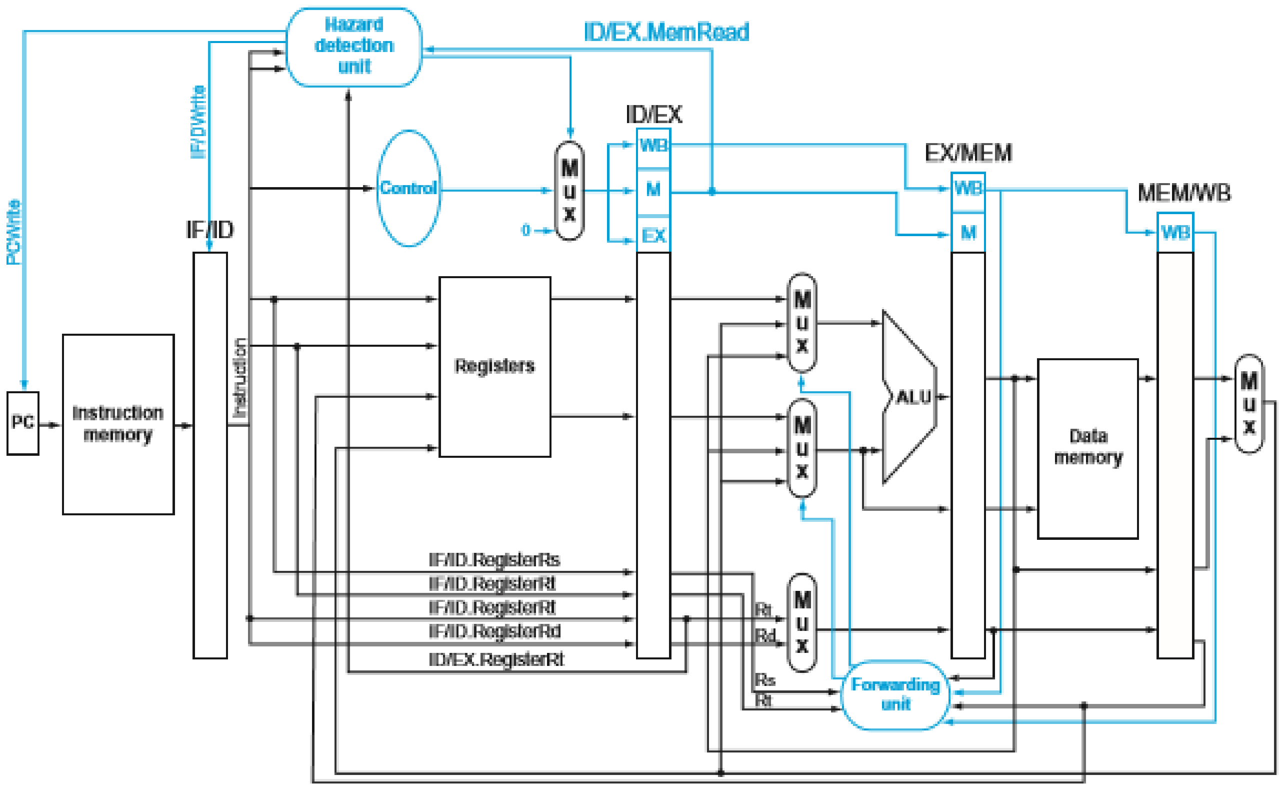
TABLE OF CONTENTS

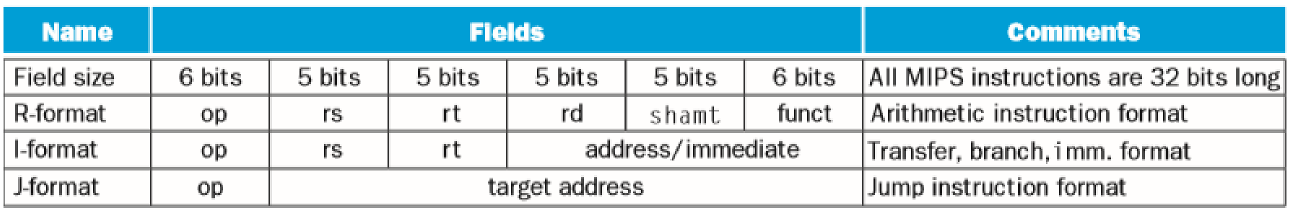
Design Details



R-format Ins ›	op	rs	rt	Rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0

I-format Ins ›	op	rs	rt	address/immediate
	31:26	25:21	20:16	15:0

J-format Ins ›	op	target address
	31:26	25:0



Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

The main module consists of:

i. Control Unit
ii. State & Data Registers
iii. ALU Control Unit
iv. Register Bank
v. Instruction Memory
vi. Data Memory
vii. ALU
viii. Adders
ix. Sign-Extension Units
x. Multiplexors
xi. Register Sets

The main module has ‘clock’ as input which is passed onto PC Register for incrementing the PC at start of each cycle (posedge), and to other modules (Register Bank, Data Memory) where write-data