FemtoRV32

Omar Elfouly
Bavly Remon

November 26, 2023

Abstract

In this project we are creating a miniature version of an RV32IM Processor. In this final milestone we implement a 5 stage pipelined RV32IM processor. In this paper we will discuss the abillities and limitations of our processor, aswell as the whole development cycle including designing and testing.

1 Introduction

In this project we were tasked to implement a 5 stage pipeline RV32 processor that supports all 40 RISC-V instructions that are included in figure 1 as well as all multiplication and division instructions such that we support the full RV32IM instruction set. They are all implemented according to the specification document provided. There is an exception however in the ECALL, EBREAK and FENCE instructions such that ECALL and FENCE would be interpreted as a pass or a NOP instruction while EBREAK would halt execution of the problem such that the program counter wouldn't increment anymore. Unlike the previous milestone we implemented a single memory for both data and instructions.

2 Development

2.1 Design and Implementation

We began this final milestone with the previous milestone that supported all 40 instructions but was single cycled 2. We went ahead and pipelined that previous design by adding a register between each stage (IF/ID, ID/EX, EX/MEM and MEM/WB). We then tested this design keeping in mind that it doesnt handle any kind of errors, those mainly being data dependencies, load use hazards and branch hazards. We can see this design in this diagram below 3

After making sure our design was pipelined successfully, we started working on handling the aforementioned hazards. We started by implementing a forward unit that supports EX/MEM to execute forwarding and MEM/WB to execute as well. We also decided to support MEM to MEM forwarding to save one clock cycle when a Load/Store hazard shows up by not stalling. We then implemented the Hazard detection unit and all the stalling it creates, its worth mentioning that we removed the hazard detection unit's ability to detect a load/store hazard to avoid stalling as we can forward mem to mem.

After testing this design that is pipelined and handles all hazards, we moved onto implementing the single memory. We decided against the slow clock/ fast clock implementation as it greatly decreases the performance and to also statisfy the bonus requirment. We designed our cpu such that it stalls the fetching phase whenever an instruction using memory is in the mem stage, i.e we don't use the memory to fetch in that clock cycle and just pass a nop instead. We also Support M instruction which was also another bonus requirement.

We approached this milestone in a more agile methodology where the documentation and implementation were done in parallel, such that we updated the diagrams whenever we implemented a new part of the code. That's why all code commits where done by Omar Elfouly while all diagram commits were done by Bavly Remon as we 2 devices open with both accounts to be able to achieve a more efficient work flow.

2.2 Challenges and Solutions

One of the unexpexted issues we faced was with implementing the single memory. While the initial idea seemed simple enough, we didnt account for all the changes we needed to implement when changing from 2 seperate memories to a single one. So, figuring out all the changes we needed to make came through testing and debugging was a bit difficult due to the pipeline, that means that even though we had the previous milestone to compare to, it still posed a challenge due to the pipelineing.

