COEN122 Final Project Report

Yen-Jung Lu, Travis Lee

Abstract

This Final project is aiming to design a 32-bit datapath structure and create a pipelined CPU with the datapath . We will have several instructions to input in each module, and the datapath represents how this procedure is processed. The instruction set includes 12 instructions in assembly language that need to be implemented at a hardware level. With pipelining, we are able to connect all our modules together and implement our instructions.

Instruction	OpCode	RegWrite	MemToReg	PCToReg	BranchN	BranchZ	Jump	JumpMem	MemRead	MemWrite
No Operation (nop)	0000	0	0	0	0	0	0	0	0	0
Save PC	1111	1	0	1	0	0	0	0	0	0
Load	1110	1	1	0	0	0	0	0	1	0
Store	0011	0	0	0	0	0	0	0	0	1
Add (R-type)	0100	1	0	0	0	0	0	0	0	0
Increment (R-type)	0101	1	0	0	0	0	0	0	0	0
Negate (R-type)	0110	1	0	0	0	0	0	0	0	0
Subtract (R-type)	0111	1	0	0	0	0	0	0	0	0
Jump	1000	0	0	0	0	0	1	0	0	0
Branch if Zero	1001	0	0	0	0	1	0	0	0	0
Jump Memory	1010	0	0	0	0	0	0	1	1	0
Branch if Negative	1011	0	0	0	1	0	0	0	0	0
SUM	0001									

Description of CPU

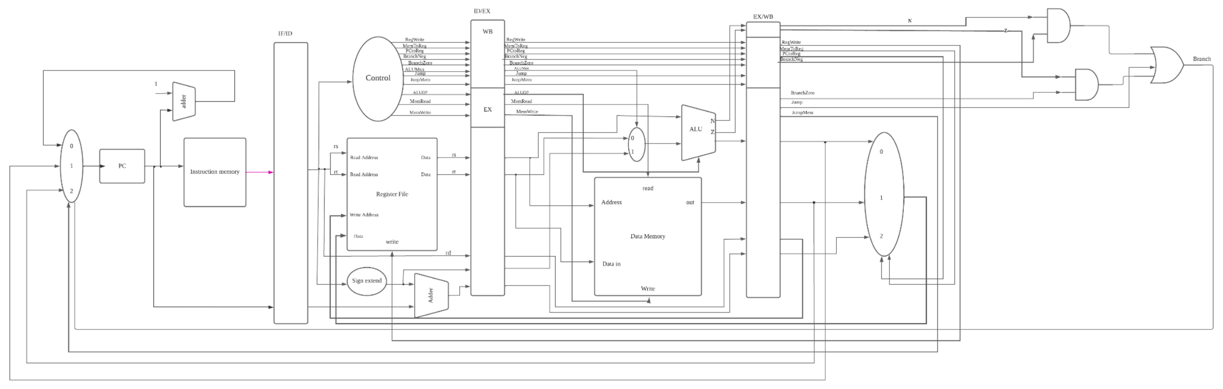


Fig.1: Our Design of Datapath

The datapath is controlled by the Clock, which updates wires and data of the system. We have four pipeline stages in our datapath, which are: IF, ID, EX, WB. Our CPU will start with the instruction fetch(IF) stage first. In this stage, it will hold the program counter, which will keep track of the instruction memory, which has all the instruction sets we need to implement in the project. As the instruction is fetched, we will have our instructions to pass to the next stage.

In the ID stage, we have a Register file module that we will pass in the register values of rs and rt from the instruction memory. The Register file will also have input from the EX/WB buffer, that is the write back value of the rd. Control module is another thing that is located in the ID stage, which can allow us to select which function we want to perform by inputting the opcode. In this stage, instructions are decoded and addresses are added in the register file.

The ID stage performs all the major operations of the CPU and also the operations for data memory. The ALU module in the ID stage calculates all the arithmetic that the CPU needs. ALU will also determine the value of N flag and Z flag, which are needed to evaluate for branching in the next stage. Moreover, we also have our Data Memory located in the EX stage, which is the main memory storage for the CPU.

The WB stage contains a Mux that has inputs from ALU, Data Memory, and Sign Extend Adder. The output of the Mux will be written back as the Data to the Register File. In addition, we have a Logic Gate set at the WB stage, which has N flag, Z flag, BranchNeg, and BranchZero as inputs. And the output of the Logic Gate set will determine what value will pass in to the Mux before our PC, which allows us to determine what value to pass in to the PC.