function is available. It also has 4 register sets between each pipeline stage to store data and control bits for each instruction to be passed on to some next stage in the processor. These register sets are named IF/ID, ID/EX, EX/MEM, MEM/WB register sets.

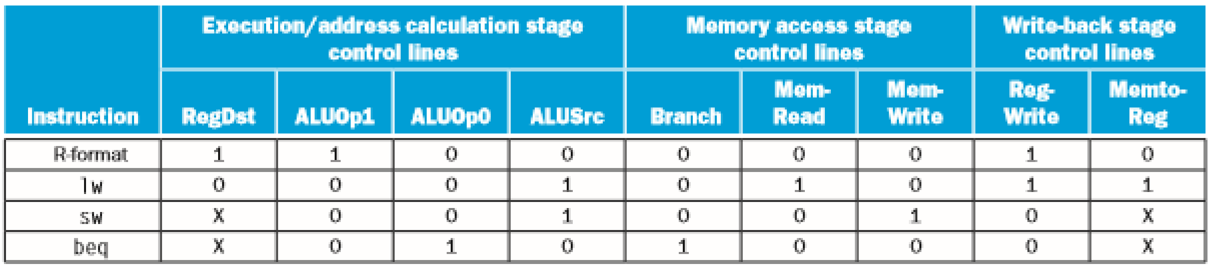
PC addresses the Instruction Memory to get the instruction, which is then decoded within the main module.

opcode	=	instruction[31:26]
rs	=	instruction[25:21]
rt	=	instruction[20:16]
rd	=	instruction[15:11]
shamt	=	instruction[10:6]
funct	=	instruction[5:0]
immediate16	=	instruction[15:0]
immjump26	=	instruction[25:0]

The opcode is passed onto the Control Unit which outputs all the Control Line values for:

RegWrite, RegDst, MemtoReg, MemRead, MemWrite, **Branch**, Jump, ALUSrc, **ALUOp**