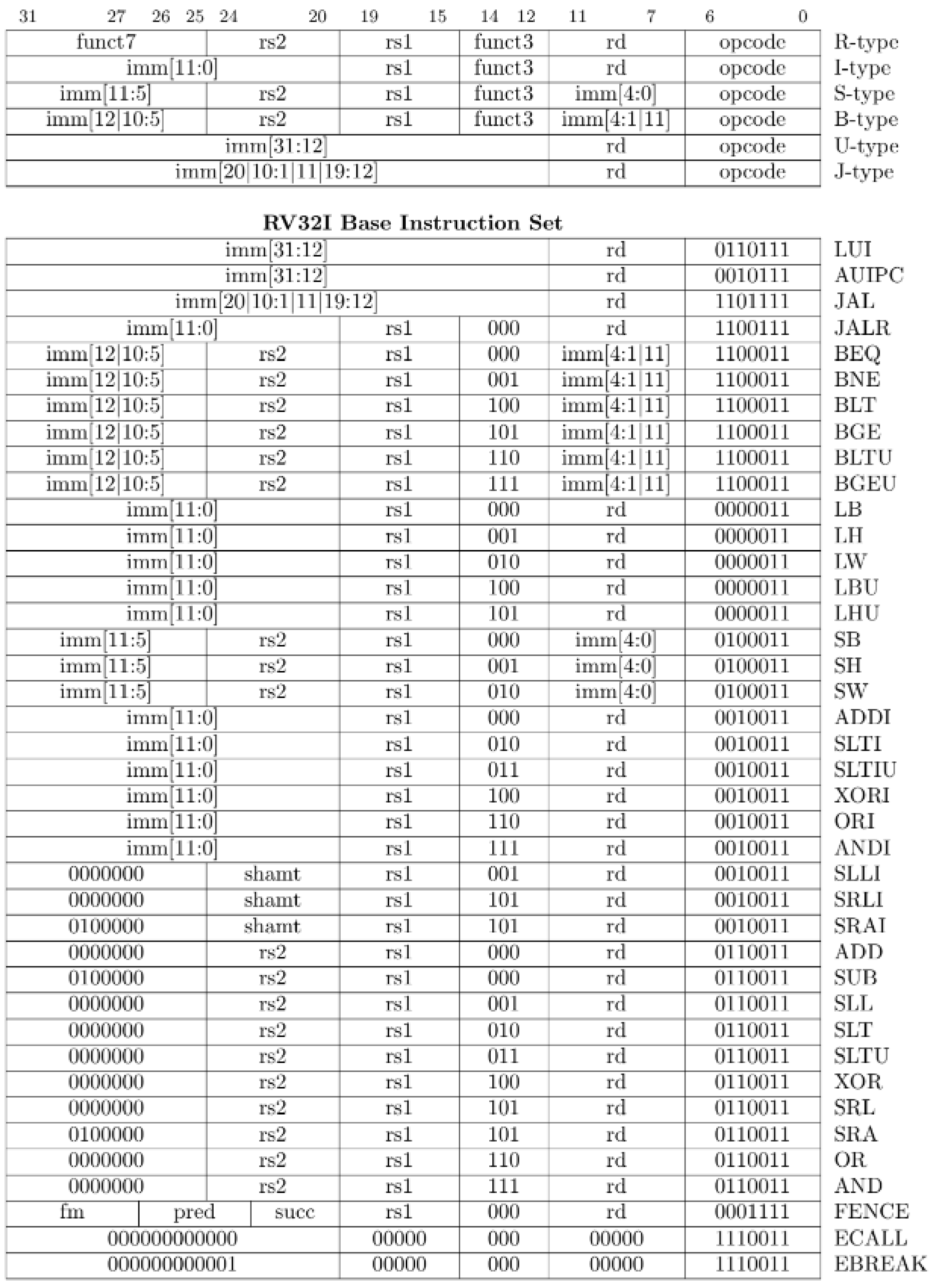
3 Testing

To test our processor, we decided to take a more rigorous approach and test each instruction individually much like the last milestone. We have included our Instruction memory that contains all the test cases that we used along with comments of the expected outputs. It will also be included in the Submission for easier viewing. We also used a small program that was also used in the lab to test all the hazards and compared our results with those we got in the lab.

```
initial begin
// LUI test cases
        {mem[3],mem[2],mem[1],mem[0]}=32'b00000000000001100100000010110111; //lui x1, 100      x1 = 100 << 12
        {mem[7],mem[6],mem[5],mem[4]}=32'b11111111111110011100000100110111; // lui x2,-100     x2 = -100 << 12
        {mem[11],mem[10],mem[9],mem[8]}=32'b00000000000000000000000110110111; //lui x3, 0      x3 = 0 << 12

        //AUIPC
        {mem[15],mem[14],mem[13],mem[12]}=   32'b00000000000000000001010010010111;//auipc x9, 1      x9 = 12 + 1<<12
        {mem[19],mem[18],mem[17],mem[16]} = 32'b00000000000000000000001000010111; //auipc x4, 0    x4 = 16 + 0<< 12
        {mem[23],mem[22],mem[21],mem[20]} = 32'b11111111111110011100001000010111; //auipc x4, -100   x4 = 20 + -100 << 1

        //JAL
        {mem[27],mem[26],mem[25],mem[24]} = 32'b00000000100000000000001011101111; // jal x5, 8    GOTO 32 x5 = 28
        {mem[31],mem[30],mem[29],mem[28]} = 32'b00000000100000000000001011101111; //jal x5, 8     GOTO 36 x5 = 32
        {mem[35],mem[34],mem[33],mem[32]} = 32'b11111111110111111111001011101111; //jal x5, -4    GOTO 28 x5 = 36



        //JALR
        {mem[39],mem[38],mem[37],mem[36]} = 32'b00000000110000101000001101100111;//jalr x6, 12(x5)   GOTO 44
x6 = 40
        {mem[47],mem[46],mem[45],mem[44]} = 32'b00000000000000110000001111100111;//jalr x7, 0(x6)    GOTO 40
x7 = 48
        {mem[43],mem[42],mem[41],mem[40]} = 32'b00000000000000111000010001100111;//jalr x8, 0(x7)    GOTO 48
x8 = 44

        //BEQ
        {mem[51],mem[50],mem[49],mem[48]} = 32'b00000000011100110000011001100011;//beq x6, x7, 12    IGNORE
        {mem[55],mem[54],mem[53],mem[52]} = 32'b00000000000000000000010001100011;//beq x0, x0, 8     GOTO 60

        {mem[59],mem[58],mem[57],mem[56]} = 32'b00000000000000000000000000110011;//NOP

        //BNE
        {mem[63],mem[62],mem[61],mem[60]} = 32'b00000000000000000001110001100011;//bne x0, x0, 24    IGNORE
        {mem[67],mem[66],mem[65],mem[64]} = 32'b00000000011100110001010001100011;//bne x6, x7, 8     GOTO 72

        {mem[71],mem[70],mem[69],mem[68]} = 32'b00000000000000000000000000110011;//NOP

        //BLT
        {mem[75],mem[74],mem[73],mem[72]} = 32'b00000000000000000100110001100011;//blt x0, x0, 24   IGNORE