Assembly Code for Sum Calculation

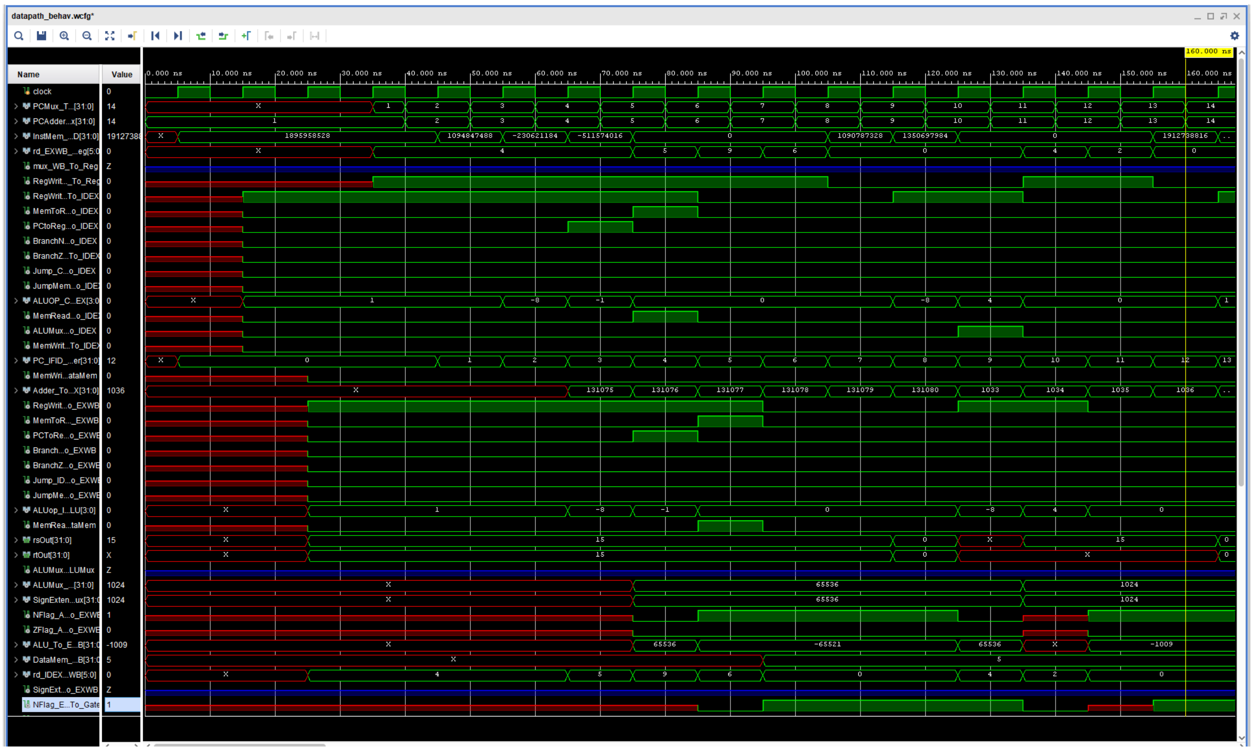
```
assign instr[0] = 32'b01110001000000100000100000000000; //SUB x4, x4,x4
assign instr[1] = 32'b01000001010000100000110000000000; // ADD x5,x2,x3
assign instr[2] = 32'b11110010010000010000000000000000; //SVPC x9,1
assign instr[3] = 32'b11100001100000100000000000000000; //LOAD x6,x2  (start of loop)
assign instr[4] = 32'b00000000000000000000000000000000; //NOP
assign instr[5] = 32'b00000000000000000000000000000000; //NOP
assign instr[6] = 32'b00000000000000000000000000000000; //NOP
assign instr[7] = 32'b01000001000001000001100000000000;  //ADD x4,x4,x6
assign instr[8] = 32'b01010000100000100000010000000000;   //INC x2,x2,x1
assign instr[9] = 32'b00000000000000000000000000000000;   //NOP
assign instr[10] = 32'b00000000000000000000000000000000;   //NOP
```

```
    assign instr[11] = 32'b00000000000000000000000000000000;   //NOP
    assign instr[12] = 32'b01110010000000100001010000000000;   //SUB x8,x2,x5
    assign instr[13] = 32'b10110010010000000000000000000000;   //BRN x9
    assign instr[14] = 32'b10010010010000000000000000000000;   //BRZ x9
```

```
    assign instr[0] = 32'b11110000010000000000000001100100; //SVPC x1, 0x100
    assign instr[1] = 32'b01010000100000010000000000000000; //Inc, x2, x1
    assign instr[2] = 32'b01100000110000010000000000000000; //Neg x3, x1
    assign instr[3] = 32'b11110010100000000000000000000110; //SVPC x10, 6(instr# of ST - 
instr# of this instr)
    assign instr[4] = 32'b10110000000010100000000000000000; //BRN x10
    assign instr[5] = 32'b00000000000000000000000000000000; //NOP
    assign instr[6] = 32'b00000000000000000000000000000000; //NOP
    assign instr[7] = 32'b00000000000000000000000000000000; //NOP
    assign instr[8] = 32'b01010000100000100000000000000000; //Inc x2, x2
    assign instr[9] = 32'b00110000010000010000000000000000; //ST x1, x1
    assign instr[10] = 32'b11100001000000001000000000000000; //Load x4, x1
    assign instr[11] = 32'b01000001010000001000010000000000;    //Add x5, x1, x2
    assign instr[12] = 32'b01110001100001000000000100000000;    //SUB x6, x4, x1
    assign instr[13] = 32'b11110010110000000000000000000011;    //SVPC x11, 3 (16-13, we 
want to get to 16)
    assign instr[14] = 32'b10010010110000000000000000000000;    //BRZ x11
    assign instr[15] = 32'b01010000100000100000000000000000;    //Inc x2, x2
    assign instr[16] = 32'b10100000010000000000000000000000;    //JM x1 (JumpMem)
    assign instr[17] = 32'b00000000000000000000000000000000;    //NOP
    assign instr[18] = 32'b00000000000000000000000000000000;    //NOP
    assign instr[19] = 32'b00000000000000000000000000000000;    //NOP
    assign instr[20] = 32'b10000000010000000000000000000000;    //J x1 (Jump)
```