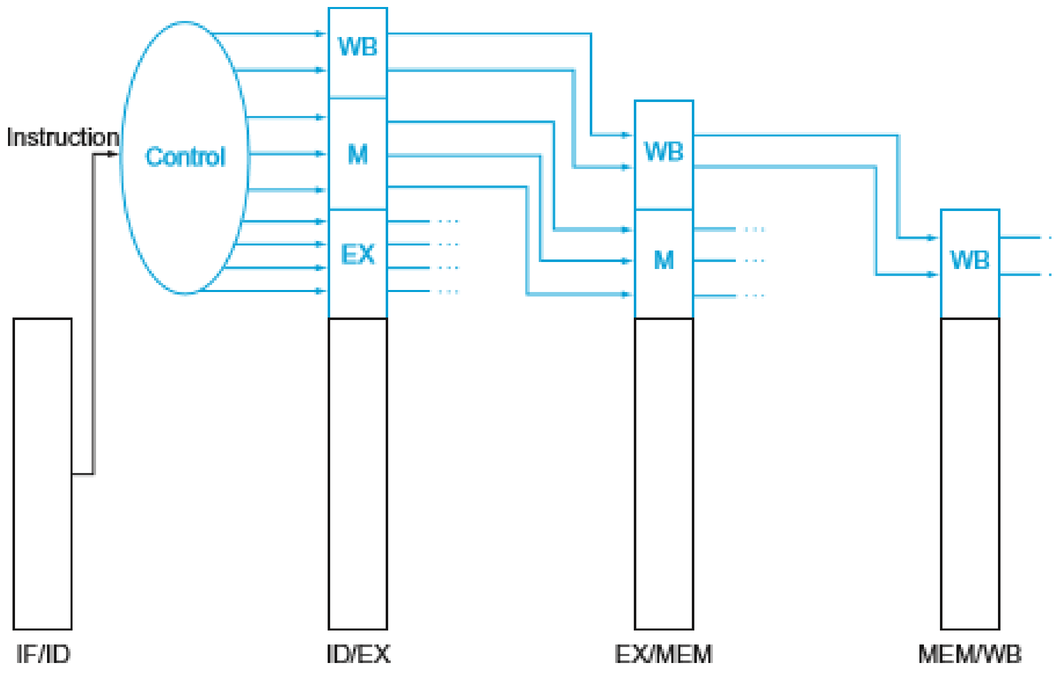
These have the following values for the different instructions:



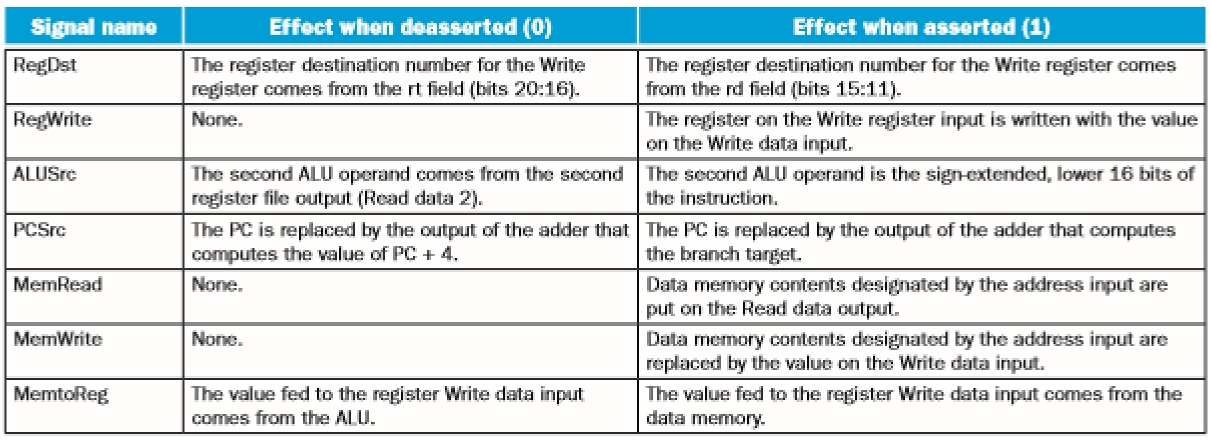
	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
Instruction	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

The control signals have been grouped into ‘EX’, ‘M’, and ‘WB’ bundles depending upon the stage where they are used. The control unit sends all these signals to be stored into the

ID/EX register set to be used in the later stages (in later cycles). In each clock cycle, at each stage, the required signals of that stage are used, and the rest are passed on to be stored in the next register set. A visual representation of this flow of signals is as follows:



The operations controlled by the control signal values are described below:



Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The data hazards that may occur during code execution are resolved using a forwarding unit in case of EX and MEM hazards, and for load-use hazards a hazard detection unit is used to stall as needed.

As for control hazards (branch), normal instruction flow is assumed, i.e. predict not taken, and if the branch condition is satisfied, the instructions in the IF/ID, ID/EX and EX/MEM registers are cleared out using a ‘flush’ signal.

