	add x1, x1, x1	1	1	2	x1 = 2	
8	blt x2, x1, L2	1	2	8	x	Branch taken (dependency on i7)
9	blt x1, x0, L3	2	0	16	x	Branch not taken

10	add x1, x1, x1	2	2	x	x1 = 4	
11	bge x0, x0, L3	0	0	8	x	Branch taken
12	bge x0, x1, L4	0	4	16	x	Branch not taken
13	add x1, x1, x1	4	4	x	x1 = 8	
14	bne x1, x2, L4 # taken after stall	8	1	8	x	Branch taken (dependency on i13)
15	bne x0, x0, L5	0	0	16	x	Branch not taken
16	add x1, x1, x1	8	8	x	x1 = 16	
17	bltu x0, x3, L5	0	-1	8	x	Branch taken
18	bltu x3, x4, L6	-1	-2	16	x	Branch not taken
19	add x1, x1, x1	16	16	x	x1 = 32	
20	bgeu x3, x1, L6	-1	32	8	x	Branch taken (dependency on i19)
21	bgeu x2, x3, L1	1	-1	16	x	Branch not taken
22	add x1, x1, x1 # x1 = 64	32	32	x	x1 = 64	
23	add x5, x1, x0 # x5 = 64	64	0	x	x5 = 64	Raw dependency on i22
24	addi x5, x5, 1 # x5 = 65	64	x	1	x5 = 65	Raw dependency on i23
25	beq x1, x5, L7 # not taken after stall	64	65	8	x	Branch taken (dependecy on i24)
26	ecall	x	x	x	x	Stop program

Test Case 4:

Purpose:

Testing jumps and U type

Assembly:

```
lui x1, 1 # x1 = 4096
auipc x2, 1 # x2 = 4100
jal x3, L1 # x3 = 12
addi x2, x2, 5 # skipped
addi x2, x2, 5 # skipped
jalr x0, x4, 0 # skipped at first; then jumps to addi (28)
L1:
jalr x4, x0, 20 # x4 =28
ecall
```

Waveform:

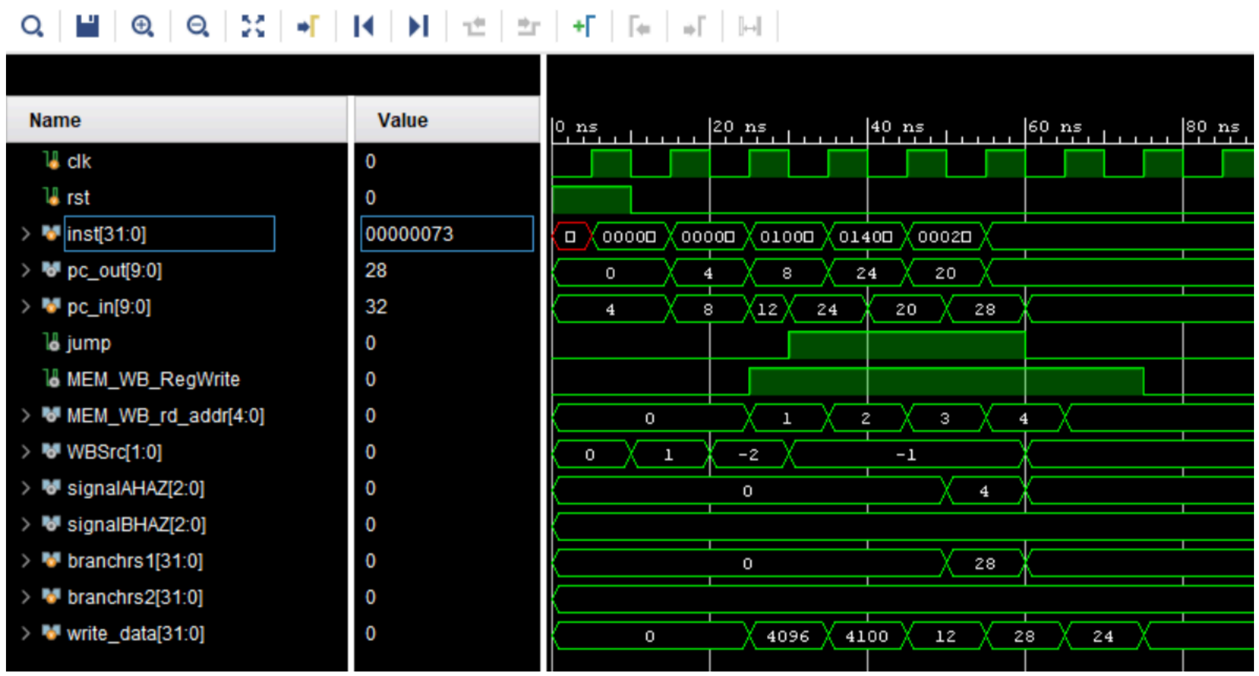


Fig. 4. *Snapshot of the simulation for test 4 with some relevant signals*

Expected output:

Order	Instruction	Rs1	Rs2	Imm	Write Data	Other
1	lui x1, 1	x	x	4096	x1 = 4096	
2	auipc x2, 1	x	x	4096	x2 = 4100	
3	jal x3, L1	x	x	24	x3 = 12	
4	jalr x4, x0, 20	0	x	20	x4 = 28	
5	jalr x0, x4, 0	28	x	0	x0 unchanged	RAW dependency on i4
6	ecall	x	x	1	x1 = 1	

Test Case 5:

Purpose:

Testing all branch hazard situations (stalling/ forwarding from different components)

Assembly:

```
addi s0, x0, 512 # base address for store/load
addi s1, x0, 1 # s1 = 1

sw s1, 0(s0)
lw s2, 0(s0) # s2 = 1
beq s1, s2, L1 # taken after stall (handles load use)
addi s2, s2, 15 # skipped but would result in s2 = 16
addi s2, s2, 15
addi s2, s2, 15

L1:
addi s2, s2, 1 # s2 = 2
lui s1, 1 # s1 = 4096
lui s2, 1 # s2 = 4096
beq s1, s2, L2 # should branch (handles lui)
```

```
addi s2, s2, -96 # skipped, but would result in s2 = 4000
addi s2, s2, 15
addi s2, s2, 15

L2:
addi s1, x0, 18 # s1 = 18
addi s2, x0, 9 # s2 = 9
add s2, s2, s2 # s2 = 18
beq s1, s2, L3 # taken (handles R format)
sub s2, s2, s2 # would be skipped but sets s2 = 0
addi s2, s2, 15
addi s2, s2, 15

L3:
sub s2, s2, s2 # s2 = 0
addi s2, s2, 9 # s2 = 9
bne s2, s1, L4 # taken (handles I format) [note that s1 = x9 BUT since i-format, signalB
should not be set and s1 should be 18, if not taken then there's an issue]
addi s2, s2, 15
addi s2, s2, 15
addi s2, s2, 15

L4:
auipc s1, 0
addi s1, s1, 8
auipc s2, 0
beq s1, s2, L5 # taken
sub s2, s1, s2
addi s2, s2, 15
addi s2, s2, 15

L5:
auipc s0, 0
addi s0, s0, 12 # s0 had address of instruction after jal
jal s1, L6
addi x1, x0, 3 # skipped

L6:
beq s0, s1, L7 # tests branch after jump
add s1, s1, x0
add s1, s1, x0
add s1, s1, x0

L7:
ecall
```

Waveform:

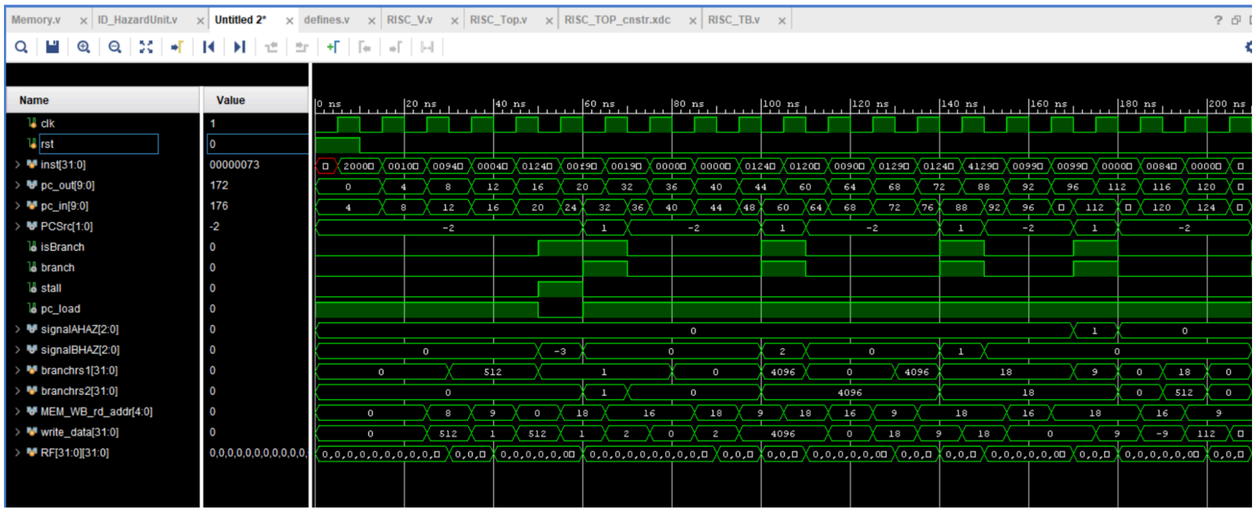


Fig. 5. *Snapshot of the simulation for test 5 with some relevant signals*

Expected output:

Order	Instruction	Rs1	Rs2	Imm	Write Data	Other
1	addi s0, x0, 512	0	x	512	x8 = 512	base address for store/load
2	addi s1, x0, 1	0	x	1	x9 = 1	
3	sw s1, 0(s0)	512	1	0	x	mem[512]
4	lw s2, 0(s0)	512	x	0	x18 = 1	
5	beq s1, s2, L1	1	1	16	x	Stalls (dependecy on i4) then taken (handles load)
6	addi s2, s2, 1	1	x	1	x18 = 2	

7	lui s1, 1	x	x	4096	x9 = 4096	
8	lui s2, 1	x	x	4096	x18 = 4096	
9	beq s1, s2, L2	4096	4096	16	x	Branches immediately (dependency on i8 so forwards) (handles lui format)
10	addi s1, x0, 18	0	x	18	x9 = 18	
11	addi s2, x0, 9 # s2 = 9	0	x	9	x18 = 9	
12	add s2, s2, s2 # s2 = 18	9	9	x	x18 = 18	
13	beq s1, s2, L3	18	18	16	x	Branches immediately (dependency on i12 so forwards) (handles R format)
14	sub s2, s2, s2	18	18	x	x18 = 0	
15	addi s2, s2, 9	9	9	x	x18 = 9	
16	bne s2, s1, L4	9	18	16	x	Branches immediately (dependency on i15 so forwards) (handles I format) [note that s1 = 9 BUT since i-format, signalB should not be set and s1 should be 18, if not taken then there's an issue]

17	auipc s1, 0	x	x	0	x9 = 112	
18	addi s1, s1, 8	112	x	8	x9 = 120	
19	auipc s2, 0	x	x	0	x18 = 120	
20	beq s1, s2, L5	120	120	16	x	Branches immediately (dependency on i19 so forwards) (handles AUIPC)
21	auipc s0, 0	x	x	0	x8 = 140	
22	addi s0, s0, 12	140	x	12	x8 = 152	# s0 had address of instruction after jal
23	jal s1, L6	x	x	x	x9 = 152	
24	beq s0, s1, L7 # tests branch after jump	152	152	16	x	Branches immediately (dependency on i23 so forwards) (handles jump)
25	ecall	x	x	x	x	Stops program

Incremental Testing
Throughout the project, starting from the single cycle implementation, we thoroughly tested and debugged all instructions following all modifications or additions to the project. Those include adding the pipelining regs (with added nops), using a single memory for instructions and data, adding the forwarding logic, as well as adding the branch and branch hazards logic. This greatly helped us spot errors early on and be able to pinpoint their cause without having to go through the entire datapath. The tests included in this report are the final tests after everything has been implemented, with additional edge cases and scenarios that helped us spot elusive bugs.

Problems and Solutions

P1: Program used to totally pause at stalling instead of a nop.
S1: We were using an incorrect condition to set the PCSrc, which overlapped for both jal and fence and caused the program to jump to an undefined target address and hault; used a better condition.

P2: When reading memory values from a file, the program would attempt to read a full word and place it in one byte, causing the rest of the word to be truncated and the eventual memory output to be incorrect.
S2: Created a python script to split the hex file input into 1 byte/line instead of 1 word/line.

P3: ALU started calculating wrong values after we used a single memory and implemented forwarding.
S3: After tracing we figured out that we were passing incorrect clocks to the ID_EX and RF, which resulted in the correct results being written to those regs later than required, messing up the forwarding logic.

P4: Did not stall after load-use branch RAW dependencies.
Stall not working in load-use branch: forgot to actually stall the pipeline lol
S4: Only added the stall signal and stalled the PC but forgot to actually turn the last fetched instruction into a nop.

P6: Auipc behaving incorrectly after adding branching logic
S6: Accidentally deleted pc from ID_EX pipeline reg after mistakenly believing it was no longer needed.

P7: lbu and lhu not being executed correctly.
S7: Turned out not to be an issue with load, but with the stalling and forwarding logic as we neglected to limit the stalls to branches and jumps, so the sequence of instructions preceding the loads led to an incorrect stall being identified and executed.

P8: Program would branch in a situation where it shouldn’t.
Issue not branching
S8: Mixed up the source and destination logic in the branch hazard unit and did not set an I-type condition for forwarding rs2.