Waveform for Sum Calculation Assembly Code

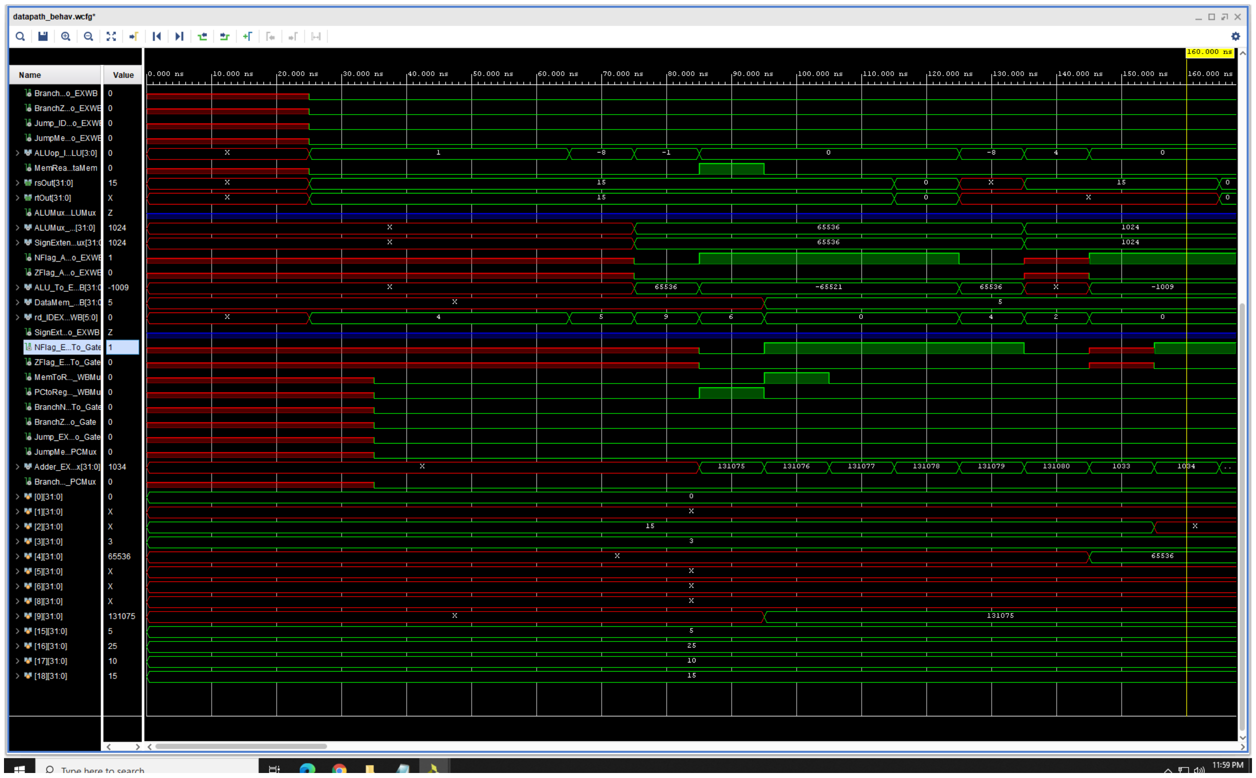
Top of Start of Waveform



datapath_behav.wcfg*

Name	Value
clock	0
PCMux_T...[31:0]	14
PCAdder...x[31:0]	14
InstMem_...D[31:0]	19127388
rd_EXWB_...eg[5:0	0
mux_WB_To_Reg	Z
RegWrit..._To_Reg	0
RegWrit...To_IDEX	0
MemToR...o_IDEX	0
PCtoReg...o_IDEX	0
BranchN...o_IDEX	0
BranchZ...To_IDEX	0
Jump_C...o_IDEX	0
JumpMem...o_IDEX	0
ALUOP_C...EX[3:0	0
MemRead...o_IDEX	0
ALUMux...o_IDEX	0
MemWrit...To_IDEX	0
PC_IFID_...er[31:0]	12
MemWri...ataMem	0
Adder_To...X[31:0]	1036
RegWrit...o_EXWB	0
MemToR..._EXWB	0
PCToRe...o_EXWB	0
Branch...o_EXWB	0
BranchZ...o_EXWB	0
Jump_ID...o_EXWB	0
JumpMe...o_EXWB	0
ALUop_I...LU[3:0]	0
MemRea...taMem	0
rsOut[31:0]	15
rtOut[31:0]	X
ALUMux...LUMux	Z
ALUMux_...[31:0]	1024
SignExten...ux[31:0	1024
NFlag_A...o_EXWB	1
ZFlag_A...o_EXWB	0
ALU_To_E...B[31:0	-1009
DataMem_...B[31:0	5
rd_IDEX...WB[5:0]	0
SignExt...o_EXWB	Z
NFlag_E...To_Gate	1