Dataflow Examples

Consider the following instructions:

(1) // addi x1, x0, 7

{6'b000001, 5'b00000, 5'b00001, 16'b0000000000000111}

(2) // addi x2, x0, 3

{6'b000001, 5'b00000, 5'b00010, 16'b0000000000000011}

(3) // add x3, x1, x2

{6'b000000, 5'b00001, 5'b00010, 5'b00011, 5'b00000, 6'b100000}

(4) // sub x3, x1, x2

{6'b000000, 5'b00001, 5'b00010, 5'b00011, 5'b00000, 6'b100010}

Typically, due to the number of stages, an instruction will take 5 clock cycles to execute. The rest after it each take one more clock cycle, before all the instructions are executed. So about 8 clock cycles would be needed to execute these 4 instructions.

In the 1st clock cycle [1], the first 'addi' instruction is fetched, and it is stored into the IF/ID register set.

In the next clock cycle [2], the first 'addi' instruction is decoded in the ID stage and the source values, from register 0 (equal to 0) and the immediate '7' along-with the control bits are stored into the ID/EX register set. Meanwhile, in the IF stage, the second 'addi' instruction is fetched.

In the next clock cycle [3], the first 'addi' instruction has reached the EX stage, where the values 0 and 7 are added by the ALU, and the result (7+0 = 7) and the control bits for the next stage are stored into the EX/MEM register. In the ID stage, the second 'addi' instruction is decoded and the source values, from register 0 (equal to 0) and the immediate '3' along-with the control bits are stored into the ID/EX register set. Meanwhile, in the IF stage, the third instruction is fetched i.e. the 'add' instruction.

In the next clock cycle [4], the first ‘addi’ instruction has reached the MEM stage, but there is no memory access to be made and so the previous results (of the ALU) are passed on and stored into the MEM/WB register set. In the EX stage, the second ‘addi’ instruction is being processed, where the values 0 and 3 are added by the ALU, and the result (3+0 = 3) and the control bits for the next stage are stored into the EX/MEM register. In the ID stage, the ‘add’ instruction is to be decoded. Meanwhile, in the IF stage, the fourth instruction is fetched i.e. the ‘subtract’ instruction.

In the next clock cycle [5], the first ‘addi’ instruction has reached the WB stage, so the stored result (7) is written back to register 1. The second ‘addi’ instruction has reached the MEM stage, but there is no memory access to be made and so the previous results (of the ALU) are passed on and stored into the MEM/WB register set. In the EX stage, the third instruction i.e. the ‘add’ instruction has to be executed, but its operand’s values have to be forwarded from the MEM stage (value for register 2) and the WB stage (value for register 1). This is done with the help of the forwarding unit. In the ID stage, the ‘subtract’ instruction is decoded.

