Test cases (Prepared by Dr. Muhamed Mudawar)
Instructions were hand-assembled (can have errors, check Hexadecimal)

Initializing Registers (Testing I-Type ALU):

Instruction	Hexadecimal	Expected Result
lui r1, 0xc78f	0x2801c78f	r1 = 0xc78f0000 (shifted 16 bits)
ori r2, r0, 4	0x21020004	r2 = 4 = 0x00000004
addi r3, r0, -2	0x2403fffe	r3 = -2 = 0xfffffffe (sign extension)
xori r4, r0, -2	0x2204fffe	r4 = 0x0000fffe (zero extension)

Testing R-Type ALU Instructions (NO RAW hazards – NO Forwarding)

Instruction	Hexadecimal	Expected Result
add r5, r1, r1	0x04115000	r5 = 0x8f1e0000 (carry is ignored)
sub r6, r1, r2	0x05126000	r6 = 0xc78efffc
slt r7, r1, r2	0x06127000	r7 = 1 (true) r1 < 0
sltu r8, r1, r2	0x07128000	r8 = 0 (false)
and r9, r3, r4	0x00349000	r9 = 0x0000fffe
or r10, r1, r2	0x0112a000	r10 = 0xc78f0004
xor r11, r1, r3	0x0213b000	r11 = 0x3870fff2
nor r12, r1, r2	0x0312c000	r12 = 0x3870fffb
sll r13, r4, r2	0x0842d000	r13 = 0x000fffe0
srl r14, r1, r2	0x0912e000	r14 = 0x0c78f000
sra r15, r1, r2	0x0a12f000	r15 = 0xfc78f000
ror r10, r3, r2	0x0b32a000	r10 = 0xeffffff

Testing RAW hazards and Forwarding

Instruction	Hexadecimal	Expected Result
add r5, r1, r1	0x04115000	r5 = 0x8f1e0000
sub r6, r5, r4	0x05546000	r6 = 0x8f1d0002 (depends on add)
and r7, r5, r6	0x00567000	r7 = 0x8f1c0000 (depends on add/sub)
ori r5, r5, 0xf	0x2155000f	r5 = 0x8f1e000f (depends on add)

Testing SW and LW

Instruction	Hexadecimal	Expected Result
sw r1, 0(r0)	0x38010000	MEM[0] = 0xc78f0000
sw r4, 1(r0)	0x38040001	MEM[1] = 0x0000fffe
lw r5, 0(r0)	0x30050000	r5 = MEM[0] = 0xc78f0000

Testing Load delay, stalling pipelining, and forwarding after LW

Instruction	Hexadecimal	Expected Result
lw r6, 1(r0)	0x30060001	r6 = MEM[1] = 0x0000fffe
andi r7, r6,0xb	0x2067000b	r7 = 0x0000000a (stall 1 cycle)
sw r7, 2(r0)	0x38070002	MEM[2] = 0x0000000a (forward)
lw r5, 0(r0)	0x30050000	r5 = MEM[0] = 0xc78f0000
sw r5, 3(r0)	0x38050003	MEM[3] = 0xc78f0000 (forward)

Testing Branch and Jump Instructions

Instruction	Hexadecimal	Expected Result
beq r1, r1, +2	0x40110002	branch forward 2 (to bne)
add r5, r2, r2	0x04225000	should be skipped (r5 not modified)
bne r0, r1, +3	0x41010003	branch forward 3 (to j)
add r6, r2, r2	0x04226000	should be skipped (r6 not modified)
add r7, r4, r4	0x04447000	should be skipped (r7 not modified)
j 0x28	0x50000028	jump to address 0x28 (ori)
add r5, r2, r2	0x04225000	should be skipped (r5 not modified)
add r6, r4, r4	0x04446000	should be skipped (r6 not modified)
and r0, r0, r0	0x00000000	NO Operation

Fibonacci Example: Testing JAL and JR

Instruction	Hexadecimal	Expected Result
address 0x28:		Address of ori = 0x28
ori r1, r0, 5	0x21010005	r1 = 5 (5th Fibonacci element)
jal fib (0x30)	0x51000030	call Fib (r15 = address of or)
or r5, r2, r0	0x01205000	move r5 = r2 (Fib result) = 8
beq r0, r0, 0	0x40000000	branch to self (stop program)
and r0, r0, r0	0x00000000	No operation
. . .	0x00000000	
address 0x30:		Fib starts here:
ori r2, r0, 1	0x21020001	r2 = 1
ori r3, r0, 1	0x21030001	r3 = 1
add r3, r2, r3	0x04233000	loop starts here: r3 = r2 + r3
sub r2, r3, r2	0x05322000	r2 = r3 – r2
addi r1, r1, -1	0x2411ffff	r1 = r1 – 1
bne r1, r0, -3	0x4110fffd	branch backward to add if (r1 != 0)
jr r7	0x10700000	return to caller
sub r2, r2, r2	0x05222000	Should be skipped (r2 not modified)