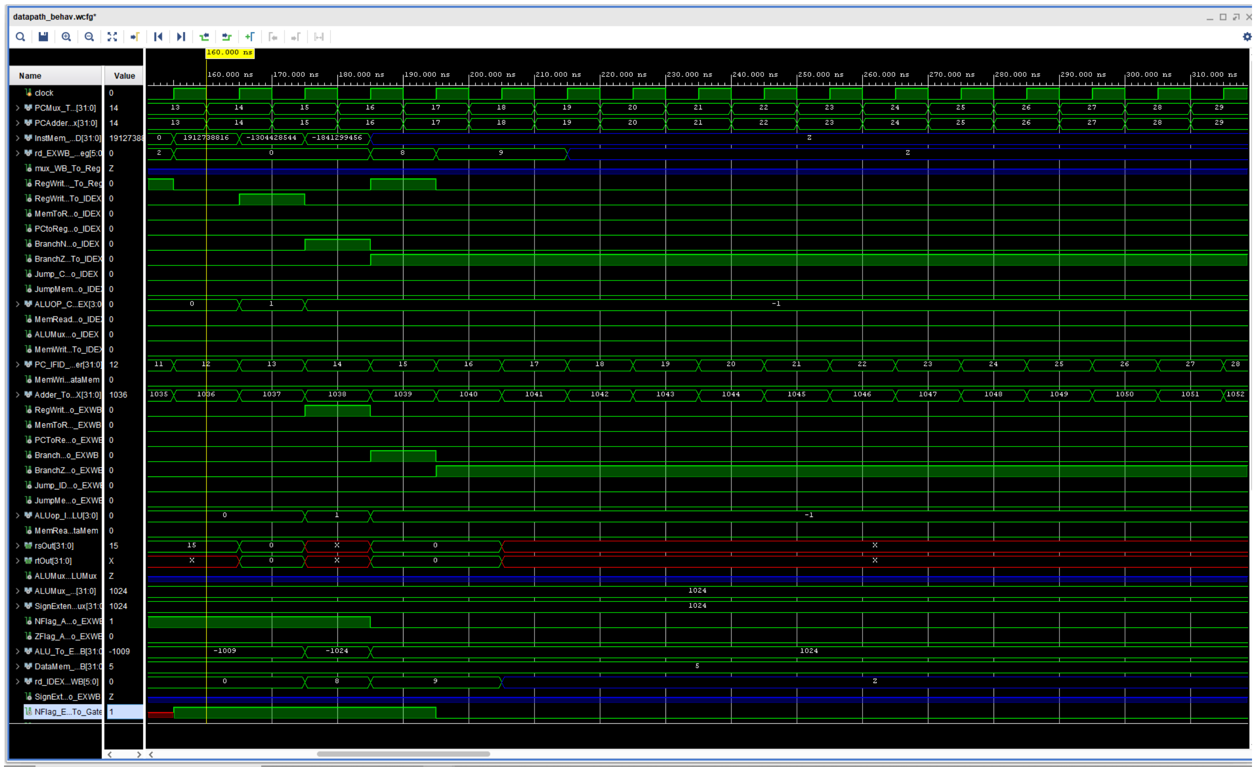
Bottom of start of waveform



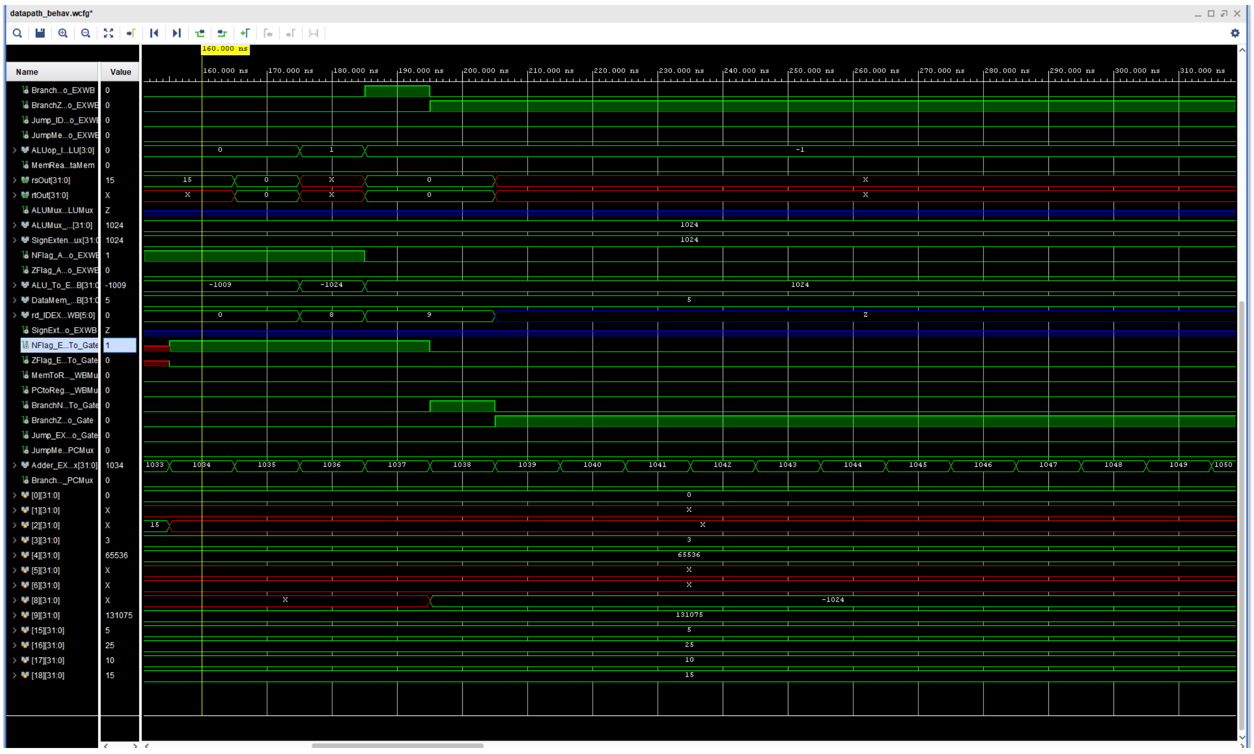
datapath_behav.wcfg*

Name	Value
Branch...o_EXWB	0
BranchZ...o_EXWB	0
Jump_ID...o_EXWB	0
JumpMe...o_EXWB	0
ALUop_I...LU[3:0]	0
MemRea...taMem	0
rsOut[31:0]	15
rtOut[31:0]	X
ALUMux...LUMux	Z
ALUMux_...[31:0]	1024
SignExten...ux[31:0	1024
NFlag_A...o_EXWB	1
ZFlag_A...o_EXWB	0
ALU_To_E...B[31:0	-1009
DataMem_...B[31:0	5
rd_IDEX...WB[5:0]	0
SignExt...o_EXWB	Z
NFlag_E...To_Gate	1
ZFlag_E...To_Gate	0
MemToR..._WBMux	0
PCtoReg..._WBMux	0
BranchN...To_Gate	0
BranchZ...o_Gate	0
Jump_EX...o_Gate	0
JumpMe...PCMux	0
Adder_EX...x[31:0]	1034
Branch..._PCMux	0
[0][31:0]	0
[1][31:0]	X
[2][31:0]	X
[3][31:0]	3
[4][31:0]	65536
[5][31:0]	X
[6][31:0]	X
[8][31:0]	X
[9][31:0]	131075
[15][31:0]	5
[16][31:0]	25
[17][31:0]	10
[18][31:0]	15