        {mem[79],mem[78],mem[77],mem[76]} = 32'b00000000011101000100010001100011;//blt x8, x7, 8    GOTO 84

        {mem[83],mem[82],mem[81],mem[80]} = 32'b00000000000000000000000000110011;//NOP

        //BGE
        {mem[87],mem[86],mem[85],mem[84]} = 32'b00000110011101000101001001100011;//bge x8, x7, 100   IGNORE
        {mem[91],mem[90],mem[89],mem[88]} = 32'b00000000000000000101011001100011;//bge x0, x0, 12    GOTO 100

        {mem[95],mem[94],mem[93],mem[92]} = 32'b00000000000000000000000000110011;//NOP
        {mem[99],mem[98],mem[97],mem[96]} = 32'b00000000000000000000000000110011;//NOP

        //BLTU
        {mem[103],mem[102],mem[101],mem[100]} = 32'b00000110010100100110001001100011;//bltu x4, x5, 100
IGNORE
        {mem[107],mem[106],mem[105],mem[104]} = 32'b00000000010000101110010001100011;//bltu x5, x4, 8
GOTO 112

        {mem[111],mem[110],mem[109],mem[108]} = 32'b00000000000000000000000000110011;//NOP

        //BGEU
        {mem[115],mem[114],mem[113],mem[112]} = 32'b11111110010000101111110011100011;//bgeu x5, x4, -8
IGNORE
        {mem[119],mem[118],mem[117],mem[116]} = 32'b00000000010100100111010001100011;//bgeu x4, x5, 8
GOTO 124
        {mem[127],mem[126],mem[125],mem[124]} = 32'b11111110010100100111111011100011;//bgeu x4,x5,-4
GOTO 120
        {mem[123],mem[122],mem[121],mem[120]} = 32'b00000000010100100111010001100011;//bgeu x4,x5,8
GOTO 128


        //Memory (store and load)
        {mem[131],mem[130],mem[129],mem[128]} = 32'b11111110010100101000000010100011;//sb x5, -31(x5)
mem[1] = 32
        {mem[135],mem[134],mem[133],mem[132]} = 32'b00000000100100101001000000100011;//sh x9, 0(x5)
mem [33:32] = 4108
        {mem[139],mem[138],mem[137],mem[136]} = 32'b00000000000100101010001000100011;//sw x1, 4(x5)
mem[39:36] = 409600

        {mem[143],mem[142],mem[141],mem[140]} = 32'b11111110000100101000010100000011;//lb x10, -31(x5)
x10 =32
        {mem[147],mem[146],mem[145],mem[144]} = 32'b00000000000000101001010110000011;//lh x11, 0(x5)
x11 = 4108
        {mem[151],mem[150],mem[149],mem[148]} = 32'b00000000010000101010011000000011;//lw x12, 4(x5)
x12 = 409600

        //I-type

        //ADDI
        {mem[155],mem[154],mem[153],mem[152]} = 32'b00000100010100000000111100010011; //addi x30, x0, 69
x30 = 69
        {mem[159],mem[158],mem[157],mem[156]} = 32'b11111111111100000000111100010011; //addi x30, x0, -1
x30 = -1
        {mem[163],mem[162],mem[161],mem[160]} = 32'b00000000000000101000111100010011; //addi x30, x5, 0
x30 = 32 + 0

        //SLTI
        {mem[167],mem[166],mem[165],mem[164]} = 32'b00000010000000101010111010010011; //slti x29, x5, 32
false
        {mem[171],mem[170],mem[169],mem[168]} = 32'b00000100010100101010111010010011; //slti x29, x5, 69
true
        {mem[175],mem[174],mem[173],mem[172]} = 32'b00000011000000111010111010010011; //slti x29, x7, 48