P9 : Ecall would halt the program, but the pc would be updated one final time and a new instruction fetched before stopping.
S9: Made the last instruction a nop.

P10: Difficult to test the program on the fpga as running synthesis and implementation took significant time due to the size of the memory
S10: We reduced the size of the memory to reduce the time needed.

Datapath

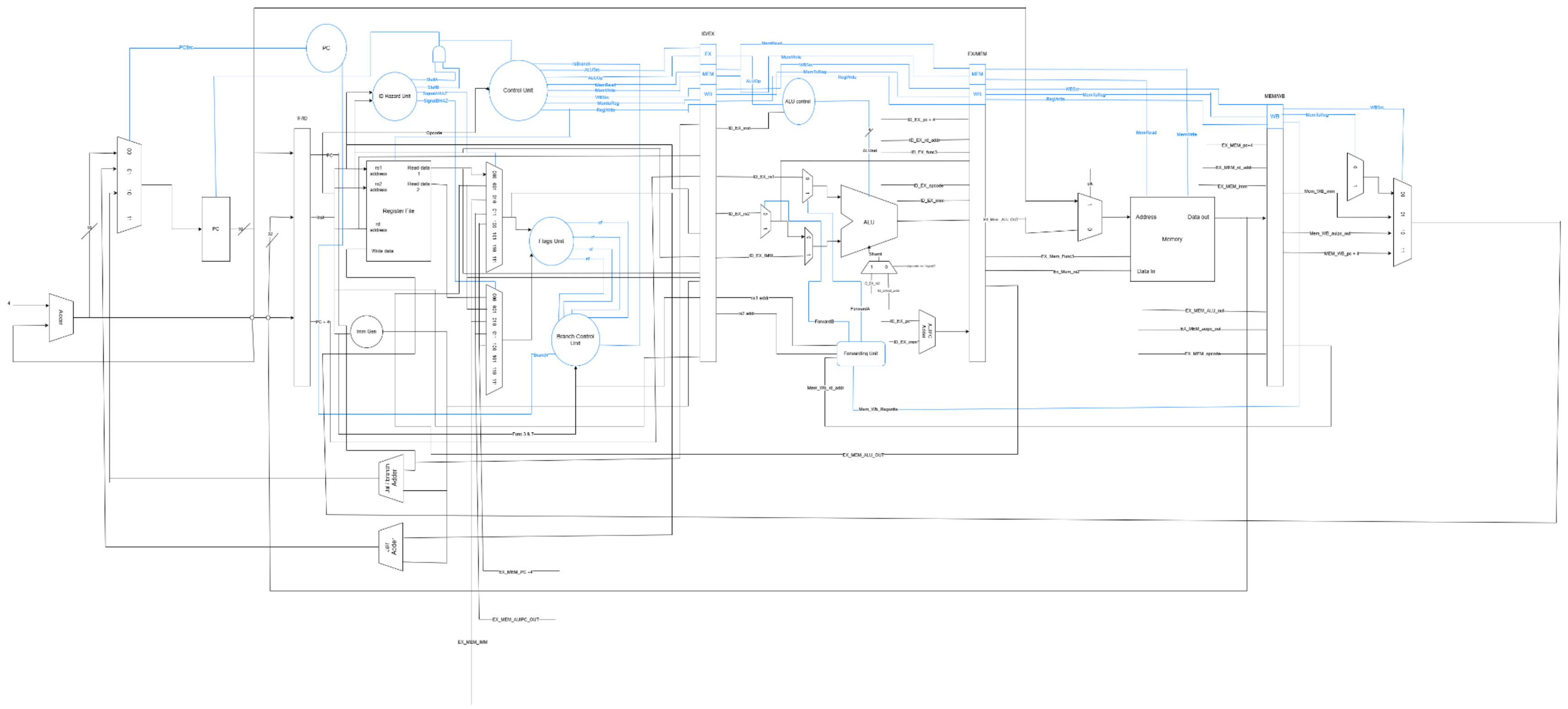


Fig. 6. *Pipelined datapath (note that the full picture is also available in the reports folder since this one is not very visible)*

Future Improvements

- Self modifying code
  - In the combined data memory we wanted to assign the first half of the memory to the instructions and the second half to the data. We would then handle any incorrect memory accesses by modifying the address to be accessed.
- Separate memory banks
  - To handle **Problem 2** more elegantly, we thought of having four different memory banks that sequentially store the bytes of each word to be able to directly load hex files into them.
- Supporting Extensions
  - We want to support other extensions in RISC-V such as the C and M extensions.
- Fixing FPGA display
  - We would like to fix the issues in the current FPGA implemengtation.