Top of end of waveform

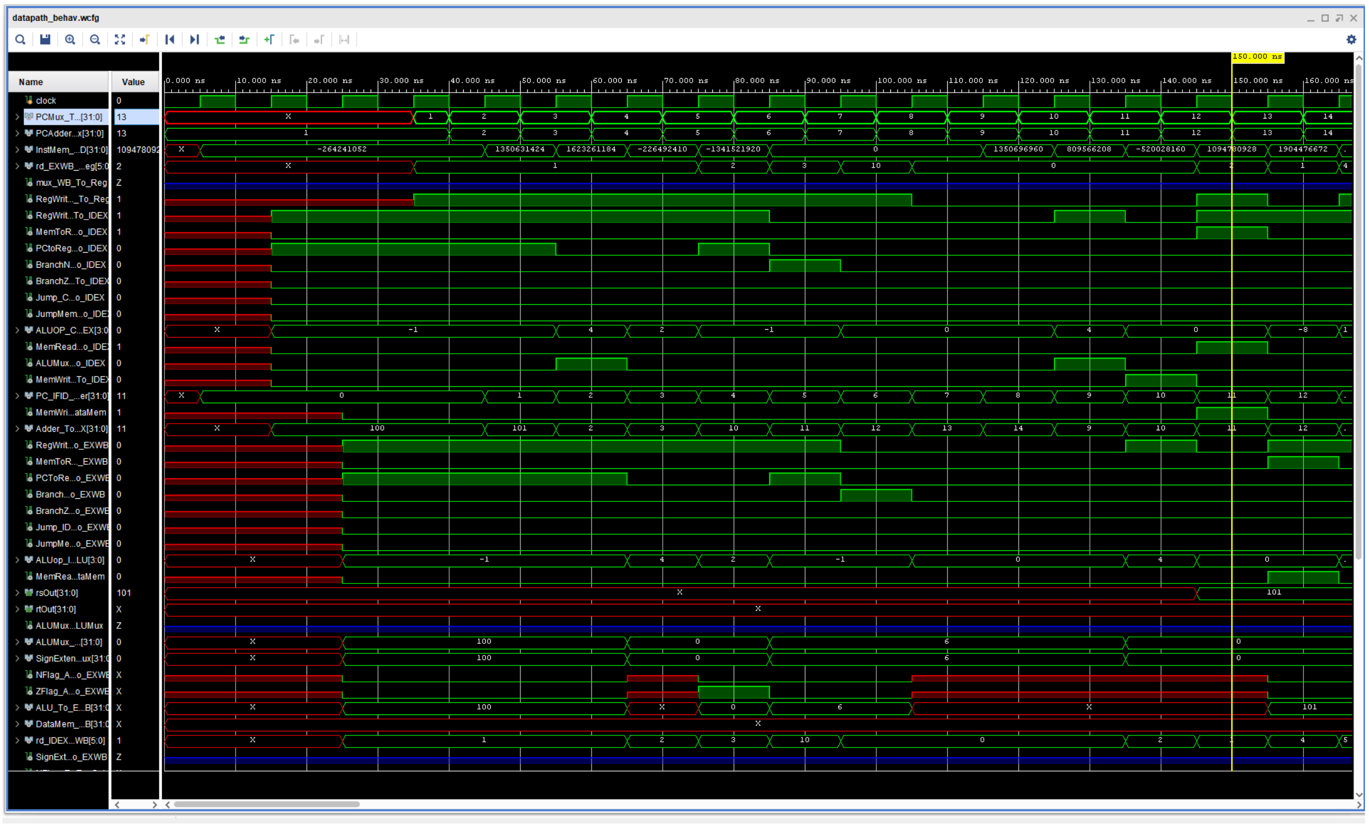


Bottom of end of waveform



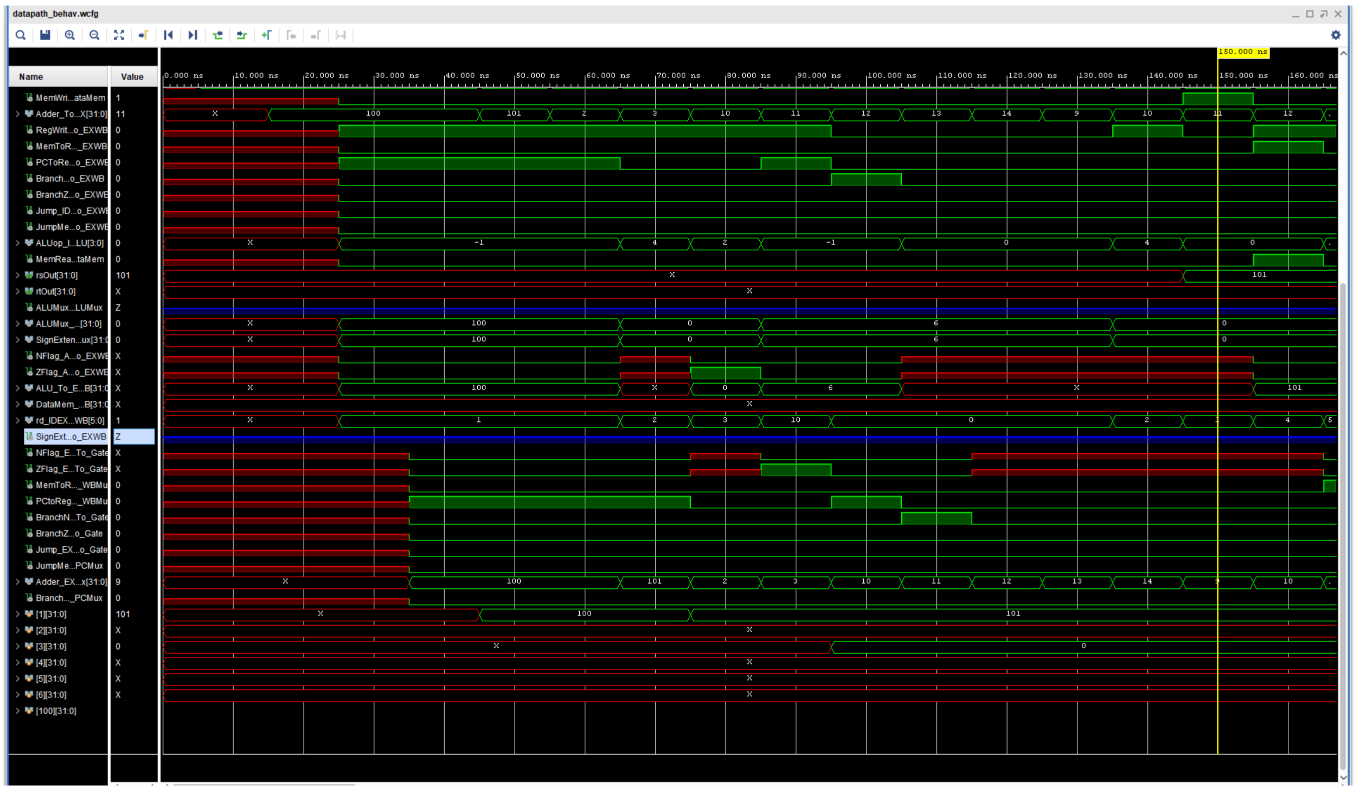
Waveform for Instruction Test Assembly Code