false
        {mem[179],mem[178],mem[177],mem[176]} = 32'b11111111111100100010111010010011; //slti x29, x4, -1
true


        //SLTIU
        {mem[179],mem[178],mem[177],mem[176]} = 32'b11111111111100101011111000010011;//sltiu x28, x5, -1
true
        {mem[183],mem[182],mem[181],mem[180]} = 32'b00000010000000101011111000010011;//sltiu x28, x5, 32
false
        {mem[187],mem[186],mem[185],mem[184]} = 32'b00000110010000101011111000010011;//sltiu x28, x5, 100 true


        //XORI
        {mem[191],mem[190],mem[189],mem[188]} = 32'b00111110011100001100111110010011;//xori x31, x1, 999
409600 ^ 999 = 410599
        {mem[195],mem[194],mem[193],mem[192]} = 32'b11011101010100001100111110010011;//xori x31, x1, -555
409600 ^ -555 = 4294557141

        //ORI
        {mem[199],mem[198],mem[197],mem[196]} = 32'b00111110100001001110110110010011;//ori x27, x9, 1000
4108 | 1000 = 5100
        {mem[203],mem[202],mem[201],mem[200]} = 32'b11111111111101001110110110010011;//ori x27, x9, -1
4108 | -1 = 4294967295 = -1
```

```
        //ANDI
        {mem[207],mem[206],mem[205],mem[204]} = 32'b11111111111101000111110100010011;//andi x26, x8, -1
44 & -1 = 44
        {mem[211],mem[210],mem[209],mem[208]} = 32'b00000000000001000111110100010011;//andi x26, x8, 0
44 & 0 = 0

        //SLLI
        {mem[215],mem[214],mem[213],mem[212]} = 32'b00000001111100101001110010010011;//slli x25,x5, 31
32 << 31 = 0
        {mem[219],mem[218],mem[217],mem[216]} = 32'b00000000000000101001110010010011;//slli x25,x5, 0
32 << 0 = 32
        {mem[223],mem[222],mem[221],mem[220]} = 32'b00000000001000101001110010010011;//slli x25,x5, 2
32 << 2 = 128

        //SRLI
        {mem[227],mem[226],mem[225],mem[224]} = 32'b00000001111100101101110010010011;//srli x25,x5, 31
32 >> 31 = 0
        {mem[231],mem[230],mem[229],mem[228]} = 32'b00000000000000101101110010010011;//srli x25,x5, 0
32 >> 0 = 32
        {mem[235],mem[234],mem[233],mem[232]} = 32'b00000000001000101101110010010011;//srli x25,x5, 2
32 >> 2 = 8

        //SRAI
        {mem[239],mem[238],mem[237],mem[236]} = 32'b01000001111100010101110010010011;//srai x25,x2, 31
-409600 >>> 31 = -1
        {mem[243],mem[242],mem[241],mem[240]} = 32'b01000000000000010101110010010011;//srai x25,x2, 0
-409600 >>> 0 = -409600
        {mem[247],mem[246],mem[245],mem[244]} = 32'b01000000001000010101110010010011;//srai x25,x2, 2
-409600 >>> 2 = -102400

        //ADD
        {mem[251],mem[250],mem[249],mem[248]} = 32'b00000000000000000000110000110011;//add x24, x0, x0
x24=0
        {mem[255],mem[254],mem[253],mem[252]} = 32'b00000000101001001000110000110011;//add x24, x9, x10
x24=4108+32=4140
        {mem[259],mem[258],mem[257],mem[256]} = 32'b00000000000100010000110000110011;//add x24, x2, x1
x24=409600 + (-409600)=0