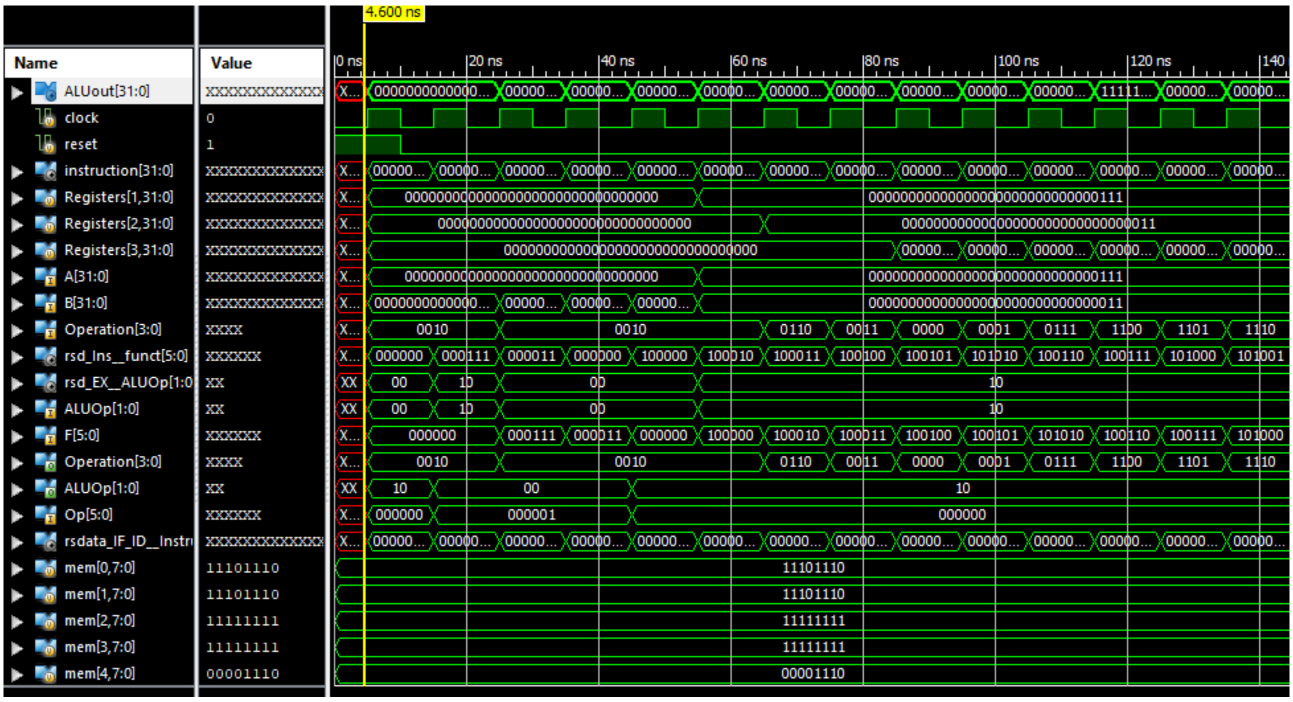
In the next clock cycle [6], the second ‘addi’ instruction has reached the WB stage, so the stored result (3) is written back to register 2. The third instruction i.e. the ‘add’ instruction has reached the MEM stage, but there is no memory access to be made and so the previous results (of the ALU) are passed on and stored into the MEM/WB register set. In the EX stage, the fourth instruction i.e. the ‘subtract’ instruction has to be executed, but its second operand’s value has to be forwarded from the WB stage (value for register 2). This is done with the help of the forwarding unit.

In the next clock cycle [7], the third (‘add’) instruction has reached the WB stage, so the stored result (7+3) is written back to register 3. The fourth instruction i.e. the ‘subtract’ instruction has reached the MEM stage, but there is no memory access to be made and so the previous results (of the ALU) are passed on and stored into the MEM/WB register set.