Top of start of waveform

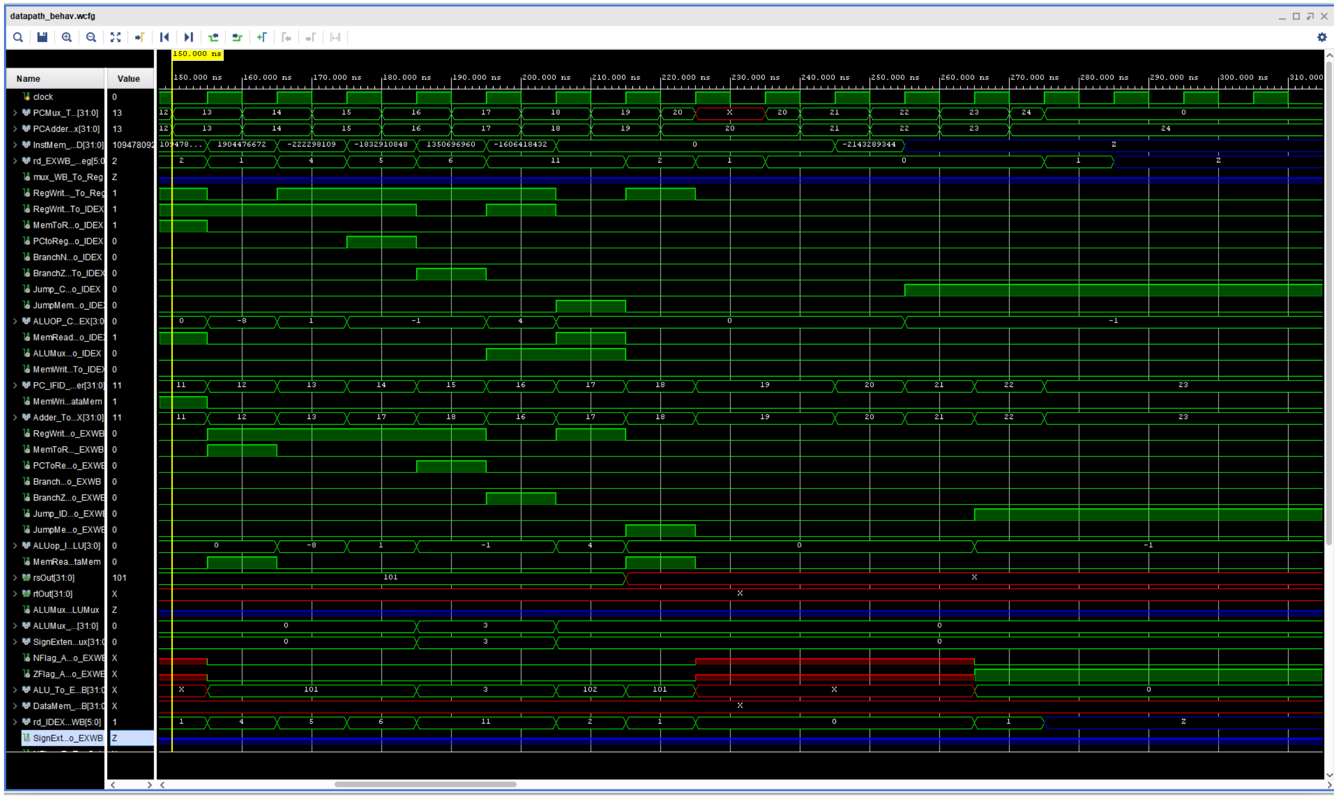

datapath_behav.wcfg

Name	Value
clock	0
PCMux_T...[31:0]	13
PCAdder...x[31:0]	13
InstMem_...D[31:0]	109478092
rd_EXWB_...eg[5:0	2
mux_WB_To_Reg	Z
RegWrit..._To_Reg	1
RegWrit...To_IDEX	1
MemToR...o_IDEX	1
PCtoReg...o_IDEX	0
BranchN...o_IDEX	0
BranchZ...To_IDEX	0
Jump_C...o_IDEX	0
JumpMem...o_IDEX	0
ALUOP_C...EX[3:0	0
MemRead...o_IDEX	1
ALUMux...o_IDEX	0
MemWrit...To_IDEX	0
PC_IFID_...er[31:0]	11
MemWri...ataMem	1
Adder_To...X[31:0]	11
RegWrit...o_EXWB	0
MemToR..._EXWB	0
PCToRe...o_EXWB	0
Branch...o_EXWB	0
BranchZ...o_EXWB	0
Jump_ID...o_EXWB	0
JumpMe...o_EXWB	0
ALUop_I...LU[3:0]	0
MemRea...taMem	0
rsOut[31:0]	101
rtOut[31:0]	X
ALUMux...LUMux	Z
ALUMux_...[31:0]	0
SignExten...ux[31:0	0
NFlag_A...o_EXWB	X
ZFlag_A...o_EXWB	X
ALU_To_E...B[31:0	X
DataMem_...B[31:0	X
rd_IDEX...WB[5:0]	1
SignExt...o_EXWB	Z