        //SUB
        {mem[263],mem[262],mem[261],mem[260]} = 32'b01000000000000000000101110110011;//sub x23, x0, x0
x23=0
        {mem[267],mem[266],mem[265],mem[264]} = 32'b01000000101001001000101110110011;//sub x23, x9, x10
x23=4108-32=4076
        {mem[271],mem[270],mem[269],mem[268]} = 32'b01000000000100010000101110110011;//sub x23, x2, x1
x23=-409600 - 409600=-819200


        //SLL
        {mem[275],mem[274],mem[273],mem[272]} = 32'b00000000000000000001101110110011;// sll x23, x0,x0
x23 = 0 << 0 = 0
        {mem[279],mem[278],mem[277],mem[276]} = 32'b00000001110000001001101110110011;// sll x23, x1, x28
x23 = 409600 << 1 = 819200
        {mem[283],mem[282],mem[281],mem[280]} = 32'b00000001110000010001101110110011;// sll x23, x2, x28
x23 = -409600 << 1 = -819200

        //SLT
        {mem[287],mem[286],mem[285],mem[284]} = 32'b00000000000000000010101100110011;// slt x22, x0, x0
x22 = 0 < 0 ? = 0
        {mem[291],mem[290],mem[289],mem[288]} = 32'b00000000000000010010101100110011;// slt x22, x2, x0
x22 = -409600 < 0 ? = 1
        {mem[295],mem[294],mem[293],mem[292]} = 32'b00000000100001010010101100110011;// slt x22, x10, x8
x22 = 32 <44? =1

        //SLTU
        {mem[299],mem[298],mem[297],mem[296]} = 32'b00000000000000000011101010110011;// sltu x21, x0, x0
x21 = 0 < 0 ? = 0
        {mem[303],mem[302],mem[301],mem[300]} = 32'b00000000000000010011101010110011;// sltu x21, x2, x0
x22 = UNSIGNED -409600 < 0 ? = 0
        {mem[307],mem[306],mem[305],mem[304]} = 32'b00000000100001010011101010110011;// sltu x21, x10, x8
x22 = 32 <44? = 1

        //XOR
        {mem[311],mem[310],mem[309],mem[308]} = 32'b00000000000000000100101000110011;// xor x20, x0, x0
x20 = 0 ^ 0 = 0
        {mem[315],mem[314],mem[313],mem[312]} = 32'b00000001110000001100101000110011;// xor x20, x1, x28
x20 = 409600 ^ 1 = 409601
        {mem[319],mem[318],mem[317],mem[316]} = 32'b00000001110001010100101000110011;// xor x20, x10, x28
x20 = 32 ^ 1 = 33

        //SRL
        {mem[323],mem[322],mem[321],mem[320]} = 32'b00000000000000000101100110110011;// srl x19, x0,x0
x19 = 0 >> 0 = 0
        {mem[327],mem[326],mem[325],mem[324]} = 32'b00000001110000001101100110110011;// srl x19, x1, x28
x19 = 409600 >> 1 = 204800
        {mem[331],mem[330],mem[329],mem[328]} = 32'b00000001110000010101100110110011;// srl x19, x2, x28
x19 = -409600 >> 1

        //SRA
        {mem[335],mem[334],mem[333],mem[332]} = 32'b01000000000000000101100100110011;// sra x18, x0, x0
x18 = 0 >>> 0 = 0
        {mem[339],mem[338],mem[337],mem[336]} = 32'b01000001110000001101100100110011;// sra x18, x1, x28
x18 = 409600 >>> 1 = 204800
        {mem[343],mem[342],mem[341],mem[340]} = 32'b01000001110000010101100100110011;// sra x18, x2, x28
x18 = -409600 >>>1 = -204800

        //OR