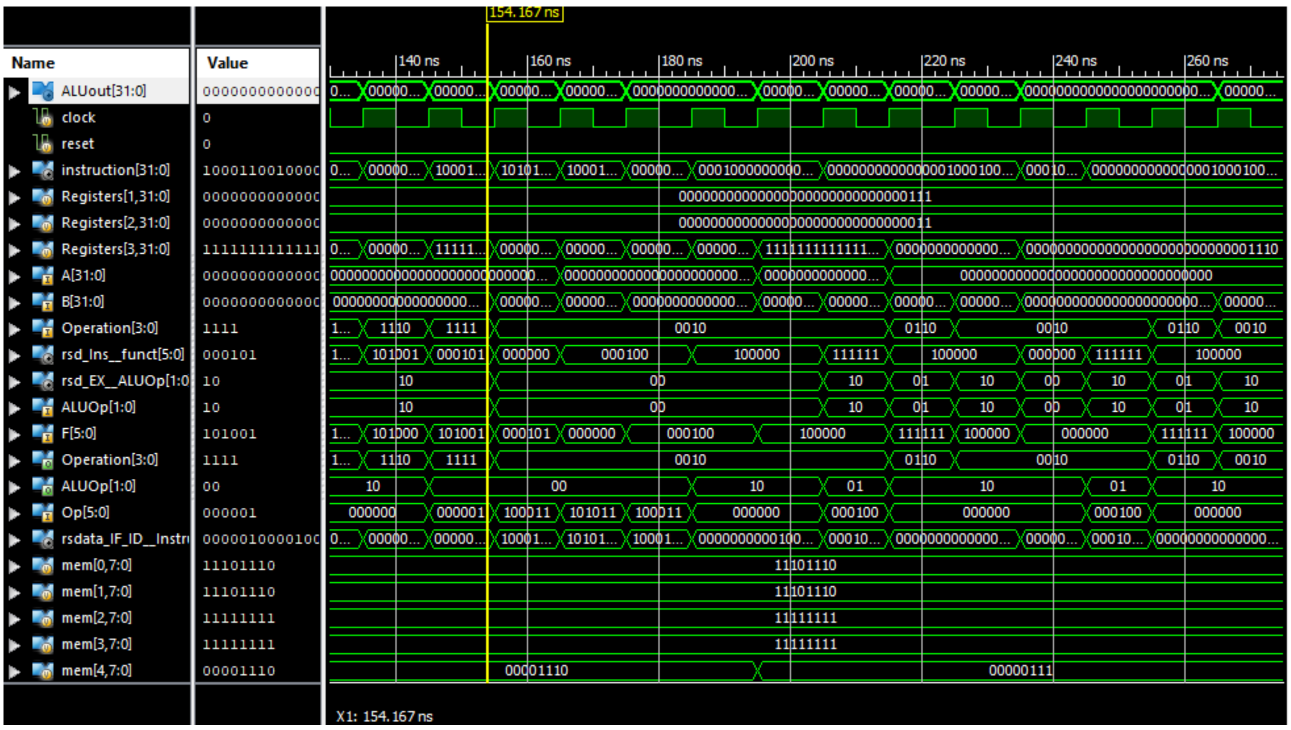
In the next clock cycle [8], the fourth (‘subtract’) instruction has reached the WB stage, so the stored result (7-3) is written back to register 3.

Simulation Preview

A few screenshots from the start of the simulation are provided below:

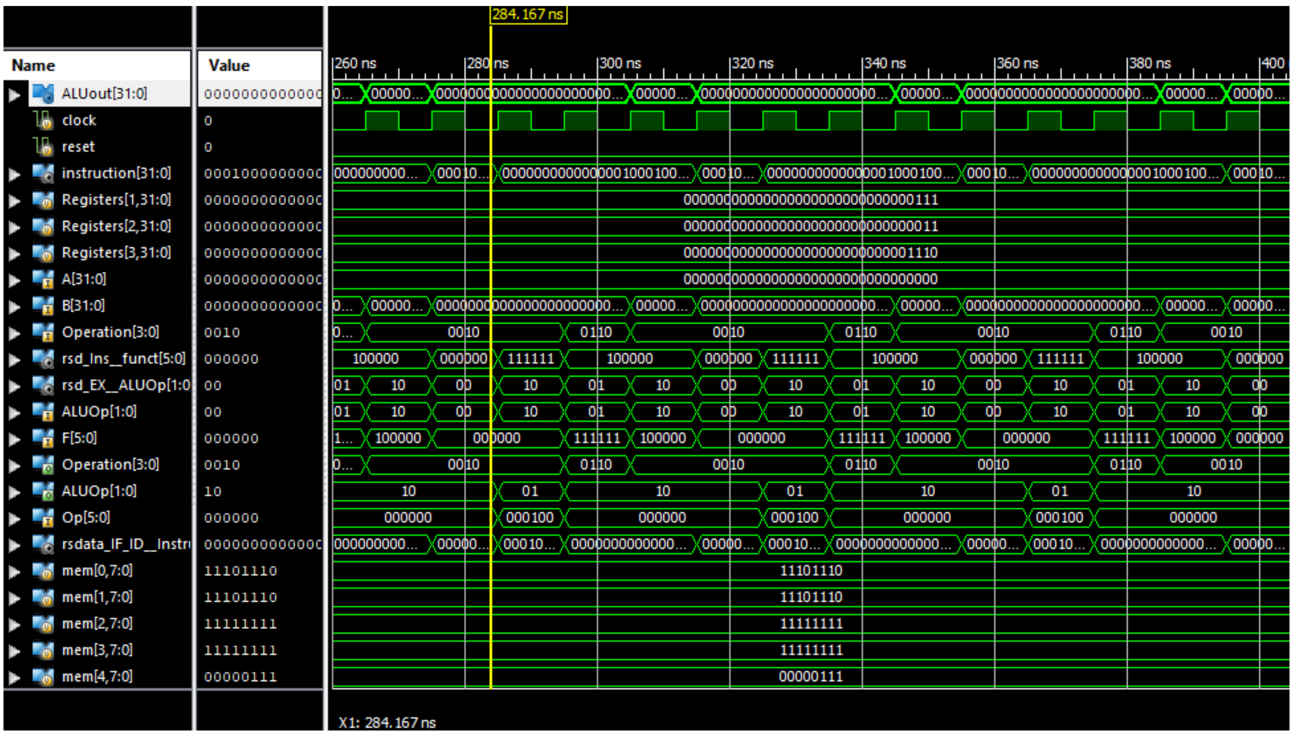


Name	Value
ALUout[31:0]	XXXXXXXXXXXXX
clock	0
reset	1
instruction[31:0]	XXXXXXXXXXXXX
Registers[1,31:0]	XXXXXXXXXXXXX
Registers[2,31:0]	XXXXXXXXXXXXX
Registers[3,31:0]	XXXXXXXXXXXXX
A[31:0]	XXXXXXXXXXXXX
B[31:0]	XXXXXXXXXXXXX
Operation[3:0]	XXXX
rsd_Ins__funct[5:0]	XXXXXX
rsd_EX__ALUOp[1:0]	XX
ALUOp[1:0]	XX
F[5:0]	XXXXXX
Operation[3:0]	XXXX
ALUOp[1:0]	XX
Op[5:0]	XXXXXX
rsdata_IF_ID__Instr	XXXXXXXXXXXXX
mem[0,7:0]	11101110
mem[1,7:0]	11101110
mem[2,7:0]	11111111
mem[3,7:0]	11111111
mem[4,7:0]	00001110



Name	Value
ALUout[31:0]	0000000000000
clock	0
reset	0
instruction[31:0]	1000110010000
Registers[1,31:0]	0000000000000
Registers[2,31:0]	0000000000000
Registers[3,31:0]	1111111111111
A[31:0]	0000000000000
B[31:0]	0000000000000
Operation[3:0]	1111
rsd_Ins__funct[5:0]	000101
rsd_EX__ALUOp[1:0]	10
ALUOp[1:0]	10
F[5:0]	101001
Operation[3:0]	1111
ALUOp[1:0]	00
Op[5:0]	000001
rsdata_IF_ID__Instr	0000010000100
mem[0,7:0]	11101110
mem[1,7:0]	11101110
mem[2,7:0]	11111111
mem[3,7:0]	11111111
mem[4,7:0]	00001110

X1: 154.167 ns



```
Name                    Value
ALUout[31:0]            0000000000000
clock                   0
reset                   0
instruction[31:0]       0001000000000
Registers[1,31:0]       0000000000000
Registers[2,31:0]       0000000000000
Registers[3,31:0]       0000000000000
A[31:0]                 0000000000000
B[31:0]                 0000000000000
Operation[3:0]          0010
rsd_Ins__funct[5:0]     000000
rsd_EX__ALUOp[1:0]      00
ALUOp[1:0]              00
F[5:0]                  000000
Operation[3:0]          0010
ALUOp[1:0]              10
Op[5:0]                 000000
rsdata_IF_ID__Instr     0000000000000
mem[0,7:0]              11101110
mem[1,7:0]              11101110
mem[2,7:0]              11111111
mem[3,7:0]              11111111
mem[4,7:0]              00000111
```