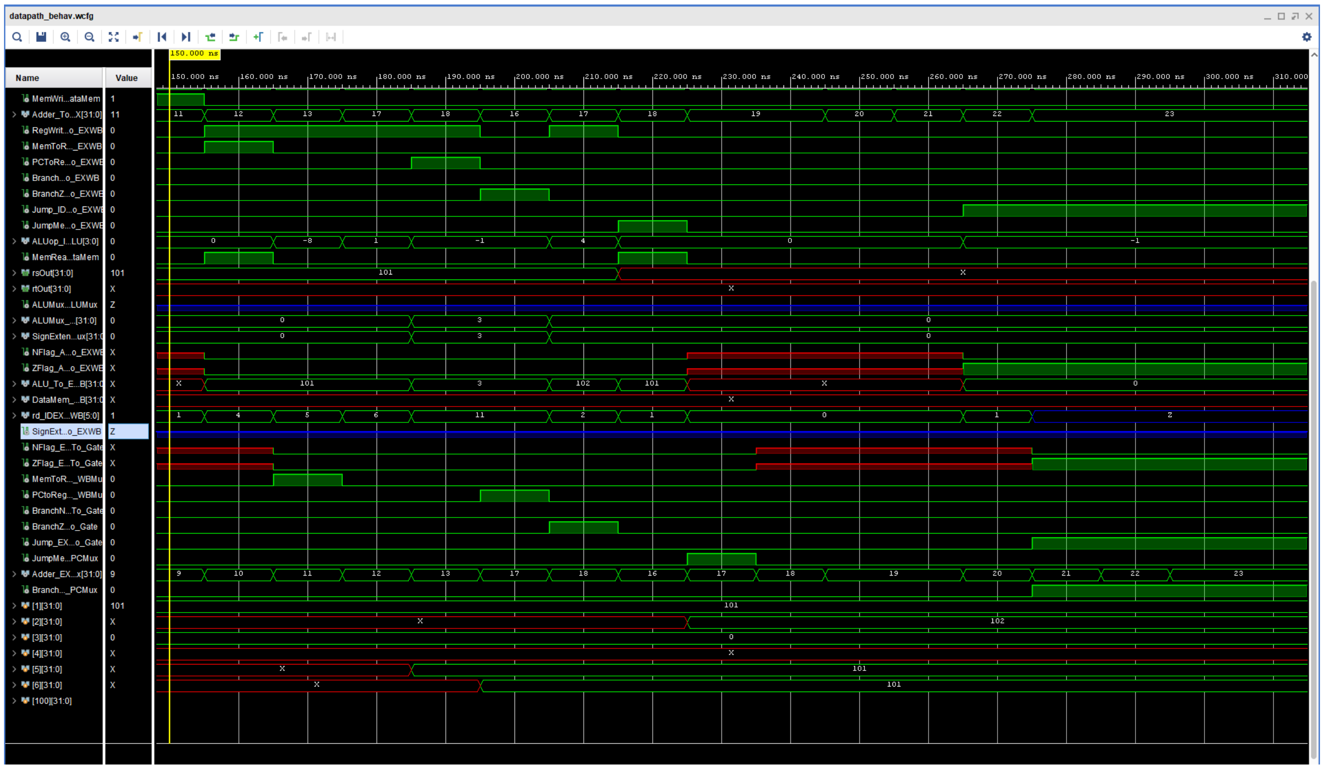
Bottom of start of waveform



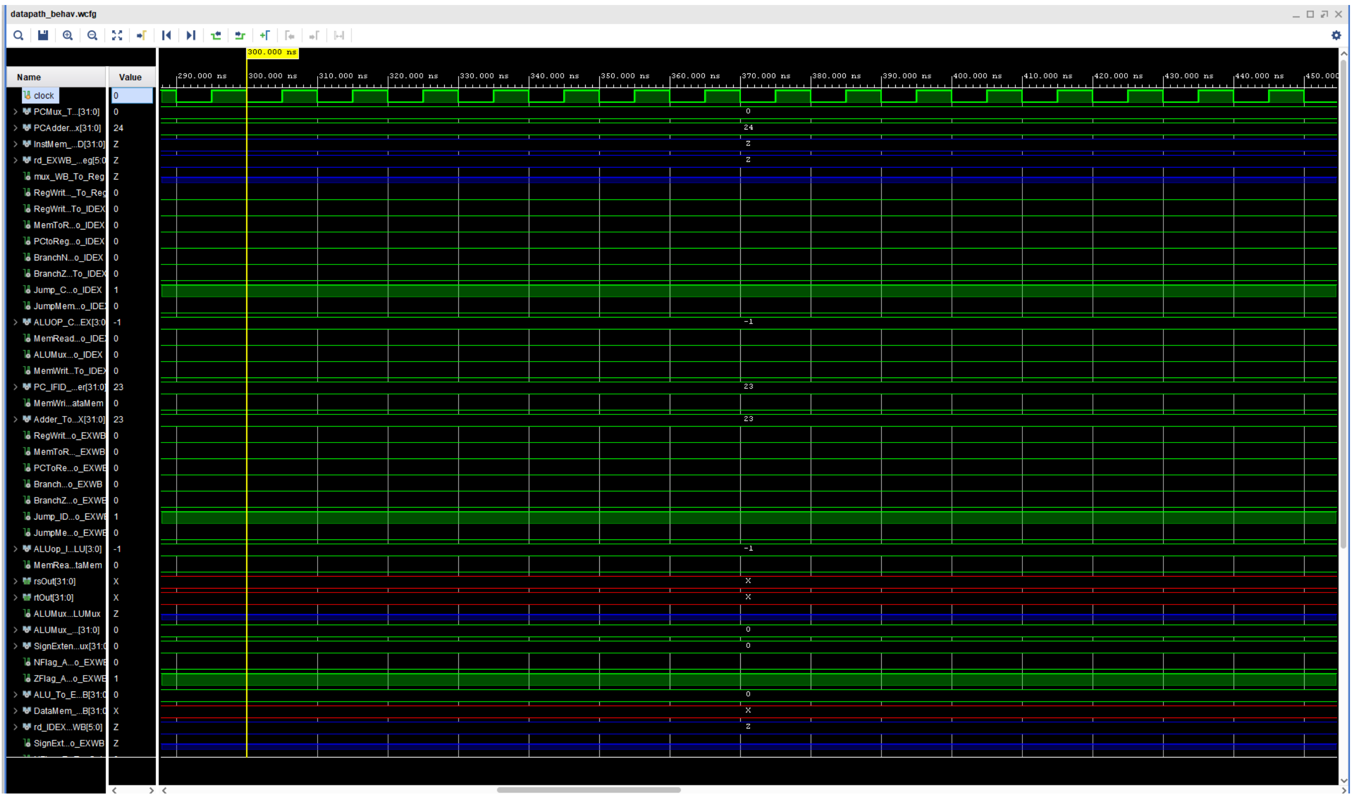
Top of middle of waveform



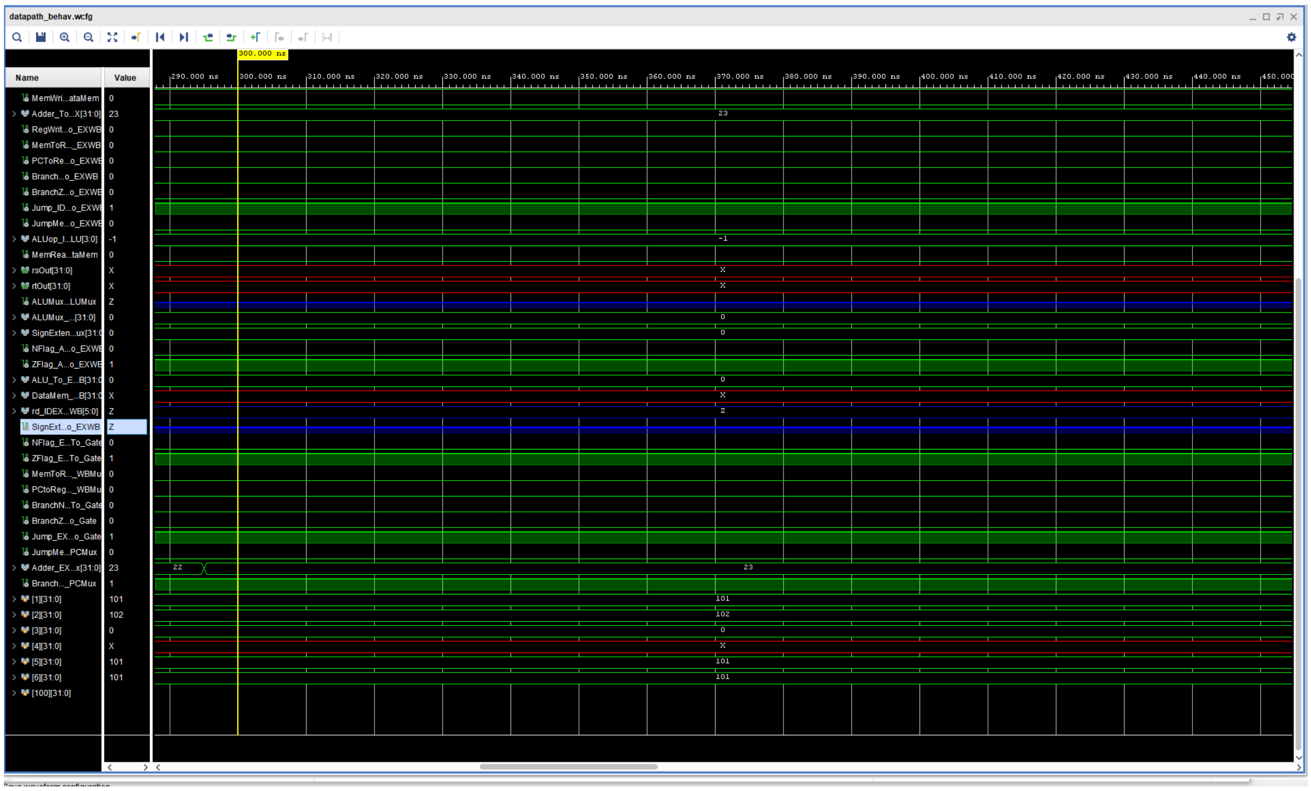
Bottom of middle of waveform



Top of end of waveform



Bottom of end of waveform


datapath_behav.wcfg

Name	Value
MemWri...ataMem	0
Adder_To...X[31:0]	23
RegWrit...o_EXWB	0
MemToR..._EXWB	0
PCToRe...o_EXWB	0
Branch...o_EXWB	0
BranchZ...o_EXWB	0
Jump_ID...o_EXWB	1
JumpMe...o_EXWB	0
ALUop_I...LU[3:0]	-1
MemRea...taMem	0
rsOut[31:0]	X
rtOut[31:0]	X
ALUMux...LUMux	Z
ALUMux_...[31:0]	0
SignExten...ux[31:0	0
NFlag_A...o_EXWB	0
ZFlag_A...o_EXWB	1
ALU_To_E...B[31:0	0
DataMem_...B[31:0	X
rd_IDEX...WB[5:0]	Z
SignExt...o_EXWB	Z
NFlag_E...To_Gate	0
ZFlag_E...To_Gate	1
MemToR..._WBMu	0
PCtoReg..._WBMu	0
BranchN...To_Gate	0
BranchZ...o_Gate	0
Jump_EX...o_Gate	1
JumpMe...PCMux	0
Adder_EX...x[31:0]	23
Branch..._PCMux	1
[1][31:0]	101
[2][31:0]	102
[3][31:0]	0
[4][31:0]	X
[5][31:0]	101
[6][31:0]	101
[100][31:0]	

Estimate of Execution time

When you analyze the cycle time, you can use the following delay data: delay of memory

(I and D memory): 2 ns., delay of register file: 1.5 ns., delay of ALU (adders): 2 ns.

Ignore the delays of all other components.

1(2ns) +1(2ns) +1(1.5ns) +3(2ns) = 11.5ns

Actual time: 185ns