        {mem[347],mem[346],mem[345],mem[344]} = 32'b00000000000000000110100010110011;// or x17, x0, x0
x17 = 0 | 0 = 0
```

```
        {mem[351],mem[350],mem[349],mem[348]} = 32'b00000000000100000110100010110011;// or x17, x0, x1
x17 = 0 | 409600 = 409600
        {mem[355],mem[354],mem[353],mem[352]} = 32'b00000001110000001110100010110011;// or x17, x1, x28
x18 = 409600 | 1 = 409601

        //AND
        {mem[359],mem[358],mem[357],mem[356]} = 32'b00000000000000000111100000110011;// and x16, x0, x0
x16 = 0 & 0 = 0
        {mem[363],mem[362],mem[361],mem[360]} = 32'b00000000000100000111100000110011;// and x16, x0, x1
x17 = 0 & 409600 = 0
        {mem[367],mem[366],mem[365],mem[364]} = 32'b00000000011001000111100000110011;// and x16, x8, x6
x16 = 44 & 40 = 40

        //FENCE
        {mem[371],mem[370],mem[369],mem[368]} = 32'b00001111111100000000000000001111;// FENCE = NOP

        //ECALL
        {mem[375],mem[374],mem[373],mem[372]} = 32'b00000000000000000000000001110011;// ECALL = NOP

        //EBREAK
        {mem[379],mem[378],mem[377],mem[376]} = 32'b00000000000100000000000001110011;// EBREAK = stop pc

    end


    Lab Program code:

            {mem[3],mem[2],mem[1],mem[0]}       = 32'b0000000_00000_00000_000_00000_0110011;  //add x0, x0, x0

        {mem[7],mem[6],mem[5],mem[4]}       = 32'b00000110010000000010000010000011; //lw x1, 100(x0)

        {mem[11],mem[10],mem[9],mem[8]}     = 32'b00000110100000000010000100000011 ; //lw x2, 104(x0)

        {mem[15],mem[14],mem[13],mem[12]} = 32'b00000110110000000010000110000011 ; //lw x3, 108(x0)

        {mem[19],mem[18],mem[17],mem[16]} = 32'b0000000_00010_00001_110_00100_0110011 ; //or x4, x1, x2

        {mem[23],mem[22],mem[21],mem[20]} = 32'b00000000001100100000010001100011; //beq x4, x3, 16

        {mem[27],mem[26],mem[25],mem[24]} = 32'b0000000_00010_00001_000_00011_0110011 ; //add x3, x1, x2

        {mem[31],mem[30],mem[29],mem[28]} = 32'b0000000_00010_00011_000_00101_0110011 ; //add x5, x3, x2

        {mem[35],mem[34],mem[33],mem[32]} = 32'b00000110010100000010100000100011; //sw x5, 112(x0)

        {mem[39],mem[38],mem[37],mem[36]} = 32'b00000111000000000010001100000011 ; //lw x6, 112(x0)

        {mem[43],mem[42],mem[41],mem[40]} = 32'b0000000_00001_00110_111_00111_0110011 ; //and x7, x6, x1

        {mem[47],mem[46],mem[45],mem[44]} = 32'b0100000_00010_00001_000_01000_0110011 ; //sub x8, x1, x2

        {mem[51],mem[50],mem[49],mem[48]} = 32'b0000000_00010_00001_000_00000_0110011 ; //add x0, x1, x2

        {mem[55],mem[54],mem[53],mem[52]} = 32'b0000000_00001_00000_000_01001_0110011 ; //add x9, x0, x1

        {mem[103],mem[102],mem[101],mem[100]}=32'd17;
        {mem[107],mem[106],mem[105],mem[104]}=32'd9;
        {mem[111],mem[110],mem[109],mem[108]}=32'd25;
```



31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Figure 1: Instructions to be supported

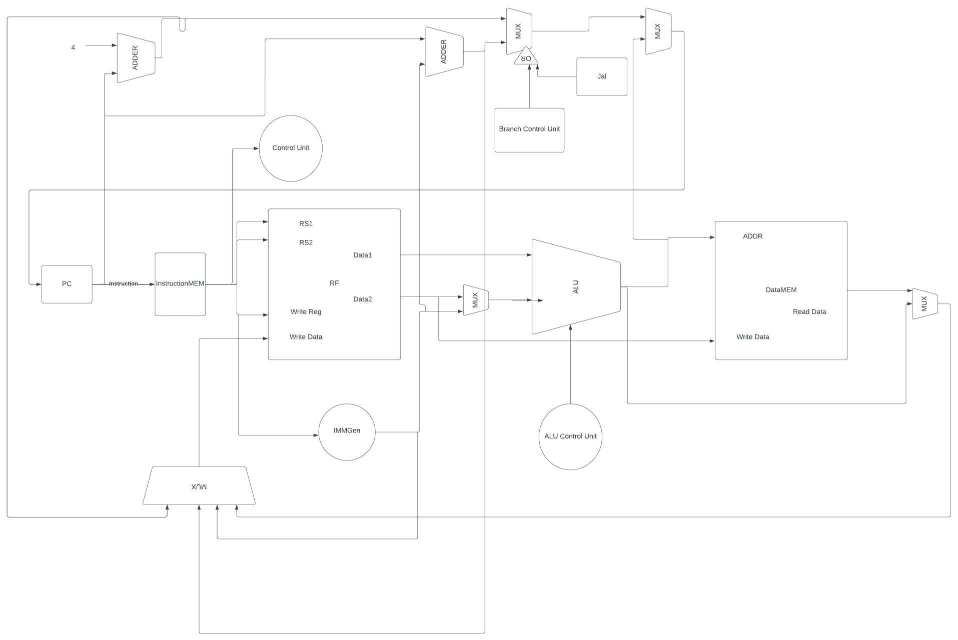


Figure 2: single cycle datapath supporting all instruction

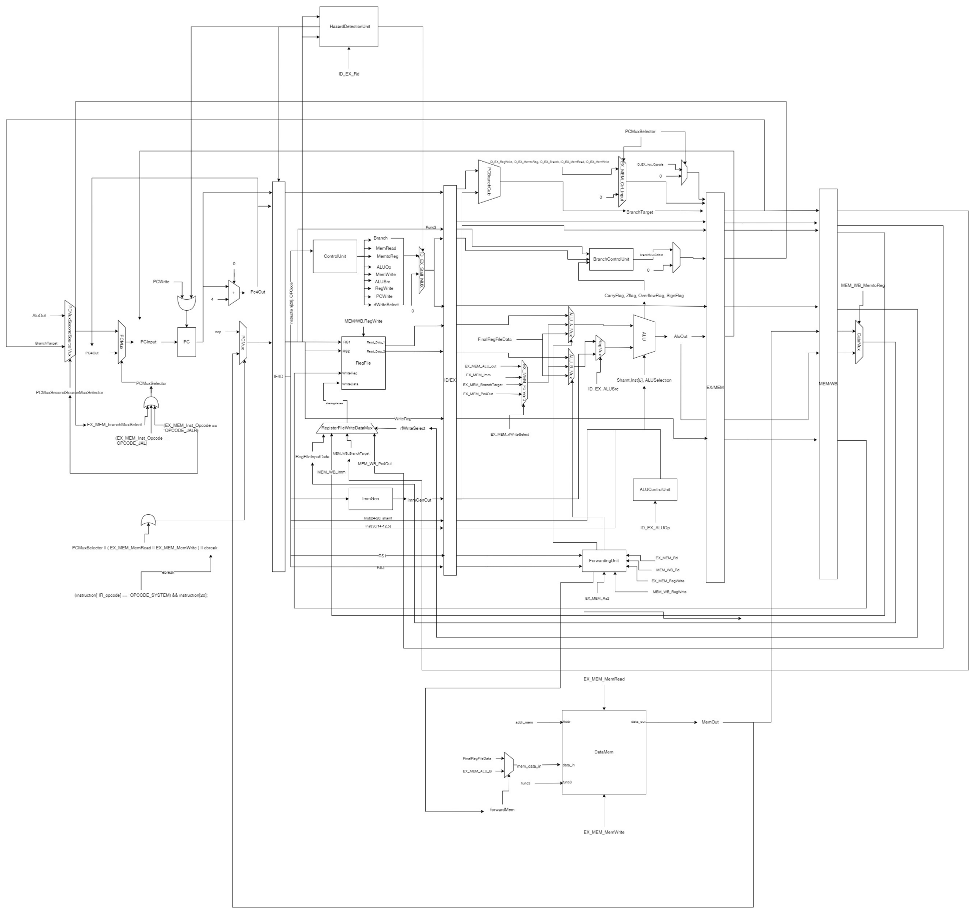


Figure 3: full 5 stage pipelined diagram